Quick review of Deep Learning

EE 5178

Kaushik Mitra
Depart. Of Electrical Engineering, IIT Madras

Machine learning

Goal: Learning from the data with minimal intervention from the user

Supervised learning:

- Learns a mapping b/w *input* and *output* pairs (x_i,y_i) e.g. image classification

f(x) → Coffee mug

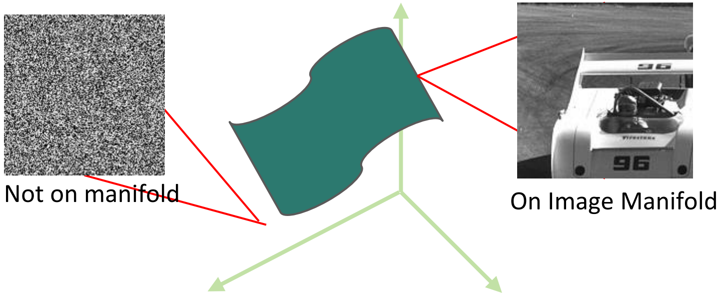
f(x) → tiger

f(x) → car

- *Applications*: image classification, object detection, scene recognition

Unsupervised learning:

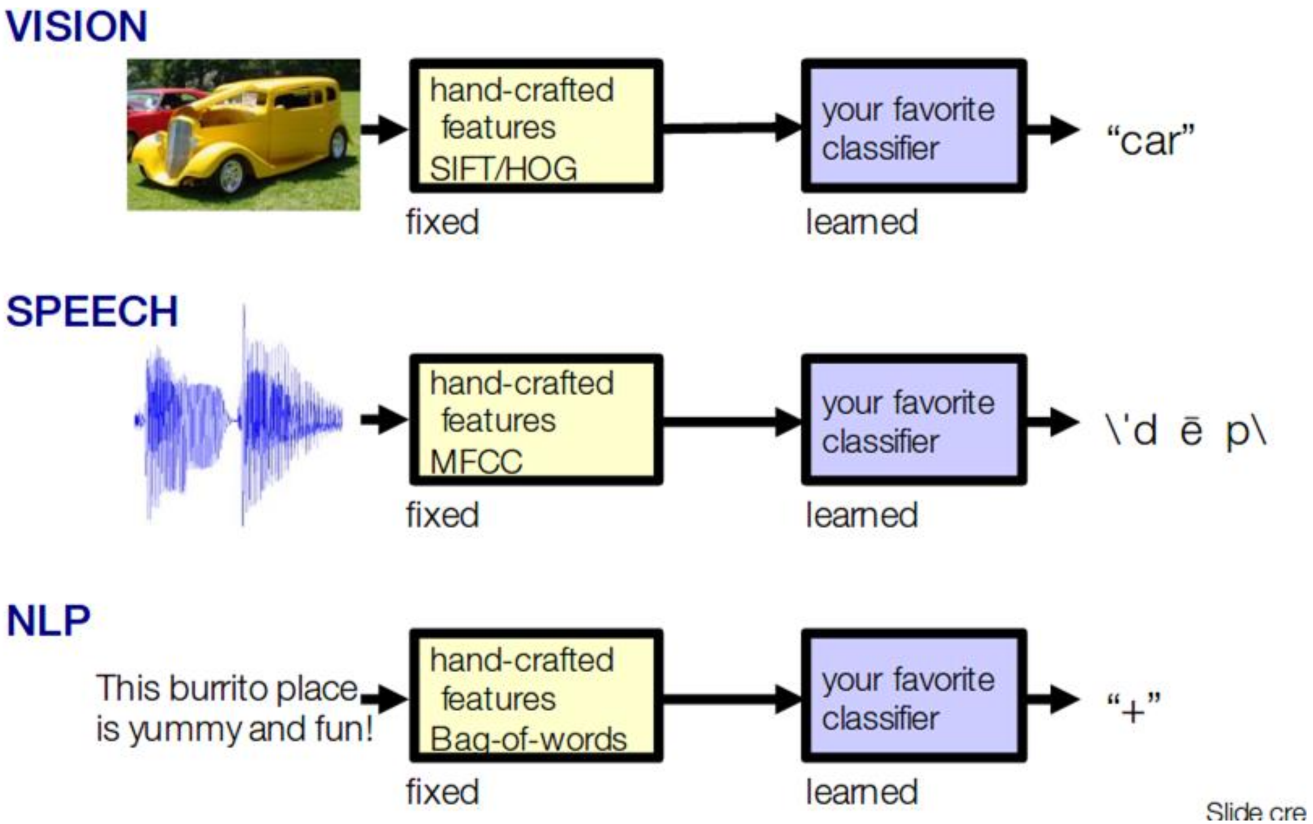
- Given only data '*x*' learn the inherent underlying structure
- Consider a 64x64 binary image



- *Applications*: clustering, dimensionality reduction, density estimation

Traditional approaches

- Manual feature extraction (SIFT/HOG)
- Classifier is learned independent of feature extraction

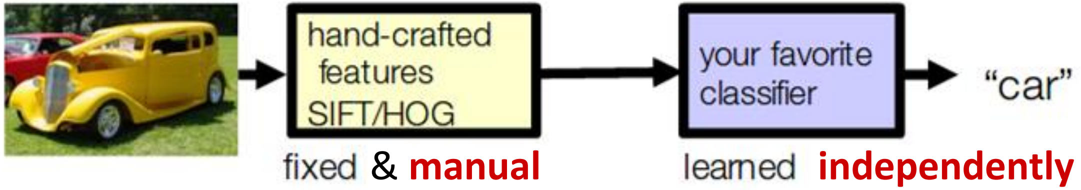


*slide courtesy, Yoshua Bengio and Yahn Lecun

Traditional approaches vs Deep learning

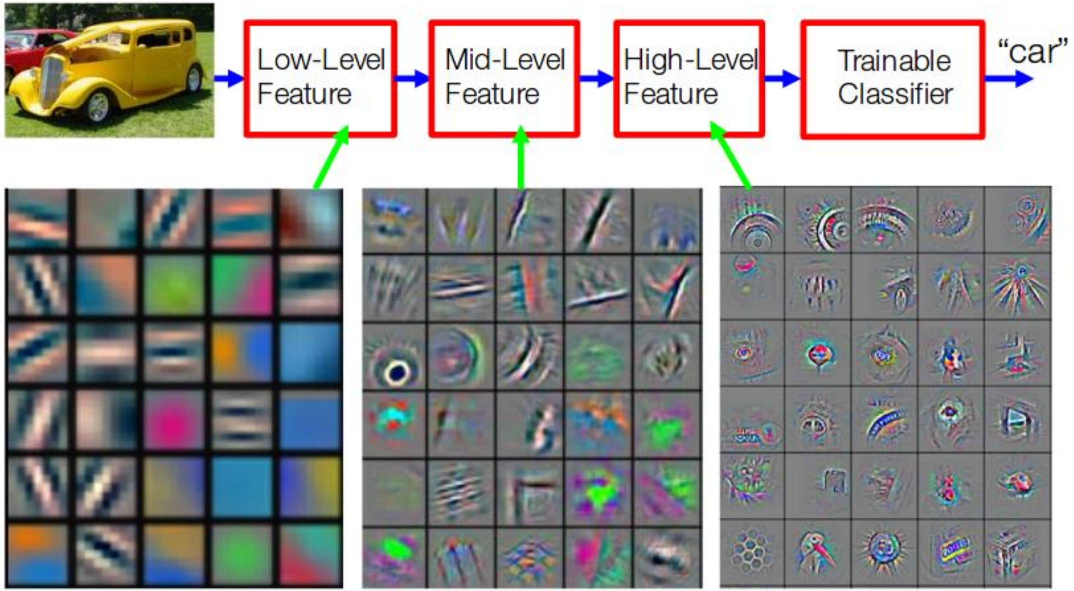
What's wrong with the traditional approaches?

