Logistics

• Proposal grades and comments are available on blackboard.
• Assignment 2 is out.
• Due on 02/25
• Start Early!
• No need to print the code. Just submit it on Blackboard.
What we learned last time

• Perceptron (and kernels)

• Support Vector Machines
Perceptron

\[ f(x) = w_1 x_1 + \ldots + w_6 x_6 \]
Perceptron

• Weighted combination
  – The output of the neuron is a linear combination of the inputs

• Decision Function
  – At the end the results are combined into

\[ f(x) = \sigma \left( \sum_{i=1}^{n} w_i x_i + b \right) \]
Perceptron Algorithm

• Nothing happens if we classify \((x_i, y_i)\) correctly
• If we see incorrectly classified observation we update \(w\) and \(b\)
• Positive reinforcement of observations

\[
\text{argument: } X := \{x_1, \ldots, x_m\} \subset \mathcal{X} \text{ (data)} \\
Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\} \text{ (labels)} \\
\text{function } (w, b) = \text{Perceptron}(X, Y) \\
\text{initialize } w, b = 0 \\
\text{repeat} \\
\quad \text{Pick } (x_i, y_i) \text{ from data} \\
\quad \text{if } y_i(w \cdot x_i + b) \leq 0 \text{ then} \\
\quad \quad w' = w + y_i x_i \\
\quad \quad b' = b + y_i \\
\quad \text{until } y_i(w \cdot x_i + b) > 0 \text{ for all } i \\
\text{end}
\]
Perceptron Algorithm

• About the solution
  – Classification can be written in terms of dot products:

\[ w \cdot x + b = \sum_{j \in E} y_j x_j \cdot x + b \]

argument:  \( X := \{x_1, \ldots, x_m\} \subset X \) (data)
          \( Y := \{y_1, \ldots, y_m\} \subset \{-1, 1\} \) (labels)

function \( (w, b) = \text{Perceptron}(X, Y) \)

initialize \( w, b = 0 \)
repeat
  Pick \((x_i, y_i)\) from data
  if \( y_i(w \cdot x_i + b) \leq 0 \) then
    \[ w' = w + y_i x_i \]
    \[ b' = b + y_i \]
  until \( y_i(w \cdot x_i + b) > 0 \) for all \( i \)
end
$x \rightarrow \varphi(x)$
Perceptron on Features

argument:  \( X := \{x_1, \ldots, x_m\} \subset \mathcal{X} \) (data)
\( Y := \{y_1, \ldots, y_m\} \subset \{\pm 1\} \) (labels)

function \((w, b) = \text{Perceptron}(X, Y, \eta)\)
  initialize \( w, b = 0 \)
  repeat
    Pick \((x_i, y_i)\) from data
    if \(y_i(w \cdot \Phi(x_i) + b) \leq 0\) then
      \[ w' = w + y_i\Phi(x_i) \]
      \[ b' = b + y_i \]
    until \(y_i(w \cdot \Phi(x_i) + b) > 0\) for all \(i\)
  end

Important detail
\[
  w = \sum_j y_j\Phi(x_j) \text{ and hence } f(x) = \sum_j y_j(\Phi(x_j) \cdot \Phi(x)) + b
\]
Kernels

• Definition

• A kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a symmetric function in its arguments for which the following property holds

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$ for some feature map $\Phi$
Some choices of kernel functions

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Kernel Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$\langle x, x' \rangle$</td>
</tr>
<tr>
<td>Laplacian RBF</td>
<td>$\exp(-\lambda | x - x' |)$</td>
</tr>
<tr>
<td>Gaussian RBF</td>
<td>$\exp(-\lambda | x - x' |^2)$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$(\langle x, x' \rangle + c)^d$, $c \geq 0$, $d \in \mathbb{N}$</td>
</tr>
</tbody>
</table>
Support Vector Machine (SVM)

• SVMs (Vapnik, 1990’s) choose the linear separator with the largest margin.
Support vector machines

\[
\text{minimize}_{w,b} \quad w \cdot w \\
(w \cdot x_j + b) y_j \geq 1, \quad \forall j
\]

- Example of a convex optimization problem
  - A quadratic program
  - Polynomial-time algorithms to solve!
- Hyperplane defined by support vectors
  - Could use them as a lower-dimension basis to write down line, although we haven’t seen how yet
- More on these later

**Non-support Vectors:**
- everything else
- moving them will not change \( w \)

**Support Vectors:**
- data points on the canonical lines
Allowing for slack: “Soft margin SVM”

\[
\begin{align*}
    \text{minimize}_{w, b} & \quad w \cdot w + C \sum_j \xi_j \\
    \left( w \cdot x_j + b \right) y_j & \geq 1 - \xi_j, \quad \forall j \quad \xi_j \geq 0
\end{align*}
\]

“slack variables”

**Slack penalty** $C > 0$:
- $C = \infty \rightarrow$ have to separate the data!
- $C = 0 \rightarrow$ ignores the data entirely!
- Select using validation data

For each data point:
- If margin $\geq 1$, don’t care
- If margin $< 1$, pay linear penalty
Dual for the non-separable case

**Primal:**

\[
\begin{align*}
\text{minimize}_{w,b} & \quad \frac{1}{2} w \cdot w + C \sum_j \xi_j \\
(w \cdot x_j + b) y_j & \geq 1 - \xi_j, \quad \forall j \\
\xi_j & \geq 0, \quad \forall j
\end{align*}
\]

