CS224N Midterm Review

Nishith Khandwala, Barak Oshri, Lisa Wang, Juhi Naik

Announcements

- HW2 due today!
  - We have fixed the AFS space problem!
  - Updated submission instructions: don’t include the trained weights for your model.
  - Sorry for the inconvenience :(

- Project proposal due today!
  - You may submit a proposal even if you don’t have a mentor yet.

Midterm

- Feb 14, 4:30-5:50, Memorial Auditorium
- Alternate exam: Feb 13, 4:30-5:50, 260-113
- One cheatsheet allowed (letter sized, double-sided)
- Covers all the lectures so far
- Approximate question breakdown:
  - ⅓ multiple choice and true false
  - ⅓ short answer,
  - ⅓ more involved questions
- SCPD: Either turm up or have an exam monitor pre-registered with SCPD!!

Review Outline

- Word Vector Representations
- Neural Networks
- Backpropagation / Gradient Calculation
- RNNs
- Dependency Parsing

Word Vector Representations

CS224N Midterm Review
Nishith Khandwala

Word Vectors

***Definition*:** A vector (also referred to as an embedding) that captures the meaning of a word.

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

We will now review word2vec and GloVe.

Word2Vec

***Task*:** Learn word vectors to encode the probability of a word given its context.

Consider the following example with context window size = 2:

"I understand the word2vec model now"

Context words | Center word | Context words

Word2Vec

Task: Learn word vectors to encode the probability of a word given its context.

For each word, we want to learn 2 vectors:

- **v** : **input vector**
- **u** : **output vector**

We will see how **u** and **v** are used for the word2vec model in a bit…

Word2Vec

Task: Learn word vectors to encode the probability of a word given its context.

Two algorithms:

- **Skipgram:** predicts the probability of context words from a center word.
- **Continuous Bag-of-Words (CBOW):** predicts a center word from the surrounding context in terms of word vectors.

Word2Vec: Skipgram

- Predicts the probability of context words from a center word.
- Let’s look at the previous example again:

"I understand the word2vec model now”

Context words   Center word   Context words

Word2Vec: Skipgram

- Predicts the probability of context words from a center word.
- Let’s look at the previous example again:

"I ? ? word2vec ? ? ”

Context words | Center word | Context words

Word2Vec: Skipgram

"I ___?___ ___?___ word2vec ___?___ ___?___"

- Generate a one-hot vector, $\mathbf{w_c}$ of the center word, "word2vec". It is a |VocabSize|-dim vector with a 1 at the word index and 0 elsewhere.
- Look up the input vector, $\mathbf{v_c}$ in $\mathbf{V}$ using $\mathbf{w_c}$. $\mathbf{V}$ is the input vector matrix.
- Generate a score vector, $\mathbf{z = Uv_c}$ where $\mathbf{U}$ is the output vector matrix.

continued...

Word2Vec: Skipgram

"I ___?___ ___?___ word2vec ___?___ ___?___ "

- Turn the score vector into probabilities, $\hat{y}$ = **softmax(z)**.
- $[\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}]$: probabilities of observing each context word (**m** is the context window size)
- Minimize cost given by: (**F** can be neg-sample or softmax-CE)

$$J_{\text{skip-gram}}(\text{word}_{c-m \ldots c+m}) = \sum_{-m \leq j \leq m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)$$

Word2Vec: Continuous Bag-of-Words (CBOW)

- Predicts a center word from the surrounding context in terms of word vectors.
- Let’s look at the previous example again:

"I understand the word2vec model now”

Context words | Center word | Context words

Word2Vec: Continuous Bag-of-Words (CBOW)

- Predicts a center word from the surrounding context in terms of word vectors.
- Let's look at the previous example again:

"I understand the ? model now"

Context words | Center word | Context words

Word2Vec: Continuous Bag-of-Words (CBOW)

"I understand the ? model now"

- Generate one-hot vectors, $\mathbf{w}_{c-m}, \ldots, \mathbf{w}_{c-1}, \mathbf{w}_{c+1}, \ldots, \mathbf{w}_{c+m}$ for the context words.
- Look up the input vectors, $\mathbf{v}_{c-m}, \ldots, \mathbf{v}_{c-1}, \mathbf{v}_{c+1}, \ldots, \mathbf{v}_{c+m}$ in $\mathbf{V}$ using the one-hot vectors. $\mathbf{V}$ is the input vector matrix.
- Average these vectors to get $\mathbf{v}_{avg} = (\mathbf{v}_{c-m} + \ldots + \mathbf{v}_{c-1} + \mathbf{v}_{c+1} + \ldots + \mathbf{v}_{c+m})/2m$

continued...

Word2Vec: Continuous Bag-of-Words (CBOW)

"I understand the ? model now"

- Generate a score vector, **z = Uv$_{avg}$** where **U** is the output vector matrix.
- Turn the score vector into probabilities, **ŷ = softmax(z)**.
- **ŷ**: probability of the center word.
- Minimize cost given by: (**F** can be neg-sample or softmax-CE)

$$J_{CBOW}(word_{c-m...c+m}) = F(w_c, v_{avg})$$

GloVe

- Like Word2Vec, GloVe is a set of vectors that capture the semantic information (i.e. meaning) about words.
- Unlike Word2Vec, Glove makes use of global co-occurrence statistics.

“GloVe consists of a weighted least squares model that trains on global word-word co-occurrence counts.”

GloVe

Co-occurrence Matrix (window-based):

Corpus:

- I like Deep Learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

GloVe

- Let **X** be the word-word co-occurrence counts matrix.
  - $\mathbf{X_i}$ is the number of times any word **k** appears in the context of word **i**.
  - $\mathbf{X_{ij}}$ is the number of times word **j** occurs in the context of word **i**.
- Like the case in Word2Vec, each word has 2 vectors, **input (v)** and **output (u)**.
- The cost function:

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

GloVe

- Let **X** be the word-word co-occurrence counts matrix.
    - $\mathbf{X_i}$ is the number of times any word **k** appears in the context of word **i**.
    - $\mathbf{X_{ij}}$ is the number of times word **j** occurs in the context of word **i**.
- Like the case in Word2Vec, each word has 2 vectors, **input (v)** and **output (u)**.
- The cost function:

Least squares!