- Compositional feature abstraction is missing



3 key ideas of deep learning

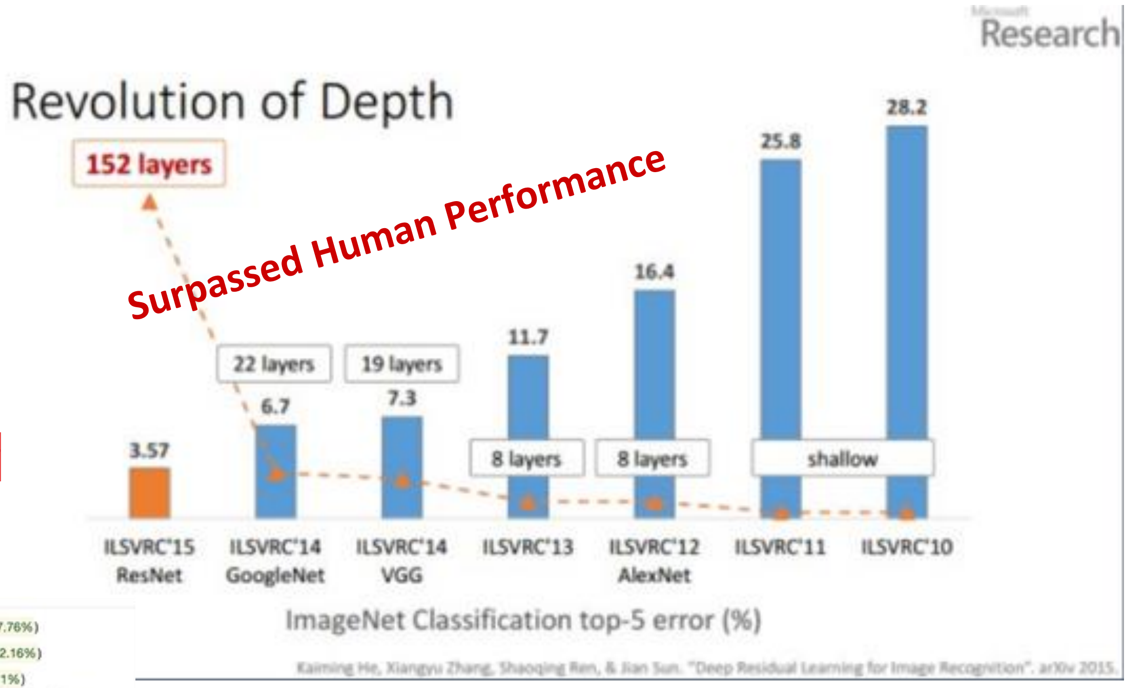
- (Hierarchical) Compositionality
  - Cascade of nonlinear functions
  - Multiple layers of abstractions
- End-to-End learning
  - Learning (task-driven) representations
  - Learning to extract features
- Distributed Representation
  - No single neuron encodes everything
  - Group of neurons work together



*slide courtesy, Yoshua Bengio and Yahn Lecun

Image Classification

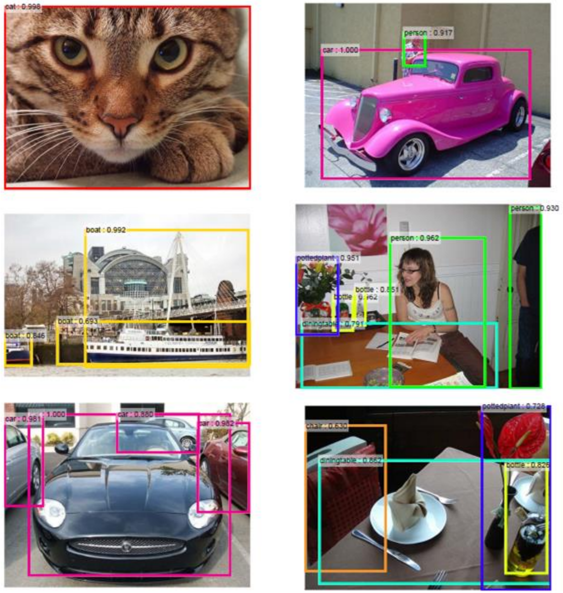
LeNet by Lecun et al. 1998 (MNIST)
AlexNet by Krish et al. NIPS’ 12
VGGNet by Simonyan et al. ICLR’ 15
GoogLeNet by Szegedy et al. CVPR’ 15
ResNet by He et al., CVPR’ 17 best paper



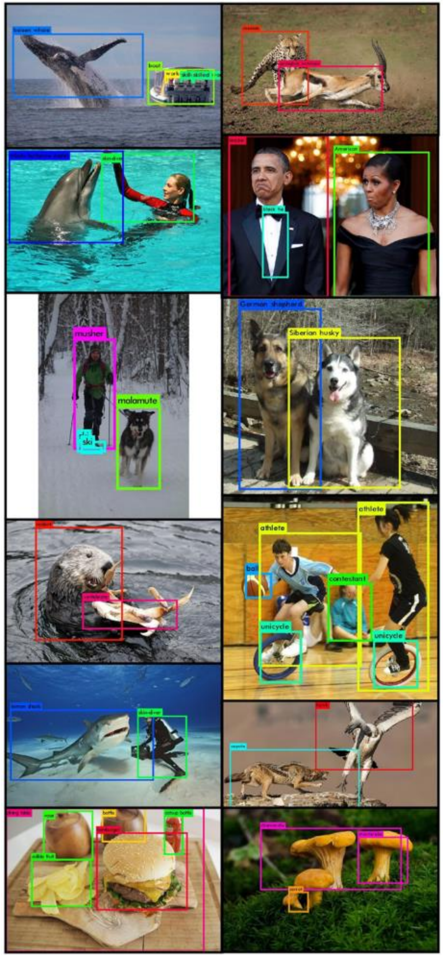
*pic courtesy: Kaiming He

Object Detection

OverFeat by Sermanet et al., ICLR' 14 (NYU)
R-CNN by Girshick et al., CVPR' 14 (UCB)
SPP by He et al., ECCV' 14 (MSR)
Fast R-CNN by Girshick et al., arxiv (MSR)
Faster R-CNN by Ren et al., NIPS' 15 (MSR)
YOLO by Redmon et al., arxiv 2015
YOLO 9000 by Redmon et al., CVPR' 17



Faster RCNN detections

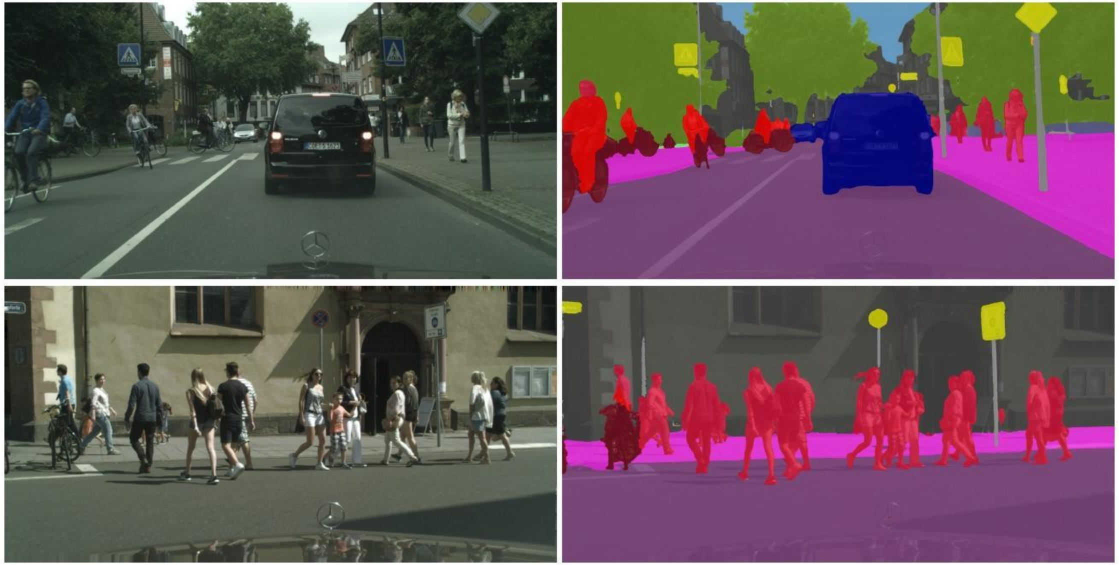


YOLO 9000 detections

pic courtesy: http://mrulafi.blogspot.in/2016_05_01_archive.html

Image Segmentation

FCN by Long et al. CVPR' 15
DeepLab by Chen et al. arxiv 2015
CRFS as RNNs by Zheng et al. ICCV' 15