**Dual:**

\[
\begin{align*}
\text{maximize}_{\alpha} & \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \\
\sum_i \alpha_i y_i & = 0 \\
C & \geq \alpha_i \geq 0
\end{align*}
\]

**Solve for w,b, \alpha:**

\[
\begin{align*}
w & = \sum_i \alpha_i y_i x_i \\
b & = y_k - w \cdot x_k \\
\text{for any } k \text{ where } C > \alpha_k > 0
\end{align*}
\]
Dual formulation only depends on dot-products of the features!

\[
\max_{\alpha \geq 0, \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j)
\]

First, we introduce a feature mapping:

\[
x_i x_j \rightarrow \Phi(x_i) \cdot \Phi(x_j)
\]

Next, replace the dot product with an equivalent kernel function:

\[
\maximize_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\]

\[
K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)
\]

\[
\sum_i \alpha_i y_i = 0
\]
Today’s Outline

• Feedforward neural network

• Training neural networks

• Restricted Boltzmann machine

Some slides are borrowed from Hugo Larochelle.
ARTIFICIAL NEURON

**Topics:** connection weights, bias, activation function

- Neuron pre-activation (or input activation):
  \[ a(x) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x} \]

- Neuron (output) activation
  \[ h(x) = g(a(x)) = g(b + \sum_i w_i x_i) \]

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

Topics: connection weights, bias, activation function

range determined by $g(\cdot)$

bias $b$ only changes the position of the riff

(from Pascal Vincent’s slides)
**Topics:** linear activation function

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

\[ g(a) = a \]
**Topics:** sigmoid activation function

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

\[ g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)} \]
**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

\[ g(a) = \text{reclin}(a) = \max(0, a) \]
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
ARTIFICIAL NEURON

Topics: capacity, decision boundary of neuron

• Could do binary classification:
  ‣ with sigmoid, can interpret neuron as estimating $p(y = 1|x)$
  ‣ also known as logistic regression classifier
  ‣ if greater than 0.5, predict class 1
  ‣ otherwise, predict class 0

(similar idea can apply with tanh)
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
Topics: single hidden layer neural network

- Hidden layer pre-activation:
  \[ a(x) = b^{(1)} + W^{(1)}x \]
  \[ (a(x)_i = b^{(1)}_i + \sum_j W^{(1)}_{i,j} x_j) \]

- Hidden layer activation:
  \[ h(x) = g(a(x)) \]

- Output layer activation:
  \[ f(x) = o\left(b^{(2)} + w^{(2)^T}h^{(1)}x\right) \]
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 $p(y = c | x)$

- We use the softmax activation function at the output:

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

  - strictly positive
  - sums to one

- Predicted class is the one with highest estimated probability
Topics: multilayer neural network

• Could have $L$ hidden layers:
  
  * layer pre-activation for $k > 0$: $(h^{(0)}(x) = x)$
  
  $a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$

  * hidden layer activation ($k$ from 1 to $L$):
  
  $h^{(k)}(x) = g(a^{(k)}(x))$

  * output layer activation ($k = L + 1$):
  
  $h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)$
**Topics:** single hidden layer neural network

- Hidden layer pre-activation:
  \[ a(x) = b^{(1)} + W^{(1)}x \]
  \[ (a(x)_i = b^{(1)}_i + \sum_j W^{(1)}_{i,j} x_j) \]

- Hidden layer activation:
  \[ h(x) = g(a(x)) \]

- Output layer activation:
  \[ f(x) = o \left( b^{(2)} + w^{(2)^T} h^{(1)} x \right) \]

![Diagram of a neural network with layers and activation functions]
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)
CAPACITY OF NEURAL NETWORK

**Topics:** single hidden layer neural network

(from Pascal Vincent’s slides)
Topics: universal approximation

• Universal approximation theorem (Hornik, 1991):
  > “a single hidden layer neural network with a linear output unit can approximate
    any continuous function arbitrarily well, given enough hidden units”

• The result applies for sigmoid, tanh and many other hidden layer activation functions

• This is a good result, but it doesn’t mean there is a learning algorithm that can find the necessary parameter values!
Demonstration of two layer NN

- http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
**NEURAL NETWORK**

**Topics:** parallel with the visual cortex

[picture from Simon Thorpe]
NEURAL NETWORK

Topics: parallel with the visual cortex

[picture from Simon Thorpe]
Topics: parallel with the visual cortex

[picture from Simon Thorpe]
Topics: parallel with the visual cortex
NEURAL NETWORK

Topics: parallel with the visual cortex

[picture from Simon Thorpe]
Topics: parallel with the visual cortex

[picture from Simon Thorpe]
Topics: parallel with the visual cortex

[picture from Simon Thorpe]
BIOLOGICAL NEURONS

Topics: synapse, axon, dendrite

- We estimate around $10^{10}$ and $10^{11}$ the number of neurons in the human brain:
  - they receive information from other neurons through their dendrites
  - the “process” the information in their cell body (soma)
  - they send information through a “cable” called an axon
  - the point of connection between the axon branches and other neurons’ dendrites are called synapses
BILOGICAL NEURONS

Topics: synapse, axon, dendrite

(from Hyvärinen, Hurri and Hoyer’s book)
**BIOLOGICAL NEURONS**

**Topics:** action potential, firing rate

- An action potential is an electrical impulse that travels through the axon:
  - this is how neurons communicate
  - it generates a “spike” in the electric potential (voltage) of the axon
  - an action potential is generated at neuron only if it receives enough (over some threshold) of the “right” pattern of spikes from other neurons

