

Recurrent Neural Networks

Deep Learning Lecture 6

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2017

(Slides credit to Stanford CS231n and Hinton et al.)

Table of Contents

- 1 Motivation
- 2 Basic RNN
- 3 LSTM
- 4 Example: simple character-level language model
- 5 Example: image captioning
- 6 Overview of echo state networks
- 7 Conclusions

- A new pbworks wiki. Please share whatever you found with others!
- HW 2 will be due in **one week**
 - 3% bonus for the first correct submitter
 - As the winner of HW 1, Naim is out for this round

- A new pbworks wiki. Please share whatever you found with others!
- HW 2 will be due in **one week**
 - 3% bonus for the first correct submitter
 - As the winner of HW 1, Naim is out for this round

- Tensorflow 1.0 is now available in the OU supercomputer **schooner**
- Request account at http://www.ou.edu/content/oscer/support/accounts/new_account.html
- Use the group name **ouecedeeplrn**
- Try “module load TensorFlow” to access it
- Presenters: please try take advantage of it and let us know if they work :)
- For presenters of other packages, please ask the Norman side to install whatever packages you are presenting as well. They appear to be rather responsive
- Obada will present a small tutorial of how to use schooner after the break

- Tensorflow 1.0 is now available in the OU supercomputer **schooner**
- Request account at http://www.ou.edu/content/oscer/support/accounts/new_account.html
- Use the group name **ouecedeeplrn**
- Try “module load TensorFlow” to access it
- Presenters: please try take advantage of it and let us know if they work :)
- For presenters of other packages, please ask the Norman side to install whatever packages you are presenting as well. They appear to be rather responsive
- Obada will present a small tutorial of how to use schooner after the break

- Tensorflow 1.0 is now available in the OU supercomputer **schooner**
- Request account at http://www.ou.edu/content/oscer/support/accounts/new_account.html
- Use the group name **ouecedeeplrn**
- Try “module load TensorFlow” to access it
- Presenters: please try take advantage of it and let us know if they work :)
- For presenters of other packages, please ask the Norman side to install whatever packages you are presenting as well. They appear to be rather responsive
- Obada will present a small tutorial of how to use schooner after the break

Presentation starting next week!

Date	Student	Package
3/3	Aakash Soubhi	Tensorflow Tensorflow
3/10	Ahmad A Tamer	Theano Theano
3/24	Ahmad M Obada	Keras Keras
4/3	Muhanad Siraj	Caffe Caffe
4/10	Dong Varun	Torch Lasagne
4/17	Naim	MatConvNet

Review and Overview

- We looked into couple use cases of CNNs last week
 - Recognition and localization
 - Object detection
 - Some use of CNNs for arts
- Up to now, the network models we have studied are all memoryless
- We will discuss a non-memoryless model—recurrent neural networks today

Why non-memoryless models

- Almost all natural signals are sequential if we take time into account (we just cannot escape time)
- Memory is needed to remember the past
- They also offer a simplified solution for some problems (for example, number addition)
- They can treat some unsupervised problems as supervised problems
 - Consider prediction of a stock: unsupervised? Supervised?

Why non-memoryless models

- Almost all natural signals are sequential if we take time into account (we just cannot escape time)
- Memory is needed to remember the past
- They also offer a simplified solution for some problems (for example, number addition)
- They can treat some unsupervised problems as supervised problems
 - Consider prediction of a stock: unsupervised? Supervised?

Why non-memoryless models

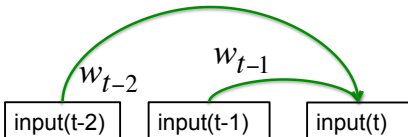
- Almost all natural signals are sequential if we take time into account (we just cannot escape time)
- Memory is needed to remember the past
- They also offer a simplified solution for some problems (for example, number addition)
- They can treat some unsupervised problems as supervised problems
 - Consider prediction of a stock: unsupervised? Supervised?

[Hinton 2012, week 7]

Memoryless models for sequences

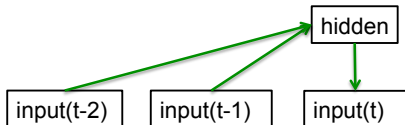
- Autoregressive models

Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.



- Feed-forward neural nets

These generalize autoregressive models by using one or more layers of non-linear hidden units. *e.g. Bengio's first language model.*



Beyond memoryless models

- If we provide some memories (hidden states) to our models, it will significantly increase the expressive power of the model
- We could store information for a long period of time in the hidden states
- Typically we do not know the exact values of the hidden states (that is why “hidden”). In many cases, the best we could do is just to infer a probability distribution over the hidden states
- Let's look at two classic examples

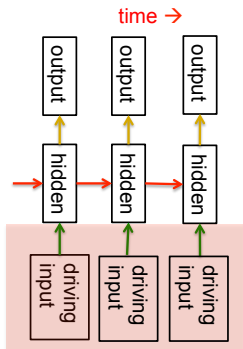
Beyond memoryless models

- If we provide some memories (hidden states) to our models, it will significantly increase the expressive power of the model
- We could store information for a long period of time in the hidden states
- Typically we do not know the exact values of the hidden states (that is why “hidden”). In many cases, the best we could do is just to infer a probability distribution over the hidden states
- Let's look at two classic examples

Beyond memoryless models

- If we provide some memories (hidden states) to our models, it will significantly increase the expressive power of the model
- We could store information for a long period of time in the hidden states
- Typically we do not know the exact values of the hidden states (that is why “hidden”). In many cases, the best we could do is just to infer a probability distribution over the hidden states
- Let's look at two classic examples

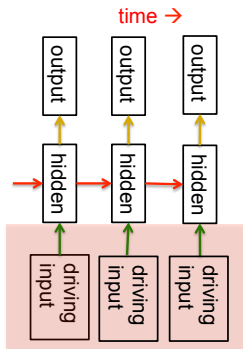
Linear dynamical systems (Engineers love them!)



- These are generative models with real **continuous** values as hidden states that cannot be observed directly
 - The hidden state has linear dynamics with Gaussian noise and produces the observations subjected to linear Gaussian noise
 - There can also be driving inputs
- To predict next output, we need to infer the hidden state

[Hinton 2012, Week 7]

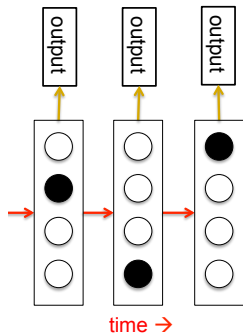
Linear dynamical systems (Engineers love them!)



- These are generative models with real **continuous** values as hidden states that cannot be observed directly
 - The hidden state has linear dynamics with Gaussian noise and produces the observations subjected to linear Gaussian noise
 - There can also be driving inputs
- To predict next output, we need to infer the hidden state

[Hinton 2012, Week 7]

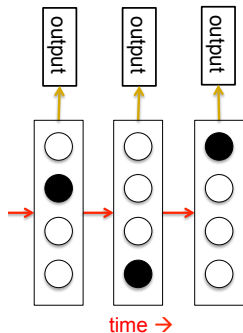
Hidden Markov Models (Computer scientists love them!)



- Hidden Markov Models (HMMs) have a **discrete** one-of- N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The output produced by a state are also stochastic
 - We don't know which state produced a given output. So the state is "hidden"
 - We can represent the probability distribution across N states with N numbers
- To predict next output, we need to infer the probability distribution over the hidden state

[Hinton 2012, Week 7]

Hidden Markov Models (Computer scientists love them!)



- Hidden Markov Models (HMMs) have a **discrete** one-of- N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The output produced by a state are also stochastic
 - We don't know which state produced a given output. So the state is "hidden"
 - We can represent the probability distribution across N states with N numbers
- To predict next output, we need to infer the probability distribution over the hidden state

[Hinton 2012, Week 7]

A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data
 - At each time step it must select one of its hidden states. So with N hidden states it can only remember $\log(N)$ bits about what it generated so far
- Consider the information that the first half of an utterance contains about the second half:
 - The syntax needs to fit (e.g. number and tense agreement)
 - The semantics needs to fit. The intonation needs to fit
 - The accent, rate, volume, and vocal tract characteristics must all fit
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half 2^{100} states

[Hinton 2012, week 7]

A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data
 - At each time step it must select one of its hidden states. So with N hidden states it can only remember $\log(N)$ bits about what it generated so far
- Consider the information that the first half of an utterance contains about the second half:
 - The syntax needs to fit (e.g. number and tense agreement)
 - The semantics needs to fit. The intonation needs to fit
 - The accent, rate, volume, and vocal tract characteristics must all fit
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half 2^{100} states

[Hinton 2012, week 7]

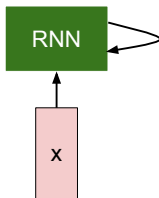
A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data
 - At each time step it must select one of its hidden states. So with N hidden states it can only remember $\log(N)$ bits about what it generated so far
- Consider the information that the first half of an utterance contains about the second half:
 - The syntax needs to fit (e.g. number and tense agreement)
 - The semantics needs to fit. The intonation needs to fit
 - The accent, rate, volume, and vocal tract characteristics must all fit
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half 2^{100} states

[Hinton 2012, week 7]

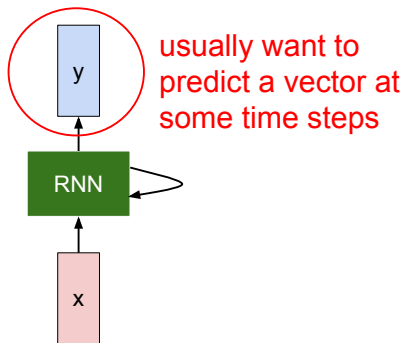
Recurrent neural networks (RNNs)

Recurrent Neural Network



Recurrent neural networks (RNNs)

Recurrent Neural Network



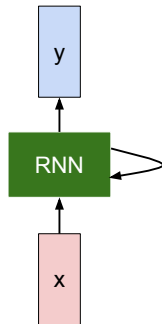
Recurrent neural networks (RNNs)

Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state some function with parameters W old state input vector at some time step



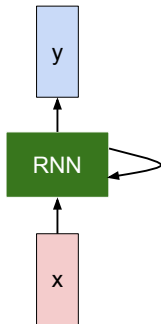
Recurrent neural networks (RNNs)

Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

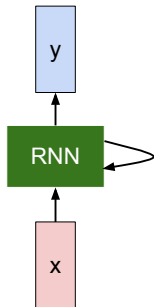
Notice: the same function and the same set of parameters are used at every time step.



Recurrent neural networks (RNNs)

(Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector h :



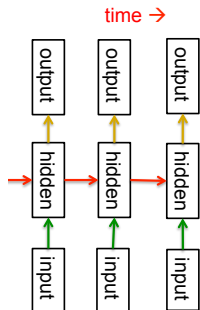
$$h_t = f_W(h_{t-1}, x_t)$$



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

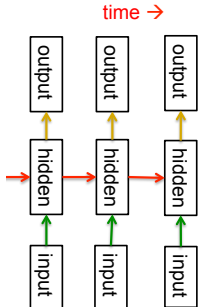
Recurrent neural networks (RNNs)



- RNNs are very powerful, because they combine two properties:
 - Distributed hidden state that allows them to store a lot of information about the past efficiently
 - Non-linear dynamics that allows them to update their hidden state in complicated ways
- With enough neurons and time, RNNs can compute anything that can be computed by your computer

[Hinton 2012, week 7]

Recurrent neural networks (RNNs)



- RNNs are very powerful, because they combine two properties:
 - Distributed hidden state that allows them to store a lot of information about the past efficiently
 - Non-linear dynamics that allows them to update their hidden state in complicated ways
- With enough neurons and time, RNNs can compute anything that can be computed by your computer

[Hinton 2012, week 7]

RNNs vs finite state machines

- An RNN can emulate a finite state machine but it is exponentially more powerful
 - An RNN with N hidden neurons has 2^N hidden activities (“states”)
- In contrast, the RNN only has $O(N^2)$ weights. Some wild analogy, if our brains are actually like RNNs
 - We are structurally quite similar (weights with maximally $O(N^2)$ different)
 - But we could behave significantly different based on experience (2^N different experiences)
- For a concrete comparison, if we have to remember an additional thing with information content close to the memory limit of an RNN
 - We just need to double the neurons of the RNN
 - But we need to square the number of states for finite state machines

RNNs vs finite state machines

- An RNN can emulate a finite state machine but it is exponentially more powerful
 - An RNN with N hidden neurons has 2^N hidden activities (“states”)
- In contrast, the RNN only has $O(N^2)$ weights. Some wild analogy, if our brains are actually like RNNs
 - We are structurally quite similar (weights with maximally $O(N^2)$ different)
 - But we could behave significantly different based on experience (2^N different experiences)
- For a concrete comparison, if we have to remember an additional thing with information content close to the memory limit of an RNN
 - We just need to double the neurons of the RNN
 - But we need to square the number of states for finite state machines

RNNs vs finite state machines

- An RNN can emulate a finite state machine but it is exponentially more powerful
 - An RNN with N hidden neurons has 2^N hidden activities (“states”)
- In contrast, the RNN only has $O(N^2)$ weights. Some wild analogy, if our brains are actually like RNNs
 - We are structurally quite similar (weights with maximally $O(N^2)$ different)
 - But we could behave significantly different based on experience (2^N different experiences)
- For a concrete comparison, if we have to remember an additional thing with information content close to the memory limit of an RNN
 - We just need to double the neurons of the RNN
 - But we need to square the number of states for finite state machines

RNNs vs finite state machines

- An RNN can emulate a finite state machine but it is exponentially more powerful
 - An RNN with N hidden neurons has 2^N hidden activities (“states”)
- In contrast, the RNN only has $O(N^2)$ weights. Some wild analogy, if our brains are actually like RNNs
 - We are structurally quite similar (weights with maximally $O(N^2)$ different)
 - But we could behave significantly different based on experience (2^N different experiences)
- For a concrete comparison, if we have to remember an additional thing with information content close to the memory limit of an RNN
 - We just need to double the neurons of the RNN
 - But we need to square the number of states for finite state machines

Recurrent neural networks (RNNs)

- What kinds of behaviour can RNNs exhibit?
 - They can oscillate. *Good for motor control?*
 - They can settle to point attractors. *Good for retrieving memories?*
 - They can behave chaotically. *Bad for information processing?*
 - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects (Hinton 2012)
- But the computational power of RNNs makes them very hard to train
 - As you will see, with some similar issues that plague deep feedforward nets
 - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer)

[Hinton 2012, week 7]

Recurrent neural networks (RNNs)

- What kinds of behaviour can RNNs exhibit?
 - They can oscillate. *Good for motor control?*
 - They can settle to point attractors. *Good for retrieving memories?*
 - They can behave chaotically. *Bad for information processing?*
 - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects (Hinton 2012)
- But the computational power of RNNs makes them very hard to train
 - As you will see, with some similar issues that plague deep feedforward nets
 - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer)

[Hinton 2012, week 7]

Recurrent neural networks (RNNs)

- What kinds of behaviour can RNNs exhibit?
 - They can oscillate. *Good for motor control?*
 - They can settle to point attractors. *Good for retrieving memories?*
 - They can behave chaotically. *Bad for information processing?*
 - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects (Hinton 2012)
- But the computational power of RNNs makes them very hard to train
 - As you will see, with some similar issues that plague deep feedforward nets
 - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer)

[Hinton 2012, week 7]

Recurrent neural networks (RNNs)

- What kinds of behaviour can RNNs exhibit?
 - They can oscillate. *Good for motor control?*
 - They can settle to point attractors. *Good for retrieving memories?*
 - They can behave chaotically. *Bad for information processing?*
 - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects (Hinton 2012)
- But the computational power of RNNs makes them very hard to train
 - As you will see, with some similar issues that plague deep feedforward nets
 - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer)

[Hinton 2012, week 7]

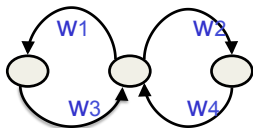
Recurrent neural networks (RNNs)

- What kinds of behaviour can RNNs exhibit?
 - They can oscillate. *Good for motor control?*
 - They can settle to point attractors. *Good for retrieving memories?*
 - They can behave chaotically. *Bad for information processing?*
 - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects (Hinton 2012)
- But the computational power of RNNs makes them very hard to train
 - As you will see, with some similar issues that plague deep feedforward nets
 - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer)

[Hinton 2012, week 7]

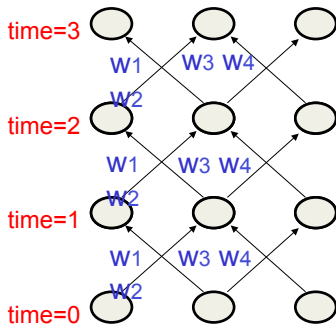
Expanding RNN as feedforward nets

The equivalence between feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.