*Pic courtesy: Kundu et al. CVPR 2016 on City scapes dataset

Style transfer

Neural style transfer by Gatys et al., CVPR' 16

Deep Photo Style Transfer by Luan et al., CVPR' 17

Actual | **Style** | **Actual with style**

Artistic

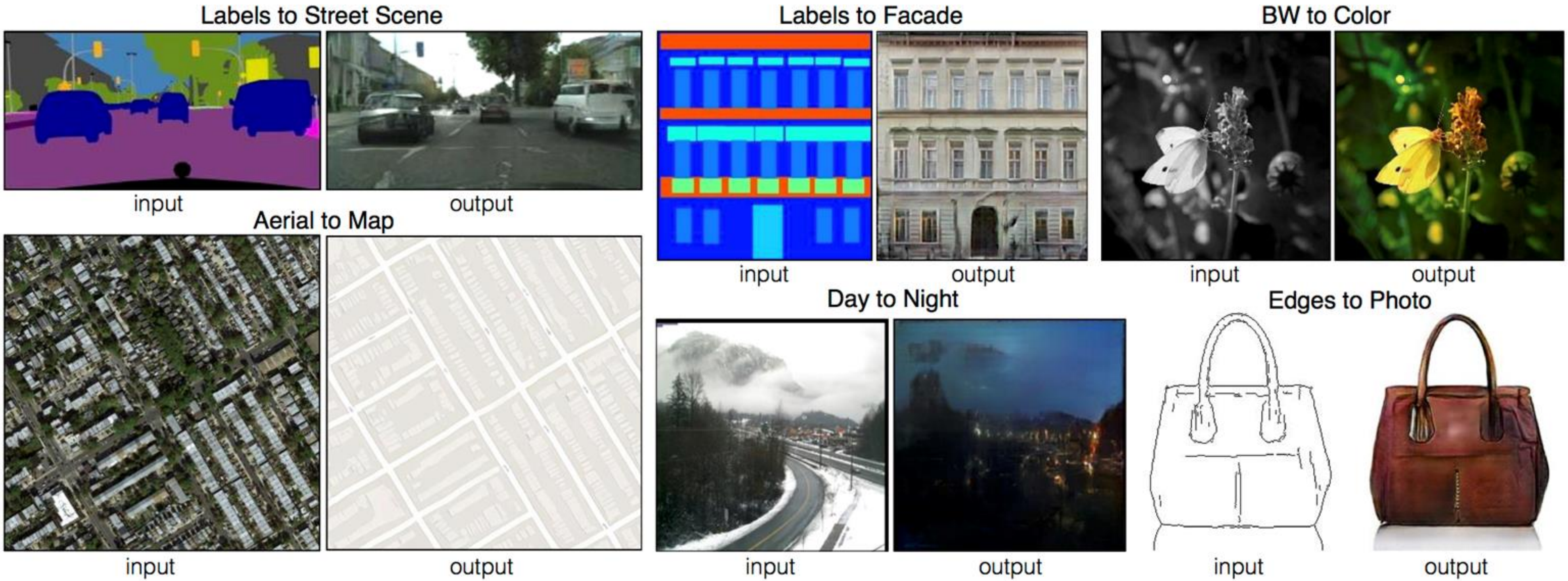
*Pic courtesy: http://deepart.io

Photo Realistic

*Pic courtesy: Deep Photo Style Transfer, Luan et al. CVPR 2017

Artistic applications

Image-to-Image Translation with Conditional Adversarial Nets by Isola et al., CVPR' 17



Image, Video and Audio Generation

PixelRNN/CNN by Gregor et al., ICML' 16 Best paper (Samples from ImageNet)

DCGAN Chintala et al. ICLR' 16
Sample bedroom images

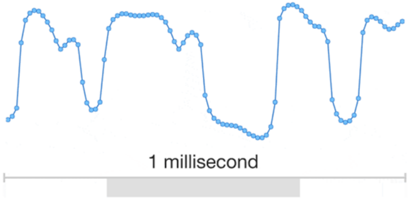
Generating **videos** with scene dynamics, by Vondrick et al., NIPS' 16

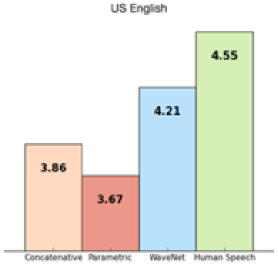
Ocean

Golf

Trains

WaveNet for **audio** synthesis by Oord et al. 2016, Deepmind





Check out wavenet's generated music
piano clips

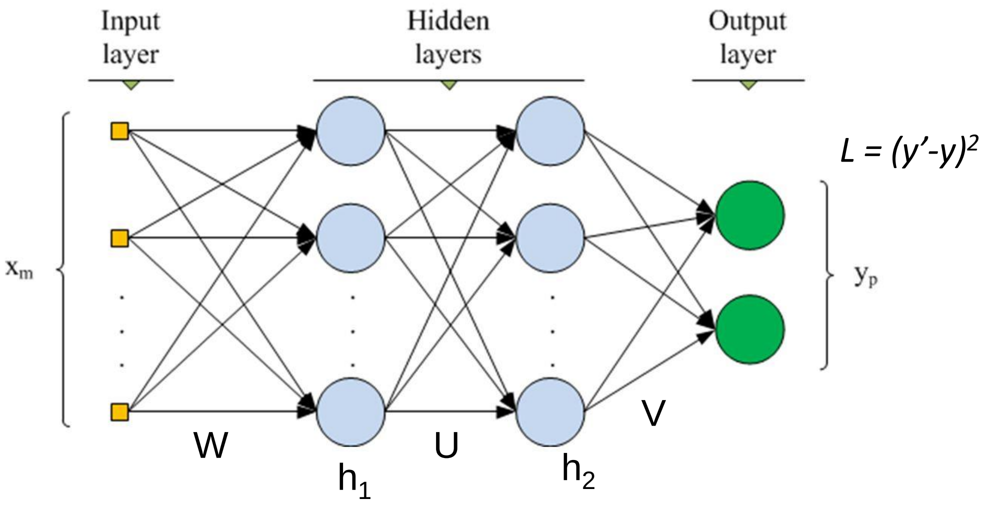
1. Basic Neural Networks

Multi-layer perceptrons (MLP) is a feed forward neural network with hidden layers

$h_1 = f(Wx + b_1)$, f is an activation function

$h_2 = g(Uh_1 + b_2)$

$y' = Vh_2 + b_3$



Hidden layers *increase* abstraction

– hence, better to have more hidden layers than a single layer with large number of neurons

Universal Approximation Theorem:

– simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters

*pic courtesy, Sancho course

1. Basic Neural Networks

What we will learn in the course about Neural Networks?

Introduction

- McCulloch and Pits model
- Rosenblatt's perceptron

Perceptrons

- Geometry and linear separability
- XoR problem
- Multi-layer perceptron (MLP)

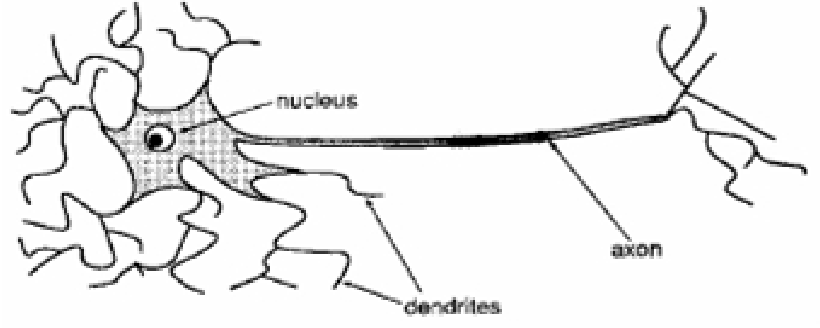
Training MLPs

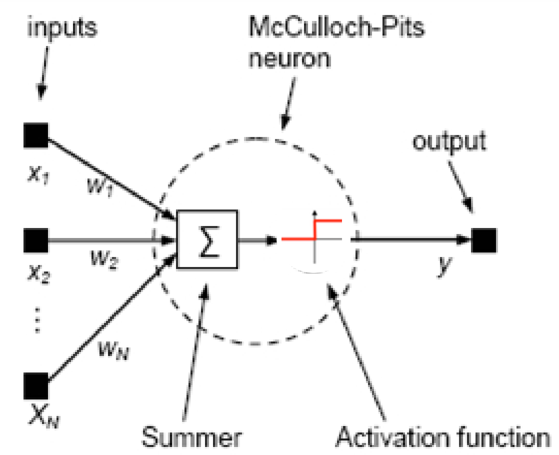
- Error back propagation
- Loss functions

Introduction to Neural Networks

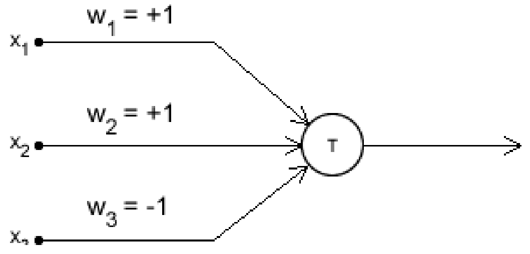
Perceptron, XOR Problem, Multi-layer Perceptron (MLP), Cost Functions, Activation functions and Output units

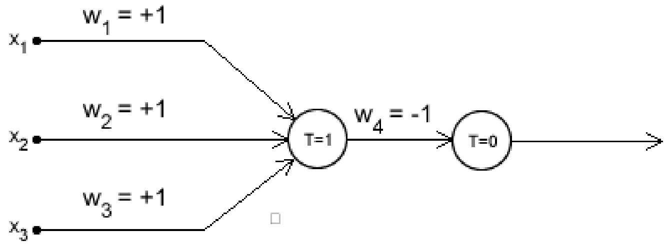
McCulloch - Pits model





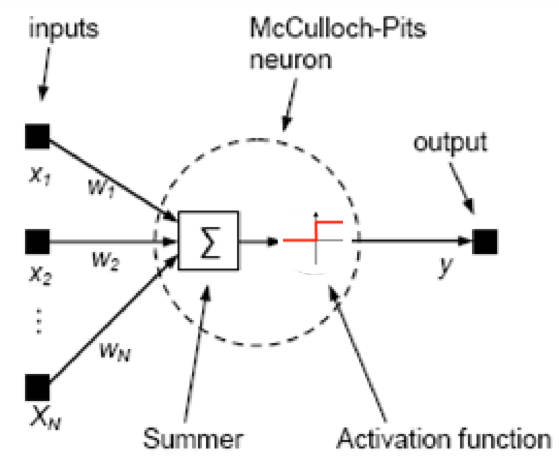
$$\sum_{i=1}^{n} w_i x_i > \mu$$



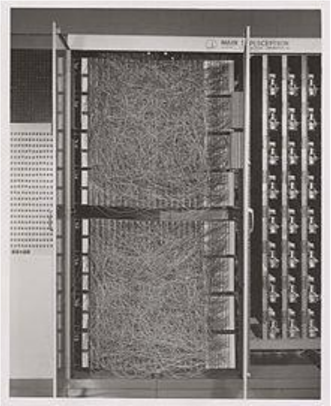


NOR gate

The Perceptron - Rosenblatt (1953)



