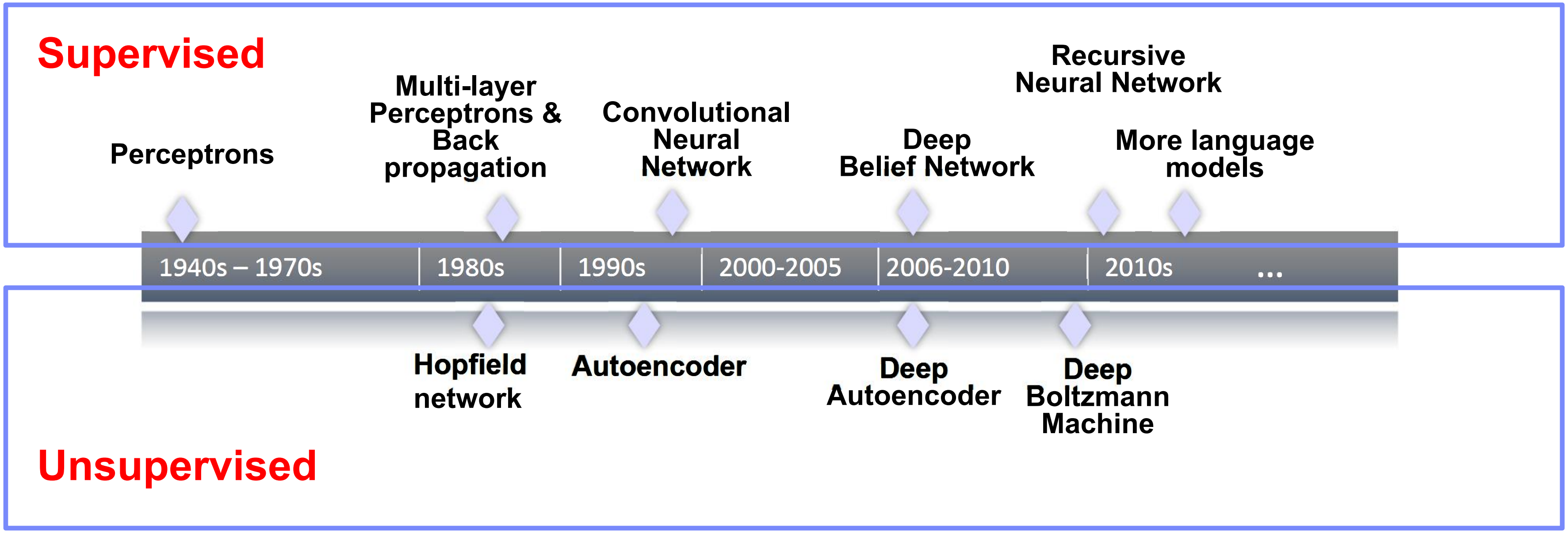
CS291K – Advanced Data Mining

Instructor: Xifeng Yan
Computer Science
University of California at Santa Barbara

Training Neural Networks

Lecturer : Fangqiu Han
Computer Science
University of California at Santa Barbara

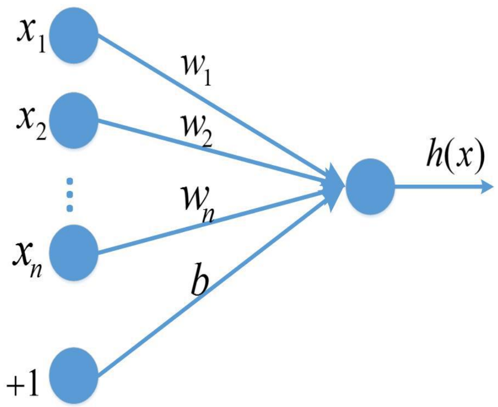
Neural network timeline



Outline

- Trainging Perceptron
- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Perceptron: the simplest neural network



x: n-dimension input
w: combination weights
b: bias

$$h(x) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(w^T x + b)$$

$f(\cdot)$ is called Activation function, e.g.,

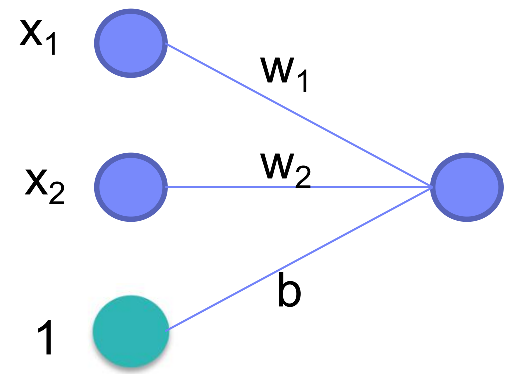
Step function: $$f(z) = \begin{cases} +1 & if\ z > 0 \\ -1 & otherwise \end{cases}$$

Training Perceptron

- Initialize all parameters to 0.
- Pick training data using any policy that ensures that all training data will keep getting picked.

Perceptron Training Example

- Task: Train a classifier using perceptron on two data points:
  - (1,1) with label 1; (2,-3) with label -1.
- Initialize w_1=w_2=b=0.
- Apply a round-robin order on data points.



$$h(x) = f(w^T x + b)$$

$$f(z) = \begin{cases} +1 & \text{if } z>0 \\ -1 & \text{otherwise} \end{cases}$$

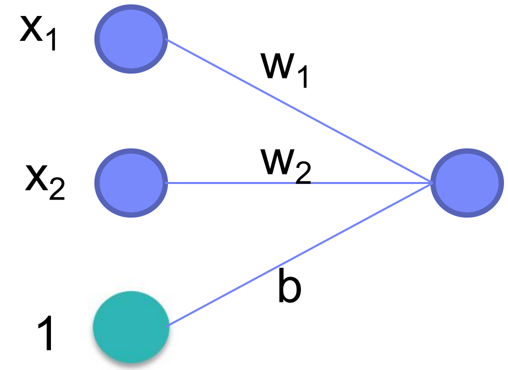
Training Perceptron

- Initialize all parameters to 0.

- Pick training cases using any policy that ensures that every training case will keep getting picked.
  - If the output unit is correct, leave its weights alone.
  - If the output unit incorrectly outputs a -1, add the input vector to the weight vector: w += x
  - If the output unit incorrectly outputs an 1, subtract the input vector from the weight vector: w -= x

Perceptron Training Example

- Task: Train a classifier using perceptron on two data points:
  - (1,1) with label 1; (2,-3) with label -1.
- Initialize $w_1=w_2=b=0$.
- Apply a round-robin order on data points:
  - 1. data: (1,1), $f(w^Tx+b)=-1$. Update $w_1=w_2=b=1$.
  - 2. data: (3,-2), $f(w^Tx+b)=1$. Update $w_1=-1$, $w_2=3$, $b=0$.
  - 3. data: (1,1), $f(w^Tx+b)=1$. Do nothing.
  - 4. data: (3,-2), $f(w^Tx+b)=-1$. Do nothing.
  - Converged.



$$h(x) = f(w^T x + b)$$

$$f(z) = \begin{cases} +1 & \text{if } z>0 \\ -1 & \text{otherwise} \end{cases}$$

Training Perceptron

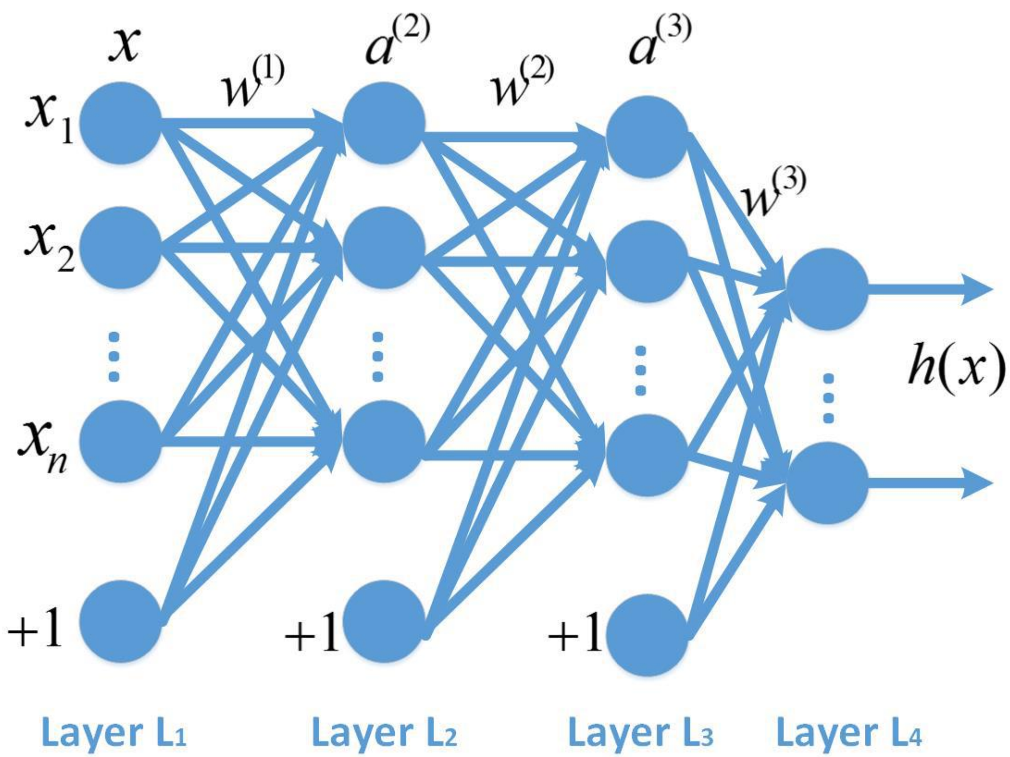
- Initialize all parameters to 0.

