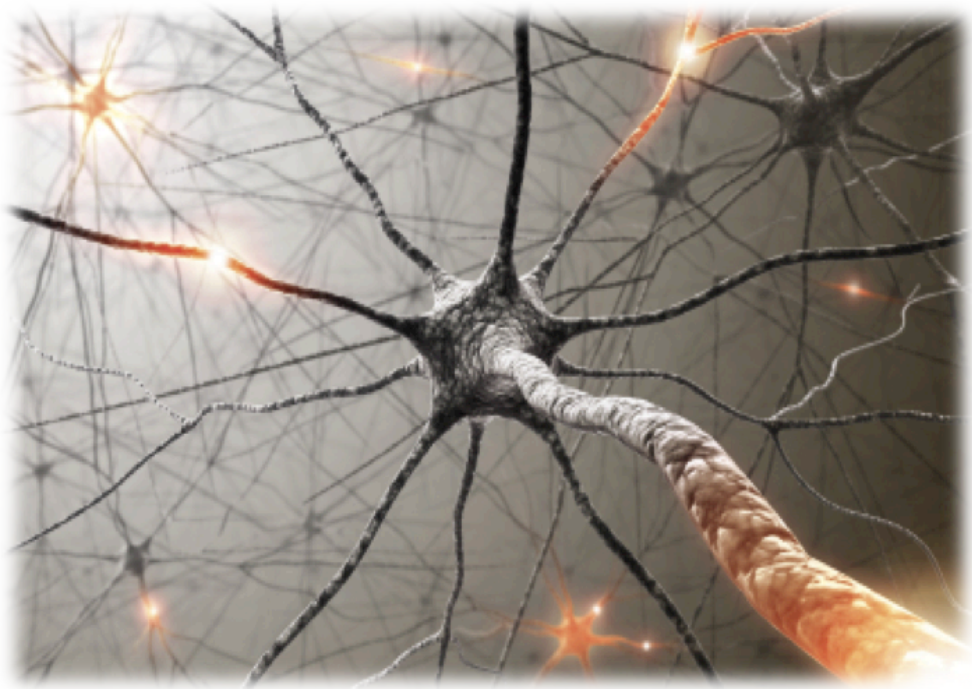
Neural Networks



Awni Hannun

CS221: Artificial Intelligence (Autumn 2013)

Outline

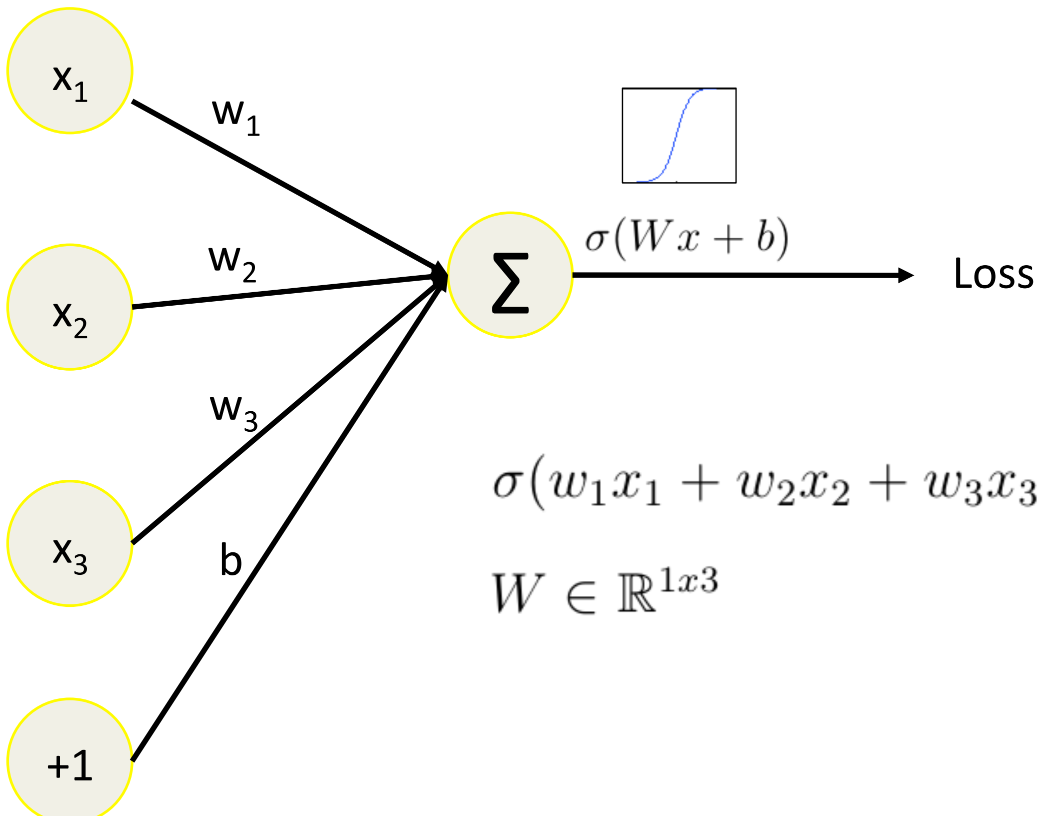
1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions

Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions

What is a Neural Network?

Logistic regression as a "neuron"

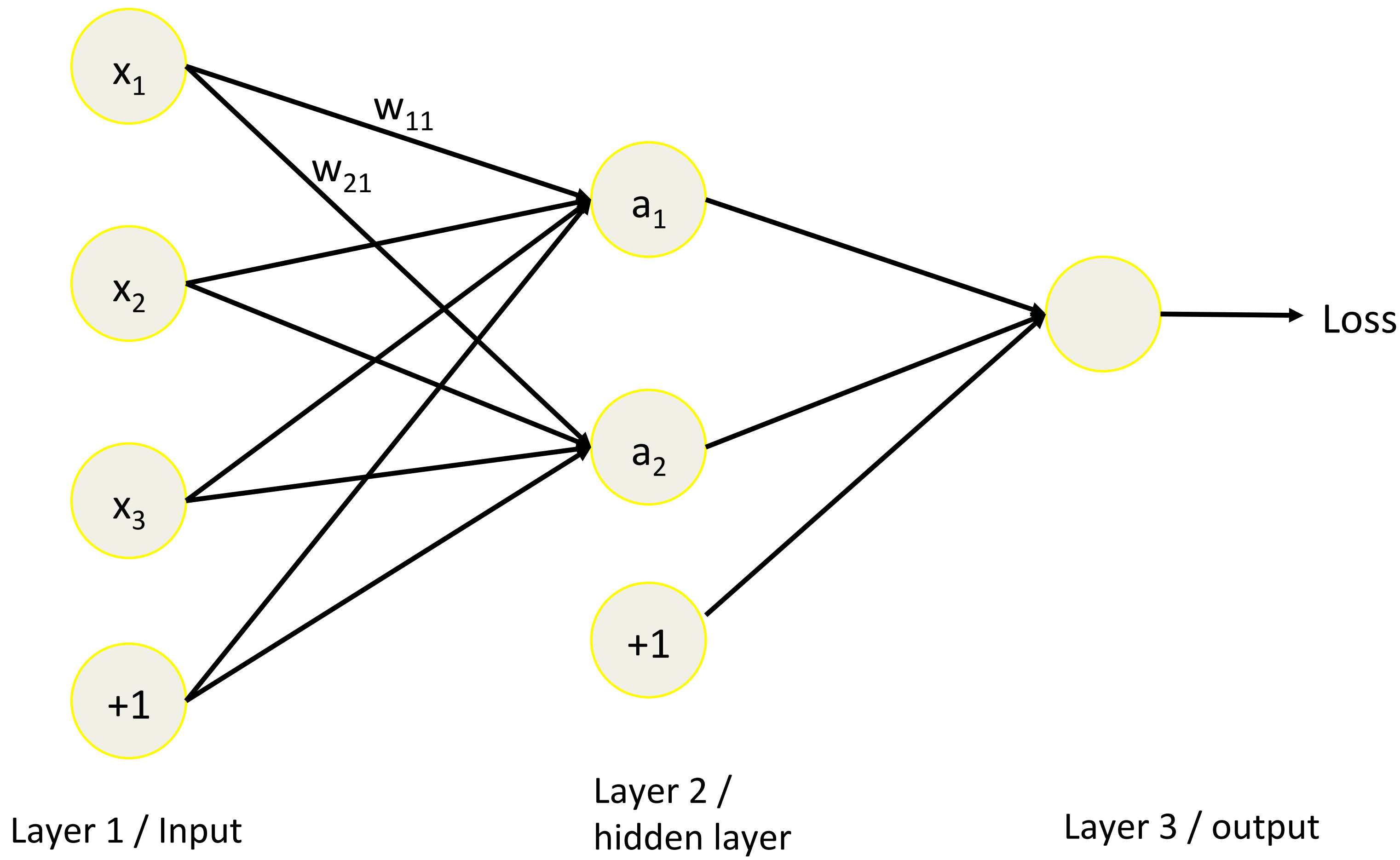


$$\sigma(w_1x_1 + w_2x_2 + w_3x_3 + b) = \sigma(Wx + b)$$

$$W \in \mathbb{R}^{1x3}$$

What is a Neural Network?

Stack many logistic units to create a Neural Network



Why Neural Networks?

Too many reasons, here are a few –

1. Highly expressive (universal approximators)

2. Deep learning, hierarchical representations

3. Supervised learning
   - -Binary classification
   - -Multiclass Classification
   - -Regression

4. Unsupervised Learning
   - -Feature learning
   - -Dimensionality reduction
   - -Generative models

Notation

$l = 1, ..., L$ - l-th layer

$W^{(l)} \in \mathbb{R}^{mxn}$ - weights for layer l

$b^{(l)} \in \mathbb{R}^{m}$ - bias for layer l

$\sigma(z)$ - activation function (for the folllowing σ is the sigmoid function)

$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$ - input to $(l+1)$-st layer

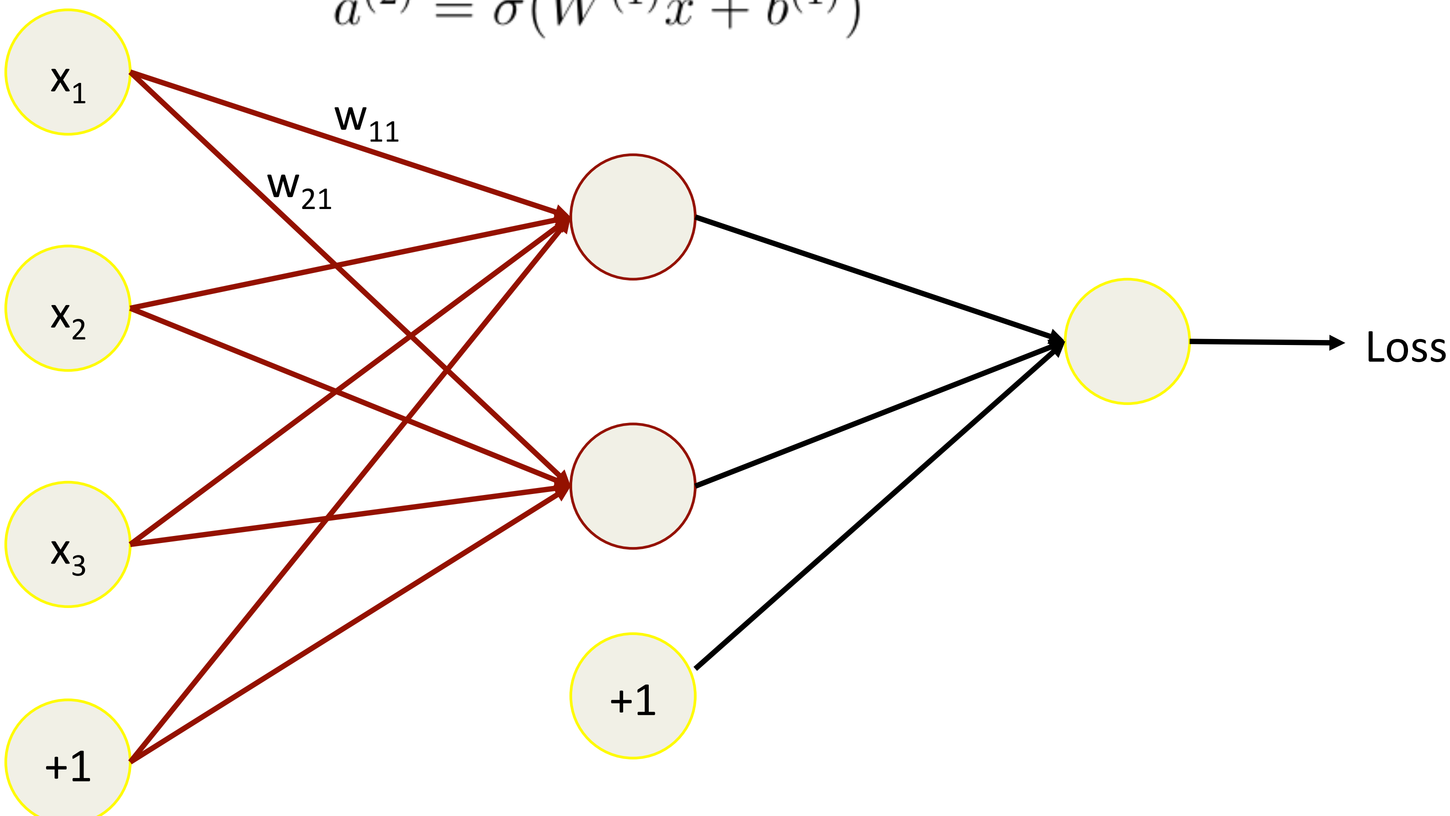
$a^{(l+1)} = \sigma(z^{(l+1)})$ - activation of $(l+1)$-st layer, let $a^{(1)} = x$