- Neurons can generate several such spikes every seconds:
  - the frequency of the spikes, called firing rate, is what characterizes the activity of a neuron
    - neurons are always firing a little bit, (spontaneous firing rate), but they will fire more, given the right stimulus
BIOLOGICAL NEURONS

Topics: action potential, firing rate

• Firing rates of different input neurons combine to influence the firing rate of other neurons:
  ‣ depending on the dendrite and axon, a neuron can either work to increase (excite) or decrease (inhibit) the firing rate of another neuron

• This is what artificial neurons approximate:
  ‣ the activation corresponds to a “sort of” firing rate
  ‣ the weights between neurons model whether neurons excite or inhibit each other
  ‣ the activation function and bias model the thresholded behavior of action potentials
How to train a neural network?

**Topics:** multilayer neural network

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

**Topics:** empirical risk minimization, regularization

- Empirical risk minimization
  - framework to design learning algorithms

\[
\arg \min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)
\]

- \( l(f(x^{(t)}; \theta), y^{(t)}) \) is a loss function
- \( \Omega(\theta) \) is a regularizer (penalizes certain values of \( \theta \))

- 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)
The Learning Algorithm

**Topics:** stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\theta$ \quad ($\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$)
  - for $N$ iterations
    - for each training example \quad $(x^{(t)}, y^{(t)})$
      \[ \sqrt{\Delta} = -\nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta) \]
      \[ \sqrt{\theta} \leftarrow \theta + \alpha \Delta \]
  
- To apply this algorithm to neural network training, we need
  - the loss function $l(f(x^{(t)}; \theta), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)})$
  - the regularizer $\Omega(\theta)$ (and the gradient $\nabla_{\theta} \Omega(\theta)$)
  - initialization method
**LOSS FUNCTION**

**Topics:** loss function for classification

- Neural network estimates $f(x)_c = p(y = c | x)$
  - we could maximize the probabilities of $y^{(t)}$ given $x^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood

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

  - 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 $\theta$ \quad ( $\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$ )
  - for N iterations
    - for each training example \quad $(x^{(t)}, y^{(t)})$
      \quad $\Delta = -\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_\theta \Omega(\theta)$
      \quad $\theta \leftarrow \theta + \alpha \Delta$
    - \quad \{ training epoch \} \quad \{ iteration over all examples \}

- To apply this algorithm to neural network training, we need
  - the loss function \quad $l(f(x^{(t)}; \theta), y^{(t)})$
  - a procedure to compute the parameter gradients \quad $\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)})$
  - the regularizer \quad $\Omega(\theta)$ \quad (and the gradient $\nabla_\theta \Omega(\theta)$)
  - 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)
Topics: loss gradient at output

- Partial derivative:
  \[
  \frac{\partial}{\partial f(x)_c} - \log f(x)_y = \frac{-1_{(y=c)}}{f(x)_y}
  \]

- Gradient:
  \[
  \nabla f(x) - \log f(x)_y \\
  = -\frac{1}{f(x)_y} \begin{bmatrix}
  1_{(y=0)} \\
  \vdots \\
  1_{(y=C-1)}
\end{bmatrix} \\
  = \frac{-e(y)}{f(x)_y}
  \]
**Topics:** loss gradient at output pre-activation

- Partial derivative:
  \[
  \frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y = -(1_{(y=c)} - f(x)_c)
  \]

- Gradient:
  \[
  \nabla a^{(L+1)}(x) \log f(x)_y = -(e(y) - f(x))
  \]
\frac{\partial}{\partial a^{(L+1)}(x)_c} - \log f(x)_y
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} - \log f(x)_y \\
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} - \log f(x)_y
\]
\[
= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y
\]
\[
= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} - \log f(x)_y
\]

\[
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y
\]

\[
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y
\]

\[
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})}
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y \\
-1 \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y \\
-1 \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y \\
-1 \frac{\partial}{\partial a^{(L+1)}(x)_c} \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \\
-1 \frac{\partial}{\partial a^{(L+1)}(x)_c} \left( \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right) - \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \left( \frac{\partial}{\partial a^{(L+1)}(x)_c} \sum_{c'} \exp(a^{(L+1)}(x)_{c'}) \right) \left( \sum_{c'} \exp(a^{(L+1)}(x)_{c'}) \right)^2
\]

\[
\frac{\partial g(x)}{\partial x} \frac{h(x)}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y \\
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y \\
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y \\
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \\
= \frac{-1}{f(x)_y} \left( \frac{\exp(a^{(L+1)}(x)_y) - \exp(a^{(L+1)}(x)_{c'}) \left( \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(x)_{c'}) \right)^2} \right) \\
= \frac{-1}{f(x)_y} \left( \frac{\delta_{(y=c)} \exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} - \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \frac{\exp(a^{(L+1)}(x)_c)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right)
\]
\[
\begin{align*}
\frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y &= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y \\
&= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y \\
&= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \\
&= -\frac{1}{f(x)_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(x)_c} \exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right) - \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \left( \sum_{c'} \exp(a^{(L+1)}(x)_{c'}) \right)^2 \\
&= -\frac{1}{f(x)_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} - \frac{\exp(a^{(L+1)}(x)_c)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right) \\
&= -\frac{1}{f(x)_y} \left( 1_{(y=c)} \text{softmax}(a^{(L+1)}(x))_y - \text{softmax}(a^{(L+1)}(x))_y \text{softmax}(a^{(L+1)}(x))_c \right)
\end{align*}
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y = -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y
\]

