

Recurrent Neural Networks

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2018

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

Table of Contents

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

- We looked into couple use cases of CNNs previously
 - 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?

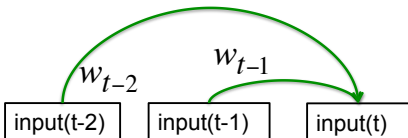
Why non-memoryless models

- 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?

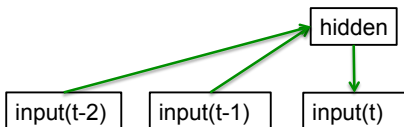
Engineering hacks [Hinton 2012, week 7]

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.*



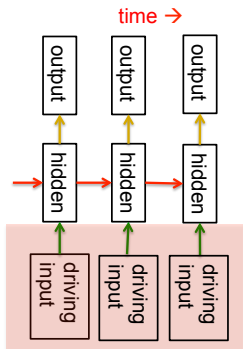
Non-memoryless models

- **Benefit:** memories increase the expressive power of the model
- 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

Non-memoryless models

- Benefit: memories increase the expressive power of the model
- Typically we do not know the exact values of the hidden states (that is why “hidden”). In many cases, the best we could do is just to infer a probability distribution over the hidden states
- Let's look at two classic examples

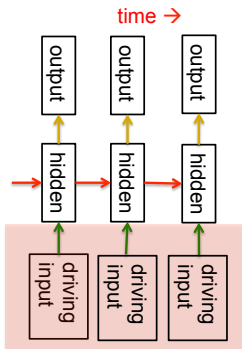
Linear dynamical systems (Engineers love them!)



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

[Hinton 2012, Week 7]

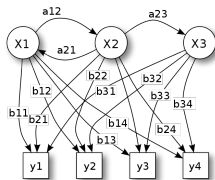
Linear dynamical systems (Engineers love them!)



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

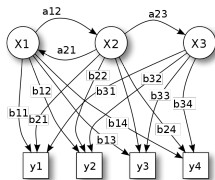
[Hinton 2012, Week 7]

State-Space Models (Computer scientists love them!)



- State-Space Models or 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

State-Space Models (Computer scientists love them!)



- State-Space Models or 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

A fundamental limitation of state space models

- The only information stored in the model is which state the model currently is in
 - So with N hidden states it can only remember a maximum $\log(N)$ bits of information
- Consider the speech prediction of one half from earlier 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 state space models

- The only information stored in the model is which state the model currently is in
 - So with N hidden states it can only remember a maximum $\log(N)$ bits of information
- Consider the speech prediction of one half from earlier 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]

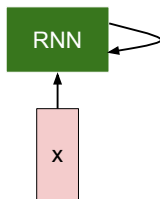
A fundamental limitation of state space models

- The only information stored in the model is which state the model currently is in
 - So with N hidden states it can only remember a maximum $\log(N)$ bits of information
- Consider the speech prediction of one half from earlier half
 - The syntax needs to fit (e.g. number and tense agreement)
 - The semantics needs to fit. The intonation needs to fit
 - The accent, rate, volume, and vocal tract characteristics must all fit
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half 2^{100} states

[Hinton 2012, week 7]

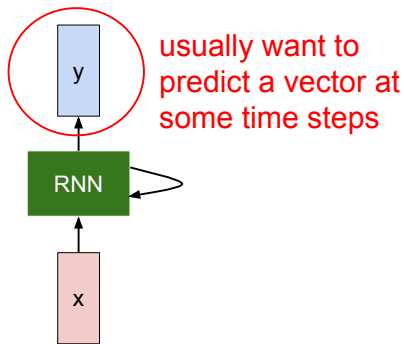
Recurrent neural networks (RNNs)

Recurrent Neural Network



Recurrent neural networks (RNNs)

Recurrent Neural Network



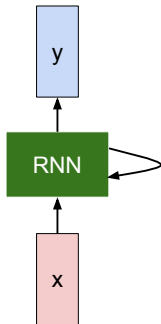
Recurrent neural networks (RNNs)

Recurrent Neural Network

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

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

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



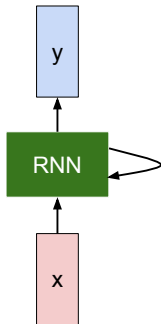
Recurrent neural networks (RNNs)

Recurrent Neural Network

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

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

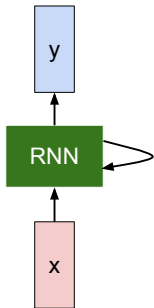
Notice: the same function and the same set of parameters are used at every time step.



Recurrent neural networks (RNNs)

(Vanilla) Recurrent Neural Network

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



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

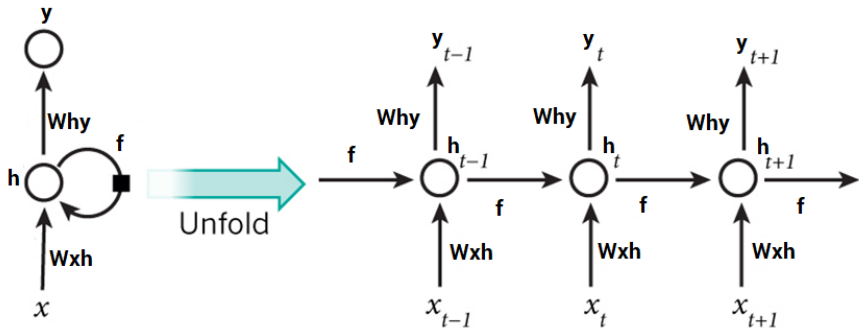


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

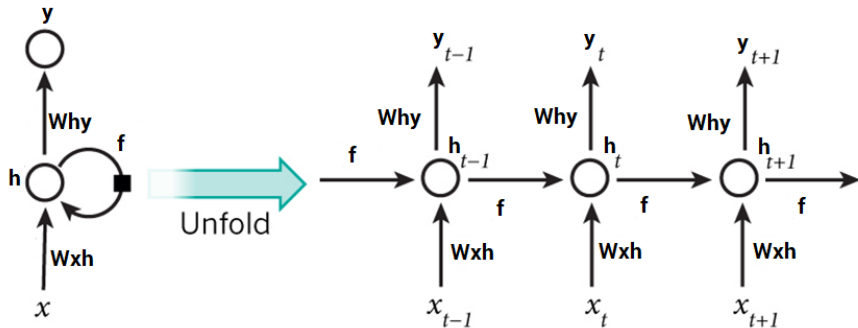
Back-Propagation Through Time (BPTT)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- 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



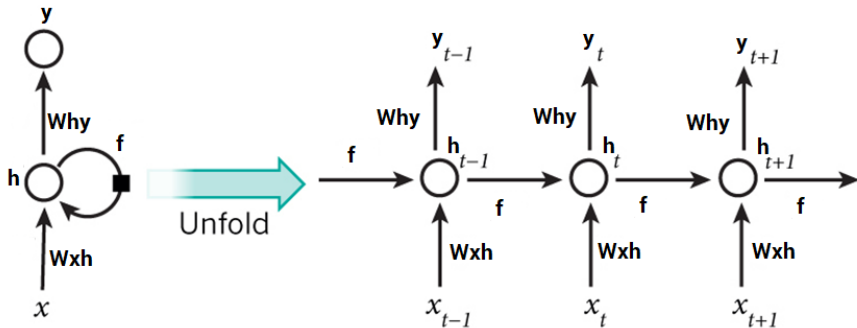
Back-Propagation Through Time (BPTT)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- 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)

- For training, we can unroll all the time step to form a stack of activities and backprop will then similar to regular backprop
- 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 irritating 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

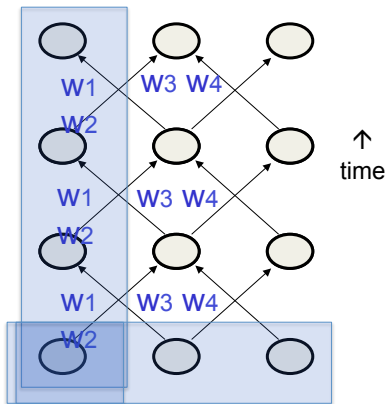
An irritating extra issue

- 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

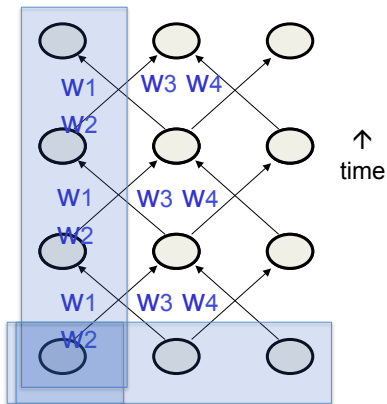
- We need to specify the initial activity state of all the hidden and output units
- We could just fix these initial states to have some default value like 0.5
- But it is better to treat the initial states as learned parameters
- We learn them in the same way as we learn the weights
 - Start off with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



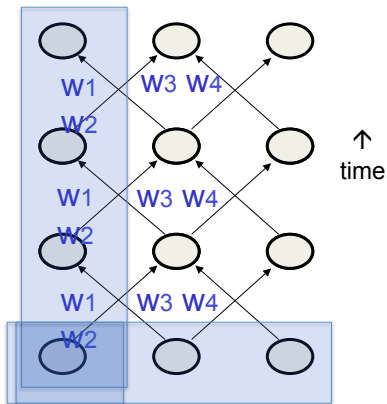
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



