CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang

Outline

- Maximum Entropy
- Feedforward Neural Networks
- Recurrent Neural Networks

Introduction

- So far we've looked at "generative models"
  - Language models, Naive Bayes
- But there is now much use of conditional or discriminative probabilistic models in NLP, Speech, IR (and ML generally)
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent, retargetable NLP modules

Joint vs. Conditional Models

- We have some data {(*d*, *c*)} of paired observations *d* and hidden classes *c*.
- Joint (generative) models place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):

$P(c,d)$

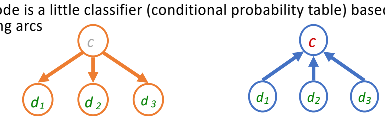
  - All the classic statistic NLP models:
    - *n*-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

Joint vs. Conditional Models

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data: $P(c|d)$
  - Logistic regression, conditional loglinear or maximum entropy models, conditional random fields
  - Also, SVMs, (averaged) perceptron, etc. are discriminative classifiers (but not directly probabilistic)

Bayes Network/Graphical Models

- Bayes network diagrams draw circles for random variables, and lines for direct dependencies
- Some variables are observed; some are hidden
- Each node is a little classifier (conditional probability table) based on incoming arcs



Naive Bayes — Generative

Logistic Regression (aka Maximum Entropy) — Discriminative

Conditional vs. Joint Likelihood

- A *joint* model gives probabilities $P(d,c)$ and tries to maximize this joint likelihood.
  - It turns out to be trivial to choose weights: just relative frequencies.
- A *conditional* model gives probabilities $P(c|d)$. It takes the data as given and models only the conditional probability of the class.
  - We seek to maximize conditional likelihood.
  - Harder to do.
  - More closely related to classification error. (Easy to tune!)

Maximum Entropy (MaxEnt)

- Or logistic regression

Features

- In these slides and most maxent work: *features (or feature functions)* f are elementary pieces of evidence that link aspects of what we observe d with a category c that we want to predict
- A feature is a function with a **bounded** real value: $f: C \times D \to \mathbb{R}$

Example Task: Named Entity Type

LOCATION *in Arcadia* — LOCATION *in Québec* — DRUG *taking Zantac* — PERSON *saw Sue*

Example features

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$

LOCATION *in Arcadia* — LOCATION *in Québec* — DRUG *taking Zantac* — PERSON *saw Sue*

- Models will assign to each feature a *weight:*
  - A positive weight votes that this configuration is likely correct
  - A negative weight votes that this configuration is likely incorrect

Features

- In NLP uses, usually a feature specifies
  1. an indicator function – a yes/no boolean matching function – of properties of the input and a particular class

$$f_i(c, d) \equiv [\Phi(d) \wedge c = c_j] \qquad \text{[Value is 0 or 1]}$$

- Each feature picks out a data subset and suggests a label for it

Feature-Based Models

- The decision about a data point is based only on the features active at that point.

Text Categorization	Word-Sense Disambiguation	POS Tagging
Data: BUSINESS: Stocks hit a yearly low …	Data: … to restructure bank:MONEY debt.	Data: DT JJ NN … The previous fall …
Label: BUSINESS	Label: MONEY	Label: NN
Features {…, stocks, hit, a, yearly, low, …}	Features {…, w_{-1}=restructure, w_{+1}=debt, L=12, …}	Features {w=fall, t_{-1}=JJ w_{-1}=previous}

Other Maxent Classifier Examples

- You can use a maxent classifier whenever you want to assign data points to one of a number of classes:
  - Sentence boundary detection (Mikheev 2000)
    - Is a period end of sentence or abbreviation?
  - Sentiment analysis (Pang and Lee 2002)
    - Word unigrams, bigrams, POS counts, …
  - Prepositional phrase attachment (Ratnaparkhi 1998)
    - Attach to verb or noun? Features of head noun, preposition, etc.
  - Parsing decisions in general (Ratnaparkhi 1997; Johnson et al. 1999, etc.)