Outline

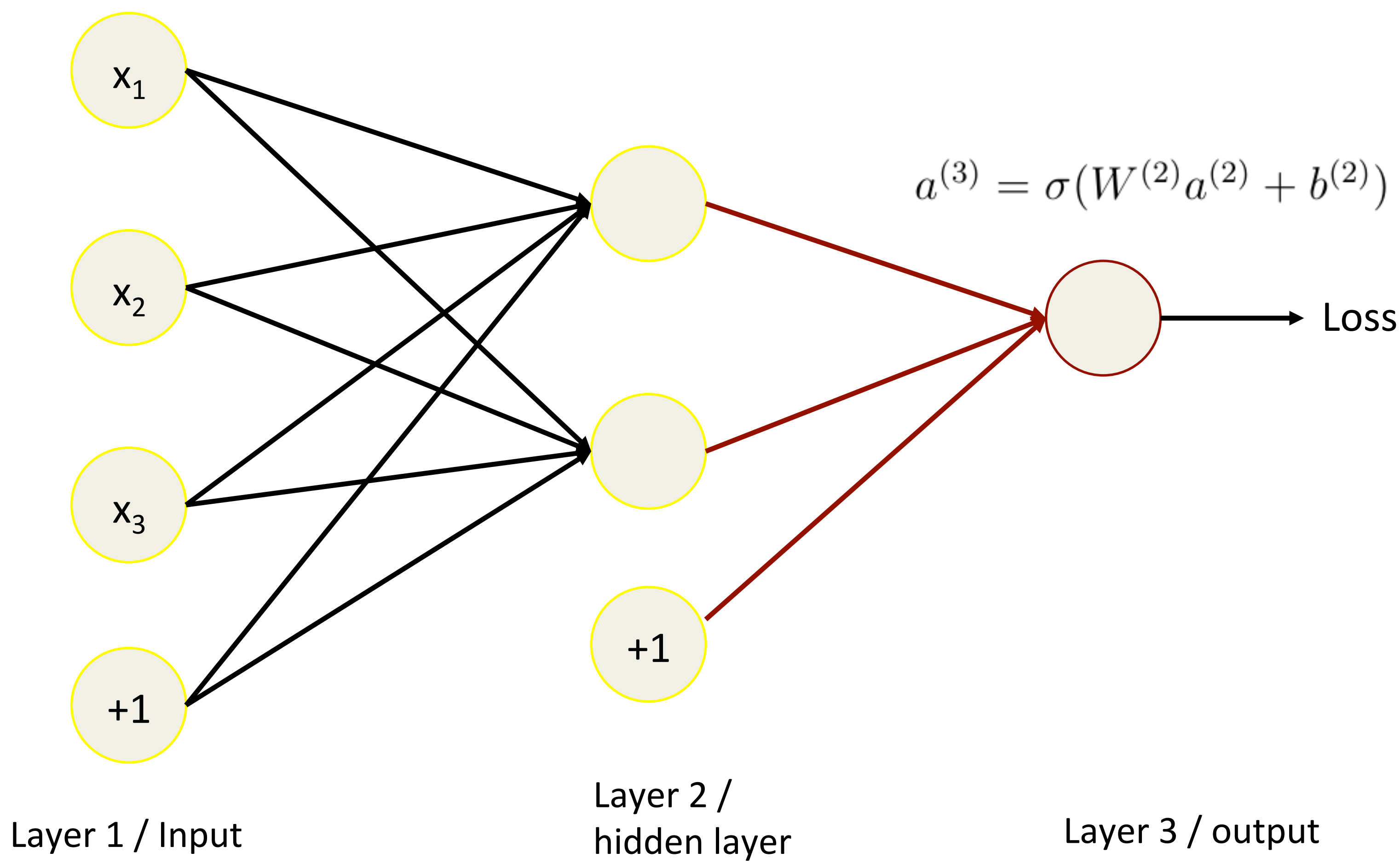
1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions

Forward Propagation

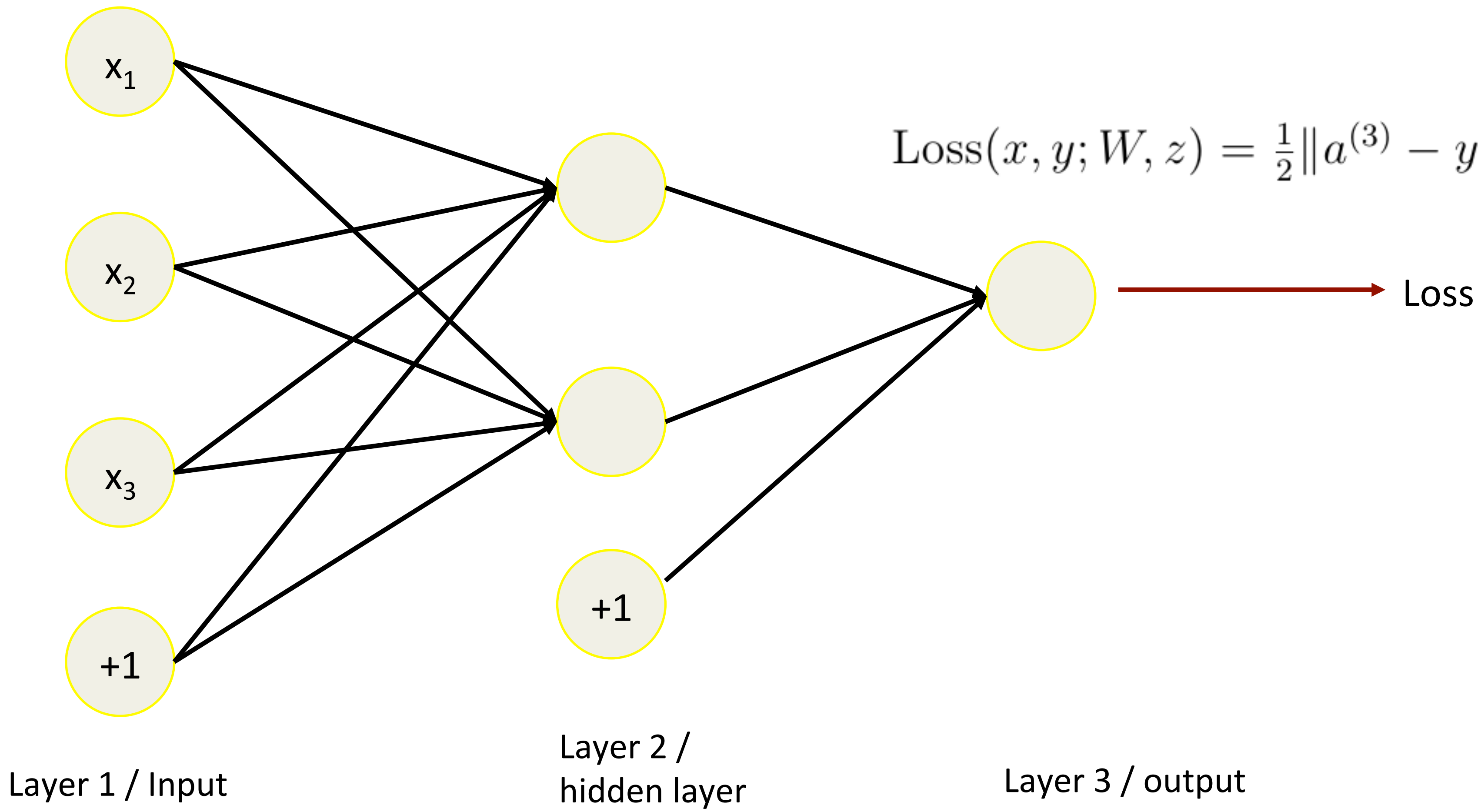
$$a^{(2)} = \sigma(W^{(1)}x + b^{(1)})$$



Forward Propagation



Forward Propagation

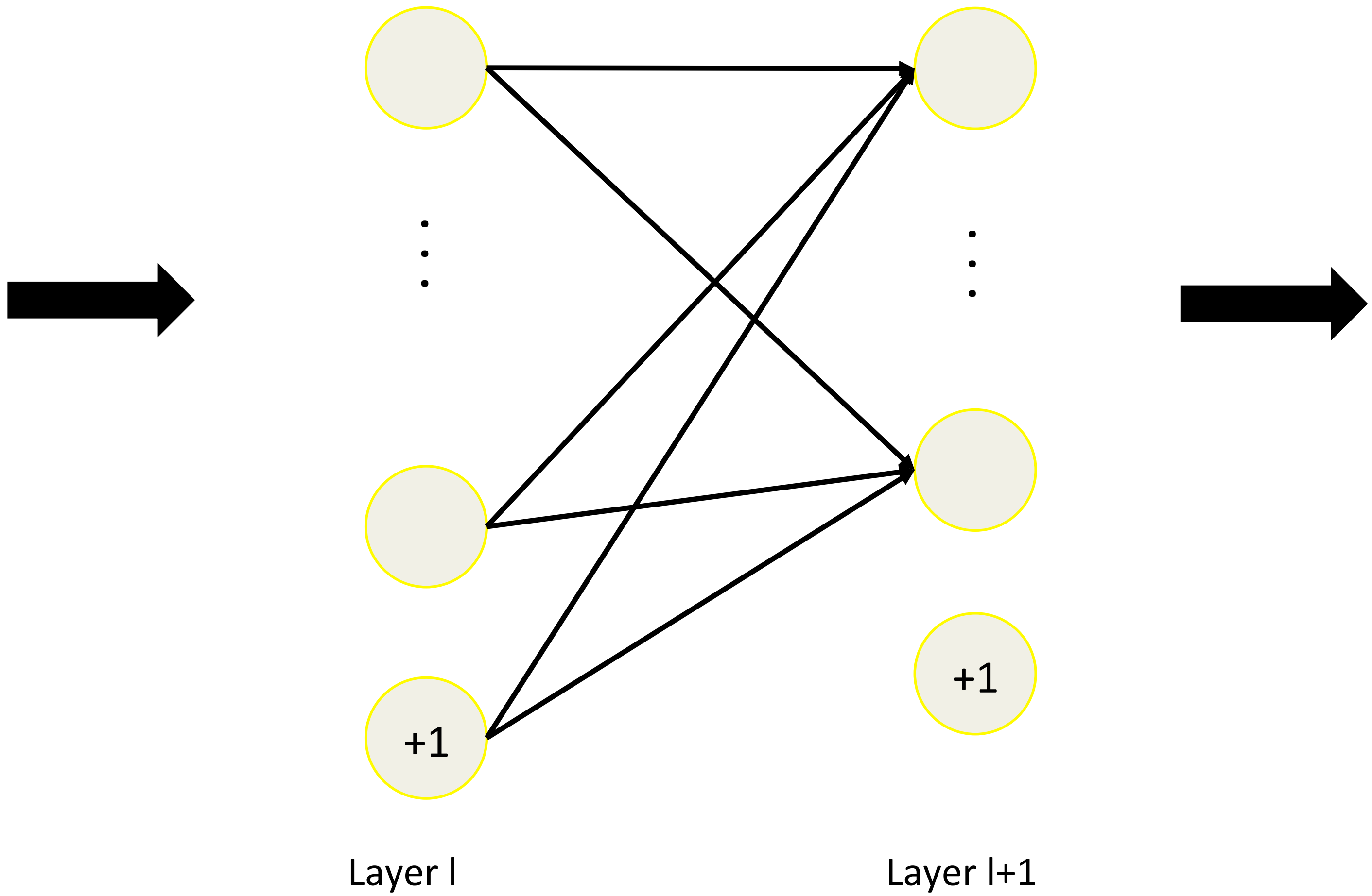


$$\text{Loss}(x, y; W, z) = \frac{1}{2}\|a^{(3)} - y\|_2^2$$

Forward Propagation

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = \sigma(z^{(l+1)})$$



Forward Propagation

Summary: Feed forward pass is just function composition + cost calculation

$$f(x) = \sigma\left(W^{(2)}\sigma\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)$$

$$\text{Loss}(x, y; W, z) = \frac{1}{2}\|f(x) - y\|_2^2$$

Outline

1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions

Training

Use gradient based updates to learn parameters for Network

$$\text{Loss}(x, y; W, z) = \frac{1}{2}\|a^{(L)} - y\|_2^2$$

$$W \leftarrow W - \eta \nabla_W \text{Loss}(x, y; W, b)$$

$$b \leftarrow b - \eta \nabla_b \text{Loss}(x, y; W, b)$$

Training: Backpropagation

Backpropagation

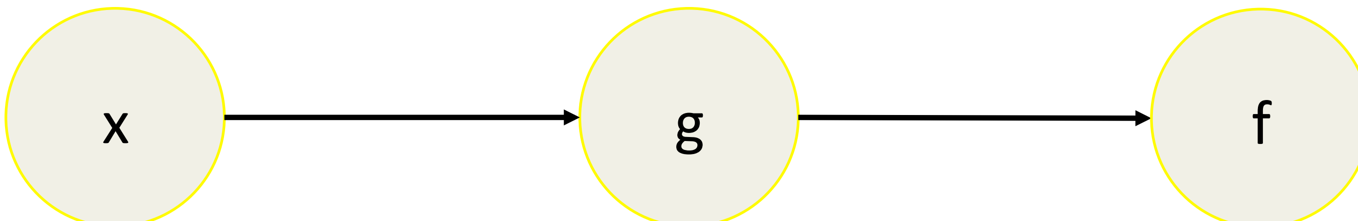
Algorithm to compute the derivative of the Loss function with respect to the parameters of the Network