- Pick training cases using any policy that ensures that every training case will keep getting picked.
  - If the output unit is correct, leave its weights alone.
  - If the output unit incorrectly outputs a -1, add the input vector to the weight vector: w += x
  - If the output unit incorrectly outputs an 1, subtract the input vector from the weight vector: w -= x

- This is guaranteed to find a set of weights that gets the right answer for all the training cases if any such set exists.

Multi-layer Perceptrons

- Second generation (1980s)



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Parameters { $w^{(1)}$, $w^{(2)}$, $w^{(3)}$, $b^{(1)}$, $b^{(2)}$, $b^{(3)}$ } to be learnt.

Parameter training

Preprocess the data?

- A training set of m data points, $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(\mathrm{m})}, y^{(\mathrm{m})})\}$

- Objective function

Activation functions? Regularization?

$$\min \ \mathrm{H} = \frac{1}{2m}\sum_{i=1}^{m}\left\| h(x^{(i)}) - y^{(i)} \right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\left\| w^{(l)} \right\|_F^2$$

where,

$\frac{1}{2m}\sum_{i=1}^{m}\left\| h(x^{(i)}) - y^{(i)} \right\|^2$: average sum-of-squares error term

$\frac{\lambda}{2}\sum_{l=1}^{L}\left\| w^{(l)} \right\|_F^2$: weight decay term; L : the number of layers

Parameter training

- Task:

  Find w, b minimize $E = [\mathrm{t} - h(\mathrm{x})]^2$

- Algorithm:

  1. Initialize: w, b   How to initialize?
  2. For data x and label t

     Predict the label of x: $y = f(w^T x + b)$

     Update the parameters by gradient descent:

     $w \leftarrow w - \eta\,(\nabla_w E)$ and $b \leftarrow b - \eta\,(\nabla_b E)$

     where $E = [\mathrm{t} - h(\mathrm{x})]^2$ How to compute gradient?
  3. Repeat until convergence

How to update parameters?

Outline

- Trainging Perceptron
- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Optimization algorithm

- Gradient descent

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \alpha \frac{\partial H}{\partial w_{ij}^{(l)}}$$

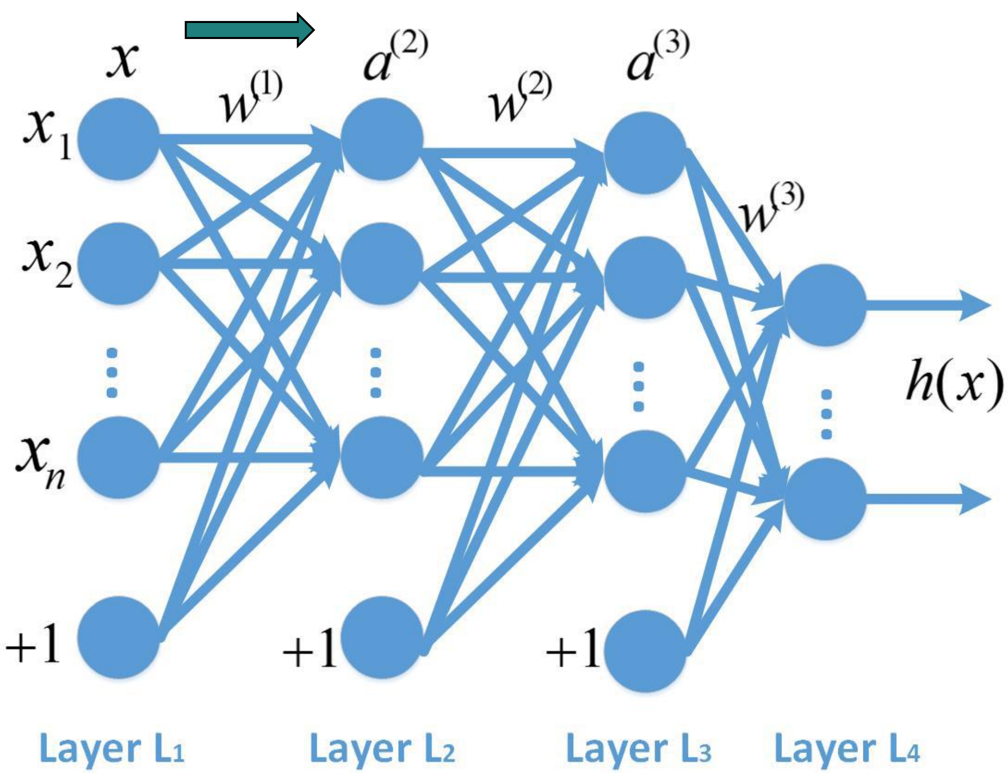
$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial H}{\partial b_i^{(l)}}$$

- Backpropagation algorithm: a systematic way

  to compute $\frac{\partial H}{\partial w_{ij}^{(l)}}$ and $\frac{\partial H}{\partial b_i^{(l)}}$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers L*2*, L*3*, and so on up to the output layer h(x).



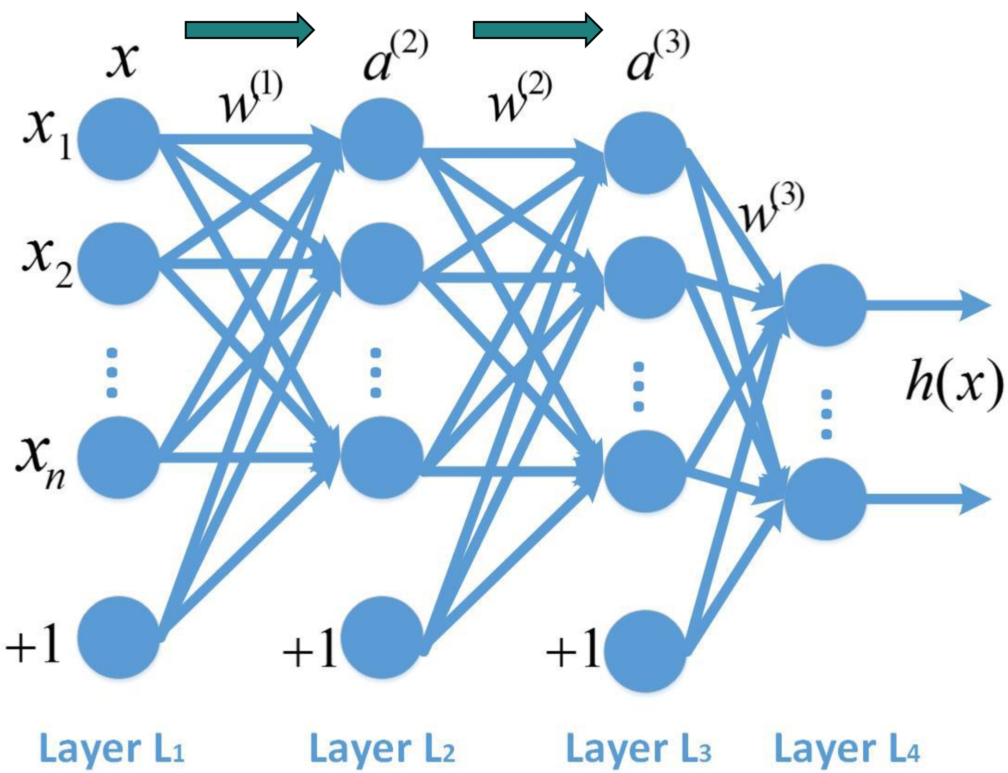
Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers *L2*, *L3*, and so on up to the output layer *h(x)*.