Feature-Based Linear Classifiers

- Linear classifiers at classification time:
  - Linear function from feature sets $\{f_i\}$ to classes $\{c\}$.
  - Assign a weight λ_i to each feature f_i.
  - We consider each class for sample d
  - For a pair (c,d), features vote with their weights:
    - vote(c) = $\Sigma\lambda_i f_i(c,d)$

PERSON	LOCATION	DRUG
in Québec	*in Québec*	*in Québec*

  - Choose the class c which maximizes $\Sigma\lambda_i f_i(c,d)$

Feature-Based Linear Classifiers

- $f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
- $f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
- $f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

$f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
$f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
$f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

- Maximum Entropy:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

Makes votes positive (numerator)
Normalizes votes (denominator)

$f_1(c, d) \equiv [c = \text{LOCATION} \wedge w_{-1} = \text{“in”} \wedge \text{isCapitalized}(w)]$ -> weight 1.8
$f_2(c, d) \equiv [c = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$ -> weight -0.6
$f_3(c, d) \equiv [c = \text{DRUG} \wedge \text{ends}(w, \text{“c”})]$ -> weight 0.3

- Maximum Entropy:
  - Make a probabilistic model from the linear combination $\Sigma\lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

Makes votes positive (numerator)
Normalizes votes (denominator)

  - P(LOCATION|*in Québec*) = $e^{1.8}e^{-0.6}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.586$
  - P(DRUG|*in Québec*) = $e^{0.3}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.238$
  - P(PERSON|*in Québec*) = $e^{0}/(e^{1.8}e^{-0.6} + e^{0.3} + e^{0}) = 0.176$
- The weights are the parameters of the probability model, combined via a “soft max” function

Feature-Based Linear Classifiers

- Exponential models:
  - Given this model form, we will choose parameters $\{\lambda_i\}$ that *maximize the conditional likelihood* of the data according to this model.
  - We construct not only classifications, but probability distributions over classifications.
    - There are other (good!) ways of discriminating classes – SVMs, boosting, even perceptrons – but these methods are not as trivial to interpret as distributions over classes.

Outline

- Maximum Entropy
- Feedforward Neural Networks
- Recurrent Neural Networks

Neural Network Learning

- Learning approach based on modeling adaptation in biological neural systems.
- Perceptron: Initial algorithm for learning simple neural networks (single layer) developed in the 1950's.
- Backpropagation: More complex algorithm for learning multi-layer neural networks developed in the 1980's.

ARTIFICIAL NEURON

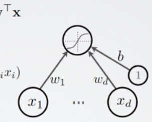
Topics: connection weights, bias, activation function

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

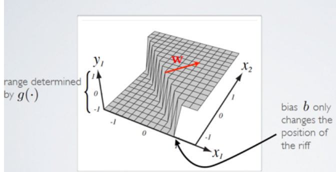
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



- $\mathbf{w}$ are the connection weights
- b is the neuron bias
- $g(\cdot)$ is called the activation function

ARTIFICIAL NEURON

Topics: connection weights, bias, activation function

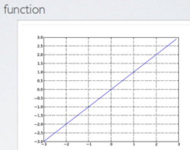


(from Pascal Vincent's slides)

ACTIVATION FUNCTION

Topics: linear activation function

- Performs no input squashing
- Not very interesting...

