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

Logistics

- 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

Logistics

- 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

Schooner

- 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

Schooner

- 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

Schooner

- 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?

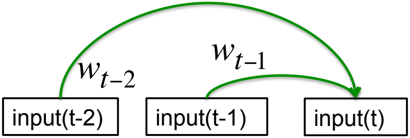
Why non-memoryless models

- 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?

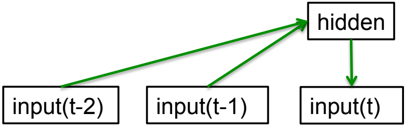
[Hinton 2012, week 7]

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

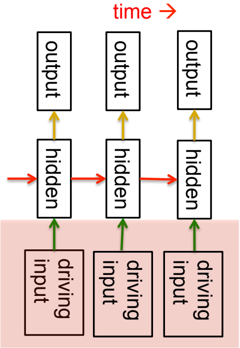
Beyond memoryless models

- 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

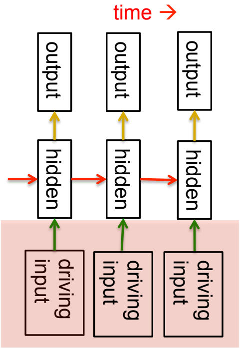
Linear dynamical systems (Engineers love them!)



[Hinton 2012, Week 7]

- 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

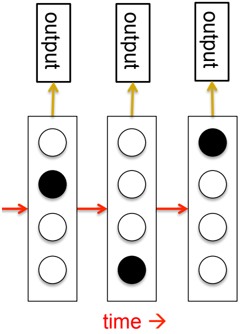
Linear dynamical systems (Engineers love them!)



- 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]

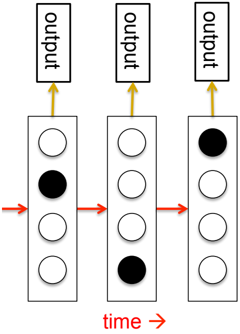
Hidden Markov Models (Computer scientists love them!)



[Hinton 2012, Week 7]

- 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

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]

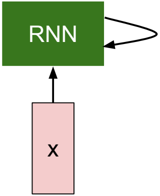
A fundamental limitation of HMMs

- 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]

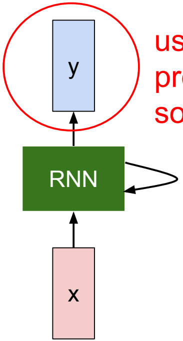
Recurrent neural networks (RNNs)

Recurrent Neural Network



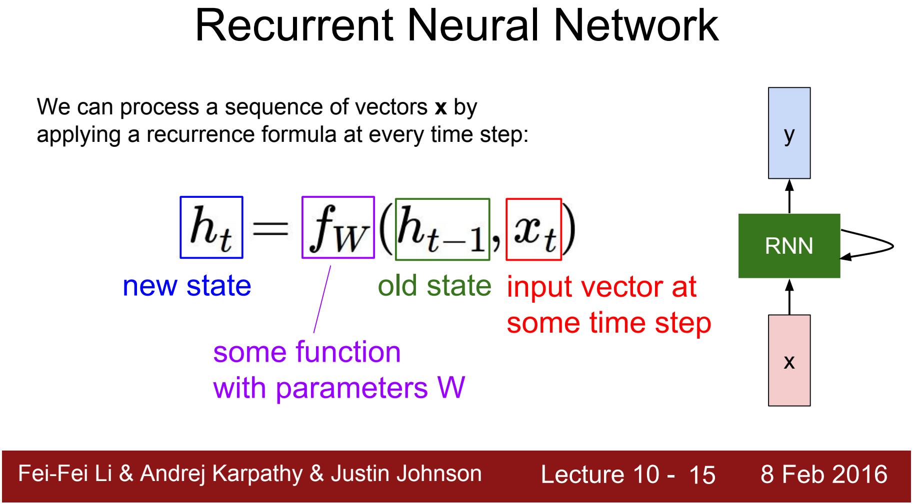
Recurrent neural networks (RNNs)

Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent neural networks (RNNs)



Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

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

new state

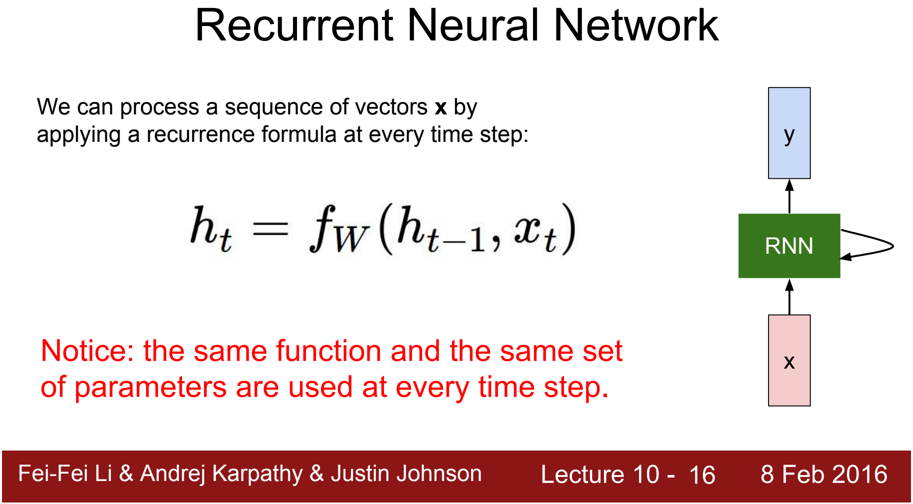
old state

input vector at some time step

some function with parameters W

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 15 8 Feb 2016

Recurrent neural networks (RNNs)



Recurrent Neural Network

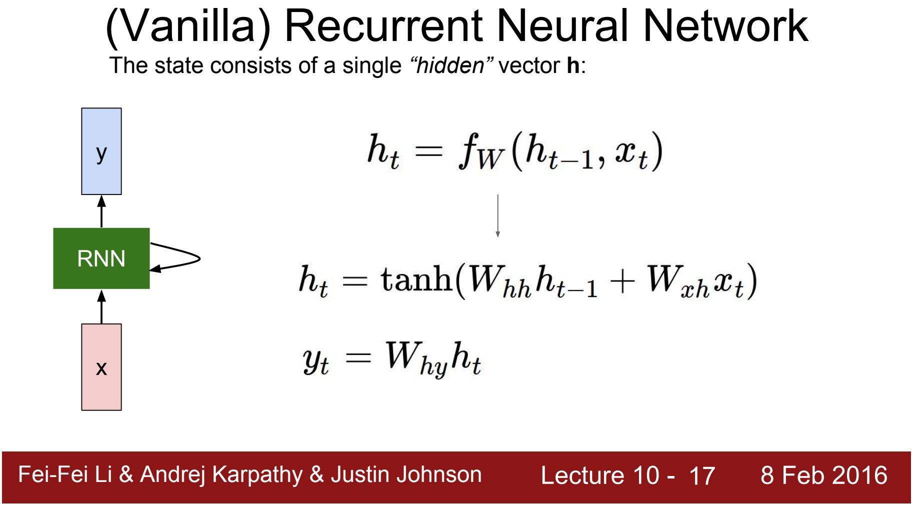
We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

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

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 16 8 Feb 2016

Recurrent neural networks (RNNs)



(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:

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

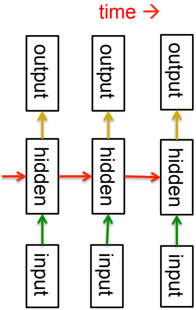
$$\downarrow$$

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

$$y_t = W_{hy}h_t$$

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 17 8 Feb 2016

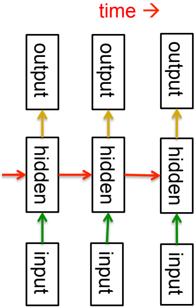
Recurrent neural networks (RNNs)



[Hinton 2012, week 7]

- 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

Recurrent neural networks (RNNs)



[Hinton 2012, week 7]

- 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

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]

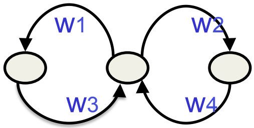
Recurrent neural networks (RNNs)

- 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]

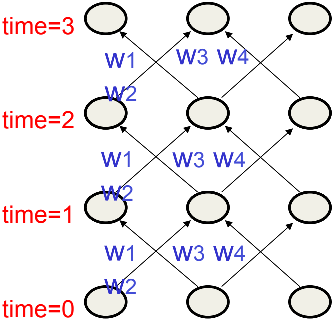
Expanding RNN as feedforward nets

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

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

$$\textit{We need}: \ \Delta w_1^{(1)} = \Delta w_2^{(2)}$$

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

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

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

[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

$$\begin{aligned}
&\textit{To constrain}: \quad w_1^{(1)} = w_1^{(2)}\\
&\textit{We need}: \quad \Delta w_1^{(1)} = \Delta w_2^{(2)}\\
&\textit{Compute}: \quad \frac{\partial E}{\partial w_1^{(1)}} \; \textit{and} \; \frac{\partial E}{\partial w_1^{(2)}}\\
&\textit{Use}: \quad \frac{\partial E}{\partial w_1^{(1)}} + \frac{\partial E}{\partial w_1^{(2)}}\\
&\qquad \textit{for both } w_1^{(1)} \textit{ and } w_1^{(2)}
\end{aligned}$$

[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

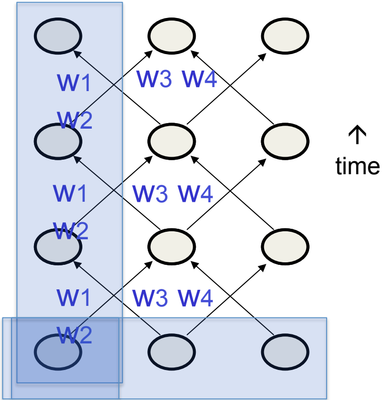
An irritative extra issue

- 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

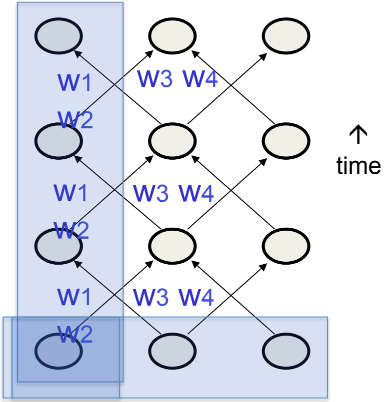
- We need to specify the initial activity state of all the hidden and output units
- We could just fix these initial states to have some default value like 0.5
- But it is better to treat the initial states as learned parameters
- We learn them in the same way as we learn the weights
    - Start off 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



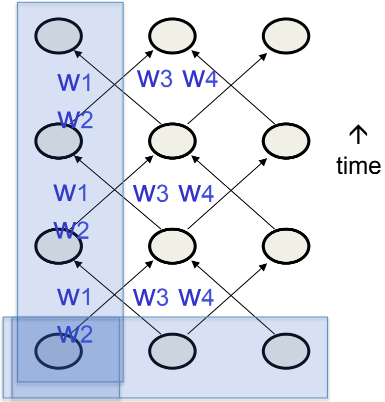
- We can specify inputs in several ways:
  - Specify the initial states of all the units
  - Specify the initial states of a subset of the units
  - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



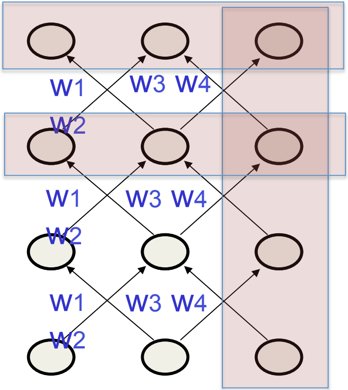
- We can specify inputs in several ways:
  - Specify the initial states of all the units
  - Specify the initial states of a subset of the units
  - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



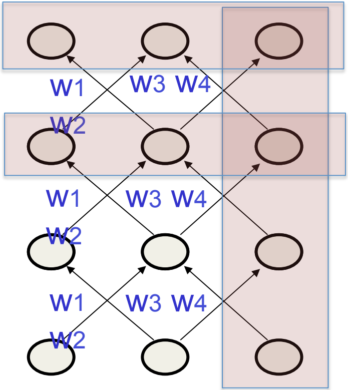
- We can specify inputs in several ways:
  - Specify the initial states of all the units
  - Specify the initial states of a subset of the units
  - Specify the states of the same subset of the units at every time step

Teaching recurrent networks to learn signals



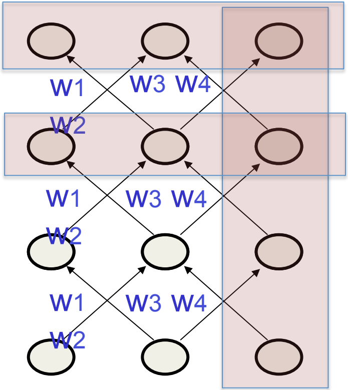
- We can specify targets in several ways:
  - Specify desired final activities of all the units
  - Specify desired activities of all units for the last few steps
    - Good for learning attractors
  - Specify the desired activity of a subset of the units.
    - The other units are input or hidden units.