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

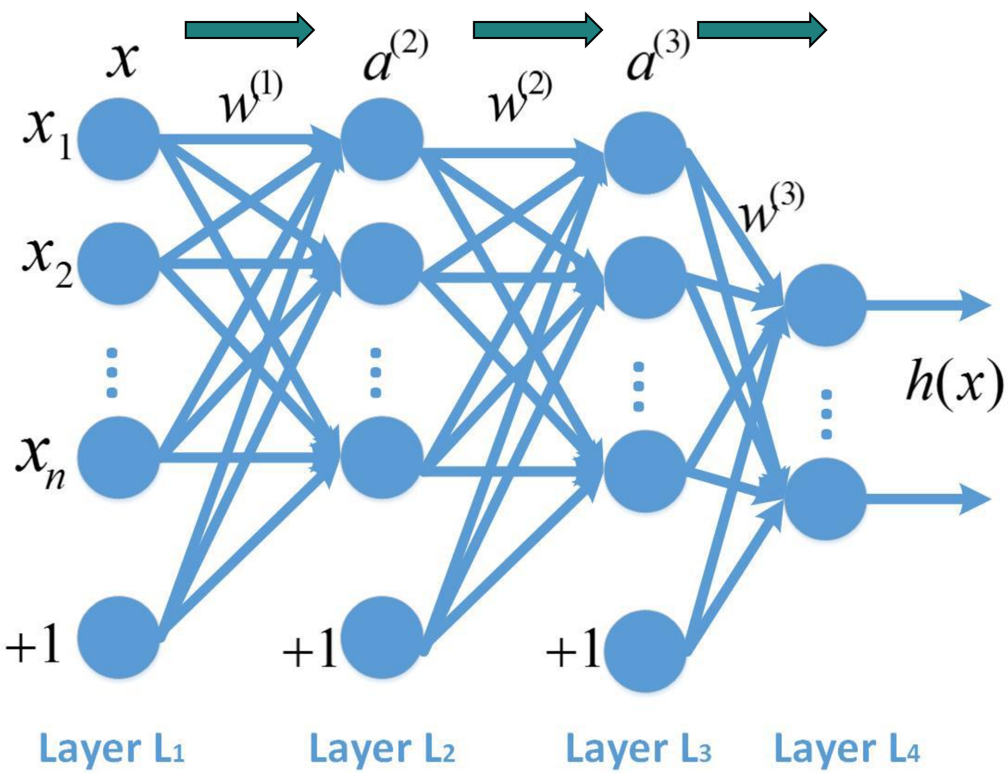
Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Backpropagation

- Perform a **feedforward pass**, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.



Input and output of 2nd layer:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Input and output of 3rd layer:

$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Output layer:

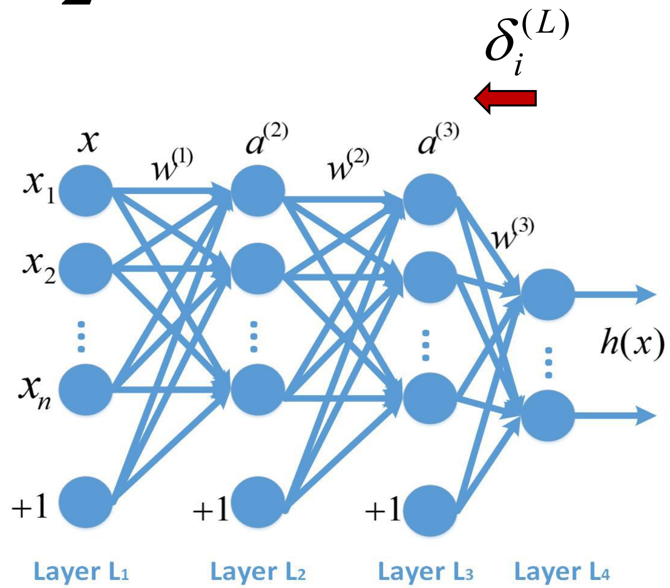
$$h(x) = f(w^{(3)}a^{(3)} + \mathrm{b}^{(3)})$$

Backpropagation

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.

- For each output unit i in the output layer, set

$$\delta_i^{(L)} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(L)}) \cdot f'(z_i^{(L)})$$



Chain rule

$$\delta_i^{(L)} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(L)}) \cdot f'(z_i^{(L)})$$

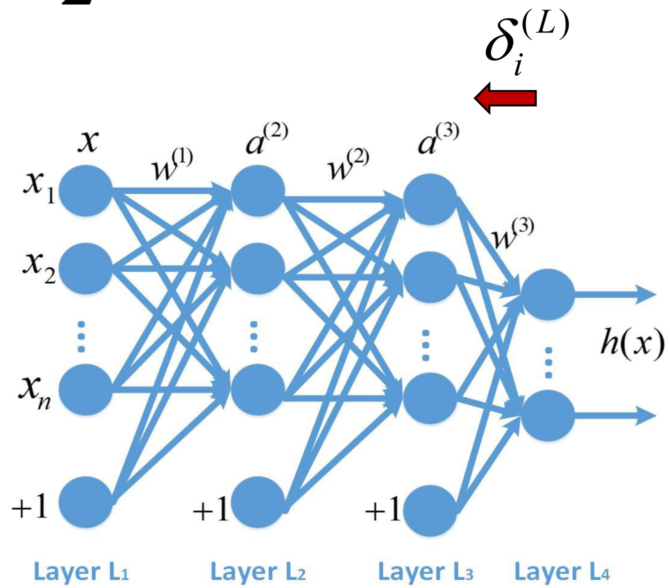
$$\delta_i^L = \frac{\partial ||y - h(x)||^2}{2\, \partial z_i^L} = \frac{\partial ||y - h(x)||^2}{2\, \partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L}$$

Backpropagation

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.

- For each output unit i in the output layer, set

$$\delta_i^{(L)} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(L)}) \cdot f'(z_i^{(L)})$$



Backpropagation

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.

- For each output unit i in the output layer, set

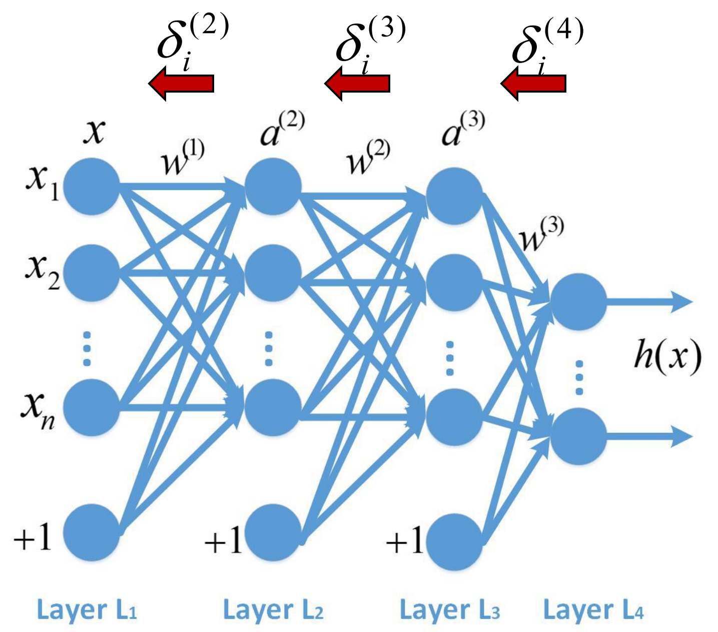
$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

- For $l = n_l - 1, n_l - 2, \ldots, 2$

  For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l + 1} w_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

H\h N\h N\H



Chain rule

$$\delta_i = \frac{\partial E}{\partial z_i}$$

$$z_{i+1} = w_i f(z_i) + b_i$$

$$\delta_i = \frac{\partial E}{\partial z_i} = \frac{\partial E}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i} = w_i \delta_{i+1} f'(z_i)$$

Backpropagation

- Perform a feedforward pass, computing the activations for layers *L2*, *L3*, and so on up to the output layer *h(x)*.