$$\nabla_W \text{Loss}(x, y; W, b)$$

$$\nabla_b \text{Loss}(x, y; W, b)$$

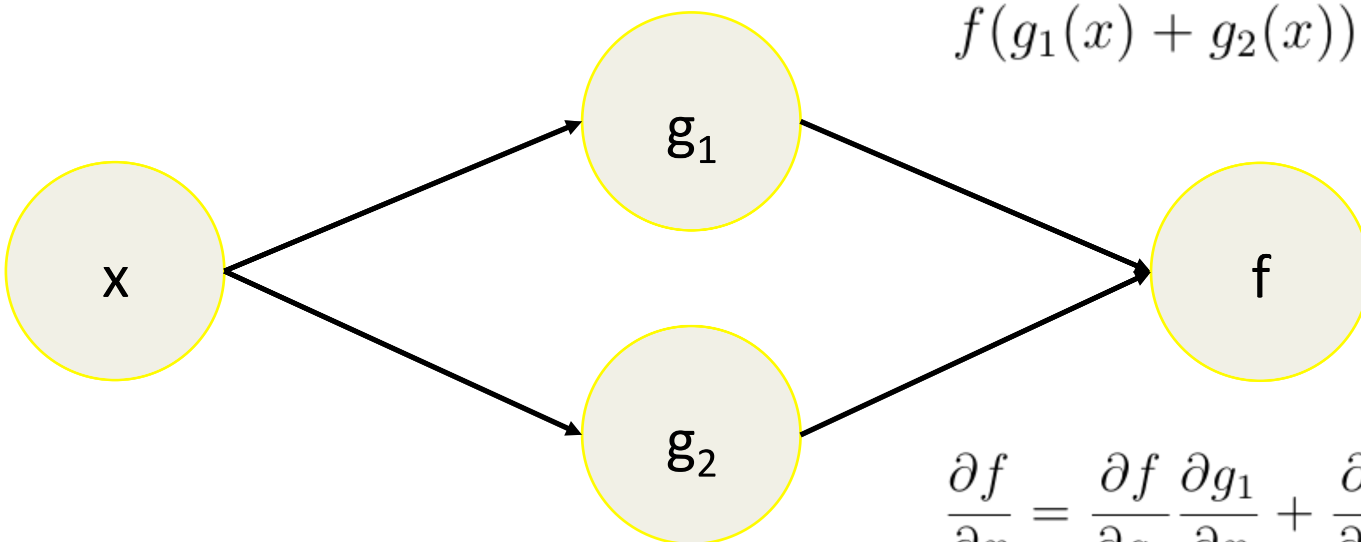
Chain Rule

$$(f \circ g)(x) = f(g(x))$$



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

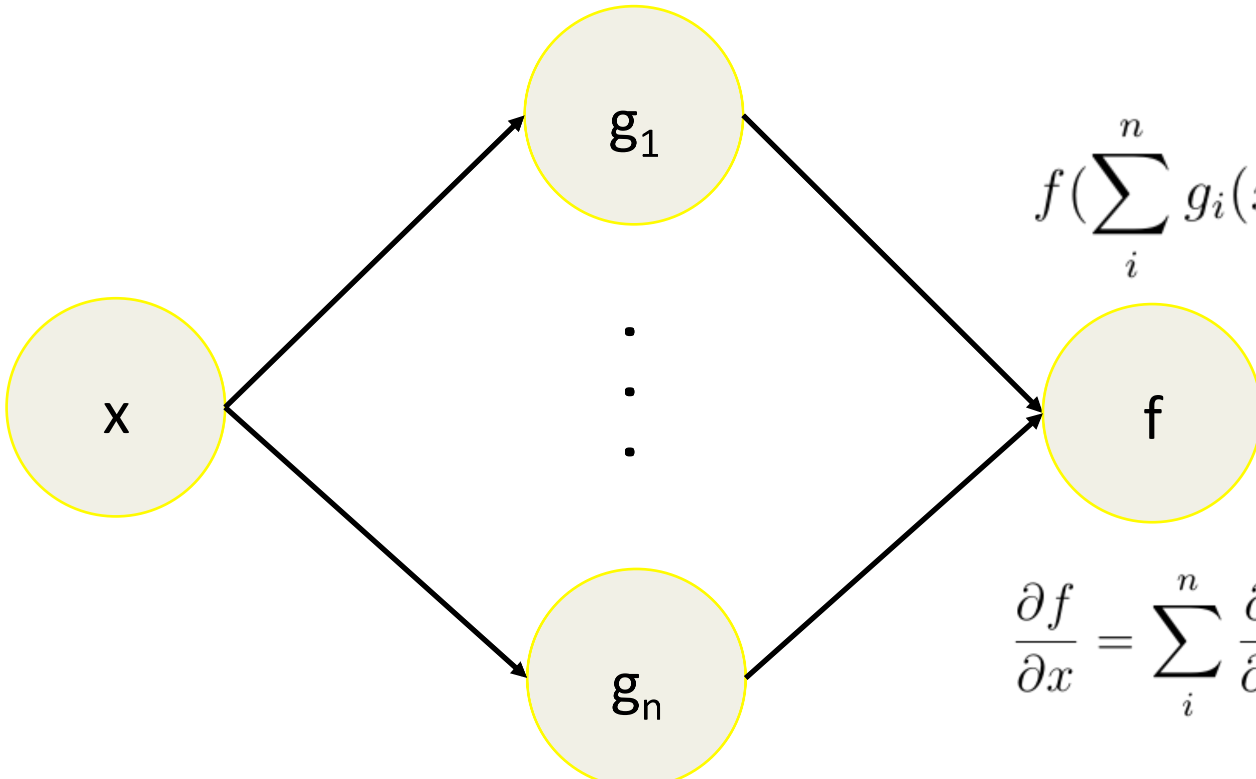
Chain Rule



$$f(g_1(x) + g_2(x))$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x}$$

Chain Rule



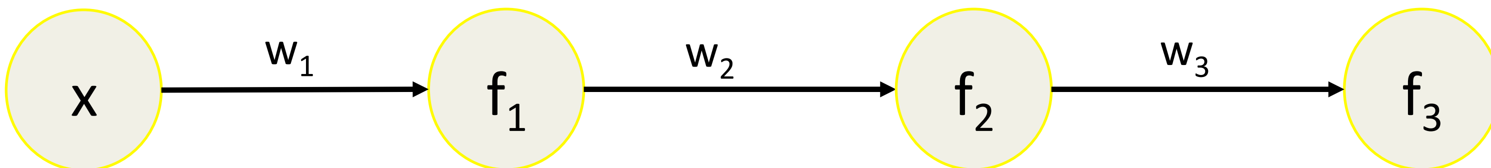
$$f\left(\sum_i^n g_i(x)\right)$$

$$\frac{\partial f}{\partial x} = \sum_i^n \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial x}$$

Backpropagation

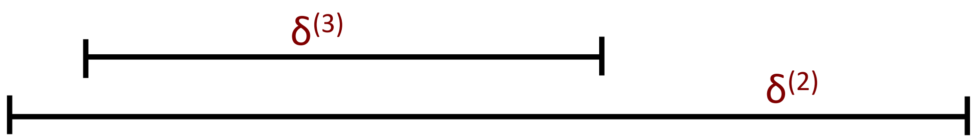
Idea: apply chain rule recursively

$$f(x) = f_3(w_3 f_2(w_2 f_1(w_1 x)))$$

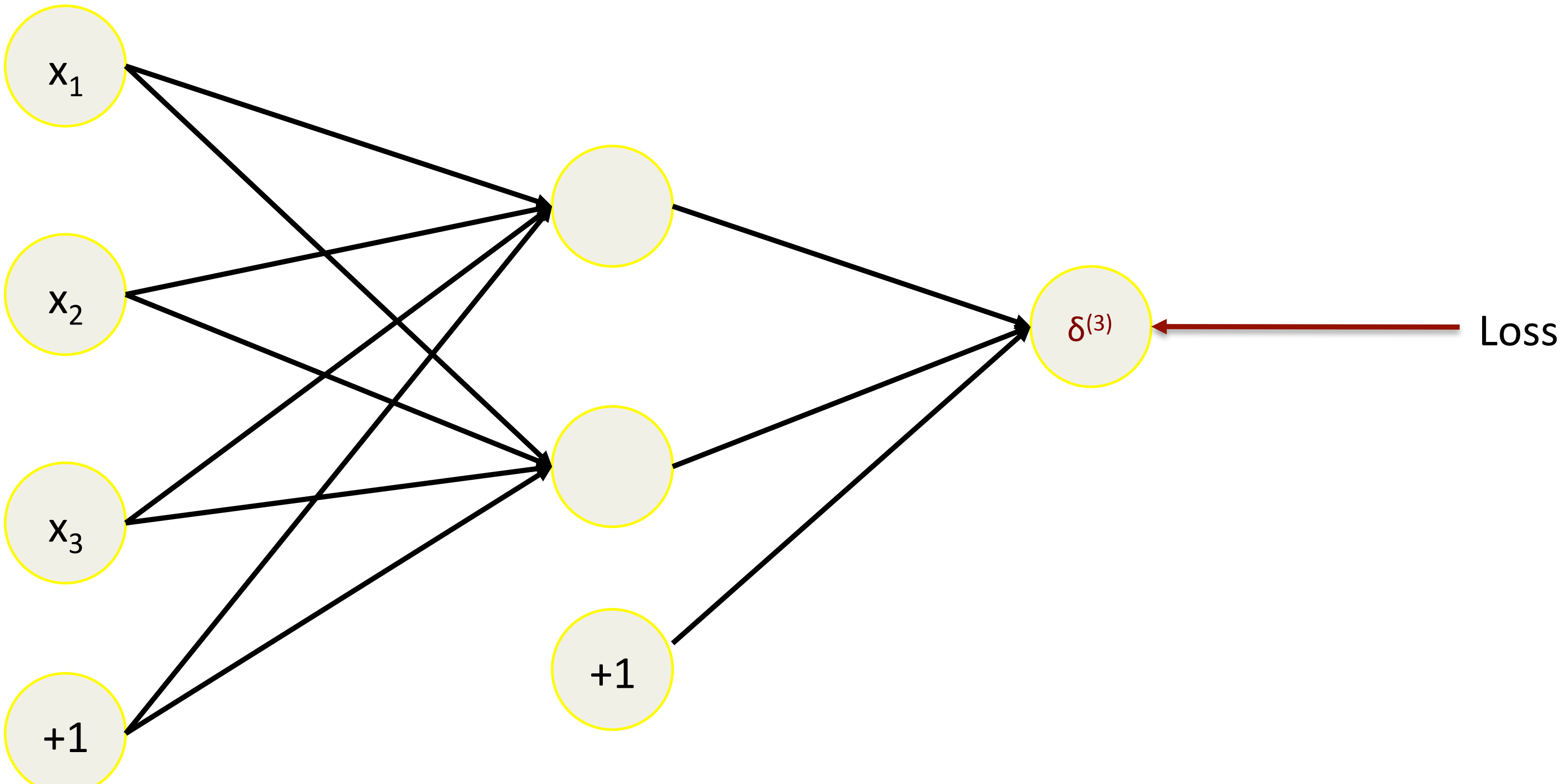


$$\frac{df}{dx} = f_3'(w_3 f_2(w_2 f_1(w_1 x))) \frac{d}{dx}(w_3 f_2(w_2 f_1(w_1 x)))$$

$$\frac{df}{dx} = w_3 f_3'(w_3 f_2(w_2 f_1(w_1 x))) f_2'(w_2 f_1(w_1 x)) \frac{d}{dx}(w_2 f_1(w_1 x))$$



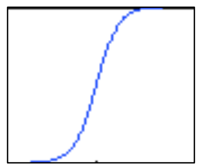
Backpropagation



Backpropagation

Derivative of sigmoid

$$\sigma(z) =$$

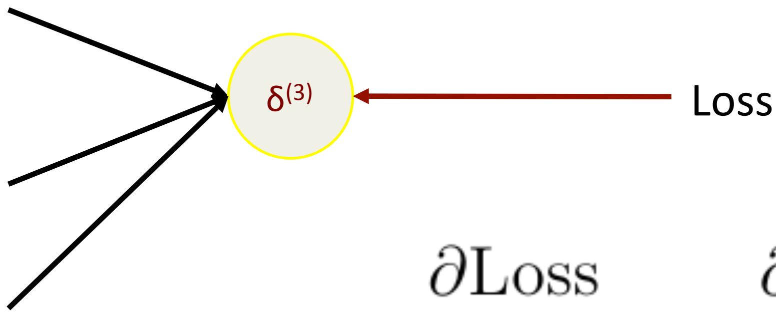


$$= \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \sigma(z)(1-\sigma(z))$$