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

GloVe

- Let **X** be the word-word co-occurrence counts matrix.
  - **X_i** is the number of times any word **k** appears in the context of word **i**.
  - **X_{ij}** is the number of times word **j** occurs in the context of word **i**.
- Like the case in Word2Vec, each word has 2 vectors, **input (v)** and **output (u)**.
- The cost function:

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

Iterating over every pair of words in X!

GloVe

- Let **X** be the word-word co-occurrence counts matrix.
  - $\mathbf{X_i}$ is the number of times any word **k** appears in the context of word **i**.
  - $\mathbf{X_{ij}}$ is the number of times word **j** occurs in the context of word **i**.
- Like the case in Word2Vec, each word has 2 vectors, **input (v)** and **output (u)**.
- The cost function:

Need log since X_{ij} can be very large!

$$\hat{J} = \sum_{i=1}^{W} \sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

GloVe

In the end, we have **V** and **U** from all the input and output vectors, **v** and **u.**

Both capture similar co-occurrence information, and so the word vector for a word can be simply obtained by summing **u** and **v** up!

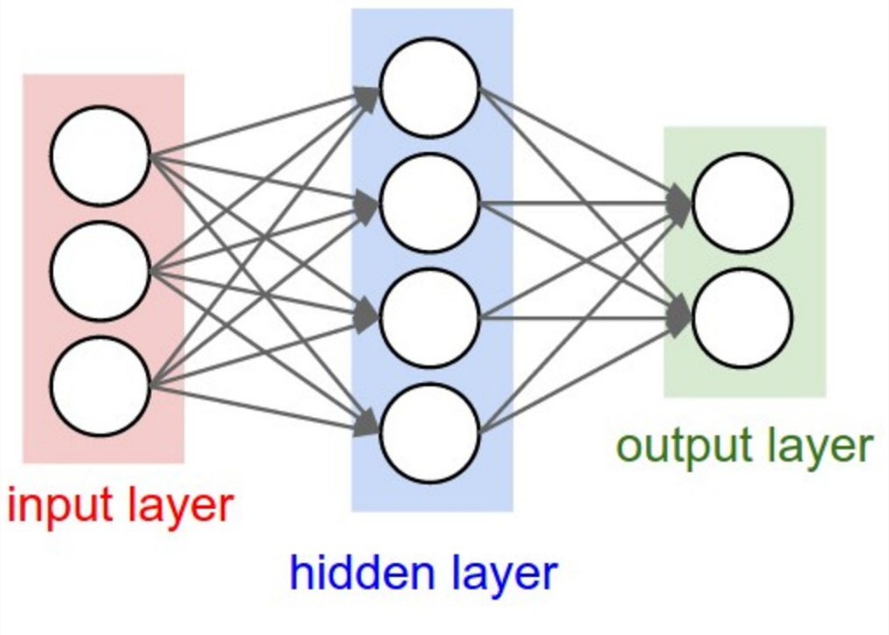
Neural Networks

CS224N Midterm Review
Nishith Khandwala

Overview

- Neural Network Basics
- Activation Functions
- Stochastic Gradient Descent (SGD)
- Regularization (Dropout)
- Training Tips and Tricks

Neural Network (NN) Basics



Dataset: **(x, y)** where **x**: inputs, **y**: labels

Steps to train a 1-hidden layer NN:

- Do a forward pass: **ŷ = f(xW + b)**
- Compute loss: **loss(y, ŷ)**
- Compute gradients using **backprop**
- Update weights using an **optimization** algorithm, like **SGD**
- Do **hyperparameter tuning** on **Dev** set
- **Evaluate** NN on **Test** set

Activation Functions: Sigmoid

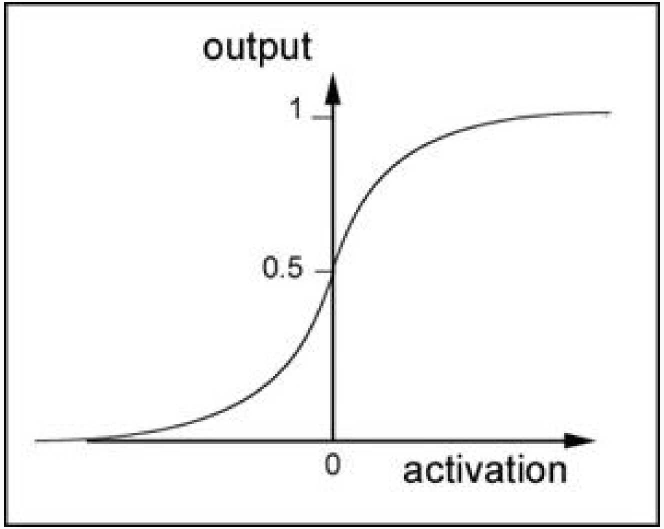
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Properties:

- Squashes input between 0 and 1.

Problems:

- Saturation of neurons kills gradients.
- Output is not centered at 0.



Activation Functions: Tanh

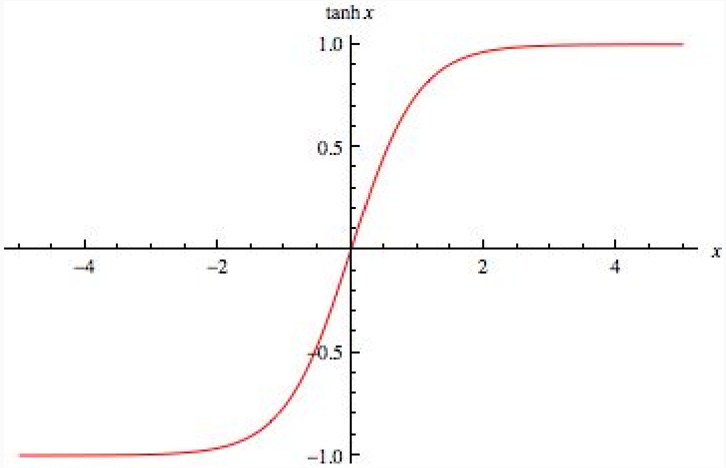
$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Properties:

- Squashes input between -1 and 1.
- Output centered at 0.

Problems:

- Saturation of neurons kills gradients.