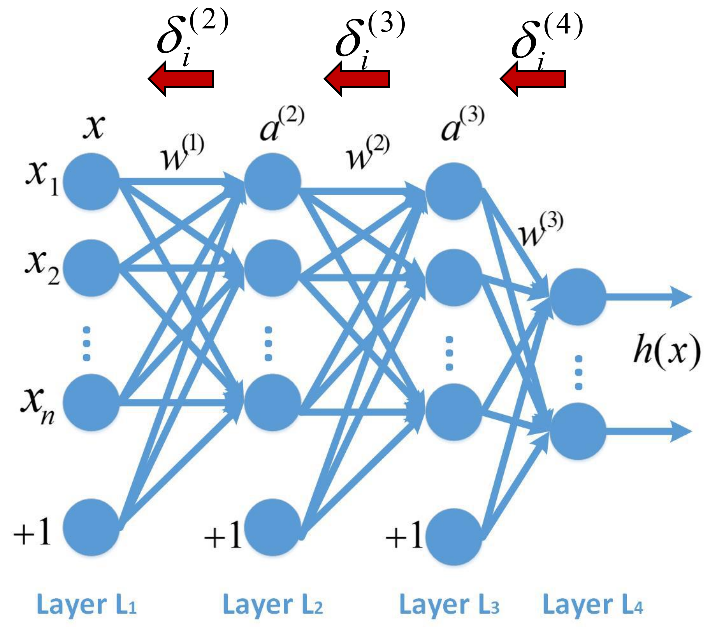
- For each output unit *i* in the output layer, set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

- For $l = n_l - 1, n_l - 2, \ldots, 2$

  For each node *i* in layer *l*, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

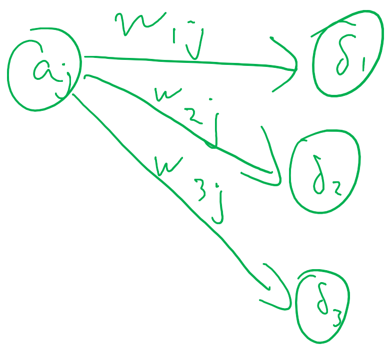


Backpropagation

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.

- For each output unit i in the output layer, set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$



- For $l = n_l - 1, n_l - 2, \dots, 2$

  For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

- Compute the partial derivatives in each layer,

$$\frac{\partial H}{\partial w_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} + \lambda \cdot w_{ij}^{(l)} \quad ; \quad \frac{\partial H}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

Gradient Checking (important!)

- Definition of derivative

  For function $J(\theta)$ with parameter θ

$$\frac{d}{d\theta}J(\theta)=\lim_{\varepsilon\to 0}\frac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon}$$

- Comparison

$$\frac{\|A-B\|_F}{\|A+B\|_F}\leq\delta$$

Where, A are the derivatives obtained by backpropagation; B are those obtained by definition; δ, usually, $\leq 10^{-9}$

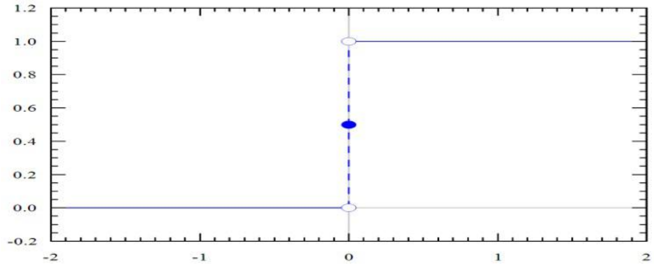
Outline

- Trainging Perceptron
- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Activation functions

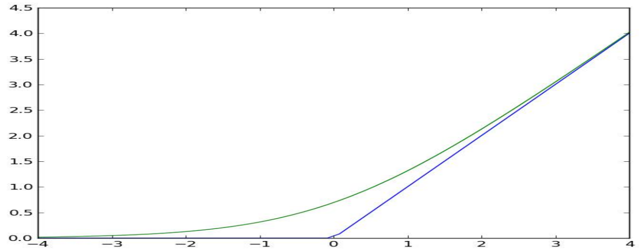
- Step function:

$$f(z) = \begin{cases} +1, z > 0 \\ 0, z \leq 0 \end{cases}$$



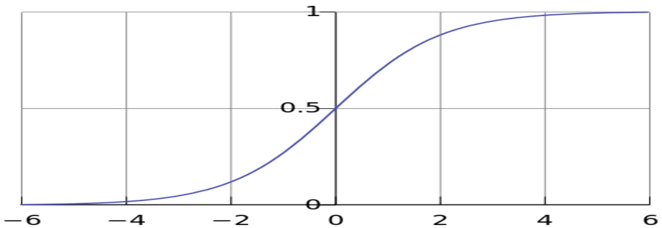
- Rectifier function:

$$f(z) = \max\ \{0, z\}$$



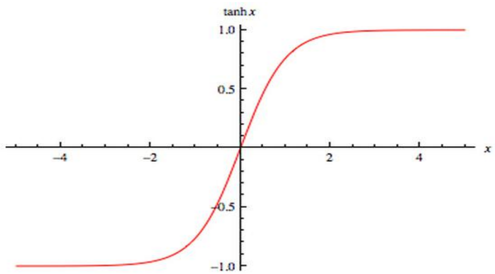
- Sigmoid function

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



- Hyperbolic tan function

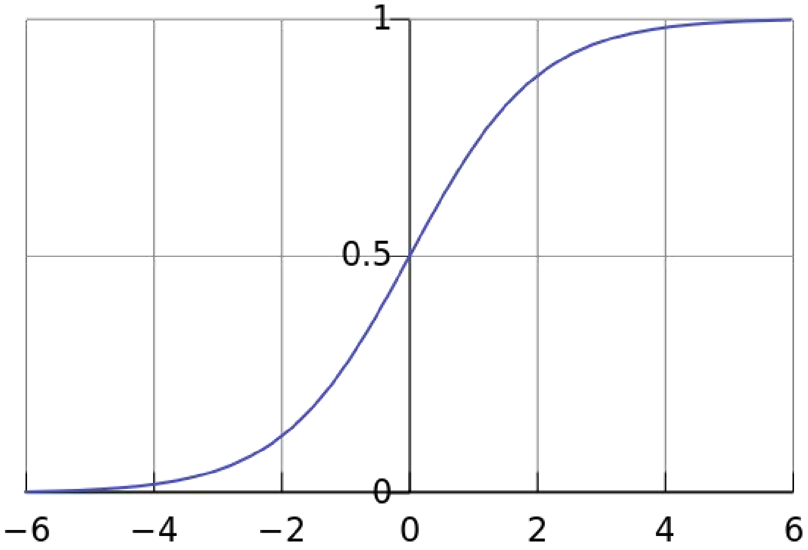
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



- Stochastic binary neural

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

Sigmoid Function

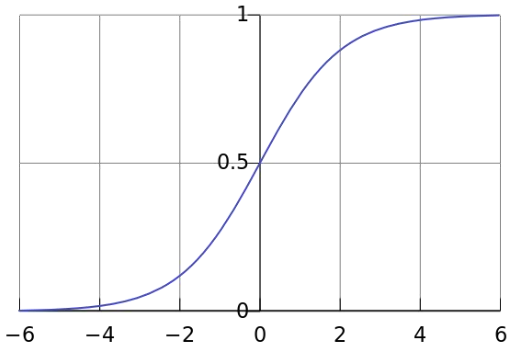


$$f(x) = \frac{1}{1+e^{-x}}$$

Pros:
- Squashes numbers to range [0,1].
- Historically popular since they have nice interpretation as a "firing rate" of a neuron.

Cons:
- exp() is a bit compute expensive.
- Saturated neurons "kill" the gradients.

Sigmoid Function



Forward pass:

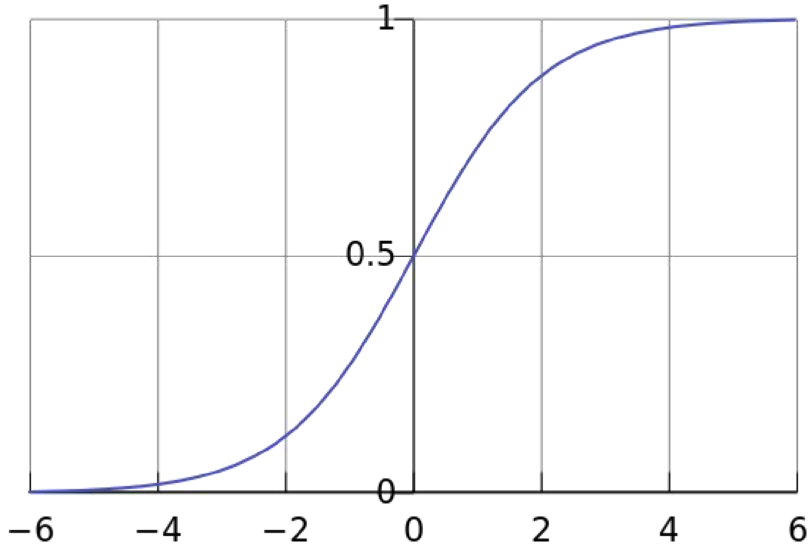
$$x \longrightarrow f(x) = \frac{1}{1+e^{-x}} \longrightarrow f(x)$$

Backward pass:

$$\frac{\partial H(x)}{\partial f(x)} \frac{\partial f(x)}{\partial x} \longleftarrow f(x) = \frac{1}{1+e^{-x}} \longleftarrow \frac{\partial H(x)}{\partial f(x)}$$

What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Sigmoid Function



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

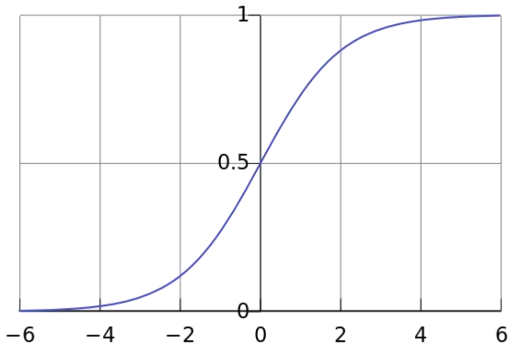
Pros:
- Squashes numbers to range [0,1].
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron.