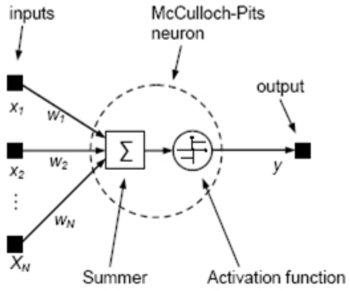
$$\sum_{i=1}^{n} w_i x_i > \mu$$



*Pic courtesy, wikipedia

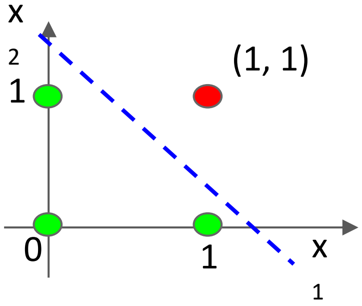
The Mark 1 Perceptron
By Rosenblat
for digit recognition

Perceptron - geometrical interpretation



$$\sum_{i=1}^{n} w_i x_i > \mu$$, What does this inequality imply in 2D case? Half plane

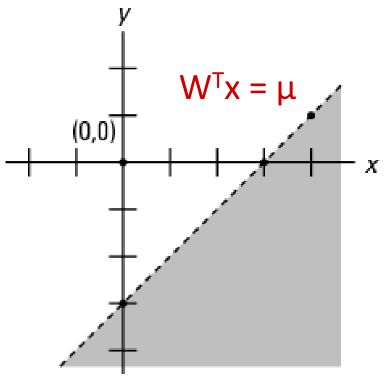
$\mathbb{X}$	*AND*
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	1



Solve for W, μ:

$x_1 + x_2 > 1.5$

$w_1 = 1$, $w_2 = 1$ and $\mu = 1.5$



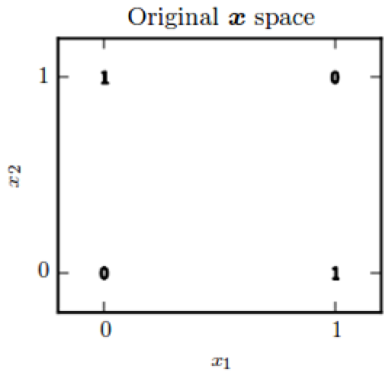
*Pic courtesy, [cliffsnotes]

Any function that is linearly separable can be computed by a perceptron

Perceptron - Limitations

Goal: learn the XoR function (f^*)

$\mathbb{X}$	f^*
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0



The data is not linearly separable

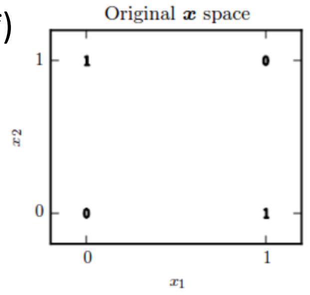
How to tackle this problem?

- Can we use more than one line?
- Yes, but how?

*Slide courtesy, Ian Goodfellow et al., deep learning book

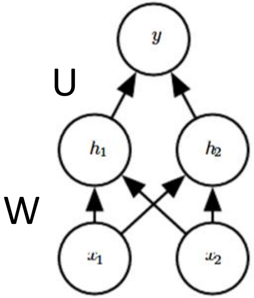
Perceptron - Limitations

Goal: learn the XoR function (f^*)



How to tackle this problem?

- Add a hidden layer with two units



$y = f^{(2)}(h; U, c)$

$y = f^{(2)}(\, f^{(1)}(x)\,)$

$h = f^{(1)}(x; W, b)$

What should $f^{(1)}$ compute?

- If its linear again the composition still remains linear

$f^{(2)}(h) = U^T h$ and $h = Wx$

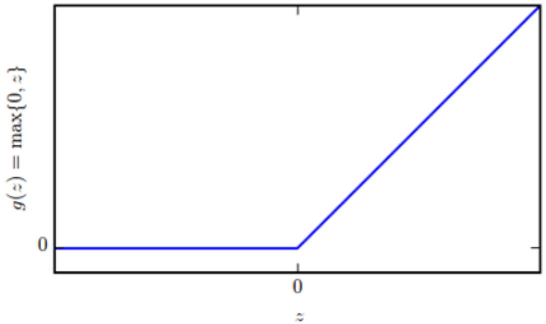
$y = U^T W x = W' x$

- $f^{(1)}$ should be nonlinear to extract useful features

  $h = f^{(1)}(x; W, b) = g(Wx+b)$

- g is referred as activation function commonly

- We will use ReLU here
  - ❑ Rectified Linear Unit (widely used)
  - ❑ $g(z) = \max\{0,z\}$



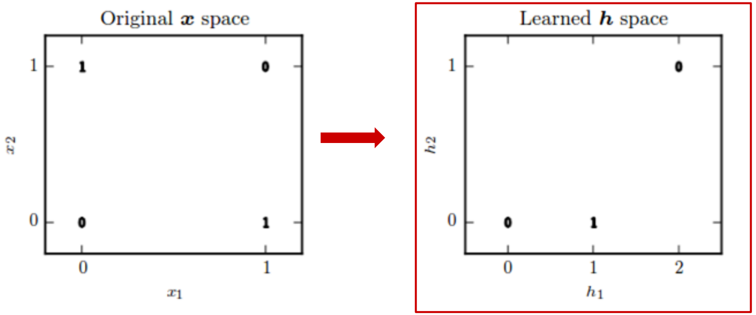
*Slide courtesy, Ian Goodfellow et al., deep learning book

Goal: learn the XoR function (f^*)

Perceptron - Limitations

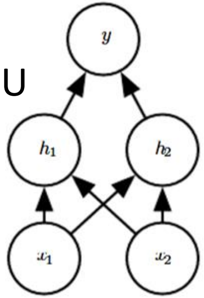
How to tackle this problem?

- Add a hidden layer with two units
- Use ReLU activation in 1st layer



U

W



$$y = U^T h + c\ ; \quad y = U^T \underbrace{\max\{0, Wx+b\}}_{\text{ReLU}} + c$$

$$h = g(Wx+b) = \max\{0, Wx+b\}$$

Let,

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$c = 0$$

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad WX = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

$$WX + b = \begin{bmatrix} 0 & 1 & 1 & 2 \\ -1 & 0 & 0 & 1 \end{bmatrix} \qquad \boxed{h = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Upon ReLU}}$$

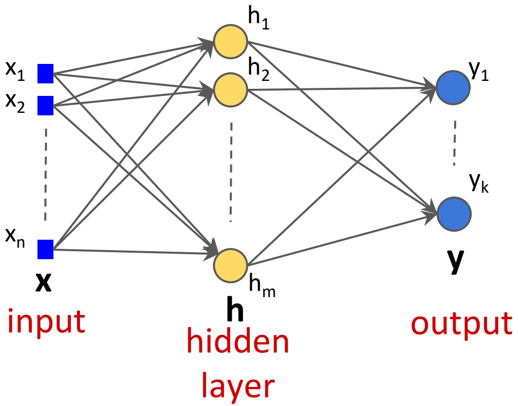
After layer2

$$[0\ 1\ 1\ 0]$$

*Slide courtesy, Ian Goodfellow et al., deep learning book

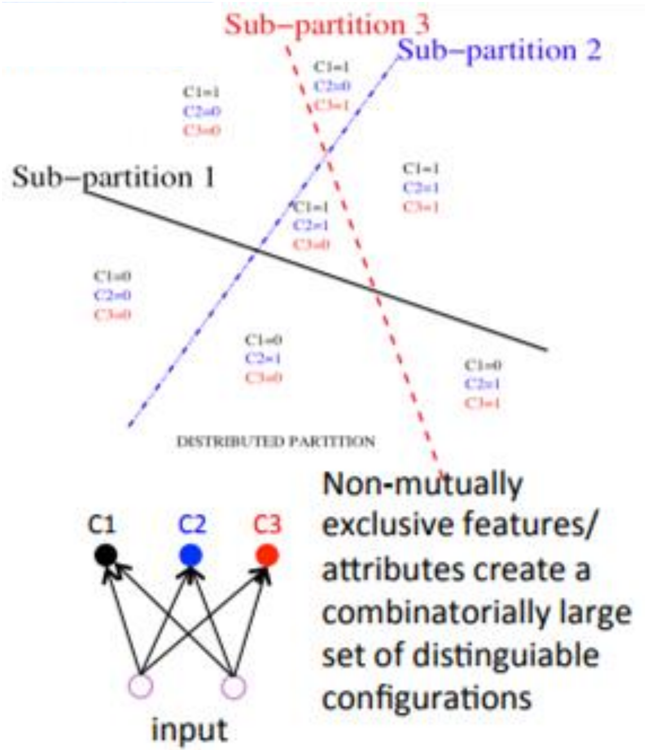
Multi-layer Perceptrons (MLP)

A typical feed forward neural network



$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b}_1); \quad \mathbf{y} = g(U\mathbf{h} + \mathbf{b}_2)$$