[Hinton 2012, week 7]

Backpropagation with weight constraints

To constrain : $w_1^{(1)} = w_1^{(2)}$

We need : $\Delta w_1^{(1)} = \Delta w_1^{(2)}$

Compute : $\frac{\partial E}{\partial w_1^{(1)}}$ and $\frac{\partial E}{\partial w_1^{(2)}}$

Use : $\frac{\partial E}{\partial w_1^{(1)}} + \frac{\partial E}{\partial w_1^{(2)}}$

for both $w_1^{(1)}$ and $w_1^{(2)}$

[Hinton 2012, week 7]

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
 - So if the weights started off satisfying the constraints, they will continue to satisfy them

Backpropagation with weight constraints

To constrain : $w_1^{(1)} = w_1^{(2)}$

We need : $\Delta w_1^{(1)} = \Delta w_1^{(2)}$

Compute : $\frac{\partial E}{\partial w_1^{(1)}}$ and $\frac{\partial E}{\partial w_1^{(2)}}$

Use : $\frac{\partial E}{\partial w_1^{(1)}} + \frac{\partial E}{\partial w_1^{(2)}}$

for both $w_1^{(1)}$ and $w_1^{(2)}$

[Hinton 2012, week 7]

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
 - So if the weights started off satisfying the constraints, they will continue to satisfy them

Back-Propagation Through Time (BPTT)

- In previous slides, we considered the recurrent net as a layered, feed-forward net with shared weights and then trained the feed-forward net with weight constraints
- Equivalently, we can also think of this training algorithm in the time domain:
 - The forward pass builds up a stack of the activities of all the units at each time step
 - The backward pass peels activities off the stack to compute the error derivatives at each time step
 - After the backward pass we add together the derivatives at all the different times for each weight

Back-Propagation Through Time (BPTT)

- In previous slides, we considered the recurrent net as a layered, feed-forward net with shared weights and then trained the feed-forward net with weight constraints
- Equivalently, we can also think of this training algorithm in the time domain:
 - The forward pass builds up a stack of the activities of all the units at each time step
 - The backward pass peels activities off the stack to compute the error derivatives at each time step
 - After the backward pass we add together the derivatives at all the different times for each weight

Back-Propagation Through Time (BPTT)

- In previous slides, we considered the recurrent net as a layered, feed-forward net with shared weights and then trained the feed-forward net with weight constraints
- Equivalently, we can also think of this training algorithm in the time domain:
 - The forward pass builds up a stack of the activities of all the units at each time step
 - The backward pass peels activities off the stack to compute the error derivatives at each time step
 - After the backward pass we add together the derivatives at all the different times for each weight

Back-Propagation Through Time (BPTT)

- In previous slides, we considered the recurrent net as a layered, feed-forward net with shared weights and then trained the feed-forward net with weight constraints
- Equivalently, we can also think of this training algorithm in the time domain:
 - The forward pass builds up a stack of the activities of all the units at each time step
 - The backward pass peels activities off the stack to compute the error derivatives at each time step
 - After the backward pass we add together the derivatives at all the different times for each weight

An irritative extra issue

- We need to specify the initial activity state of all the hidden and output units
- We could just fix these initial states to have some default value like 0.5
- But it is better to treat the initial states as learned parameters
- We learn them in the same way as we learn the weights
 - Start off with an initial random guess for the initial states
 - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
 - Adjust the initial states by following the negative gradient

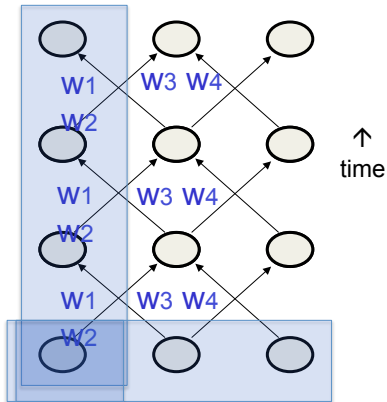
An irritative extra issue

- We need to specify the initial activity state of all the hidden and output units
- We could just fix these initial states to have some default value like 0.5
- But it is better to treat the initial states as learned parameters
- We learn them in the same way as we learn the weights
 - Start off with an initial random guess for the initial states
 - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
 - Adjust the initial states by following the negative gradient

An irritative extra issue

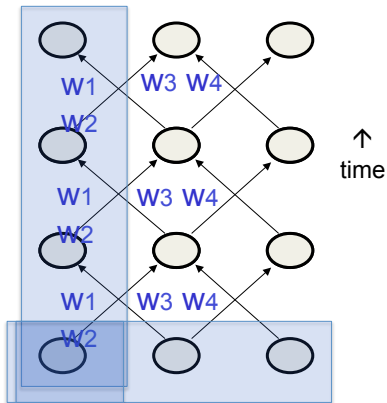
- We need to specify the initial activity state of all the hidden and output units
- We could just fix these initial states to have some default value like 0.5
- But it is better to treat the initial states as learned parameters
- We learn them in the same way as we learn the weights
 - Start off with an initial random guess for the initial states
 - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
 - Adjust the initial states by following the negative gradient

Providing inputs to recurrent networks



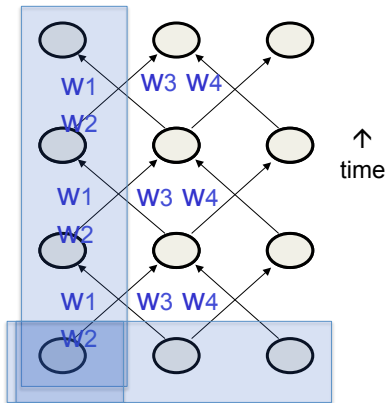
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



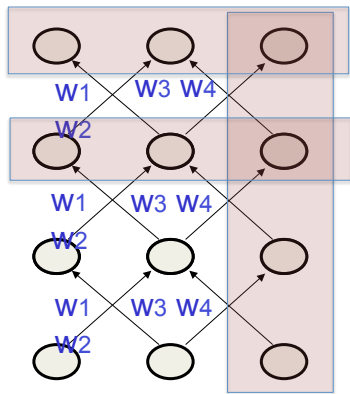
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



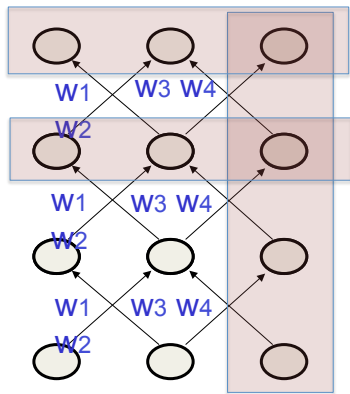
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Teaching recurrent networks to learn signals



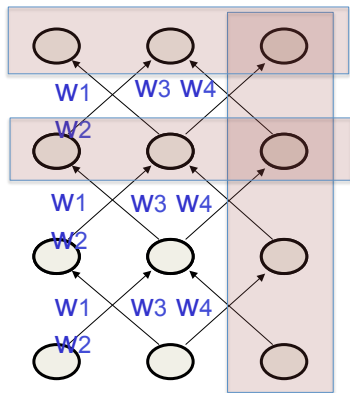
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



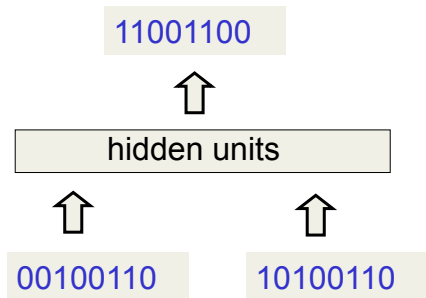
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



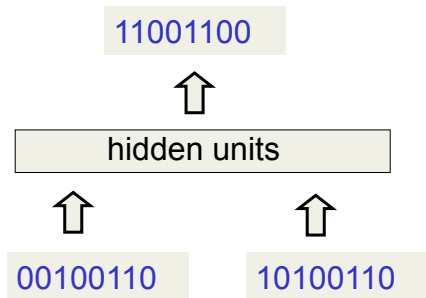
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Toy problem for RNN: binary addition



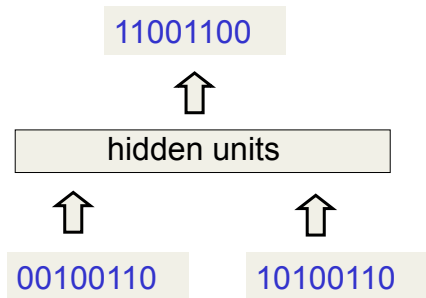
- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

Toy problem for RNN: binary addition



- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

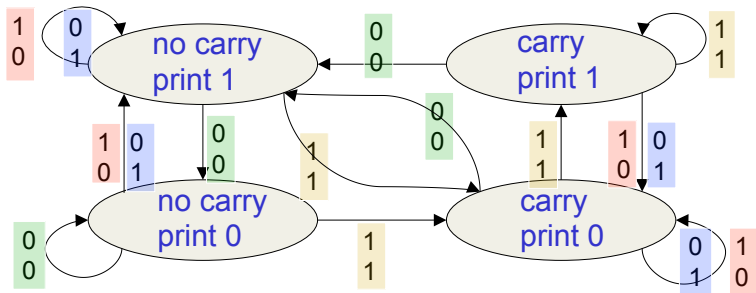
Toy problem for RNN: binary addition



- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary addition task

We are trying to learn this!

The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.

A little bit detail

$$x = [b_8, b_7, \dots, b_1]$$

$$y = [c_8, c_7, \dots, c_1]$$

$$z = x + y = [d_8, d_7, \dots, d_1]$$

$$\hat{z} = [\hat{d}_8, \hat{d}_7, \dots, \hat{d}_1]$$

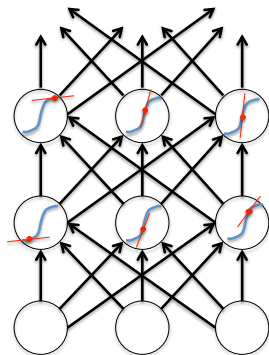
$$\text{Hidden unit: } h_i = \text{sigm}(W_{x,h}[b_i, c_i]^T + W_{h,h}h_{i-1})$$

$$\text{Output: } \hat{d}_i = \text{sigm}(W_{h,z}h_i)$$

https://github.com/llSourcecell/recurrent_neural_net_demo

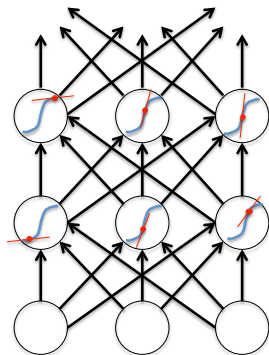
Demo

Why training RNN is difficult? The backward pass is linear



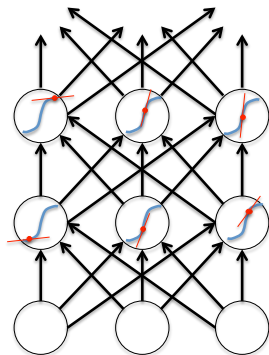
- There is a big difference between the forward and backward passes
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double
 - The forward pass determines the slope of the linear function used for backpropagating through each neuron

Why training RNN is difficult? The backward pass is linear



- There is a big difference between the forward and backward passes
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double
 - The forward pass determines the slope of the linear function used for backpropagating through each neuron

Why training RNN is difficult? The backward pass is linear



- There is a big difference between the forward and backward passes
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double
 - The forward pass determines the slope of the linear function used for backpropagating through each neuron

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago
 - So RNNs have difficulty dealing with long-range dependencies

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
 - If the weights are small, the gradients shrink exponentially.
 - If the weights are big the gradients grow exponentially
- Typical feed-forward neural nets can cope with these exponential effects when they only have a few hidden layers
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish
 - We could avoid this by initializing the weights very carefully
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago
 - So RNNs have difficulty dealing with long-range dependencies

Passing gradient to many steps back

- Recall

$$h_t = \tanh(W_{hh}^{(t)} h_{t-1} + W_{xh}^{(t)} x_t)$$

- To see how W_{xh} at the first time step affects the hidden layer at time t , compute

$$\begin{aligned} \frac{\partial h_t}{\partial W_{xh}^{(1)}} &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \dots \frac{\partial h_1}{\partial W_{xh}^{(1)}} \\ &= \left(\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)} \right) \frac{\partial h_1}{\partial W_{xh}^{(1)}}, \end{aligned}$$

where $\tanh^{(\tau)} = \tanh'(W_{hh}^{(\tau)} h_{\tau-1} + W_{xh}^{(\tau)} x_{\tau})$.

- $\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)}$ can either explode or vanish when t is big

Passing gradient to many steps back

- Recall

$$h_t = \tanh(W_{hh}^{(t)} h_{t-1} + W_{xh}^{(t)} x_t)$$

- To see how W_{xh} at the first time step affects the hidden layer at time t , compute

$$\begin{aligned} \frac{\partial h_t}{\partial W_{xh}^{(1)}} &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \dots \frac{\partial h_1}{\partial W_{xh}^{(1)}} \\ &= \left(\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)} \right) \frac{\partial h_1}{\partial W_{xh}^{(1)}}, \end{aligned}$$

where $\tanh^{(\tau)} = \tanh'(W_{hh}^{(\tau)} h_{\tau-1} + W_{xh}^{(\tau)} x_{\tau})$.

- $\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)}$ can either explode or vanish when t is big

Passing gradient to many steps back

- Recall

$$h_t = \tanh(W_{hh}^{(t)} h_{t-1} + W_{xh}^{(t)} x_t)$$

- To see how W_{xh} at the first time step affects the hidden layer at time t , compute

$$\begin{aligned} \frac{\partial h_t}{\partial W_{xh}^{(1)}} &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \dots \frac{\partial h_1}{\partial W_{xh}^{(1)}} \\ &= \left(\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)} \right) \frac{\partial h_1}{\partial W_{xh}^{(1)}}, \end{aligned}$$

where $\tanh^{(\tau)} = \tanh'(W_{hh}^{(\tau)} h_{\tau-1} + W_{xh}^{(\tau)} x_{\tau})$.

- $\prod_{\tau=2}^t \tanh^{(\tau)} W_{hh}^{(\tau)}$ can either explode or vanish when t is big

Understanding gradient flow dynamics

Cute backprop signal video: <http://imgur.com/gallery/vaNahKE>

```
H = 5 # dimensionality of hidden state
T = 50 # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```


Understanding gradient flow dynamics