Cons:
- exp() is a bit compute expensive.
- Saturated neurons "kill" the gradients.
- Sigmoid outputs are not zerocentered.

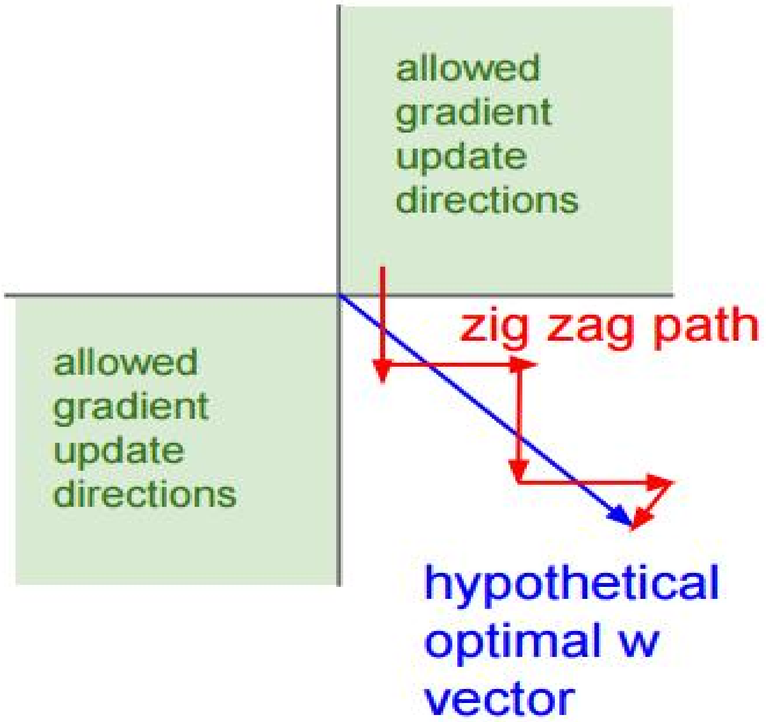
Sigmoid Function

Consider what happens when the input to a neuron x is always positive:

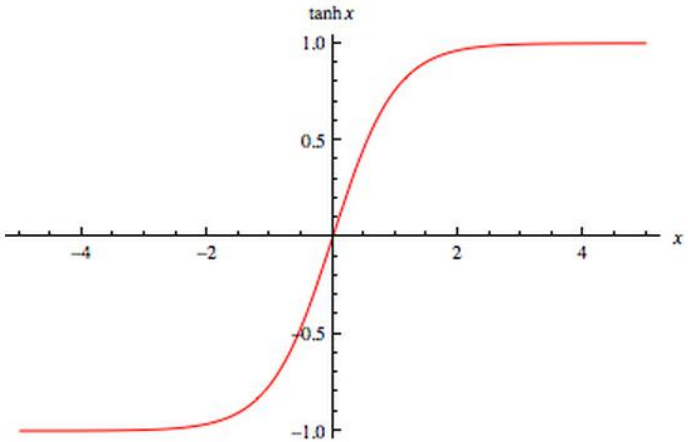
- Always all positive or all negative!



$$f\left(\sum_i w_i x_i + b\right)$$



Activation Function: tanh

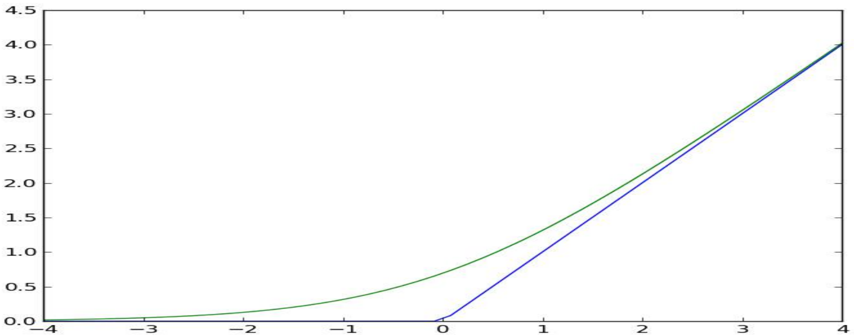


$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Pros:
- Squashes numbers to range [-1,1].
- Outputs are zerocentered.

Cons:
- exp() is a bit compute expensive.
- Saturated neurons “kill” the gradients.

Activation Function: Rectified Linear Unit (ReLU)

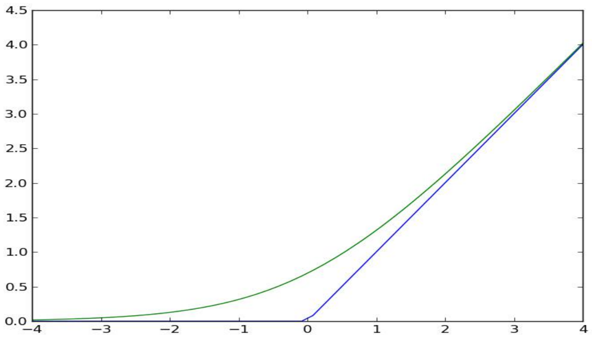


$$f(x) = \max(x, 0)$$

Pros:
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

Cons:
- Not zero-centered output.
- An annoyance...

ReLU Function



Forward pass:

$$x \longrightarrow f(x) = \max(0, x) \longrightarrow f(x)$$

Backward pass:

$$\frac{\partial H(x)}{\partial f(x)} \frac{\partial f(x)}{\partial x} \longleftarrow f(x) = \max(0, x) \longleftarrow \frac{\partial H(x)}{\partial f(x)}$$

What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Backpropagation

- Perform a feedforward pass, computing the activations for layers *L2*, *L3*, and so on up to the output layer *h(x)*.

- For each output unit *i* in the output layer, set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

- For $l = n_l - 1, n_l - 2, \ldots, 2$

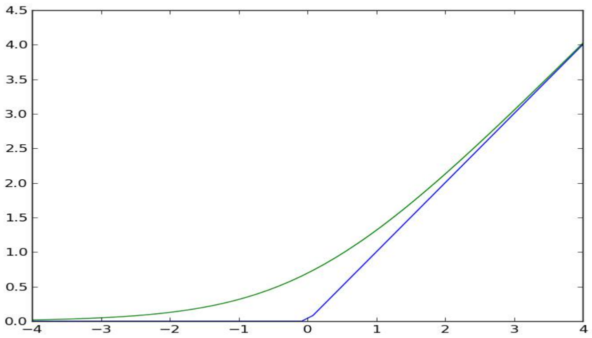
  For each node *i* in layer *l*, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

- Compute the partial derivatives in each layer,

$$\frac{\partial H}{\partial w_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} + \lambda \cdot w_{ij}^{(l)} \quad ; \quad \frac{\partial H}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

ReLU Function



Forward pass:

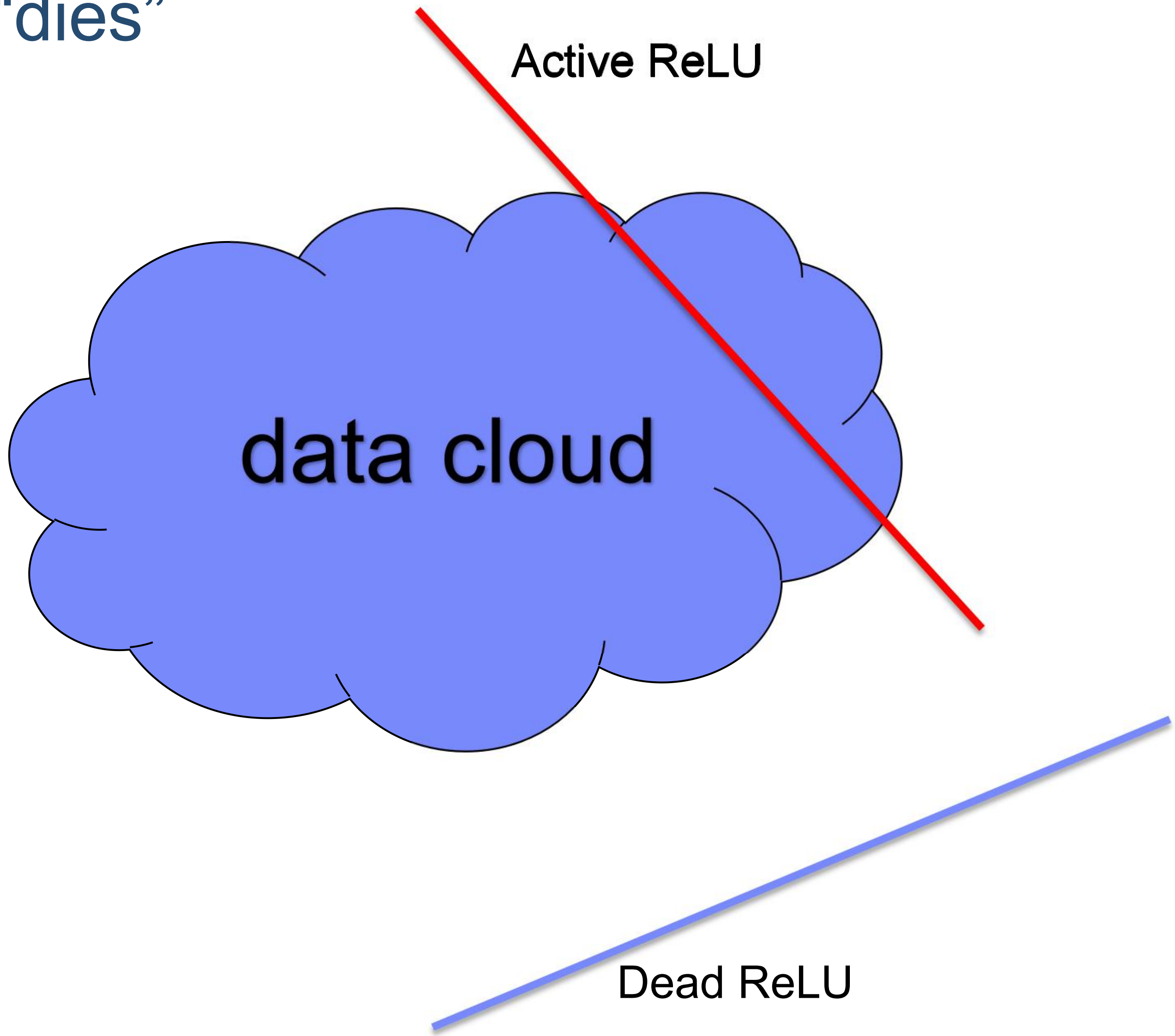
$$x \longrightarrow f(x) = \max(0, x) \longrightarrow f(x)$$

Backward pass:

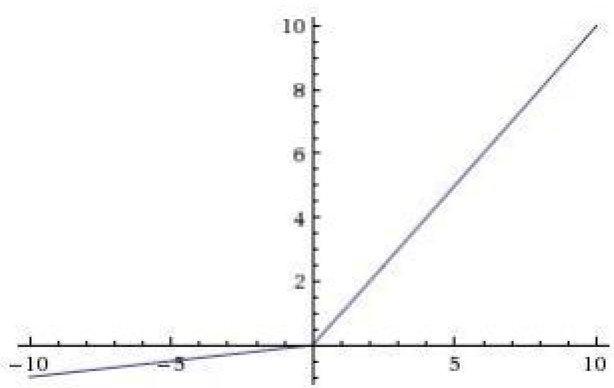
$$\frac{\partial H(x)}{\partial f(x)} \frac{\partial f(x)}{\partial x} \longleftarrow f(x) = \max(0, x) \longleftarrow \frac{\partial H(x)}{\partial f(x)}$$

What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

ReLU "dies"



Alternatives



Pros:
- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Will not die

Cons:
- Not zero-centered output.

Leak ReLU:

$$f(x) = \max(0.01x, x)$$

Parametric ReLU (PReLU):

$$f(x) = \max(\alpha x, x)$$

Alternatives: Maxout

$$f(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$$

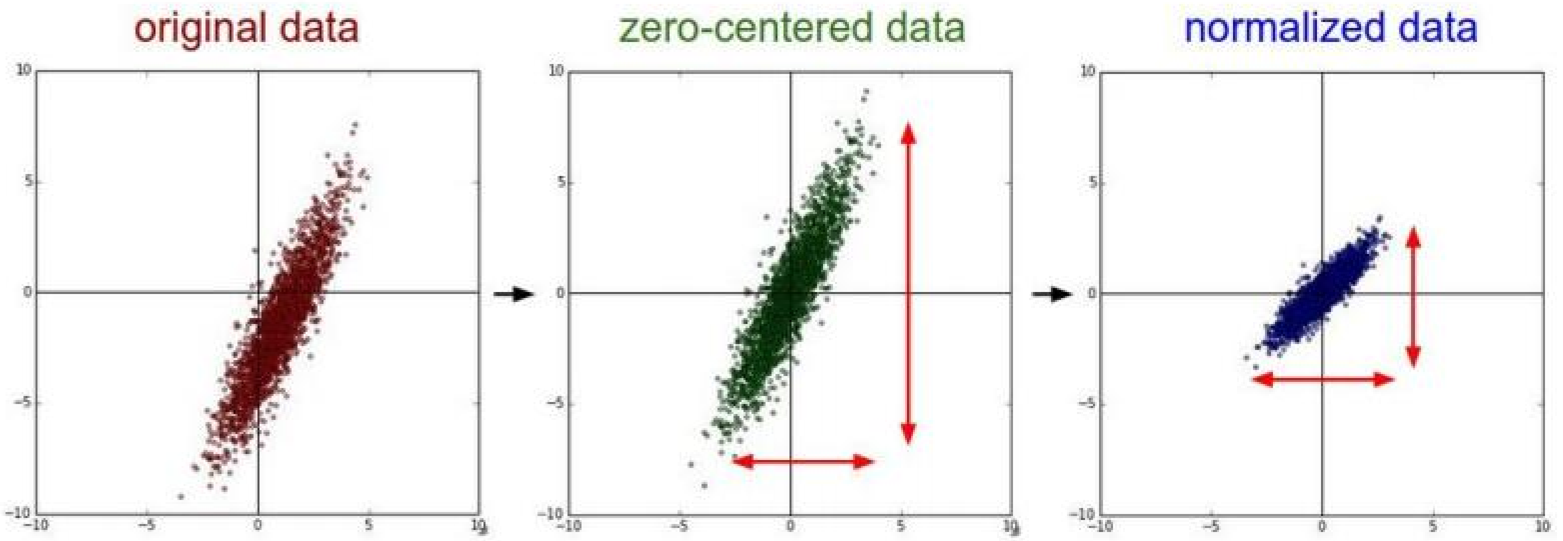
Pros:
- Does not saturate
- Very computationally efficient.
- Converges much faster than sigmoid/tanh in practice (e.g. 6x).
- Will not die.

Cons:
- Does not have the basic form of dot product nonlinearity.
- Not zero-centered output.
- Doubles the number of parameters.

Activation Function: in practise

- Use ReLU. Be careful with your learning rates.
- Try out Leaky ReLU/PReLU/Maxout.
- Try out tanh but don't expect much.
- Don't use sigmoid.

Data Preprocessing



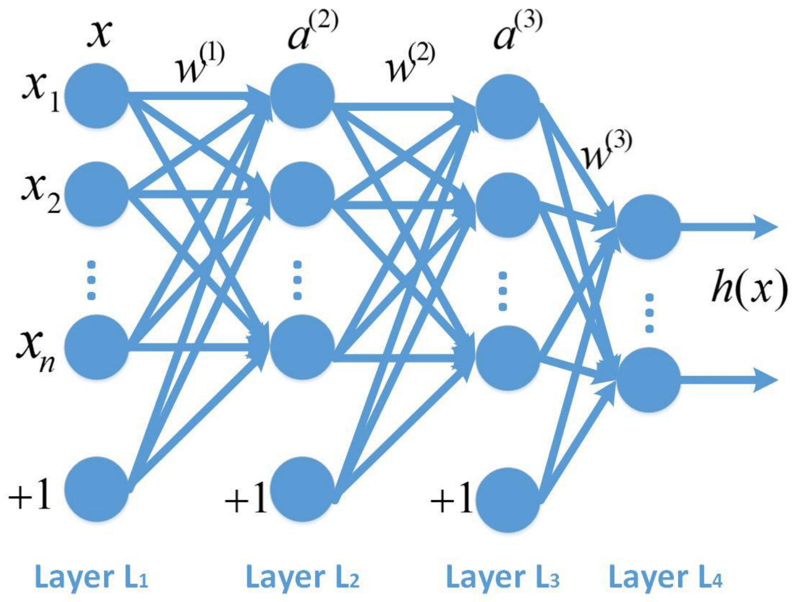

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

Outline

- Trainging Perceptron
- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Weight initialization



1. Can we initialize W=0? ✗

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

2. Can we initialize W=const? ✗

All nuerons will learn the same thing.