Teaching recurrent networks to learn signals



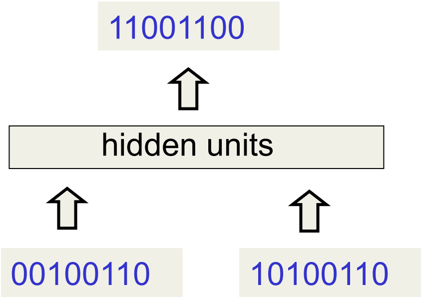
- We can specify targets in several ways:
  - Specify desired final activities of all the units
  - Specify desired activities of all units for the last few steps
    - Good for learning attractors
  - Specify the desired activity of a subset of the units.
    - The other units are input or hidden units.

Teaching recurrent networks to learn signals



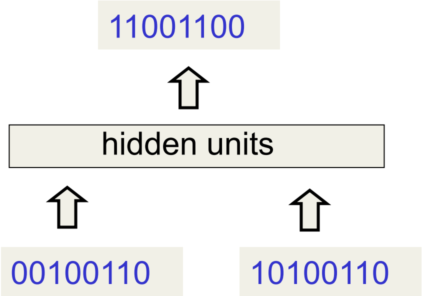
- We can specify targets in several ways:
  - Specify desired final activities of all the units
  - Specify desired activities of all units for the last few steps
    - Good for learning attractors
  - Specify the desired activity of a subset of the units.
    - The other units are input or hidden units.

Toy problem for RNN: binary addition



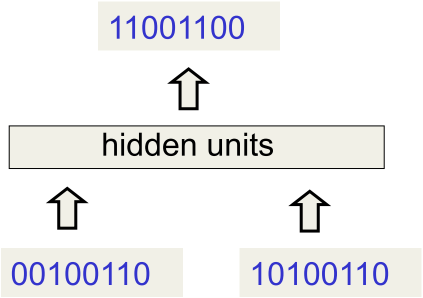
- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently
  - We must decide in advance the maximum number of digits in each number
  - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As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

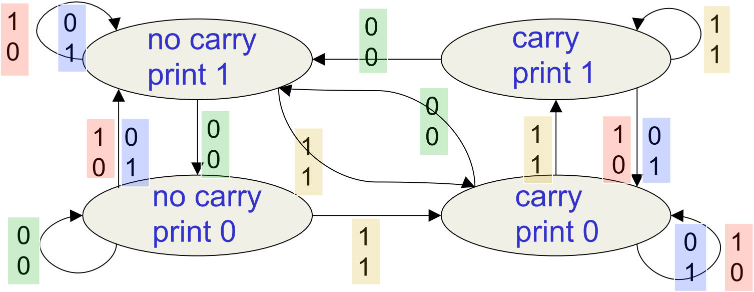
Toy problem for RNN: binary addition



- 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

$$x = [b_8, b_7, \cdots, b_1]$$
$$y = [c_8, c_7, \cdots, c_1]$$
$$z = x + y = [d_8, d_7, \cdots, d_1]$$
$$\hat{z} = [\hat{d}_8, \hat{d}_7, \cdots, \hat{d}_1]$$

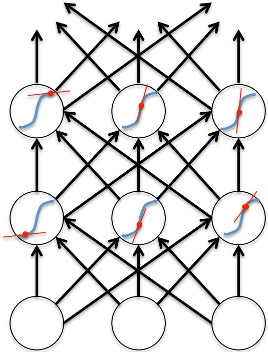
Hidden unit: $h_i = sigm(W_{x,h}[b_i, c_i]^T + W_{h,h}h_{i-1})$

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

`https://github.com/llSourcell/recurrent_neural_net_demo`

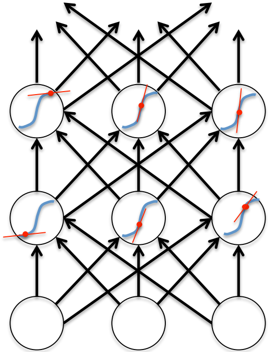
Demo

Why training RNN is difficulty? The backward pass is linear



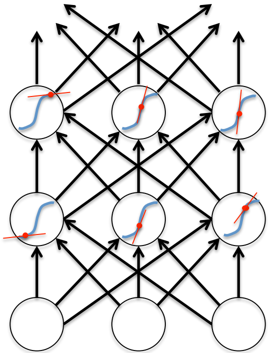
- There is a big difference between the forward and backward passes
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives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

Why training RNN is difficulty? 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 difficulty? 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
$$\begin{aligned}\frac{\partial h_t}{\partial W_{xh}^{(1)}} &= \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \cdots \frac{\partial h_1}{\partial W_{xh}^{(1)}} \\ &= \left(\prod_{\tau=2}^{t} \dot{\tanh}^{(\tau)} W_{hh}^{(t)} \right) \frac{\partial h_1}{\partial W_{xh}^{(1)}},\end{aligned}$$
where $\dot{\tanh}^{(\tau)} = \tanh'(W_{hh}^{(\tau)} h_{\tau-1} + W_{xh}^{(\tau)} x_\tau)$.
- $\prod_{\tau=2}^{t} \dot{\tanh}^{(\tau)} W_{hh}^{(t)}$ 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

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

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

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

Passing gradient to many steps back

- Recall

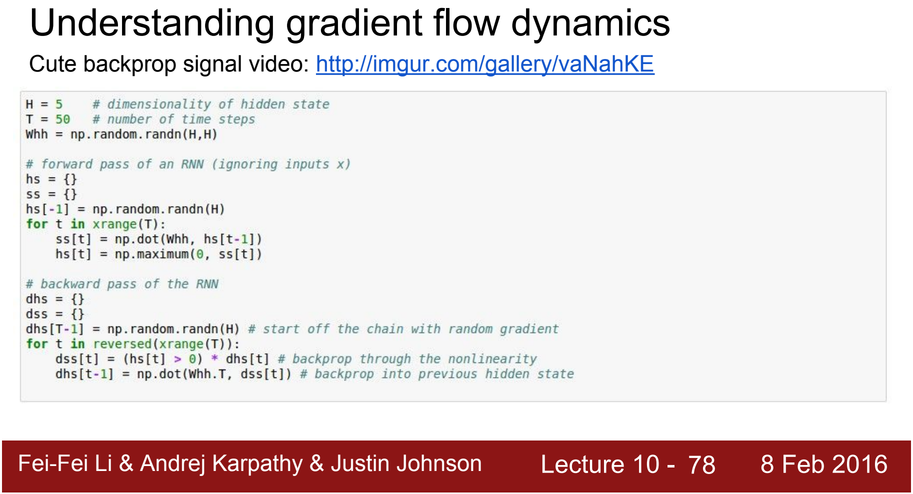
$$h_t = \tanh(W_{hh}^{(t)} h_{t-1} + W_{xh}^{(t)} x_t)$$

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

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

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

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

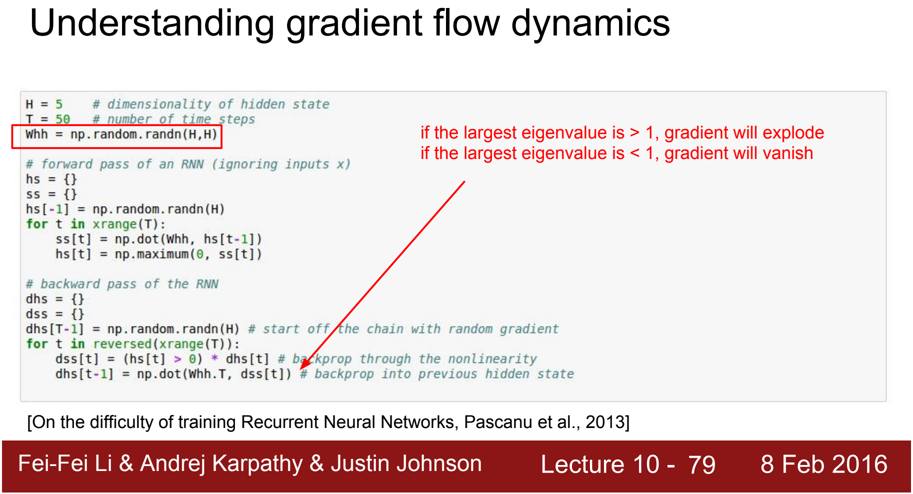


Understanding gradient flow dynamics

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
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 10 - 78   8 Feb 2016



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]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 79 8 Feb 2016

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

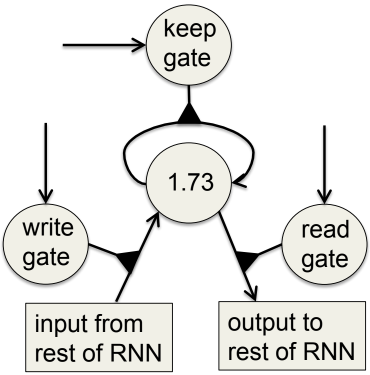
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

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

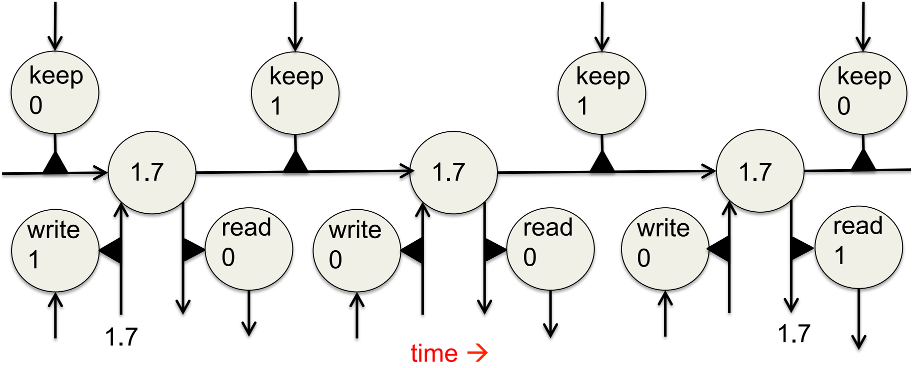
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell so long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