```

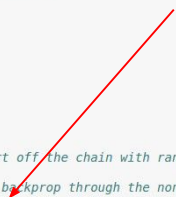
H = 5 # dimensionality of hidden state
T = 50 # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state

```

if the largest eigenvalue is > 1 , gradient will explode
 if the largest eigenvalue is < 1 , gradient will vanish



[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Four effective ways to learn an RNN

- **Long Short Term Memory:**
Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:**
Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:**
Initialize the input→ hidden and hidden→hidden and output→ hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden→output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

Four effective ways to learn an RNN

- **Long Short Term Memory:**
Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:**
Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:**
Initialize the input→ hidden and hidden→hidden and output→ hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden→output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

Four effective ways to learn an RNN

- **Long Short Term Memory:**
Make the RNN out of little modules that are designed to remember values for a long time
- **Hessian Free Optimization:**
Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature
 - The HF optimizer (Martens & Sutskever, 2011) is good at this
- **Echo State Networks:**
Initialize the input→ hidden and hidden→hidden and output→ hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input
 - ESNs only need to learn the hidden→output connections
- **Good initialization with momentum:** Initialize like in Echo State Networks, but then learn all of the connections using momentum

Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

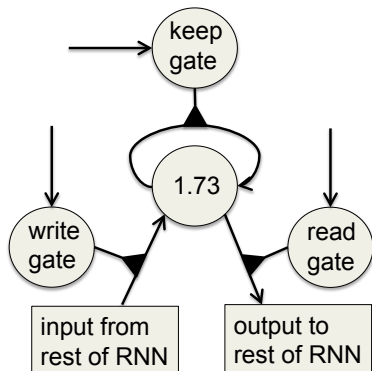
Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

Long Short Term Memory (LSTM)

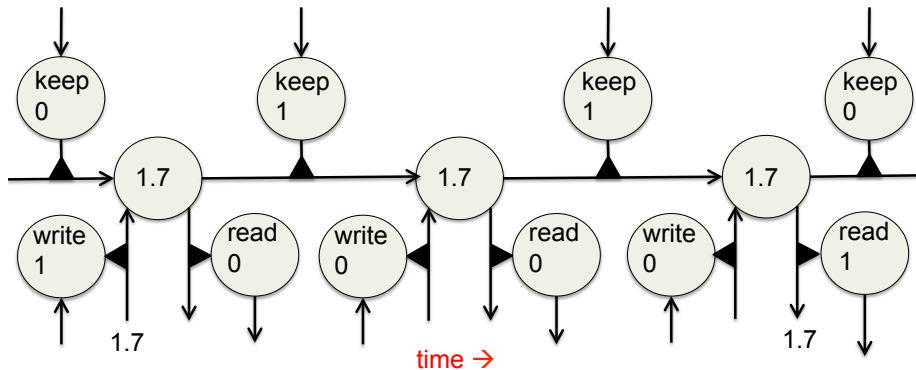
- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

Implementing a memory cell in a neural network



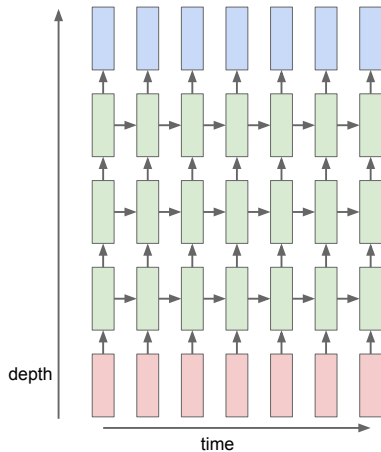
- To preserve information for a long time in the activities of an RNN, we use a circuit that implements an analog memory cell
 - A linear unit that has a self-link with a weight of 1 will maintain its state
 - Information is stored in the cell by activating its write gate
 - Information is retrieved by activating the read gate.
 - We can backpropagate through this circuit because logistics are have nice derivatives

Backpropagation through a memory cell



RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

 $h \in \mathbb{R}^n.$ $W^l [n \times 2n]$


RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$h \in \mathbb{R}^n, \quad W^l [n \times 2n]$$

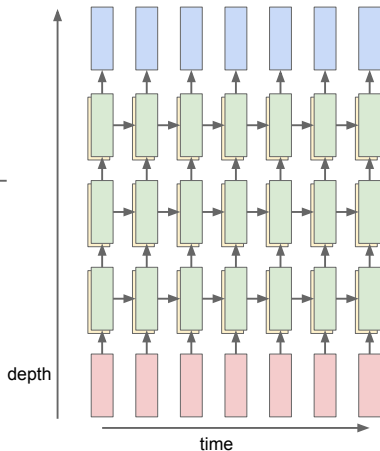
LSTM:

$$W^l [4n \times 2n]$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

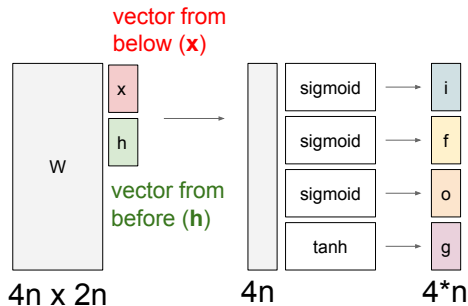
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



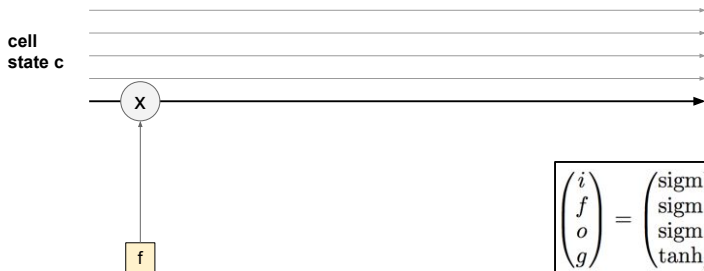
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



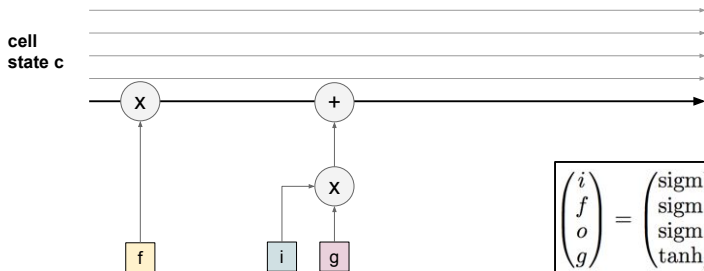
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



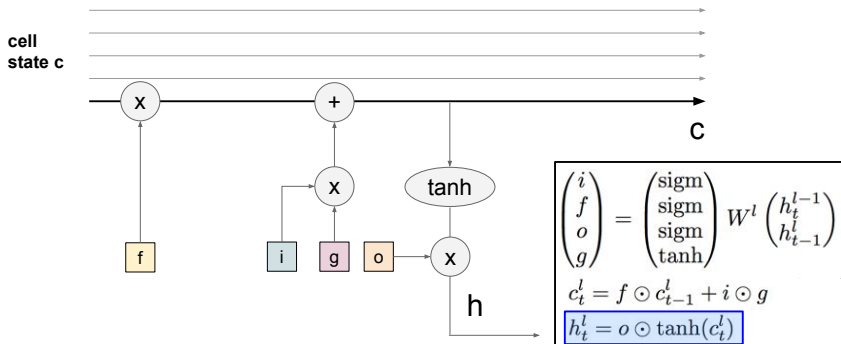
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^{l-1} \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



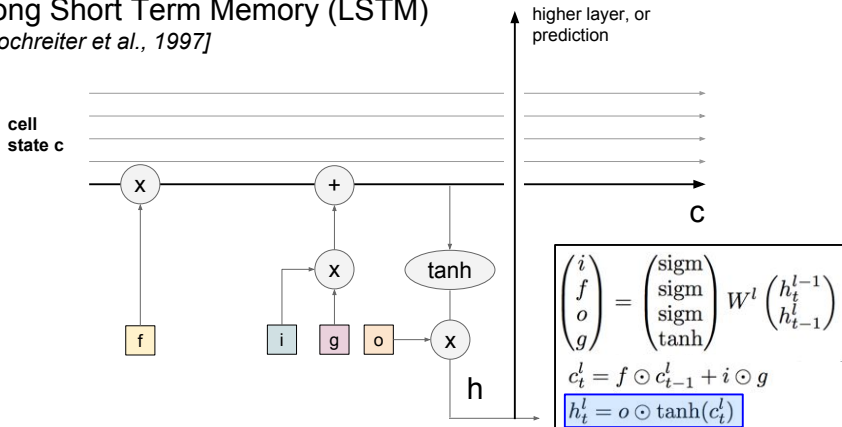
Fei-Fei Li & Andrej Karpathy & Justin Johnson

Lecture 10 - 73

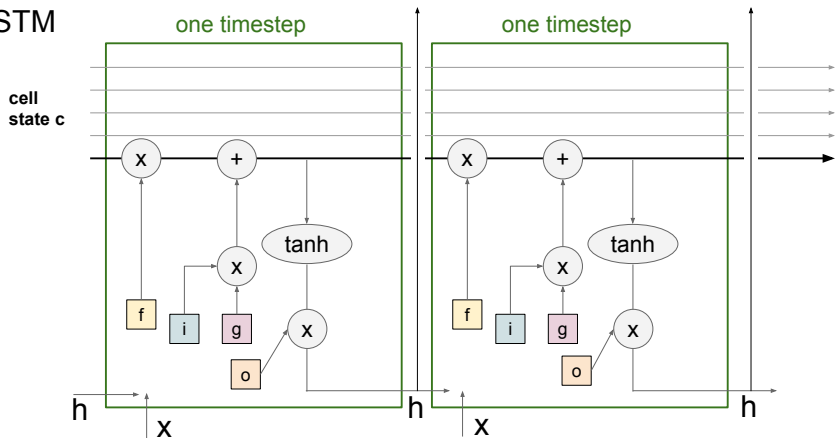
8 Feb 2016

Long Short Term Memory (LSTM)

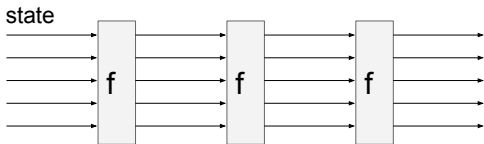
[Hochreiter et al., 1997]



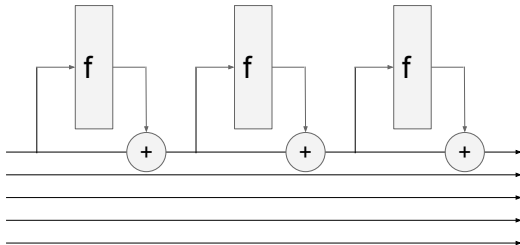
LSTM

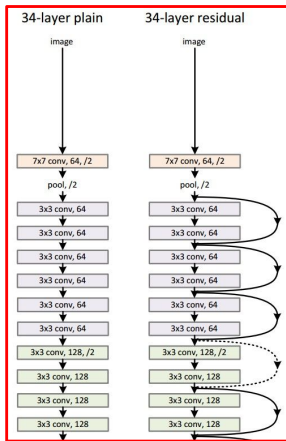


RNN



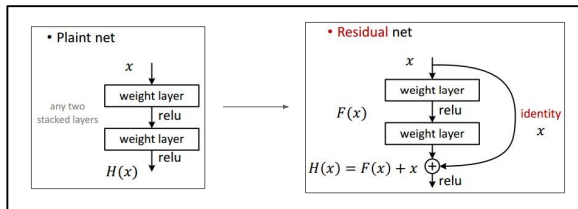
LSTM
(ignoring
forget gates)



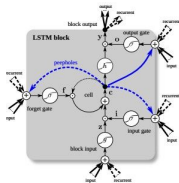


Recall: “PlainNets” vs. ResNets

ResNet is to PlainNet what LSTM is to RNN, kind of.



LSTM variants and friends



[LSTM: A Search Space Odyssey, Greff et al., 2015]

GRU [Learning phrase representations using rnn encoder-decoder for statistical machine translation, Cho et al. 2014]

$$\begin{aligned}
 r_t &= \text{sigm}(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
 z_t &= \text{sigm}(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
 \tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
 h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
 \end{aligned}$$

[An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015]

MUT1:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

MUT2:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\
 r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

MUT3:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

Modelling text: Advantages of working with characters

- The web is composed of character strings
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see)
- Pre-processing text to get words is a big hassle
 - What about morphemes (prefixes, suffixes etc)
 - What about subtle effects like “sn” words?
 - What about New York vs new York Minster roof?
 - What about Finnish
 - ymmärtämättömyydellänsäkään

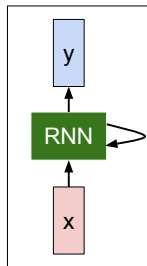
Modelling text: Advantages of working with characters

- The web is composed of character strings
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see)
- Pre-processing text to get words is a big hassle
 - What about morphemes (prefixes, suffixes etc)
 - What about subtle effects like “sn” words?
 - What about New York vs new York Minster roof?
 - What about Finnish
 - ymmärtämättömyydellänsäkään

Character-level language model example

Vocabulary:
[h,e,l,o]

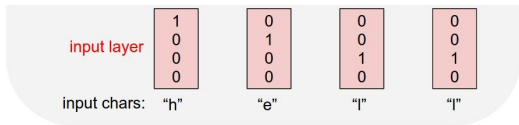
Example training
sequence:
“hello”



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

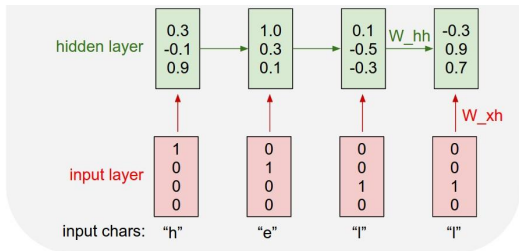


Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"

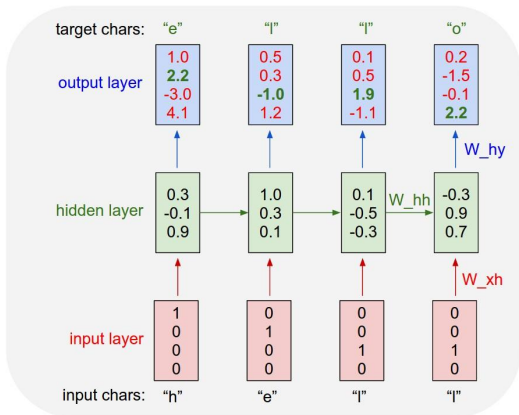
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



min-char-rnn.py gist: 112 lines of Python

```

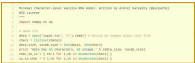
1 """
2 Minimal character-level vanilla RNN model. Written by Andrej Karpathy (karpathy@cs.cmu.edu)
3 """
4
5 import numpy as np
6
7 # data loader
8
9 # data loader
10 # data loader
11 # data loader
12 # data loader
13 # data loader
14 # data loader
15 # data loader
16 # data loader
17 # data loader
18 # data loader
19 # data loader
20 # data loader
21 # data loader
22 # data loader
23 # data loader
24 # data loader
25 # data loader
26 # data loader
27 # data loader
28 # data loader
29 # data loader
30 # data loader
31 # data loader
32 # data loader
33 # data loader
34 # data loader
35 # data loader
36 # data loader
37 # data loader
38 # data loader
39 # data loader
40 # data loader
41 # data loader
42 # data loader
43 # data loader
44 # data loader
45 # data loader
46 # data loader
47 # data loader
48 # data loader
49 # data loader
50 # data loader
51 # data loader
52 # data loader
53 # data loader
54 # data loader
55 # data loader
56 # data loader
57 # data loader
58 # data loader
59 # data loader
60 # data loader
61 # data loader
62 # data loader
63 # data loader
64 # data loader
65 # data loader
66 # data loader
67 # data loader
68 # data loader
69 # data loader
70 # data loader
71 # data loader
72 # data loader
73 # data loader
74 # data loader
75 # data loader
76 # data loader
77 # data loader
78 # data loader
79 # data loader
80 # data loader
81 # data loader
82 # data loader
83 # data loader
84 # data loader
85 # data loader
86 # data loader
87 # data loader
88 # data loader
89 # data loader
90 # data loader
91 # data loader
92 # data loader
93 # data loader
94 # data loader
95 # data loader
96 # data loader
97 # data loader
98 # data loader
99 # data loader
100 # data loader
101 # data loader
102 # data loader
103 # data loader
104 # data loader
105 # data loader
106 # data loader
107 # data loader
108 # data loader
109 # data loader
110 # data loader
111 # data loader
112 # data loader

```

<https://gist.github.com/karpathy/d4dee566867f8291f086>

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 22 8 Feb 2016

min-char-rnn.py gist



```

1 # Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
2 BSD License
3 """
4 import numpy as np
5
6 # data I/O
7 data = open('input.txt', 'r').read() # should be simple plain text file
8 chars = list(set(data))
9 data_size, vocab_size = len(data), len(chars)
10 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
11 char_to_ix = { ch:i for i,ch in enumerate(chars) }
12 ix_to_char = { i:ch for i,ch in enumerate(chars) }
13
14 # ...
15
16 # ...
17
18 # ...
19
20 # ...
21
22 # ...
23
24 # ...
25
26 # ...
27
28 # ...
29
30 # ...
31
32 # ...
33
34 # ...
35
36 # ...
37
38 # ...
39
40 # ...
41
42 # ...
43
44 # ...
45
46 # ...
47
48 # ...
49
50 # ...
51
52 # ...
53
54 # ...
55
56 # ...
57
58 # ...
59
60 # ...
61
62 # ...
63
64 # ...
65
66 # ...
67
68 # ...
69
70 # ...
71
72 # ...
73
74 # ...
75
76 # ...
77
78 # ...
79
80 # ...
81
82 # ...
83
84 # ...
85
86 # ...
87
88 # ...
89
90 # ...
91
92 # ...
93
94 # ...
95
96 # ...
97
98 # ...
99
100 # ...

```

Data I/O

```

1 """
2 Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }

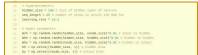
```

min-char-rnn.py gist

```

1 #!/usr/bin/env python
2 """
3 Example: character-level language model, trained by simple backprop (BPTRT)
4 """
5
6 # hyperparameters
7 hidden_size = 100 # size of hidden layer of neurons
8 seq_length = 25 # number of steps to unroll the RNN for
9 learning_rate = 1e-1
10
11 # model parameters
12 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
13 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
14 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
15 bh = np.zeros((hidden_size, 1)) # hidden bias
16 by = np.zeros((vocab_size, 1)) # output bias
17
18 # training
19
20 # load data
21 data = open('data.txt').read()
22 vocab = set(data)
23 vocab_size = len(vocab)
24
25 # create character to index mapping
26 char2idx = {}
27 for i, c in enumerate(vocab):
28     char2idx[c] = i
29
30 # create index to character mapping
31 idx2char = {}
32 for i, c in enumerate(vocab):
33     idx2char[i] = c
34
35 # create training data
36 train_data = []
37 for i in range(0, len(data)-seq_length):
38     train_data.append(data[i:i+seq_length])
39
40 # create validation data
41 val_data = data[-seq_length:]
42
43 # create training and validation iterators
44 train_iter = iter(train_data)
45 val_iter = iter(val_data)
46
47 # initialize model
48 model = minCharRNN(hidden_size, vocab_size, vocab_size)
49
50 # training loop
51 for epoch in range(1, 10000):
52     # train
53     for i, train_data in enumerate(train_iter):
54         # forward pass
55         hidden = model.initHidden()
56         output, hidden = model.unroll(train_data, hidden)
57         # backward pass
58         model.backward(output, hidden)
59         # update parameters
60         model.updateParameters()
61     # validate
62     for i, val_data in enumerate(val_iter):
63         # forward pass
64         hidden = model.initHidden()
65         output, hidden = model.unroll(val_data, hidden)
66         # backward pass
67         model.backward(output, hidden)
68         # update parameters
69         model.updateParameters()
70     # print progress
71     print('epoch %d, train loss: %f, val loss: %f' % (epoch, model.train_loss, model.val_loss))
72
73 # save model
74 model.save('model.pkl')
75
76 # load model
77 model = minCharRNN.load('model.pkl')
78
79 # generate text
80 model.initHidden()
81 text = ' '
82 for i in range(1, 1000):
83     # forward pass
84     output, hidden = model.unroll(text, model.initHidden())
85     # print character
86     print(idx2char[output[0][0]], end='')
87     # update hidden state
88     model.initHidden()
89     # update text
90     text += idx2char[output[0][0]]
91
92 # done
93 """

```



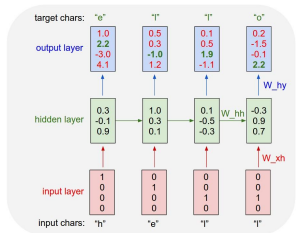
Initializations

```

15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias

```

recall:



min-char-rnn.py gist

