Neural Networks

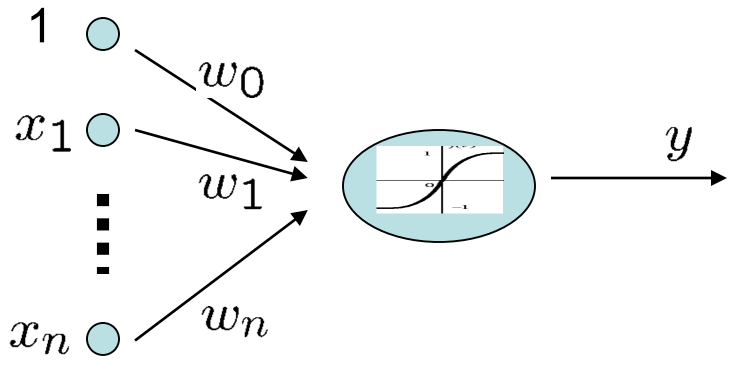
CS534

Key concepts:
Neuron and activation functions
Multilayer Perceptron (MLP) neural networks
Universal function approximator
Back-propogation training
Basics of neural network training
A brief intro to CNN

Motivations

- Analogy to biological systems, which are the best examples of robust learning systems
- Consider human brain:
  - Neuron “switching time” ~ 10^{-3} S
  - Scene recognition can be done in 0.1 S
  - There is only time for about a hundred serial steps for performing such tasks
- We need to exploit massive parallelism!

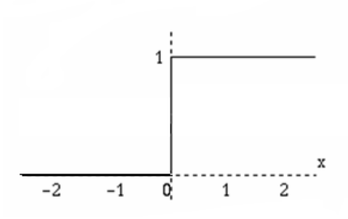
Neural Network Neurons



- Receives n inputs (plus a bias term)
- Multiplies each input by its weight
- Applies activation function to the sum of results
- Outputs result

