



Natural Language Processing

Angel Xuan Chang

angelxuanchang.github.io/nlp-class

adapted from lecture slides from Anoop Sarkar

Simon Fraser University

February 06, 2020

Natural Language Processing

Angel Xuan Chang

angelxuanchang.github.io/nlp-class

adapted from lecture slides from Anoop Sarkar

Simon Fraser University

February 06, 2020

Part 1: Feedforward neural networks

Log-linear models versus Neural networks

Feedforward neural networks

Motivating example: XOR

Stochastic Gradient Descent

Computation Graphs

Log linear model

- ▶ Let there be m features, $f_k(\mathbf{x}, y)$ for $k = 1, \dots, m$
- ▶ Define a parameter vector $\mathbf{v} \in \mathbb{R}^m$
- ▶ A log-linear model for classification into labels $y \in \mathcal{Y}$:

$$\Pr(y \mid \mathbf{x}; \mathbf{v}) = \frac{\exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

Advantages

The feature representation $\mathbf{f}(\mathbf{x}, y)$ can represent any aspect of the input that is useful for classification.

Disadvantages

The feature representation $\mathbf{f}(\mathbf{x}, y)$ has to be designed by hand which is time-consuming and error-prone.

Log linear model

Figure from [1]

Disadvantages: number of combined features can explode

farmers eat	steak →	high	cows eat	steak →	low
	hay →	low		hay →	high
farmers grow	steak →	low	cows grow	steak →	low
	hay →	high		hay →	low

Neural Networks

Advantages

- ▶ Neural networks replace hand-engineered features with **representation learning**
- ▶ Empirical results across many different domains show that learned representations give significant improvements in accuracy
- ▶ Neural networks allow end to end training for complex NLP tasks and do not have the limitations of multiple chained pipeline models

Disadvantages

For many tasks linear models are much faster to train compared to neural network models

Alternative Form of Log linear model

Log-linear model:

$$\Pr(y \mid \mathbf{x}; \mathbf{v}) = \frac{\exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{v} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

Alternative form using functions:

$$\Pr(y \mid x; v) = \frac{\exp(v(y) \cdot f(x) + \gamma_y)}{\sum_{y' \in \mathcal{Y}} \exp(v(y') \cdot f(x) + \gamma_{y'})}$$

- ▶ Feature vector $f(x)$ maps input x to \mathbb{R}^d
- ▶ Parameters $v(y) \in \mathbb{R}^d$ and $\gamma_y \in \mathbb{R}$ for each $y \in \mathcal{Y}$
- ▶ We assume $v(y) \cdot f(x)$ is a dot product. Using matrix multiplication it would be $v(y) \cdot f(x)^T$
- ▶ Let $v = \{(v(y), \gamma_y) : y \in \mathcal{Y}\}$

Log-linear models versus Neural networks

Feedforward neural networks

Motivating example: XOR

Stochastic Gradient Descent

Computation Graphs

Representation Learning: Feedforward Neural Network

Replace hand-engineered features f with learned features ϕ :

$$\Pr(y \mid x; \theta, v) = \frac{\exp(v(y) \cdot \phi(x; \theta) + \gamma_y)}{\sum_{y' \in \mathcal{Y}} \exp(v(y') \cdot \phi(x; \theta) + \gamma_{y'})}$$

- ▶ Replace $f(x)$ with $\phi(x; \theta) \in \mathbb{R}^d$ where θ are new parameters
- ▶ Parameters θ are learned from training data
- ▶ Using θ the model ϕ maps input x to \mathbb{R}^d : a learned representation from x
- ▶ $x \in \mathbb{R}^d$ is a pre-trained vector of size d
- ▶ We will use feedforward neural networks to define $\phi(x; \theta)$
- ▶ $\phi(x; \theta)$ will be a **non-linear** mapping to \mathbb{R}^d
- ▶ ϕ replaces f which was a **linear** model

A Single Neuron aka Perceptron

A single neuron maps input $x \in \mathbb{R}^d$ to output h :

$$h = g(w \cdot x + b)$$

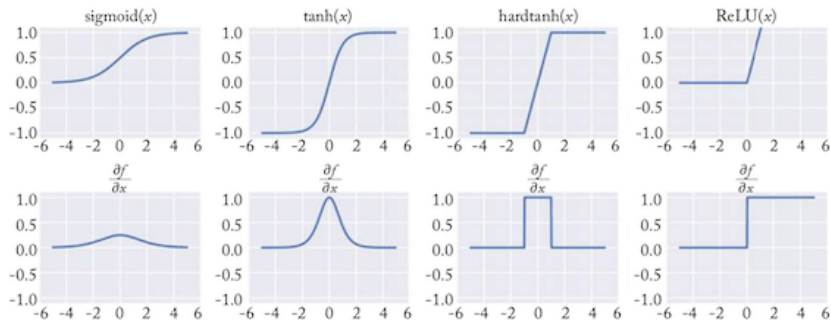
- ▶ Weight vector $w \in \mathbb{R}^d$, a bias $b \in \mathbb{R}$ are the parameters of the model learned from training data
- ▶ *Transfer function* (also called *activation function*)