Activation Functions: ReLU

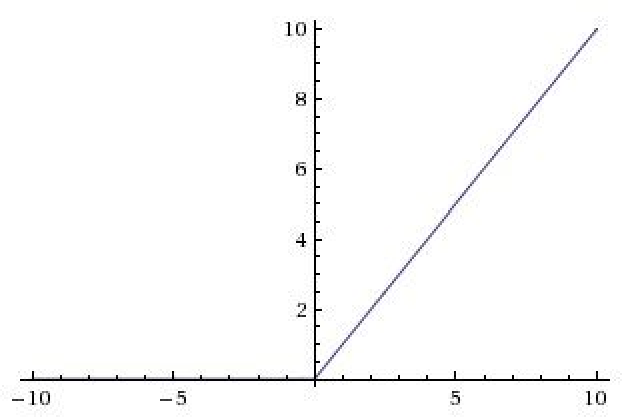
$$relu(x) = max(0, x)$$

Properties:

- No saturation
- Computationally cheap
- Empirically known to converge faster

Problems:

- Output not centered at 0
- When input < 0, ReLU gradient is 0. Never changes.



Stochastic Gradient Descent (SGD)

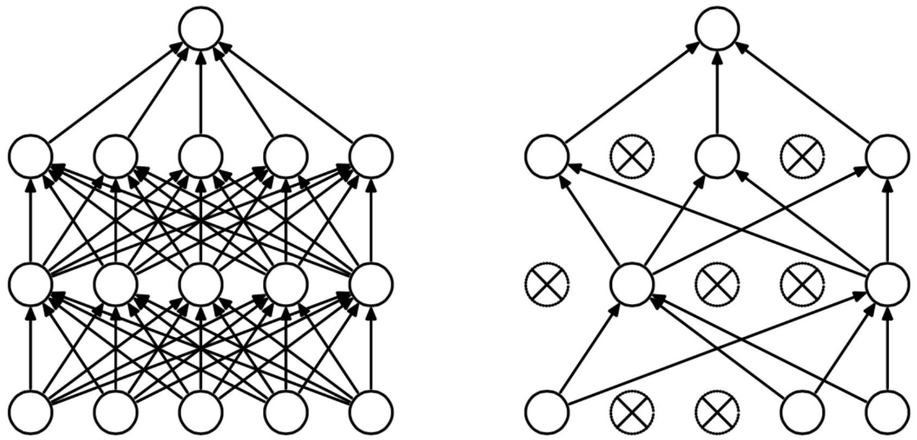
$$\theta \leftarrow \theta - \alpha \nabla_\theta J$$

- Stochastic Gradient Descent (SGD)
  - $\boldsymbol{\theta}$: weights/parameters
  - $\boldsymbol{\alpha}$: learning rate
  - J : loss function
- SGD update happens after every training example.
- Minibatch SGD (sometimes also abbreviated as SGD) considers a small batch of training examples at once, averages their loss and updates $\boldsymbol{\theta}$.

Regularization: Dropout

- **Randomly drop neurons** at forward pass during training.
- **At test time, turn dropout off.**
- **Prevents overfitting** by forcing network to learn redundancies.

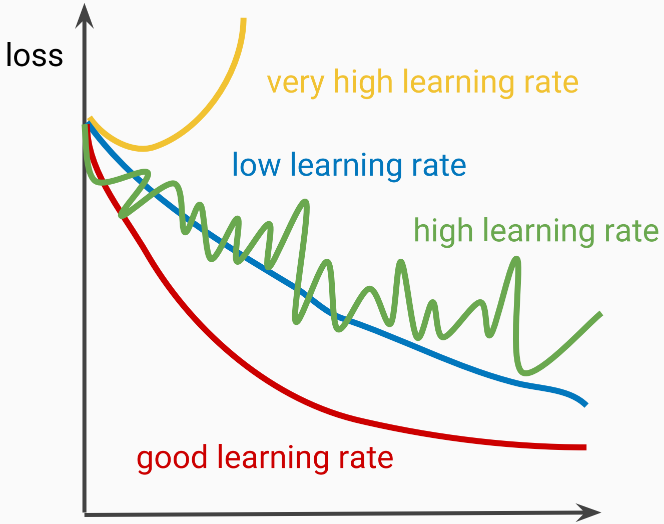
Think about dropout as **training an ensemble of networks.**



(a) Standard Neural Net (b) After applying dropout.

Training Tips and Tricks

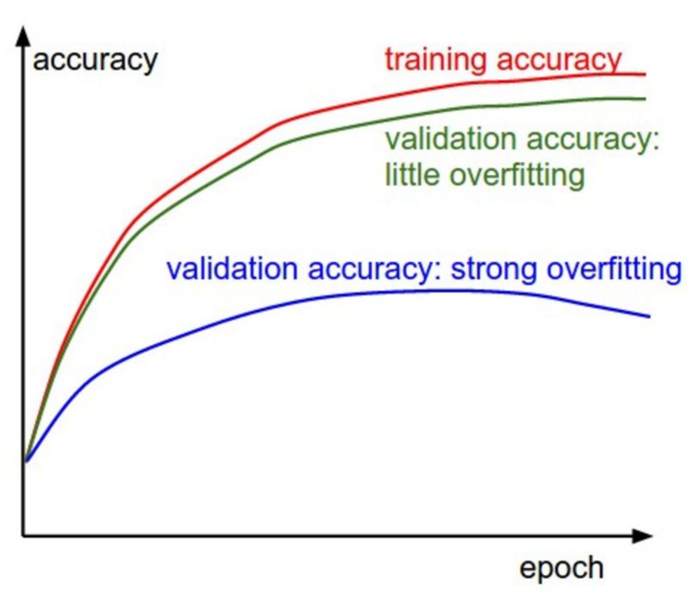
- Learning rate:
  - If loss curve seems to be unstable (jagged line), **decrease** learning rate.
  - If loss curve appears to be “linear”, **increase** learning rate.



Training Tips and Tricks

- Regularization (Dropout, L2 Norm, …):
  - If the gap between train and dev accuracies is large (overfitting), **increase** the regularization constant.

DO NOT test your model on the **test** set until overfitting is no longer an issue.