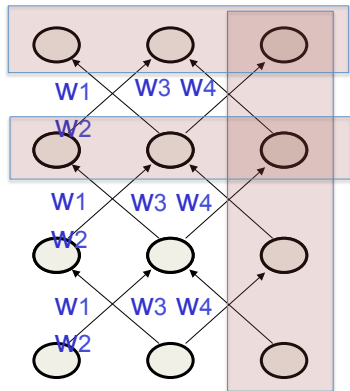
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Providing inputs to recurrent networks



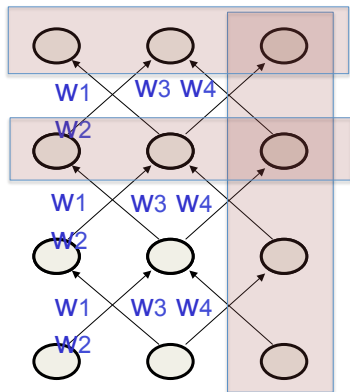
- We can specify inputs in several ways:
 - Specify the initial states of all the units
 - Specify the initial states of a subset of the units
 - Specify the states of the same subset of the units at every time step

Teaching recurrent networks to learn signals



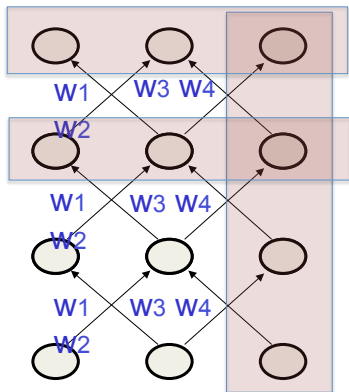
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



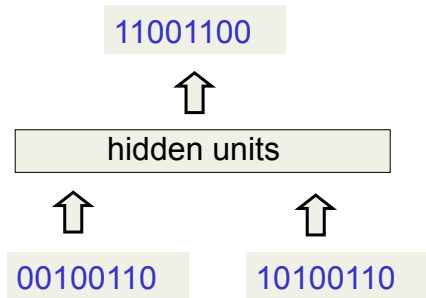
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Teaching recurrent networks to learn signals



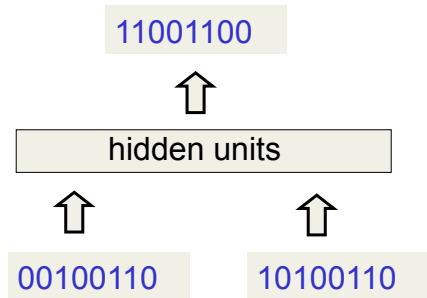
- We can specify targets in several ways:
 - Specify desired final activities of all the units
 - Specify desired activities of all units for the last few steps
 - Good for learning attractors
 - Specify the desired activity of a subset of the units.
 - The other units are input or hidden units.

Toy problem for RNN: binary addition



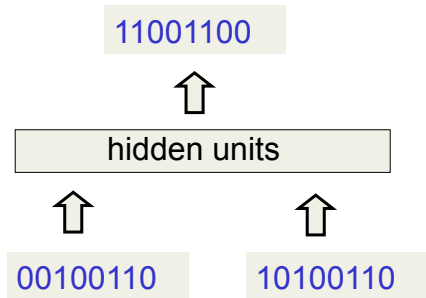
- We can train a feedforward net to do binary addition, but...
 - We must decide in advance the maximum number of digits in each number
 - We expect weights to process different bits to be the same, but it is tricky to enforce that
- As a result, feedforward nets do not generalize well for the binary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

Toy problem for RNN: binary addition



- We can train a feedforward net to do binary addition, but...
 - 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

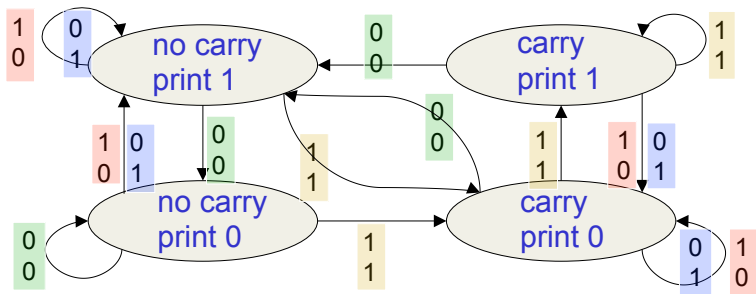
Toy problem for RNN: binary addition



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

We are trying to learn this!

The algorithm for binary addition



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

A little bit detail

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

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

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

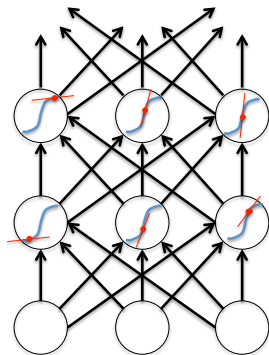
$$\hat{z} = [\hat{d}_8, \hat{d}_7, \dots, \hat{d}_1]$$

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

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

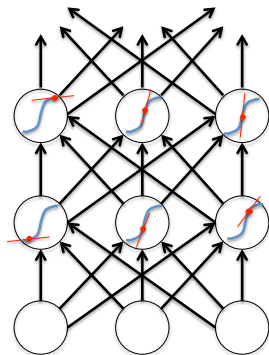
https://github.com/llSourcecell/recurrent_neural_net_demo

Why training RNN is difficulty? The backward pass is linear



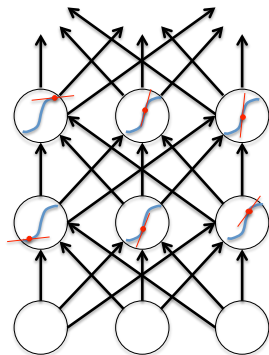
- There is a big difference between the forward and backward passes
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives 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, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - 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, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - 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, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - 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, the dependency of the current target output from an input many time-steps ago tends to be numerically unstable
 - So RNNs have difficulty dealing with long-range dependencies