$$g : \mathbb{R} \rightarrow \mathbb{R}$$

- ▶ It is important that g is a **non-linear** transfer function
- ▶ Linear $g(z) = \alpha \cdot z + \beta$ for constants α, β (linear perceptron)

Activation Functions and their Gradients

from [2], Fig. 4.3



The sigmoid Transfer Function: σ

sigmoid transfer function:

$$g(z) = \frac{1}{1 + \exp(-z)}$$

Derivative of sigmoid:

$$\frac{dg(z)}{dz} = g(z)(1 - g(z))$$

The tanh Transfer Function

tanh transfer function:

$$g(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$$

Derivative of tanh:

$$\frac{dg(z)}{dz} = 1 - g(z)^2$$

Alternatives to tanh

hardtanh:

$$g(z) = \begin{cases} 1 & \text{if } z > 1 \\ -1 & \text{if } z < -1 \\ z & \text{otherwise} \end{cases}$$

$$\frac{dg(z)}{dz} = \begin{cases} 1 & \text{if } -1 \leq z \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

softsign:

$$g(z) = \frac{z}{1 + |z|}$$

$$\frac{dg(z)}{dz} = \begin{cases} \frac{1}{(1+z)^2} & \text{if } z \geq 0 \\ \frac{-1}{(1+z)^2} & \text{if } z < 0 \end{cases}$$

The ReLU Transfer Function

Rectified Linear Unit (ReLU):

$$g(z) = \{z \text{ if } z \geq 0 \text{ or } 0 \text{ if } z < 0\}$$

or equivalently $g(z) = \max\{0, z\}$

Derivative of ReLU:

$$\frac{dg(z)}{dz} = \{1 \text{ if } z > 0 \text{ or } 0 \text{ if } z < 0\}$$

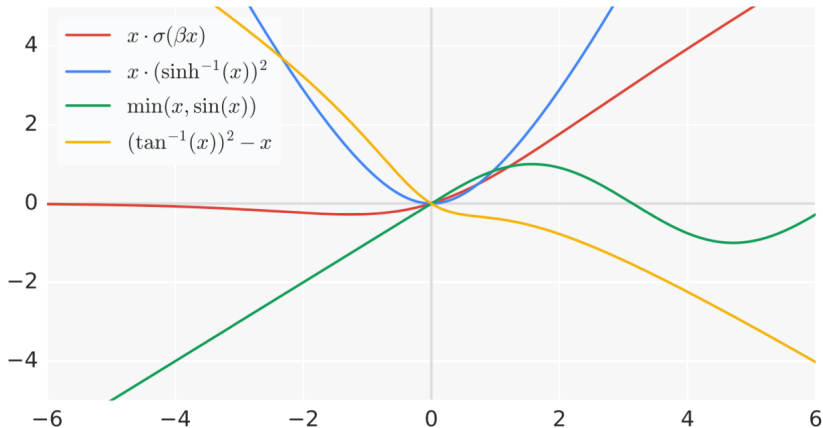
non-differentiable or undefined if $z = 0$

(in practice: choose a value for $z = 0$)

Desperately Seeking Transfer Functions

from [3]

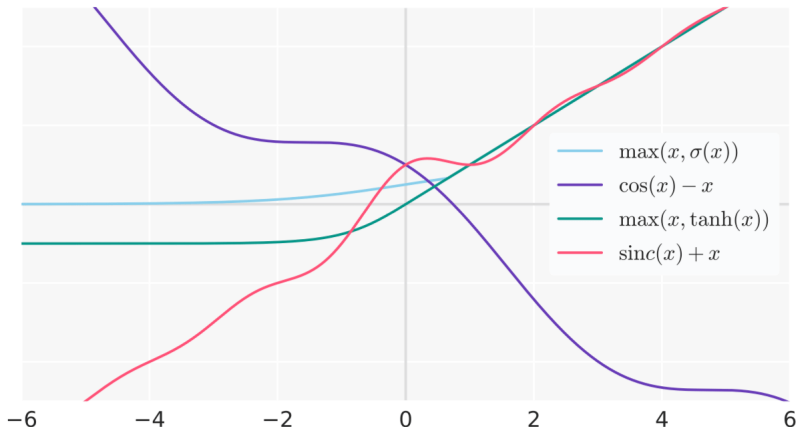
Enumeration of non-linear functions



Desperately Seeking Transfer Functions

from [3]

Enumeration of non-linear functions



The Swish Transfer Function [3]

Enumeration of activation functions:

Swish was the end result of comparing all the auto-generated activation functions for accuracy on standard datasets.