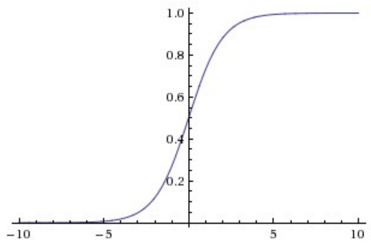
Commonly Used Activation Functions

- **Step function:** $f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$



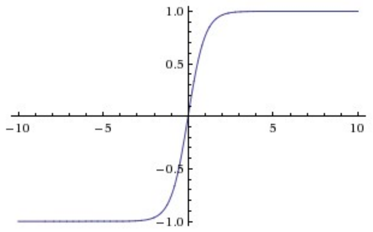
- **Sigmoid function:**

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



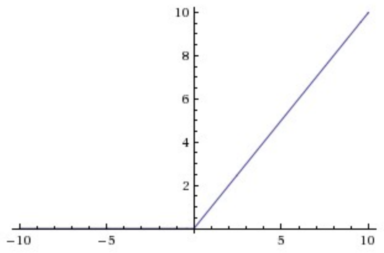
- **Tanh function:**

$$\tanh(x) = 2\sigma(2x) - 1$$

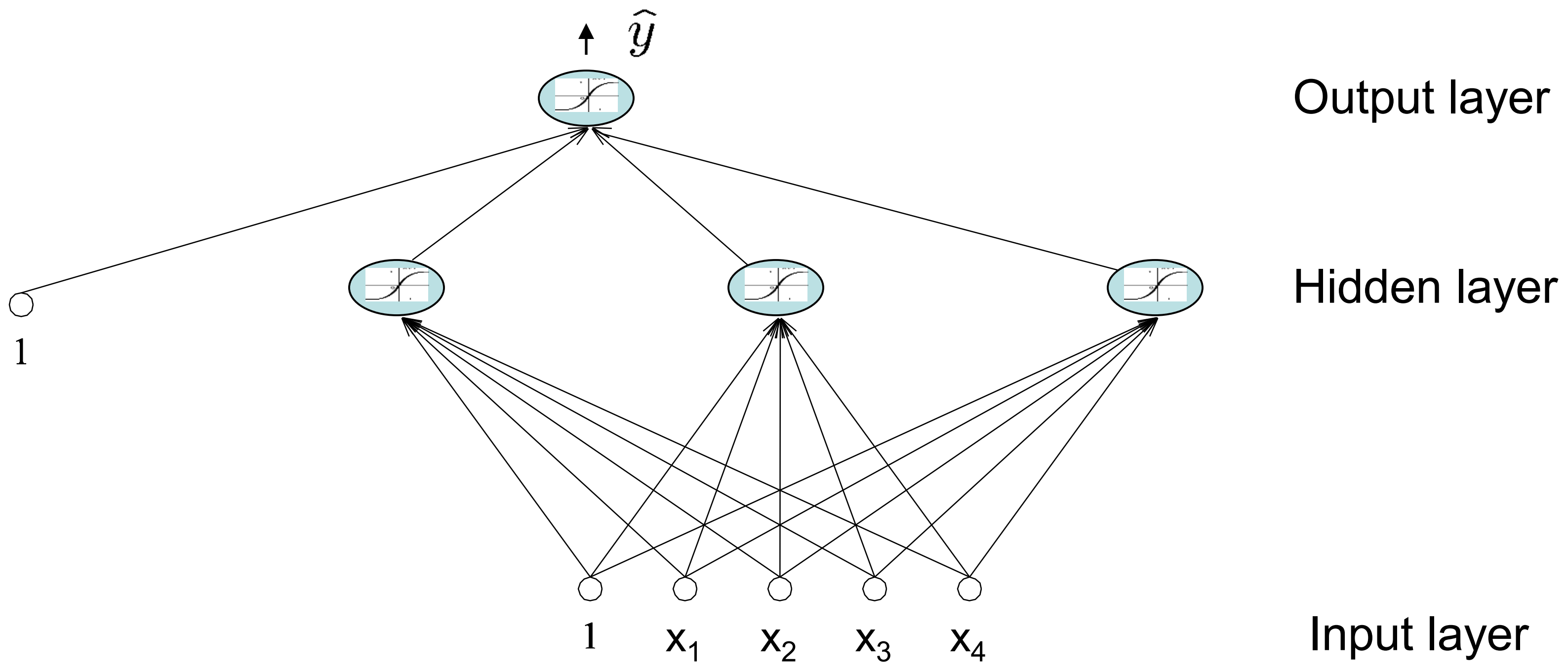


- **Rectified Linear Unit (ReLu)**:

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



Basic Multilayer Neural Network

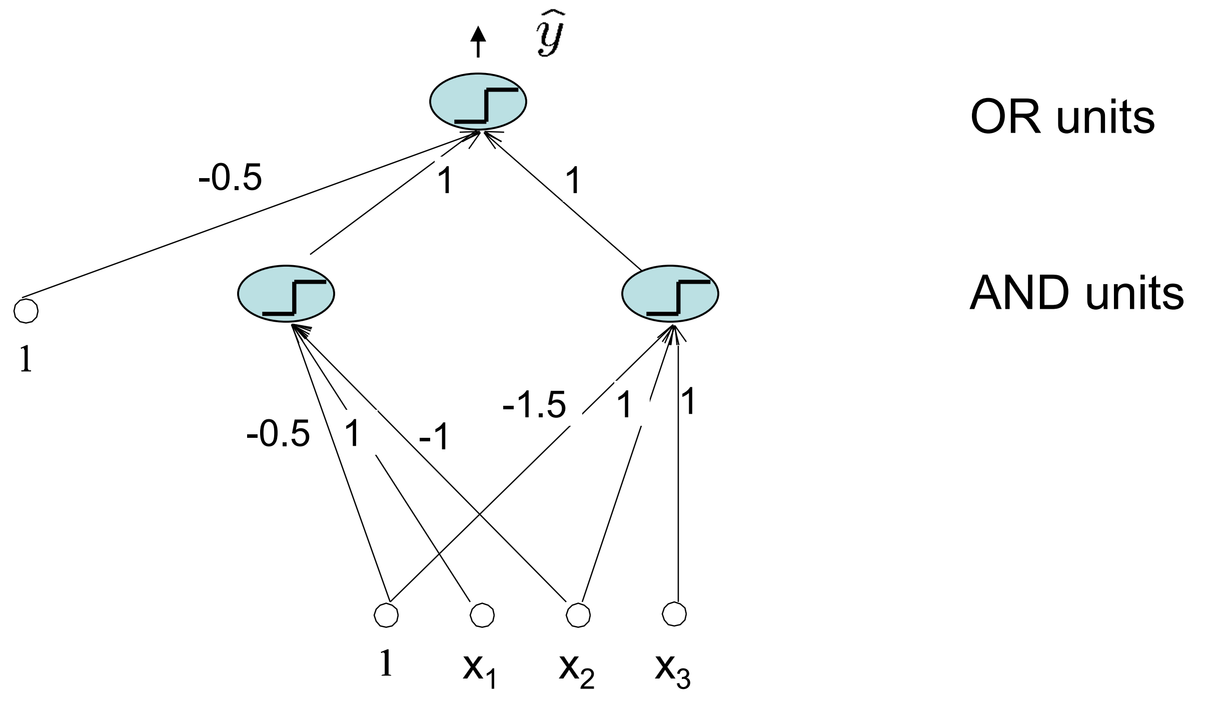


- Each layer receives its inputs from the previous layer and forwards its outputs to the next – feed forward structure
- Output layer: sigmoid activation function for classification, and linear activation function for regression
- Referred to as a two-layer network (2 layer of weights)

Representational Power

- <u>Any Boolean Formula</u>
  - Consider a formula in disjunctive normal form:

$$(x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3)$$



Representational Power (cont.)