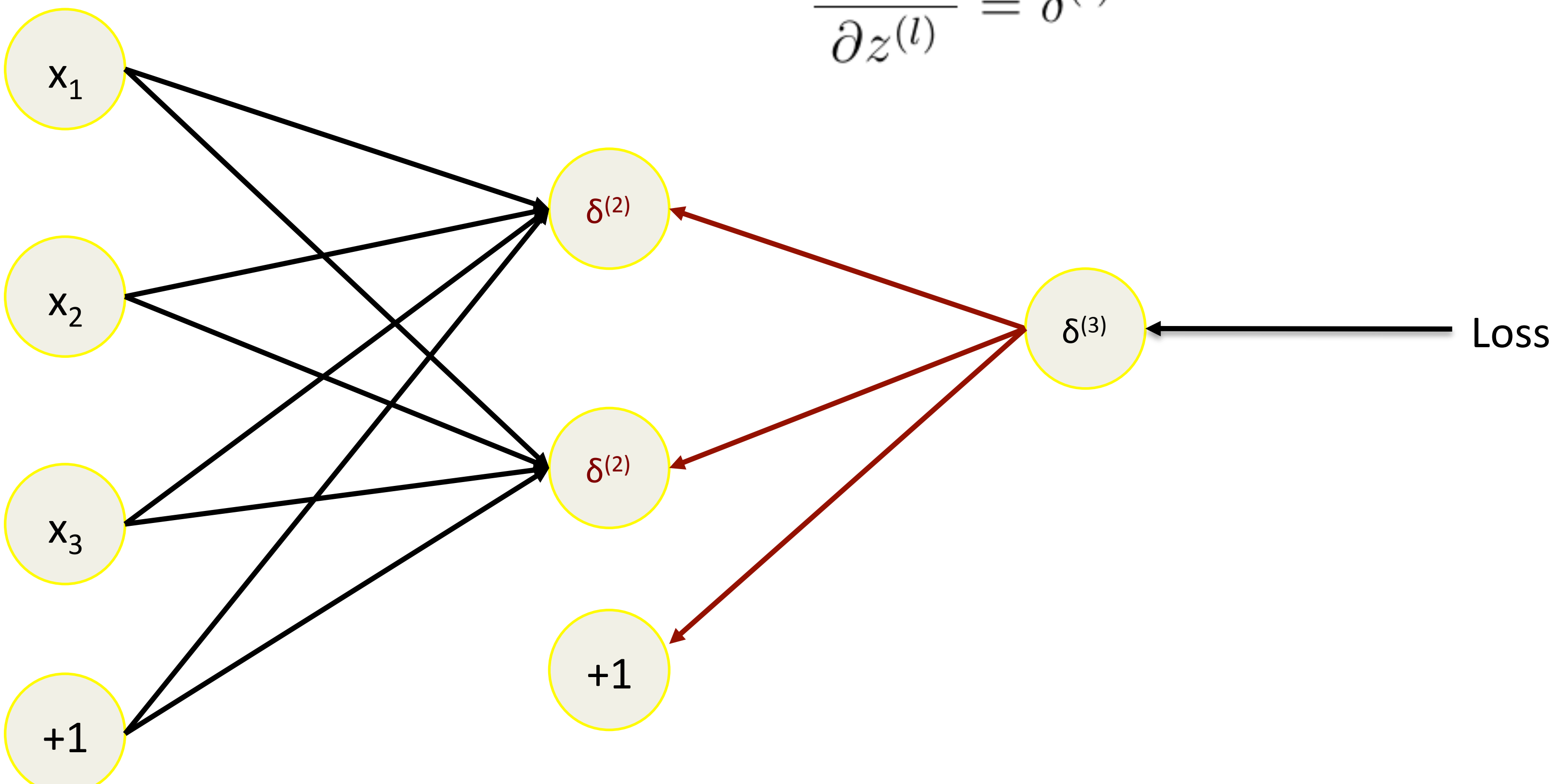
Backpropagation



$$
\begin{aligned}
\frac{\partial \text{Loss}}{\partial z^{(3)}} &= \frac{\partial}{\partial z^{(3)}} \left(\frac{1}{2} \|a^{(3)} - y\|_2^2 \right) \\
&= \frac{\partial}{\partial a^{(3)}} \left(\frac{1}{2} \|a^{(3)} - y\|_2^2 \right) \frac{\partial a^{(3)}}{\partial z^{(3)}} \\
&= \left(a^{(3)} - y\right) \left(a^{(3)}(1 - a^{(3)})\right) \\
&= \delta^{(3)}
\end{aligned}
$$

Backpropagation

$$\frac{\partial \text{Loss}}{\partial z^{(l)}} = \delta^{(l)}$$



Backpropagation

Recursively compute delta at each hidden layer

$$\delta_i^{(l)} = \frac{\partial \text{Loss}}{\partial z_i^{(l)}} = \frac{\partial \text{Loss}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}}$$

$$\begin{aligned} \frac{\partial \text{Loss}}{\partial a_i^{(l)}} &= \sum_{j=1}^{m} \frac{\partial \text{Loss}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} \\ &= \sum_{j=1}^{m} \delta_j^{(l+1)} w_{ji}^{(l)} \\ &= (w_i^{(l)})^T \delta_j^{(l+1)} \end{aligned}$$

$$\frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} = (a_i^{(l)})(1 - a_i^{(l)})$$

$$\delta_i^{(l)} = (w_i^{(l)})^T \delta_j^{(l+1)} (a_i^{(l)})(1 - a_i^{(l)})$$

$$\delta^{(l)} = (w^{(l)})^T \delta^{(l+1)} \circ (a^{(l)})(1 - a^{(l)})$$

Backpropagation

Compute gradient of Loss w.r.t. weights

$$\frac{\partial \text{Loss}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}}$$

$$= \frac{\partial \text{Loss}}{\partial z_i^{(l+1)}} \frac{\partial (W^{(l)} a^{(l)} + b^{(l)})_i}{\partial w_{ij}^{(l)}}$$

$$= \delta_i^{(l+1)} a_j^{(l)}$$

$$\nabla_{W^{(l)}} \text{Loss} = \delta^{(l+1)} \left(a^{(l)}\right)^T$$

$$\nabla_{b^{(l)}} \text{Loss} = \delta^{(l)}$$

Backpropagation

<u>Backpropagation Algorithm</u>

1. Feed forward input (x, y), computing activation for layers $l = 2, ..., L$

2. For the output layer, L set:

$$\delta^{(L)} = a^{(L)}(1 - a^{(L)})(a^{(L)} - y)$$

3. For layers $l = 2, ..., L - 1$ set:

$$\delta^{(l)} = (W^{(l)})^T \delta^{(l+1)} \circ a^{(l)}(1 - a^{(l)})$$

4. Compute gradient with respect to parameters $W^{(l)}, b^{(l)}$ as:

$$\nabla_{W^{(l)}} \text{Loss}(x, y; W, b) = \delta^{(l+1)}(a^{(l)})^T$$

$$\nabla_{b^{(l)}} \text{Loss}(x, y; W, b) = \delta^{(l+1)}$$

Training: Backpropagation

<u>SGD Algorithm</u>

run SGD as usual using backpropagation to compute derivative of Loss w.r.t. params

For each input (x, y) in the training set

$$\nabla_{W,b}\text{Loss}(x, y; W, b) = \texttt{Backpropagate}(x, y)$$

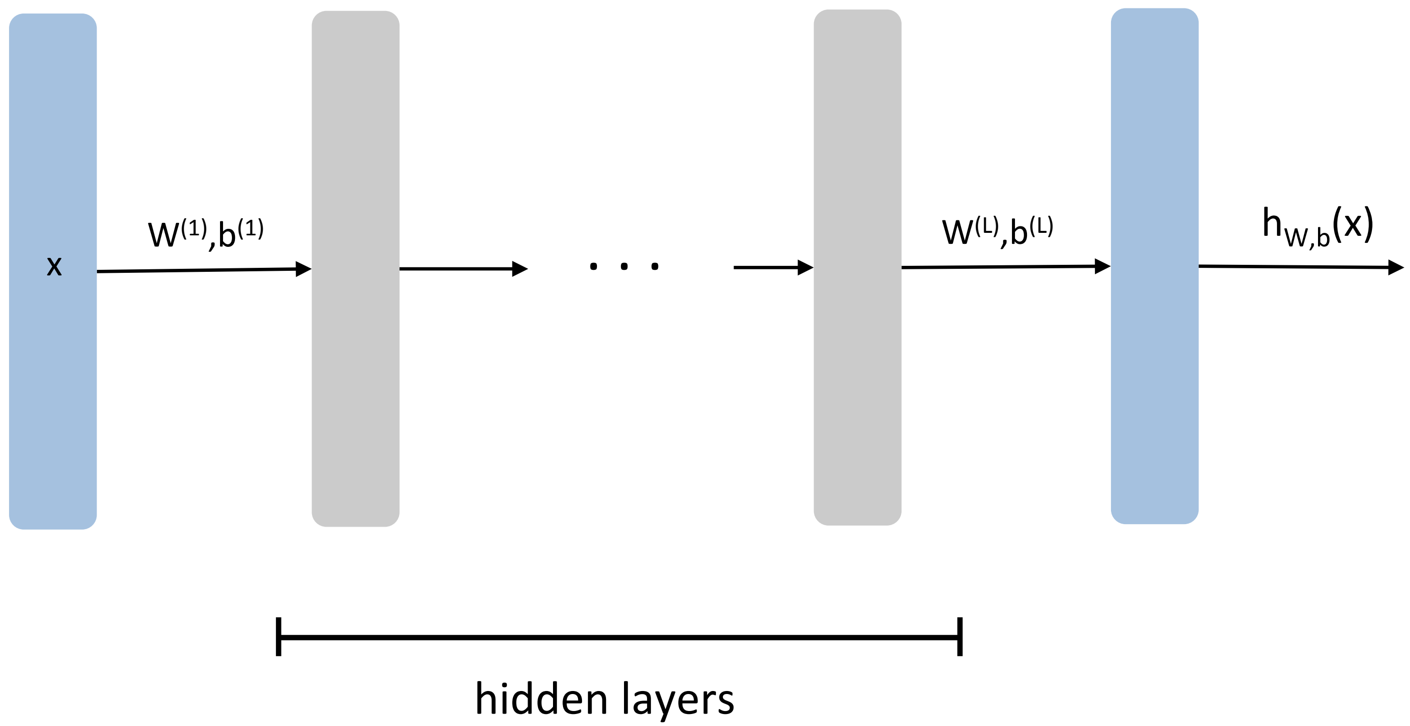
$$W \leftarrow W - \eta \nabla_W \text{Loss}(x, y; W, b)$$

$$b \leftarrow b - \eta \nabla_b \text{Loss}(x, y; W, b)$$

Outline

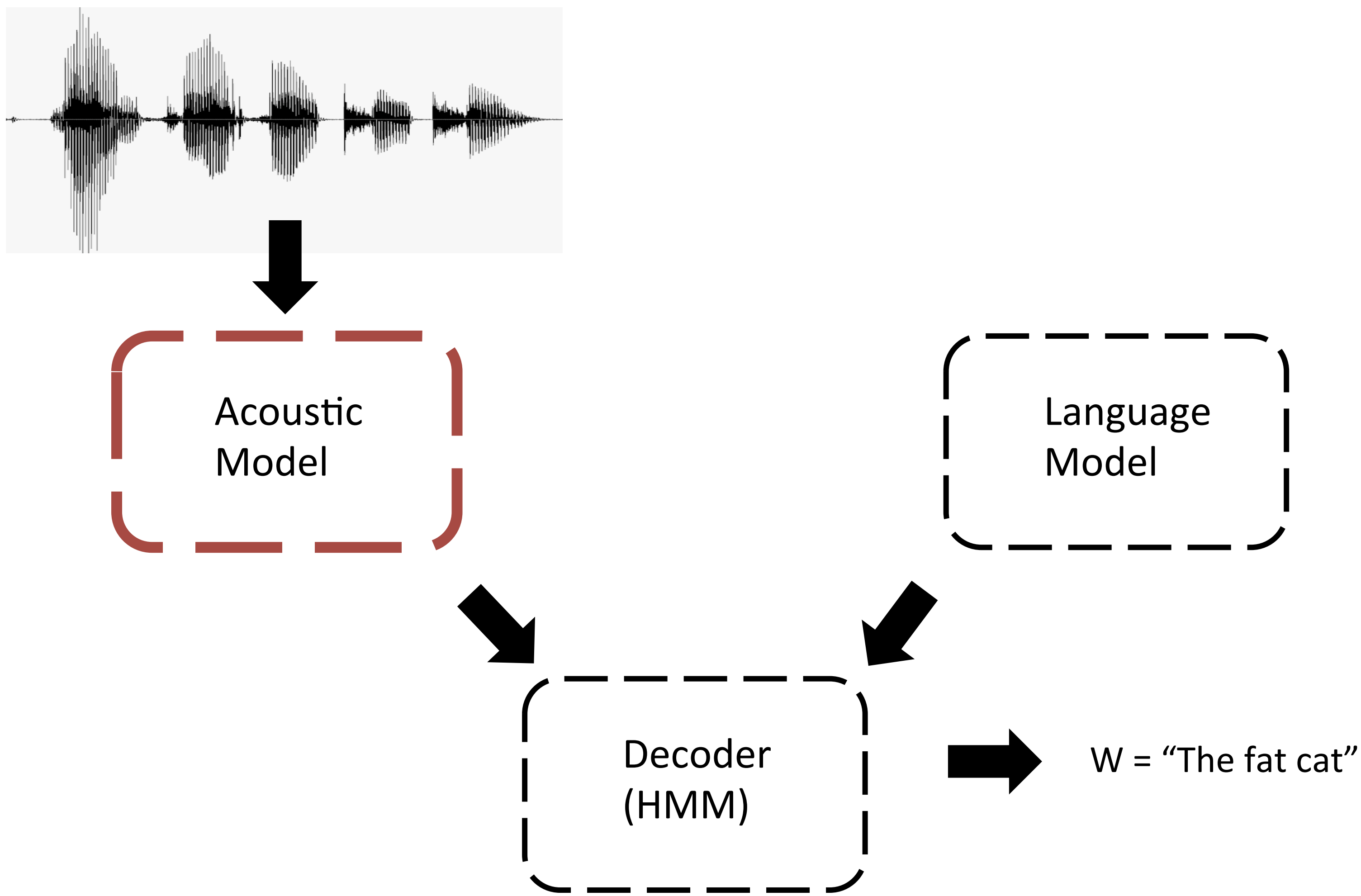
1. Overview: Neural Networks
2. Feed forward calculation
3. Training : Backpropagation
4. Applications and Extensions