```

1 from random import choice, randint, random, seed, shuffle, sample, shuffle, sample, shuffle, sample
2 import sys
3
4 # Model parameters
5 W = [[0.0 for _ in range(256)] for _ in range(256)]
6 W_in = [[0.0 for _ in range(256)] for _ in range(256)]
7 W_out = [[0.0 for _ in range(256)] for _ in range(256)]
8 W_hh = [[0.0 for _ in range(256)] for _ in range(256)]
9 W_hl = [[0.0 for _ in range(256)] for _ in range(256)]
10 W_lm = [[0.0 for _ in range(256)] for _ in range(256)]
11 W_lm_in = [[0.0 for _ in range(256)] for _ in range(256)]
12 W_lm_out = [[0.0 for _ in range(256)] for _ in range(256)]
13
14 # Model state
15 h = [0.0 for _ in range(256)]
16 l = [0.0 for _ in range(256)]
17
18 # Model training
19 def train():
20     # Load data
21     data = open('data.txt').read().strip()
22     vocab = set(data)
23     char_to_ix = {char: ix for ix, char in enumerate(vocab)}
24     ix_to_char = {ix: char for ix, char in enumerate(vocab)}
25     data_ix = [char_to_ix[char] for char in data]
26     data_ix = data_ix + [0] * (1000000 - len(data_ix))
27
28     # Prepare training data
29     seq_length = 100
30     n_batches = len(data_ix) // seq_length
31     for batch in range(n_batches):
32         # Sample a batch of data
33         start = batch * seq_length
34         end = start + seq_length
35         data_batch = data_ix[start:end]
36
37         # Compute gradients
38         # ... (omitted)
39
40         # Update parameters
41         # ... (omitted)
42
43     # Save model
44     # ... (omitted)
45
46 # Main loop
47 if __name__ == '__main__':
48     train()
49 
```

```

50 # Main loop
51 n, p = 0, 0
52 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(WWhy)
53 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
54 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
55 while True:
56     # prepare inputs (we're sweeping from left to right in steps seq_length long)
57     if p+seq_length+1 >= len(data) or n == 0:
58         hprev = np.zeros((hidden_size,1)) # reset RNN memory
59         p = 0 # go from start of data
60         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
61         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
62
63     # sample from the model now and then
64     if n % 100 == 0:
65         sample_ix = sample(hprev, inputs[0], 200)
66         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
67         print '----\n%s \n----' % (txt, )
68
69     # forward seq_length characters through the net and fetch gradient
70     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
71     smooth_loss = smooth_loss * 0.999 + loss * 0.001
72     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
73
74     # perform parameter update with Adagrad
75     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
76                                 [dWxh, dWhh, dWhy, dbh, dby],
77                                 [mWxh, mWhh, mWhy, mbh, mby]):
78         mem += dparam * dparam
79         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
80
81     p += seq_length # move data pointer
82     n += 1 # iteration counter
83 
```



Main loop

```

81 n, p = 0, 0
82 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(WWhy)
83 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
84 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
85 while True:
86     # prepare inputs (we're sweeping from left to right in steps seq_length long)
87     if p+seq_length+1 >= len(data) or n == 0:
88         hprev = np.zeros((hidden_size,1)) # reset RNN memory
89         p = 0 # go from start of data
90         inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
91         targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
92
93     # sample from the model now and then
94     if n % 100 == 0:
95         sample_ix = sample(hprev, inputs[0], 200)
96         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
97         print '----\n%s \n----' % (txt, )
98
99     # forward seq_length characters through the net and fetch gradient
100     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101     smooth_loss = smooth_loss * 0.999 + loss * 0.001
102     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104     # perform parameter update with Adagrad
105     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                 [dWxh, dWhh, dWhy, dbh, dby],
107                                 [mWxh, mWhh, mWhy, mbh, mby]):
108         mem += dparam * dparam
109         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
110
111     p += seq_length # move data pointer
112     n += 1 # iteration counter
113 
```

min-char-rnn.py gist

```

1 #!/usr/bin/env python3
2 """
3 Example: character-level neural net model, trained by stochastic gradient descent
4 """
5
6 # Imports
7
8 # Random number generator
9 import random
10
11 # Data
12 data = 'The cat sat on the mat.'
13 vocab = 'abcdefghijklmnopqrstuvwxyz '
14
15 # Vocabulary
16 vocab_size = len(vocab)
17 char_to_ix = {char:ix for ix,char in enumerate(vocab)}
18 ix_to_char = {ix:char for ix,char in enumerate(vocab)}
19
20 # Parameters
21 seq_length = 100 # size of hidden state of network
22 hidden_size = 128 # size of hidden state of network
23
24 # Hyperparameters
25 learning_rate = 0.01 # learning rate
26 num_epochs = 10 # number of epochs to train the model
27 num_batches_per_epoch = 1000 // len(data) # number of batches per epoch
28
29 # Model
30
31 # Define the model
32
33 # Define the model
34
35 # Define the model
36
37 # Define the model
38
39 # Define the model
40
41 # Define the model
42
43 # Define the model
44
45 # Define the model
46
47 # Define the model
48
49 # Define the model
50
51 # Define the model
52
53 # Define the model
54
55 # Define the model
56
57 # Define the model
58
59 # Define the model
60
61 # Define the model
62
63 # Define the model
64
65 # Define the model
66
67 # Define the model
68
69 # Define the model
70
71 # Define the model
72
73 # Define the model
74
75 # Define the model
76
77 # Define the model
78
79 # Define the model
80
81 # Define the model
82
83 # Define the model
84
85 # Define the model
86
87 # Define the model
88
89 # Define the model
90
91 # Define the model
92
93 # Define the model
94
95 # Define the model
96
97 # Define the model
98
99 # Define the model
100
101 # Define the model
102
103 # Define the model
104
105 # Define the model
106
107 # Define the model
108
109 # Define the model
110
111 # Define the model
112
113 # Define the model
114
115 # Define the model
116
117 # Define the model
118
119 # Define the model
120
121 # Define the model
122
123 # Define the model
124
125 # Define the model
126
127 # Define the model
128
129 # Define the model
130
131 # Define the model
132
133 # Define the model
134
135 # Define the model
136
137 # Define the model
138
139 # Define the model
140
141 # Define the model
142
143 # Define the model
144
145 # Define the model
146
147 # Define the model
148
149 # Define the model
150
151 # Define the model
152
153 # Define the model
154
155 # Define the model
156
157 # Define the model
158
159 # Define the model
160
161 # Define the model
162
163 # Define the model
164
165 # Define the model
166
167 # Define the model
168
169 # Define the model
170
171 # Define the model
172
173 # Define the model
174
175 # Define the model
176
177 # Define the model
178
179 # Define the model
180
181 # Define the model
182
183 # Define the model
184
185 # Define the model
186
187 # Define the model
188
189 # Define the model
190
191 # Define the model
192
193 # Define the model
194
195 # Define the model
196
197 # Define the model
198
199 # Define the model
200
201 # Define the model
202
203 # Define the model
204
205 # Define the model
206
207 # Define the model
208
209 # Define the model
210
211 # Define the model
212
213 # Define the model
214
215 # Define the model
216
217 # Define the model
218
219 # Define the model
220
221 # Define the model
222
223 # Define the model
224
225 # Define the model
226
227 # Define the model
228
229 # Define the model
230
231 # Define the model
232
233 # Define the model
234
235 # Define the model
236
237 # Define the model
238
239 # Define the model
240
241 # Define the model
242
243 # Define the model
244
245 # Define the model
246
247 # Define the model
248
249 # Define the model
250
251 # Define the model
252
253 # Define the model
254
255 # Define the model
256
257 # Define the model
258
259 # Define the model
260
261 # Define the model
262
263 # Define the model
264
265 # Define the model
266
267 # Define the model
268
269 # Define the model
270
271 # Define the model
272
273 # Define the model
274
275 # Define the model
276
277 # Define the model
278
279 # Define the model
280
281 # Define the model
282
283 # Define the model
284
285 # Define the model
286
287 # Define the model
288
289 # Define the model
290
291 # Define the model
292
293 # Define the model
294
295 # Define the model
296
297 # Define the model
298
299 # Define the model
300
301 # Define the model
302
303 # Define the model
304
305 # Define the model
306
307 # Define the model
308
309 # Define the model
310
311 # Define the model
312
313 # Define the model
314
315 # Define the model
316
317 # Define the model
318
319 # Define the model
320
321 # Define the model
322
323 # Define the model
324
325 # Define the model
326
327 # Define the model
328
329 # Define the model
330
331 # Define the model
332
333 # Define the model
334
335 # Define the model
336
337 # Define the model
338
339 # Define the model
340
341 # Define the model
342
343 # Define the model
344
345 # Define the model
346
347 # Define the model
348
349 # Define the model
350
351 # Define the model
352
353 # Define the model
354
355 # Define the model
356
357 # Define the model
358
359 # Define the model
360
361 # Define the model
362
363 # Define the model
364
365 # Define the model
366
367 # Define the model
368
369 # Define the model
370
371 # Define the model
372
373 # Define the model
374
375 # Define the model
376
377 # Define the model
378
379 # Define the model
380
381 # Define the model
382
383 # Define the model
384
385 # Define the model
386
387 # Define the model
388
389 # Define the model
390
391 # Define the model
392
393 # Define the model
394
395 # Define the model
396
397 # Define the model
398
399 # Define the model
400
401 # Define the model
402
403 # Define the model
404
405 # Define the model
406
407 # Define the model
408
409 # Define the model
410
411 # Define the model
412
413 # Define the model
414
415 # Define the model
416
417 # Define the model
418
419 # Define the model
420
421 # Define the model
422
423 # Define the model
424
425 # Define the model
426
427 # Define the model
428
429 # Define the model
430
431 # Define the model
432
433 # Define the model
434
435 # Define the model
436
437 # Define the model
438
439 # Define the model
440
441 # Define the model
442
443 # Define the model
444
445 # Define the model
446
447 # Define the model
448
449 # Define the model
450
451 # Define the model
452
453 # Define the model
454
455 # Define the model
456
457 # Define the model
458
459 # Define the model
460
461 # Define the model
462
463 # Define the model
464
465 # Define the model
466
467 # Define the model
468
469 # Define the model
470
471 # Define the model
472
473 # Define the model
474
475 # Define the model
476
477 # Define the model
478
479 # Define the model
480
481 # Define the model
482
483 # Define the model
484
485 # Define the model
486
487 # Define the model
488
489 # Define the model
490
491 # Define the model
492
493 # Define the model
494
495 # Define the model
496
497 # Define the model
498
499 # Define the model
500
501 # Define the model
502
503 # Define the model
504
505 # Define the model
506
507 # Define the model
508
509 # Define the model
510
511 # Define the model
512
513 # Define the model
514
515 # Define the model
516
517 # Define the model
518
519 # Define the model
520
521 # Define the model
522
523 # Define the model
524
525 # Define the model
526
527 # Define the model
528
529 # Define the model
530
531 # Define the model
532
533 # Define the model
534
535 # Define the model
536
537 # Define the model
538
539 # Define the model
540
541 # Define the model
542
543 # Define the model
544
545 # Define the model
546
547 # Define the model
548
549 # Define the model
550
551 # Define the model
552
553 # Define the model
554
555 # Define the model
556
557 # Define the model
558
559 # Define the model
560
561 # Define the model
562
563 # Define the model
564
565 # Define the model
566
567 # Define the model
568
569 # Define the model
570
571 # Define the model
572
573 # Define the model
574
575 # Define the model
576
577 # Define the model
578
579 # Define the model
580
581 # Define the model
582
583 # Define the model
584
585 # Define the model
586
587 # Define the model
588
589 # Define the model
590
591 # Define the model
592
593 # Define the model
594
595 # Define the model
596
597 # Define the model
598
599 # Define the model
600
601 # Define the model
602
603 # Define the model
604
605 # Define the model
606
607 # Define the model
608
609 # Define the model
610
611 # Define the model
612
613 # Define the model
614
615 # Define the model
616
617 # Define the model
618
619 # Define the model
620
621 # Define the model
622
623 # Define the model
624
625 # Define the model
626
627 # Define the model
628
629 # Define the model
630
631 # Define the model
632
633 # Define the model
634
635 # Define the model
636
637 # Define the model
638
639 # Define the model
640
641 # Define the model
642
643 # Define the model
644
645 # Define the model
646
647 # Define the model
648
649 # Define the model
650
651 # Define the model
652
653 # Define the model
654
655 # Define the model
656
657 # Define the model
658
659 # Define the model
660
661 # Define the model
662
663 # Define the model
664
665 # Define the model
666
667 # Define the model
668
669 # Define the model
670
671 # Define the model
672
673 # Define the model
674
675 # Define the model
676
677 # Define the model
678
679 # Define the model
680
681 # Define the model
682
683 # Define the model
684
685 # Define the model
686
687 # Define the model
688
689 # Define the model
690
691 # Define the model
692
693 # Define the model
694
695 # Define the model
696
697 # Define the model
698
699 # Define the model
700
701 # Define the model
702
703 # Define the model
704
705 # Define the model
706
707 # Define the model
708
709 # Define the model
710
711 # Define the model
712
713 # Define the model
714
715 # Define the model
716
717 # Define the model
718
719 # Define the model
720
721 # Define the model
722
723 # Define the model
724
725 # Define the model
726
727 # Define the model
728
729 # Define the model
730
731 # Define the model
732
733 # Define the model
734
735 # Define the model
736
737 # Define the model
738
739 # Define the model
740
741 # Define the model
742
743 # Define the model
744
745 # Define the model
746
747 # Define the model
748
749 # Define the model
750
751 # Define the model
752
753 # Define the model
754
755 # Define the model
756
757 # Define the model
758
759 # Define the model
760
761 # Define the model
762
763 # Define the model
764
765 # Define the model
766
767 # Define the model
768
769 # Define the model
770
771 # Define the model
772
773 # Define the model
774
775 # Define the model
776
777 # Define the model
778
779 # Define the model
780
781 # Define the model
782
783 # Define the model
784
785 # Define the model
786
787 # Define the model
788
789 # Define the model
790
791 # Define the model
792
793 # Define the model
794
795 # Define the model
796
797 # Define the model
798
799 # Define the model
800
801 # Define the model
802
803 # Define the model
804
805 # Define the model
806
807 # Define the model
808
809 # Define the model
810
811 # Define the model
812
813 # Define the model
814
815 # Define the model
816
817 # Define the model
818
819 # Define the model
820
821 # Define the model
822
823 # Define the model
824
825 # Define the model
826
827 # Define the model
828
829 # Define the model
830
831 # Define the model
832
833 # Define the model
834
835 # Define the model
836
837 # Define the model
838
839 # Define the model
840
841 # Define the model
842
843 # Define the model
844
845 # Define the model
846
847 # Define the model
848
849 # Define the model
850
851 # Define the model
852
853 # Define the model
854
855 # Define the model
856
857 # Define the model
858
859 # Define the model
860
861 # Define the model
862
863 # Define the model
864
865 # Define the model
866
867 # Define the model
868
869 # Define the model
870
871 # Define the model
872
873 # Define the model
874
875 # Define the model
876
877 # Define the model
878
879 # Define the model
880
881 # Define the model
882
883 # Define the model
884
885 # Define the model
886
887 # Define the model
888
889 # Define the model
890
891 # Define the model
892
893 # Define the model
894
895 # Define the model
896
897 # Define the model
898
899 # Define the model
900
901 # Define the model
902
903 # Define the model
904
905 # Define the model
906
907 # Define the model
908
909 # Define the model
910
911 # Define the model
912
913 # Define the model
914
915 # Define the model
916
917 # Define the model
918
919 # Define the model
920
921 # Define the model
922
923 # Define the model
924
925 # Define the model
926
927 # Define the model
928
929 # Define the model
930
931 # Define the model
932
933 # Define the model
934
935 # Define the model
936
937 # Define the model
938
939 # Define the model
940
941 # Define the model
942
943 # Define the model
944
945 # Define the model
946
947 # Define the model
948
949 # Define the model
950
951 # Define the model
952
953 # Define the model
954
955 # Define the model
956
957 # Define the model
958
959 # Define the model
960
961 # Define the model
962
963 # Define the model
964
965 # Define the model
966
967 # Define the model
968
969 # Define the model
970
971 # Define the model
972
973 # Define the model
974
975 # Define the model
976
977 # Define the model
978
979 # Define the model
980
981 # Define the model
982
983 # Define the model
984
985 # Define the model
986
987 # Define the model
988
989 # Define the model
990
991 # Define the model
992
993 # Define the model
994
995 # Define the model
996
997 # Define the model
998
999 # Define the model
1000

```

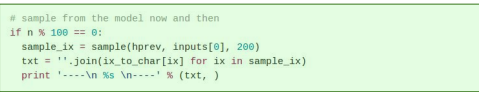


Main loop

```

801 n, p = 0, 0
802 mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
803 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
804 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
805 while True:
806     # prepare inputs (we're sweeping from left to right in steps seq_length long)
807     if p+seq_length+1 >= len(data) or n == 0:
808         hprev = np.zeros((hidden_size,1)) # reset RNN memory
809         p = 0 # go from start of data
810     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
811     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
812
813     # sample from the model now and then
814     if n % 100 == 0:
815         sample_ix = sample(hprev, inputs[0], 200)
816         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
817         print '----\n%s \n----' % (txt, )
818
819     # forward seq_length characters through the net and fetch gradient
820     loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
821     smooth_loss = smooth_loss * 0.999 + loss * 0.001
822     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
823
824     # perform parameter update with Adagrad
825     for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
826                                  [dWxh, dWhh, dWhy, dbh, dby],
827                                  [mWxh, mWhh, mWhy, mbh, mby]):
828         mem += dparam * dparam
829         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
830
831 p += seq_length # move data pointer
832 n += 1 # iteration counter

```



min-char-rnn.py gist