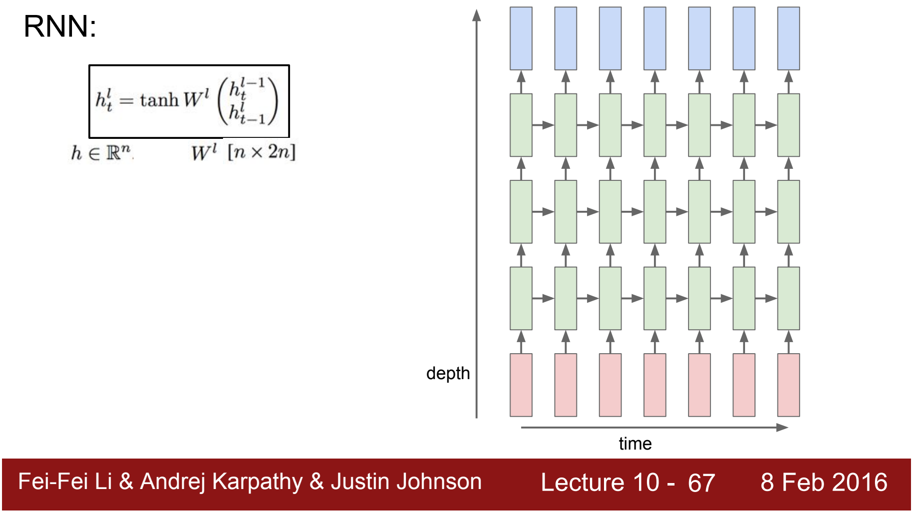
Implementing a memory cell in a neural network



- 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



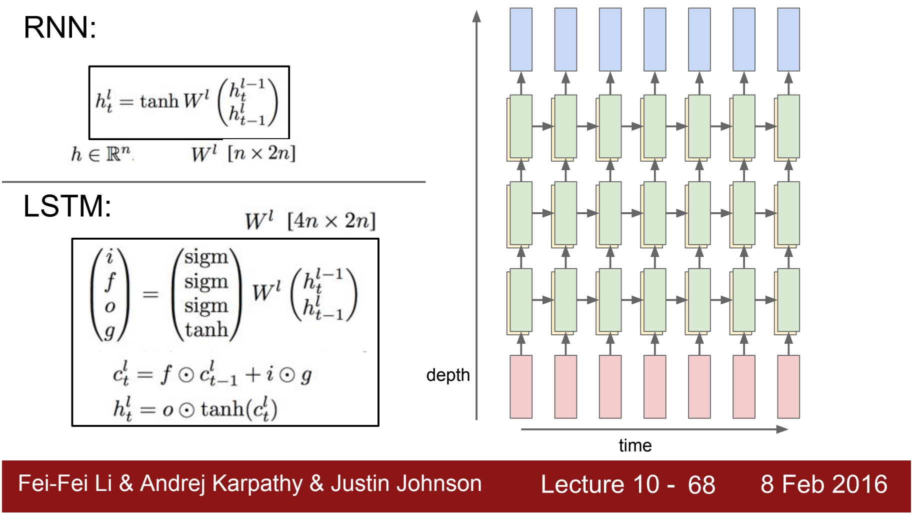


RNN:

$$\boxed{h^l_t = \tanh W^l \begin{pmatrix} h^{l-1}_t \\ h^l_{t-1} \end{pmatrix}}$$

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 67 8 Feb 2016



RNN:

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

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

LSTM:

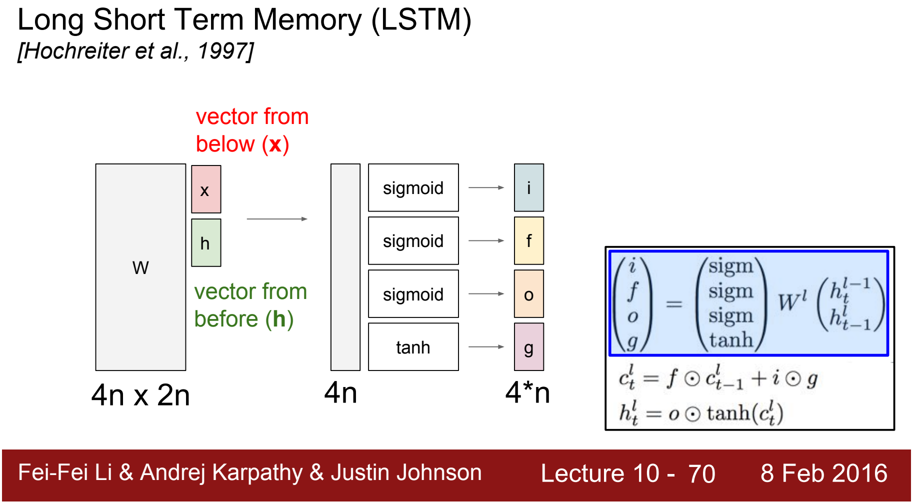
$W^l \ [4n \times 2n]$

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

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

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 68 8 Feb 2016



Long Short Term Memory (LSTM)

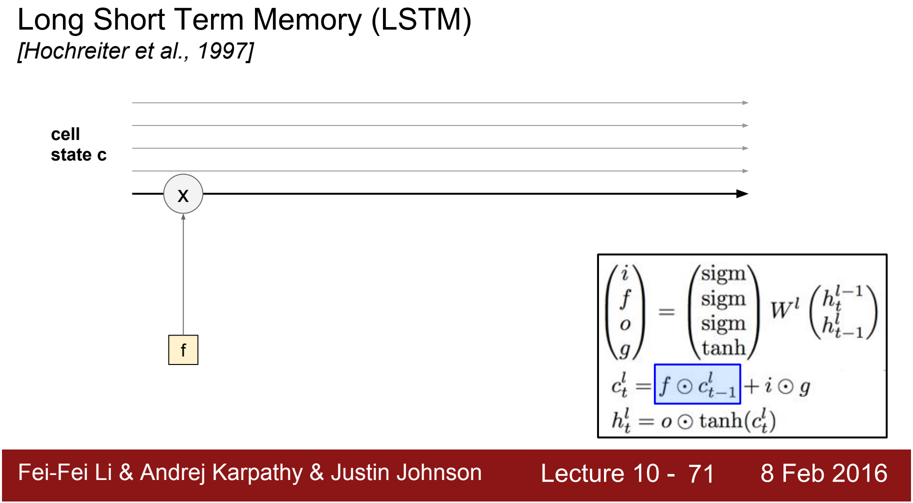
[Hochreiter et al., 1997]

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

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

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 70 8 Feb 2016



Long Short Term Memory (LSTM)

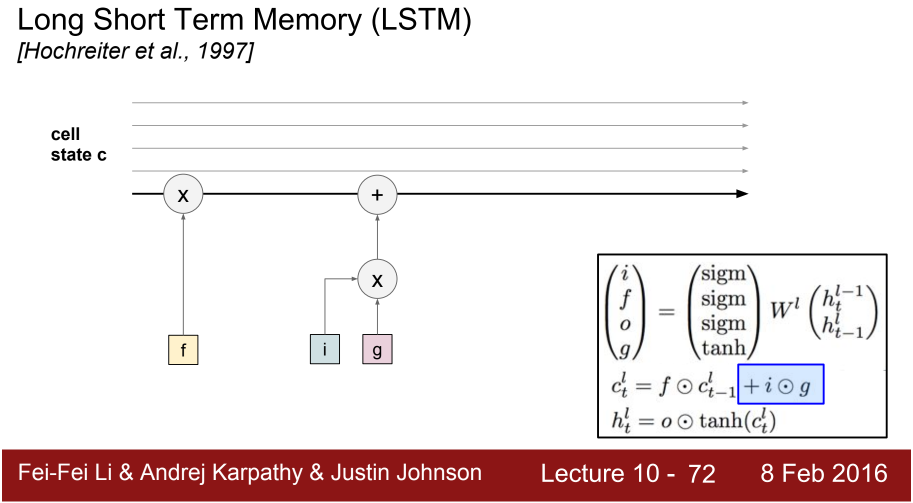
[Hochreiter et al., 1997]

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

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

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 71 8 Feb 2016



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

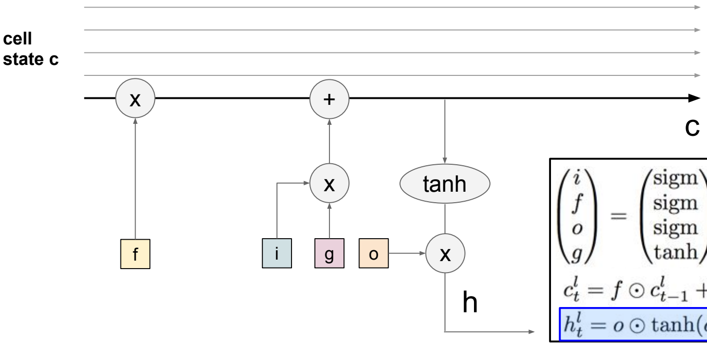
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

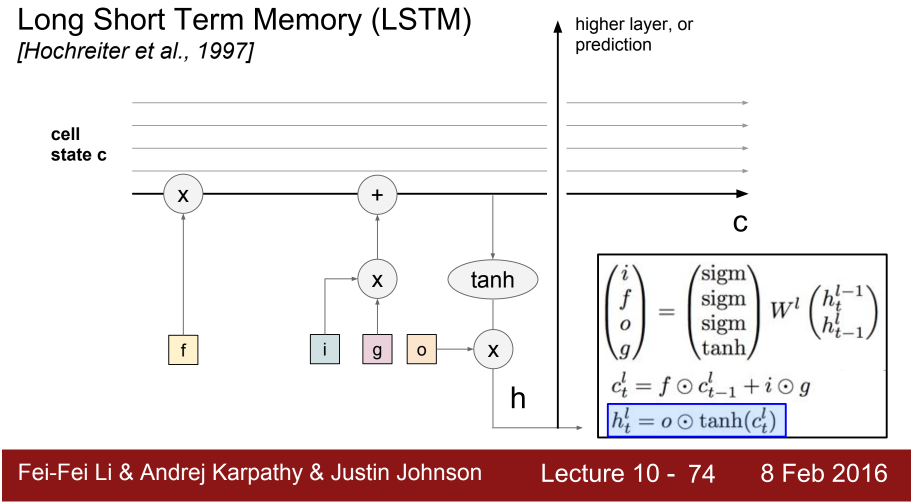


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 73 8 Feb 2016



Long Short Term Memory (LSTM)

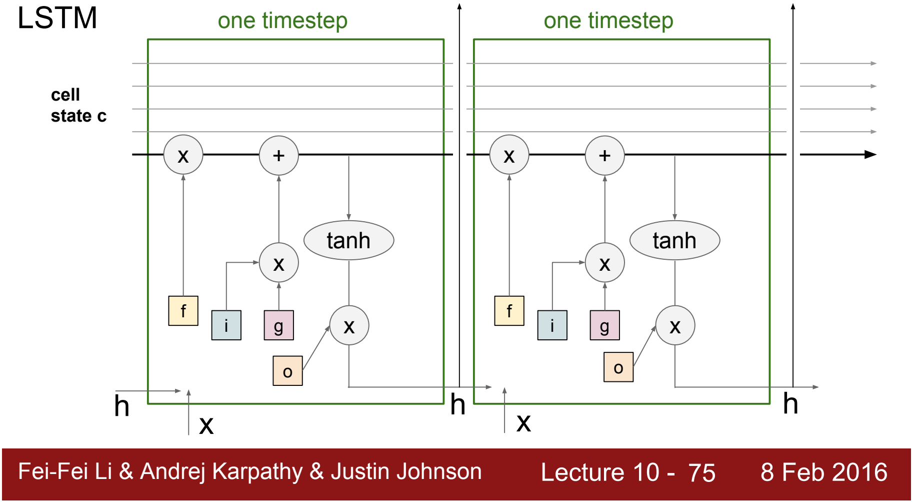
[Hochreiter et al., 1997]