- <u>Continuous functions</u>
  - Any continuous functions can be approximated arbitrarily closely by a sum of (possibly infinite) basis functions
  - Suppose we implement the hidden units to represent the basis functions, and give the output node a linear activation function. Any bounded continuous function can be approximated to arbitrary accuracy with enough hidden units.

Training: Backpropagation

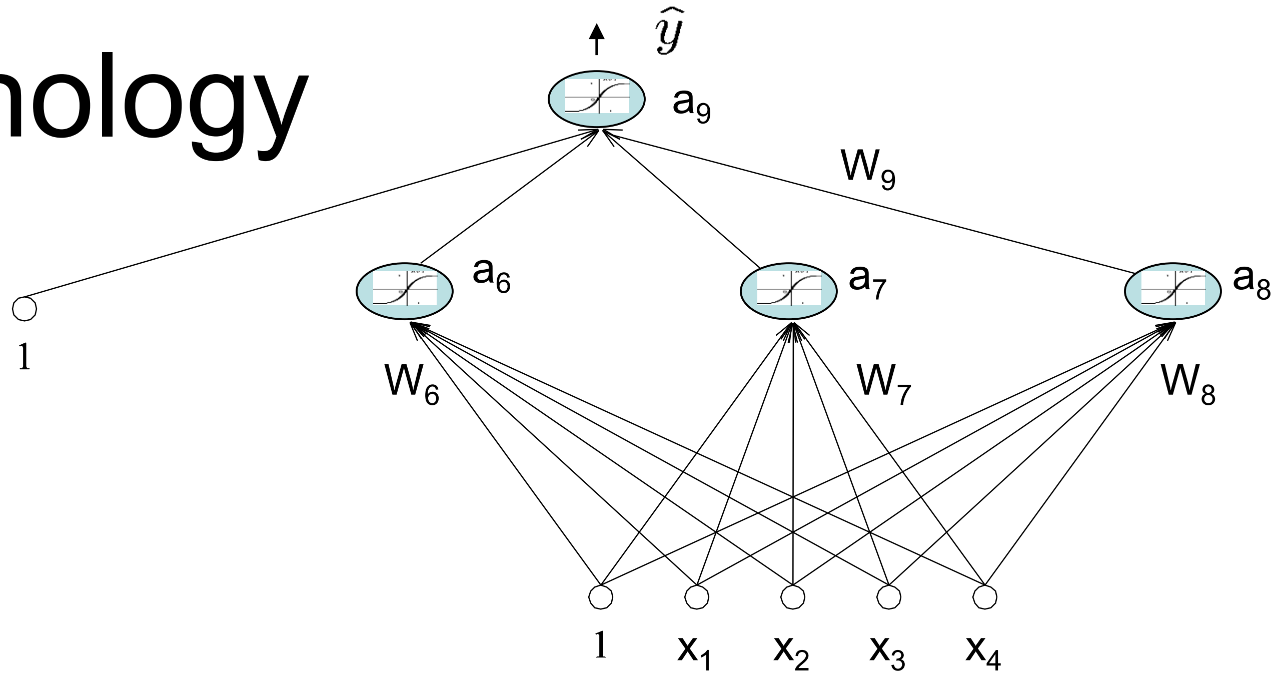
- Training of the neural net aims to find weights that minimize some loss function
- For example, for regression problem, denoting the network output for input x as $\hat{y}(x)$

$$L(\mathrm{w}) = \sum_{i=1}^{n} (\hat{y}(x_i, w) - y_i)^2$$

- For classification problems the loss can be different, e.g., negative log-likelihood
- Use gradient descent to iteratively improve the weights
- This is done from layer to layer, applying the chain rule to compute the gradient for each layer

Chain rule for gradient: $\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx}$