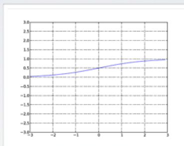


$$g(a) = a$$

ACTIVATION FUNCTION

Topics: sigmoid activation function

- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

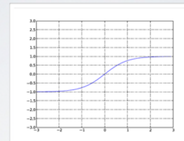


$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$$

ACTIVATION FUNCTION

Topics: hyperbolic tangent ("tanh") activation function

- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

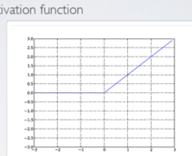


$$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

ACTIVATION FUNCTION

Topics: rectified linear activation function

- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

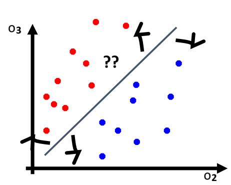


$$g(a) = \mathrm{reclin}(a) = \max(0, a)$$

```
class Neuron(object):
  # ...
  def forward(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Linear Separator

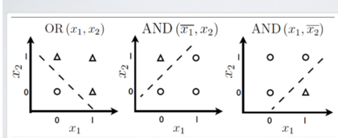
- Since one-layer neuron (aka perceptron) uses linear threshold function, it is searching for a linear separator that discriminates the classes.



ARTIFICIAL NEURON

Topics: capacity of single neuron

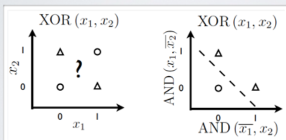
- Can solve linearly separable problems



ARTIFICIAL NEURON

Topics: capacity of single neuron

- Can't solve non linearly separable problems...



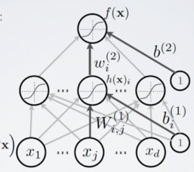
- ... unless the input is transformed in a better representation

NEURAL NETWORK

Topics: single hidden layer neural network

- Hidden layer pre-activation:
  $\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$
  $\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$
- Hidden layer activation:
  $\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$
- Output layer activation:
  $f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$

output activation function



NEURAL NETWORK

Topics: softmax activation function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability $p(y = c|\mathbf{x})$
- We use the softmax activation function at the output:

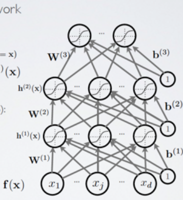
$$\mathbf{o}(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

NEURAL NETWORK

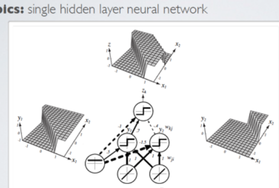
Topics: multilayer neural network

- Could have L hidden layers:
  - layer pre-activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$
    $\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$
  - hidden layer activation (k from 1 to L):
    $\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$
  - output layer activation ($k=L+1$):
    $\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$



CAPACITY OF NEURAL NETWORK

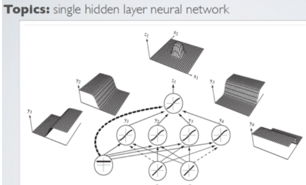
Topics: single hidden layer neural network



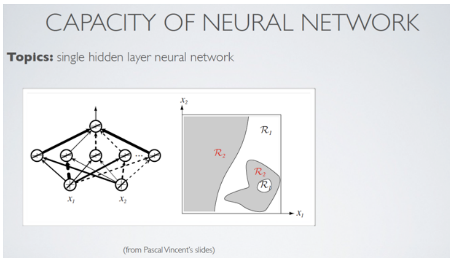
(from Pascal Vincent's slides)

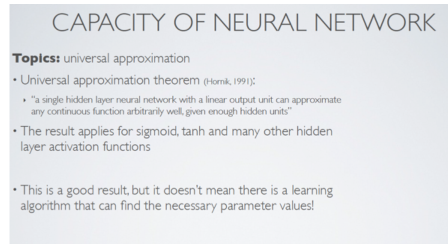
CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network