\[
= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax}(a^{(L+1)}(x))_y
\]

\[
= -\frac{1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})}
\]

\[
= -\frac{1}{f(x)_y} \left( \frac{1_\{y=c\} \exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} - \frac{\exp(a^{(L+1)}(x)_y)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \frac{\exp(a^{(L+1)}(x)_c)}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right)
\]

\[
= -\frac{1}{f(x)_y} \left( 1_\{y=c\} \text{softmax}(a^{(L+1)}(x))_y - \text{softmax}(a^{(L+1)}(x))_y \text{softmax}(a^{(L+1)}(x))_c \right)
\]

\[
= -\frac{1}{f(x)_y} \left( 1_\{y=c\} f(x)_y - f(x)_y f(x)_c \right)
\]
\[
\frac{\partial}{\partial a^{(L+1)}(x)_c} \log f(x)_y - \log f(x)_y = \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} f(x)_y = \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \text{softmax} (a^{(L+1)}(x))_y \\
= \frac{-1}{f(x)_y} \frac{\partial}{\partial a^{(L+1)}(x)_c} \exp(a^{(L+1)}(x)_y) \frac{1}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} = \frac{-1}{f(x)_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(x)_c} \exp(a^{(L+1)}(x)_y) \sum_{c'} \exp(a^{(L+1)}(x)_{c'})}{\exp(a^{(L+1)}(x)_y) \sum_{c'} \exp(a^{(L+1)}(x)_{c'})} - \frac{\exp(a^{(L+1)}(x)_y) \sum_{c'} \exp(a^{(L+1)}(x)_{c'})}{\left( \sum_{c'} \exp(a^{(L+1)}(x)_{c'}) \right)^2} \right) \\
= \frac{-1}{f(x)_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(x)_y) \sum_{c'} \exp(a^{(L+1)}(x)_{c'})}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} - \frac{\exp(a^{(L+1)}(x)_y) \sum_{c'} \exp(a^{(L+1)}(x)_{c'})}{\sum_{c'} \exp(a^{(L+1)}(x)_{c'})} \right) = \frac{-1}{f(x)_y} \left( 1_{(y=c)} \text{softmax}(a^{(L+1)}(x))_y - \text{softmax}(a^{(L+1)}(x))_y \text{softmax}(a^{(L+1)}(x))_c \right) \\
= \frac{-1}{f(x)_y} \left( 1_{(y=c)} f(x)_y - f(x)_y f(x)_c \right) = \frac{-1}{f(x)_y} \left( 1_{(y=c)} - f(x)_c \right)
\]
Gradient Computation

- Output layer gradient (o)
- Hidden layer gradient (h)
- Activation function gradient (a)
- Parameter gradient (W, b)
Topics: loss gradient at hidden layer

... this is getting complicated!!
**Topics:** chain rule

- If a function $p(a)$ can be written as a function of intermediate results $q_i(a)$ then we have:

$$\frac{\partial p(a)}{\partial a} = \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a}$$

- We can invoke it by setting
  - $a$ to a unit in layer
  - $q_i(a)$ to a pre-activation in the layer above
  - $p(a)$ is the loss function
Topics: loss gradient at hidden layers

- Partial derivative:

\[
\frac{\partial}{\partial h^{(k)}(x)_j} \log f(x)_y
\]

= \sum_i \frac{\partial - \log f(x)_y}{\partial a^{(k+1)}(x)_i} \frac{\partial a^{(k+1)}(x)_i}{\partial h^{(k)}(x)_j}

= \sum_i \frac{\partial - \log f(x)_y}{\partial a^{(k+1)}(x)_i} W^{(k+1)}_{i,j}

= (W^{k+1})^T (\nabla a^{k+1}(x) - \log f(x)_y)

Reminder:

\[a^{(k)}(x)_i = b^{(k)}_i + \sum_j W^{(k)}_{i,j} h^{(k-1)}(x)_j\]
**Topics:** loss gradient at hidden layers

- Gradient:
  \[
  \nabla h^{(k)}(x) = \log f(x)_y
  \]
  \[
  = W^{(k+1)T} \left( \nabla a^{(k+1)}(x) - \log f(x)_y \right)
  \]

**REMEMBER**
\[
a^{(k)}(x)_i = b^{(k)}_i + \sum_j W^{(k)}_{i,j} h^{(k-1)}(x)_j
\]
Topics: loss gradient at hidden layers pre-activation

- Partial derivative:

\[
\frac{\partial}{\partial a^{(k)}(x)_j} l(f(x), y) - \log f(x)_y = \frac{\partial}{\partial h^{(k)}(x)_j} \frac{\partial h^{(k)}(x)_j}{\partial a^{(k)}(x)_j} \frac{\partial h^{(k)}(x)_j}{\partial h^{(k)}(x)_j}
\]

\[
= \frac{\partial}{\partial h^{(k)}(x)_j} \log f(x)_y g'(a^{(k)}(x)_j)
\]

Reminder:

\[h^{(k)}(x)_j = g(a^{(k)}(x)_j)\]
Topics: loss gradient at hidden layers pre-activation