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

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

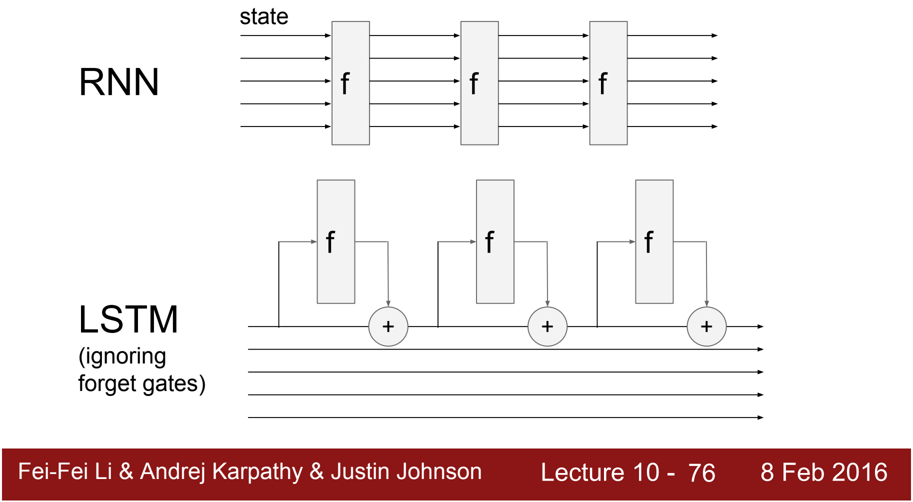
$$\boxed{h_t^l = o \odot \tanh(c_t^l)}$$

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 74 8 Feb 2016



LSTM

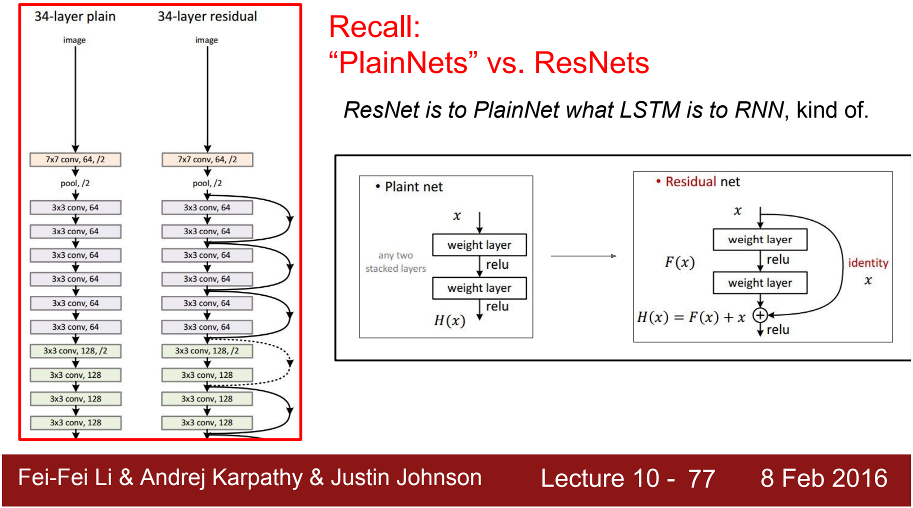
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 75 8 Feb 2016



RNN

LSTM
(ignoring
forget gates)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 76 8 Feb 2016

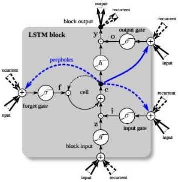


Recall:
"PlainNets" vs. ResNets

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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 77 8 Feb 2016

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]

$$\begin{aligned} r_t &= \mathrm{sigm}\left(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_{t-1} + b_{\mathrm{r}}\right) \\ z_t &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_{t-1} + b_{\mathrm{z}}) \\ \tilde{h}_t &= \tanh(W_{\mathrm{xh}}x_t + W_{\mathrm{hh}}(r_t \odot h_{t-1}) + b_{\mathrm{h}}) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned}$$

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

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}\tanh(h_t) + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

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

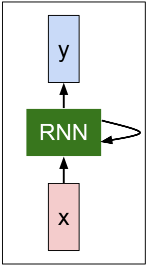
Modelling text: Advantages of working with characters

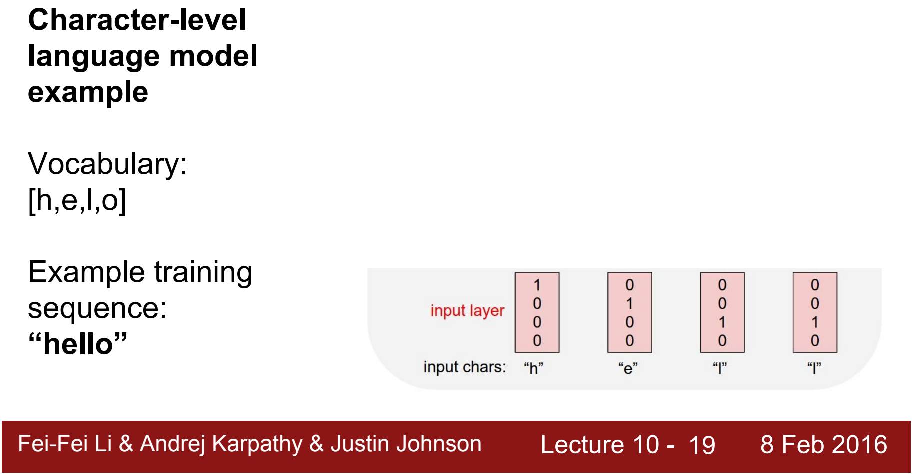
- The web is composed of character strings
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see)
- Pre-processing text to get words is a big hassle
    - What about morphemes (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”





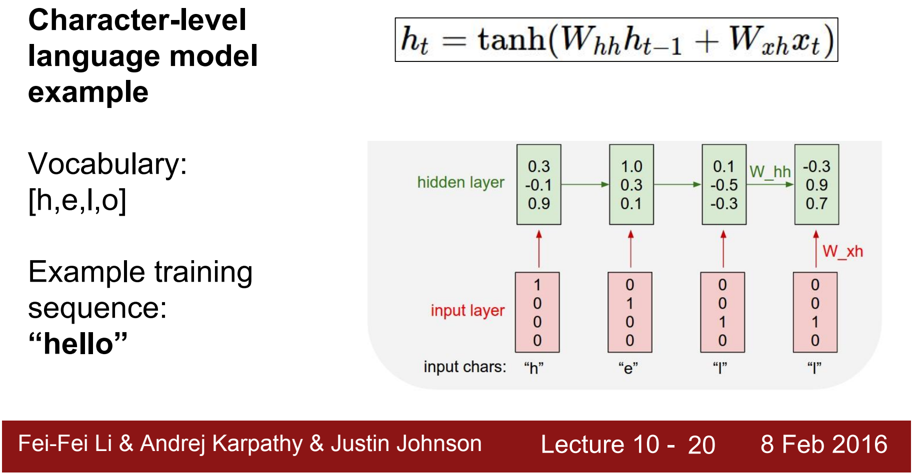
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 19 8 Feb 2016

Example: simple character-level language model



Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

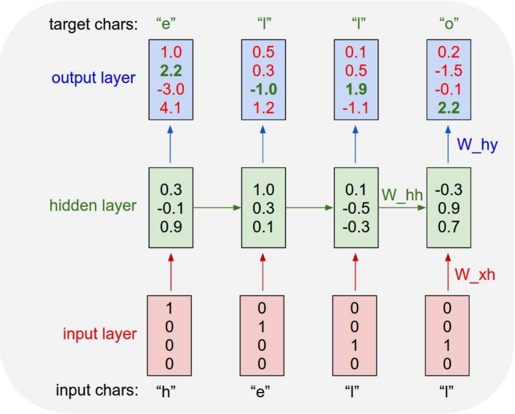
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 20 8 Feb 2016

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 21 8 Feb 2016

Example: simple character-level language model

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

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

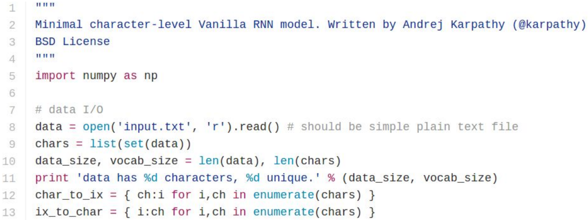
```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

(https://gist.github.com/karpathy/d4dee566867f8291f086)

Example: simple character-level language model

min-char-rnn.py gist

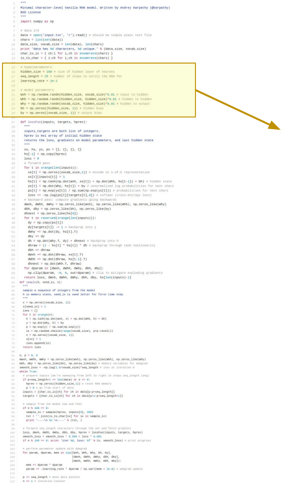
Data I/O



```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

Example: simple character-level language model

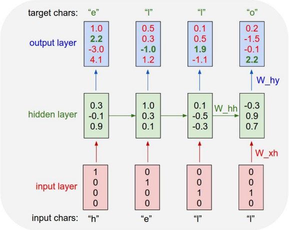
min-char-rnn.py gist



Initializations

```
# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias
```

recall:



Example: simple character-level language model

min-char-rnn.py gist

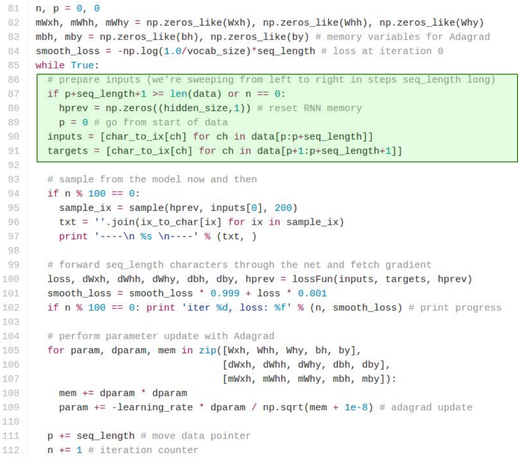
Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Example: simple character-level language model

min-char-rnn.py gist

Main loop



```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Example: simple character-level language model

min-char-rnn.py gist

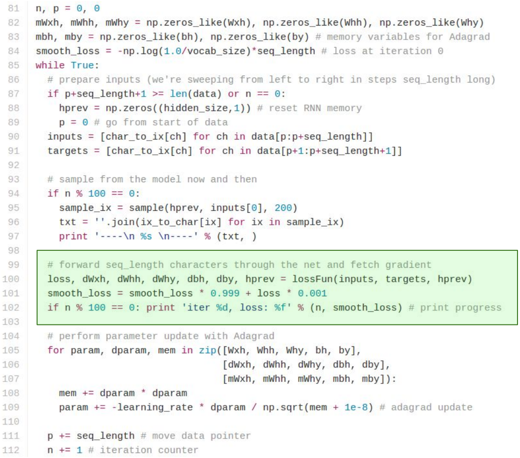
Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Example: simple character-level language model

min-char-rnn.py gist

Main loop