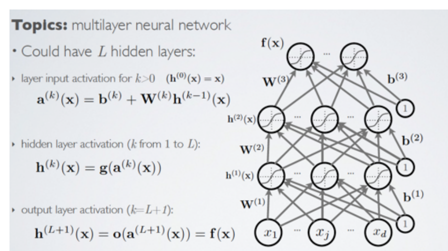


(from Pascal Vincent's slides)

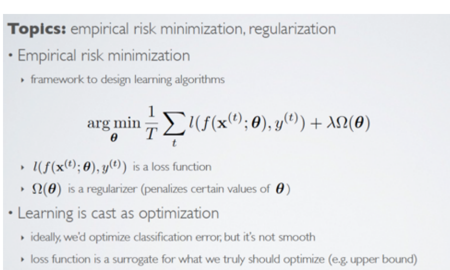




How to train a neural network?



Empirical Risk Minimization



The rest of the slides are for reference only (not covered by lecture or exam)

The Learning Algorithm

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
      - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$

    training epoch = iteration over **all** examples
- To apply this algorithm to neural network training, we need
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  - initialization method

LOSS FUNCTION

Topics: loss function for classification

- Neural network estimates $f(\mathbf{x})_c = p(y = c|\mathbf{x})$
  - we could maximize the probabilities of $y^{(t)}$ given $\mathbf{x}^{(t)}$ in the training set
- To frame as minimization, we minimize the negative log-likelihood

natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy

The Learning Algorithm

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
      - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$

    training epoch = iteration over **all** examples
- To apply this algorithm to neural network training, we need
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - ➡ a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  - initialization method

Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)

Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)

GRADIENT COMPUTATION

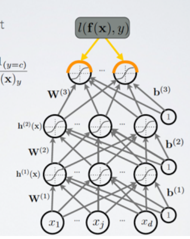
Topics: loss gradient at output

- Partial derivative:

$$\frac{\partial}{\partial f(\mathbf{x})_c} - \log f(\mathbf{x})_y \quad = \quad \frac{-1_{(y=c)}}{f(\mathbf{x})_y}$$

- Gradient:

$$\begin{aligned} & \nabla_{\mathbf{f}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = \quad & \frac{-1}{f(\mathbf{x})_y} \begin{bmatrix} 1_{(y=0)} \\ \vdots \\ 1_{(y=C-1)} \end{bmatrix} \\ = \quad & \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y} \end{aligned}$$



GRADIENT COMPUTATION

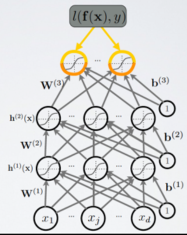
Topics: loss gradient at output pre-activation

- Partial derivative:

$$\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y$$
$$= \; -\left(1_{(y=c)} - f(\mathbf{x})_c\right)$$

- Gradient:

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y$$
$$= \; -\left(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})\right)$$



$$\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})}
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= \; & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= \; & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)\left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)}{\left(\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)^2}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{1_{(y=c)}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)\left