- Gradient:
  \[ \nabla_{a^{(k)}(x)} \log f(x)_y = (\nabla_{h^{(k)}(x)} \log f(x)_y)^T \nabla_{a^{(k)}(x)} h^{(k)}(x) \]
  \[ \nabla_{h^{(k)}(x)} \log f(x)_y \odot [\ldots, g'(a^{(k)}(x)_j), \ldots] \]

Reminder:
\[ h^{(k)}(x)_j = g(a^{(k)}(x)_j) \]
Gradient Computation

• Output layer gradient (o)

• Hidden layer gradient (h)

• Activation function gradient (a)

• Parameter gradient (W, b)
ACTIVATION FUNCTION

Topics: linear activation function gradient

- Partial derivative:
  \[ g'(a) = 1 \]

\[ g(a) = a \]
**Topics:** sigmoid activation function gradient

- Partial derivative:

  \[ g'(a) = g(a)(1 - g(a)) \]

- Sigmoid function:

  \[ g(a) = \text{sigm}(a) = \frac{1}{1 + \exp(-a)} \]
ACTIVATION FUNCTION

**Topics:** tanh activation function gradient

- Partial derivative:
  \[ g'(a) = 1 - g(a)^2 \]

\[
g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}
\]
Some Common Activation Functions

- $g_{\text{linear}}(z)$
- $g_{\text{logistic}}(z)$
- $g_{\text{tanh}}(z)$

Activation Function Derivatives

- $g'_{\text{linear}}(z)$
- $g'_{\text{logistic}}(z)$
- $g'_{\text{tanh}}(z)$
Gradient Computation

• Output layer gradient (o)

• Hidden layer gradient (h)

• Activation function gradient (a)

• Parameter gradient (W, b)
GRADIENT COMPUTATION

Topics: loss gradient of parameters

- Partial derivative (weights):

\[
\frac{\partial}{\partial W_{i,j}^{(k)}} - \log f(x)_y \\
= \frac{\partial}{\partial a^{(k)}(x)_i} \log f(x)_y \frac{\partial a^{(k)}(x)_i}{\partial W_{i,j}^{(k)}} \\
= \frac{\partial}{\partial a^{(k)}(x)_i} \log f(x)_y a^{(k-1)}_j(x)
\]

REMINDER

\[ a^{(k)}(x)_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} a^{(k-1)}(x)_j \]
Topics: loss gradient of parameters

- Gradient (weights):
  \[ \nabla_{W^{(k)}} - \log f(x)_y = (\nabla_{a^{(k)}(x)} - \log f(x)_y) \cdot h^{(k-1)}(x)^T \]

REMINDER
\[ a^{(k)}(x)_i = b^{(k)}_i + \sum_j W^{(k)}_{i,j} h^{(k-1)}(x)_j \]
**Topics:** loss gradient of parameters

- Partial derivative (biases):

\[
\frac{\partial}{\partial b_i^{(k)}} - \log f(x)_y = \frac{\partial - \log f(x)_y}{\partial a^{(k)}(x)_i} \frac{\partial a^{(k)}(x)_i}{\partial b_i^{(k)}}
\]

\[
= \frac{\partial - \log f(x)_y}{\partial a^{(k)}(x)_i}
\]

**Reminder**

\[
a^{(k)}(x)_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(x)_j
\]
**Topics:** loss gradient of parameters

- Gradient (biases):

\[
\nabla b^{(k)} - \log f(x)_y = \nabla a^{(k)}(x) - \log f(x)_y
\]

**Reminder**

\[
a^{(k)}(x)_i = b^{(k)}_i + \sum_j W^{(k)}_{i,j} h^{(k-1)}(x)_j
\]
Backpropagation
**Topics:** flow graph

- Forward propagation can be represented as an acyclic flow graph
- It's a nice way of implementing forward propagation in a modular way
  - each box could be an object with an fprop method, that computes the value of the box given its children
  - calling the fprop method of each box in the right order yield forward propagation
# 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)
**Topics:** automatic differentiation

- Each object also has a `bprop` method
  - it computes the gradient of the loss with respect to each children
  - `fprop` depends on the `fprop` of a box’s children, while `bprop` depends the `bprop` of a box’s parents
- By calling `bprop` in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each child
  - fprop depends on the fprop of a box's children, while bprop depends on the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each child
  - fprop depends on the fprop of a box’s children, while bprop depends on the bprop of a box’s parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

• Each object also has a bprop method
  ‣ it computes the gradient of the loss with respect to each children
  ‣ fprop depends on the fprop of a box’s children, while bprop depends the bprop of a box’s parents

• By calling bprop in the reverse order, we get backpropagation
  ‣ only need to reach the parameters
**Topics:** automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each child
  - fprop depends on the fprop of a box's children, while bprop depends on the bprop of a box's parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
Topics: automatic differentiation

- Each object also has a bprop method
  - it computes the gradient of the loss with respect to each children
  - fprop depends on the fprop of a box's children, while bprop depends on the bprop of a box's parents

- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters
BACKPROPAGATION

Topics: backpropagation algorithm

• This assumes a forward propagation has been made before
  
  ‣ compute output gradient (before activation)

  \[ \nabla_a^{(L+1)}(x) - \log f(x)_y \quad \iff \quad - (e(y) - f(x)) \nabla_a^{(L+1)}(x) - \log f(x)_y \quad \iff \quad - (e(y) - f(x)) \]