Understanding gradient flow dynamics

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

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

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

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

Understanding gradient flow dynamics

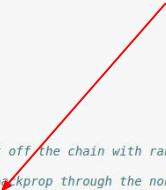
```

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

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

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

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



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

Four effective ways to learn an RNN

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

Four effective ways to learn an RNN

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

Four effective ways to learn an RNN

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

Long Short Term Memory (LSTM)

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

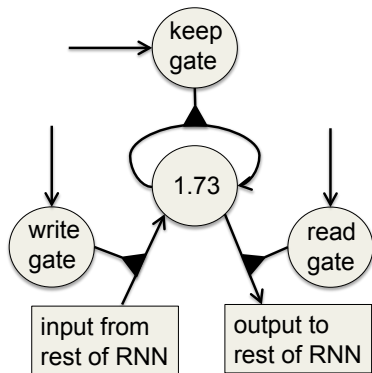
Long Short Term Memory (LSTM)

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

Long Short Term Memory (LSTM)

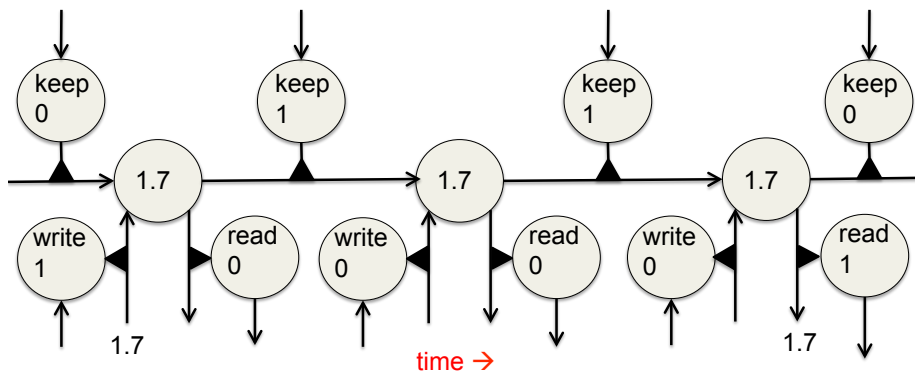
- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps)
 - Keep short-term memory for a long period of time, thus the name
- They designed a memory cell using logistic and linear units with multiplicative interactions
- Information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

Implementing a memory cell in a neural network



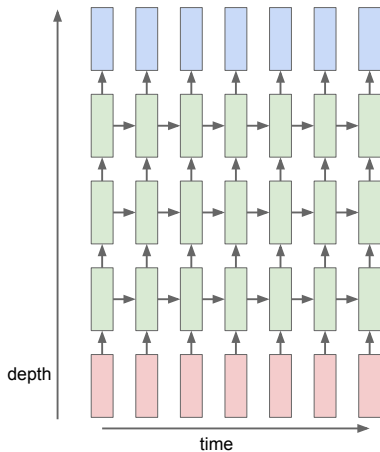
- To preserve information for a long time in the activities of an RNN, we use a circuit mimicking an analog memory cell
 - Information is kept in the cell when "keep" gate is on
 - Information is stored in the cell by activating its write gate
 - Information is retrieved by activating the read gate
 - We can backpropagate through this circuit because logistics are have nice derivatives

Backpropagation through a memory cell



RNN:

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

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


Fei-Fei Li & Andrej Karpathy & Justin Johnson

Lecture 10 - 67

8 Feb 2016

RNN:

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

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

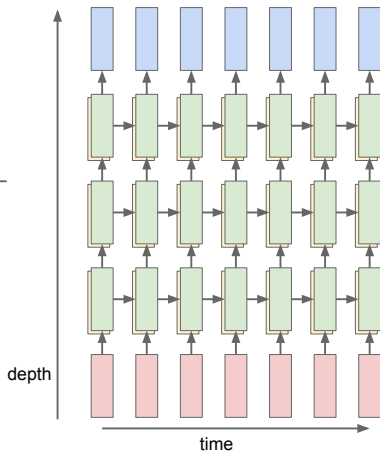
LSTM:

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

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

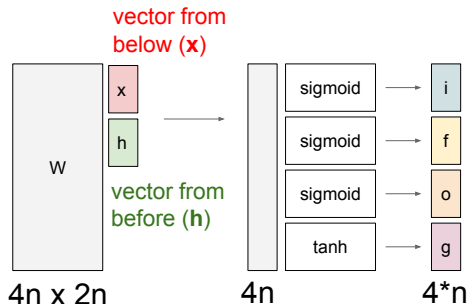
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



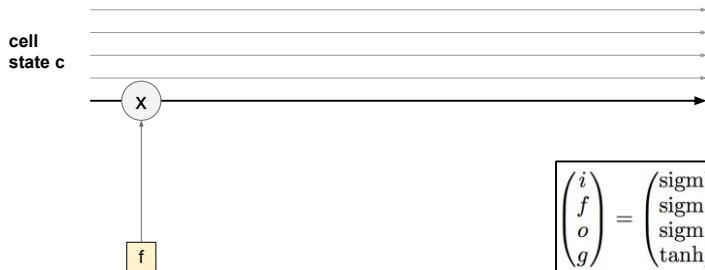
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



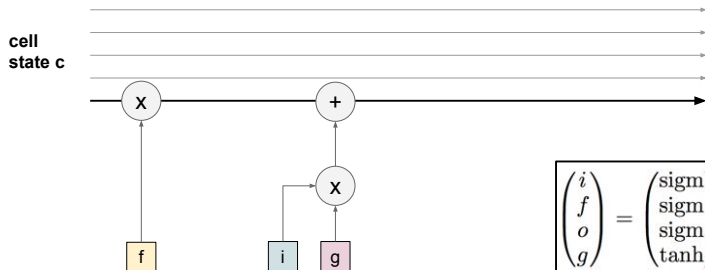
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^{l-1} \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



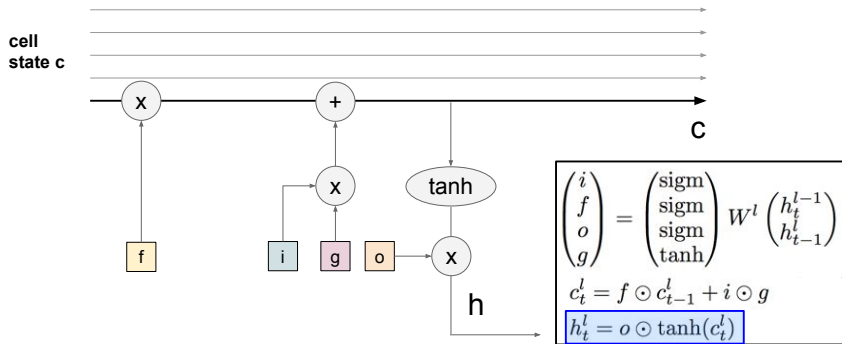
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_t^{l-1} \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



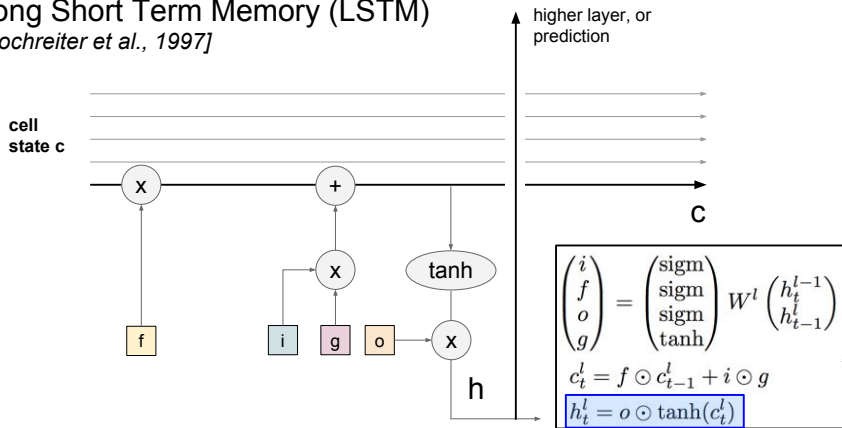
Fei-Fei Li & Andrej Karpathy & Justin Johnson

Lecture 10 - 73

8 Feb 2016

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

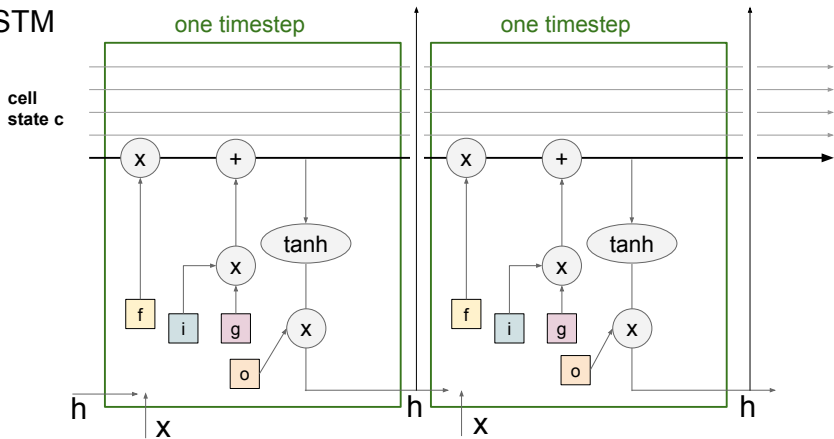


Fei-Fei Li & Andrej Karpathy & Justin Johnson

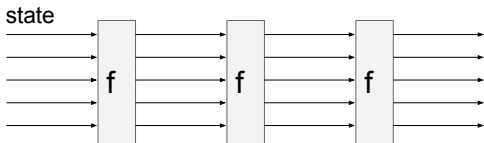
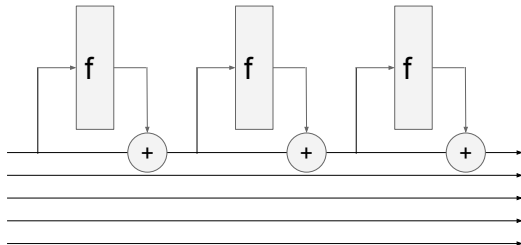
Lecture 10 - 74

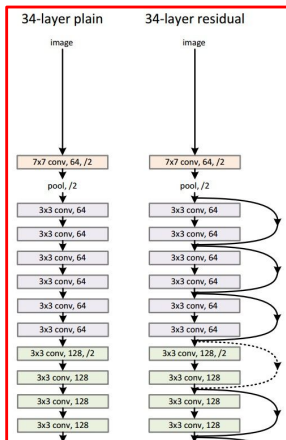
8 Feb 2016

LSTM



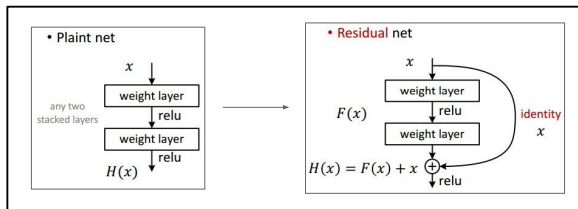
RNN

LSTM
(ignoring
forget gates)

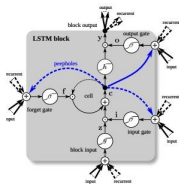


Recall: “PlainNets” vs. ResNets

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



LSTM variants and friends



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

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

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

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

MUT1:

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

MUT2:

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

MUT3:

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

Modelling text: why 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: why working with characters?

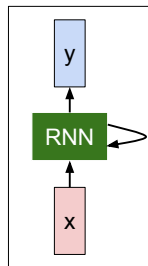
- The web is composed of character strings
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words (this turns out to be true, as we shall see)
- Pre-processing text to get words is a big hassle
 - What about morphemes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

Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

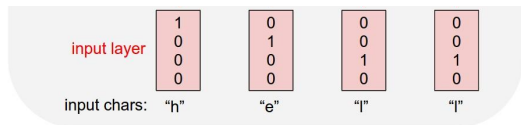


Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



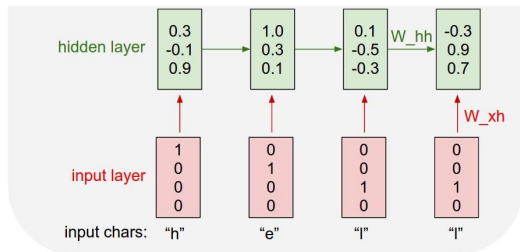
Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training
sequence:
"hello"

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

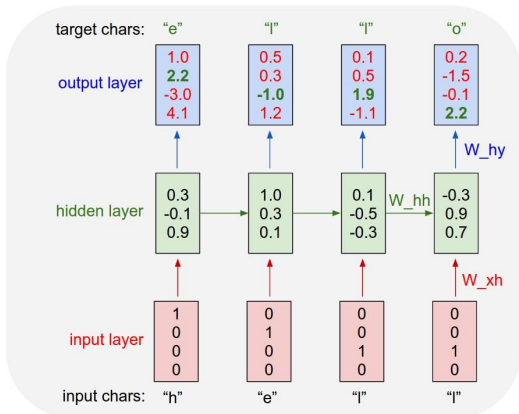


Simplest model: a first attempt

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



Sampling

- 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

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