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)}{\left(\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)^2}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{1_{(y=c)}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y\,\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c\right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)\left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)}{\left(\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)^2}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{1_{(y=c)}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y\,\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}f(\mathbf{x})_y - f(\mathbf{x})_y\,f(\mathbf{x})_c\right)
\end{aligned}$$

$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)\left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)}{\left(\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)^2}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(\frac{1_{(y=c)}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y\,\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c\right) \\
= & \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}f(\mathbf{x})_y - f(\mathbf{x})_y\,f(\mathbf{x})_c\right) \\
= & -\left(1_{(y=c)} - f(\mathbf{x})_c\right)
\end{aligned}$$

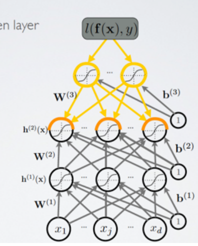
Gradient Computation

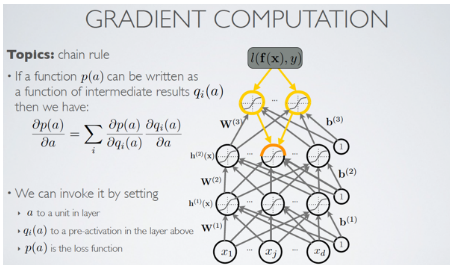
- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)

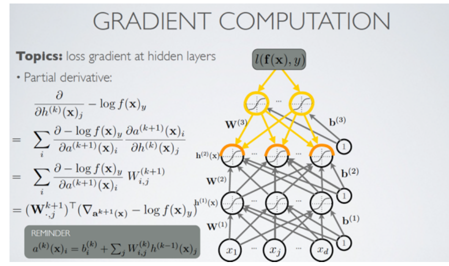
GRADIENT COMPUTATION

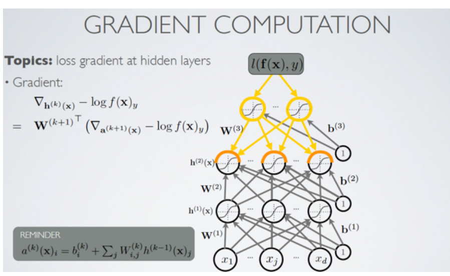
Topics: loss gradient at hidden layer

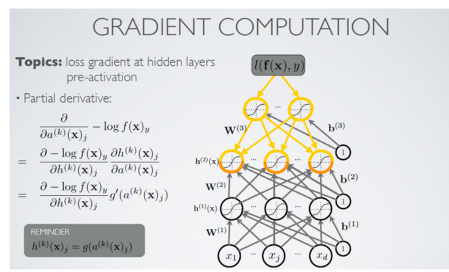
- ... this is getting complicated!!

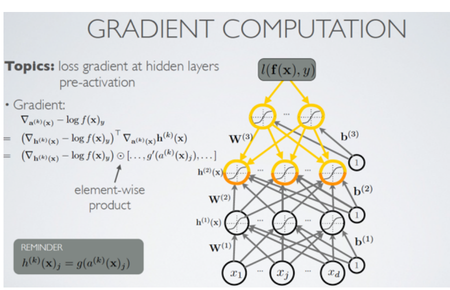






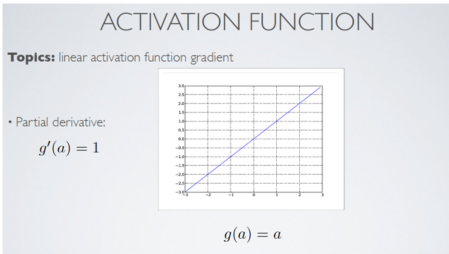


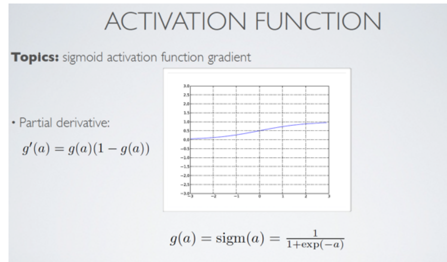


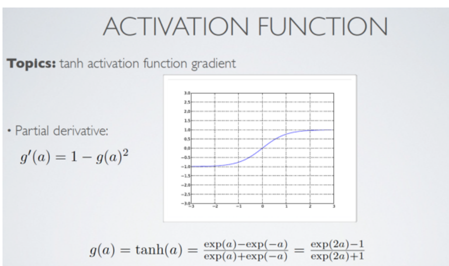


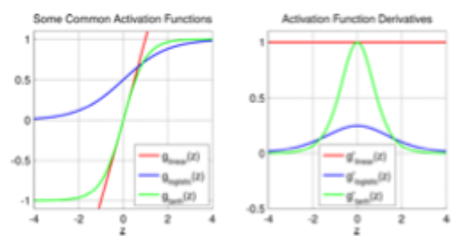
Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)



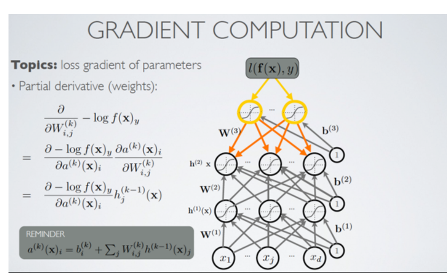


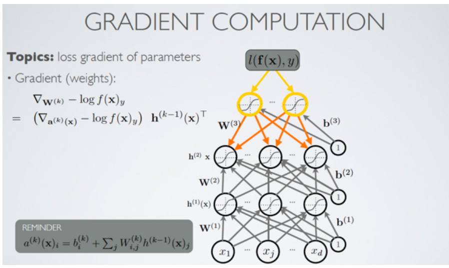


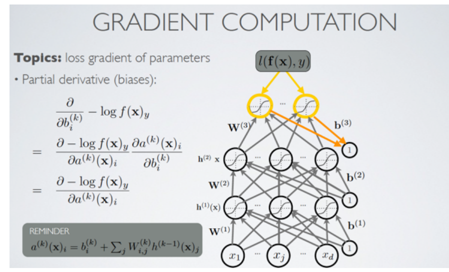


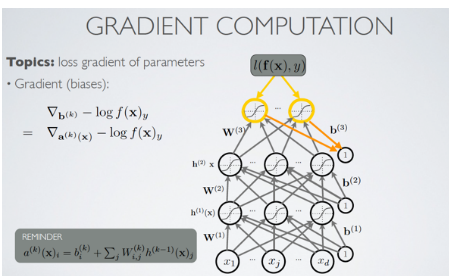
Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)

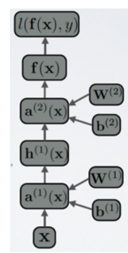


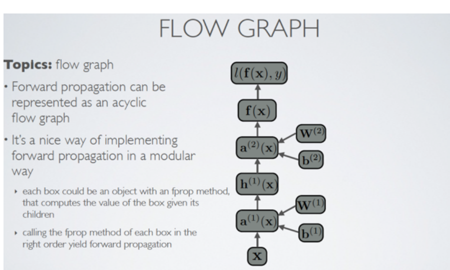




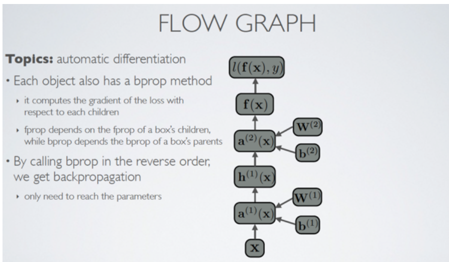


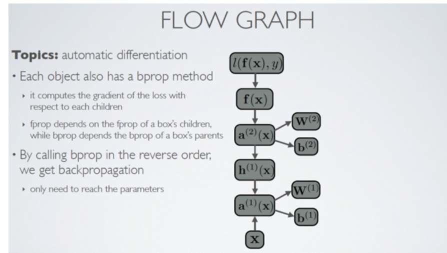
Backpropagation

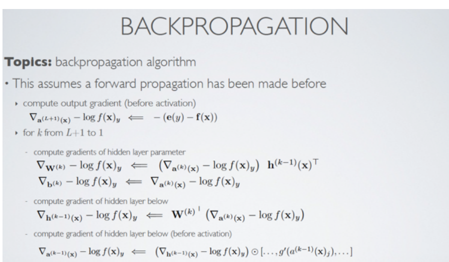


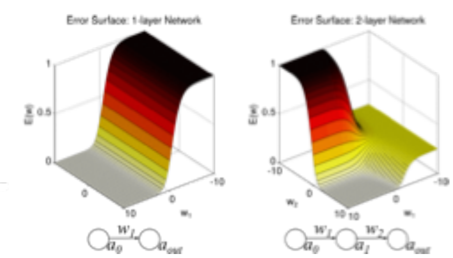


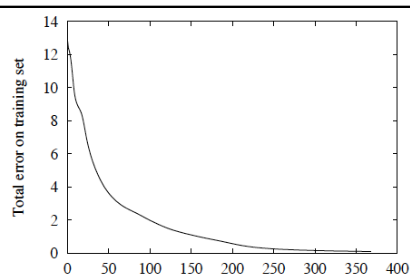

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```



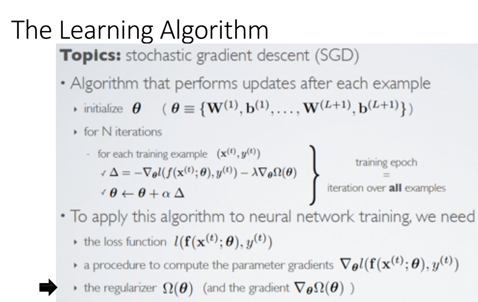








[figure from Greg Mori's slides]



REGULARIZATION

Topics: L2 regularization

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

- Gradient: $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\mathbf{W}^{(k)}$
- Only applied on weights, not on biases (weight decay)