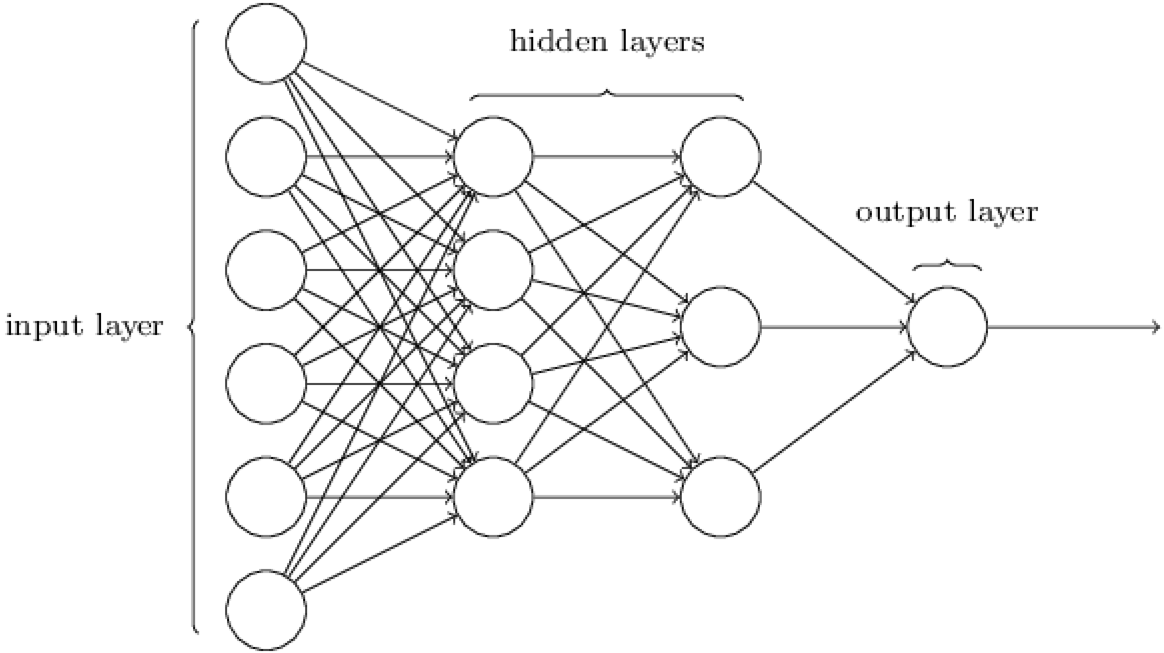
With more hidden units network is more expressible



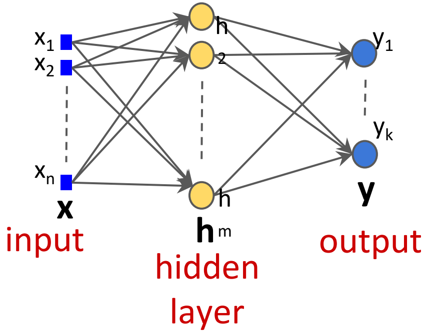
*Pic courtesy, Yoshua Bengio

Specification of a MLP

- ❖ Number of hidden layers and units in each layer
- ❖ Activation function for
  - ➢ Hidden layers
  - ➢ Output layers
- ❖ Cost function



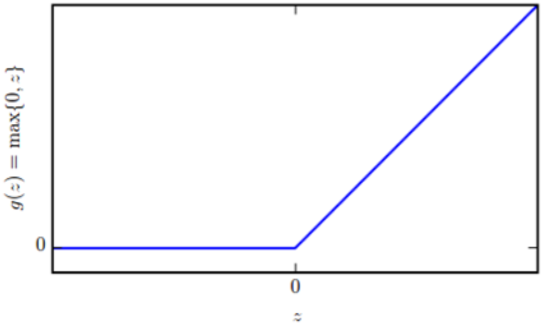
Activation functions for hidden layers



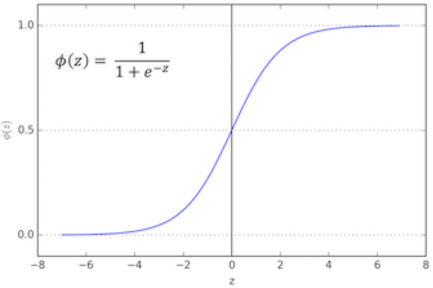
h = g(Wx+b); Affine transformation followed by activation function, g

– Very important factor in learning features

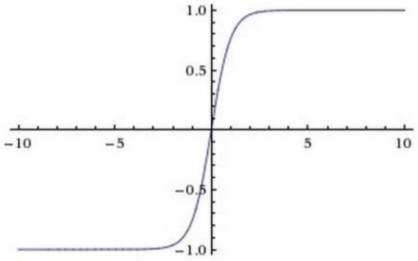
g(z) = max{0,z}, ReLU



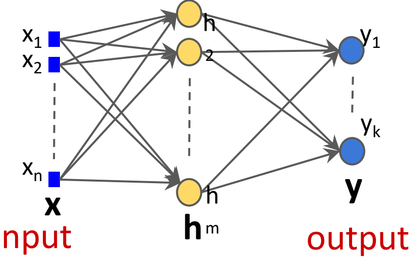
g(z) = σ(z), sigmoid



$tanh$(z) = 2σ(2z) - 1



Activation functions for Output units



- Linear units for real valued outputs
  - Activation function is left to be linear
  - Given features h,

    $$y' = Wh + b$$

  - Most commonly used with regression tasks

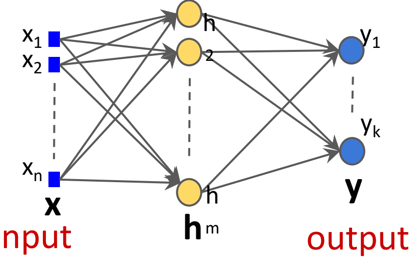
- Say you want to do binary classification
  - What kind of distribution describes output?
    Bernoulli
  - How to constrain the output - valid probability?
    Can you use linear activation?

    $$P(y=1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}.$$

  - What is the problem? *Not amenable for gradient based learning*
  - Instead, use sigmoid unit - output ∈ [0,1]

    $$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$

Activation functions for Output units



- Now, say we want to do multi-class classification (K classes)
  - ❑ Output should be K probabilities,

    $p_k = p(class = k|x)\ \forall\ k = 1$ to K

  - ❑ Can we use K sigmoid units?

    Won't be sufficient, since probabilities are not constrained to sum to 1

    $\sum_k p_k = 1$

  - ❑ We will look at softmax unit for this

    Idea is to convert a vector of real values to valid probabilities,

    How?
    - ❑ Make all the elements positive
    - ❑ Normalize the values

- Let, $\mathbf{z} = [z_1, \ldots, z_K]^T$; $\quad \mathbf{z} = W\mathbf{h} + \mathbf{b}$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

Cost functions

For regression,

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})||^2$$

$$\frac{1}{2}\sum_{\{x_i,y_i\}} ||y_i - f(x_i,\theta)||^2$$

For classification,

- Typically outputs a probability vector $q(c = k|\mathrm{x})\ \forall\ k$
- How do you compare two distributions?
  - KL divergence, KL(p||q)

$$D_{KL}(p(x)||q(x)) = \sum_{x\in X} p(x)\ln\frac{p(x)}{q(x)}$$

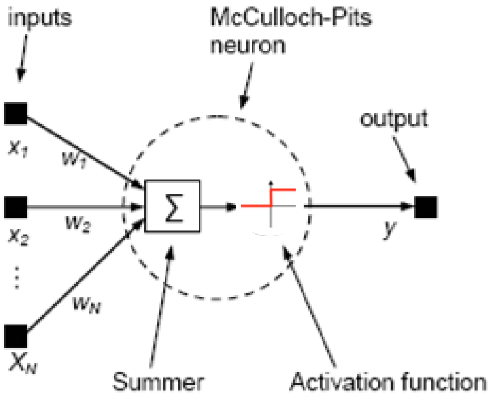
$= \sum p(\mathrm{x}) \ln p(\mathrm{x}) - p(\mathrm{x}) \ln q(\mathrm{x})$