```

1 """
2 min-char-rnn: character-level vanilla rnn model, written by Andrej Karpathy (@karpathy)
3 BSD License
4 """
5 import numpy as np
6
7 # Data I/O
8 data = open('input.txt', 'r').read() # should be simple plain text file
9 chars = list(data)
10
11 data_size, vocab_size = len(chars), len(chars)
12 print('Data has %d characters, %d unique %s (%d,%d,%d)' %
13      (data_size, vocab_size, 'chars', data_size, vocab_size, data_size))
14 char_ix_to_id = {char: i for i, char in enumerate(chars)}
15
16 # Hyperparameters
17 hidden_size = 100 # size of hidden layer of neurons
18 rnn_length = 20 # number of steps to unroll the rnn for
19 learning_rate = 1e-3
20
21 # Model parameters
22 wih = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
23 whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
24 wh0 = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to output
25 bh = np.zeros([hidden_size, 1]) # hidden bias
26 b0 = np.zeros([vocab_size, 1]) # output bias
27
28 def lstm_to_inputs, targets, hprev):
29     """
30     inputs, targets are both lists of integers.
31     hprev is the array of hidden states
32     returns the loss, gradients on model parameters, and last hidden state
33     """
34     ix, hx, yx, ph = 0, 0, 0, 0
35     h0 = [] # no memory
36     loss = 0
37     for i in range(len(inputs)):
38         x[i] = np.zeros([vocab_size, 1]) # encode as 1-hot representation
39         hx[i:i+rnn_length] = h # unroll as 1-hot x representation
40         y[i] = inputs[i]
41         yx[i] = np.zeros([vocab_size, 1]) # no output, only probabilities for next char
42         ph[i] = np.zeros([1]) # no memory, only probabilities for next char
43         loss += -np.log(list(yx[i][target[i]])) # softmax (cross-entropy loss)
44         # backward pass: compute gradients across backwards
45         dwh, dwh0, dhdy = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
46         dh, dh0 = np.zeros([hidden_size]), np.zeros([hidden_size])
47         dprev = np.zeros([hidden_size])
48         for i in reversed(range(i-i+rnn_length)):
49             dy = np.zeros([1])
50             dy[targets[i]] = 1 # backprop into y
51             dhdy = np.zeros([hidden_size, 1])
52             dh0 = 0
53             dhw = ci - h[i] + h[i+1] + dh # backward through tanh nonlinearity
54             dwh = dhw
55             dwh0 = np.zeros([1])
56             dhw = np.zeros([1])
57             dhw0 = np.zeros([1])
58             dhw0 = np.zeros([1])
59             dhw0 = np.zeros([1])
60             for dwh, dwh0, dhdy, dh0, dy:
61                 # backward pass: compute gradients across backwards
62                 return loss, dwh, dwh0, dhdy, dh0, dy, h[i:i+rnn_length]
63
64 def lstm_to_inputs, hprev):
65     """
66     hprev is the array of hidden states
67     returns the loss, gradients on model parameters, and last hidden state
68     """
69     ix, hx, yx, ph = 0, 0, 0, 0
70     h0 = [] # no memory
71     loss = 0
72     for i in range(len(inputs)):
73         x[i] = np.zeros([vocab_size, 1]) # encode as 1-hot representation
74         hx[i:i+rnn_length] = h # unroll as 1-hot x representation
75         y[i] = inputs[i]
76         yx[i] = np.zeros([vocab_size, 1]) # no output, only probabilities for next char
77         ph[i] = np.zeros([1]) # no memory, only probabilities for next char
78         loss += -np.log(list(yx[i][target[i]])) # softmax (cross-entropy loss)
79         # backward pass: compute gradients across backwards
80         dwh, dwh0, dhdy = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
81         dh, dh0 = np.zeros([hidden_size]), np.zeros([hidden_size])
82         dprev = np.zeros([hidden_size])
83         for i in reversed(range(i-i+rnn_length)):
84             dy = np.zeros([1])
85             dy[targets[i]] = 1 # backprop into y
86             dhdy = np.zeros([hidden_size, 1])
87             dh0 = 0
88             dhw = ci - h[i] + h[i+1] + dh # backward through tanh nonlinearity
89             dwh = dhw
90             dwh0 = np.zeros([1])
91             dhw = np.zeros([1])
92             dhw0 = np.zeros([1])
93             dhw0 = np.zeros([1])
94             for dwh, dwh0, dhdy, dh0, dy:
95                 # backward pass: compute gradients across backwards
96                 return loss, dwh, dwh0, dhdy, dh0, dy, h[i:i+rnn_length]
97
98 # Main loop
99 n, n0, n1 = 0, 0, 0
100 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
101 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
102 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
103 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
104 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
105 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
106 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
107 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
108 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
109 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
110 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
111 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])
112 h0, h1, h2, h3, h4, h5, h6, h7, h8, h9 = np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size]), np.zeros([hidden_size])

```

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

min-char-rnn.py gist