Backpropagation and Gradients

CS224N Midterm Review
Barak Oshri

Itinerary

- Backprop reviewed
- Matrix calculus primer
- Computing chain rule products correctly *(ie when do I transpose?)*
- Sample midterm problems

Problem Statement

$$Loss = f(x, y; \theta)$$

Given a function ***f*** with respect to inputs ***x***, labels ***y***, and parameters ***θ*** compute the gradient of ***Loss*** with respect to ***θ***

Backpropagation

$$Loss = CE(\sigma(\mathbf{x}W_1 + b_1)W_2 + b_2, y)$$

An algorithm for computing the gradient of a **compound** function as a series of **local, intermediate gradients**

Backpropagation

$$Loss = CE(\sigma(\mathbf{x}W_1 + b_1)W_2 + b_2, y)$$

1. Identify intermediate functions (forward prop)
2. Compute local gradients
3. Combine with downstream error signal to get full gradient

Modularity - Simple Example

Compound function

$$f(x, y, z) = (x + y)z$$

Intermediate Variables
(forward propagation)

$$q = x + y$$

$$f = qz$$

Modularity - Neural Network Example

Compound function

$$Loss = CE(\sigma(\mathbf{x}W_1 + b_1)W_2 + b_2, y)$$

Intermediate Variables

(forward propagation)

$$h = \mathbf{x}W_1 + b_1$$

$$z_1 = \sigma(h)$$

$$z_2 = z_1 W_2 + b_2$$

$$Loss = CE(z_2, y)$$

Intermediate **Variables**

(forward propagation)

$$h = \mathbf{x}W_1 + b_1$$

$$z_1 = \sigma(h)$$

$$z_2 = z_1 W_2 + b_2$$

$$Loss = CE(z_2, y)$$

Intermediate **Gradients**

(backward propagation)

$$\frac{\partial h}{\partial \mathbf{x}} = W_1^\top$$

$$\frac{\partial z_1}{\partial h} = \sigma'(h) = z_1 \circ (1 - z_1)$$

$$\frac{\partial z_2}{\partial z_1} = W_2^\top$$

$$\frac{\partial Loss}{\partial z_2} = Softmax(z_2) - y$$

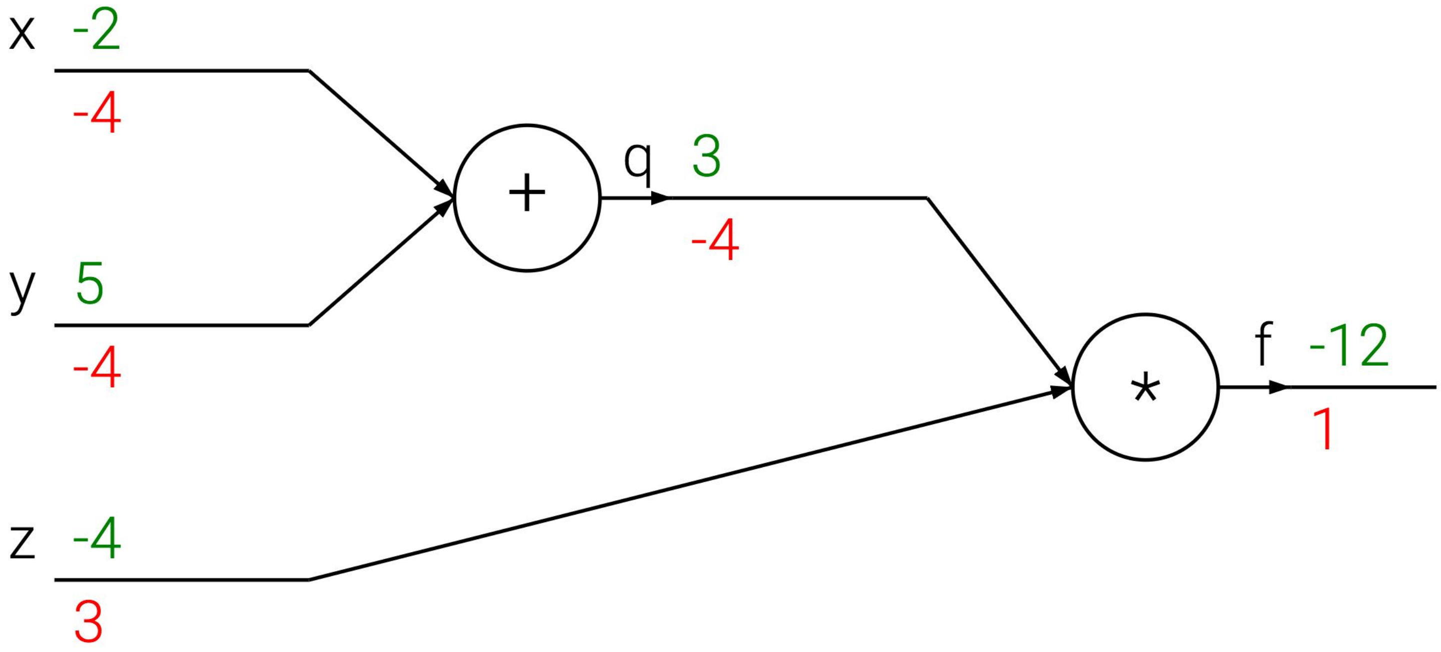
Chain Rule Behavior

$$\frac{d((f \circ g)(x))}{dx} = \frac{d(f(g(x)))}{d(g(x))} \frac{d(g(x))}{dx}$$

Key chain rule intuition:

Slopes multiply

Circuit Intuition



Matrix Calculus Primer

Scalar-by-Vector

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} \cdots \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Vector-by-Vector

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Matrix Calculus Primer

Scalar-by-Matrix

$$\frac{\partial y}{\partial A} = \begin{bmatrix} \frac{\partial y}{\partial A_{11}} & \frac{\partial y}{\partial A_{12}} & \cdots & \frac{\partial y}{\partial A_{1n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial A_{m1}} & \frac{\partial y}{\partial A_{m2}} & \cdots & \frac{\partial y}{\partial A_{mn}} \end{bmatrix}$$

Vector-by-Matrix

$$\frac{\partial y}{\partial A_{ij}} = \frac{\partial y}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial A_{ij}}$$