```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Example: simple character-level language model

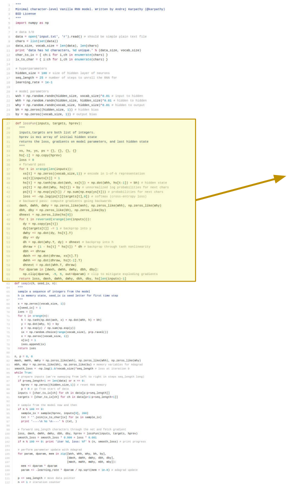
min-char-rnn.py gist

Main loop

```
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

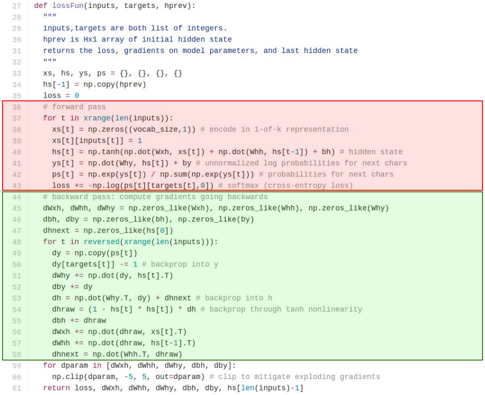
Example: simple character-level language model

min-char-rnn.py gist



Loss function

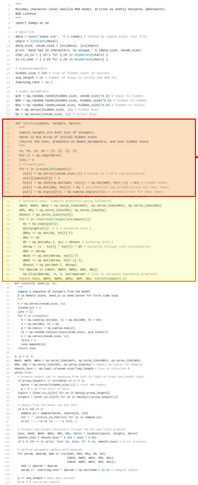
- forward pass (compute loss)
- backward pass (compute param gradient)



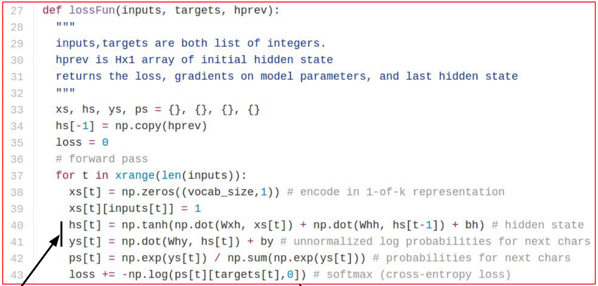
```
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

Example: simple character-level language model

min-char-rnn.py gist



```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```



```
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
```

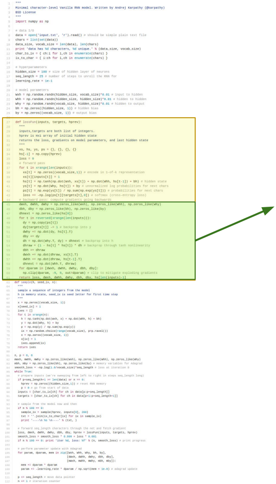
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

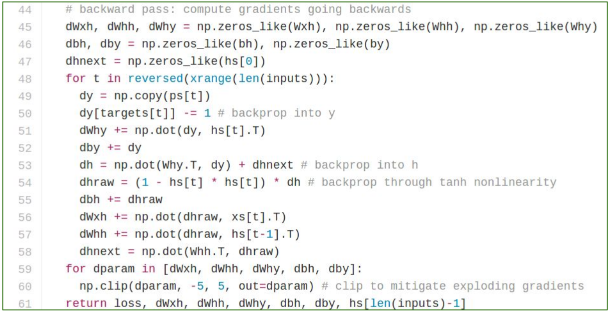
$$y_t = W_{hy} h_t$$

Softmax classifier

Example: simple character-level language model

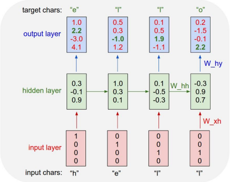
min-char-rnn.py gist



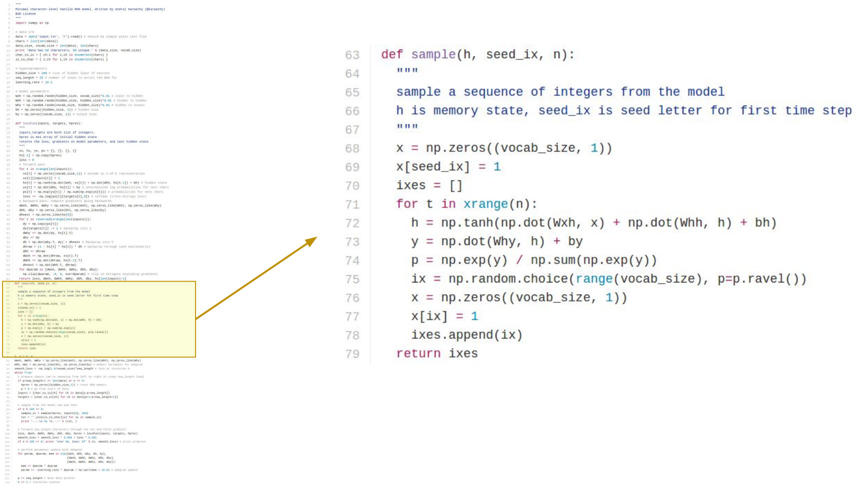


```
# backward pass: compute gradients going backwards
dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
dbh, dby = np.zeros_like(bh), np.zeros_like(by)
dhnext = np.zeros_like(hs[0])
for t in reversed(xrange(len(inputs))):
  dy = np.copy(ps[t])
  dy[targets[t]] -= 1 # backprop into y
  dWhy += np.dot(dy, hs[t].T)
  dby += dy
  dh = np.dot(Why.T, dy) + dhnext # backprop into h
  dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
  dbh += dhraw
  dWxh += np.dot(dhraw, xs[t].T)
  dWhh += np.dot(dhraw, hs[t-1].T)
  dhnext = np.dot(Whh.T, dhraw)
for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
  np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

recall:



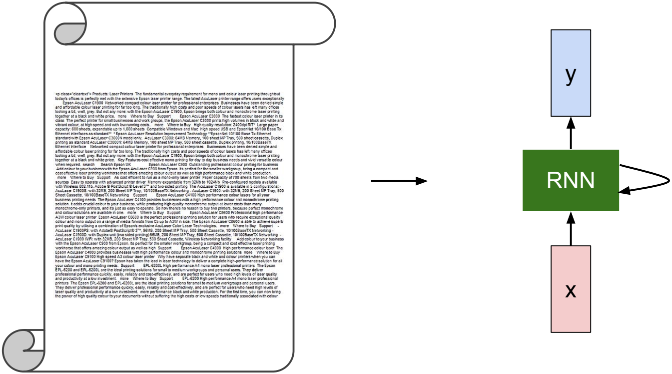
min-char-rnn.py gist



```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

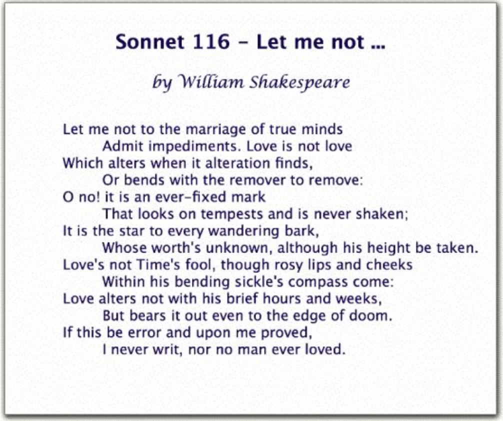
```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes
```

Example: simple character-level language model



<p class="cleartext"> Products: Laser Printers The fundamental everyday requirement for mono and colour laser printing throughtout today's offices is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900 Networked compact colour laser printer for professional enterprises Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C3000 The fastest colour laser printer in its class The perfect printer for small businesses and work groups, the Epson AcuLaser C3000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs.. more Where to Buy High quality resolution: 2400dpi RIT* Large paper capacity: 600 sheets, expandable up to 1,600 sheets Compatible Windows and Mac High speed USB and EpsonNet 10/100 Base Tx Ethernet interfaces as standard** * Epson AcuLaser Resolution Improvement Technology **EpsonNet 10/100 Base Tx Ethernet standard with Epson AcuLaser C3000N model only. AcuLaser C3000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard AcuLaser C3000N: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100BaseTX Ethernet Interface Networked compact colour laser printer for professional enterprises Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. Key Features cost effective mono printing for day to day business needs and vivid versatile colour when required. search Search Epson UK Epson AcuLaser C900 Outstanding professional colour printing for business Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only laser printer Paper capacity of 700 sheets from two media sources Easy to operate with advanced printer driver Memory expandable from 32Mb to 1024Mb Pre-configured models available with Wireless 802.11b, Adobe ® PostScript ® Level 3™ and two-sided printing The AcuLaser C1900 is available in 5 configurations: - AcuLaser C1900S: with 32MB, 200 Sheet MP Tray, 10/100BaseTX Networking - AcuLaser C1900: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking Support Epson AcuLaser C4100 High performance colour lasers for all your business printing needs The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and it's just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C8600 Professional high performance A3W colour laser printer Epson AcuLaser C8600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3W in size. The Epson AcuLaser C8600 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Color Laser Technologies. more Where to Buy Support - AcuLaser C1900PS: with Adobe® PostScript® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900D: with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1900 WiFi: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility Add colour to your business with the Epson AcuLaser C900 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high Support Epson AcuLaser C4000 High performance colour laser The Epson AcuLaser C4000 provides businesses with high performance colour and monochrome printing solutions more Where to Buy Epson AcuLaser C9100 High speed A3 colour laser printer Why have separate black and white and colour printers when you can have the Epson AcuLaser C9100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L High performance A4 mono laser professional printers The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200 High performance A4 mono laser professional printers The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour

Demo

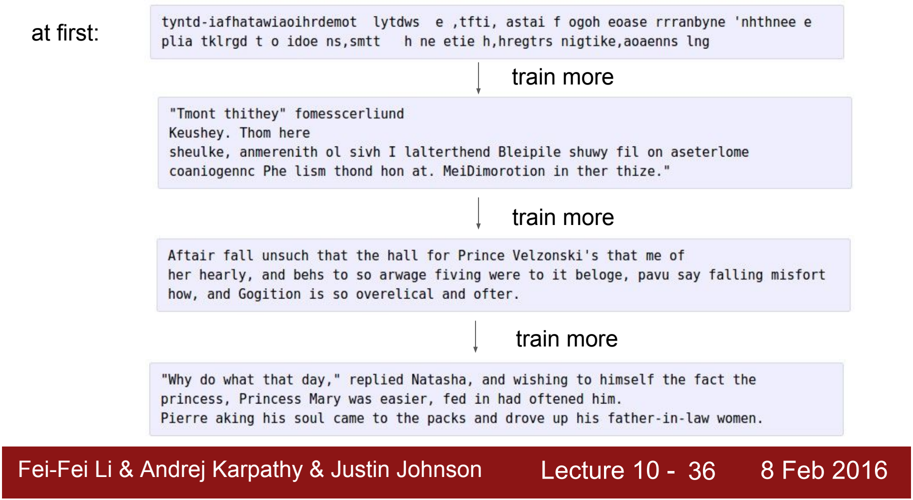


Sonnet 116 – Let me not ...

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.

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 35 8 Feb 2016



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,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. MeiDimorotion in ther thize."
```