```

1 """
2 Example character-level language model, adapted from Andre Kardecy (2015a,b)
3 """
4
5 import sys
6
7 # Model size
8 n_vocab = 27
9 n_hidden = 128
10 n_emb = 128
11 n_layers = 2
12
13 # Data
14 data = sys.argv[1]
15 vocab = {}
16 for ch in data:
17     vocab[ch] = len(vocab)
18
19 # Embedding
20 emb = torch.nn.Embedding(n_vocab, n_emb)
21
22 # LSTM
23 lstm = torch.nn.LSTM(n_emb, n_hidden, n_layers)
24
25 # Output
26 out = torch.nn.Linear(n_hidden, n_vocab)
27
28 # Loss
29 loss = torch.nn.NLLLoss()
30
31 # Training
32 for i in range(100000):
33     # Prepare inputs (we're sweeping from left to right in steps seq_length long)
34     if i % seq_length == 0:
35         # reset RNN memory
36         hprev = torch.zeros((1, n_hidden))
37         p = 0 # go from start of data
38     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
39     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
40
41     # sample from the model now and then
42     if i % 100 == 0:
43         sample_ix = sample(hprev, inputs[0], 200)
44         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
45         print '----\n%s \n----' % (txt, )
46
47     # forward seq_length characters through the net and fetch gradient
48     loss, dxwx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
49     smooth_loss = smooth_loss * 0.999 + loss * 0.001
50     if i % 100 == 0: print 'iter %d, loss: %f' % (i, smooth_loss) # print progress
51
52     # perform parameter update with Adagrad
53     for param, dparam, mem in zip([wvx, whh, why, bh, by],
54                                 [dxwx, dwhh, dwhy, dbh, dby],
55                                 [mwxh, mwhh, mwhy, mbh, mby]):
56         mem += dparam * dparam
57         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
58
59     p += seq_length # move data pointer
60     n += 1 # iteration counter
61 """
62 """
63 """
64 """
65 """
66 """
67 """
68 """
69 """
70 """
71 """
72 """
73 """
74 """
75 """
76 """
77 """
78 """
79 """
80 """
81 """
82 """
83 """
84 """
85 """
86 """
87 """
88 """
89 """
90 """
91 """
92 """
93 """
94 """
95 """
96 """
97 """
98 """
99 """
100 """
101 """
102 """
103 """
104 """
105 """
106 """
107 """
108 """
109 """
110 """
111 """
112 """
113 """
114 """
115 """
116 """
117 """
118 """
119 """
120 """
121 """
122 """
123 """
124 """
125 """
126 """
127 """
128 """
129 """
130 """
131 """
132 """
133 """
134 """
135 """
136 """
137 """
138 """
139 """
140 """
141 """
142 """
143 """
144 """
145 """
146 """
147 """
148 """
149 """
150 """
151 """
152 """
153 """
154 """
155 """
156 """
157 """
158 """
159 """
160 """
161 """
162 """
163 """
164 """
165 """
166 """
167 """
168 """
169 """
170 """
171 """
172 """
173 """
174 """
175 """
176 """
177 """
178 """
179 """
180 """
181 """
182 """
183 """
184 """
185 """
186 """
187 """
188 """
189 """
190 """
191 """
192 """
193 """
194 """
195 """
196 """
197 """
198 """
199 """
200 """
201 """
202 """
203 """
204 """
205 """
206 """
207 """
208 """
209 """
210 """
211 """
212 """
213 """
214 """
215 """
216 """
217 """
218 """
219 """
220 """
221 """
222 """
223 """
224 """
225 """
226 """
227 """
228 """
229 """
230 """
231 """
232 """
233 """
234 """
235 """
236 """
237 """
238 """
239 """
240 """
241 """
242 """
243 """
244 """
245 """
246 """
247 """
248 """
249 """
250 """
251 """
252 """
253 """
254 """
255 """
256 """
257 """
258 """
259 """
260 """
261 """
262 """
263 """
264 """
265 """
266 """
267 """
268 """
269 """
270 """
271 """
272 """
273 """
274 """
275 """
276 """
277 """
278 """
279 """
280 """
281 """
282 """
283 """
284 """
285 """
286 """
287 """
288 """
289 """
290 """
291 """
292 """
293 """
294 """
295 """
296 """
297 """
298 """
299 """
300 """
301 """
302 """
303 """
304 """
305 """
306 """
307 """
308 """
309 """
310 """
311 """
312 """
313 """
314 """
315 """
316 """
317 """
318 """
319 """
320 """
321 """
322 """
323 """
324 """
325 """
326 """
327 """
328 """
329 """
330 """
331 """
332 """
333 """
334 """
335 """
336 """
337 """
338 """
339 """
340 """
341 """
342 """
343 """
344 """
345 """
346 """
347 """
348 """
349 """
350 """
351 """
352 """
353 """
354 """
355 """
356 """
357 """
358 """
359 """
360 """
361 """
362 """
363 """
364 """
365 """
366 """
367 """
368 """
369 """
370 """
371 """
372 """
373 """
374 """
375 """
376 """
377 """
378 """
379 """
380 """
381 """
382 """
383 """
384 """
385 """
386 """
387 """
388 """
389 """
390 """
391 """
392 """
393 """
394 """
395 """
396 """
397 """
398 """
399 """
400 """
401 """
402 """
403 """
404 """
405 """
406 """
407 """
408 """
409 """
410 """
411 """
412 """
413 """
414 """
415 """
416 """
417 """
418 """
419 """
420 """
421 """
422 """
423 """
424 """
425 """
426 """
427 """
428 """
429 """
430 """
431 """
432 """
433 """
434 """
435 """
436 """
437 """
438 """
439 """
440 """
441 """
442 """
443 """
444 """
445 """
446 """
447 """
448 """
449 """
450 """
451 """
452 """
453 """
454 """
455 """
456 """
457 """
458 """
459 """
460 """
461 """
462 """
463 """
464 """
465 """
466 """
467 """
468 """
469 """
470 """
471 """
472 """
473 """
474 """
475 """
476 """
477 """
478 """
479 """
480 """
481 """
482 """
483 """
484 """
485 """
486 """
487 """
488 """
489 """
490 """
491 """
492 """
493 """
494 """
495 """
496 """
497 """
498 """
499 """
500 """
501 """
502 """
503 """
504 """
505 """
506 """
507 """
508 """
509 """
510 """
511 """
512 """
513 """
514 """
515 """
516 """
517 """
518 """
519 """
520 """
521 """
522 """
523 """
524 """
525 """
526 """
527 """
528 """
529 """
530 """
531 """
532 """
533 """
534 """
535 """
536 """
537 """
538 """
539 """
540 """
541 """
542 """
543 """
544 """
545 """
546 """
547 """
548 """
549 """
550 """
551 """
552 """
553 """
554 """
555 """
556 """
557 """
558 """
559 """
560 """
561 """
562 """
563 """
564 """
565 """
566 """
567 """
568 """
569 """
570 """
571 """
572 """
573 """
574 """
575 """
576 """
577 """
578 """
579 """
580 """
581 """
582 """
583 """
584 """
585 """
586 """
587 """
588 """
589 """
590 """
591 """
592 """
593 """
594 """
595 """
596 """
597 """
598 """
599 """
600 """
601 """
602 """
603 """
604 """
605 """
606 """
607 """
608 """
609 """
610 """
611 """
612 """
613 """
614 """
615 """
616 """
617 """
618 """
619 """
620 """
621 """
622 """
623 """
624 """
625 """
626 """
627 """
628 """
629 """
630 """
631 """
632 """
633 """
634 """
635 """
636 """
637 """
638 """
639 """
640 """
641 """
642 """
643 """
644 """
645 """
646 """
647 """
648 """
649 """
650 """
651 """
652 """
653 """
654 """
655 """
656 """
657 """
658 """
659 """
660 """
661 """
662 """
663 """
664 """
665 """
666 """
667 """
668 """
669 """
670 """
671 """
672 """
673 """
674 """
675 """
676 """
677 """
678 """
679 """
680 """
681 """
682 """
683 """
684 """
685 """
686 """
687 """
688 """
689 """
690 """
691 """
692 """
693 """
694 """
695 """
696 """
697 """
698 """
699 """
700 """
701 """
702 """
703 """
704 """
705 """
706 """
707 """
708 """
709 """
710 """
711 """
712 """
713 """
714 """
715 """
716 """
717 """
718 """
719 """
720 """
721 """
722 """
723 """
724 """
725 """
726 """
727 """
728 """
729 """
730 """
731 """
732 """
733 """
734 """
735 """
736 """
737 """
738 """
739 """
740 """
741 """
742 """
743 """
744 """
745 """
746 """
747 """
748 """
749 """
750 """
751 """
752 """
753 """
754 """
755 """
756 """
757 """
758 """
759 """
760 """
761 """
762 """
763 """
764 """
765 """
766 """
767 """
768 """
769 """
770 """
771 """
772 """
773 """
774 """
775 """
776 """
777 """
778 """
779 """
780 """
781 """
782 """
783 """
784 """
785 """
786 """
787 """
788 """
789 """
790 """
791 """
792 """
793 """
794 """
795 """
796 """
797 """
798 """
799 """
800 """
801 """
802 """
803 """
804 """
805 """
806 """
807 """
808 """
809 """
810 """
811 """
812 """
813 """
814 """
815 """
816 """
817 """
818 """
819 """
820 """
821 """
822 """
823 """
824 """
825 """
826 """
827 """
828 """
829 """
830 """
831 """
832 """
833 """
834 """
835 """
836 """
837 """
838 """
839 """
840 """
841 """
842 """
843 """
844 """
845 """
846 """
847 """
848 """
849 """
850 """
851 """
852 """
853 """
854 """
855 """
856 """
857 """
858 """
859 """
860 """
861 """
862 """
863 """
864 """
865 """
866 """
867 """
868 """
869 """
870 """
871 """
872 """
873 """
874 """
875 """
876 """
877 """
878 """
879 """
880 """
881 """
882 """
883 """
884 """
885 """
886 """
887 """
888 """
889 """
890 """
891 """
892 """
893 """
894 """
895 """
896 """
897 """
898 """
899 """
900 """
901 """
902 """
903 """
904 """
905 """
906 """
907 """
908 """
909 """
910 """
911 """
912 """
913 """
914 """
915 """
916 """
917 """
918 """
919 """
920 """
921 """
922 """
923 """
924 """
925 """
926 """
927 """
928 """
929 """
930 """
931 """
932 """
933 """
934 """
935 """
936 """
937 """
938 """
939 """
940 """
941 """
942 """
943 """
944 """
945 """
946 """
947 """
948 """
949 """
950 """
951 """
952 """
953 """
954 """
955 """
956 """
957 """
958 """
959 """
960 """
961 """
962 """
963 """
964 """
965 """
966 """
967 """
968 """
969 """
970 """
971 """
972 """
973 """
974 """
975 """
976 """
977 """
978 """
979 """
980 """
981 """
982 """
983 """
984 """
985 """
986 """
987 """
988 """
989 """
990 """
991 """
992 """
993 """
994 """
995 """
996 """
997 """
998 """
999 """
1000 """

```

Main loop

```

801 n, p = 0, 0
802 mwvx, mwhh, mwhy = np.zeros_like(wvx), np.zeros_like(whh), np.zeros_like(why)
803 mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
804 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
805 while True:
806     # prepare inputs (we're sweeping from left to right in steps seq_length long)
807     if p+seq_length+1 >= len(data) or n == 0:
808         hprev = np.zeros((hidden_size,1)) # reset RNN memory
809         p = 0 # go from start of data
810     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
811     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
812
813     # sample from the model now and then
814     if n % 100 == 0:
815         sample_ix = sample(hprev, inputs[0], 200)
816         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
817         print '----\n%s \n----' % (txt, )
818
819     # forward seq_length characters through the net and fetch gradient
820     loss, dxwx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
821     smooth_loss = smooth_loss * 0.999 + loss * 0.001
822     if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
823
824     # perform parameter update with Adagrad
825     for param, dparam, mem in zip([wvx, whh, why, bh, by],
826                                 [dxwx, dwhh, dwhy, dbh, dby],
827                                 [mwxh, mwhh, mwhy, mbh, mby]):
828         mem += dparam * dparam
829         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update
830
831     p += seq_length # move data pointer
832     n += 1 # iteration counter

```

```

825     for param, dparam, mem in zip([wvx, whh, why, bh, by],
826                                 [dxwx, dwhh, dwhy, dbh, dby],
827                                 [mwxh, mwhh, mwhy, mbh, mby]):
828         mem += dparam * dparam
829         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

```



```

825     for param, dparam, mem in zip([wvx, whh, why, bh, by],
826                                 [dxwx, dwhh, dwhy, dbh, dby],
827                                 [mwxh, mwhh, mwhy, mbh, mby]):
828         mem += dparam * dparam
829         param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

```

min-char-rnn.py gist

```

1 from __future__ import division, print_function, unicode_literals
2 import random
3 import sys
4 import time
5
6 # Global parameters
7 # Input and Output characters
8 INP_CHARACTERS = 'abcdefghijklmnopqrstuvwxyz0123456789 !@#$%^&*()-+=,./:;\'"~`|_{}[]\';
9 OUT_CHARACTERS = INP_CHARACTERS
10 # Embedding size
11 EMBEDDING_DIM = 100
12 # Hidden state size
13 HIDDEN_DIM = 100
14 # Number of hidden states
15 NUM_HIDDEN_STATES = 1000
16 # Number of layers
17 NUM_LAYERS = 2
18 # Learning rate
19 LEARNING_RATE = 0.01
20 # Batch size
21 BATCH_SIZE = 64
22 # Number of training steps
23 NUM_STEPS = 100000
24 # Print loss every
25 PRINT_LOSS_INTERVAL = 10000
26 # Print accuracy every
27 PRINT_ACC_INTERVAL = 10000
28 # Print perplexity every
29 PRINT_PERP_INTERVAL = 10000
30 # Print time every
31 PRINT_TIME_INTERVAL = 10000
32 # Print validation loss every
33 PRINT_VALID_LOSS_INTERVAL = 10000
34 # Print validation accuracy every
35 PRINT_VALID_ACC_INTERVAL = 10000
36 # Print validation perplexity every
37 PRINT_VALID_PERP_INTERVAL = 10000
38 # Print validation time every
39 PRINT_VALID_TIME_INTERVAL = 10000
39
40 # Training loop
41 def main():
42     # Initialize model
43     model = Model(EMBEDDING_DIM, HIDDEN_DIM, NUM_HIDDEN_STATES, NUM_LAYERS)
44     # Initialize parameters
45     model.initialize_parameters()
46     # Initialize data
47     data_loader = DataLoader(INP_CHARACTERS, OUT_CHARACTERS, BATCH_SIZE)
48     # Training loop
49     for step in range(NUM_STEPS):
50         # Forward pass
51         loss, grads, hprev = model.lossFun(data_loader.get_batch())
52         # Backward pass
53         model.backward(loss, grads, hprev)
54         # Update parameters
55         model.update_parameters(grads)
56         # Print loss
57         if step % PRINT_LOSS_INTERVAL == 0:
58             print('Step: %d, Loss: %f' % (step, loss))
59         # Print accuracy
60         if step % PRINT_ACC_INTERVAL == 0:
61             print('Step: %d, Accuracy: %f' % (step, data_loader.get_accuracy()))
62         # Print perplexity
63         if step % PRINT_PERP_INTERVAL == 0:
64             print('Step: %d, Perplexity: %f' % (step, data_loader.get_perplexity()))
65         # Print time
66         if step % PRINT_TIME_INTERVAL == 0:
67             print('Step: %d, Time: %f' % (step, time.time() - start_time))
68         # Print validation loss
69         if step % PRINT_VALID_LOSS_INTERVAL == 0:
70             print('Step: %d, Validation Loss: %f' % (step, model.validate()))
71         # Print validation accuracy
72         if step % PRINT_VALID_ACC_INTERVAL == 0:
73             print('Step: %d, Validation Accuracy: %f' % (step, model.validate_acc()))
74         # Print validation perplexity
75         if step % PRINT_VALID_PERP_INTERVAL == 0:
76             print('Step: %d, Validation Perplexity: %f' % (step, model.validate_perp()))
77         # Print validation time
78         if step % PRINT_VALID_TIME_INTERVAL == 0:
79             print('Step: %d, Validation Time: %f' % (step, model.validate_time()))
80     # Final loss
81     print('Final Loss: %f' % model.get_loss())
82     # Final accuracy
83     print('Final Accuracy: %f' % data_loader.get_accuracy())
84     # Final perplexity
85     print('Final Perplexity: %f' % data_loader.get_perplexity())
86     # Final time
87     print('Final Time: %f' % time.time() - start_time)
88
89 if __name__ == '__main__':
90     main()

```



Loss function

- forward pass (compute loss)
- backward pass (compute param gradient)

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs, targets are both list of integers.
30     hprev is Nx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36
37     # Forward pass
38     for t in xrange(len(inputs)):
39         xs[t] = np.zeros((vocab_size, 1)) # encode in 1-of-k representation
40         xs[t][inputs[t]] = 1
41         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
42         ys[t] = np.dot(Wyh, hs[t]) + by # unnormalized log probabilities for next chars
43         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
44         loss += -np.log(ps[t][targets[t], 0]) # softmax (cross-entropy loss)
45
46     # Backward pass: compute gradients going backwards
47     dwhx, dwhh, dwhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Wyh)
48     dbh, dby = np.zeros_like(bh), np.zeros_like(by)
49     dhnext = np.zeros_like(hs[0])
50     for t in reversed(xrange(len(inputs))):
51         dy = np.copy(ps[t])
52         dy[targets[t]] -= 1 # backprop into y
53         dwhy += np.dot(dy, hs[t].T)
54         dby = dy
55         dh = np.dot(Wyh.T, dy) + dhnext # backprop into h
56         ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
57         dbh = ddraw
58         dwhx += np.dot(ddraw, xs[t].T)
59         dwhh += np.dot(ddraw, hs[t-1].T)
60         dhnext = np.dot(Whh.T, ddraw)
61
62     for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
63         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
64     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

min-char-rnn.py gist