```

1 # coding: utf-8
2 """
3 Example: character-level language model, written by Andrej Karpathy (2015/04/23)
4 """
5
6 # imports
7
8 # model class
9
10 # data loader
11
12 # training
13
14 # inference
15
16 # main
17
18 # test
19
20 # main
21
22 # test
23
24 # main
25
26 # test
27
28 # main
29
30 # test
31
32 # main
33
34 # test
35
36 # main
37
38 # test
39
40 # main
41
42 # test
43
44 # main
45
46 # test
47
48 # main
49
50 # test
51
52 # main
53
54 # test
55
56 # main
57
58 # test
59
60 # main
61
62 # test
63
64 # main
65
66 # test
67
68 # main
69
70 # test
71
72 # main
73
74 # test
75
76 # main
77
78 # test
79
80 # main
81
82 # test
83
84 # main
85
86 # test
87
88 # main
89
90 # test
91
92 # main
93
94 # test
95
96 # main
97
98 # test
99
100 # main
101
102 # test
103
104 # main
105
106 # test
107
108 # main
109
110 # test
111
112 # main
113
114 # test
115
116 # main
117
118 # test
119
120 # main
121
122 # test
123
124 # main
125
126 # test
127
128 # main
129
130 # test
131
132 # main
133
134 # test
135
136 # main
137
138 # test
139
140 # main
141
142 # test
143
144 # main
145
146 # test
147
148 # main
149
150 # test
151
152 # main
153
154 # test
155
156 # main
157
158 # test
159
160 # main
161
162 # test
163
164 # main
165
166 # test
167
168 # main
169
170 # test
171
172 # main
173
174 # test
175
176 # main
177
178 # test
179
180 # main
181
182 # test
183
184 # main
185
186 # test
187
188 # main
189
190 # test
191
192 # main
193
194 # test
195
196 # main
197
198 # test
199
200 # main
201
202 # test
203
204 # main
205
206 # test
207
208 # main
209
210 # test
211
212 # main
213
214 # test
215
216 # main
217
218 # test
219
220 # main
221
222 # test
223
224 # main
225
226 # test
227
228 # main
229
230 # test
231
232 # main
233
234 # test
235
236 # main
237
238 # test
239
240 # main
241
242 # test
243
244 # main
245
246 # test
247
248 # main
249
250 # test
251
252 # main
253
254 # test
255
256 # main
257
258 # test
259
260 # main
261
262 # test
263
264 # main
265
266 # test
267
268 # main
269
270 # test
271
272 # main
273
274 # test
275
276 # main
277
278 # test
279
280 # main
281
282 # test
283
284 # main
285
286 # test
287
288 # main
289
290 # test
291
292 # main
293
294 # test
295
296 # main
297
298 # test
299
300 # main
301
302 # test
303
304 # main
305
306 # test
307
308 # main
309
310 # test
311
312 # main
313
314 # test
315
316 # main
317
318 # test
319
320 # main
321
322 # test
323
324 # main
325
326 # test
327
328 # main
329
330 # test
331
332 # main
333
334 # test
335
336 # main
337
338 # test
339
340 # main
341
342 # test
343
344 # main
345
346 # test
347
348 # main
349
350 # test
351
352 # main
353
354 # test
355
356 # main
357
358 # test
359
360 # main
361
362 # test
363
364 # main
365
366 # test
367
368 # main
369
370 # test
371
372 # main
373
374 # test
375
376 # main
377
378 # test
379
380 # main
381
382 # test
383
384 # main
385
386 # test
387
388 # main
389
390 # test
391
392 # main
393
394 # test
395
396 # main
397
398 # test
399
400 # main
401
402 # test
403
404 # main
405
406 # test
407
408 # main
409
410 # test
411
412 # main
413
414 # test
415
416 # main
417
418 # test
419
420 # main
421
422 # test
423
424 # main
425
426 # test
427
428 # main
429
430 # test
431
432 # main
433
434 # test
435
436 # main
437
438 # test
439
440 # main
441
442 # test
443
444 # main
445
446 # test
447
448 # main
449
450 # test
451
452 # main
453
454 # test
455
456 # main
457
458 # test
459
460 # main
461
462 # test
463
464 # main
465
466 # test
467
468 # main
469
470 # test
471
472 # main
473
474 # test
475
476 # main
477
478 # test
479
480 # main
481
482 # test
483
484 # main
485
486 # test
487
488 # main
489
490 # test
491
492 # main
493
494 # test
495
496 # main
497
498 # test
499
500 # main
501
502 # test
503
504 # main
505
506 # test
507
508 # main
509
510 # test
511
512 # main
513
514 # test
515
516 # main
517
518 # test
519
520 # main
521
522 # test
523
524 # main
525
526 # test
527
528 # main
529
530 # test
531
532 # main
533
534 # test
535
536 # main
537
538 # test
539
540 # main
541
542 # test
543
544 # main
545
546 # test
547
548 # main
549
550 # test
551
552 # main
553
554 # test
555
556 # main
557
558 # test
559
560 # main
561
562 # test
563
564 # main
565
566 # test
567
568 # main
569
570 # test
571
572 # main
573
574 # test
575
576 # main
577
578 # test
579
580 # main
581
582 # test
583
584 # main
585
586 # test
587
588 # main
589
590 # test
591
592 # main
593
594 # test
595
596 # main
597
598 # test
599
600 # main
601
602 # test
603
604 # main
605
606 # test
607
608 # main
609
610 # test
611
612 # main
613
614 # test
615
616 # main
617
618 # test
619
620 # main
621
622 # test
623
624 # main
625
626 # test
627
628 # main
629
630 # test
631
632 # main
633
634 # test
635
636 # main
637
638 # test
639
640 # main
641
642 # test
643
644 # main
645
646 # test
647
648 # main
649
650 # test
651
652 # main
653
654 # test
655
656 # main
657
658 # test
659
660 # main
661
662 # test
663
664 # main
665
666 # test
667
668 # main
669
670 # test
671
672 # main
673
674 # test
675
676 # main
677
678 # test
679
680 # main
681
682 # test
683
684 # main
685
686 # test
687
688 # main
689
690 # test
691
692 # main
693
694 # test
695
696 # main
697
698 # test
699
700 # main
701
702 # test
703
704 # main
705
706 # test
707
708 # main
709
710 # test
711
712 # main
713
714 # test
715
716 # main
717
718 # test
719
720 # main
721
722 # test
723
724 # main
725
726 # test
727
728 # main
729
730 # test
731
732 # main
733
734 # test
735
736 # main
737
738 # test
739
740 # main
741
742 # test
743
744 # main
745
746 # test
747
748 # main
749
750 # test
751
752 # main
753
754 # test
755
756 # main
757
758 # test
759
760 # main
761
762 # test
763
764 # main
765
766 # test
767
768 # main
769
770 # test
771
772 # main
773
774 # test
775
776 # main
777
778 # test
779
780 # main
781
782 # test
783
784 # main
785
786 # test
787
788 # main
789
790 # test
791
792 # main
793
794 # test
795
796 # main
797
798 # test
799
800 # main
801
802 # test
803
804 # main
805
806 # test
807
808 # main
809
810 # test
811
812 # main
813
814 # test
815
816 # main
817
818 # test
819
820 # main
821
822 # test
823
824 # main
825
826 # test
827
828 # main
829
830 # test
831
832 # main
833
834 # test
835
836 # main
837
838 # test
839
840 # main
841
842 # test
843
844 # main
845
846 # test
847
848 # main
849
850 # test
851
852 # main
853
854 # test
855
856 # main
857
858 # test
859
860 # main
861
862 # test
863
864 # main
865
866 # test
867
868 # main
869
870 # test
871
872 # main
873
874 # test
875
876 # main
877
878 # test
879
880 # main
881
882 # test
883
884 # main
885
886 # test
887
888 # main
889
890 # test
891
892 # main
893
894 # test
895
896 # main
897
898 # test
899
900 # main
901
902 # test
903
904 # main
905
906 # test
907
908 # main
909
910 # test
911
912 # main
913
914 # test
915
916 # main
917
918 # test
919
920 # main
921
922 # test
923
924 # main
925
926 # test
927
928 # main
929
930 # test
931
932 # main
933
934 # test
935
936 # main
937
938 # test
939
940 # main
941
942 # test
943
944 # main
945
946 # test
947
948 # main
949
950 # test
951
952 # main
953
954 # test
955
956 # main
957
958 # test
959
960 # main
961
962 # test
963
964 # main
965
966 # test
967
968 # main
969
970 # test
971
972 # main
973
974 # test
975
976 # main
977
978 # test
979
980 # main
981
982 # test
983
984 # main
985
986 # test
987
988 # main
989
990 # test
991
992 # main
993
994 # test
995
996 # main
997
998 # test
999
1000 # main

```

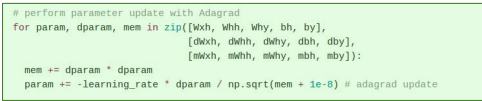


Main loop

```

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

```



min-char-rnn.py gist

```

1 # min-char-rnn.py
2 # Licensed under the MIT license
3 # Author: Andrew Senior
4 # Date: 2016-05-10
5 # Description: A simple character-level language model using an RNN.
6 # This script is a simplified version of the min-char-rnn.py script found in the
7 # min-char-rnn repository.
8 # It uses a simple RNN architecture with a single hidden layer and a linear
9 # output layer.
10 # The input is a sequence of characters, and the output is a sequence of
11 # characters.
12 # The model is trained using a simple gradient descent algorithm.
13 # The training process is controlled by the command-line arguments.
14 # The script is designed to be run from the command line.
15 # The output is a sequence of characters, which is printed to the
16 # standard output.
17 # The script is designed to be run from the command line.
18 # The output is a sequence of characters, which is printed to the
19 # standard output.
20 # The script is designed to be run from the command line.
21 # The output is a sequence of characters, which is printed to the
22 # standard output.
23 # The script is designed to be run from the command line.
24 # The output is a sequence of characters, which is printed to the
25 # standard output.
26 # The script is designed to be run from the command line.
27 # The output is a sequence of characters, which is printed to the
28 # standard output.
29 # The script is designed to be run from the command line.
30 # The output is a sequence of characters, which is printed to the
31 # standard output.
32 # The script is designed to be run from the command line.
33 # The output is a sequence of characters, which is printed to the
34 # standard output.
35 # The script is designed to be run from the command line.
36 # The output is a sequence of characters, which is printed to the
37 # standard output.
38 # The script is designed to be run from the command line.
39 # The output is a sequence of characters, which is printed to the
40 # standard output.
41 # The script is designed to be run from the command line.
42 # The output is a sequence of characters, which is printed to the
43 # standard output.
44 # The script is designed to be run from the command line.
45 # The output is a sequence of characters, which is printed to the
46 # standard output.
47 # The script is designed to be run from the command line.
48 # The output is a sequence of characters, which is printed to the
49 # standard output.
50 # The script is designed to be run from the command line.
51 # The output is a sequence of characters, which is printed to the
52 # standard output.
53 # The script is designed to be run from the command line.
54 # The output is a sequence of characters, which is printed to the
55 # standard output.
56 # The script is designed to be run from the command line.
57 # The output is a sequence of characters, which is printed to the
58 # standard output.
59 # The script is designed to be run from the command line.
60 # The output is a sequence of characters, which is printed to the
61 # standard output.
62 # The script is designed to be run from the command line.
63 # The output is a sequence of characters, which is printed to the
64 # standard output.
65 # The script is designed to be run from the command line.
66 # The output is a sequence of characters, which is printed to the
67 # standard output.
68 # The script is designed to be run from the command line.
69 # The output is a sequence of characters, which is printed to the
70 # standard output.
71 # The script is designed to be run from the command line.
72 # The output is a sequence of characters, which is printed to the
73 # standard output.
74 # The script is designed to be run from the command line.
75 # The output is a sequence of characters, which is printed to the
76 # standard output.
77 # The script is designed to be run from the command line.
78 # The output is a sequence of characters, which is printed to the
79 # standard output.
80 # The script is designed to be run from the command line.
81 # The output is a sequence of characters, which is printed to the
82 # standard output.
83 # The script is designed to be run from the command line.
84 # The output is a sequence of characters, which is printed to the
85 # standard output.
86 # The script is designed to be run from the command line.
87 # The output is a sequence of characters, which is printed to the
88 # standard output.
89 # The script is designed to be run from the command line.
90 # The output is a sequence of characters, which is printed to the
91 # standard output.
92 # The script is designed to be run from the command line.
93 # The output is a sequence of characters, which is printed to the
94 # standard output.
95 # The script is designed to be run from the command line.
96 # The output is a sequence of characters, which is printed to the
97 # standard output.
98 # The script is designed to be run from the command line.
99 # The output is a sequence of characters, which is printed to the
100 # standard output.

```

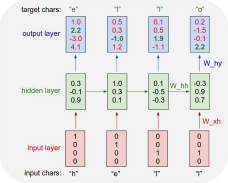
```