Vector-by-Matrix Gradients

$$\frac{\partial J}{\partial A_{ij}} = \frac{\partial J}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial A_{ij}}$$

Let $\mathbf{z} = A\mathbf{x}$

$$\frac{\partial \mathbf{z}}{\partial A_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ \\ \leftarrow i\text{'th element} \\ \\ \\ \\ \end{array}$$

$$\frac{\partial J}{\partial A_{ij}} = \delta_i \mathbf{x}_j$$

$$\Rightarrow \frac{\partial J}{\partial A} = \delta^\top \mathbf{x}$$

$$\boxed{\begin{aligned} z &= Wx \\ \frac{\partial z}{\partial x} &= W \end{aligned}}$$

$$\boxed{\begin{aligned} z &= x \\ \frac{\partial z}{\partial x} &= I \end{aligned}}$$

$$\boxed{\begin{aligned} z &= xW \\ \frac{\partial z}{\partial x} &= W^\top \end{aligned}}$$

$$\boxed{\begin{gathered} z = Wx \quad \delta = \frac{\partial J}{\partial z} \\ \frac{\partial J}{\partial W} = \delta^\top x \end{gathered}}$$

$$\boxed{\begin{gathered} z = xW \quad \delta = \frac{\partial J}{\partial z} \\ \frac{\partial J}{\partial W} = x^\top \delta \end{gathered}}$$

Backpropagation Shape Rule

When you take gradients against a **scalar**

↓

The gradient at each intermediate step has **shape of denominator**

$$X \in \mathbb{R}^{m \times n} \iff \delta_X = \frac{\delta Scalar}{\delta X} \in \mathbb{R}^{m \times n}$$

Dimension Balancing

W	$[n \times w]$	$\frac{\partial Loss}{\partial W} = \quad ?$
X	$[m \times n]$	$\frac{\partial Loss}{\partial X} = \quad ?$
$Z = XW$	$[m \times w]$	$\frac{\partial Loss}{\partial Z} = \delta$

Dimension Balancing

W	$[n \times w]$	$\frac{\partial Loss}{\partial W} = X^\top \delta$
X	$[m \times n]$	$\frac{\partial Loss}{\partial X} = \delta W^T$
$Z = XW$	$[m \times w]$	$\frac{\partial Loss}{\partial Z} = \delta$

Dimension Balancing

Dimension balancing is the “cheap” but **efficient** approach to gradient calculations in most practical settings

Read *gradient computation notes* to understand how to derive matrix expressions for gradients from **first principles**

Activation Function Gradients

$z = \sigma(h)$ is an element-wise function on each index of ***h*** (scalar-to-scalar)

Officially, $\dfrac{\partial z}{\partial h} = \begin{bmatrix} z_1(1-z_1) & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & z_n(1-z_n) \end{bmatrix}$

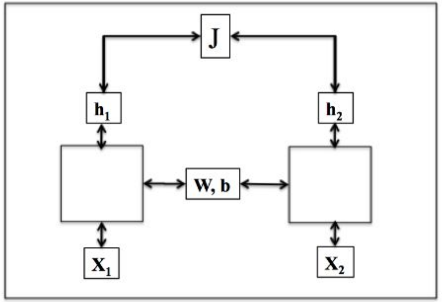
Diagonal matrix represents that z_i and h_j have no dependence if $i \neq j$

Activation Function Gradients

Element-wise multiplication (hadamard product) corresponds to **matrix product with a diagonal matrix**

$$\frac{\partial Loss}{\partial h} = \frac{\partial Loss}{\partial z} \begin{bmatrix} z_1(1-z_1) & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & z_n(1-z_n) \end{bmatrix}$$

$$= \frac{\partial Loss}{\partial z} \circ (z \circ (1-z))$$



Here is one such model to evaluate how similar two input words are using Euclidean distance. There are two input word vectors $x_1, x_2 \in \mathbb{R}^n$, shared parameters $W \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, and a single hidden layer associated with *each* input:

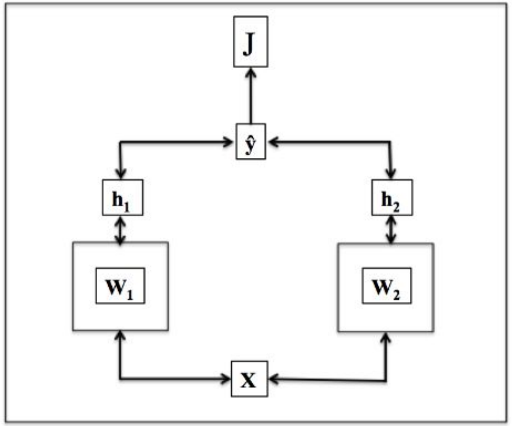
$$h_1 = \sigma(Wx_1 + b)$$

$$h_2 = \sigma(Wx_2 + b)$$

We evaluate the distance between the two activations h_1, h_2 using Euclidean distance as our similarity metric. The model objective J is

$$J = \frac{1}{2}\|h_1 - h_2\|_F^2 + \frac{\lambda}{2}\|W\|_F^2$$

where λ is a given regularization parameter. (The Frobenius norm $\|.\|_F$ is a matrix norm defined by $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j}|A_{ij}|^2}$)



Our model is:

$$h_1 = \sigma(W_1 x + b_1)$$

$$h_2 = \mathrm{relu}(W_2 x + b_2)$$

$$\hat{y} = \mathrm{softmax}(W_3(h_1 + h_2) + b_3)$$

where $x \in \mathbb{R}^n$, W_1, $W_2 \in \mathbb{R}^{m\times n}$, $W_3 \in \mathbb{R}^{k\times m}$, b_1,$b_2 \in \mathbb{R}^m$, and $b_3 \in \mathbb{R}^k$. We evaluate this model for N examples and k classes with cross entropy loss

$$J = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{k} y_j^i log(\hat{y}_j^i)$$

$$
\begin{aligned}
h_1 &= \sigma(xW_1 + b_1) \\
h_2 &= \mathrm{relu}(xW_2 + b_2) \\
\hat{y} &= \mathrm{softmax}((h_1 + h_2)W_3 + b_3) \\
J &= -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{k} y_j^i \log(\hat{y}_j^i)
\end{aligned}
$$

$$
x \in \mathbb{R}^n, W_1, W_2 \in \mathbb{R}^{n\times m}, W_3 \in \mathbb{R}^{m\times k}
$$
$$
b_1, b_2 \in \mathbb{R}^m, b_3 \in \mathbb{R}^k
$$