Swish uses the sigmoid σ :

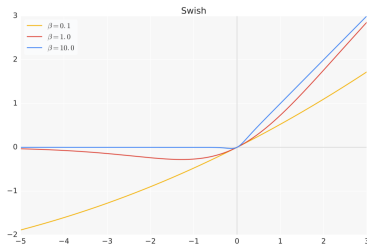
$$g(z) = z \cdot \sigma(\beta z)$$

- ▶ If $\beta = 0$ then $g(z) = \frac{z}{2}$ (a linear function; so avoid this)
- ▶ If $\beta \rightarrow \infty$ then $g(z) = \text{ReLU}$

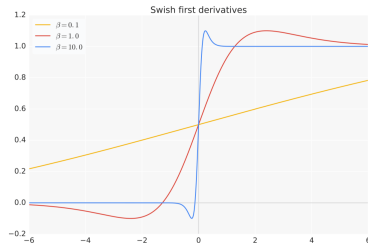
Derivative of Swish:

$$\frac{dg(z)}{dz} = \beta g(z) + \sigma(\beta z)(1 - \beta g(z))$$

The Swish Transfer Function [3]



Swish transfer function with different values of β



First derivative of the Swish transfer function

Derivatives w.r.t. parameters

Derivatives w.r.t. w :

Given

$$h = g(w \cdot x + b)$$

derivatives w.r.t. $w_1, \dots, w_j, \dots, w_d$:

$$\frac{dh}{dw_j}$$

Derivatives w.r.t. b :

derivatives w.r.t. b :

$$\frac{dh}{db}$$

Chain Rule of Differentiation

Introduce an intermediate variable $z \in \mathbb{R}$

$$z = w \cdot x + b$$

$$h = g(z)$$

Then by the chain rule to differentiate w.r.t. w :

$$\frac{dh}{dw_j} = \frac{dh}{dz} \frac{dz}{dw_j} = \frac{dg(z)}{dz} \times x_j$$

And similarly for b :

$$\frac{dh}{db} = \frac{dh}{dz} \frac{dz}{db} = \frac{dg(z)}{dz} \times 1$$

Single Layer Feedforward model

A single layer feedforward model consists of:

- ▶ An integer d specifying the input dimension. Each input to the network is $x \in \mathbb{R}^d$
- ▶ An integer m specifying the number of hidden units
- ▶ A parameter matrix $W \in \mathbb{R}^{m \times d}$. The vector $W_k \in \mathbb{R}^d$ for $1 \leq k \leq m$ is the k th row of W
- ▶ A vector $b \in \mathbb{R}^d$ of bias parameters
- ▶ A transfer function $g : \mathbb{R} \rightarrow \mathbb{R}$
 $g(z) = \text{ReLU}(z)$ or $g(z) = \tanh(z)$

Single Layer Feedforward model (continued)

For $k = 1, \dots, m$:

- ▶ The input to the k th neuron is: $z_k = W_k \cdot x + b_k$
- ▶ The output from the k th neuron is: $h_k = g(z_k)$
- ▶ Define vector $\phi(x; \theta) \in \mathbb{R}^m$ as: $\phi(x; \theta) = h_k$
- ▶ $\theta = (W, b)$ where $W \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^d$
- ▶ Size of θ is $m \times (d + 1)$ parameters

Some intuition

The neural network employs m hidden units, each with their own parameters W_k and b_k , and these neurons are used to construct a *hidden* representation $h \in \mathbb{R}^m$

Matrix Form

We can replace the operation:

$$z_k = W_k \cdot x + b \text{ for } k = 1, \dots, m$$

with

$$z = Wx + b$$

where the dimensions are as follows (vector of size m equals a matrix of size $m \times 1$):

$$\underbrace{z}_{m \times 1} = \underbrace{W}_{m \times d} \underbrace{x}_{d \times 1} + \underbrace{b}_{m \times 1}$$

$\underbrace{\hspace{10em}}_{m \times 1}$

Single Layer Feedforward model (matrix form)

A single layer feedforward model consists of:

- ▶ An integer d specifying the input dimension. Each input to the network is $x \in \mathbb{R}^d$
- ▶ An integer m specifying the number of hidden units
- ▶ A parameter matrix $W \in \mathbb{R}^{m \times d}$
- ▶ A vector $b \in \mathbb{R}^d$ of bias parameters
- ▶ A transfer function $g : \mathbb{R}^m \rightarrow \mathbb{R}^m$
 $g(z) = [\dots, \text{ReLU}(z_i), \dots]$ or
 $g(z) = [\dots, \tanh(z_i), \dots]$ or
 $g(z) = [\dots, \sigma(z_i), \dots]$ or
for $i = 1, \dots, m$

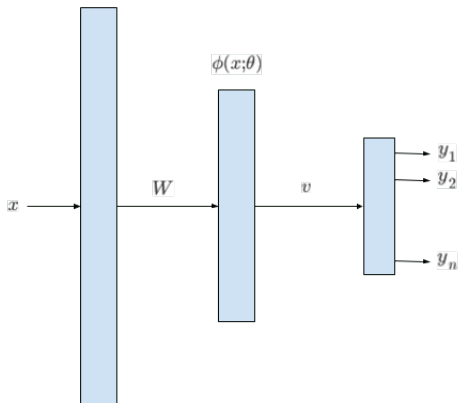
Single Layer Feedforward model (matrix form, continued)

- ▶ Vector of inputs to the hidden layer $z \in \mathbb{R}^m$: $z = Wx + b$
- ▶ Vector of outputs from hidden layer $h \in \mathbb{R}^m$: $h = g(z)$
- ▶ Define $\phi(x; \theta) = h$ where $\theta = (W, b)$
- ▶ Define $\text{softmax}_y = \frac{\exp(r_y)}{\sum_{y'} \exp(r_{y'})}$ for $r_y = v(y) \cdot h + \gamma_y$
- ▶ Let $V = [\dots, v_y, \dots]$ for $y \in \mathcal{Y}$. $v_y \in \mathbb{R}^m$ so $V \in \mathbb{R}^{|\mathcal{Y}| \times m}$.
- ▶ Let $\Gamma = [\dots, \gamma_y, \dots]$ for $y \in \mathcal{Y}$. $\Gamma \in \mathbb{R}^{|\mathcal{Y}|}$.

Putting it all together:

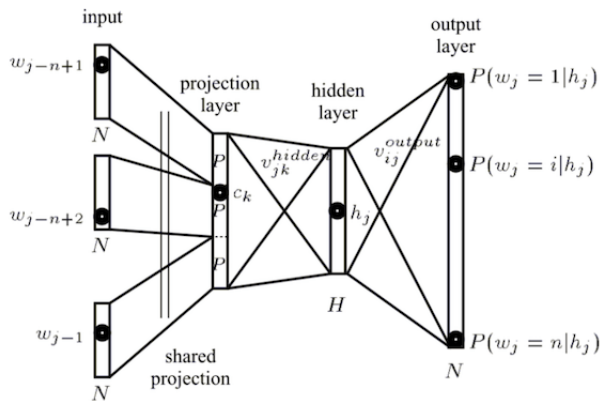
$$\underbrace{r}_{\text{vector of size } |\mathcal{Y}|} = \text{softmax}\left(\underbrace{V \cdot \phi(x; \theta) + \Gamma}_{\substack{\text{for each } y \in \mathcal{Y} \text{ an } \mathbb{R} \text{ value} \\ \text{A vector of size } \mathbb{R}^{\mathcal{Y}} \text{ that sums to 1}}}\right)$$