```

1 # Import packages
2 import numpy as np
3 import random
4 import time
5 from keras.models import Sequential
6 from keras.layers import SimpleRNN
7 from keras.optimizers import RMSprop
8 from keras.callbacks import ModelCheckpoint
9 from keras.utils.np_utils import to_categorical
10 from keras.preprocessing.text import Tokenizer
11 from keras.preprocessing.sequence import pad_sequences
12 from keras.callbacks import TensorBoard
13 from keras.callbacks import LearningRateScheduler
14 from keras.callbacks import EarlyStopping
15 from keras.callbacks import ReduceLROnPlateau
16 from keras.callbacks import Callback
17 from keras.callbacks import ProgbarLogger
18 from keras.callbacks import History
19 from keras.callbacks import CSVLogger
20 from keras.callbacks import TensorBoard
21 from keras.callbacks import LearningRateScheduler
22 from keras.callbacks import EarlyStopping
23 from keras.callbacks import ReduceLROnPlateau
24 from keras.callbacks import Callback
25 from keras.callbacks import ProgbarLogger
26 from keras.callbacks import History
27 from keras.callbacks import CSVLogger
28 from keras.callbacks import TensorBoard
29 from keras.callbacks import LearningRateScheduler
30 from keras.callbacks import EarlyStopping
31 from keras.callbacks import ReduceLROnPlateau
32 from keras.callbacks import Callback
33 from keras.callbacks import ProgbarLogger
34 from keras.callbacks import History
35 from keras.callbacks import CSVLogger
36 from keras.callbacks import TensorBoard
37 from keras.callbacks import LearningRateScheduler
38 from keras.callbacks import EarlyStopping
39 from keras.callbacks import ReduceLROnPlateau
40 from keras.callbacks import Callback
41 from keras.callbacks import ProgbarLogger
42 from keras.callbacks import History
43 from keras.callbacks import CSVLogger
44 from keras.callbacks import TensorBoard
45 from keras.callbacks import LearningRateScheduler
46 from keras.callbacks import EarlyStopping
47 from keras.callbacks import ReduceLROnPlateau
48 from keras.callbacks import Callback
49 from keras.callbacks import ProgbarLogger
50 from keras.callbacks import History
51 from keras.callbacks import CSVLogger
52 from keras.callbacks import TensorBoard
53 from keras.callbacks import LearningRateScheduler
54 from keras.callbacks import EarlyStopping
55 from keras.callbacks import ReduceLROnPlateau
56 from keras.callbacks import Callback
57 from keras.callbacks import ProgbarLogger
58 from keras.callbacks import History
59 from keras.callbacks import CSVLogger
60 from keras.callbacks import TensorBoard
61 from keras.callbacks import LearningRateScheduler
62 from keras.callbacks import EarlyStopping
63 from keras.callbacks import ReduceLROnPlateau
64 from keras.callbacks import Callback
65 from keras.callbacks import ProgbarLogger
66 from keras.callbacks import History
67 from keras.callbacks import CSVLogger
68 from keras.callbacks import TensorBoard
69 from keras.callbacks import LearningRateScheduler
70 from keras.callbacks import EarlyStopping
71 from keras.callbacks import ReduceLROnPlateau
72 from keras.callbacks import Callback
73 from keras.callbacks import ProgbarLogger
74 from keras.callbacks import History
75 from keras.callbacks import CSVLogger
76 from keras.callbacks import TensorBoard
77 from keras.callbacks import LearningRateScheduler
78 from keras.callbacks import EarlyStopping
79 from keras.callbacks import ReduceLROnPlateau
80 from keras.callbacks import Callback
81 from keras.callbacks import ProgbarLogger
82 from keras.callbacks import History
83 from keras.callbacks import CSVLogger
84 from keras.callbacks import TensorBoard
85 from keras.callbacks import LearningRateScheduler
86 from keras.callbacks import EarlyStopping
87 from keras.callbacks import ReduceLROnPlateau
88 from keras.callbacks import Callback
89 from keras.callbacks import ProgbarLogger
90 from keras.callbacks import History
91 from keras.callbacks import CSVLogger
92 from keras.callbacks import TensorBoard
93 from keras.callbacks import LearningRateScheduler
94 from keras.callbacks import EarlyStopping
95 from keras.callbacks import ReduceLROnPlateau
96 from keras.callbacks import Callback
97 from keras.callbacks import ProgbarLogger
98 from keras.callbacks import History
99 from keras.callbacks import CSVLogger
100 from keras.callbacks import TensorBoard
    
```

```

101 # Create data
102 # Create a list of all characters in the text
103 chars = sorted(set(text))
104 char_to_int = dict((c,i) for (i,c) in enumerate(chars))
105 int_to_char = dict((i,c) for (i,c) in enumerate(chars))
106 vocab_size = len(chars)
107 # Create a tokenizer
108 tokenizer = Tokenizer(char_to_int)
109 tokenizer.fit_on_texts(text)
110 # Create a data generator
111 def generate_data(text, char_to_int, vocab_size, max_len):
112     # Create a list of all characters in the text
113     chars = sorted(set(text))
114     char_to_int = dict((c,i) for (i,c) in enumerate(chars))
115     int_to_char = dict((i,c) for (i,c) in enumerate(chars))
116     vocab_size = len(chars)
117     # Create a tokenizer
118     tokenizer = Tokenizer(char_to_int)
119     tokenizer.fit_on_texts(text)
120     # Create a data generator
121     data_generator = DataGenerator(text, char_to_int, vocab_size, max_len)
122     return data_generator
123 # Create a model
124 model = Sequential()
125 model.add(SimpleRNN(128, input_shape=(max_len, vocab_size)))
126 model.compile(optimizer=RMSprop())
127 # Create a callback
128 callbacks = [ModelCheckpoint('min-char-rnn.h5'),
129             TensorBoard('min-char-rnn'),
130             LearningRateScheduler(lambda x: 1e-4 * x),
131             EarlyStopping(monitor='loss', patience=10),
132             ReduceLROnPlateau(monitor='loss', patience=10)]
133 # Train the model
134 model.fit_generator(generate_data(text, char_to_int, vocab_size, max_len),
135                  steps_per_epoch=1000,
136                  epochs=100,
137                  callbacks=callbacks)
138 # Save the model
139 model.save('min-char-rnn.h5')
140 # Load the model
141 model = load_model('min-char-rnn.h5')
142 # Predict the next character
143 def predict(text, char_to_int, vocab_size, max_len):
144     # Create a list of all characters in the text
145     chars = sorted(set(text))
146     char_to_int = dict((c,i) for (i,c) in enumerate(chars))
147     int_to_char = dict((i,c) for (i,c) in enumerate(chars))
148     vocab_size = len(chars)
149     # Create a tokenizer
150     tokenizer = Tokenizer(char_to_int)
151     tokenizer.fit_on_texts(text)
152     # Create a data generator
153     data_generator = DataGenerator(text, char_to_int, vocab_size, max_len)
154     return data_generator
155 # Create a model
156 model = Sequential()
157 model.add(SimpleRNN(128, input_shape=(max_len, vocab_size)))
158 model.compile(optimizer=RMSprop())
159 # Create a callback
160 callbacks = [ModelCheckpoint('min-char-rnn.h5'),
161             TensorBoard('min-char-rnn'),
162             LearningRateScheduler(lambda x: 1e-4 * x),
163             EarlyStopping(monitor='loss', patience=10),
164             ReduceLROnPlateau(monitor='loss', patience=10)]
165 # Train the model
166 model.fit_generator(generate_data(text, char_to_int, vocab_size, max_len),
167                  steps_per_epoch=1000,
168                  epochs=100,
169                  callbacks=callbacks)
170 # Save the model
171 model.save('min-char-rnn.h5')
172 # Load the model
173 model = load_model('min-char-rnn.h5')
174 # Predict the next character
175 def predict(text, char_to_int, vocab_size, max_len):
176     # Create a list of all characters in the text
177     chars = sorted(set(text))
178     char_to_int = dict((c,i) for (i,c) in enumerate(chars))
179     int_to_char = dict((i,c) for (i,c) in enumerate(chars))
180     vocab_size = len(chars)
181     # Create a tokenizer
182     tokenizer = Tokenizer(char_to_int)
183     tokenizer.fit_on_texts(text)
184     # Create a data generator
185     data_generator = DataGenerator(text, char_to_int, vocab_size, max_len)
186     return data_generator
    
```

```

27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wxh, xs[t]) + np.dot(whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
    
```

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Softmax classifier

min-char-rnn.py gist

```

1 #!/usr/bin/env python
2 """
3 Example character-level language model, written by Andrej Kardec (2016)
4 """
5
6 # Imports
7 import sys
8
9 # Model class
10 class Model(object):
11     """
12     A simple character-level language model.
13     """
14     def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
15         self.vocab_size = vocab_size
16         self.embedding_dim = embedding_dim
17         self.hidden_dim = hidden_dim
18         self.output_dim = output_dim
19
20     def build(self):
21         """
22         Build the model layers.
23         """
24         self.W = np.random.randn(self.embedding_dim, self.vocab_size)
25         self.U = np.random.randn(self.hidden_dim, self.embedding_dim)
26         self.V = np.random.randn(self.output_dim, self.embedding_dim)
27
28     def forward(self, x):
29         """
30         Forward pass: compute hidden and output states.
31         """
32         # Embedding
33         x_emb = self.W[x]
34
35         # Hidden state
36         h = np.zeros(self.hidden_dim)
37         for i in range(len(x)):
38             h = np.tanh(self.U[h] + x_emb[i])
39
40         # Output state
41         y = self.V[h]
42         return y
43
44     def backward(self, y, dy):
45         """
46         Backward pass: compute gradients.
47         """
48         # Output gradient
49         dy = dy
50
51         # Hidden state gradient
52         dh = np.zeros(self.hidden_dim)
53         for i in range(len(x)-1, -1, -1):
54             dh = self.U[h[i]] + self.V[h[i]] * dy
55             dy = dy * (1 - hs[i]**2)
56
57         # Input gradient
58         dx = self.W[h[i]] * dy
59         return dx
60
61     def clip(self, x, clip_value):
62         """
63         Clip the gradient.
64         """
65         return np.clip(x, -clip_value, clip_value)
66
67     def loss(self, x, y):
68         """
69         Compute the loss.
70         """
71         y_hat = self.forward(x)
72         loss = -np.sum(y * np.log(y_hat))
73         return loss
74
75     def train(self, x, y, clip_value):
76         """
77         Train the model.
78         """
79         loss = self.loss(x, y)
80         dy = -y / y_hat
81         dx = self.backward(y_hat, dy)
82         dx = self.clip(dx, clip_value)
83         self.W += dx
84         self.U += dx
85         self.V += dx
86
87     def predict(self, x):
88         """
89         Predict the next character.
90         """
91         y_hat = self.forward(x)
92         return np.argmax(y_hat)
93
94     def generate(self, x, num_chars):
95         """
96         Generate a sequence of characters.
97         """
98         x = x + self.predict(x)
99         for i in range(num_chars):
100             x = x + self.predict(x)
101         return x
102
103     def __str__(self):
104         """
105         Print the model parameters.
106         """
107         return str(self.W) + str(self.U) + str(self.V)
108
109     def __repr__(self):
110         """
111         Print the model class name.
112         """
113         return self.__class__.__name__
114
115 if __name__ == '__main__':
116     # Example usage
117     vocab_size = 27
118     embedding_dim = 100
119     hidden_dim = 100
120     output_dim = 27
121
122     model = Model(vocab_size, embedding_dim, hidden_dim, output_dim)
123     model.build()
124
125     x = 'hello world'
126     y = 'ello world'
127
128     model.train(x, y, 5)
129
130     x = 'hello world'
131     y_hat = model.predict(x)
132     print(y_hat)
133
134     x = 'hello world'
135     y_hat = model.generate(x, 10)
136     print(y_hat)
137
138     print(model)
139
140 """
141 """

```

```

142 """
143 Example character-level language model, written by Andrej Kardec (2016)
144 """
145
146 # Imports
147 import sys
148
149 # Model class
150 class Model(object):
151     """
152     A simple character-level language model.
153     """
154     def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
155         self.vocab_size = vocab_size
156         self.embedding_dim = embedding_dim
157         self.hidden_dim = hidden_dim
158         self.output_dim = output_dim
159
160     def build(self):
161         """
162         Build the model layers.
163         """
164         self.W = np.random.randn(self.embedding_dim, self.vocab_size)
165         self.U = np.random.randn(self.hidden_dim, self.embedding_dim)
166         self.V = np.random.randn(self.output_dim, self.embedding_dim)
167
168     def forward(self, x):
169         """
170         Forward pass: compute hidden and output states.
171         """
172         # Embedding
173         x_emb = self.W[x]
174
175         # Hidden state
176         h = np.zeros(self.hidden_dim)
177         for i in range(len(x)):
178             h = np.tanh(self.U[h] + x_emb[i])
179
180         # Output state
181         y = self.V[h]
182         return y
183
184     def backward(self, y, dy):
185         """
186         Backward pass: compute gradients.
187         """
188         # Output gradient
189         dy = dy
190
191         # Hidden state gradient
192         dh = np.zeros(self.hidden_dim)
193         for i in range(len(x)-1, -1, -1):
194             dh = self.U[h[i]] + self.V[h[i]] * dy
195             dy = dy * (1 - hs[i]**2)
196
197         # Input gradient
198         dx = self.W[h[i]] * dy
199         return dx
200
201     def clip(self, x, clip_value):
202         """
203         Clip the gradient.
204         """
205         return np.clip(x, -clip_value, clip_value)
206
207     def loss(self, x, y):
208         """
209         Compute the loss.
210         """
211         y_hat = self.forward(x)
212         loss = -np.sum(y * np.log(y_hat))
213         return loss
214
215     def train(self, x, y, clip_value):
216         """
217         Train the model.
218         """
219         loss = self.loss(x, y)
220         dy = -y / y_hat
221         dx = self.backward(y_hat, dy)
222         dx = self.clip(dx, clip_value)
223         self.W += dx
224         self.U += dx
225         self.V += dx
226
227     def predict(self, x):
228         """
229         Predict the next character.
230         """
231         y_hat = self.forward(x)
232         return np.argmax(y_hat)
233
234     def generate(self, x, num_chars):
235         """
236         Generate a sequence of characters.
237         """
238         x = x + self.predict(x)
239         for i in range(num_chars):
240             x = x + self.predict(x)
241         return x
242
243     def __str__(self):
244         """
245         Print the model parameters.
246         """
247         return str(self.W) + str(self.U) + str(self.V)
248
249     def __repr__(self):
250         """
251         Print the model class name.
252         """
253         return self.__class__.__name__
254
255 if __name__ == '__main__':
256     # Example usage
257     vocab_size = 27
258     embedding_dim = 100
259     hidden_dim = 100
260     output_dim = 27
261
262     model = Model(vocab_size, embedding_dim, hidden_dim, output_dim)
263     model.build()
264
265     x = 'hello world'
266     y = 'ello world'
267
268     model.train(x, y, 5)
269
270     x = 'hello world'
271     y_hat = model.predict(x)
272     print(y_hat)
273
274     x = 'hello world'
275     y_hat = model.generate(x, 10)
276     print(y_hat)
277
278     print(model)
279
280 """
281 """

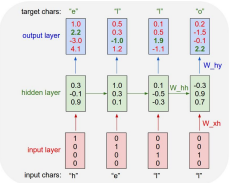
```

```

44 # backward pass: compute gradients going backwards
45 dwxh, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
46 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47 dhnext = np.zeros_like(hs[0])
48 for t in reversed(xrange(len(inputs))):
49     dy = np.copy(ps[t])
50     dy[targets[t]] -= 1 # backprop into y
51     dwhy += np.dot(dy, hs[t].T)
52     dby += dy
53     dh = np.dot(why.T, dy) + dhnext # backprop into h
54     dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55     dbh += dhraw
56     dwxh += np.dot(dhraw, xs[t].T)
57     dwhh += np.dot(dhraw, hs[t-1].T)
58     dhnext = np.dot(whh.T, dhraw)
59 for dparam in [dwxh, dwhh, dwhy, dbh, dby]:
60     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61 return loss, dwxh, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]

```

recall:



min-char-rnn.py gist

```

1 #
2 # LICENSE: UNLICENSED PUBLIC DOMAIN (LICENSE BY MAKING PUBLIC) (UNLICENSED)
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 #
31 #
32 #
33 #
34 #
35 #
36 #
37 #
38 #
39 #
40 #
41 #
42 #
43 #
44 #
45 #
46 #
47 #
48 #
49 #
50 #
51 #
52 #
53 #
54 #
55 #
56 #
57 #
58 #
59 #
60 #
61 #
62 #
63 #
64 #
65 #
66 #
67 #
68 #
69 #
70 #
71 #
72 #
73 #
74 #
75 #
76 #
77 #
78 #
79 #
80 #
81 #
82 #
83 #
84 #
85 #
86 #
87 #
88 #
89 #
90 #
91 #
92 #
93 #
94 #
95 #
96 #
97 #
98 #
99 #
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #