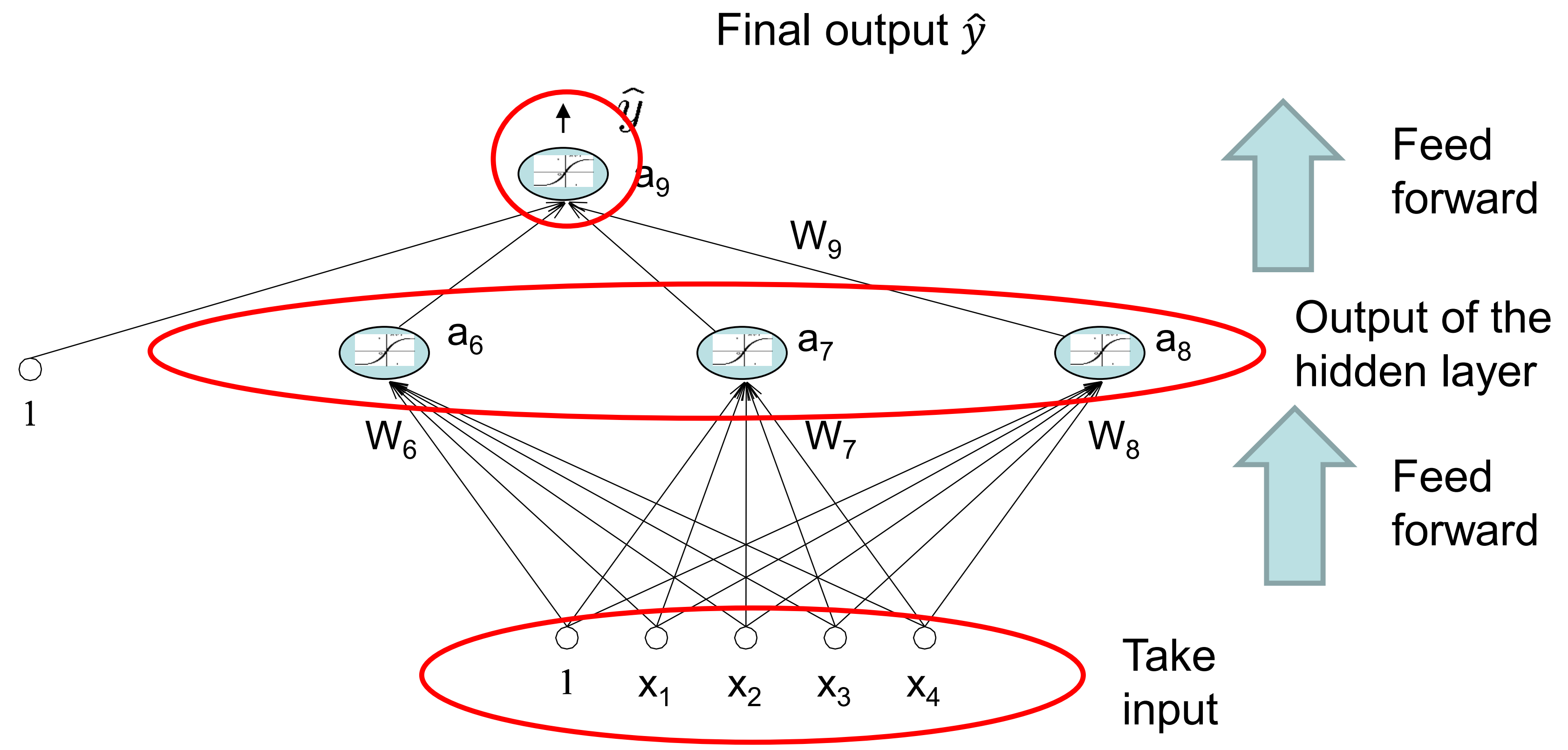
Terminology



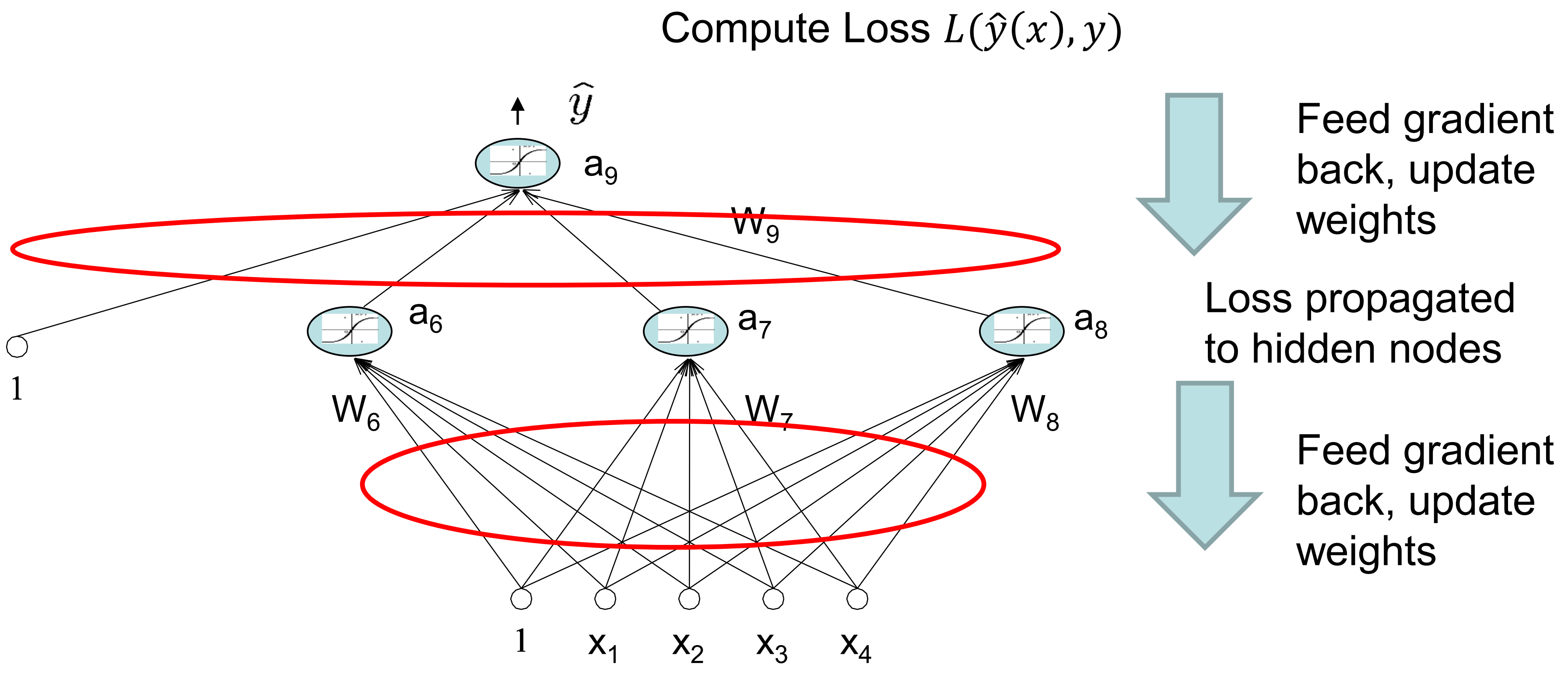
- $\mathbf{X} = [1, x_1, x_2, x_3, x_4]^T$ – the input vector with the bias term
- $A = [1, a_6, a_7, a_8]^T$ – the output of the hidden layer with the bias term
- $\boldsymbol{W}_i$ represents the weight vector leading to node i
- $w_{i,j}$ represents the weight connecting from the j-th node to the i-th node
  - $w_{9,6}$ is the weight connecting from a_6 to a_9
- We will use σ to represent the activation function, so

$$\hat{y} = \sigma(W_9 \cdot [1, a_6, a_7, a_8]^T) = \sigma(W_9 \cdot [1, \sigma(W_6 \cdot X), \sigma(W_7 \cdot X), \sigma(W_8 \cdot X)]^T)$$

Training: the forward pass



Training: the backward pass


Compute Loss $L(\hat{y}(x), y)$
$\hat{y}$
a_9
W_9
a_6
a_7
a_8
1
W_6
W_7
W_8
1
x_1
x_2
x_3
x_4
Feed gradient back, update weights
Loss propagated to hidden nodes
Feed gradient back, update weights

The calculation of the gradient will depend on the loss function and the activation function – but often it is not complicated

E.g., if we use the same loss as logistic regression, we have the same update rule for updating the outer most weight layer

Example: Sum Squared Error

- We adjust the weights of the neural network to minimize the Sum squared error (SSE) on training set.