Model: Deep NN



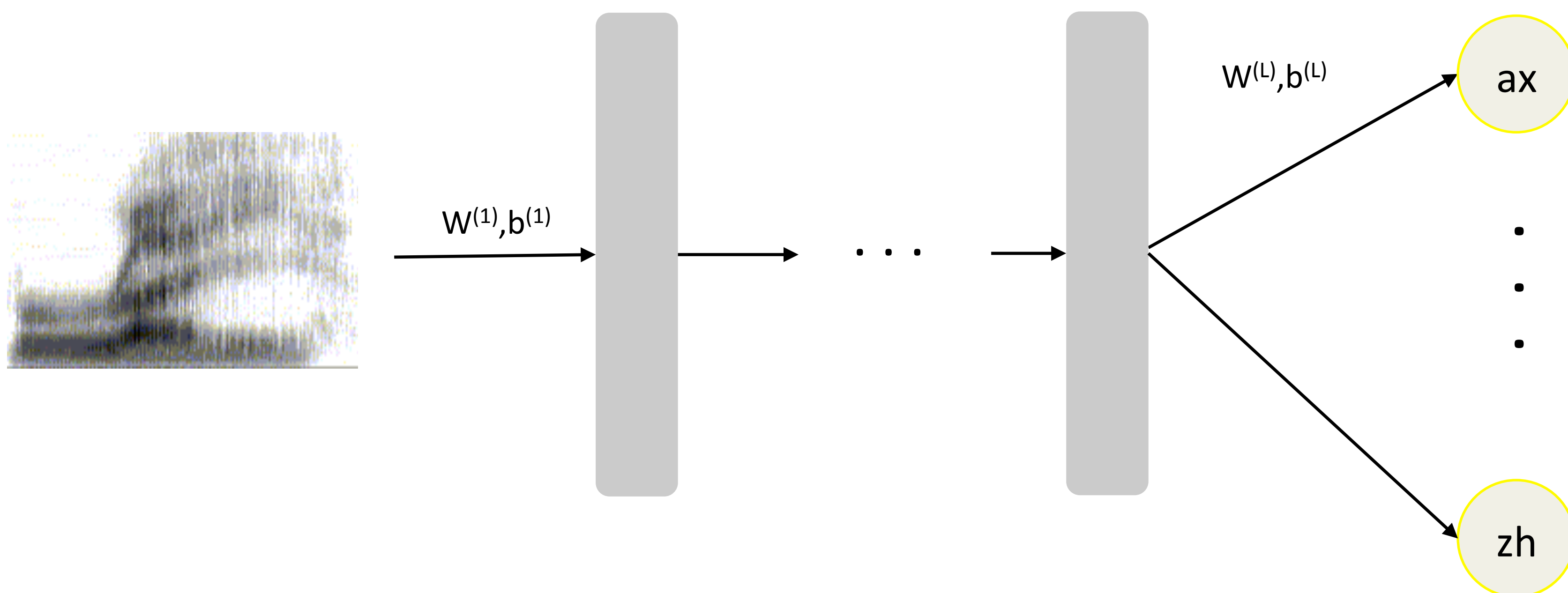
Applications: Deep NN

Speech Recognition



Applications: Deep NN

Speech Recognition



Convert output of NN to observation probabilities using Bayes Thm

$$p(o|s) = \frac{p(s|o)p(o)}{p(s)}$$

Model: Convolutional NN

Ideas:

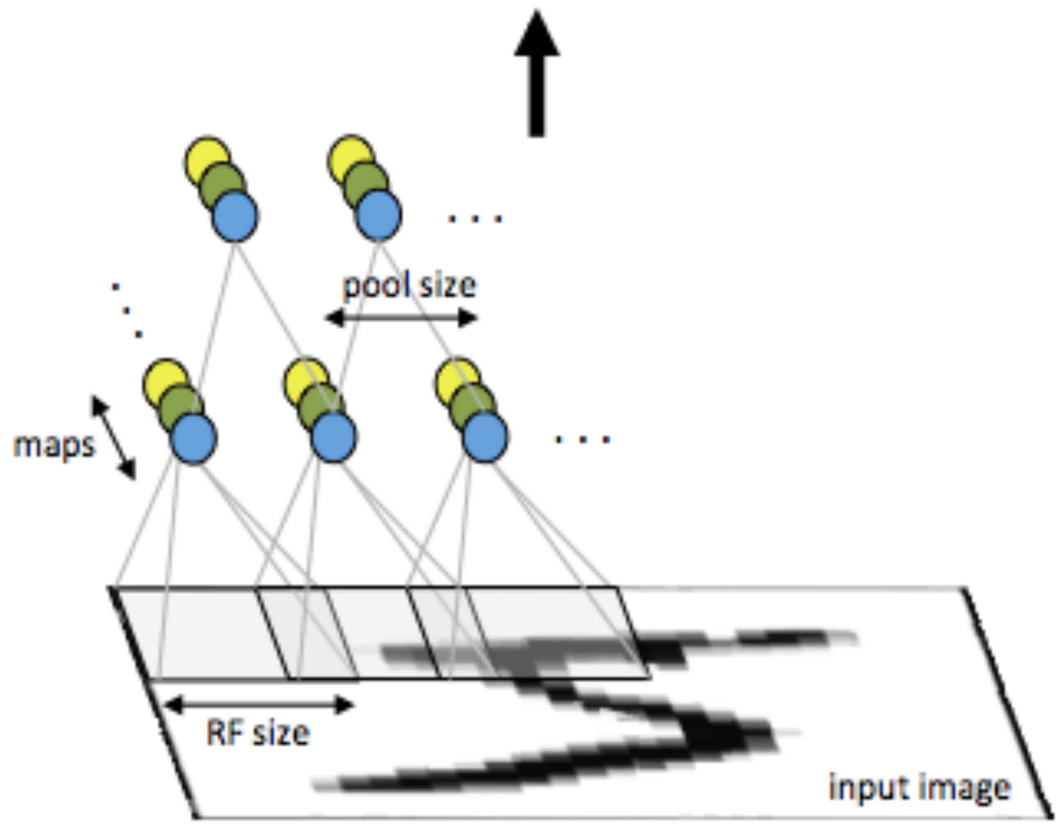
-small receptive fields

-tie weights, re-use features

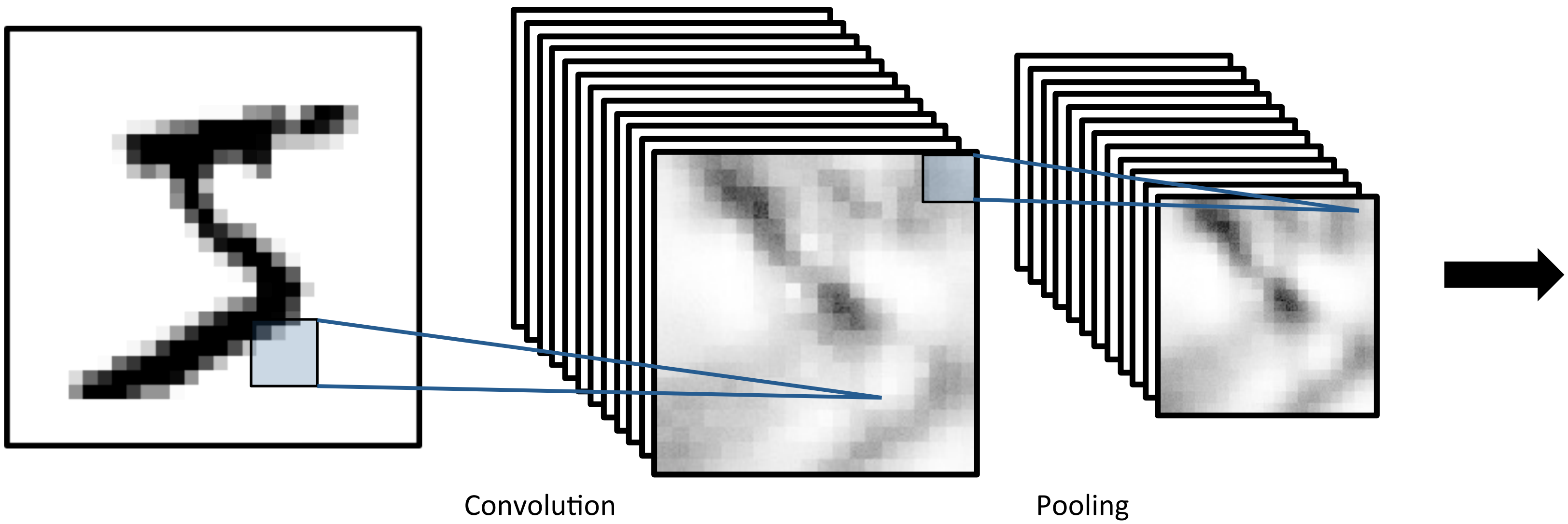
-pooling gives translation invariance

-[convolution demo]

-[pooling demo]



Model: Convolutional NN



Applications: Convolutional NN

Computer Vision

ImageNet object recognition of 22k classes (state-of-the-art) - ~37% error rate

beer bottle

water bottle

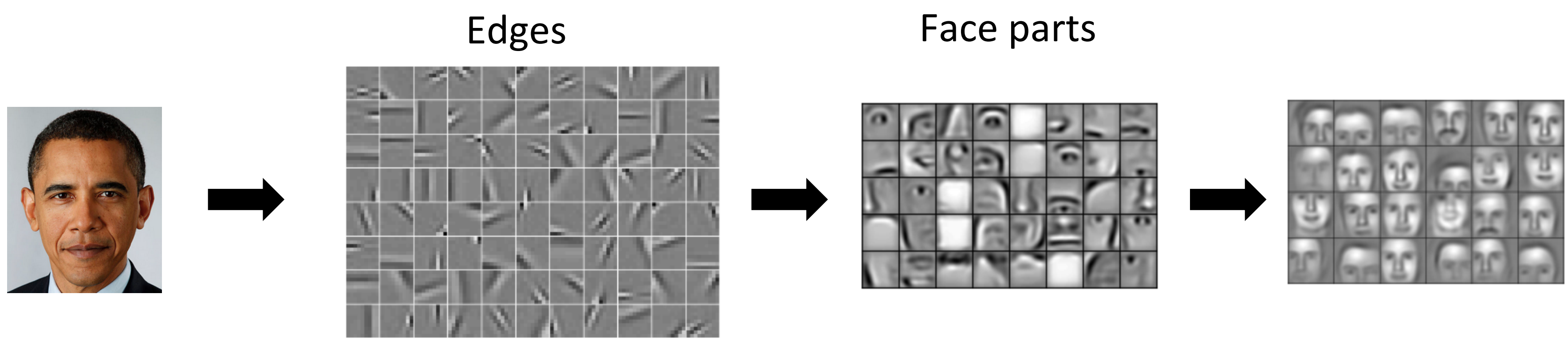
soda bottle

Egyptian cat

Tabby cat

Applications: Convolutional NN

Computer Vision – feature learning

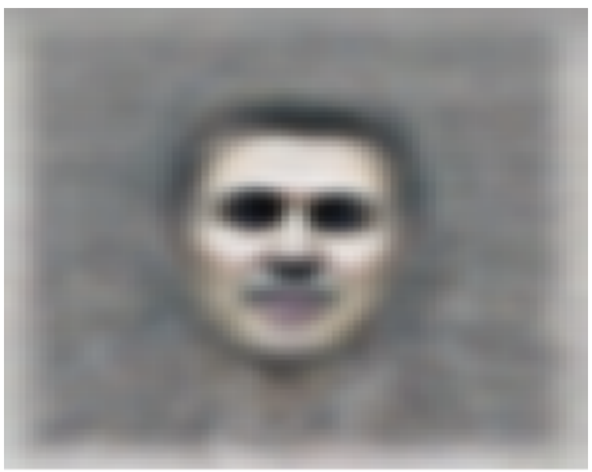


Images from: Lee, H. et al. *Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks*. Communications of the ACM, vol. 54, no. 10, pp. 95-103, 2011