3. Can we initialize W=small random? ✗

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

Backpropagation

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer $h(x)$.

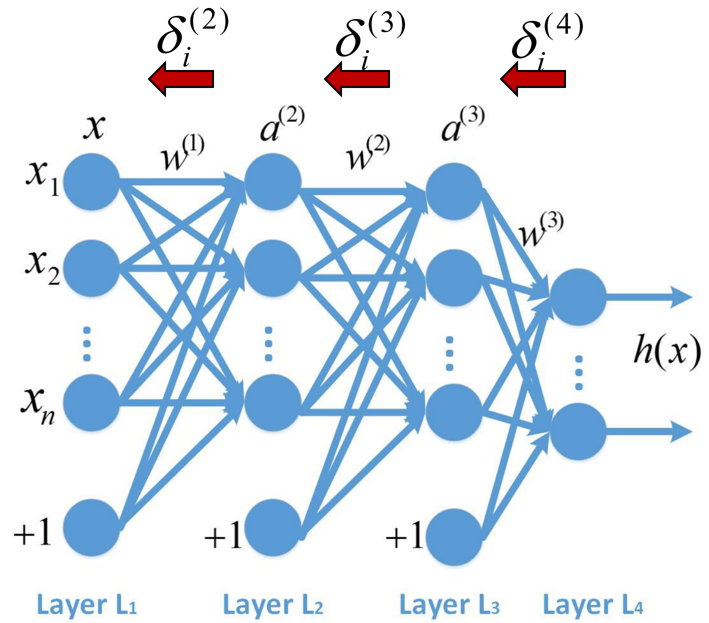
- For each output unit i in the output layer, set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

- For $l = n_l - 1, n_l - 2, \ldots, 2$

  For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_l+1} w_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$



Proper initialization is an active area of research…

- Understanding the difficulty of training deep feedforward neural networks. Glorot and Bengio, 2010
- Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. Saxe et al, 2013
- Random walk initialization for training very deep feedforward networks. Sussillo and Abbott, 2014
- Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. He et al., 2015
- Data-dependent Initializations of Convolutional Neural Networks. Krähenbühl et al., 2015
- All you need is a good init. Mishkin and Matas, 2015

Why should we do?

- Try initialize initializate uniformly from [-1,1]
- Try initialize initializate from 0-mean normal distribution
- Try initialize parameters from well-trained models

Transfer Learning with CNNs

1. Train on ImageNet

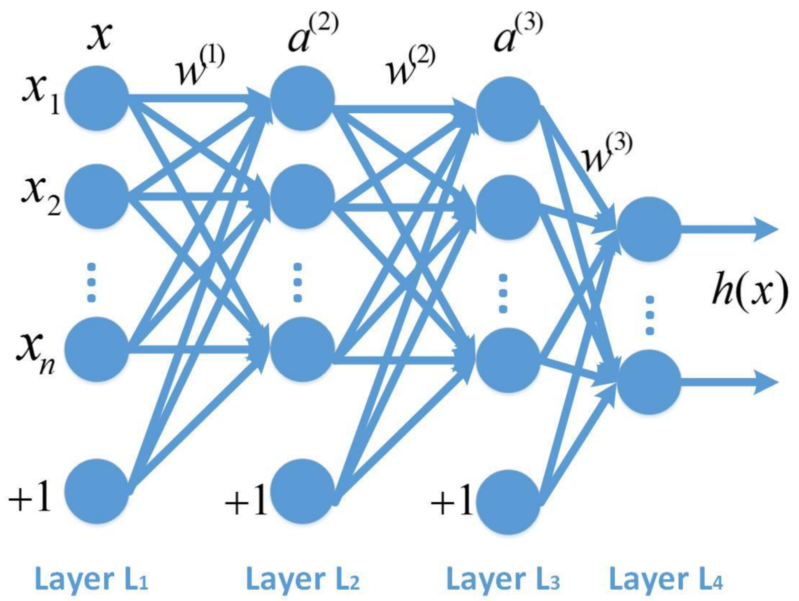
2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Regularization



- Why need regularization? Overfitting!
- L2/L1 Regularization
- Dropout

$$\min \ \mathrm{H} = \frac{1}{2m}\sum_{i=1}^{m}\left\|h(x^{(i)}) - y^{(i)}\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\left\|w^{(l)}\right\|_F^2$$

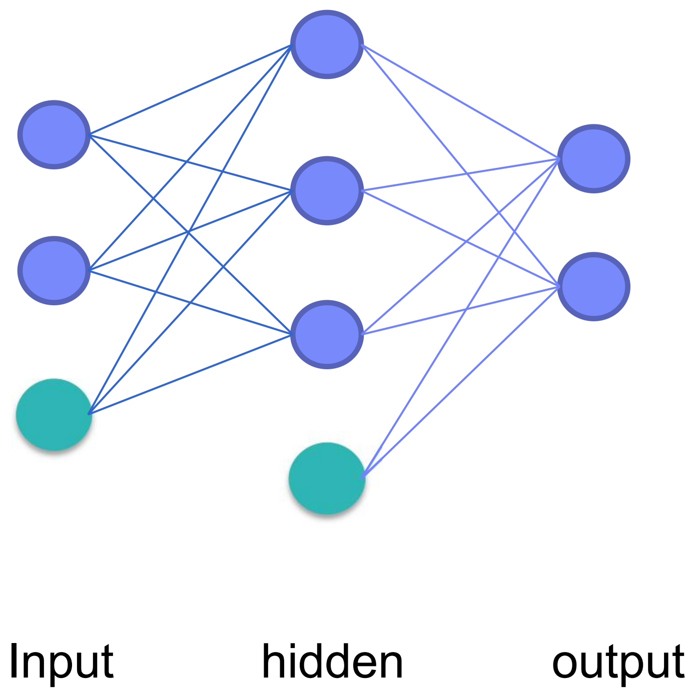
L2/L1 Regularization example

Demo

Dropout

- It appears to be hard to do massive model averaging in deep neural networks:
  - Each net takes a long time to learn.
  - At test time we don't want to run lots of different large neural nets.
- You need to use many different types of model and then combine them to make predictions at test time.
  - Decision Tree V.S. Random Forest

Dropout



- Consider a neural net with one hidden layer.
- Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
- So we are randomly sampling from 2^h different architectures.
  - All architectures share weights

Dropout Training

- We sample from 2^h models. So only a few of the models ever get trained, and they only get one training example.

- The sharing of the weights means that every model is very strongly regularized.
  - It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.
  - It pulls the weights towards what other models want.

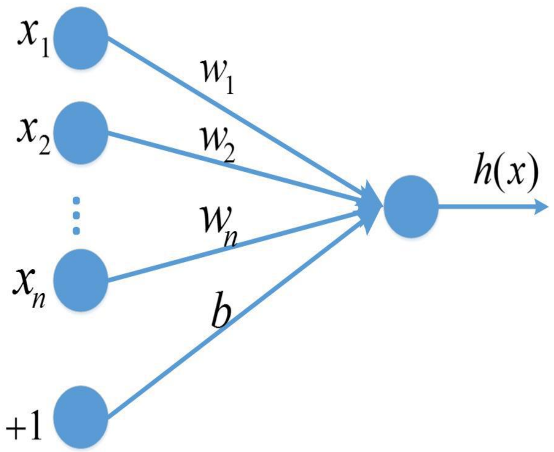
Dropout Testing

- We could sample many different architectures and take the geometric mean of their output distributions.

- It is faster to use all of the hidden units, but to halve their outgoing weights.
  - This exactly computes the geometric mean of the predictions of all 2^H models.

A familiar example of dropout

- Do logistic regression, but for each training case, dropout all but one of the inputs.
- At test time, use all of the inputs.
- This is called "Naïve Bayes".