  ‣ for \( k \) from \( L+1 \) to \( 1 \)

  - compute gradients of hidden layer parameter

  \[ \nabla_W^{(k)} - \log f(x)_y \quad \iff \quad (\nabla_a^{(k+1)}(x) - \log f(x)_y) \cdot h^{(k-1)}(x)^T \]

  \[ \nabla_b^{(k)} - \log f(x)_y \quad \iff \quad \nabla_a^{(k)}(x) - \log f(x)_y \]

  - compute gradient of hidden layer below

  \[ \nabla_h^{(k-1)}(x) - \log f(x)_y \quad \iff \quad W^{(k)} \cdot (\nabla_a^{(k)}(x) - \log f(x)_y) \]

  - compute gradient of hidden layer below (before activation)

  \[ \nabla_a^{(k-1)}(x) - \log f(x)_y \quad \iff \quad (\nabla_h^{(k-1)}(x) - \log f(x)_y) \odot [\ldots, g'(a^{(k-1)}(x)_j), \ldots] \]
Total error on training set vs. Number of epochs
3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error
The Learning Algorithm

**Topics:** stochastic gradient descent (SGD)

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

- To apply this algorithm to neural network training, we need
  - the loss function $l(f(x^{(t)}; \theta), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)})$
  - the regularizer $\Omega(\theta)$ (and the gradient $\nabla_{\theta} \Omega(\theta)$)
  - initialization method
**Topics:** L2 regularization

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

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

Topics: L1 regularization

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

- Gradient: \( \nabla_{W^{(k)}} \Omega(\theta) = \text{sign}(W^{(k)}) \)
  - where \( \text{sign}(W^{(k)})_{i,j} = 1_{W_{i,j}^{(k)}>0} - 1_{W_{i,j}^{(k)}<0} \)

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

  - $l(f(x^{(t)}; \theta), y^{(t)})$ is a loss function
  - $\Omega(\theta)$ is a regularizer (penalizes certain values of $\theta$)

- 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)
The Learning Algorithm

**Topics:** stochastic gradient descent (SGD)

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

- To apply this algorithm to neural network training, we need
  - the loss function $l(f(x^{(t)}; \theta), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)})$
  - the regularizer $\Omega(\theta)$ (and the gradient $\nabla_\theta \Omega(\theta)$ )
  - initialization method
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 $W^{(k)}_{i,j}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
    - 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 $\theta$  
    $\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$
  - for N iterations
    - for each training example $(x^{(t)}, y^{(t)})$
      $$\Delta = -\nabla_{\theta} l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_{\theta} \Omega(\theta)$$
      $$\theta \leftarrow \theta + \alpha \Delta$$
  - iteration over all examples
  - training epoch

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

• TensorFlow
  – https://www.tensorflow.org/
  – Python and C++

• Theano
  – http://deeplearning.net/software/theano/
  – Python

• Torch
  – http://torch.ch/
  – LuaJIT
Unsupervised Learning with Neural Networks

• Unsupervised learning: only use the inputs for learning
  – automatically extract meaningful features for your data
  – leverage the availability of unlabeled data
  – add a data-dependent regularizer to training

$(-\log p(x^{(t)}))$
Unsupervised Learning with Neural Networks

- Restricted Boltzmann machines
RESTRICTED BOLTZMANN MACHINE

Topics: RBM, visible layer, hidden layer, energy function

Energy function: 
\[ E(x, h) = -h^T W x - c^T x - b^T h \]
\[ = - \sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \]

Distribution: 
\[ p(x, h) = \exp(-E(x, h))/Z \]
**EXAMPLE OF DATA SET: MNIST**

<table>
<thead>
<tr>
<th>3</th>
<th>8</th>
<th>6</th>
<th>9</th>
<th>6</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>8</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>3</th>
<th>8</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>9</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
FILTERS
(LAROCHELLE ET AL., JMLR2009)
**Topics:** RBM, visible layer, hidden layer, energy function

Energy function: \[ E(x, h) = -h^T W x - c^T x - b^T h \]

\[ = -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \]

Distribution: \[ p(x, h) = \exp(-E(x, h))/Z \]

**RESTRICTED BOLTZMANN MACHINE**
Inference

• Conditional distributions: $P(h|x)$, $P(x|h)$

• Sample distribution: $P(x)$
**Topics:** conditional distributions

- **p(h|x):**
  \[ p(h|x) = \prod p(h_j|x) \]
  \[ p(h_j = 1|x) = \frac{1}{1 + \exp(-(b_j + W_{j.x}))} \]
  \[ = \text{sigm}(b_j + W_{j.x}) \]

- **p(x|h):**
  \[ p(x|h) = \prod p(x_k|h) \]
  \[ p(x_k = 1|h) = \frac{1}{1 + \exp(-(c_k + h^\top W_{.k}))} \]
  \[ = \text{sigm}(c_k + h^\top W_{.k}) \]

*j*th row of \(W\)