Applications: Google Brain

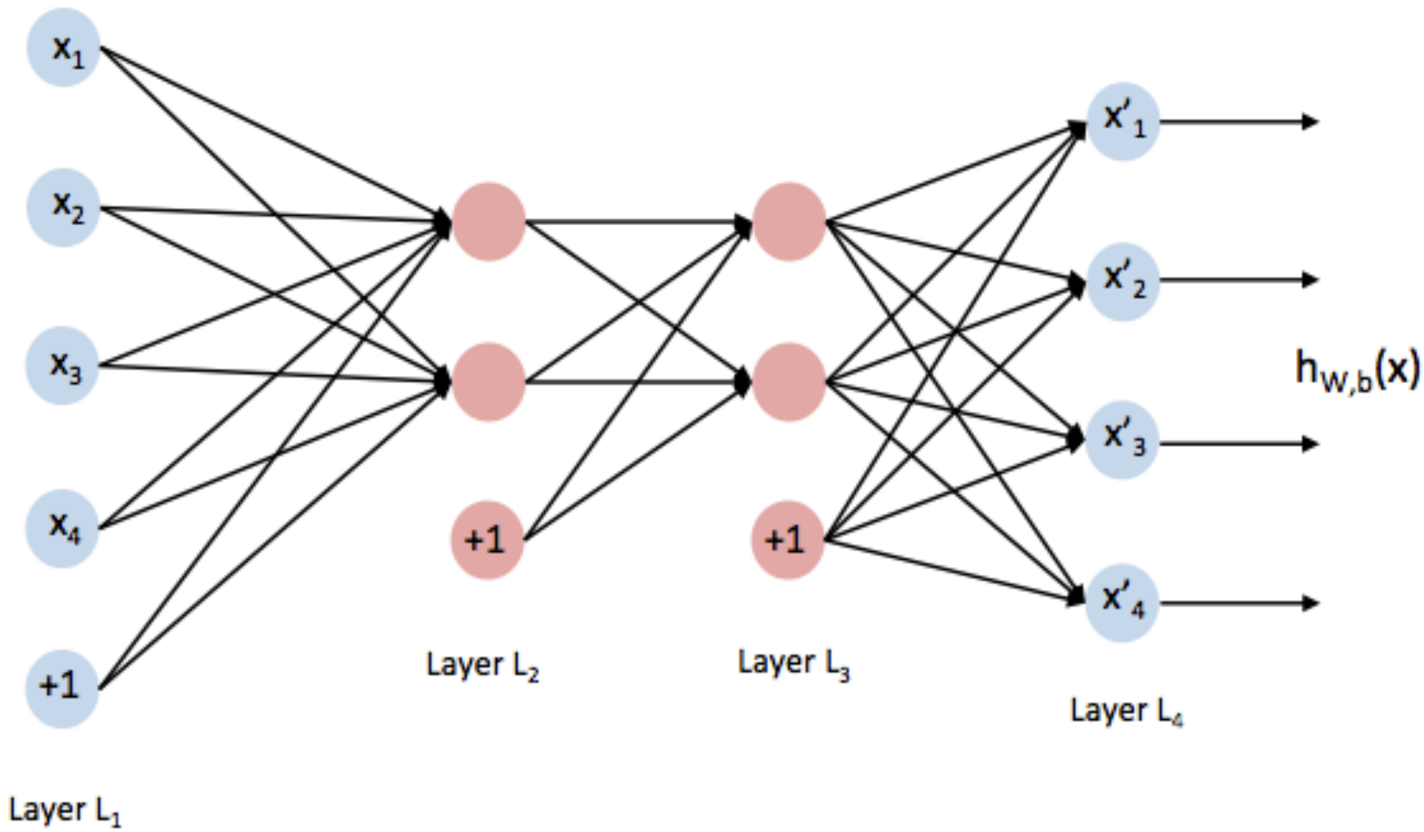
Major differences from Conv Net:

-Unsupervised training on 10 Million images from YouTube
-Untied weights, i.e. locally connected (1+ Billion parameters)



Images from: Le, Q. et. al., Building high-level features using large scale unsupervised learning. ICML, 2012.

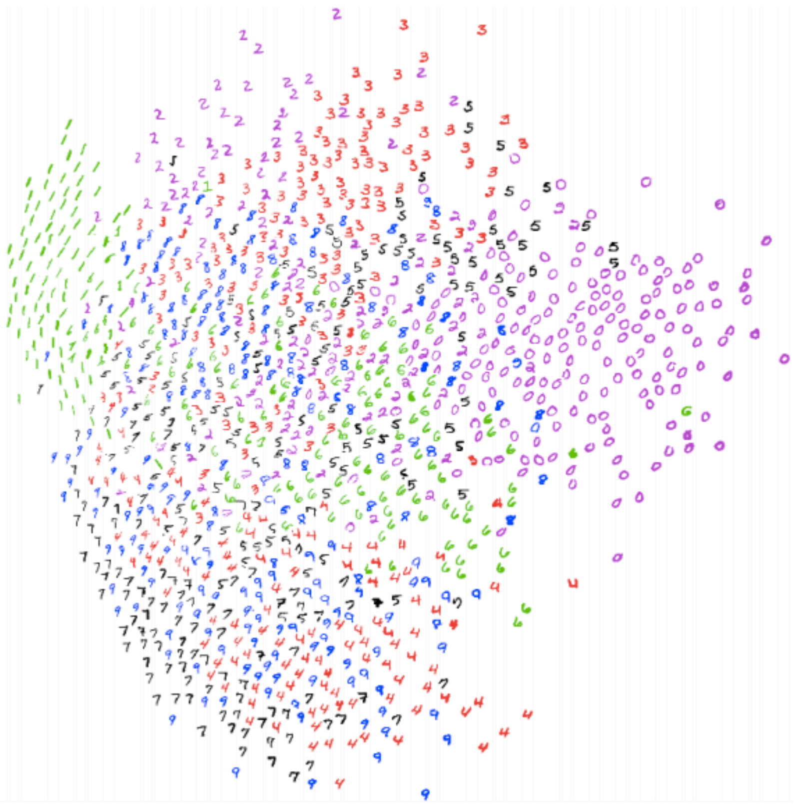
Model Type: Auto-encoder



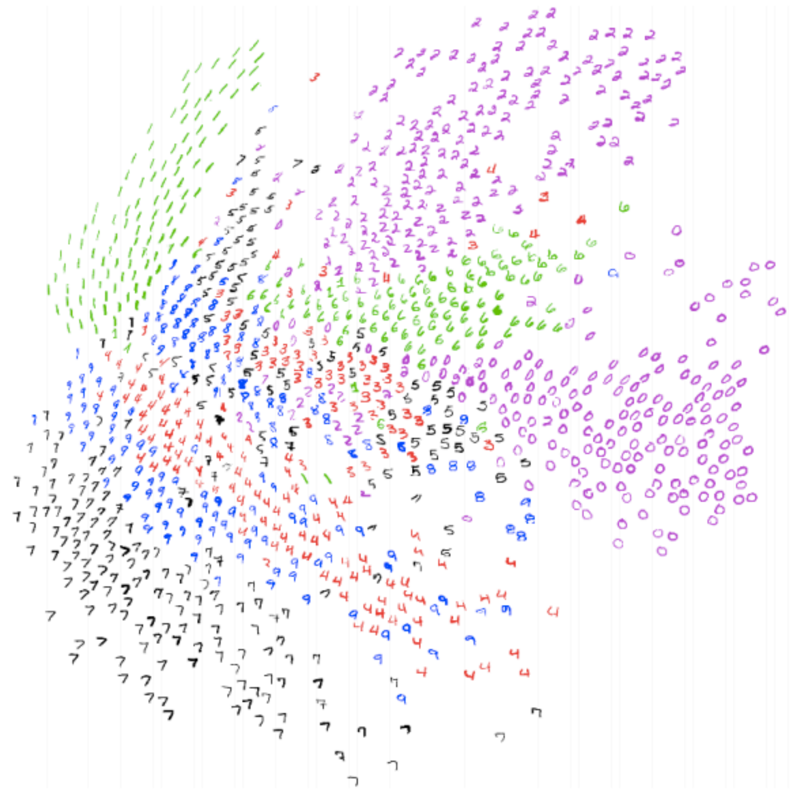
Applications: Auto-encoder

Dimensionality reduction

PCA : First two principle components



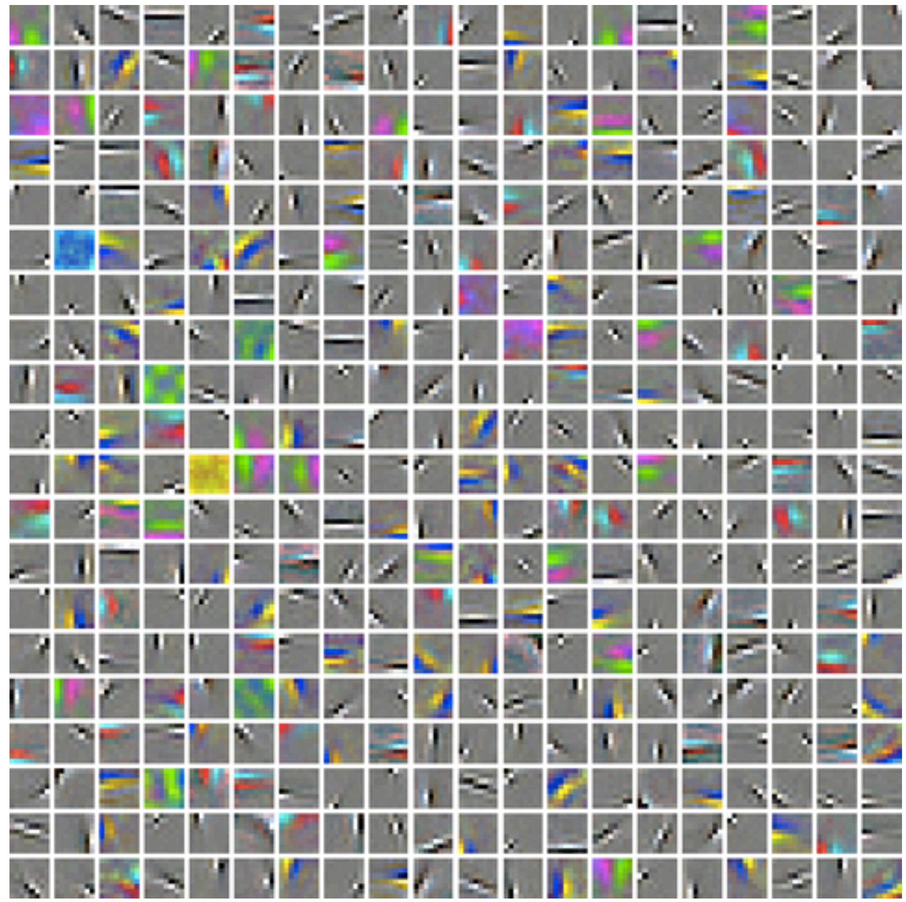
AE : 784-1000-500-250-2



Images from: Hinton, G. E. and Salakhutdinov, R. R. (2006) *Reducing the dimensionality of data with neural networks*. Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.

Applications: Auto-encoder

Learn features with
sparse auto-encoder

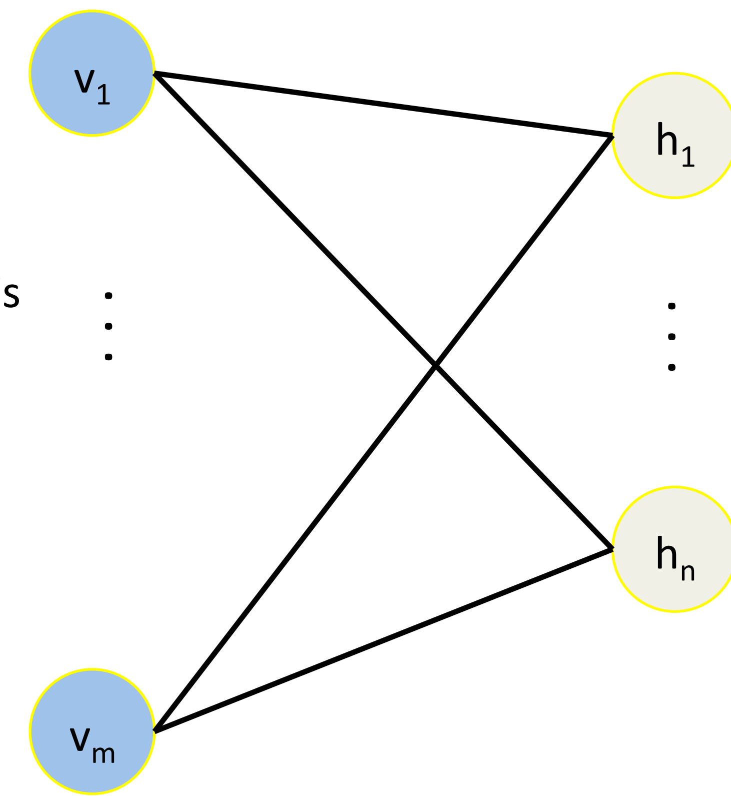


Model: Restricted Boltzmann Machine (RBM)