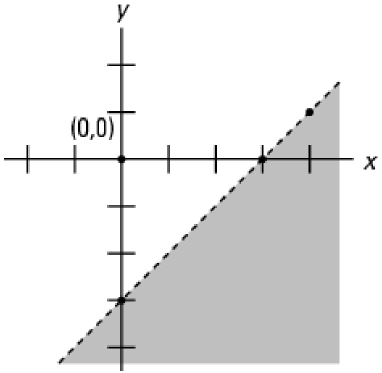
$= -H(p) + H(p,q)$

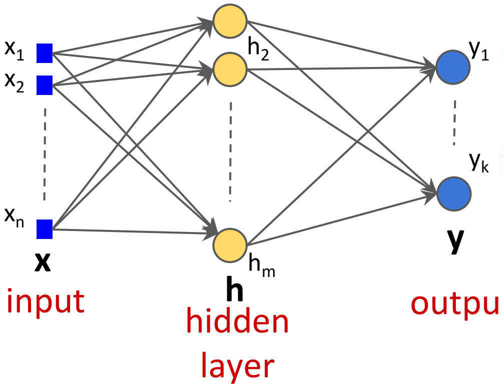
Entropy cross-entropy

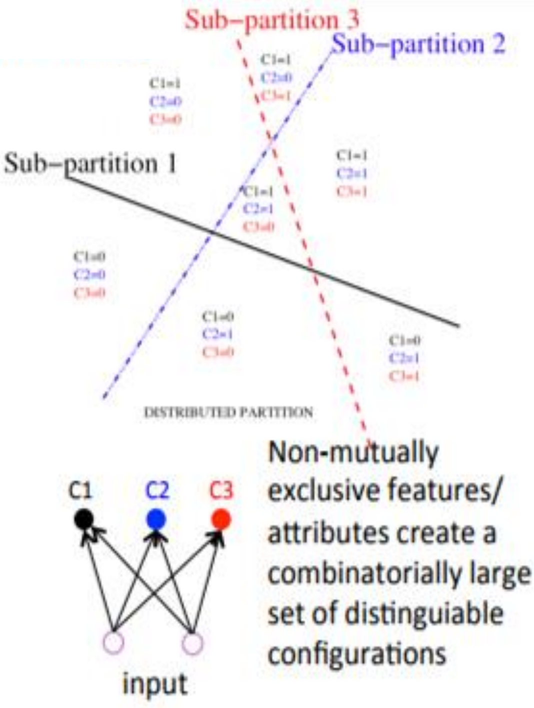
$$J(\theta) = \sum_{x_i,y_i} H(p(x_i), q(x_i))$$

What we learnt till now:







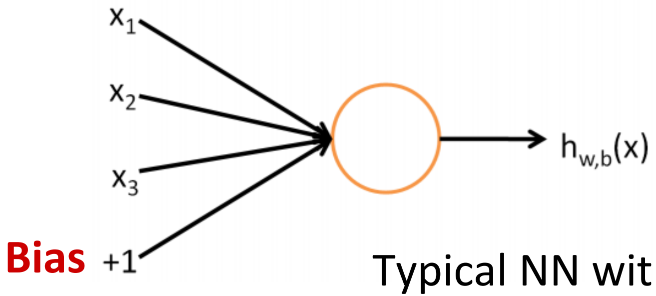


- Network specification
  - Number of hidden layers and units in each layer
  - Activation function for
    - Hidden layers
    - Output layers
  - Cost function

How to learn the network parameters?

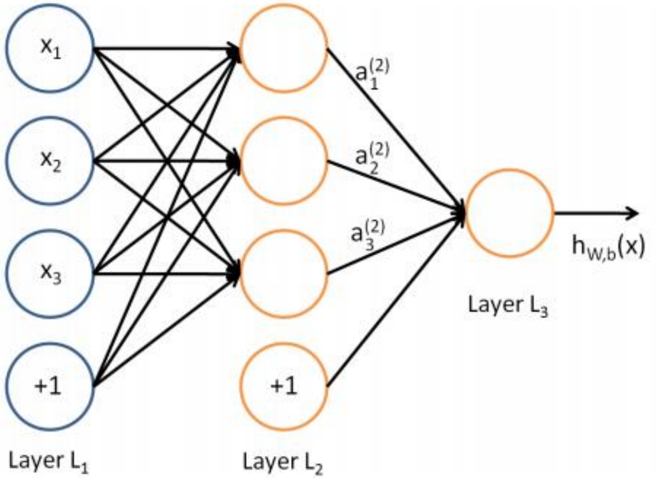
Error back propagation

How to learn the parameters?



$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

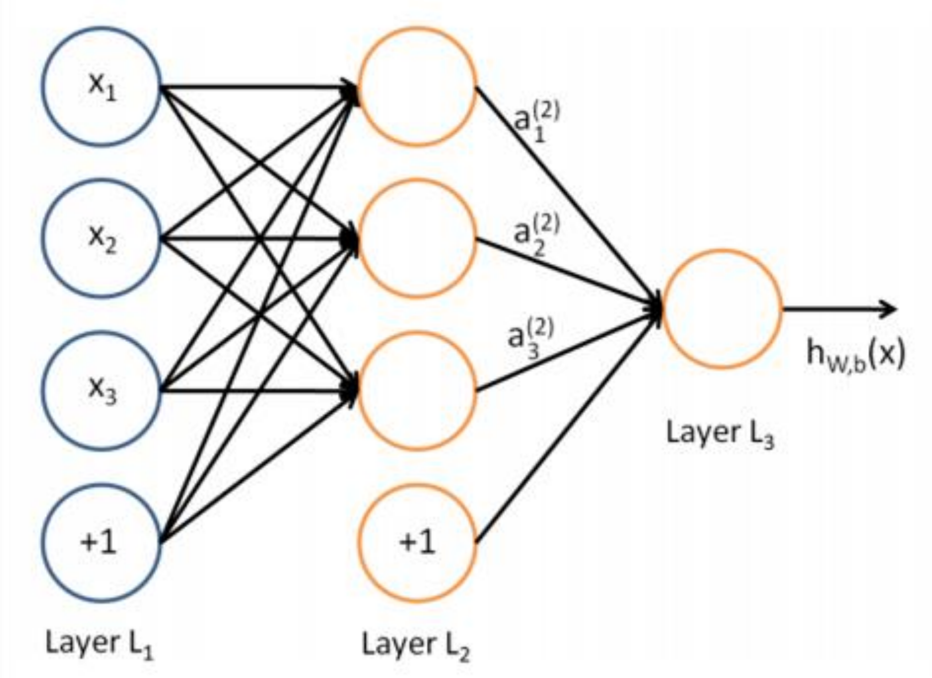
Typical NN with many such units



- One hidden layer
  - 3 neuron units
- One output

*Slide courtesy, [sparse autoencoder by Andrew Ng]

How to learn the parameters?



$W^{(1)}, b^{(1)}$ $W^{(2)}, b^{(2)}$

L_l – Layer l

$a_i^{(l)}$ – activation of unit i in layer l

$W_{ij}^{(l)}$ – Weight from j^{th} unit in l to i^{th} unit in $l+1$

$b_i^{(l)}$ – bias to unit i in layer $l+1$

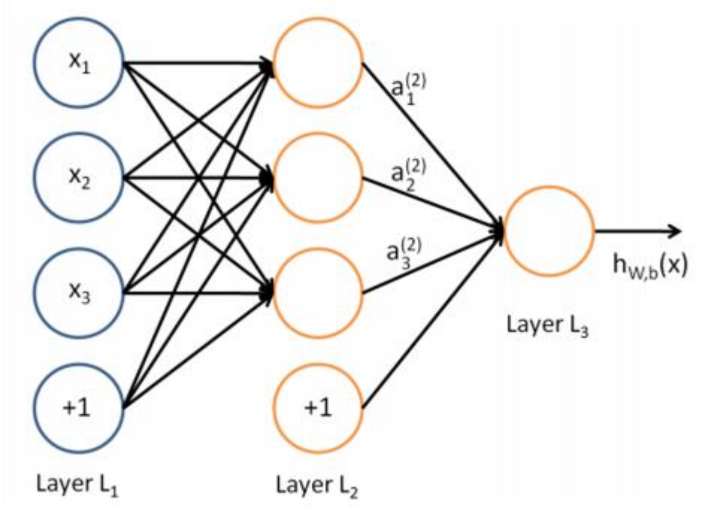
Parameters:

$$(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$$

$$W^{(1)} \in \mathbb{R}^{3\times 3}, W^{(2)} \in \mathbb{R}^{1\times 3}$$

*Slide courtesy, sparse autoencoder by Andrew Ng

How to learn the parameters?



Layer 2,

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})
\end{aligned}
$$

Layer 3,

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

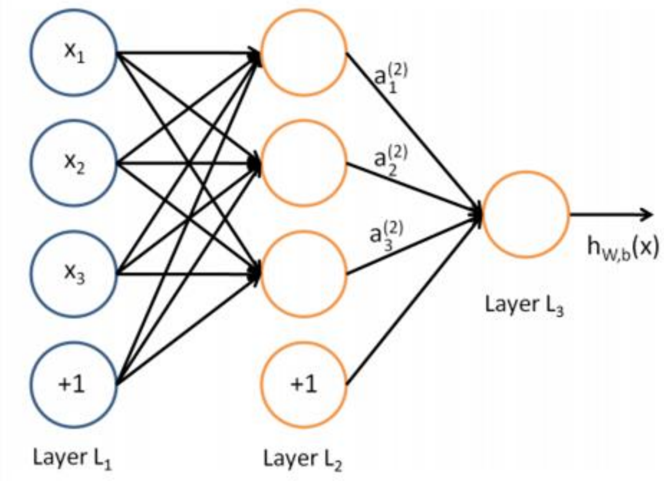
Simplification

Let, $z_i^{(l)}$ denote weighted sum for the activation $a_i^{(l)}$

$$a_i^{(l)} = f(z_i^{(l)})$$ *f*(.) applies the function point wise

*Slide courtesy, sparse autoencoder by Andrew Ng

How to learn the parameters?



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})
\end{aligned}
$$

$$
h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
$$

Let, $z_i^{(l)}$ denote weighted sum for the activation $a_i^{(l)}$

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
a^{(l+1)} &= f(z^{(l+1)})
\end{aligned}
$$

*Slide courtesy, [sparse autoencoder by Andrew Ng]

How to learn the parameters?