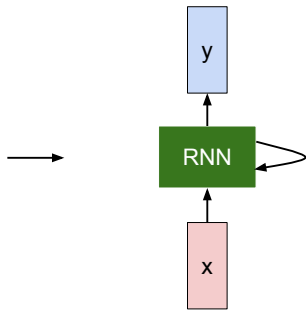
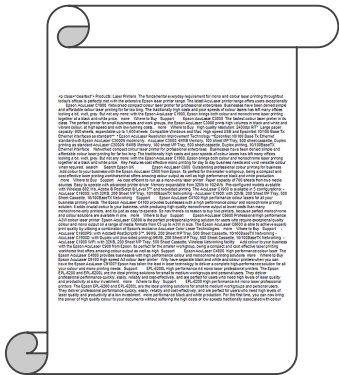
44 # backward pass: compute gradients going backwards
45 dwdx, dwhh, dwhy = np.zeros_like(wxh), np.zeros_like(whh), np.zeros_like(why)
46 dbh, dby = np.zeros_like(bh), np.zeros_like(by)
47 dhnext = np.zeros_like(hs[0])
48 for t in reversed(xrange(len(inputs))):
49     dy = np.copy(ps[t])
50     dy[targets[t]]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

```



recall:





Sonnet 116 – Let me not ...

by William Shakespeare

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

at first:

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

↓
 train more

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

↓
 train more

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

PANDARUS:

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

Second Senator:

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

DUKE VINCENTIO:

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

Second Lord:

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

Clown:

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

VIOLA:

I'll drink it.


VIOLA:

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

KING LEAR:





















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

open source textbook on algebraic geometry

 **The Stacks Project**

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

Browse chapters

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

Parts

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

Statistics

The Stacks project now consists of

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

Latex source 

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

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

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

and the comparably in the fibre product covering we have to prove the lemma generated by $\prod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fpf} and $U \rightarrow T$ 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 $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

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

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

To prove study we see that $\mathcal{F}_{|U}$ is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_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

$$\overline{M}^\bullet = \mathcal{I}^\bullet \otimes_{\otimes_{\text{Spec}(k)} \mathcal{O}_{S, \mathfrak{p}}} \mathcal{O}_{S, \mathfrak{p}} - i_{X'}^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{\text{fpf}}^{\text{ppf}}, (\text{Sch}/S)_{\text{fpf}}$$

and

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

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

Proof. See discussion of sheaves of sets. \square

The result to prove any open covering follows from the less of Example ?? . It may replace S by $X_{\text{spaces}, \text{etale}}$ which gives an open subspace of X and T equal to $S_{Z_{\text{ar}}}$, 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 $\text{Proj}_X(A) = \text{Spec}(B)$ over U compatible with the complex

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

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

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

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

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

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \mathcal{A}_2$ 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 next functor (??). On the other hand, by Lemma ?? we see that

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

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

Proof. Omitted. □

Lemma 0.1. Let \mathcal{C} be a set of the construction.
 Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

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

.

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

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

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

Lemma 0.2. This is an integer Z is injective.
Proof. See Spaces, Lemma ?? □

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

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

be a morphism of algebraic spaces over S and Y .

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

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

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

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

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

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

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

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

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_x \rightarrow \mathcal{O}_{X_{\text{étale}}} \rightarrow \mathcal{O}_X^1(\mathcal{O}_{X,x}) \rightarrow \mathcal{O}_X^2(\mathcal{O}_{X,x})$$

is an isomorphism of covering of $\mathcal{O}_{X,x}$. If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.
 The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S .
 If \mathcal{F} is a scheme theoretic image points. □

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

This repository Search Explore Gist Blog Help karpthy +- ⚙️ 📄

torvalds / linux Watch - 3,711 Star 23,054 Fork 9,141

Linux kernel source tree

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

branch: master - linux / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux latest commit 4b1786927d

torvalds authored 9 hours ago

Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	8 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/...	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

Code 74 Pull requests Pulse Graphs

HTTPS clone URL https://github.c Clone in Desktop Download ZIP

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 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000ffffffff) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}

```

Generated C code

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

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

```

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

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

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

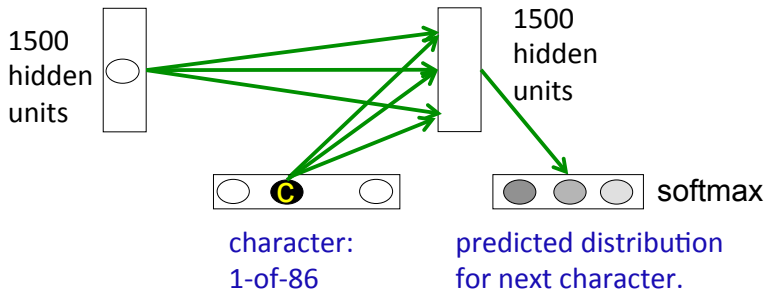
static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

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

```


Ideal model?

An obvious recurrent neural net

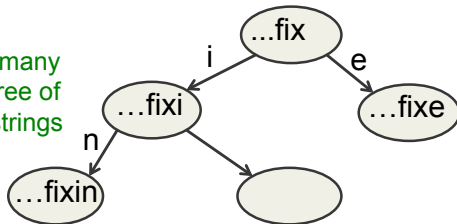


It's a lot easier to predict 86 characters than 100,000 words.

A slight tweak: 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 .



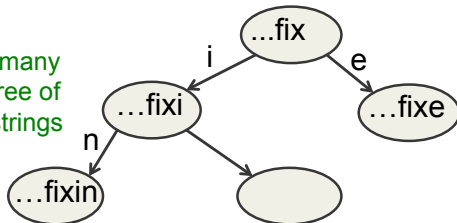
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- 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 simple model tried to capture this but is kind of indirect

A slight tweak: 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 .



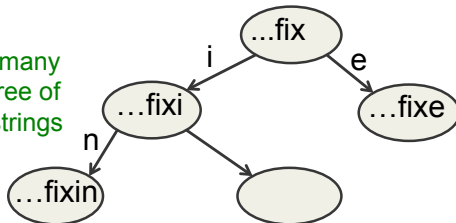
In an RNN, each node is a hidden state vector. The next character must transform this to a new node.

- 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 simple model tried to capture this but is kind of indirect

A slight tweak: 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 simple model tried to capture this but is kind of indirect

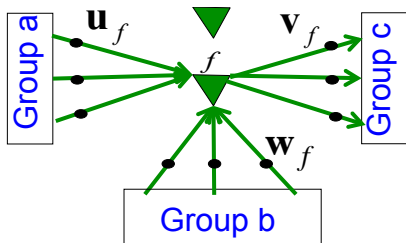
Multiplicative connections

- We may prepare a different transition matrix for each input
 - But this requires $86 \times 1500 \times 1500$ parameters (let say we have 1500 hidden variables)
 - And 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

- We may prepare a different transition matrix for each input
 - But this requires $86 \times 1500 \times 1500$ parameters (let say we have 1500 hidden variables)
 - And this could make the net overfit
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
 - We want a different transition matrix for each of the 86 characters, but we want these 86 character-specific weight matrices to share parameters (the characters 9 and 8 should have similar matrices)

Using factors to implement multiplicative interactions

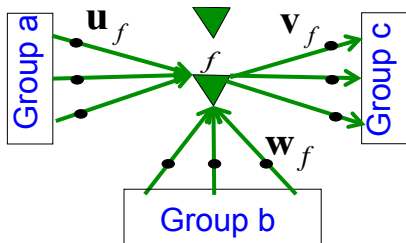


Vector input to group c :

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

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

Using factors to implement multiplicative interactions

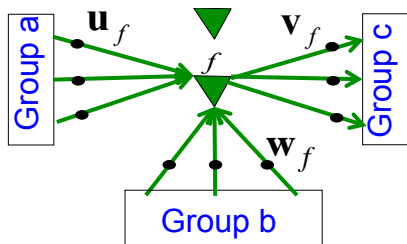


Vector input to group c :

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

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

Using factors to implement multiplicative interactions



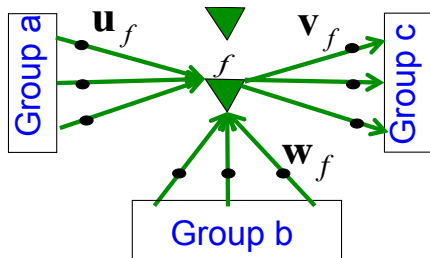
Vector input to group c :

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

input from group b input from group a

- 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

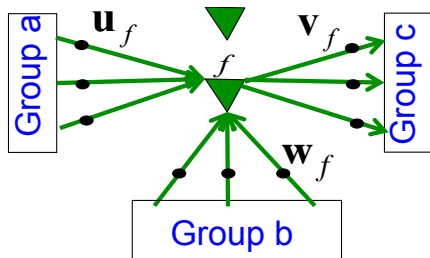


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- 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