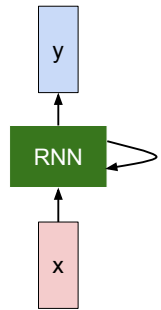
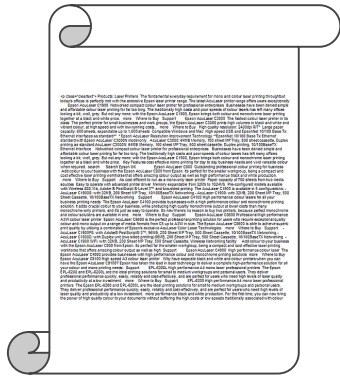
```



```

63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73         y = np.dot(Wyh, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79     return ixes

```



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 34 8 Feb 2016

Demo

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
 plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng

↓
 train more

"Tmont thithey" fomesscerliund
 Keushey. Thom here
 sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
 coaniogennc Phe lism thond hon at. MeiDimorotio in ther thize."

↓
 train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
 her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
 how, and Gogition is so overelical and offer.

↓
 train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
 princess, Princess Mary was easier, fed in had oftened him.
 Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little grain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.


VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:





















O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

open source textbook on algebraic geometry

 **The Stacks Project**

[home](#) [about](#) [tags explained](#) [tag lookup](#) [browse](#) [search](#) [bibliography](#) [recent comments](#) [blog](#) [add slogans](#)

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex 	pdf 
	2. Conventions	online	tex 	pdf 
	3. Set Theory	online	tex 	pdf 
	4. Categories	online	tex 	pdf 
	5. Topology	online	tex 	pdf 
	6. Sheaves on Spaces	online	tex 	pdf 
	7. Sites and Sheaves	online	tex 	pdf 
	8. Stacks	online	tex 	pdf 
	9. Fields	online	tex 	pdf 
	10. Commutative Algebra	online	tex 	pdf 

Parts

- [Preliminaries](#)
- [Schemes](#)
- [Topics in Scheme Theory](#)
- [Algebraic Spaces](#)
- [Topics in Geometry](#)
- [Deformation Theory](#)
- [Algebraic Stacks](#)
- [Miscellany](#)

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source 

For $\bigoplus_{n=1, \dots, m}$ where $\mathcal{L}_{m^*} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparably in the fibre product covering we have to prove the lemma generated by $\prod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ?? . Hence we obtain a scheme S and any open subset $W \subset U$ in $Sch(G)$ such that $\text{Spec}(R')$ $\rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $GL_{S'}(Z'/S')$ and we win. \square

To prove study we see that \mathcal{F}_U is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_i exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\tilde{M}^* = Z^* \otimes_{\text{Spec}(k)} \mathcal{O}_{S_n} - i_{X'}^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (Sch/S)_{fppf}^{opp}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ?? . It may replace S by $X_{spaces, \acute{e}tale}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ?? . Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (2) by the construction in the description.

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\text{Proj}_X(A) = \text{Spec}(B)$ over U compatible with the complex

$$\text{Set}(A) = \Gamma(X, \mathcal{O}_{X, \mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S . Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X . But given a scheme U and a surjective étale morphism $U \rightarrow X$. Let $U \cap U = \prod_{i=1, \dots, n} U_i$ be the scheme X over S at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$. \square

The following lemma surjective retrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{x_0, \dots, 0}$.

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_i \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \bar{\Delta}_2$ works.

Lemma 0.3. In Situation ?? . Hence we may assume $\mathfrak{q}' = 0$.

Proof. We will use the property we see that \mathfrak{p} is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F -algebra where δ_{n+1} is a scheme over S . \square

Proof. Omitted. □

Lemma 0.1. *Let \mathcal{C} be a set of the construction.*

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. *This is an integer Z is injective.*

Proof. See Spaces, Lemma ?? □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $U \subset X$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

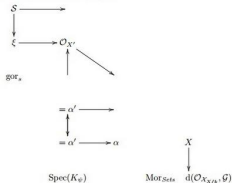
be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type \mathcal{F} . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??
A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a "field"

$$\mathcal{O}_{X, x} \rightarrow \mathcal{F}_x \rightarrow \mathcal{O}_{X, x} \rightarrow \mathcal{O}_{X, x} \rightarrow \mathcal{O}_{X, x} \rightarrow \mathcal{O}_{X, x}$$

is an isomorphism of covering of $\mathcal{O}_{X, x}$. If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S . If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum $\mathcal{O}_{X, x}$ is a closed immersion, see Lemma ??
This is a sequence of \mathcal{F} is a similar morphism.

The screenshot shows the GitHub repository for the Linux kernel source tree. At the top, there are navigation links for 'Explore', 'Gist', 'Blog', and 'Help'. The repository name is 'torvalds / linux'. Below this, there are statistics: 'Watch - 3,711', 'Star 23,054', and 'Fork 9,141'. The main content area displays the commit history, with a table listing recent merges and updates. The right sidebar contains options for 'Code', 'Pull requests' (74), 'Pulse', 'Graphs', and 'HTTPS clone URL' (https://github.com/torvalds/linux.git).

Linux kernel source tree

520,037 commits | 1 branch | 420 releases | 5,039 contributors

branch: master - linux / +

Commit	Author	Time
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	torvalds	9 hours ago
Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/hab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/ihex2hw.c: restore missing default in switch statement	2 months ago
fs	vfs: read file_handle only once in handle_to_path	4 days ago
include	Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip	a day ago
init	init: fix regression by supporting devices with major:minor:offset to...	a month ago

HTTPS clone URL
https://github.com/torvalds/linux.git

You can clone with HTTPS, SSH, or Subversion.

Cclone in Desktop

Download ZIP

```

static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}

```

Generated C code


```
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/kevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)    (func)

#define SWAP_ALLOCATE(nr)    (e)
#define emulate_sigs()    arch_get_unaligned_child()
#define access_rw(TST)    asm volatile("movd %!esp, %!0, %!3" : : "r" (0)); \
    if (__type & DO_READ)