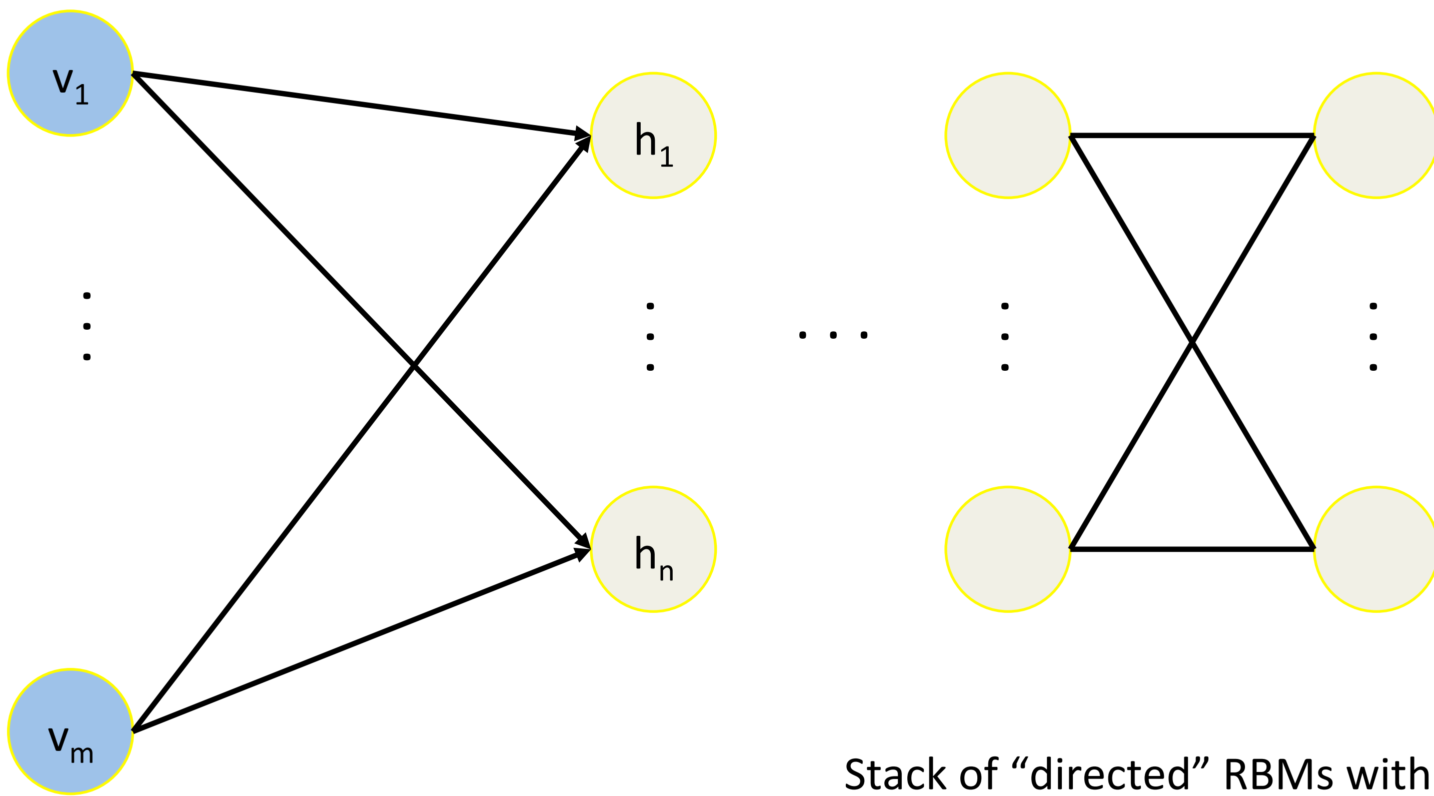
Bipartite Markov Random Field

Edges are undirected.

We'll learn about MRFs later!



Model: Deep Boltzmann Machine (DBN)

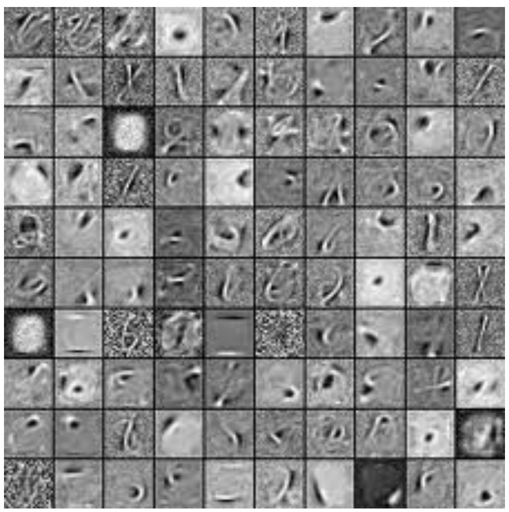


Stack of "directed" RBMs with RBM on top.

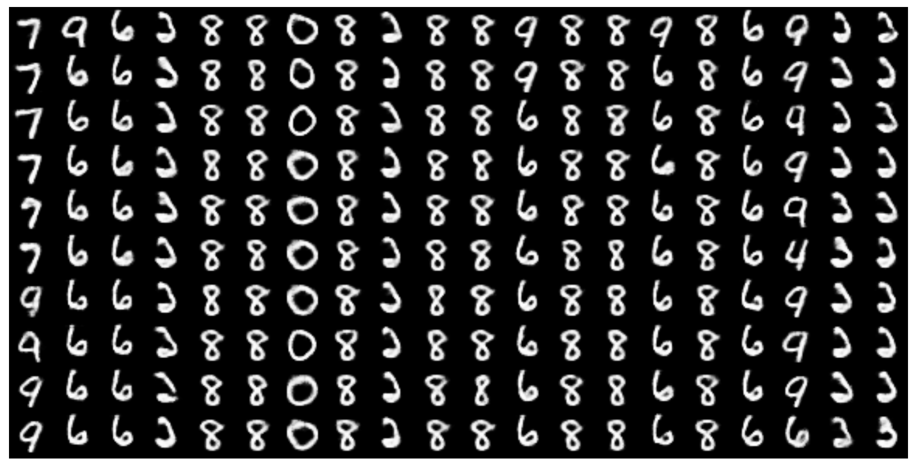
Applications: RBM / DBN

Most common: Pre-training for a DNN

Learning Features

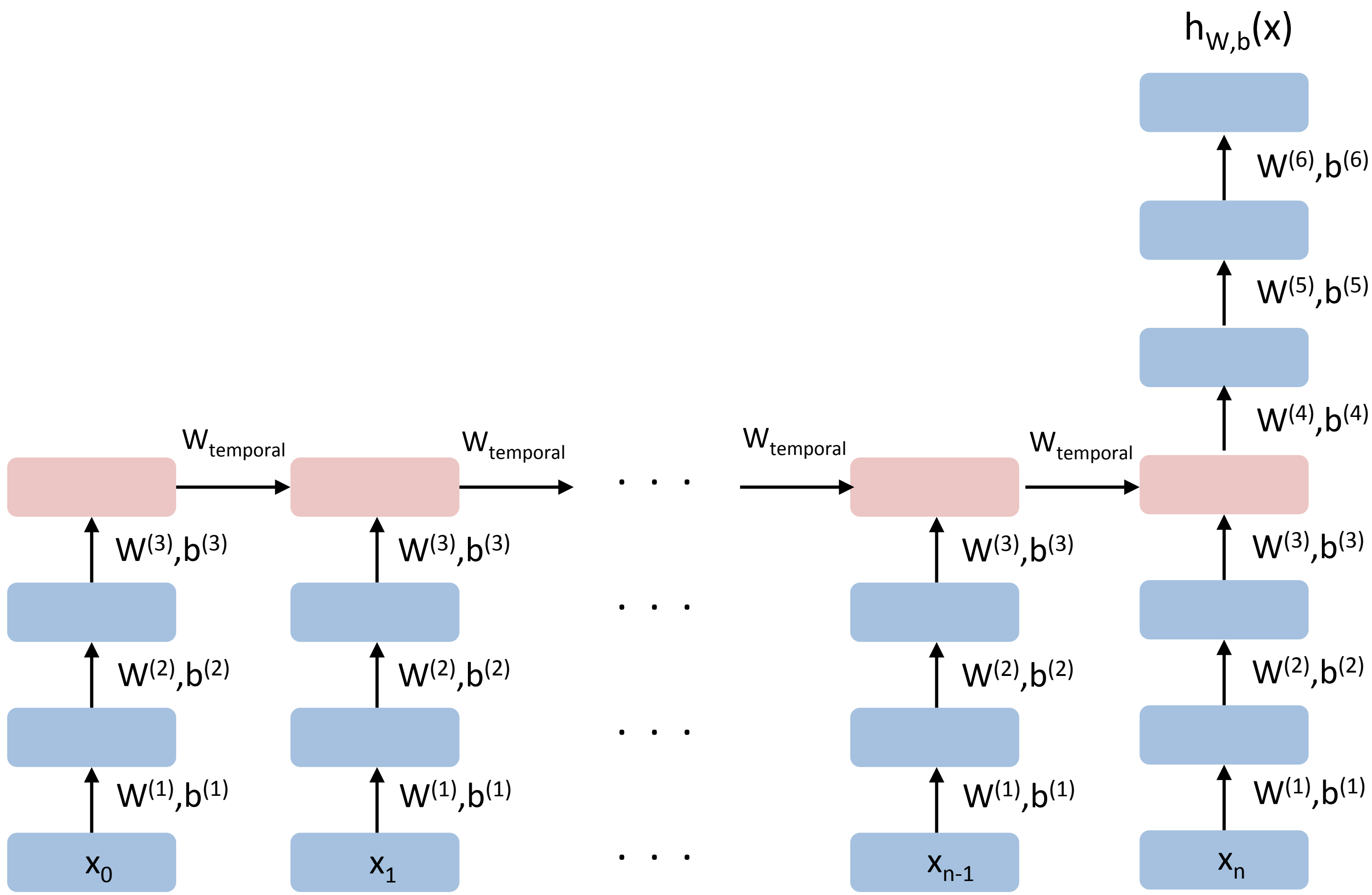


Generating Samples using MCMC
(We'll learn about that later!)



Images: deeplearning.net

Model: Recurrent NN



Applications: Recurrent NN

Any Sequence-based problem:

1. Speech recognition
2. Handwriting recognition
   (state-of-the-art)
3. Stock-market prediction
4. Vision

A. Graves. *Generating Sequences With Recurrent Neural Networks*
http://www.cs.toronto.edu/~graves/handwriting.html

Where to from here?

Online Tutorials:

Stanford Deep Learning Tutorial –
http://ufldl.stanford.edu/tutorial/index.php/UFLDL_Tutorial

http://deeplearning.net/

Reading:

Chris Bishop - *Neural Networks for Pattern Recognition*

Raul Rojas – *Neural Networks* (online)