Feedforward neural network

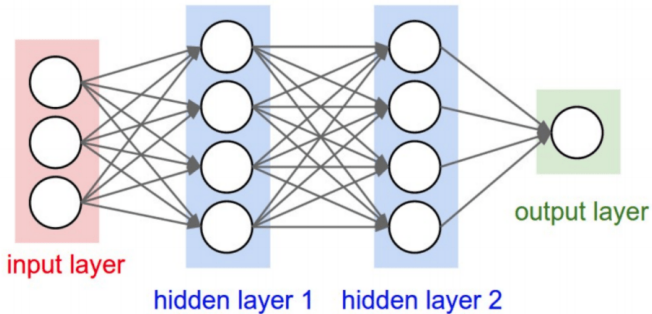


n-gram Feedforward neural network

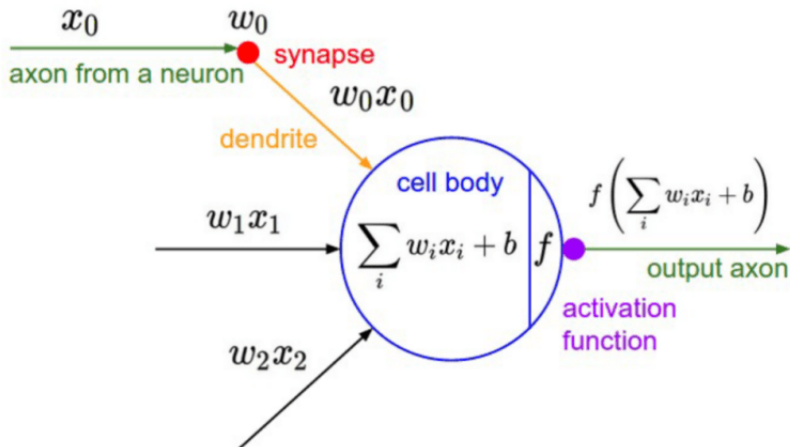
from [5]



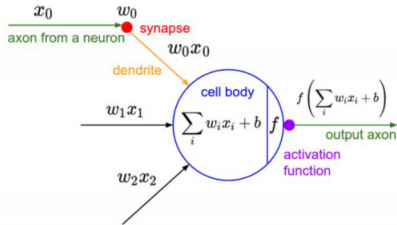
Feed-forward Network



Neuron

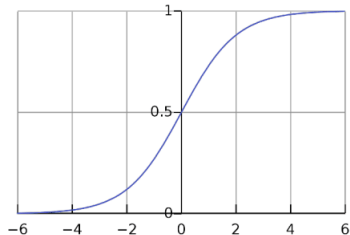


Neuron

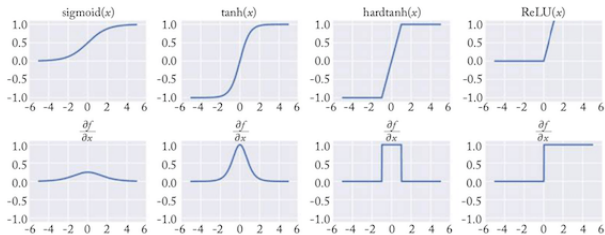


$$f(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\top \mathbf{x} + b)$$



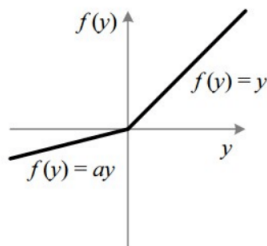
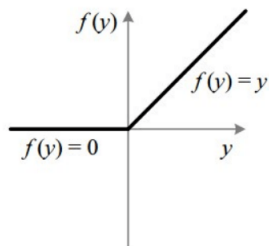
Activation Functions



Leaky ReLU

Leaky ReLU

$$f(z) = \begin{cases} z & z \geq 0 \\ 0.01z & z < 0 \end{cases}$$



Log-linear models versus Neural networks

Feedforward neural networks

Motivating example: XOR

Stochastic Gradient Descent

Computation Graphs

Motivating example: the XOR problem

From *Deep Learning* by Goodfellow, Bengio, Courville

We will assume a training set where each label is in the set $\mathcal{Y} = \{-1, +1\}$

There are four training examples:

$$x^1 = [0, 0], y^1 = -1$$

$$x^2 = [0, 1], y^2 = +1$$

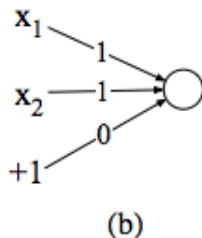
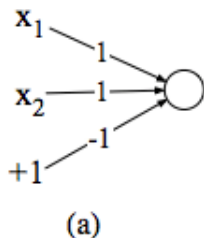
$$x^3 = [1, 0], y^3 = +1$$

$$x^4 = [1, 1], y^4 = -1$$

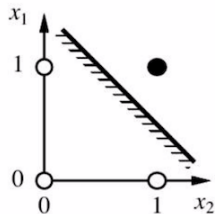
No solution with single neuron (perceptron)

Perceptron can compute **and** and **or**

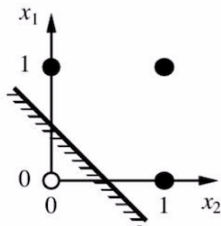
$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



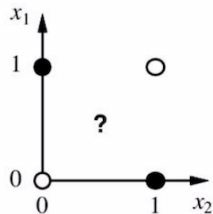
Motivating example: the XOR problem



x_1 and x_2



x_1 or x_2



x_1 xor x_2

Motivating example: the XOR problem

Theorem

For examples (x^i, y^i) for $i = 1, \dots, 4$ as defined previously for the feedforward neural network:

$$\Pr(y \mid x; W, b, v) = \frac{\exp(v(y) \cdot g(Wx + b) + \gamma_y)}{\sum_{y' \in \mathcal{Y}} \exp(v(y') \cdot g(Wx + b) + \gamma_{y'})}$$

where $x \in \mathbb{R}^2$ ($d = 2$) and let $m = 2$ so $W \in \mathbb{R}^{2 \times 2}$ and $b \in \mathbb{R}^2$ and g is a ReLU transfer function.