- Can be interpreted as having a Gaussian prior over the weights

REGULARIZATION

Topics: L1 regularization

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

- Gradient: $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = \text{sign}(\mathbf{W}^{(k)})$
  - where $\text{sign}(\mathbf{W}^{(k)})_{i,j} = 1_{\mathbf{W}_{i,j}^{(k)}>0} - 1_{\mathbf{W}_{i,j}^{(k)}<0}$
- Also only applied on weights
- Unlike L2, L1 will push certain weights to be exactly 0
- Can be interpreted as having a Laplacian prior over the weights

Empirical Risk Minimization

Topics: empirical risk minimization, regularization

- Empirical risk minimization
  - framework to design learning algorithms

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

  - $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is a loss function
  - $\Omega(\boldsymbol{\theta})$ is a regularizer (penalizes certain values of $\boldsymbol{\theta}$)
- Learning is cast as optimization
  - ideally, we'd optimize classification error, but it's not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)

[http://cs231n.github.io/neural-networks-1/]

The Learning Algorithm

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta}$ $\quad(\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\})$
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
      - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$

    training epoch = iteration over **all** examples
- To apply this algorithm to neural network training, we need
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  - ➡ initialization method

INITIALIZATION

Topics: initialization

- For biases
  - initialize all to 0
- For weights
  - Can't initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - Can't initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: sample $\mathbf{W}_{i,j}^{(k)}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$ — size of $\mathbf{h}^{(k)}(\mathbf{x})$
    - the idea is to sample around 0 but break symmetry
    - other values of b could work well (not an exact science) (see Glorot & Bengio, 2010)

The Learning Algorithm

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\boldsymbol{\theta}$ ($\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$)
  - for N iterations
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$
      - ✓ $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$

    training epoch = iteration over **all** examples
- To apply this algorithm to neural network training, we need
  - the loss function $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - the regularizer $\Omega(\boldsymbol{\theta})$ (and the gradient $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$)
  - initialization method

Toolkits

- TensorFlow
  - https://www.tensorflow.org/
- Theano (not maintained any more)
  - http://deeplearning.net/software/theano/
- PyTorch
  - http://pytorch.org/