Backprop Menu for Success

1. Write down variable graph

2. Compute derivative of cost function

3. Keep track of error signals

4. Enforce shape rule on error signals

5. Use matrix balancing when deriving over a linear transformation

RNNs, Language Models, LSTMs, GRUs

CS224N Midterm Review

Lisa Wang, Juhi Naik

RNNs

- Review of RNNs
- RNN Language Models
- Vanishing Gradient Problem
- GRUs
- LSTMs

Midterm Question: Applied RNNs

4) (2 points) You now have a distributed representation of each patient note (note-vector). You assume that a patient's past medical history is informative of their current illness. As such, you apply a recurrent neural network to predict the current illness based on the patient's current and previous note-vectors. Explain why a recurrent neural network would yield better results than a feed-forward network in which your input is the summation (or average) of past and current note-vectors?

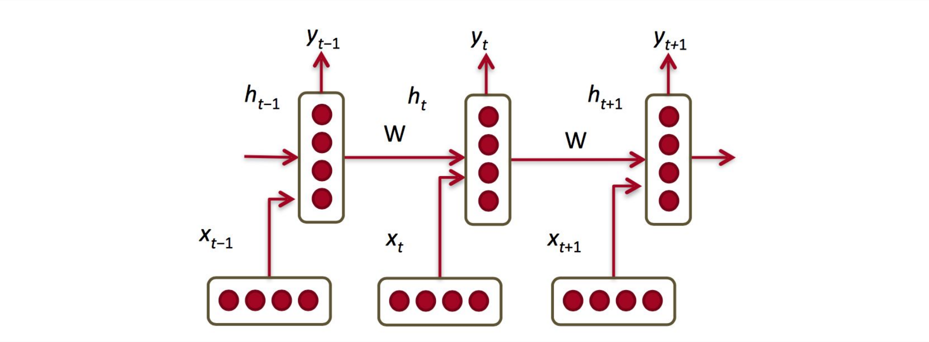
From CS224D Midterm Spring 2016, Problem 3.4

Midterm Question: Applied RNNs

4) (2 points) You now have a distributed representation of each patient note (note-vector). You assume that a patient's past medical history is informative of their current illness. As such, you apply a recurrent neural network to predict the current illness based on the patient's current and previous note-vectors. Explain why a recurrent neural network would yield better results than a feed-forward network in which your input is the summation (or average) of past and current note-vectors?

RNNs allows you to capture temporal relationships (e.g. sequence of events). You would lose that information by summing your note-vectors.

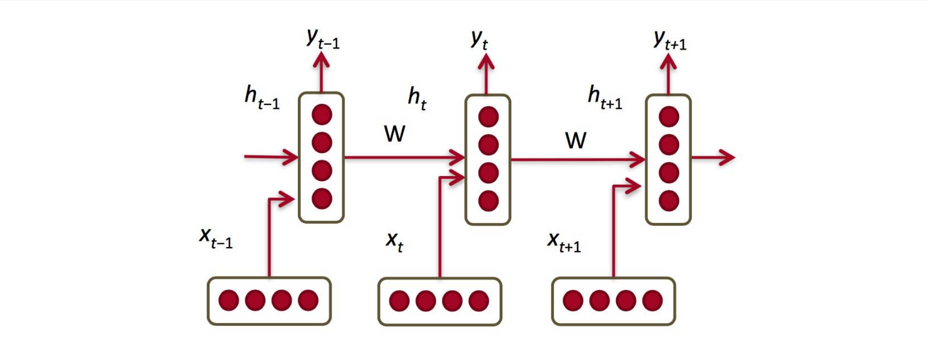
RNN Review



Key points:

- Weights are shared (tied) across timesteps (W_{xh}, W_{hh}, W_{hy})
- Hidden state at time *t* depends on previous hidden state and new input
- Backpropagation across timesteps (use unrolled network)

RNN Review



RNNs are good for:

- Learning representations for sequential data with temporal relationships
- Predictions can be made at every timestep, or at the end of a sequence

RNN Language Model

- Language Modeling (LM): task of computing probability distributions over sequence of words $P(w_1, \ldots, w_T)$
- Important role in speech recognition, text summarization, etc.
- RNN Language Model:

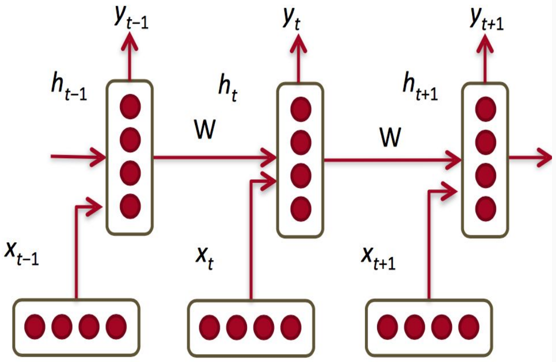
Given list of word vectors: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

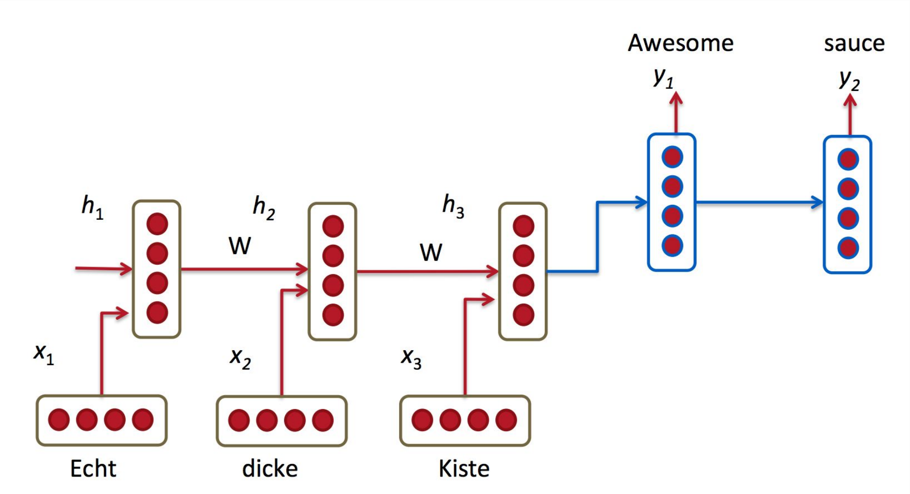
$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$



RNN Language Model for Machine Translation



- Encoder for source language
- Decoder for target language
- Different weights in encoder and decoder sections of the RNN (Could see them as two chained RNNs)

Vanishing Gradient Problem

- Backprop in RNNs: recursive gradient call for hidden layer
- Magnitude of gradients of typical activation functions between 0 and 1.