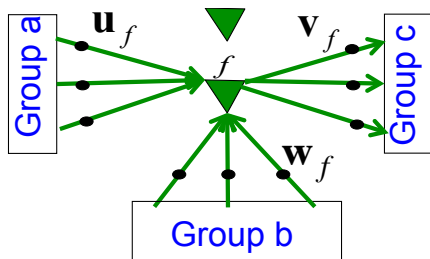


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- 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

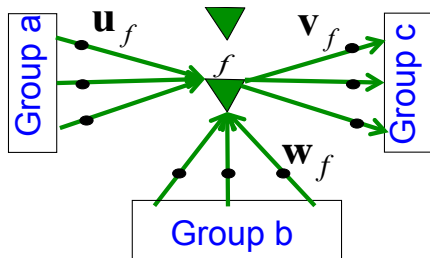


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f)v_f \\
 &= (b^T w_f)v_f(u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- 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

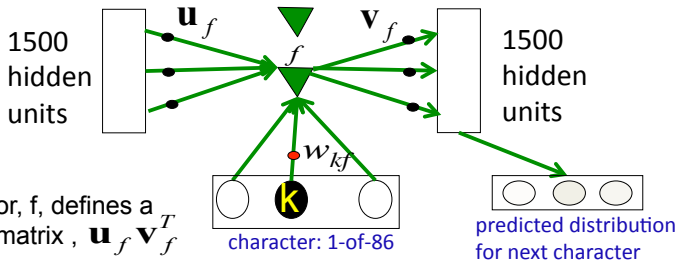


$$\begin{aligned}
 c_f &= (b^T w_f)(a^T u_f) v_f \\
 &= (b^T w_f) v_f (u_f^T a) \\
 &= \underbrace{(b^T w_f)}_{\text{scalar coefficient}} \underbrace{(v_f u_f^T)}_{\text{outer product transition matrix with rank 1}} a
 \end{aligned}$$

- 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 minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

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 minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

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 minimize the error using conjugate gradient method. Then they make another approximation to the curvature matrix and minimize again

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

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

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

Training the model

- Ilya Sutskever used 5 million strings of 100 characters taken from wikipedia. For each string he starts predicting at the 11th character
- Using the HF optimizer, it took a month on a GPU board to get a really good model (back in 2011) text

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

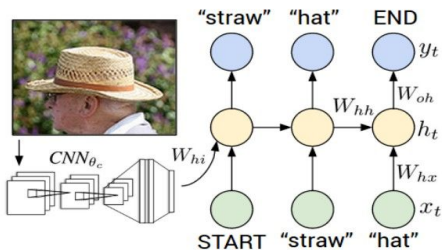
Result: what does it know?

- 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.

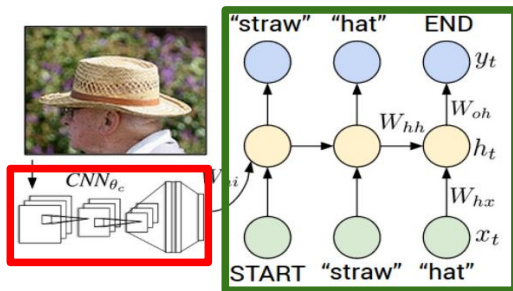
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