Then there are parameter settings $v(-1)$, $v(+1)$, γ_{-1} , γ_{+1} , W , b such that

$$p(y^i \mid x^i; v) > 0.5 \text{ for } i = 1, \dots, 4$$

Motivating example: the XOR problem

Proof Sketch

Define $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ Then for each input x calculate values of $z = Wx + b$ and $h = g(z)$:

$$x = [0, 0] \Rightarrow z = [0, -1] \Rightarrow h = [0, 0]$$

$$x = [1, 0] \Rightarrow z = [1, 0] \Rightarrow h = [1, 0]$$

$$x = [0, 1] \Rightarrow z = [1, 0] \Rightarrow h = [1, 0]$$

$$x = [1, 1] \Rightarrow z = [2, 1] \Rightarrow h = [2, 1]$$

Motivating example: the XOR problem

Proof Sketch (continued)

$$\begin{aligned} p(+1 \mid x; v) &= \frac{\exp(v(+1) \cdot h + \gamma_{+1})}{\exp(v(+1) \cdot h + \gamma_{+1}) + \exp(v(-1) \cdot h + \gamma_{-1})} \\ &= \frac{1}{1 + \exp(-(u \cdot h + \gamma))} \end{aligned}$$

To satisfy $P(y^i \mid x^i; v) > 0.5$ for $i = 1, \dots, 4$ we have to find parameters $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$ such that:

$$u \cdot [0, 0] + \gamma < 0$$

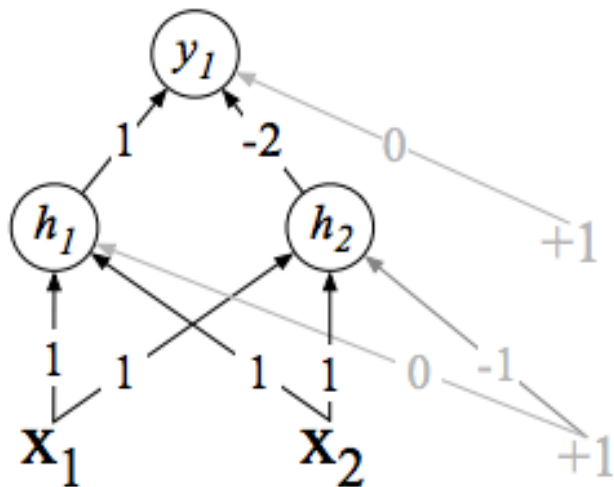
$$u \cdot [1, 0] + \gamma > 0$$

$$u \cdot [1, 0] + \gamma > 0$$

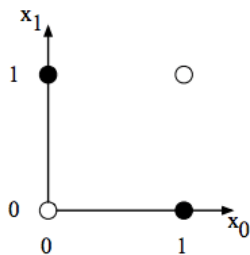
$$u \cdot [2, 1] + \gamma < 0$$

$u = [1, -2]$ and $\gamma = -0.5$ satisfies these constraints.

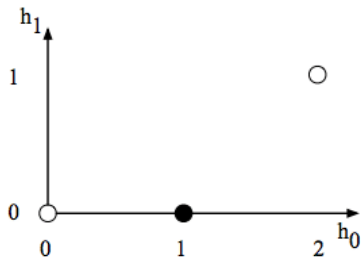
Solving the XOR problem



Solving the XOR problem



a) The original x space

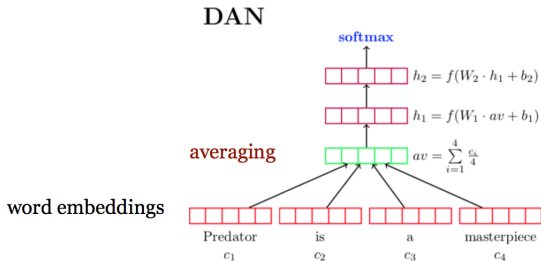


b) The new h space

Text Classification

Neural Bag-of-Words (NBOW)

- Deep Averaging Networks (DAN) for Text Classification



(Iyyer et 2015): Deep Unordered Composition Rivals Syntactic Methods for Text Classification

Word embeddings: re-train or not?