Given *m* training examples

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

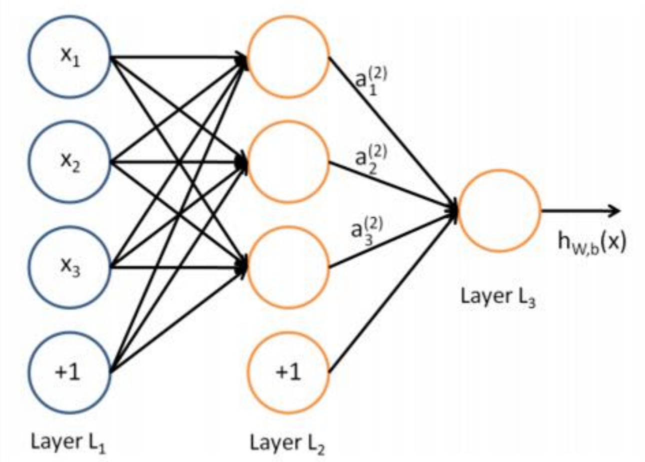
Minimize:

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

Assume we are solving a regression problem

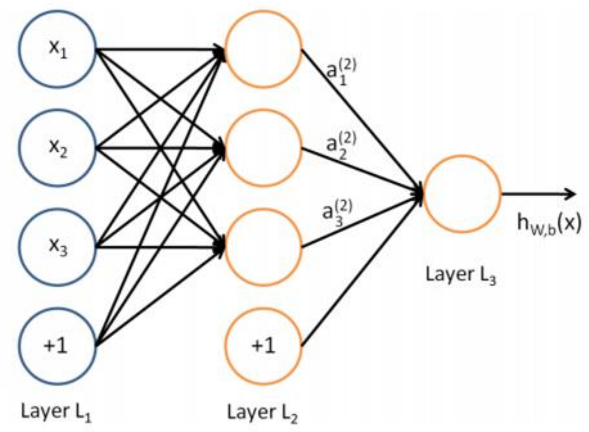
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right]$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right]$$



*Slide courtesy, [sparse autoencoder by Andrew Ng]

How to learn the parameters?



Minimize:
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right]$$
$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right]$$

Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$
$$b_i^{(l)} := b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

How to evaluate these partial derivatives?

Error back-propagation

*Slide courtesy, [sparse autoencoder by Andrew Ng]

Error back propagation

Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

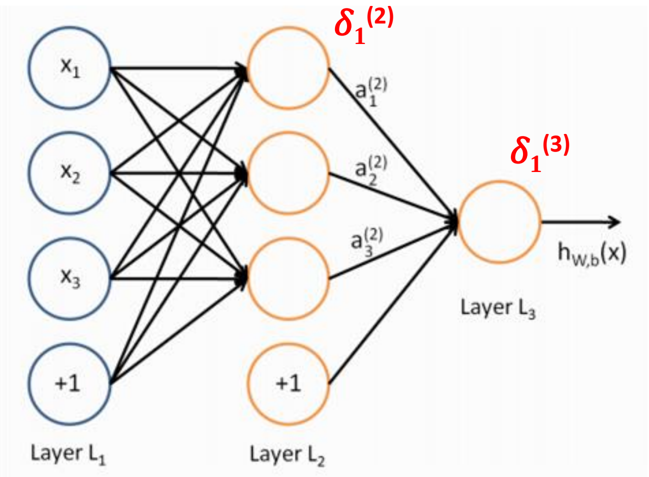
$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] = \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right]$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \frac{\partial}{\partial W_{ij}^{(l)}} \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right]$$

- ❖ Overall gradient can be computed by computing gradients wrt individual data terms
- ❖ Perform back propagation for computing individual data gradients
- ❖ Average them to get the overall gradient

Back-propagation algorithm



Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

Idea:

First, forward pass the data to calc. all responses

In backward pass, for each unit i in layer l calculate **error** term $\delta_i^{(l)}$ - measures how much unit i is responsible for output error

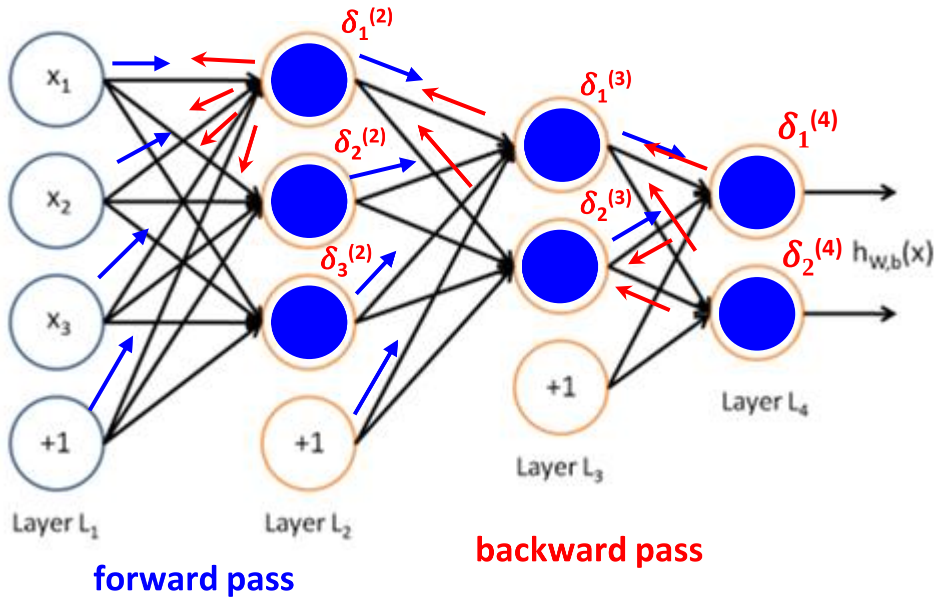
- For output unit in last layer (n_l), this is easy

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

- How to measure $\delta_i^{(l)}$ for hidden units?

*Slide courtesy, sparse autoencoder by Andrew Ng

Preview of back-propagation



1. Perform a feedforward pass
   - Computing activations L_1, L_2 and so on ...

2. For each output unit i in layer L_4 (output layer), set

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

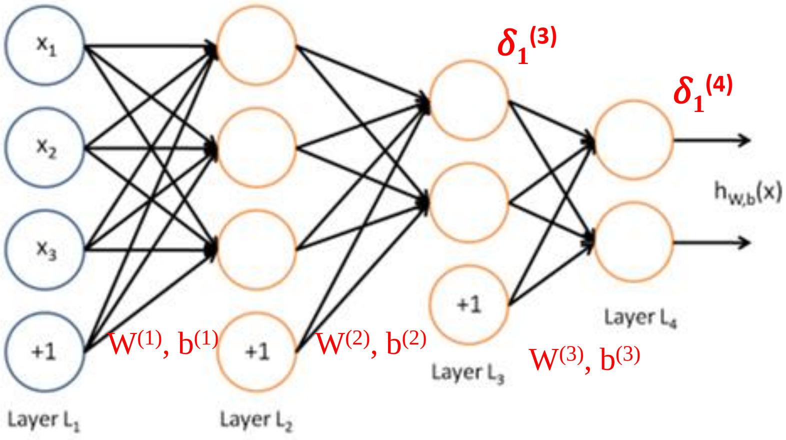
3. Starting from last but one layer to 2nd layer;
   $l = n_l - 1, n_l - 2, \ldots\ldots, 2$
   - For each node i in layer l, set $\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$

4. Compute the desired partial derivatives, as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \qquad \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

*Slide courtesy, sparse autoencoder by Andrew Ng

Back-propagation algorithm



Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$J(W,b;x,y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

$$h_{W,b}(x) = a^{(4)} = f(z^{(4)}); \; z^{(4)} = W^{(3)} a^{(3)} + b^{(3)}$$

For last layer:

$$\frac{\partial J}{\partial W_{ij}^{(3)}} = \underbrace{\boxed{\frac{\partial J}{\partial z_i^4}}}_{\delta_i^{(4)}} \underbrace{\boxed{\frac{\partial z_i^4}{\partial W_{ij}^{(3)}}}}_{\boxed{a_j^{(3)}}}$$

$$\boxed{\frac{\partial J}{\partial z_i^{\,4}} = -(y_i - a_i^{(4)}) \;.f'(z_i^4)}$$

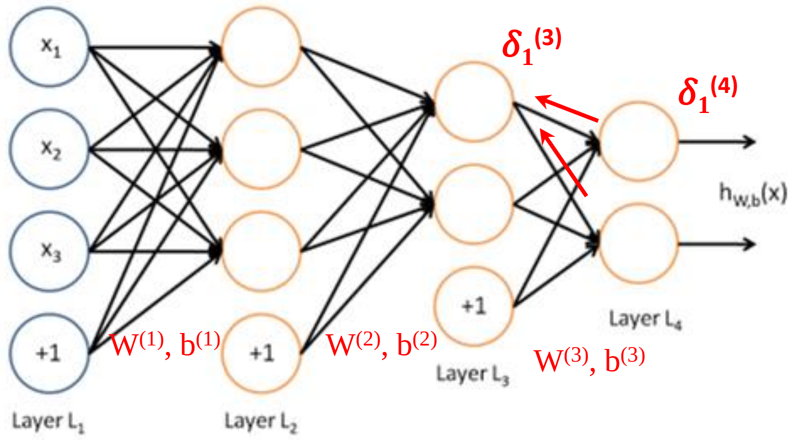
$\delta_i^{(4)}$ **error term**

$$\boxed{\frac{\partial z_i^4}{\partial W_{ij}^3} = a_j^{(3)}}$$

$$\frac{\partial J}{\partial W_{ij}^{(l)}} = \delta_i^{(l+1)} a_j^{(l)} \qquad \frac{\partial J}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