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



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

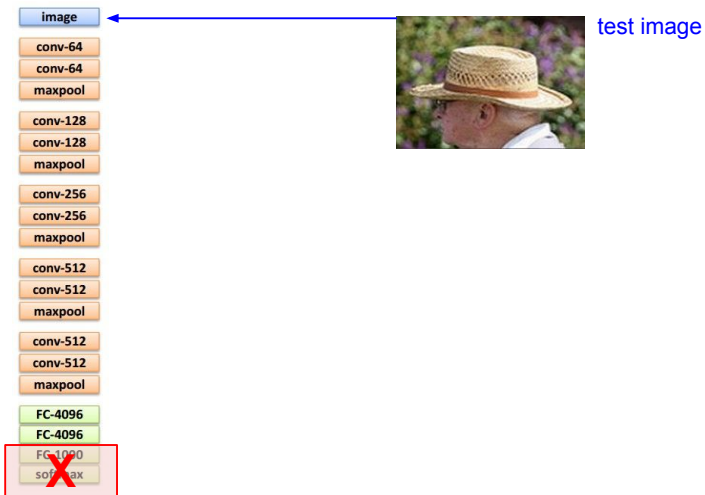
FC-4096

FC-1000

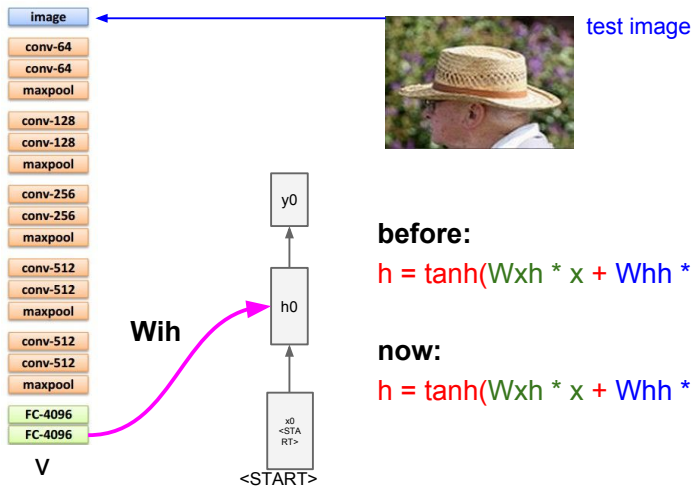
softmax

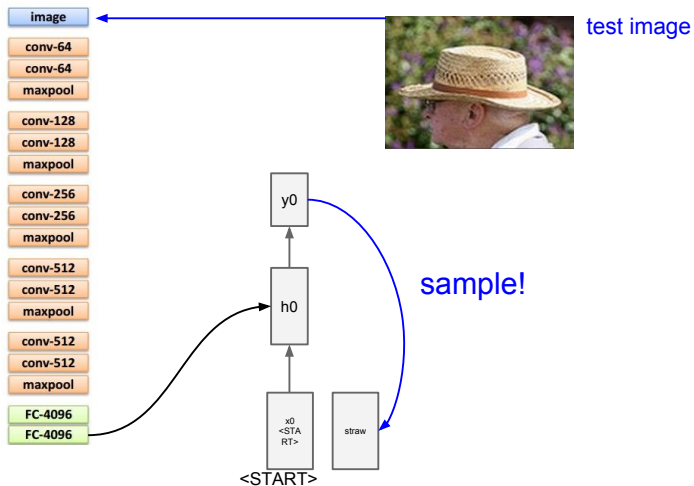


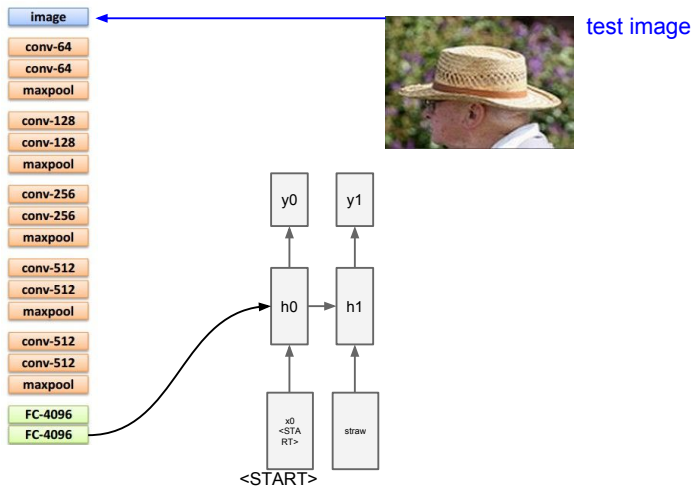
test image

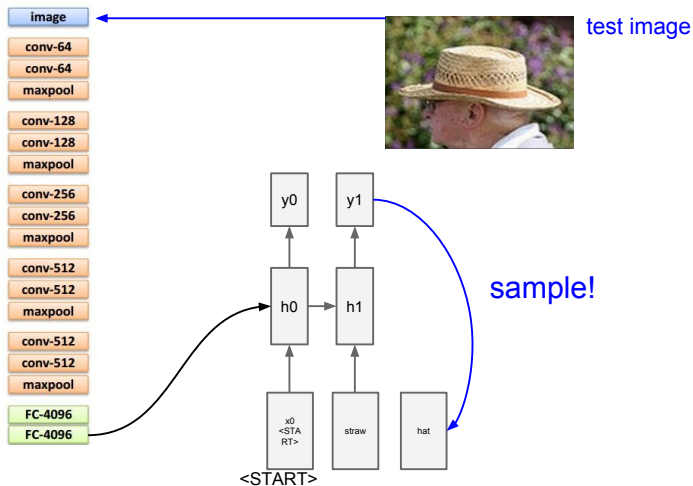


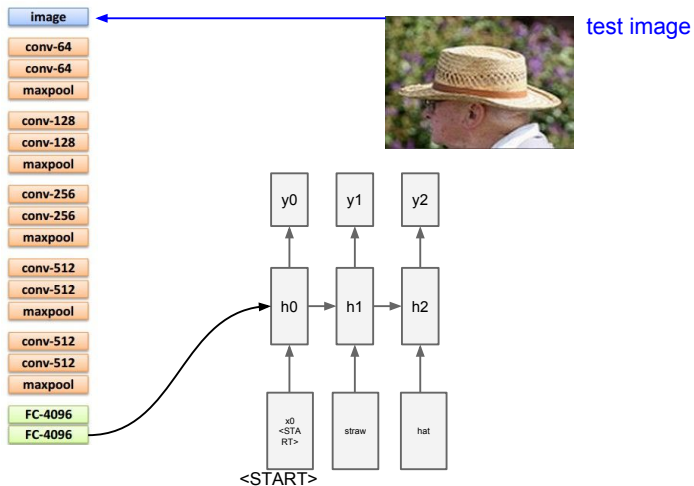












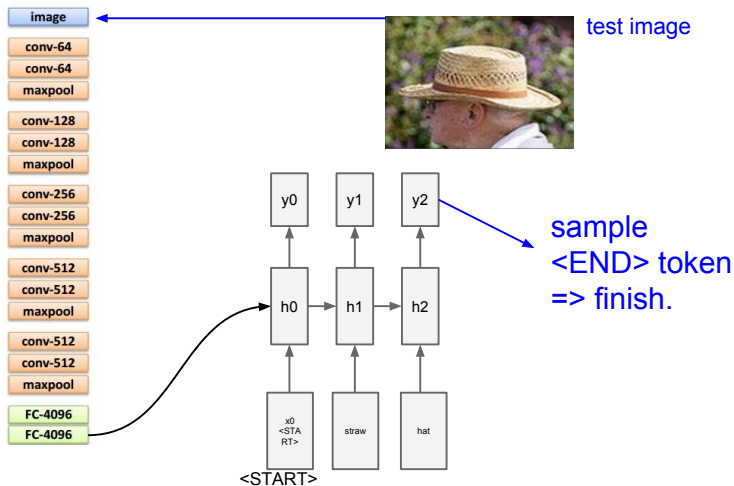


Image Sentence Datasets

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



Microsoft COCO

[*Tsung-Yi Lin et al. 2014*]

mscoco.org

currently:

~120K images

~5 sentences each



"man in black shirt is playing guitar."



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



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"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."



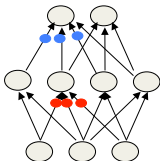
"a woman holding a teddy bear in front of a mirror."



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

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 intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - 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 input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - 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 input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - 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 input→hidden weights and sparsenesses
 - This is often necessary

How to set random connections in echo state networks

- Set the hidden→hidden weights so that the intensity of activity stays about the same after each iteration
 - Set the largest eigenvalue to 1
 - 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 input→hidden weights and sparsenesses
 - This is often necessary

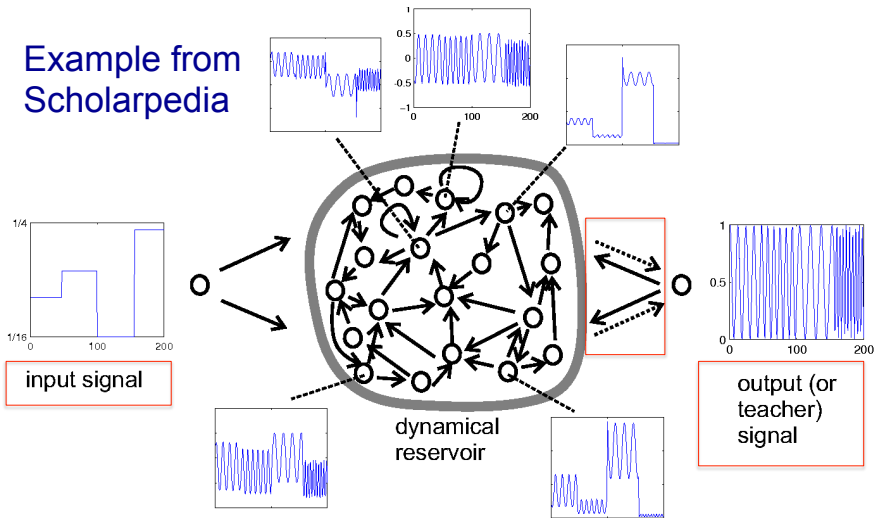
A simple example of an echo state network

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

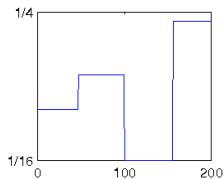
TARGET OUTPUT SEQUENCE A sine wave with the currently specified frequency

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

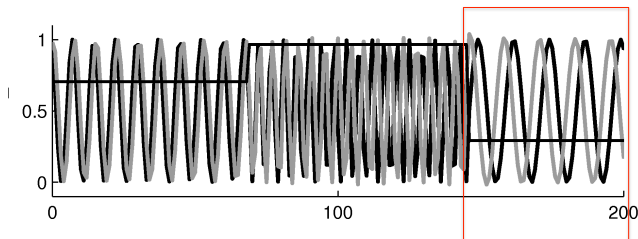
Example from Scholarpedia



The target and predicted outputs after learning



input signal



Beyond echo state networks

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

Beyond echo state networks

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

Beyond echo state networks

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

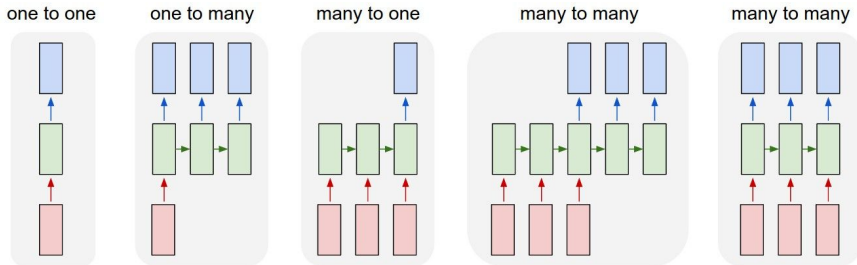
Beyond echo state networks

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

Conclusions

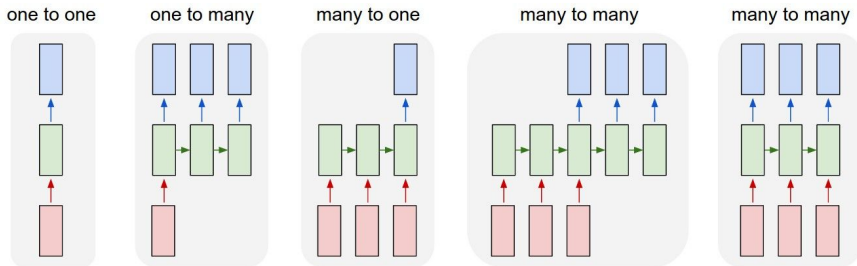
- RNNs allow a lot of flexibility in architecture design and have many applications

Recurrent Networks offer a lot of flexibility:



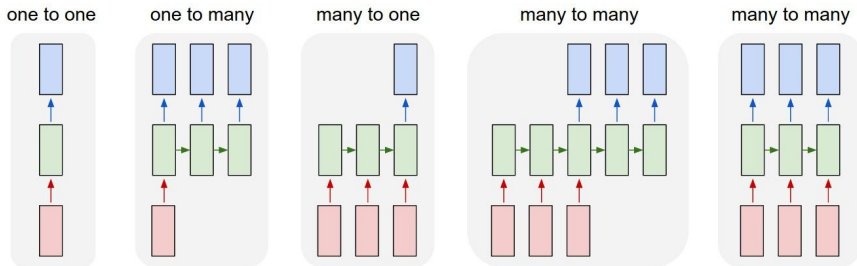
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



e.g. **Image Captioning**
image → sequence of words

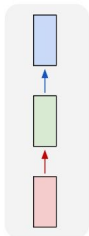
Recurrent Networks offer a lot of flexibility:



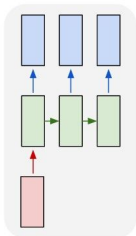
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

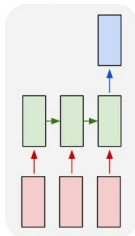
one to one



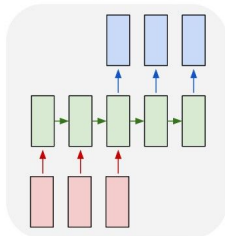
one to many



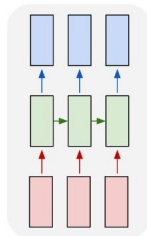
many to one



many to many



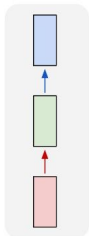
many to many



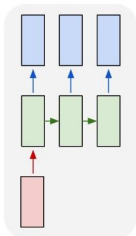
e.g. **Machine Translation**
seq of words \rightarrow seq of words

Recurrent Networks offer a lot of flexibility:

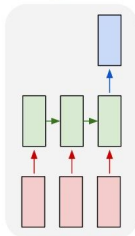
one to one



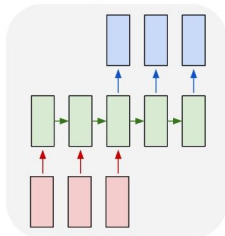
one to many



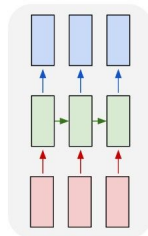
many to one



many to many



many to many



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

Conclusions

- RNNs allow a lot of flexibility in architecture design and have many applications
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better 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 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 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 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 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