- Word embeddings can be treated as parameters too!

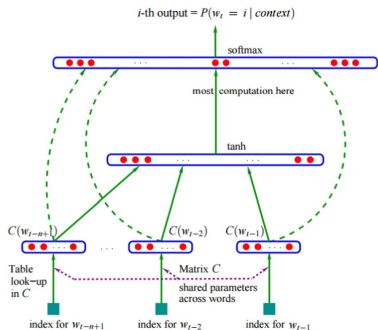
$$\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{w}^{(o)}, b^{(o)}, \mathbf{E}_{emb}\}$$

- When the training set is small, don't re-train word embeddings (think of them as features!).
- Most cases: initialize word embeddings using pre-trained ones (word2vec, Glove) and re-train them for the task
- When you have enough data, you can just randomly initialize them and train from scratch (e.g. machine translation)



Feedforward Neural LMs

- N-gram models: $P(\text{mat}|\text{the cat sat on the})$



- Input layer (context size $n = 5$):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

concatenation

- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^h$$

- Output layer (softmax)

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i | \text{context}) = \text{softmax}_i(\mathbf{z})$$

Loss functions

Binary classification

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$

$$\mathcal{L}(y, y^*) = -y^* \log y - (1 - y^*) \log (1 - y)$$

• Regression

$$y = \mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)}$$

$$\mathcal{L}_{\text{MSE}}(y, y^*) = (y - y^*)^2$$

Multi-class classification (C classes)

$$y_i = \text{softmax}_i(\mathbf{W}^{(o)} \mathbf{h}_2 + \mathbf{b}^{(o)}) \quad \mathbf{W}^{(o)} \in \mathbb{R}^{C \times d_2}, \mathbf{b}^{(o)} \in \mathbb{R}^C$$

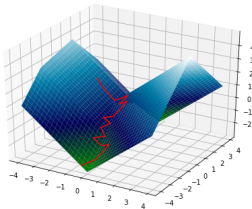
$$\mathcal{L}(y, y^*) = - \sum_{i=1}^C y_i^* \log y_i$$

The question again becomes how to compute: $\nabla_{\theta} \mathcal{L}(\theta)$

Optimization

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} J(\theta)$$

- ▶ Recall logistic regression is convex: one global minimum
- ▶ Neural networks are non-convex and not easy to optimize
- ▶ Use stochastic gradient descent and family of "adaptive" optimizers that scale the parameter adjustment by an accumulated gradient
 - ▶ **SGD**, Adam, Adagrad, RMSProp, ...



Log-linear models versus Neural networks

Feedforward neural networks

Motivating example: XOR

Stochastic Gradient Descent

Computation Graphs

Simple stochastic gradient descent

Inputs:

- ▶ Training examples (x^i, y^i) for $i = 1, \dots, n$
- ▶ A feedforward representation $\phi(x; \theta)$
- ▶ Integer T specifying the number of updates
- ▶ A sequence of learning rates: η^1, \dots, η^T where $\eta^t \in [0, 1]$
 - ▶ One should experiment with learning rates: 0.001, 0.01, 0.1, 1
 - ▶ Bottou (2012) suggests a learning rate $\eta^t = \frac{\eta^1}{1 + \eta^1 \times \lambda \times t}$ where λ is a hyperparameter that can be tuned experimentally

Initialization:

Set $v = (v(y), \gamma_y)$ for all y , and θ to random values

Gradient descent

Algorithm:

- ▶ For $t = 1, \dots, T$
 - ▶ Select an integer i uniformly at random from $\{1, \dots, n\}$
 - ▶ Define $L(\theta, \nu) = -\log P(y_i \mid x_i; \theta, \nu)$
 - ▶ For each parameter θ_j and $\nu_k(y)$ and γ_y (for each label y):

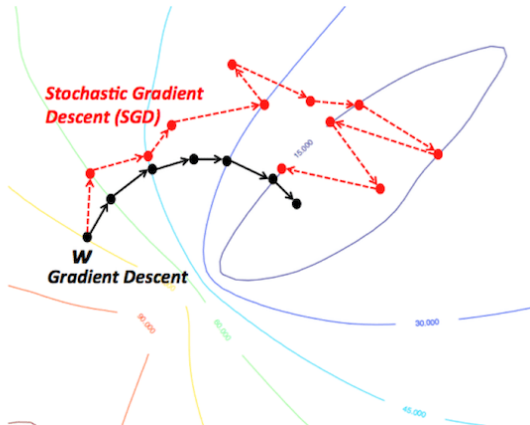
$$\theta_j = \theta_j - \eta^t \times \frac{dL(\theta, \nu)}{d\theta_j}$$