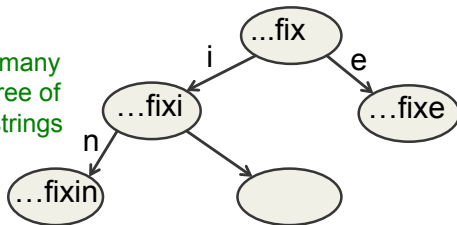
static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}
}
```

Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



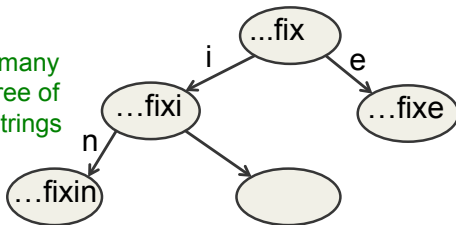
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier model does not quite catch that

Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



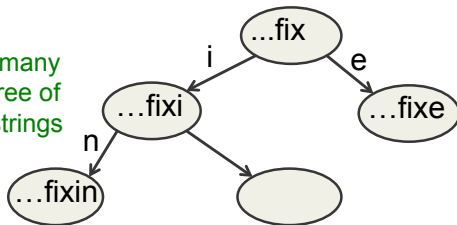
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier model does not quite catch that

Ideal tree model

An ideal model considers all previous input characters and the current character

There are exponentially many nodes in the tree of all character strings of length N .



In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- The next hidden representation needs to depend on the **conjunction** of the current character and the current hidden representation
 - We expect under each hidden state vector and each current character, we should have a different transition matrix. The earlier model does not quite catch that

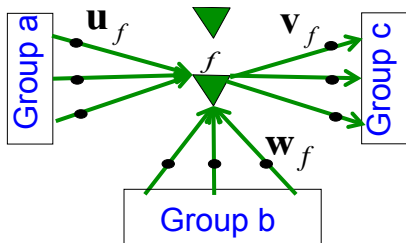
Multiplicative connections

- Instead of using the inputs to the recurrent net to provide additive extra input to the hidden units, we could use the current input character to choose the whole hidden-to-hidden weight matrix
 - But this requires $86 \times 1500 \times 1500$ parameters
 - This could make the net overfit
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
 - We want a different transition matrix for each of the 86 characters, but we want these 86 character-specific weight matrices to share parameters (the characters 9 and 8 should have similar matrices)

Multiplicative connections

- Instead of using the inputs to the recurrent net to provide additive extra input to the hidden units, we could use the current input character to choose the whole hidden-to-hidden weight matrix
 - But this requires $86 \times 1500 \times 1500$ parameters
 - This could make the net overfit
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
 - We want a different transition matrix for each of the 86 characters, but we want these 86 character-specific weight matrices to share parameters (the characters 9 and 8 should have similar matrices)

Using factors to implement multiplicative interactions

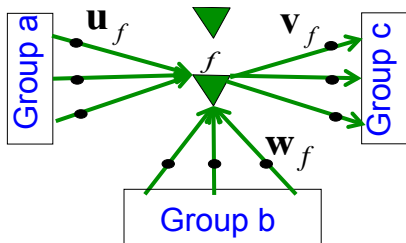


Vector input to group c:

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group } b} \underbrace{(a^T u_f)}_{\text{Scalar input from group } a} v_f$$

- We can get groups *a* and *b* to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement multiplicative interactions

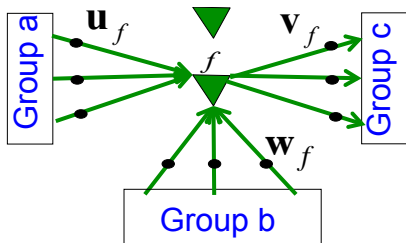


Vector input to group c:

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group } b} \underbrace{(a^T u_f)}_{\text{Scalar input from group } a} v_f$$

- We can get groups a and b to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement multiplicative interactions

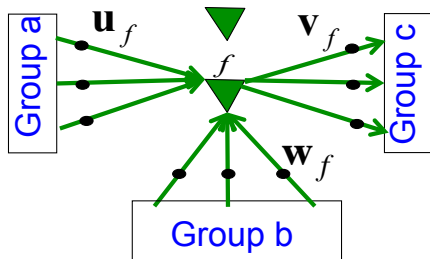


Vector input to group c :

$$c_f = \underbrace{(b^T w_f)}_{\text{Scalar input from group } b} \underbrace{(a^T u_f)}_{\text{Scalar input from group } a} v_f$$

- We can get groups a and b to interact multiplicatively by using “factors”
 - Each factor first computes a weighted sum for each of its input groups
 - Then it sends the product of the weighted sums to its output group

Using factors to implement a set of basis matrices

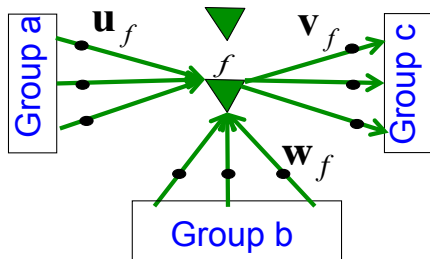


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$c = \left(\sum_f (b^T w_f)(v_f u_f^T) \right) a$$

Using factors to implement a set of basis matrices

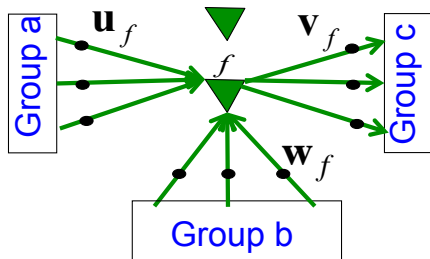


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$c = \left(\sum_f (b^T w_f)(v_f u_f^T) \right) a$$

Using factors to implement a set of basis matrices

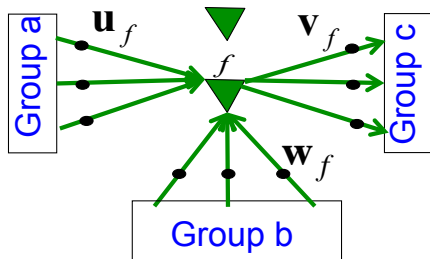


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$c = \left(\sum_f (b^T w_f)(v_f u_f^T) \right) a$$

Using factors to implement a set of basis matrices

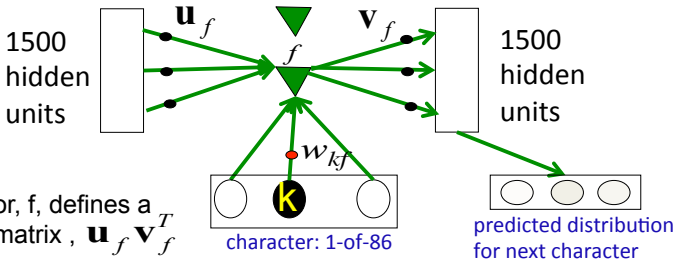


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- We can think about factors another way:
 - Each factor defines a rank 1 transition matrix from a to c

$$c = \left(\sum_f (b^T w_f)(v_f u_f^T) \right) a$$

Using 3-way factors to allow a character to create a whole transition matrix



Each factor, f , defines a rank one matrix, $\mathbf{u}_f \mathbf{v}_f^T$

Each character, k , determines a gain w_{kf} for each of these matrices.

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then, assuming that approximation is correct, they minimize the error using an efficient technique called conjugate gradient. Then they make another approximation to the curvature matrix and minimize again
 - For RNNs, it is important to add a penalty to avoid changing any of the hidden activities too much

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then, assuming that approximation is correct, they minimize the error using an efficient technique called conjugate gradient. Then they make another approximation to the curvature matrix and minimize again
 - For RNNs, it is important to add a penalty to avoid changing any of the hidden activities too much

Some note on optimization

- To optimize efficiently, they use Hessian-free (HF) method to minimize the cost
- HF is a second order method similar to Newton methods and LBFGS that take advantage of the curvature (Hessian) matrix
- In the HF method, they make an approximation to the curvature matrix and then, assuming that approximation is correct, they minimize the error using an efficient technique called conjugate gradient. Then they make another approximation to the curvature matrix and minimize again
 - For RNNs, it is important to add a penalty to avoid changing any of the hidden activities too much

Conjugate gradient

- There is an alternative to going to the minimum in one step by multiplying by the inverse of the curvature matrix
- Use a sequence of steps each of which finds the minimum along one direction
- Make sure that each new direction is “conjugate” to the previous directions so you do not mess up the minimization you already did.
 - “conjugate” means that as you go in the new direction, you do not change the gradients in the previous directions

Experiment setup

- Start the model with its default hidden state.
- Give it a “burn-in” sequence of characters and let it update its hidden state after each character.
- Then look at the probability distribution it predicts for the next character.
- Pick a character randomly from that distribution and tell the net that this was the character that actually occurred.
 - i.e. tell it that its guess was correct, whatever it guessed.
- Continue to let it pick characters until bored.
- Look at the character strings it produces to see what it “knows”.

Result

He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters' sisters in lower coil trains were always operated on the line of the **ephemerable** street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade."

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges **s** (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges **s** (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungeli **ni del Rey** (first try)
- The meaning of life is **literary recognition.** (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: some completions produced by the model

- Sheila thrunges (most frequent)
- People thrunge (most frequent next character is space)
- Shiela, Thrungelini del Rey (first try)
- The meaning of life is literary recognition. (6 th try)
- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. (one of the first 10 tries for a model trained for longer)

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

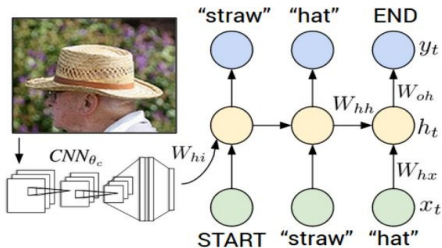
Result: what does it know?

- It knows a huge number of words and a lot about proper names, dates, and numbers
- It is good at balancing quotes and brackets
 - It can count brackets: none, one, many
- It knows a lot about syntax but its very hard to pin down exactly what grammar it actually “knows”
- It knows a lot of weak semantic associations
 - E.g. it knows Plato is associated with Wittgenstein and cabbage is associated with vegetable

RNNs for predicting the next word

- Tomas Mikolov and his collaborators have recently trained quite large RNNs on quite large training sets using backprop through time (BPTT)
 - They do better than feed-forward neural nets
 - They do better than the best other models
 - They do even better when averaged with other models
- RNNs require much less training data to reach the same level of performance as other models
- RNNs improve faster than other methods as the dataset gets bigger
 - This is going to make them very hard to beat

Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

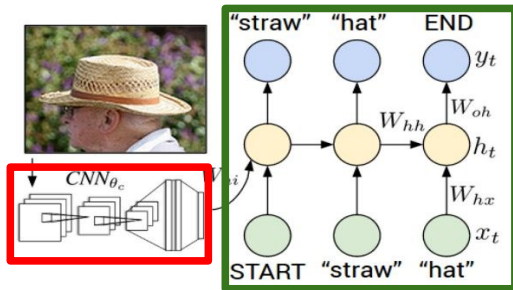
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network

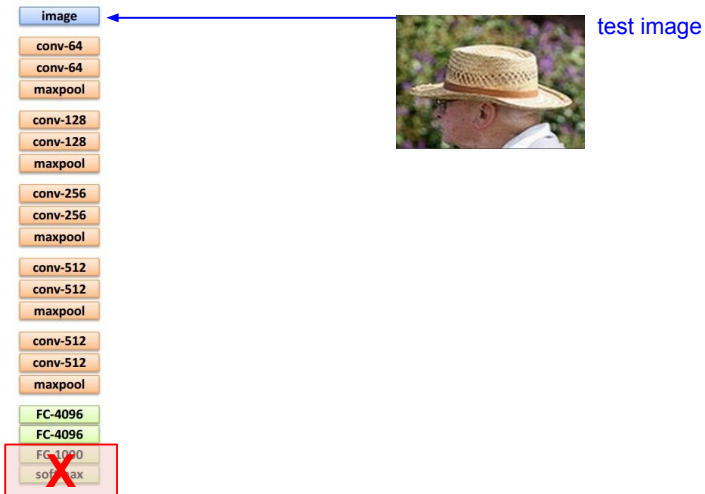


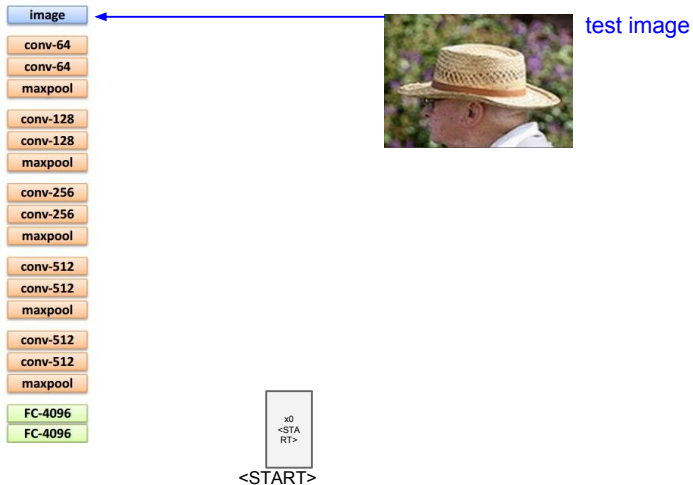
Convolutional Neural Network

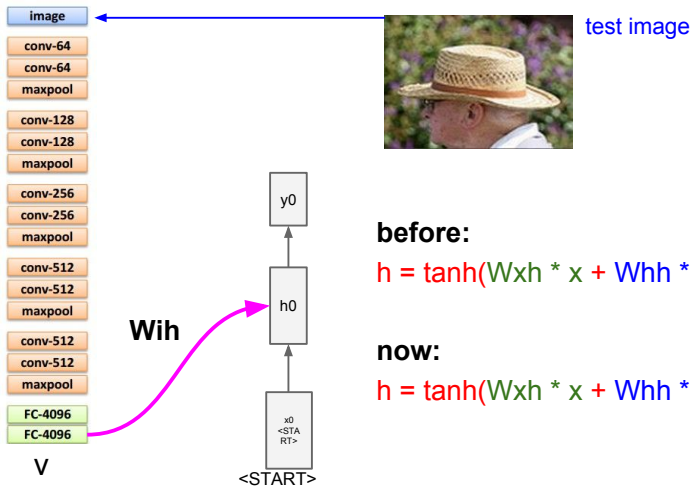


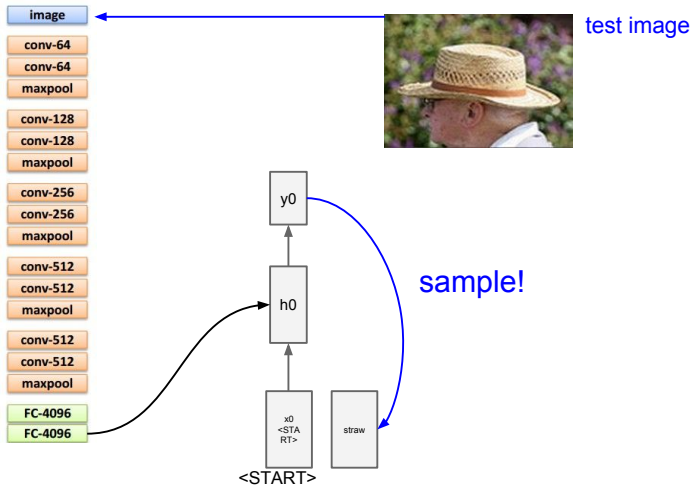
test image

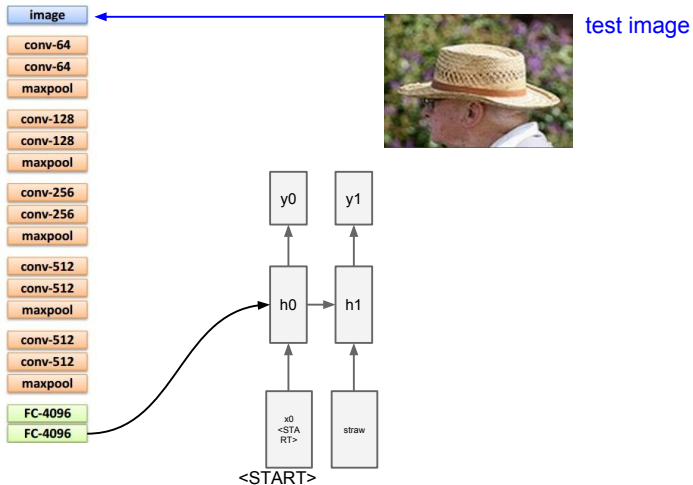


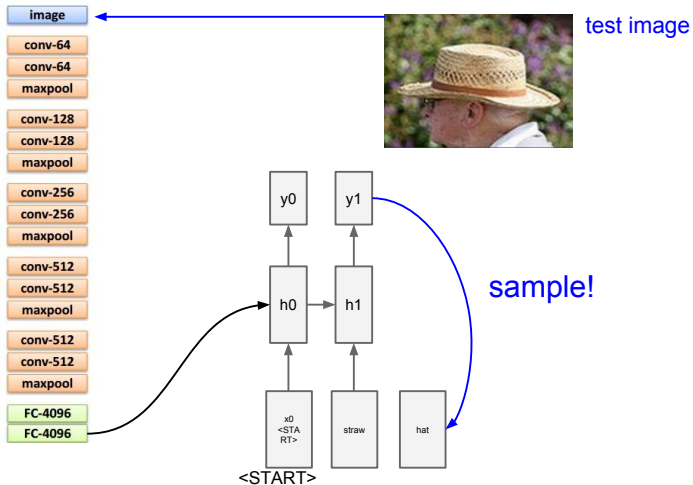


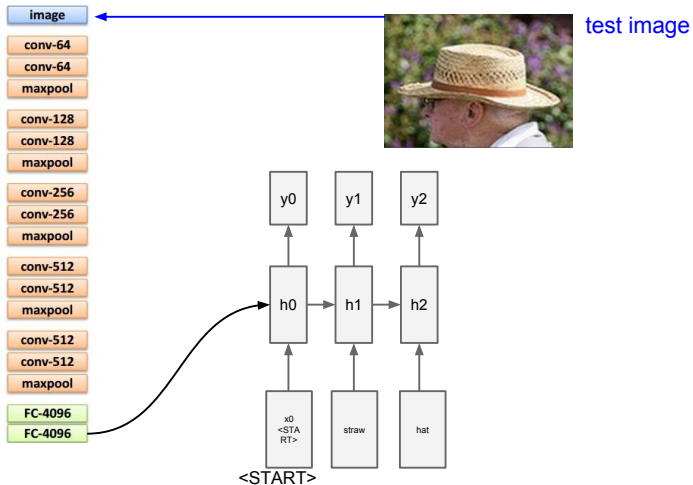












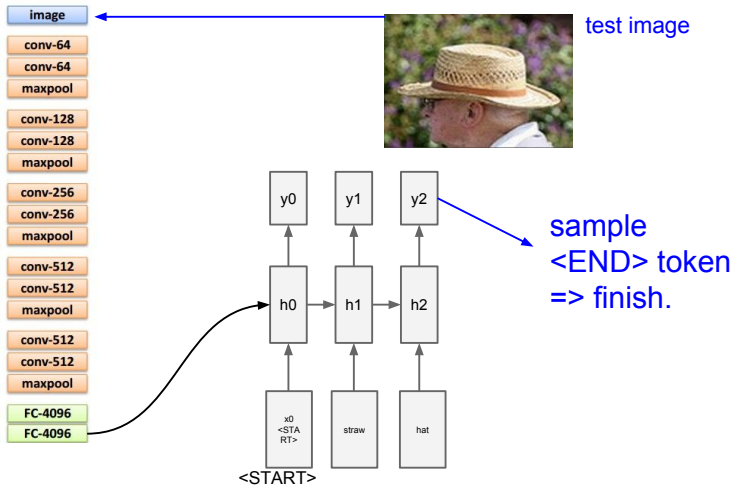


Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.



Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

currently:

~120K images

~5 sentences each



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."

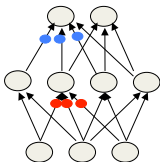


"boy is doing backflip on wakeboard."

More examples

The key idea of echo state networks (perceptrons again?)

- A very simple way to learn a feedforward network is to make the early layers random and fixed.
- Then we just learn the last layer which is a linear model that uses the transformed inputs to predict the target outputs.
 - A big random expansion of the input vector can help.



- The equivalent idea for RNNs is to fix the **input→hidden** connections and the **hidden→hidden** connections at random values and only learn the **hidden→output** connections.
 - The learning is then very simple (assuming linear output units).
 - Its important to set the random connections very carefully so the RNN does not explode or die.

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the length of the activity vector stays about the same after each iteration
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the length of the activity vector stays about the same after each iteration
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the length of the activity vector stays about the same after each iteration
 - This allows the input to echo around the network for a long time
- Use sparse connectivity (i.e. set most of the weights to zero)
 - This creates lots of loosely coupled oscillators
- Choose the scale of the input→hidden connections very carefully
 - They need to drive the loosely coupled oscillators without wiping out the information from the past that they already contain
- The learning is so fast that we can try many different scales for the weights and sparsenesses
 - This is often necessary

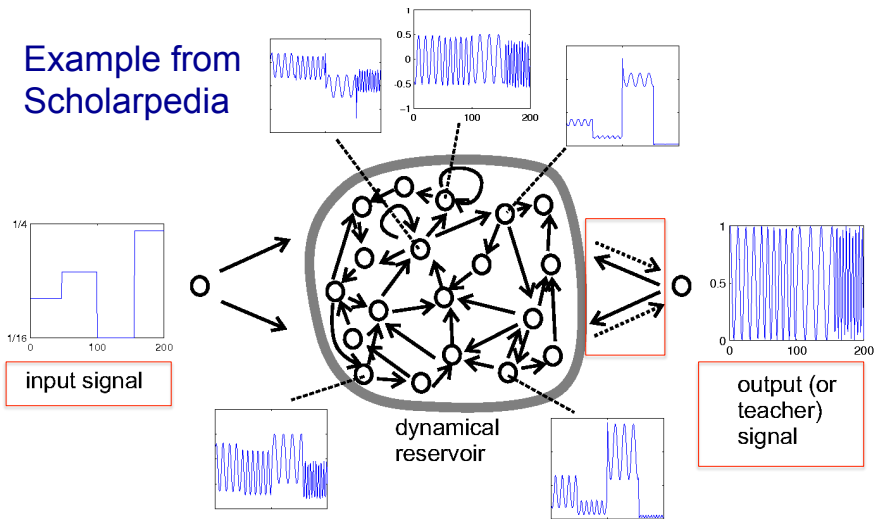
A simple example of an echo state network

INPUT SEQUENCE A real-valued time-varying value that specifies the frequency of a sine wave

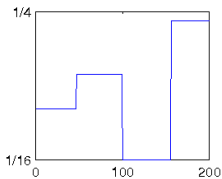
TARGET OUTPUT SEQUENCE A sine wave with the currently specified frequency

LEARNING METHOD Fit a linear model that takes the states of the hidden units as input and produces a single scalar output

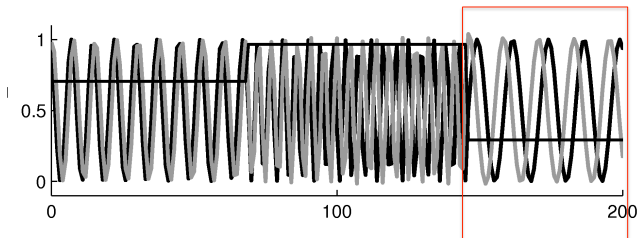
Example from Scholarpedia



The target and predicted outputs after learning



input signal



Beyond echo state networks

- Good aspects of ESNs Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights
- Ilya Sutskever (2012) has shown that if the weights are initialized using the ESN methods, RNNs can be trained very effectively
 - He uses rmsprop with momentum

Beyond echo state networks

- Good aspects of ESNs Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights
- Ilya Sutskever (2012) has shown that if the weights are initialized using the ESN methods, RNNs can be trained very effectively
 - He uses rmsprop with momentum

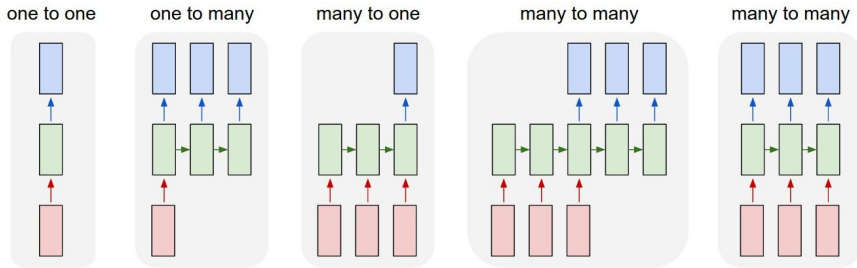
Beyond echo state networks

- Good aspects of ESNs Echo state networks can be trained very fast because they just fit a linear model
- They demonstrate that it is very important to initialize weights sensibly
- They can do impressive modeling of one-dimensional time-series
 - but they cannot compete seriously for high-dimensional data like pre-processed speech
- Bad aspects of ESNs They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights
- Ilya Sutskever (2012) has shown that if the weights are initialized using the ESN methods, RNNs can be trained very effectively
 - He uses rmsprop with momentum

Conclusions

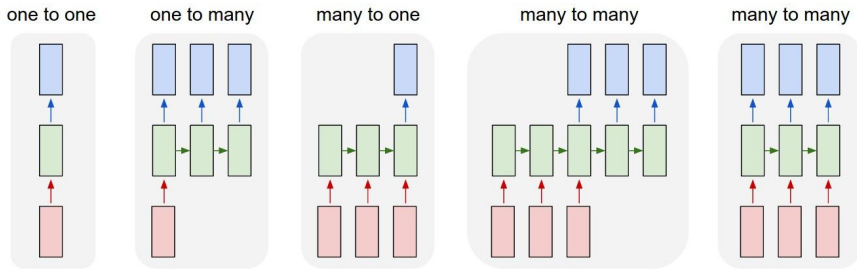
- RNNs allow a lot of flexibility in architecture design and have many applications

Recurrent Networks offer a lot of flexibility:



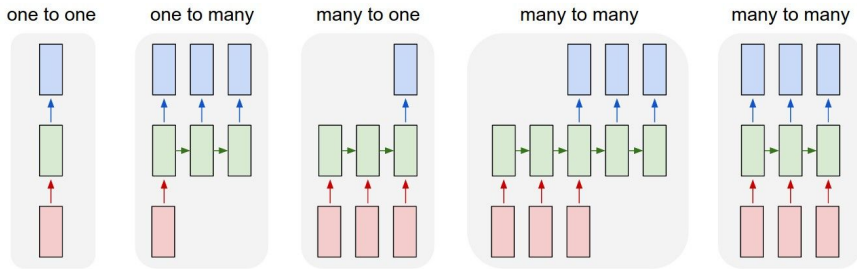
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



e.g. **Image Captioning**
image -> sequence of words

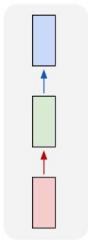
Recurrent Networks offer a lot of flexibility:



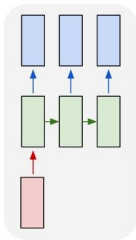
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

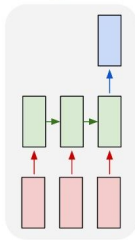
one to one



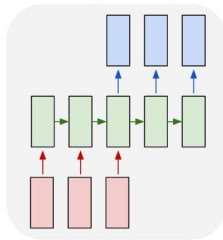
one to many



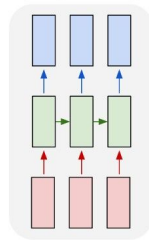
many to one




many to many



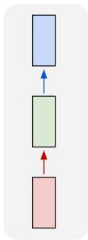
many to many



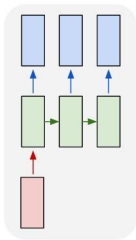

 e.g. **Machine Translation**
 seq of words \rightarrow seq of words

Recurrent Networks offer a lot of flexibility:

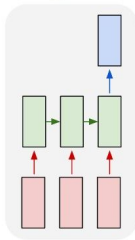
one to one



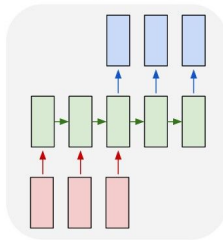
one to many



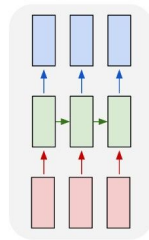
many to one



many to many



many to many



e.g. **Video classification on frame level**

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better optimization techniques such as Hessian-free methods could be used to avoid gating structures like LSTM
- Echo state networks are another possibility but may not work very well for high dimensional inputs