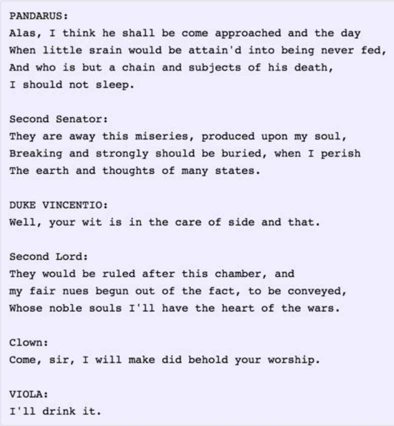
train more

```
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 ofter.
```

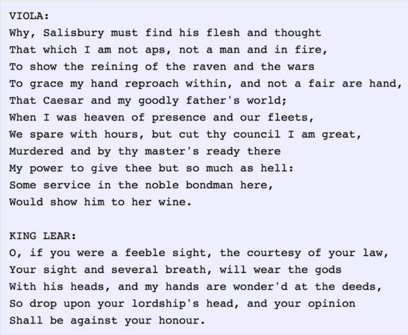
train more

```
"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.
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 36 8 Feb 2016

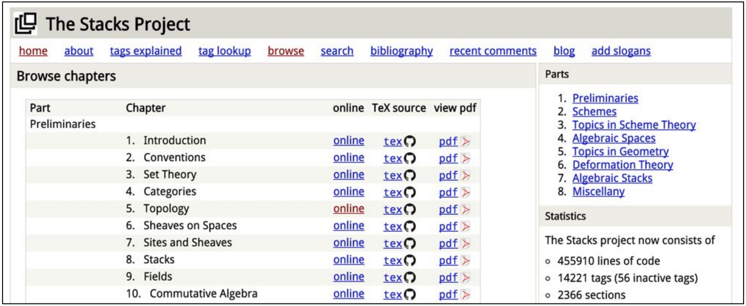


```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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 nues 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 aps, 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

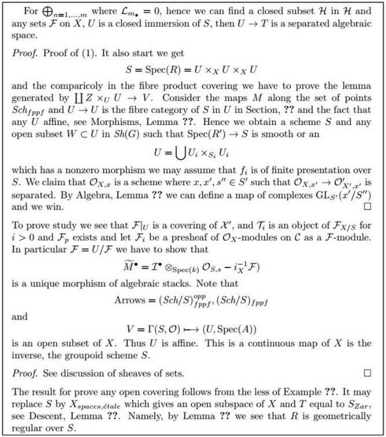
1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

The Stacks project now consists of

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

Latex source



For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, 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 $U \to T$ is a separated algebraic space.

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

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

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ 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 $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ 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 $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. ☐

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 $\mathcal{F}_p$ 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

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

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

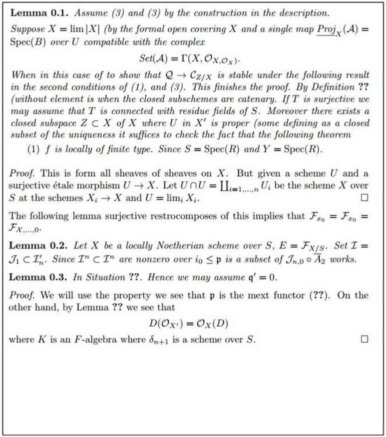
and

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

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. ☐

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 (3) by the construction in the description.*

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

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

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *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* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

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

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

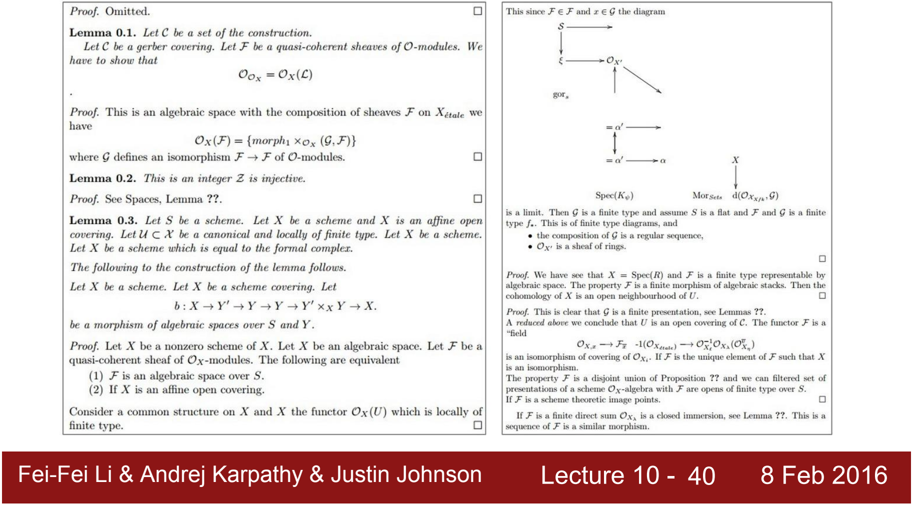
Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$*. Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$*. Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

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 mext 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. ☐



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 $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

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

Lemma 0.2. *This is an integer $\mathcal{Z}$ 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 $\mathcal{U} \subset \mathcal{X}$ 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

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

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. (1) $\mathcal{F}$ is an algebraic space over S.
2. (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

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & {} & & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[rd] & & & \\
\mathrm{gor}_s & {} \arrow[u] & {} & & \\
& =\alpha' \arrow[r] \arrow[d, leftrightarrow] & {} & & \\
& =\alpha' \arrow[r] & \alpha & & X \arrow[d] \\
& \mathrm{Spec}(K_\psi) & & \mathrm{Mor}_{Sets} & \mathrm{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})
\end{tikzcd}$$

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 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.

☐

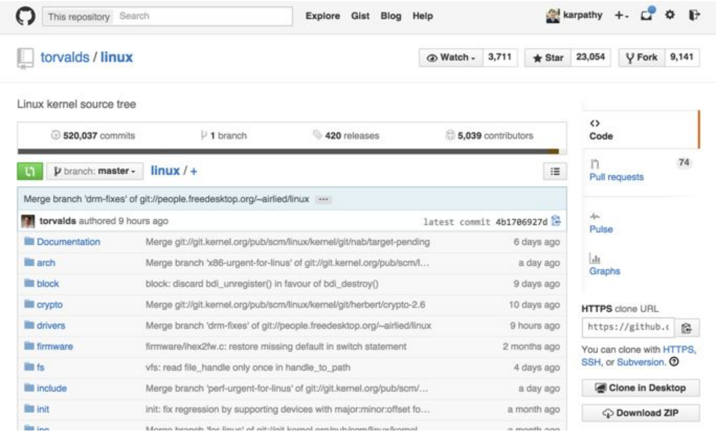
Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. ☐

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

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \ \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. 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 $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.



Linux kernel source tree

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

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux

torvalds authored 9 hours ago — latest commit 4b1706927d

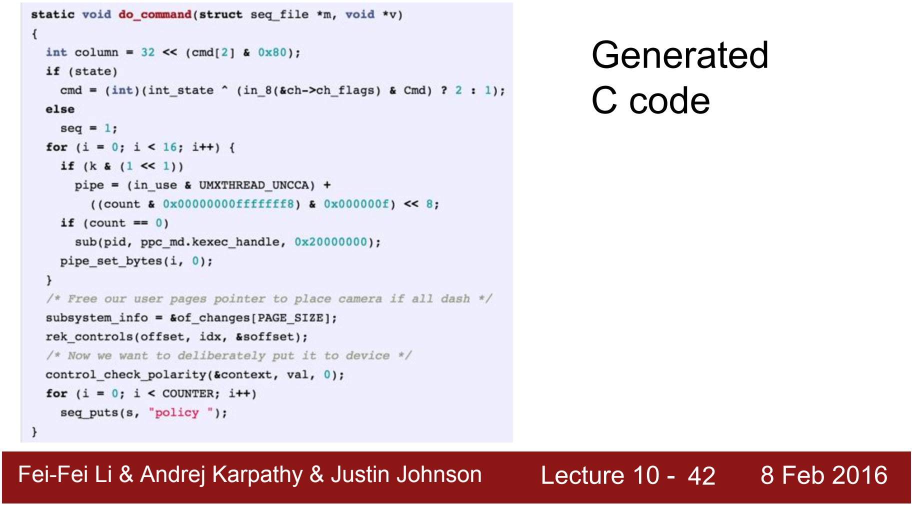
Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l…	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/ihex2fw.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/…	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo…	a month ago

HTTPS clone URL

`https://github.c`

You can clone with HTTPS, SSH, or Subversion.

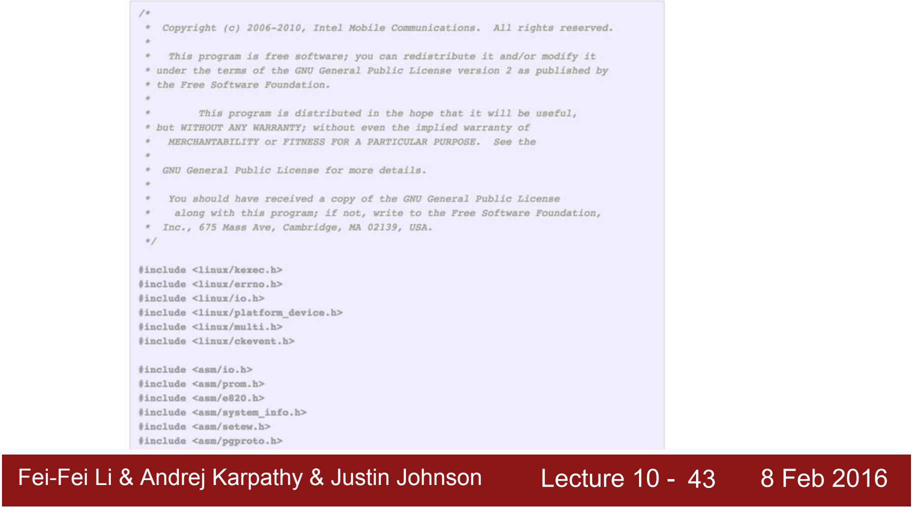
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 41 8 Feb 2016



```
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 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 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 ");
}
```

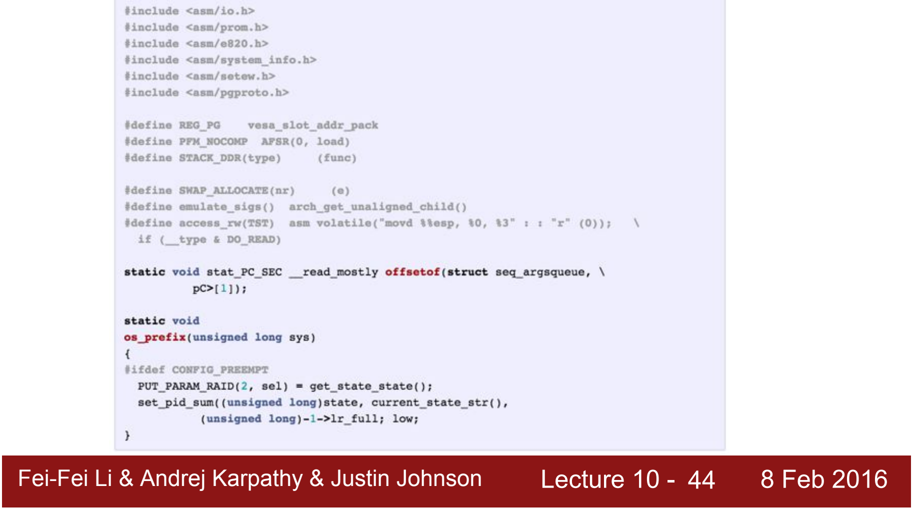
Generated
C code

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 42 8 Feb 2016



```
/*
 *  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/ckevent.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>
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 43 8 Feb 2016



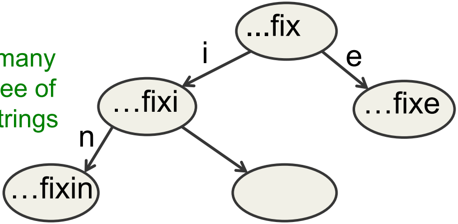
```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#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;
}
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 44 8 Feb 2016

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.