*k*th column of \(W\)
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]

\[ = \frac{\exp(h^T W x + c^T x + b^T h)/Z}{\sum_{h' \in \{0, 1\}^H} \exp(h'^T W x + c^T x + b^T h')/Z} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]
\[ = \frac{\exp(h^T W x + e^T x + b^T h)/Z}{\sum_{h' \in \{0,1\}^H} \exp(h'^T W x + e^T x + b^T h')/Z} \]
$$p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')}$$

$$= \frac{\exp(h^\top W x + e^\top x + b^\top h)/Z}{\sum_{h'\in\{0,1\}^H} \exp(h'^\top W x + e^\top x + b^\top h')/Z}$$

$$= \frac{\exp(\sum_{j} h_j W_j . x + b_j h_j)}{\sum_{h'_1\in\{0,1\}} \cdots \sum_{h'_H\in\{0,1\}} \exp(\sum_{j} h'_j W_j . x + b_j h'_j)}$$
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]

\[ = \frac{\exp(h^\top W x + e^\top x + b^\top h)/Z}{\sum_{h'\in\{0,1\}^H} \exp(h'^\top W x + e^\top x + b^\top h')/Z} \]

\[ = \frac{\exp(\sum_j h_j W_j x + b_j h_j)}{\sum_{h'_1\in\{0,1\}} \cdots \sum_{h'_H\in\{0,1\}} \exp(\sum_j h'_j W_j x + b_j h'_j)} \]

\[ = \frac{\prod_j \exp(h_j W_j x + b_j h_j)}{\sum_{h'_1\in\{0,1\}} \cdots \sum_{h'_H\in\{0,1\}} \prod_j \exp(h'_j W_j x + b_j h'_j)} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]
\[ = \frac{\exp(h^T W x + e^T x + b^T h)}{\sum_{h' \in \{0,1\}^H} \exp(h'^T W x + e^T x + b^T h')/\mathcal{Z}} \]
\[ = \frac{\exp(\sum_j h_j W_j x + b_j h_j)}{\sum_{h_1' \in \{0,1\}} \cdots \sum_{h_H' \in \{0,1\}} \exp(\sum_j h_j' W_j x + b_j h_j')} \]
\[ = \frac{\prod_j \exp(h_j W_j x + b_j h_j)}{\left(\sum_{h_1' \in \{0,1\}} \exp(h_1' W_{1} x + b_1 h_1')\right) \cdots \left(\sum_{h_H' \in \{0,1\}} \exp(h_H' W_{H} x + b_H h_H')\right)} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]

\[ = \frac{\exp(h^T W x + e^T x + b^T h)/Z}{\sum_{h' \in \{0,1\}^H} \exp(h'^T W x + e^T x + b^T h')/Z} \]

\[ = \frac{\exp(\sum_j h_j W_j x + b_j h_j)}{\sum_{h'_1 \in \{0,1\}} \cdots \sum_{h'_H \in \{0,1\}} \exp(\sum_j h'_j W_j x + b_j h'_j) \prod_j \exp(h_j W_j x + b_j h_j)} \]

\[ = \frac{\prod_j \exp(h_j W_j x + b_j h_j)}{\left(\sum_{h'_1 \in \{0,1\}} \exp(h'_1 W_1 x + b_1 h'_1)\right) \cdots \left(\sum_{h'_H \in \{0,1\}} \exp(h'_H W_H x + b_H h'_H)\right)} \]

\[ = \frac{\prod_j \exp(h_j W_j x + b_j h_j)}{\prod_j \left(\sum_{h'_j \in \{0,1\}} \exp(h'_j W_j x + b_j h'_j)\right)} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} \]

\[ = \frac{\exp(h^T W x + e^T x + b^T h) / Z}{\sum_{h' \in \{0,1\}^H} \exp(h'^T W x + e^T x + b^T h') / Z} \]

\[ = \frac{\exp(\sum_{j} h_j W_j x + b_j h_j)}{\sum_{h'_1 \in \{0,1\}} \cdots \sum_{h'_H \in \{0,1\}} \exp(\sum_{j} h'_j W_j x + b_j h'_j)} \]

\[ = \frac{\exp(\sum_{j} h_j W_j x + b_j h_j)}{\prod_j \exp(h'_j W_j x + b_j h'_j)} \]

\[ = \frac{\exp(\sum_{j} h'_j W_j x + b_j h'_j)}{\prod_j (1 + \exp(b_j + W_j x))} \]
\[ p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')} = \frac{\exp(h^T W x + e^T x + b^T h) / Z}{\sum_{h' \in \{0,1\}^H} \exp(h'^T W x + e^T x + b^T h') / Z} = \frac{\exp(\sum_j h_j W_j . x + b_j h_j)}{\sum_{h'_1 \in \{0,1\}} \cdots \sum_{h'_H \in \{0,1\}} \prod_j \exp(h'_j W_j . x + b_j h'_j)} = \frac{\prod_j \exp(h_j W_j . x + b_j h_j)}{\prod_j (\sum_{h'_j \in \{0,1\}} \exp(h'_j W_j . x + b_j h'_j))} = \frac{\prod_j \exp(h_j W_j . x + b_j h_j)}{\prod_j (1 + \exp(b_j + W_j . x))} = \prod_j \frac{\exp(h_j W_j . x + b_j h_j)}{1 + \exp(b_j + W_j . x)} = \prod_j p(h_j | x) \]
\[ p(h_j = 1 | x) = \frac{\exp(b_j + W_j \cdot x)}{1 + \exp(b_j + W_j \cdot x)} \]
\[ p(h_j = 1|x) = \frac{\exp(b_j + W_j.x)}{1 + \exp(b_j + W_j.x)} \]

\[ = \frac{1}{1 + \exp(-b_j - W_j.x)} \]

\[ = \text{sigm}(b_j + W_j.x) \]
Inference

• Conditional distributions: $P(h|x), P(x|h)$

• Sample distribution: $P(x)$
RESTRICTED BOLTZMANN MACHINE

Topics: RBM, visible layer, hidden layer, energy function

Energy function: \[ E(x, h) = -h^T W x - c^T x - b^T h \]
\[ = - \sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \]