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right\|$$

- When terms less than 1, product can get small very quickly
- Vanishing gradients → RNNs fail to learn, since parameters barely update.
- GRUs and LSTMs to the rescue!

Gated Recurrent Units (GRUs)

- Reset gate, r_t
- Update gate, z_t
- r_t and z_t control long-term and short-term dependencies (mitigates vanishing gradients problem)

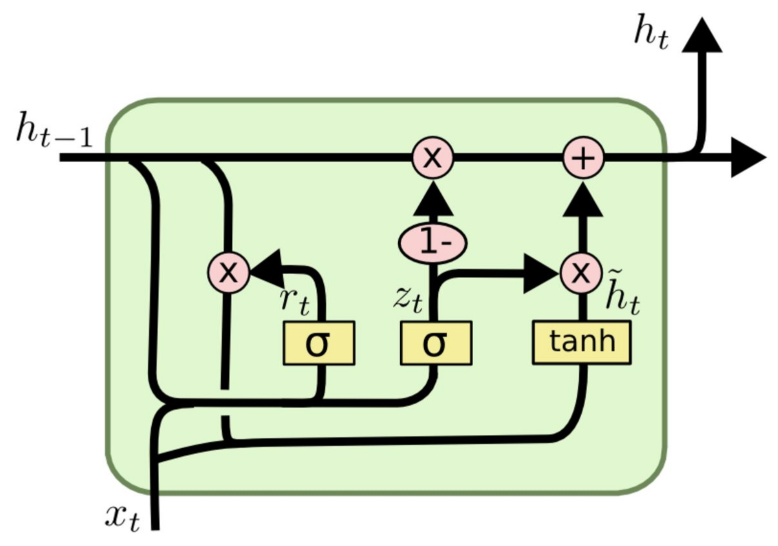
$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

Gated Recurrent Units (GRUs)



$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\
\tilde{h}_t &= \tanh(W x_t + r_t \circ U h_{t-1}) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t
\end{aligned}
$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Midterm Questions: GRUs

What are the dimensions of the W's and U's? (six matrices)

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

Midterm Questions: GRUs

What are the dimensions of the W's and U's? (six matrices)

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$
$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

W_z , W_r , W : d_h x d_x

U_z , U_r , U : d_h x d_h

Midterm Questions: GRUs

True/False. If the update gate z_t is close to 0, the net does not update its state significantly.

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$
$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

Midterm Questions: GRUs

True/False. If the update gate z_t is close to 0, the net does not update its state significantly.

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

True. In this case, $h_t \approx h_{t-1}$

Midterm Questions: GRUs

True/False. If the update gate z_t is close to 1 and the reset gate r_t is close to 0, the net remembers the past state very well.

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

Midterm Questions: GRUs

True/False. If the update gate z_t is close to 1 and the reset gate r_t is close to 0, the net remembers the past state very well.

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W x_t + r_t \circ U h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

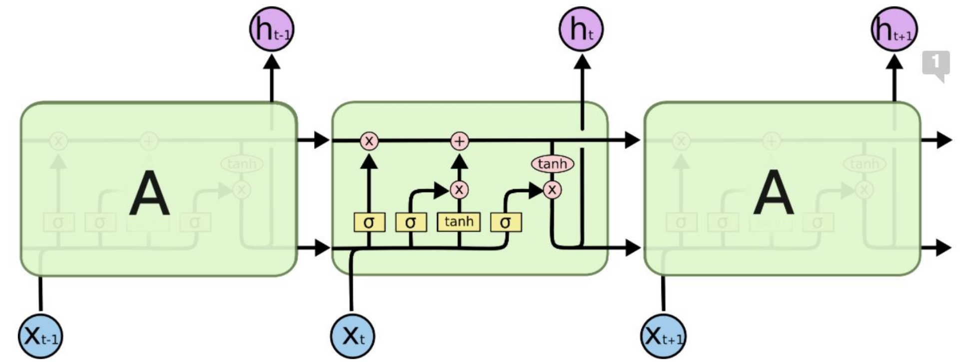
False. In this case, h_t depends strongly on input x_t and not on h_t-1.

LSTMs

- i_t: Input gate - How much does current input matter
- f_t: Input gate - How much does past matter
- o_t: Output gate - How much should current cell be exposed
- c_t: New memory - Memory from current cell

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) \\
\widetilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned}
$$

LSTMs



The repeating module in an LSTM contains four interacting layers.

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$i_t = \sigma\left(W^{(i)} x_t + U^{(i)} h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1}\right)$$
$$\tilde{c}_t = \tanh\left(W^{(c)} x_t + U^{(c)} h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

Midterm Questions: LSTMs

True/False. If x_t is the 0 vector, then $h_t = h_{t-1}$

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. If x_t is the 0 vector, then $h_t = h_{t-1}$

False. Due to the transformations by Us and non-linearities, they are generally unequal

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. If f_t is very small or zero, then error will not be back-propagated to earlier time steps

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. If f_t is very small or zero, then error will not be back-propagated to earlier time steps

False. i_t and $\tilde{c}_t$ depend on h_{t-1}

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. The entries of f_t, i_t and o_t are non-negative.

$$i_t = \sigma\left(W^{(i)} x_t + U^{(i)} h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)} x_t + U^{(c)} h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. The entries of f_t, i_t and o_t are non-negative.