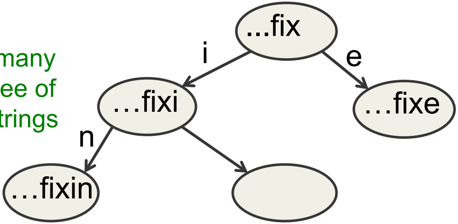
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- 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.



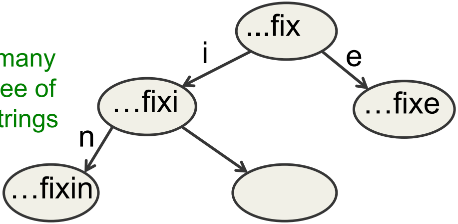
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- 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

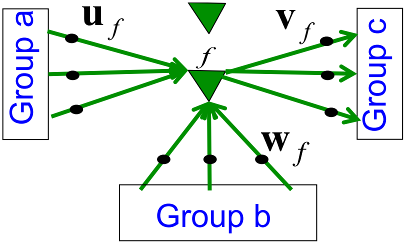
Multiplicative connections

- 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 86x1500x1500 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 86x1500x1500 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

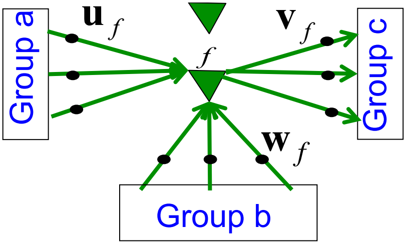


Vector input to group *c*:

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

- 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

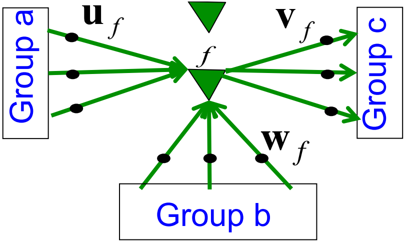


Vector input to group *c*:

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

- 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

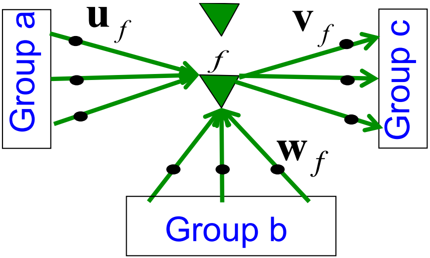


Vector input to group c:

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

- 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

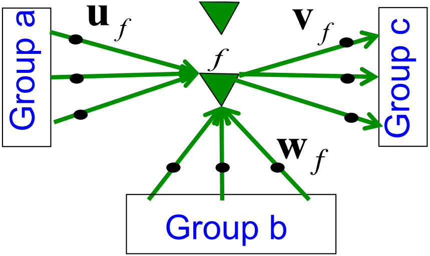


$$
\begin{aligned}
& c_f \\
&= (b^T w_f)(a^T u_f) v_f \\
&= (b^T w_f) v_f (u_f^T a) \\
&= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
\end{aligned}
$$

- 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

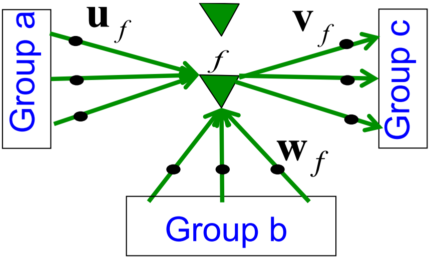


$$
\begin{aligned}
& c_f \\
&= (b^T w_f)(a^T u_f) v_f \\
&= (b^T w_f) v_f (u_f^T a) \\
&= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
\end{aligned}
$$

- 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

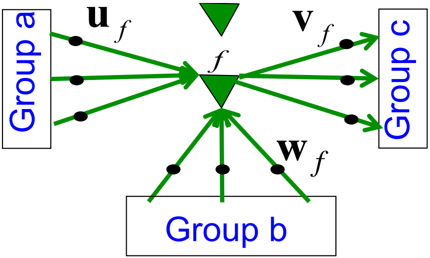


$$\begin{aligned} & c_f \\ =& (b^T w_f)(a^T u_f) v_f \\ =& (b^T w_f) v_f (u_f^T a) \\ =& \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a \end{aligned}$$

- 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

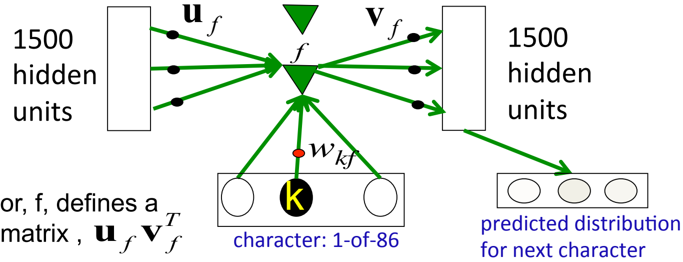


$$
\begin{aligned}
& c_f \\
&= (b^T w_f)(a^T u_f) v_f \\
&= (b^T w_f) v_f (u_f^T a) \\
&= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
\end{aligned}
$$

- 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 (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 (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 (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

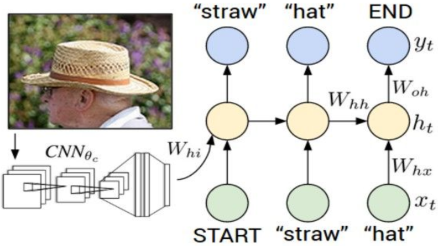
Result: what does it know?

- 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

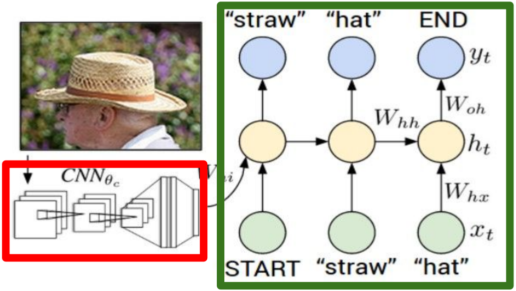
- Tomas Mikolov and his collaborators have recently trained quite large RNNs on quite large training sets using backprop through time (BPTT)
    - They do better than feed-forward neural nets
    - They do better than the best other models
    - They do even better when averaged with other models
- RNNs 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

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 52 8 Feb 2016

test image

Example: image captioning

test image

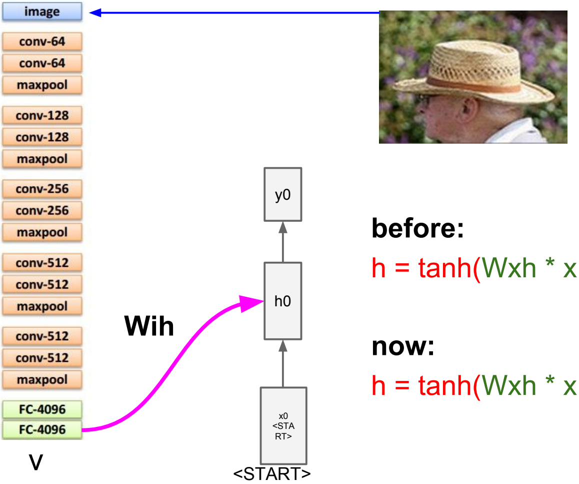

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
X

test image

Example: image captioning

test image

<START>

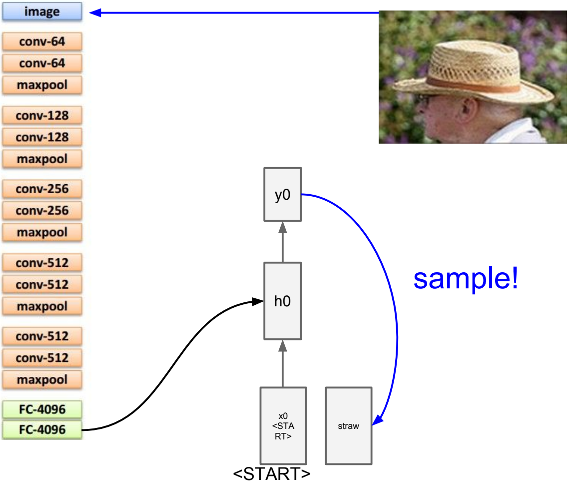


before:

$h = \tanh(Wxh * x + Whh * h)$

now:

$h = \tanh(Wxh * x + Whh * h + \mathbf{Wih} * \mathbf{v})$



test image

sample!

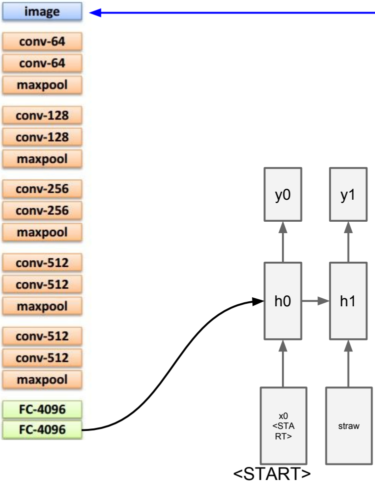
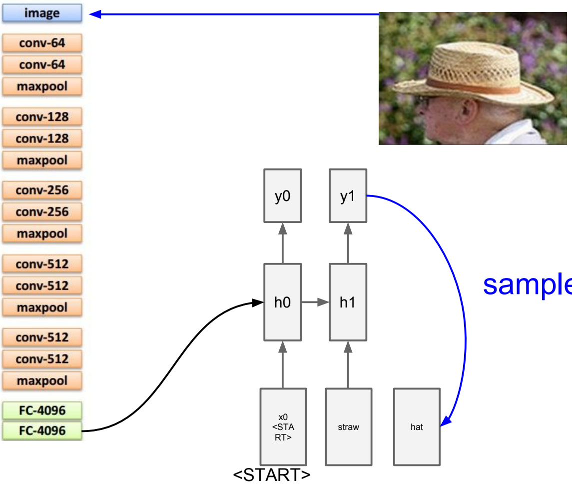


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0
<STA
RT>
straw
<START>

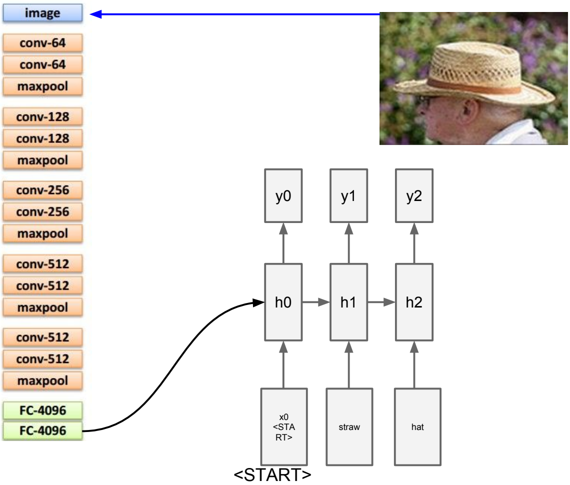
test image



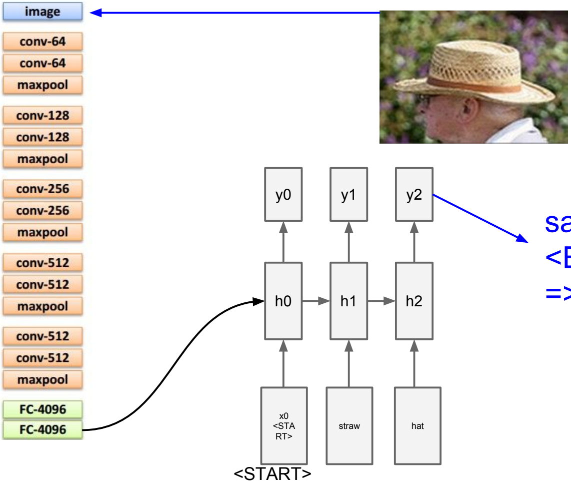
test image

sample!