*Slide courtesy, sparse autoencoder by Andrew Ng

Back-propagation algorithm



$$h_{W,b}(x) = a^{(4)} = f(z^{(4)});\ z^{(4)} = W^{(3)}a^{(3)} + b^{(3)}$$

$$a^{(3)} = f(z^{(3)});\ z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$J(W,b;x,y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

For layers other than last:

$$\frac{\partial J}{\partial W_{ij}^{(2)}} = \boxed{\frac{\partial J}{\partial z_i^{(3)}}} \boxed{\frac{\partial z_i^{(3)}}{\partial W_{ij}^{(2)}}}$$

$a_j^{(2)}$

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

$$\frac{\partial J}{\partial W_{ij}^{(l)}} = \delta_i^{(l+1)} a_j^{(l)} \qquad \frac{\partial J}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

$\delta_i^{(3)}$
error term

$$\boxed{\begin{aligned} \frac{\partial J}{\partial z_i^{(3)}} &= \frac{\partial J}{\partial a_i^{(3)}} \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} \\ &= \left(\sum_j \frac{\partial J}{\partial z_j^{(4)}} \frac{\partial z_j^{(4)}}{\partial a_i^{(3)}} \right) f'(z_i^{(3)}) \end{aligned}}$$

$\delta_j^{(4)}$
Layer - (l+1)

$W_{ji}^{(3)}$

*Slide courtesy, [sparse autoencoder by Andrew Ng]

Back-propagation algorithm

1. Perform a feedforward pass
   - Computing activations L_1, L_2 and so on ...

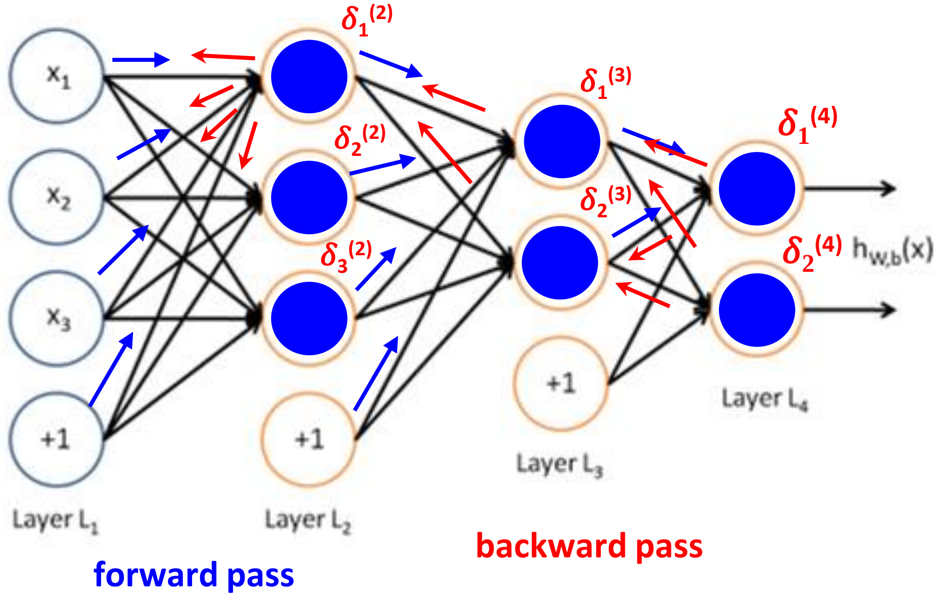
2. For each output unit i in layer L_4 (output layer), set

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

3. Starting from last but one layer to 2nd layer;
   $l = n_l - 1, n_l - 2, \ldots, 2$
   - For each node i in layer l, set $\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$

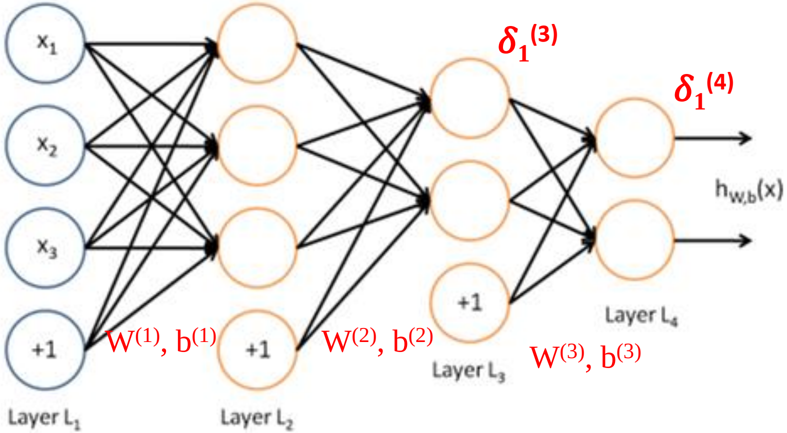
4. Compute the desired partial derivatives, as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \qquad \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$



*Slide courtesy, sparse autoencoder by Andrew Ng

Back-propagation algorithm



$$h_{W,b}(x) = a^{(4)} = f(z^{(4)}); \; z^{(4)} = W^{(3)}a^{(3)} + b^{(3)}$$

Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$J(W,b;x,y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

Partial derivatives:

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

$$\frac{\partial J}{\partial W_{ij}^{(l)}} = \delta_i^{(l+1)} a_j^{(l)} \qquad \frac{\partial J}{\partial b_i^{(l)}} = \delta_i^{(l+1)}$$

Matrix notation:

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$$

$$\frac{\partial J}{\partial W^{(l)}} = \delta^{(l+1)} (a^{(l)})^T \qquad \frac{\partial J}{\partial b^{(l)}} = \delta^{(l+1)}$$

*Slide courtesy, sparse autoencoder by Andrew Ng

Back-propagation algorithm

1. Perform a feedforward pass
   - Computing activations L_1, L_2 and so on ...

2. For each output unit i in layer L_4 (output layer), set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n)})$$

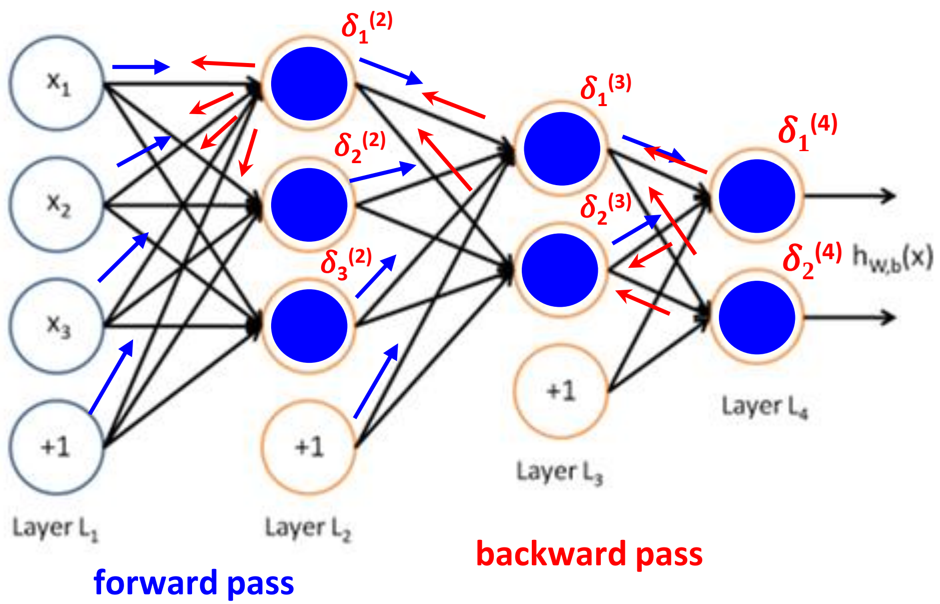
3. Starting from last but one layer to 2nd layer;

   $l = n_l\text{-}1, n_l\text{-}2, \ldots.., 2$

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives, as:

$$\begin{aligned} \nabla_{W^{(l)}} J(W,b;x,y) &= \delta^{(l+1)} (a^{(l)})^T, \\ \nabla_{b^{(l)}} J(W,b;x,y) &= \delta^{(l+1)}. \end{aligned}$$



*Slide courtesy, sparse autoencoder by Andrew Ng

Summary: Error back propagation

Gradient descent:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \frac{\partial}{\partial W_{ij}^{(l)}} \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right]$$

- ❖ Perform back propagation for computing individual gradient wrt each data
- ❖ Average them to get the overall gradient

Basic Neural Networks

What we will learn in the course about Neural Networks?

Introduction

- McCulloch and Pits model
- Rosenblatt’s perceptron

Perceptrons

- Geometry and linear separability
- XoR problem
- Multi-layer perceptron (MLP)

Training MLPs

- Error back propagation
- Loss functions