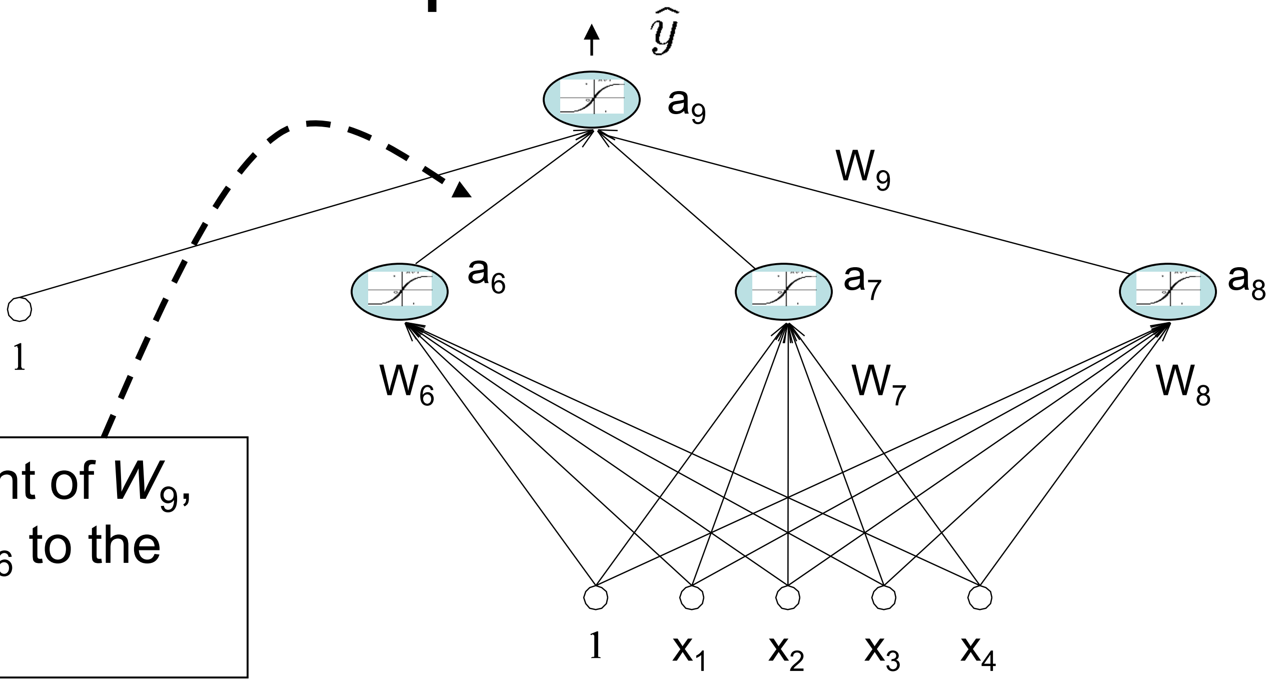
$$J(W) = \frac{1}{2}\sum_{i=1}^{N}(\widehat{y}^i - y^i)^2$$

$$J_i(W) = \frac{1}{2}(\widehat{y}^i - y^i)^2$$

- **<u>Useful fact</u>:** the derivative of the sigmoid activation function is

$$\frac{d\sigma(x)}{dx} = \sigma(x)\big(1 - \sigma(x)\big)$$

Gradient Descent: Output Unit

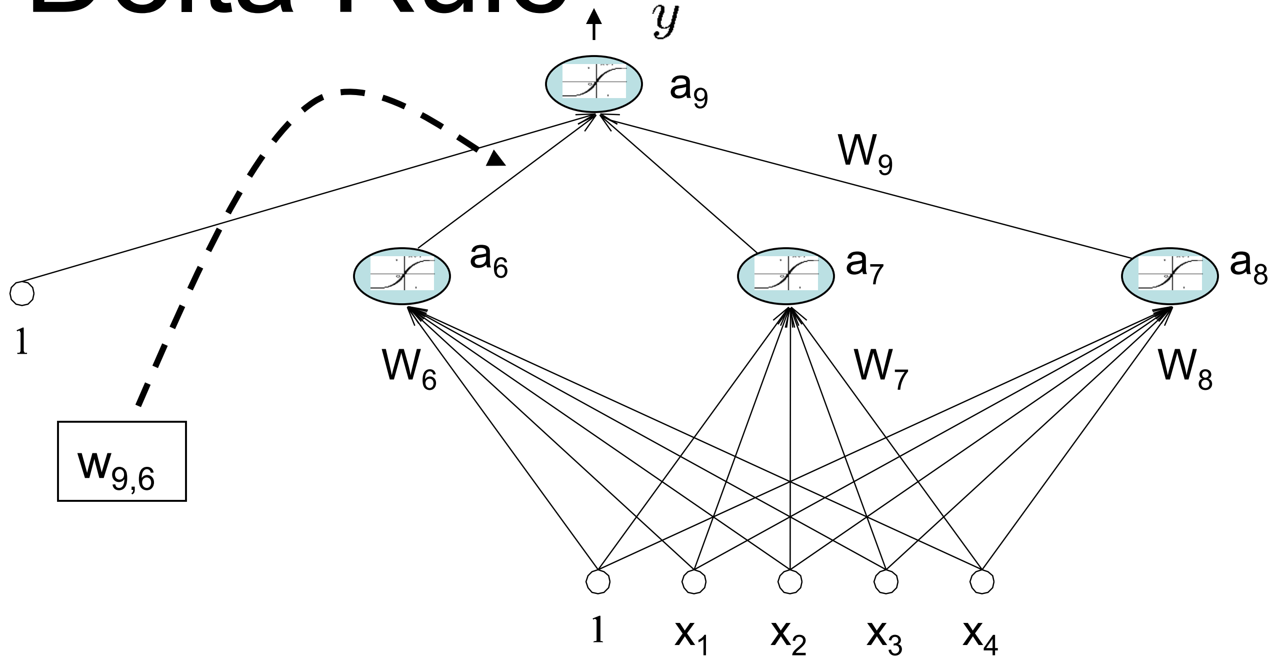


$w_{9,6}$ is a component of W_9, connecting from a_6 to the output node.