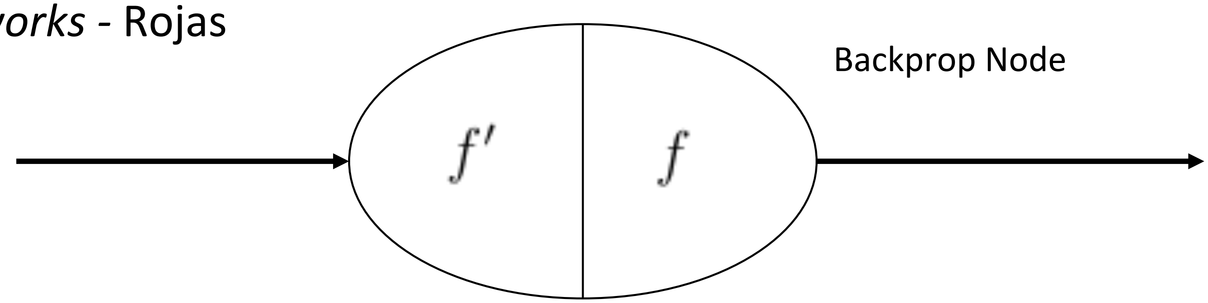
Much much more online…

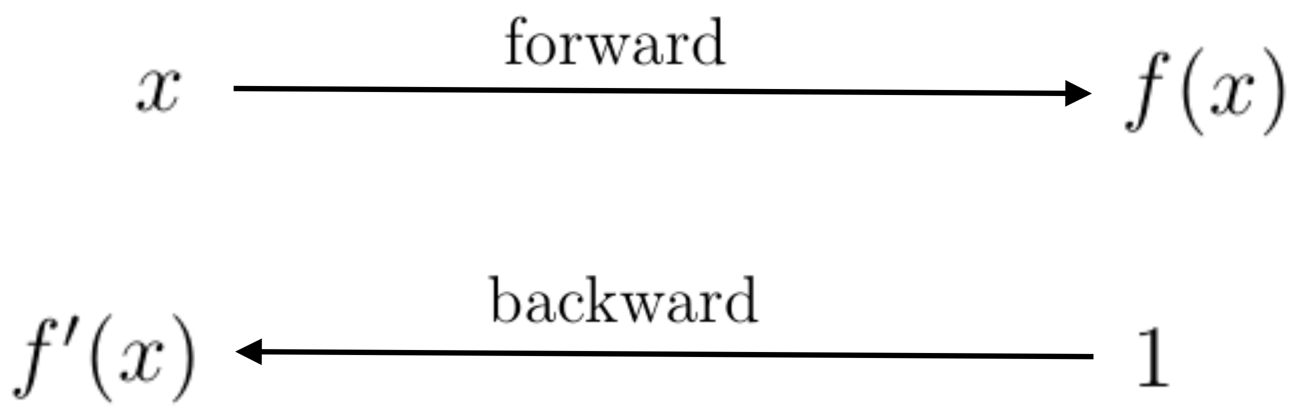
Appendix

Backpropagation

Presentation following
Neural Networks - Rojas

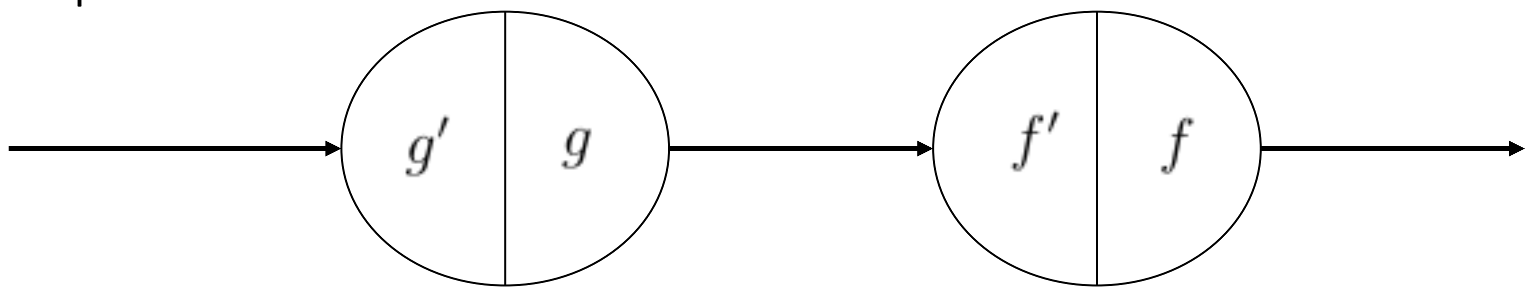
Backprop Node





Backpropagation

Composition

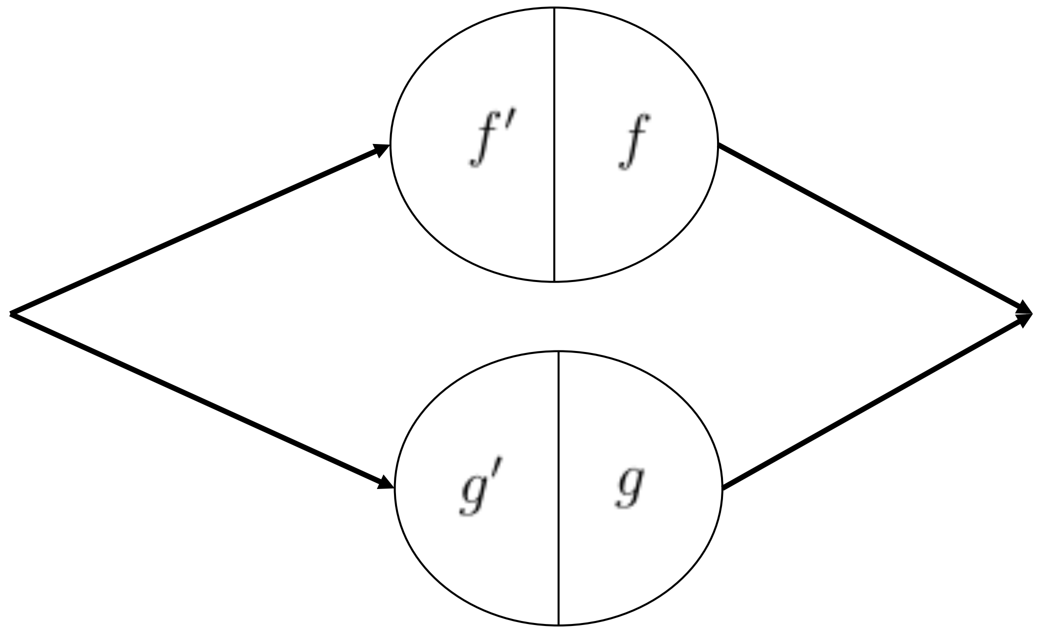


$$x \xrightarrow{\text{forward}} f(g(x))$$

$$f'(g(x))g'(x) \xleftarrow{\text{backward}} 1$$

Backpropagation

Addition

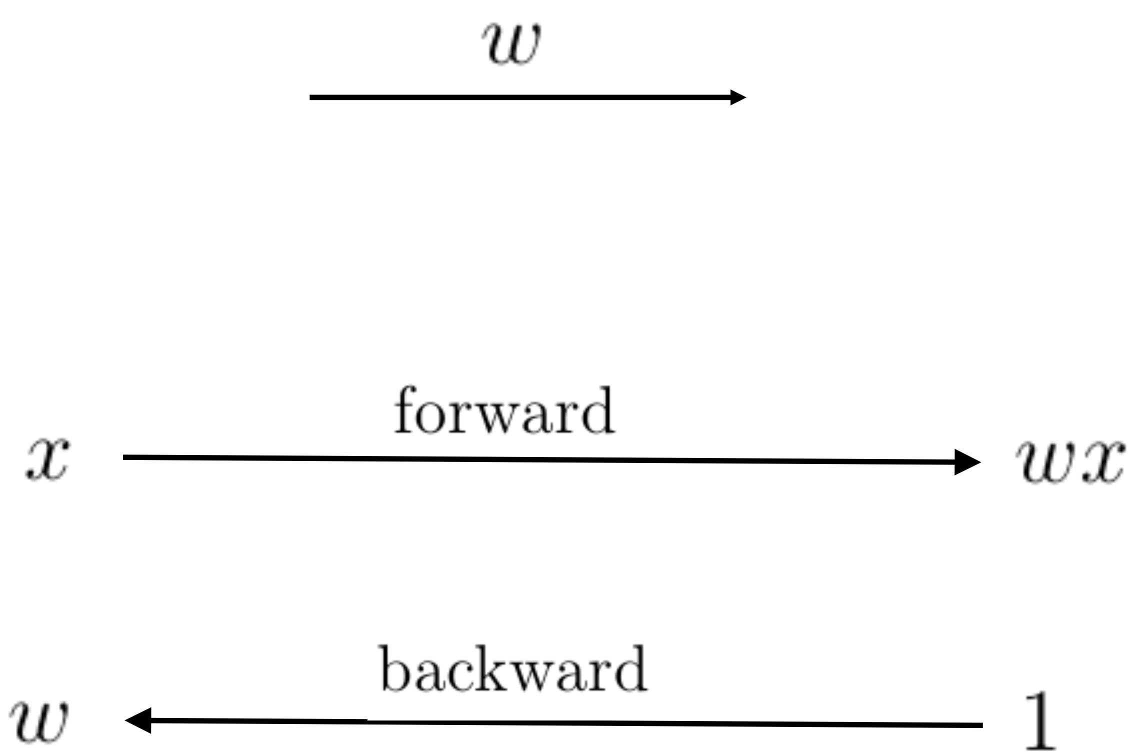


$$x \xrightarrow{\text{forward}} f(x) + g(x)$$

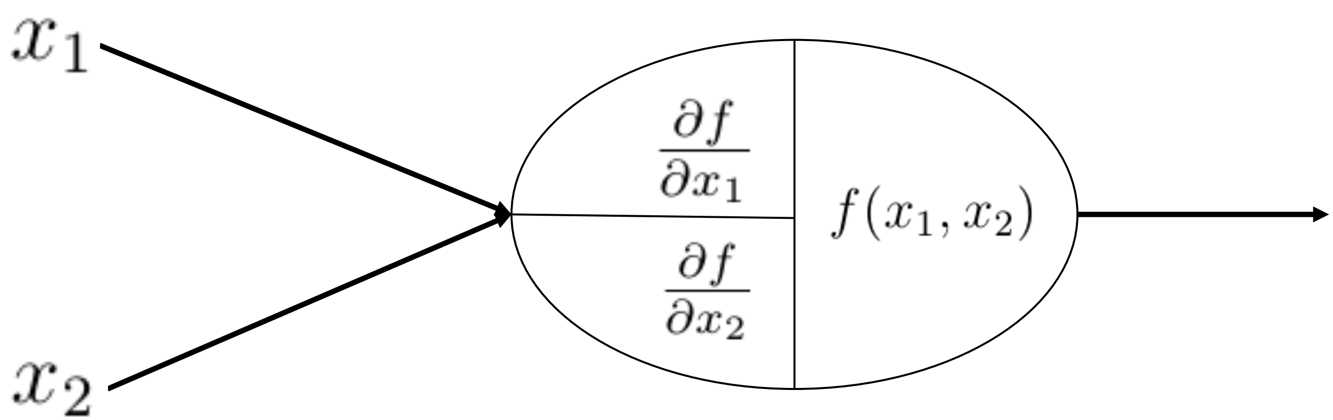
$$f'(x) + g'(x) \xleftarrow{\text{backward}} 1$$

Backpropagation

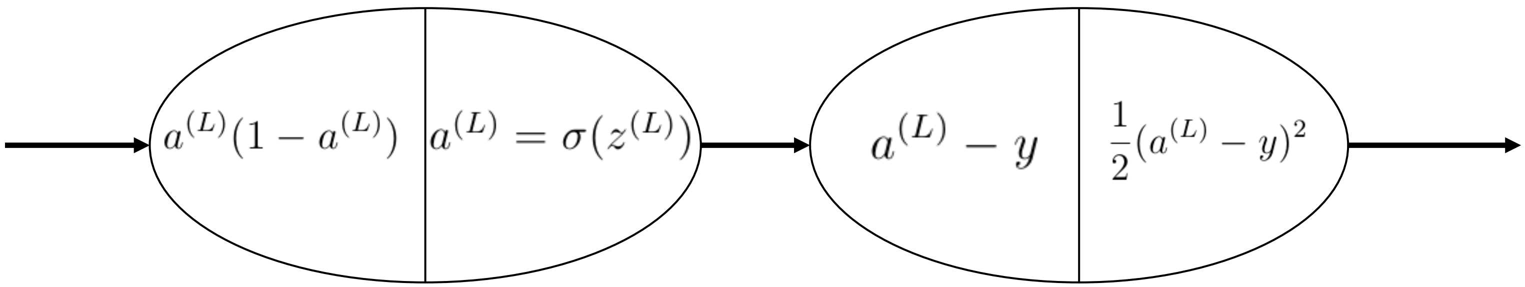
Scaling



Backpropagation



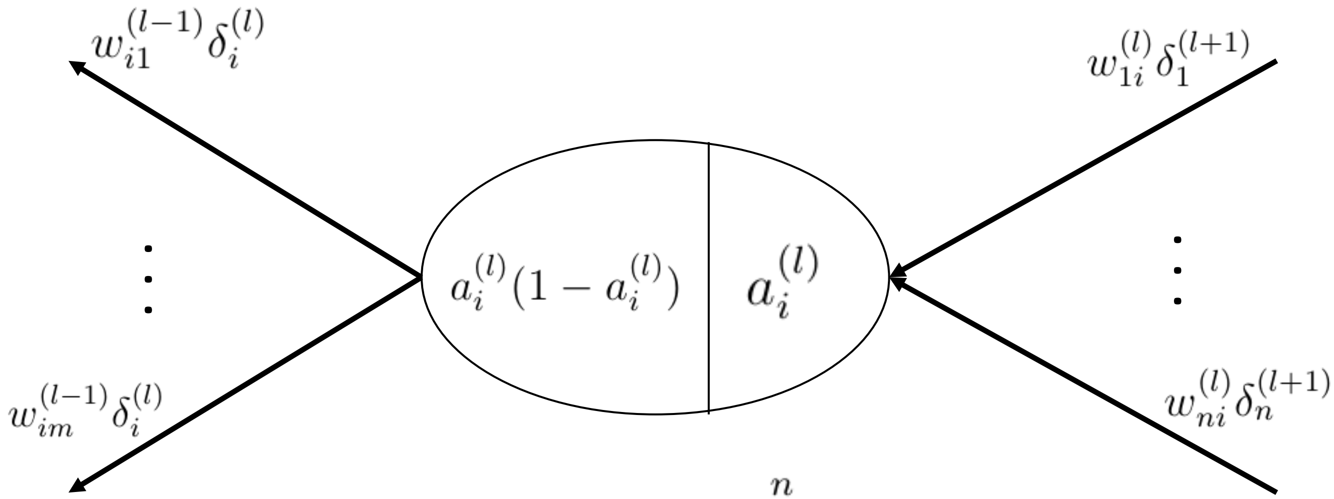
Backpropagation



backward

$$\delta^{(L)} = a^{(L)}(1 - a^{(L)})(a^{(L)} - y) \longleftarrow a^{(L)} - y \longleftarrow 1$$

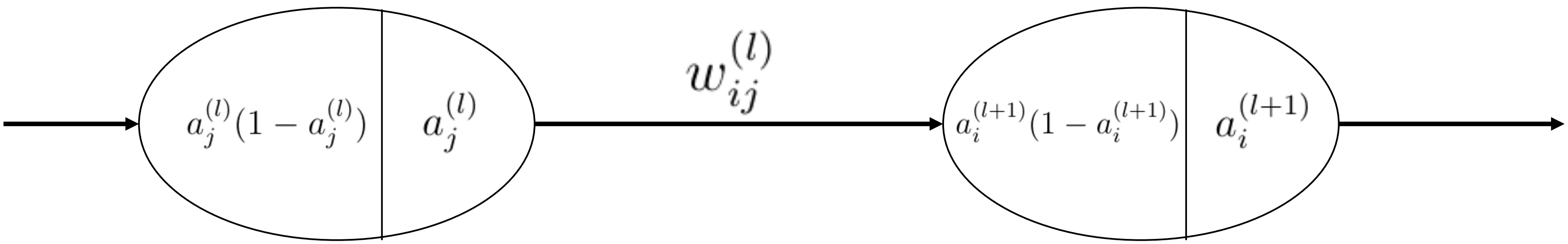
Backpropagation



$$\delta_i^{(l)} = a_i^{(l)}(1-a_i^{(l)})\sum_{j=1}^{n} w_{ji}^{(l)}\delta_j^{(l+1)}$$
$$= a_i^{(l)}(1-a_i^{(l)})(w_i^{(l)})^T\delta^{(l+1)}$$

$$\delta^{(l)} = (W^{(l)})^T\delta^{(l+1)} \circ a^{(l)}(1-a^{(l)})$$

Backpropagation



$$\frac{\partial \mathrm{Loss}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathrm{Loss}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial w_{ij}^{(l)}}$$

$$= \frac{\partial \mathrm{Loss}}{\partial z_i^{(l+1)}} \frac{\partial (W^{(l)} a^{(l)} + b^{(l)})_i}{\partial w_{ij}^{(l)}}$$

$$= \delta_i^{(l+1)} a_j^{(l)}$$

$$\nabla_{W^{(l)}} \mathrm{Loss} = \delta^{(l+1)} \left(a^{(l)}\right)^T$$