$$\nu_k(y) = \nu_k(y) - \eta^t \times \frac{dL(\theta, \nu)}{d\nu_k(y)}$$

$$\gamma(y) = \gamma(y) - \eta^t \times \frac{dL(\theta, \nu)}{d\gamma(y)}$$

- ▶ **Output:** parameters $\theta, \nu = (\nu(y), \gamma_y)$ for all y

SGD



Log-linear models versus Neural networks

Feedforward neural networks

Motivating example: XOR

Stochastic Gradient Descent

Computation Graphs

Complex neural networks

Neural network with a loss function

Consider a neural network trained using a **squared-error loss**. For the correct answer y^* the output value y is compared using the function $(y^* - y)^2$.

$$h' = W_{xh}x + b_h$$

$$h = \tanh(h')$$

$$y = w_{hy}h + b_y$$

$$\ell = (y^* - y)^2$$

Derivative wrt loss

$$h' = W_{xh}x + b_h$$

$$h = \tanh(h')$$

$$y = w_{hy}h + b_y$$

$$\ell = (y^* - y)^2$$

We want to compute $\frac{d\ell}{db_y}$, $\frac{d\ell}{dw_{hy}}$, $\frac{d\ell}{db_h}$, $\frac{d\ell}{dW_{xh}}$

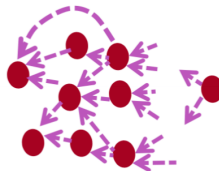
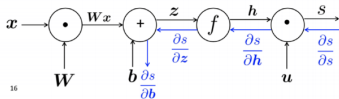
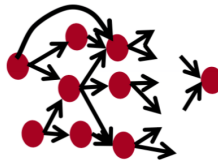
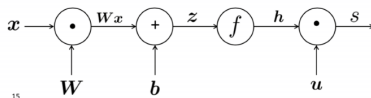
$$\frac{d\ell}{db_y} = \frac{d\ell}{dy} \frac{dy}{db_y}$$

$$\frac{d\ell}{dw_{hy}} = \frac{d\ell}{dy} \frac{dy}{dw_{hy}}$$

$$\frac{d\ell}{db_h} = \frac{d\ell}{dy} \frac{dy}{dh} \frac{dh}{dh'} \frac{dh'}{db_h}$$

$$\frac{d\ell}{dW_{xh}} = \frac{d\ell}{dy} \frac{dy}{dh} \frac{dh}{dh'} \frac{dh'}{dW_{xh}}$$

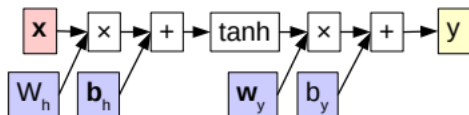
Computation graphs and automatic differentiation



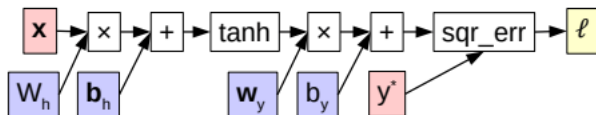
Computation graphs and automatic differentiation

Figure from [1]

Graph for the Function Itself



Graph for the Training Objective



Computation graphs and automatic differentiation

- ▶ Automatic differentiation is a two-step dynamic programming algorithm that operates over the second graph and performs:
 - Forward calculation** which traverses the nodes in the graph in topological order, calculating the actual result of the computation.
 - Back propagation** which traverses the nodes in reverse topological order, calculating the gradients.
- ▶ Many neural network toolkits can perform auto differentiation for very large computation graphs.

- [1] **Graham Neubig**
Neural Networks for NLP
2018.
- [2] **Yoav Goldberg**
Neural Network Methods for Natural Language Processing
2017.
- [3] **Prajit Ramachandran, Barret Zoph, Quoc V. Le**
Searching for Activation Functions
2017.
- [4] **Xavier Glorot, Yoshua Bengio**
Understanding the difficulty of training deep feedforward
neural networks
2010.
- [5] **Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian
Jauvin**
A Neural Probabilistic Language Model
2003.

Acknowledgements

Many slides borrowed or inspired from lecture notes by Anoop Sarkar, Danqi Chen, Karthik Narasimhan, Dan Jurafsky, Michael Collins, Chris Dyer, Kevin Knight, Chris Manning, Philipp Koehn, Adam Lopez, Graham Neubig, Richard Socher and Luke Zettlemoyer from their NLP course materials.

All mistakes are my own.

A big thank you to all the students who read through these notes and helped me improve them.