$$\frac{d\sigma(x)}{dx} = \sigma(x)\big(1-\sigma(x)\big)$$

$$\begin{aligned}
\frac{\partial J_i(W)}{\partial w_{9,6}} &= \frac{\partial}{\partial w_{9,6}} \frac{1}{2}(\hat{y}^i - y^i)^2 \\
&= \frac{1}{2} \cdot 2 \cdot (\hat{y}^i - y^i) \cdot \frac{\partial}{\partial w_{9,6}} \sigma(W_9 \cdot A^i) \\
&= (\hat{y}^i - y^i) \cdot \sigma(W_9 \cdot A^i)(1 - \sigma(W_9 \cdot A^i)) \cdot \frac{\partial}{\partial w_{9,6}} W_9 \cdot A^i \\
&= (\hat{y}^i - y^i)\hat{y}^i(1 - \hat{y}^i) \cdot a_6^i
\end{aligned}$$

The Delta Rule



- Define $\delta_9^i = (\widehat{y}^i - y^i)\widehat{y}^i(1-\widehat{y}^i)$

  then
$$\begin{aligned}\frac{\partial J_i(W)}{\partial w_{9,6}} &= (\widehat{y}^i - y^i)\widehat{y}^i(1-\widehat{y}^i)\cdot a_6^i \\ &= \delta_9^i \cdot a_6^i\end{aligned}$$

Extending to the whole vector W_9: $\frac{\partial J_i}{\partial W_9} = \delta_9^i \; A^i$

Di-secting the delta rule

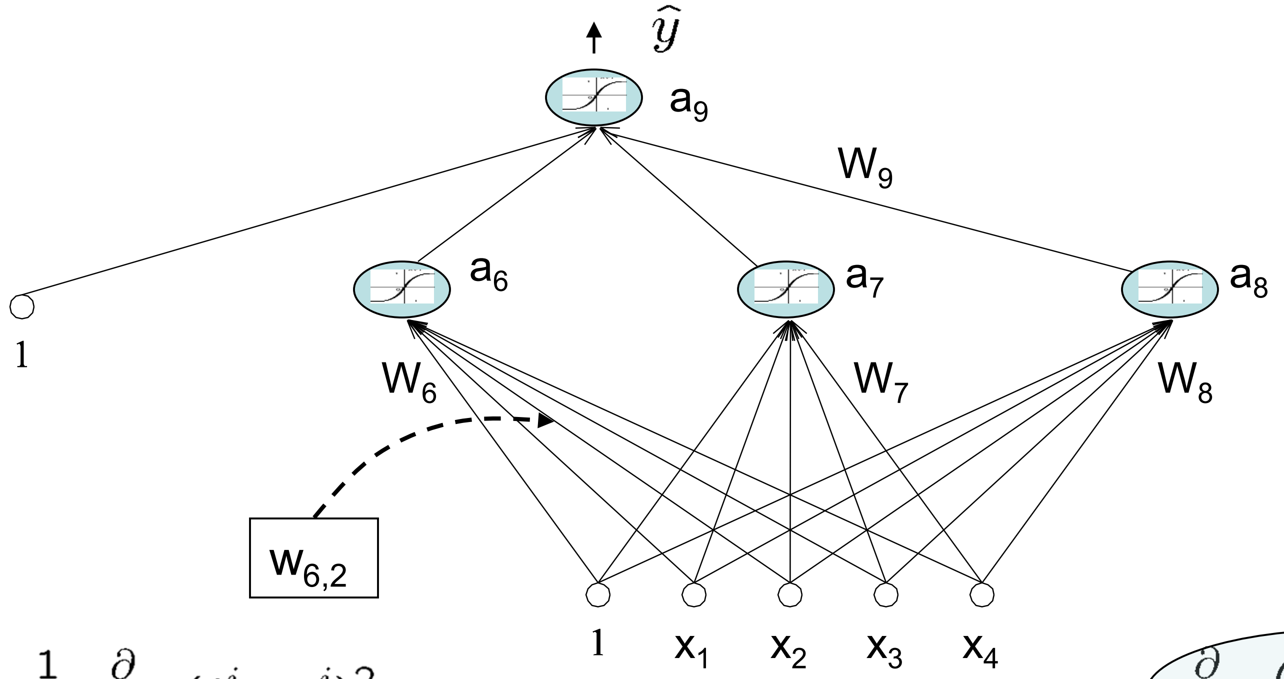
- Consider a general loss function defined on $\hat{y}^i$:

$$L(\hat{y}^i)$$

Where $\hat{y}^i = f\left(\mathbf{W}_9^T \mathrm{A}^i\right)$, f is the activation function

$$\frac{dL}{d\mathrm{W}_{\mathbf{9}}} = \frac{dL(\hat{y}^i)}{d\hat{y}^i} \times \frac{d\hat{y}^i}{d\mathrm{W}_{\mathbf{9}}} = \underbrace{(L' \cdot f')}_{\delta_9^i}\mathrm{A}^i$$