True. The range of sigmoid is (0,1)

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. f_t, i_t and o_t can be viewed as probability distributions (entries sum to 1 and each entry is between 0 and 1)

$$i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Midterm Questions: LSTMs

True/False. f_t, i_t and o_t can be viewed as probability distributions (entries sum to 1 and each entry is between 0 and 1)

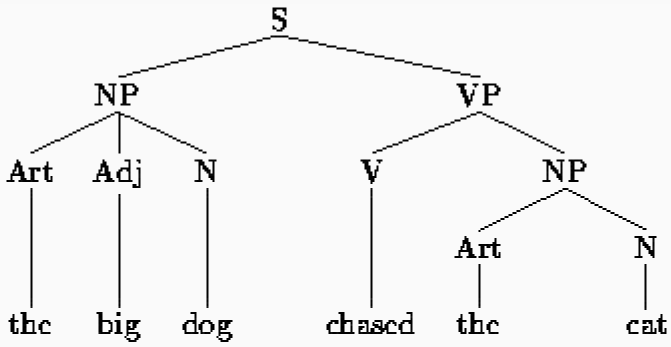
False. Sigmoid is applied independently element-wise. The sum need not be 1.

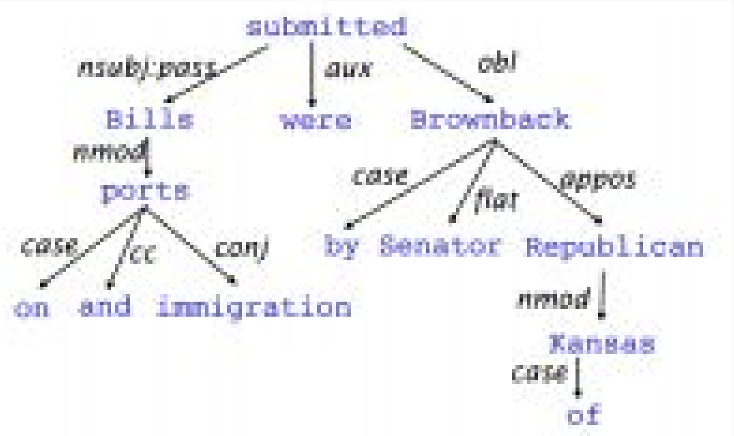
$$i_t = \sigma\left(W^{(i)} x_t + U^{(i)} h_{t-1}\right)$$
$$f_t = \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1}\right)$$
$$o_t = \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1}\right)$$
$$\widetilde{c}_t = \tanh\left(W^{(c)} x_t + U^{(c)} h_{t-1}\right)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ \tanh\left(c_t\right)$$

Dependency Parsing

CS224N Midterm Review
Juhi Naik

Two views of Linguistic Structure





Constituency Structure uses phrase structure grammar to organize words into nested constituents.

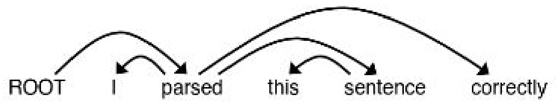
Dependency Structure uses dependency grammar to identify which words depend on which other words (and how).

Dependency Parsing

- Binary Asymmetric relations between words.
- Typed with the name of the grammatical relation.
- Arrow goes from head of the dependency to the dependent.
- Usually forms a connected, acyclic, single-head tree.

Greedy deterministic transition based parsing

- Bottom up actions analogous to shift-reduce parser
- States defined as a triple of words in buffer, words in stack and set of parsed dependencies.
- Transitions:
  - Shift
  - Left-Arc
  - Right-Arc



stack	buffer	new dependency	transition
[ROOT]	[I, parsed, this, sentence, correctly]		Initial Configuration
[ROOT, I]	[parsed, this, sentence, correctly]		SHIFT
[ROOT, I, parsed]	[this, sentence, correctly]		SHIFT
[ROOT, parsed]	[this, sentence, correctly]	parsed→I	LEFT-ARC

- Discriminative classification used to decide next transition at every step.
- Features: Top of stack word, first buffer word, POS, lookahead etc.
- Evaluation metrics: UAS (Unlabelled Attachment Score), LAS (Labelled Attachment Score)

Projectivity

Projective arcs have no crossing arcs when the words are laid in linear order.

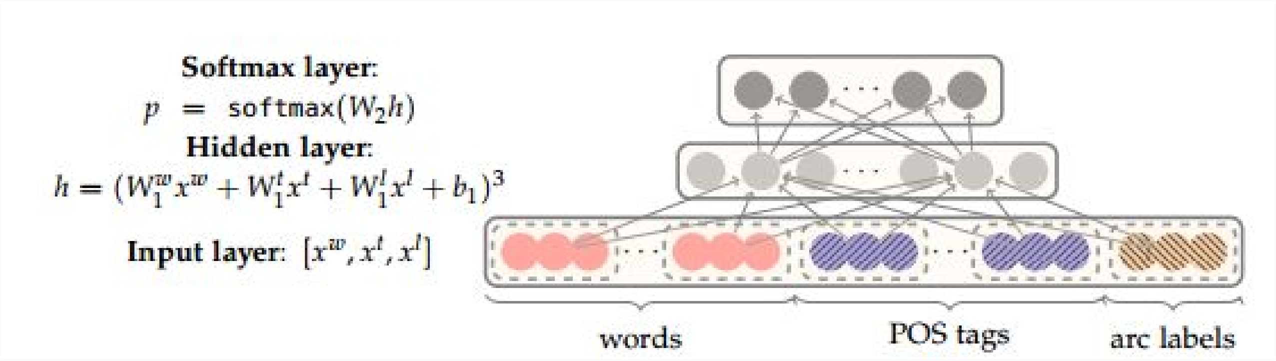
However, some sentences have non-projective dependency structure

Handling non-projectivity

- Declare defeat (when non-projectivity is rare and doesn't affect our accuracy much)
- Use post-processor to identify and resolve these non-projective dependencies
- Use a parsing mechanism that doesn't have projectivity constraint.
    - Add extra transitions to greedy transition based parsing.
    - Other algorithms …

Neural Dependency Parsing

- Instead of sparse, one-hot vector representations used in the previous methods, we use embedded vector representations for each feature.
- Features used:
    - Vector representation of first few words in buffer and stack and their dependents
    - POS tags for those words
    - Arc labels for dependents



Acknowledgements

- Andrej Karpathy, Research Scientist, OpenAI
- Christopher Olah, Research Scientist, Google Brain

Good luck on the midterm!