Distribution: \[ p(x, h) = \exp(-E(x, h))/Z \]

partition function (intractable)
\textbf{FREE Energy}

\textbf{Topics:} free energy

- What about $p(x)$?

\[
p(x) = \sum_{h \in \{0,1\}^H} p(x, h) = \sum_{h \in \{0,1\}^H} \exp(-E(x, h))/Z
\]

\[
= \exp \left( c^\top x + \sum_{j=1}^H \log(1 + \exp(b_j + W_{j,x})) \right)/Z
\]

\[
= \exp(-F(x))/Z
\]

free energy
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp(h^T W x + c^T x + b^T h) / Z \]
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp(h^T W x + c^T x + b^T h) / Z \]

\[ = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right) / Z \]
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp(h^\top W x + c^\top x + b^\top h) / Z \]

\[ = \exp(c^\top x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_{j} h_j W_j x + b_j h_j \right) / Z \]

\[ = \exp(c^\top x) \left( \sum_{h_1 \in \{0,1\}} \exp(h_1 W_1 x + b_1 h_1) \right) \cdots \left( \sum_{h_H \in \{0,1\}} \exp(h_H W_H x + b_H h_H) \right) / Z \]
\[
p(x) = \sum_{h \in \{0,1\}^H} \exp(h^T W x + c^T x + b^T h) / Z
\]

\[
= \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right) / Z
\]

\[
= \exp(c^T x) \left( \sum_{h_1 \in \{0,1\}} \exp(h_1 W_1 x + b_1 h_1) \right) \cdots \left( \sum_{h_H \in \{0,1\}} \exp(h_H W_H x + b_H h_H) \right) / Z
\]

\[
= \exp(c^T x) (1 + \exp(b_1 + W_1 x)) \cdots (1 + \exp(b_H + W_H x)) / Z
\]
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp(h^T W x + c^T x + b^T h) / Z \]

\[ = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \exp \left( \sum_j h_j W_j . x + b_j h_j \right) / Z \]

\[ = \exp(c^T x) \left( \sum_{h_1 \in \{0,1\}} \exp(h_1 W_1 . x + b_1 h_1) \right) \cdots \left( \sum_{h_H \in \{0,1\}} \exp(h_H W_H . x + b_H h_H) \right) / Z \]

\[ = \exp(c^T x) (1 + \exp(b_1 + W_1 . x)) \ldots (1 + \exp(b_H + W_H . x)) / Z \]

\[ = \exp(c^T x) \exp(\log(1 + \exp(b_1 + W_1 . x))) \ldots \exp(\log(1 + \exp(b_H + W_H . x))) / Z \]
\[ p(x) = \sum_{h \in \{0,1\}^H} \exp(h^T W x + c^T x + b^T h)/Z \]

\[ = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \ldots \sum_{h_H \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right) /Z \]

\[ = \exp(c^T x) \left( \sum_{h_1 \in \{0,1\}} \exp(h_1 W_1 x + b_1 h_1) \right) \ldots \left( \sum_{h_H \in \{0,1\}} \exp(h_H W_H x + b_H h_H) \right) /Z \]

\[ = \exp(c^T x) \left( 1 + \exp(b_1 + W_1 x) \right) \ldots \left( 1 + \exp(b_H + W_H x) \right) /Z \]

\[ = \exp(c^T x) \exp(\log(1 + \exp(b_1 + W_1 x))) \ldots \exp(\log(1 + \exp(b_H + W_H x))) /Z \]

\[ = \exp \left( c^T x + \sum_{j=1}^H \log(1 + \exp(b_j + W_j x)) \right) /Z \]
RESTRICTED BOLTZMANN MACHINE

Topics: free energy

\[ h \ p(x) = \exp \left( c^T x + \sum_{j=1}^{H} \log(1 + \exp(b_j + W_j . x)) \right) / Z \]

\[ = \exp \left( c^T x + \sum_{j=1}^{H} \text{softplus}(b_j + W_j . x) \right) / Z \]
RESTRICTED BOLTZMANN MACHINE

Topics: free energy

\[ p(x) = \exp \left( c^T x + \sum_{j=1}^{H} \log(1 + \exp(b_j + W_j x)) \right) / Z \]

\[ = \exp \left( c^T x + \sum_{j=1}^{H} \text{softplus}(b_j + W_j x) \right) / Z \]

"feature" expected in \( x \)

bias of each feature

bias the prob of each \( x_i \)
Useful Resources for Deep Learning

• Book for deep learning
  – Deep Learning
  – By Ian Goodfellow, Yoshua Bengio, and Aaron Courville
  – http://www.deeplearningbook.org/
Useful Resources for Deep Learning

• Tutorials
  – General
  – Natural language processing
  – Computer Vision
    • https://sites.google.com/site/deeplearningcvpr2014/
What we learned today

• Feedforward neural network

• Training neural networks

• Restricted Boltzmann machine
Homework

• Read Murphy CH 16.5 and CH 28.
• Read slides from Hugo Larochelle.
  – http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html