Outline

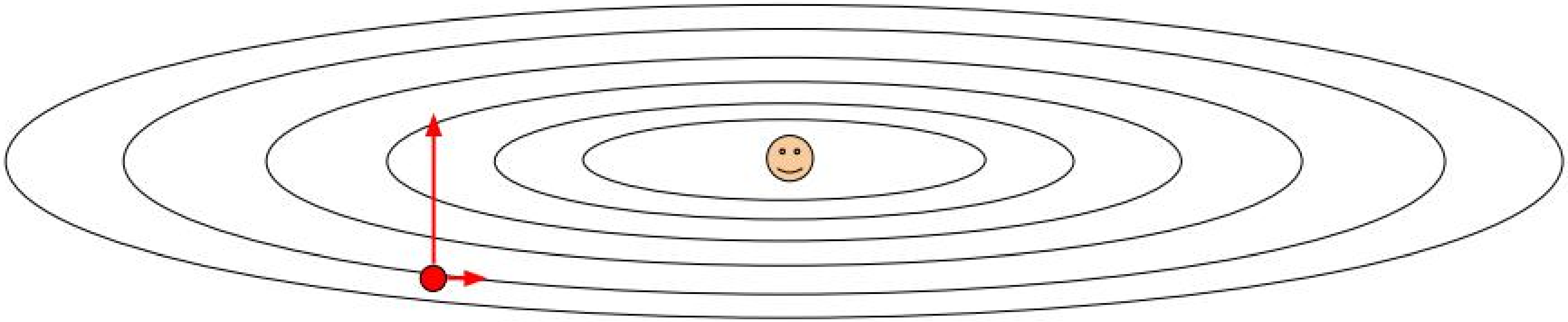
- Trainging Perceptron

- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Parameter updates: mini-batch gradient descent

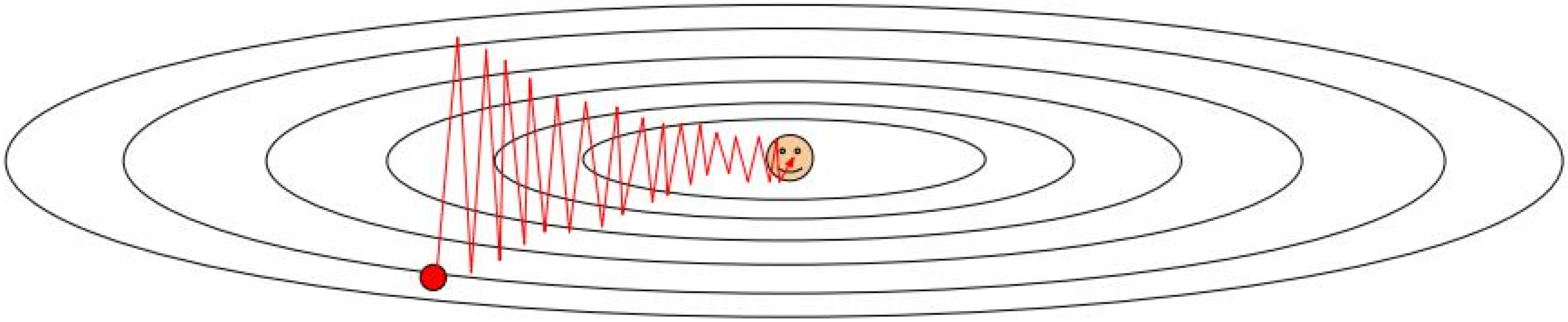
- Sample a batch of data
- Forward prop it through the network, get loss
- Backprop to calculate the gradients
- Update the parameters using the gradients
- Repeat until convergence

Parameter updates: momentum update



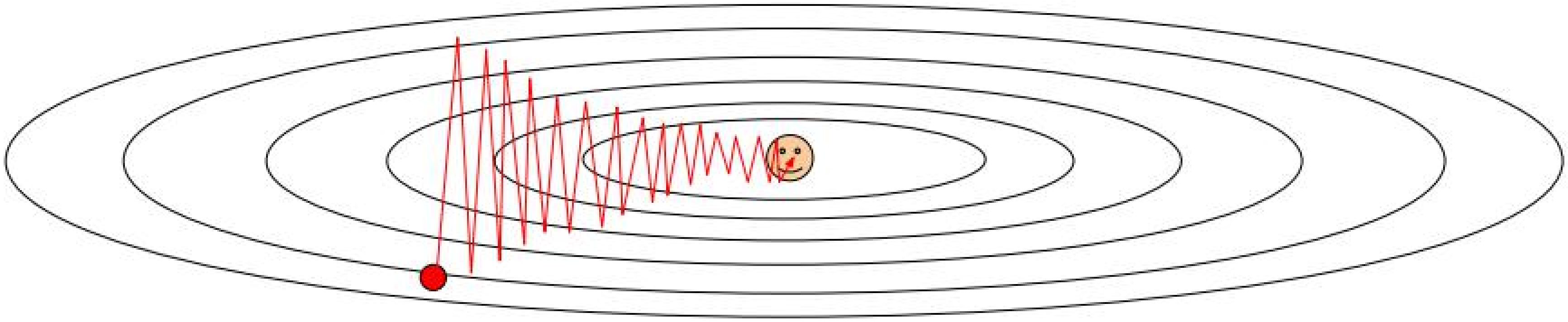
Suppose loss function is steep vertically but shallow horizontally

Parameter updates: momentum update



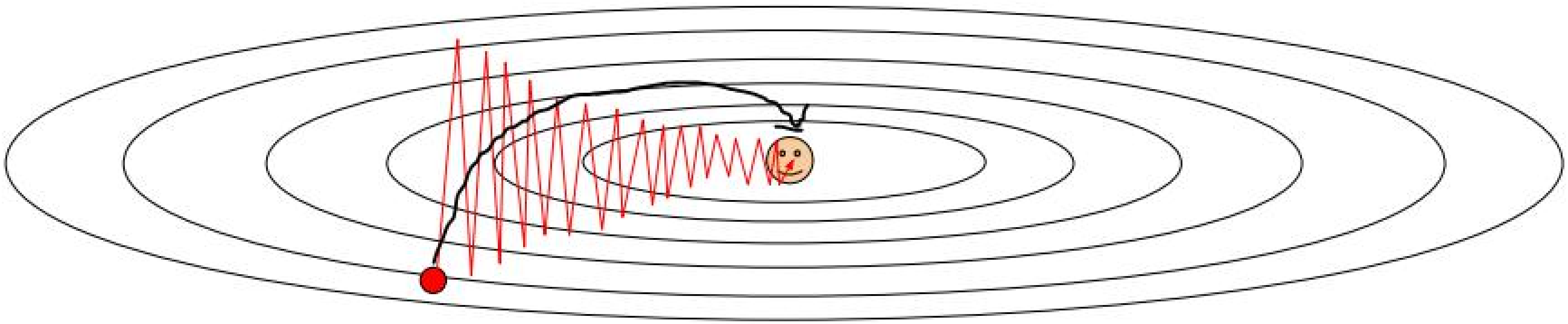
Suppose loss function is steep vertically but shallow horizontally

Parameter updates: momentum update



$$w = w - \eta \frac{\partial E}{\partial w}$$

Parameter updates: momentum update



$$v = \mu v - \eta \frac{\partial E}{\partial w}$$

$$w = w + v$$

$$where, \mu = 0.5 / 0.9 / 0.99$$

Parameter updates: 2nd order method

- Second-order Taylor expansion:

$$f_T(x) = f_T(x_n + \Delta x) \approx f(x_n) + f'(x_n)\Delta x + \frac{1}{2} f''(x_n)\Delta x^2$$

- Search Δx that minimize f(x+Δx):

$$0 = \frac{d}{d\Delta x}\left(f(x_n) + f'(x_n)\Delta x + \frac{1}{2} f''(x_n)\Delta x^2\right) = f'(x_n) + f''(x_n)\Delta x$$

$$\Delta x = \frac{f'(x)}{f''(x)}, \qquad x = x + \frac{f'(x)}{f''(x)}$$

- Generalize to higher dimension:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [\mathbf{H} f(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n),\ n \geq 0.$$

Parameter updates: L-BFGS

- Quasi-Newton methods (BGFS most popular): instead of inverting the Hessian (O(n^3)), approximate inverse Hessian with rank 1 updates over time (O(n^2) each).
- L-BFGS (Limited memory BFGS): Does not form/store the full inverse Hessian
- Usually works very well in full batch, but not for mini-batch
- No learning ratio

Outline

- Trainging Perceptron
- Trainging feed forward nueral networks (MLP)
  - Backpropagation algorithm
  - Gradient checking
  - Activation functions
  - Preprocessing
  - Weight initialization
  - Regularization
  - Parameter updates

Questions?

All together!

- Step 1: Preprocess the data
- Step 2: Choose the architecture
- Step 3: Implement the Backpropagation algorithm
  - Gradient check
- Step 3: Turn off regularization, have a try
  - Make sure that you can overfit small portion of the training data.
- Step 4: Start with small regularization, have a try
  - Make sure that you see the loss go down.
  - If not, adjust learning rate.