Example: image captioning



test image



test image

sample <END> token => finish.

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."

"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."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

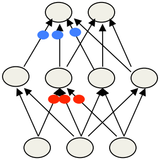
"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

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

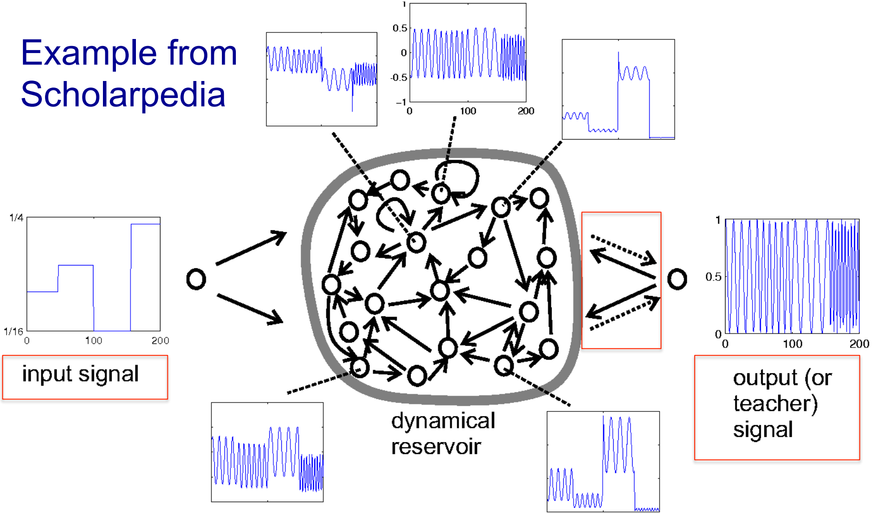
A simple example of an echo state network

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

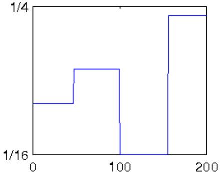


input signal

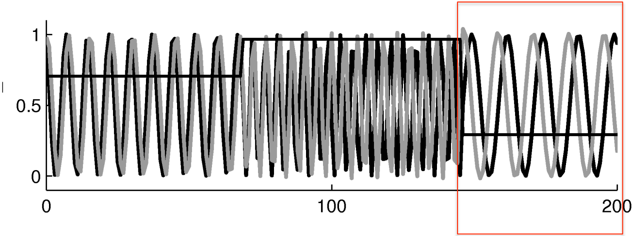
dynamical reservoir

output (or teacher) signal

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

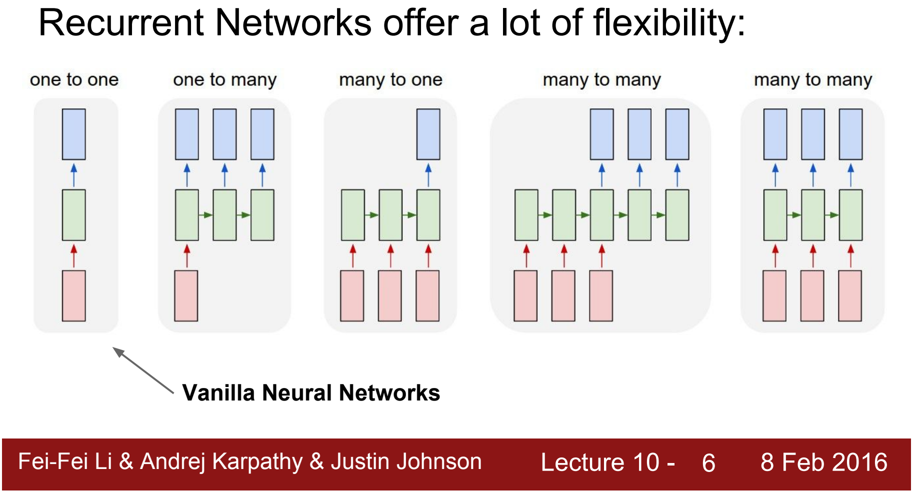
Beyond echo state networks

- 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

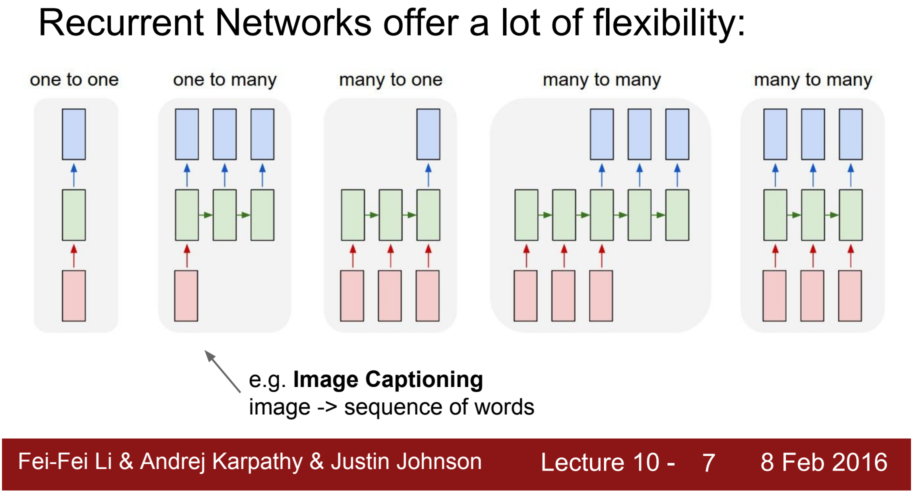
- RNNs allow a lot of flexibility in architecture design and have many applications



Recurrent Networks offer a lot of flexibility:

Vanilla Neural Networks

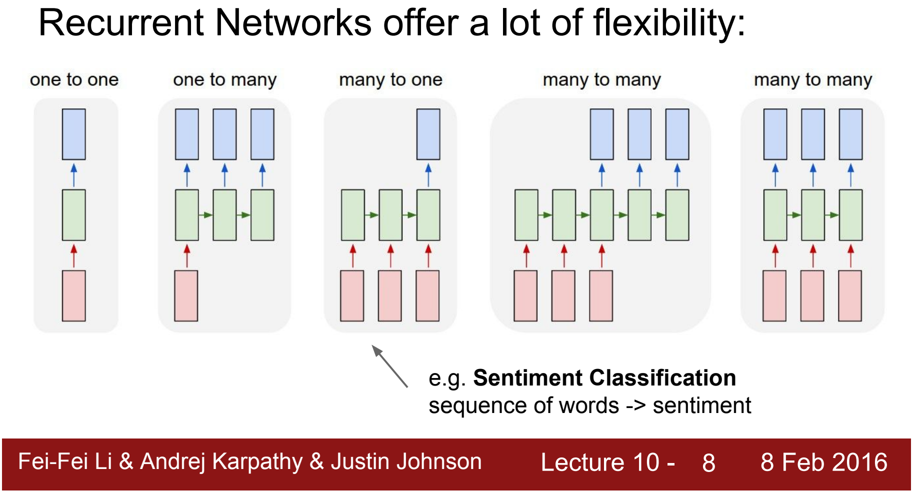
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 6 8 Feb 2016



Recurrent Networks offer a lot of flexibility:

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

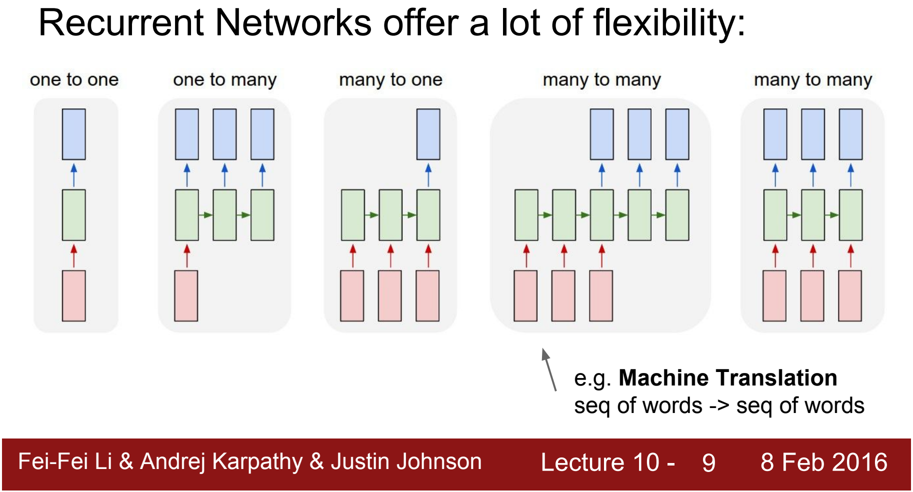
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 7 8 Feb 2016



Recurrent Networks offer a lot of flexibility:

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

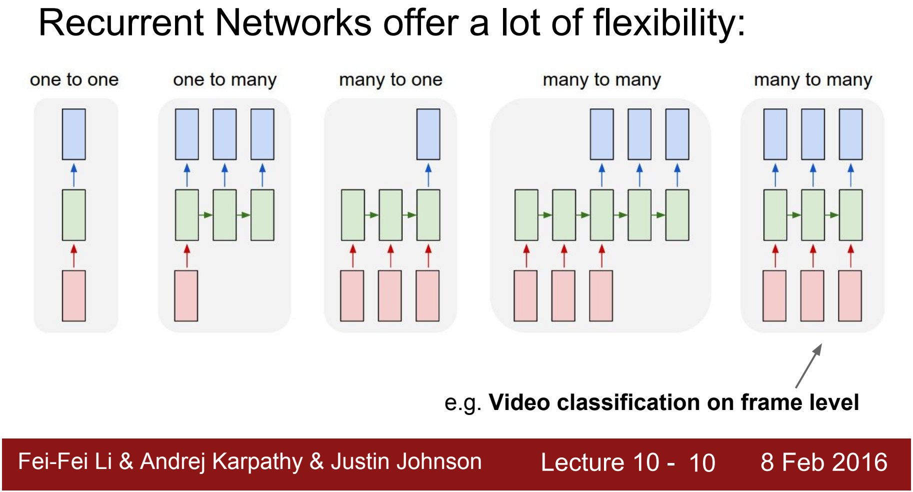
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 8 8 Feb 2016



Recurrent Networks offer a lot of flexibility:

e.g. **Machine Translation**
seq of words -> seq of words

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 10 - 9 8 Feb 2016



Recurrent Networks offer a lot of flexibility:

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

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

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