Derivation: Hidden Units



$$\frac{\partial J_i(W)}{\partial w_{6,2}} = \frac{1}{2}\frac{\partial}{\partial w_{6,2}}(\widehat{y}^i - y^i)^2$$

$$= (\widehat{y}^i - y^i)\cdot\sigma(W_9\cdot A^i)(1-\sigma(W_9\cdot A^i))\cdot\frac{\partial}{\partial w_{6,2}}(W_9\cdot A^i)$$

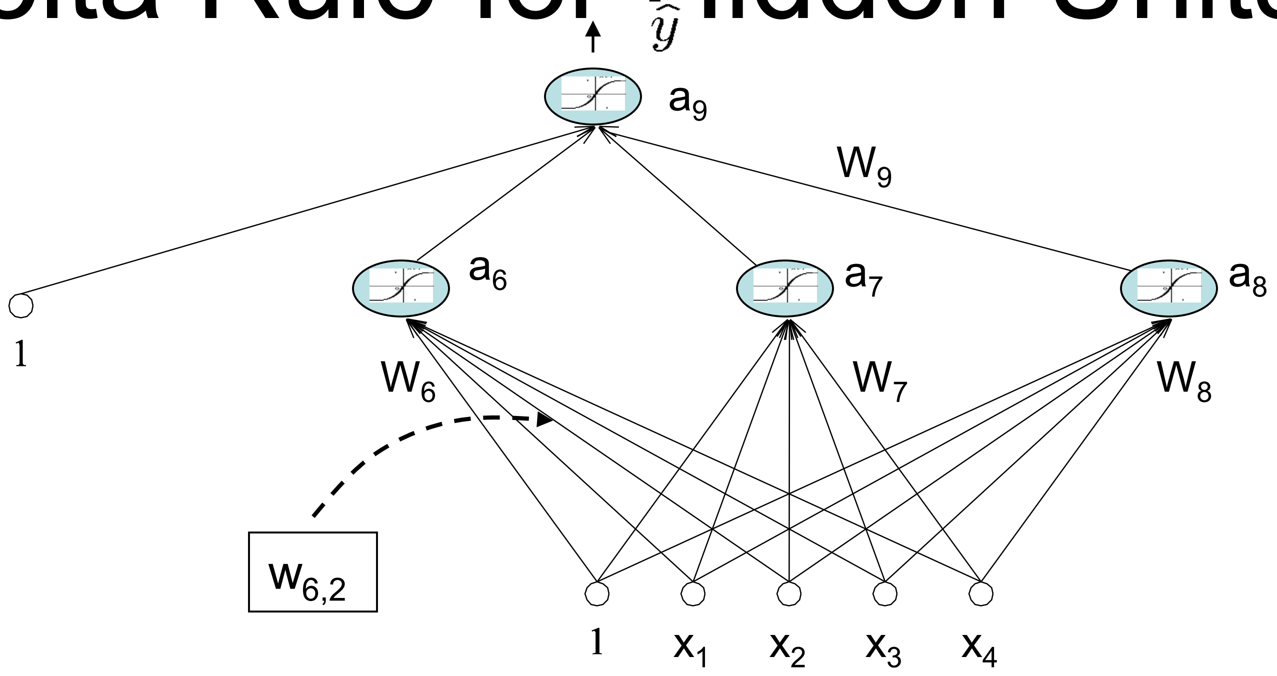
$$= \frac{\partial}{\partial w_{6,2}}(w_{9,6}\cdot a_6^i)$$

$$= \delta_9^i\cdot w_{9,6}\cdot\frac{\partial}{\partial w_{6,2}}\sigma(W_6\cdot X^i)$$

$$= \delta_9^i\cdot w_{9,6}\cdot\sigma(W_6\cdot X^i)(1-\sigma(W_6\cdot X^i))\cdot\frac{\partial}{\partial w_{6,2}}(W_6\cdot X^i)$$

$$= \delta_9^i\cdot w_{9,6}\cdot a_6(1-a_6)\cdot x_2^i$$

Delta Rule for Hidden Units



Define $\delta_6^i = \delta_9^i \cdot w_{9,6} \cdot a_6^i(1 - a_6^i)$

and rewrite as

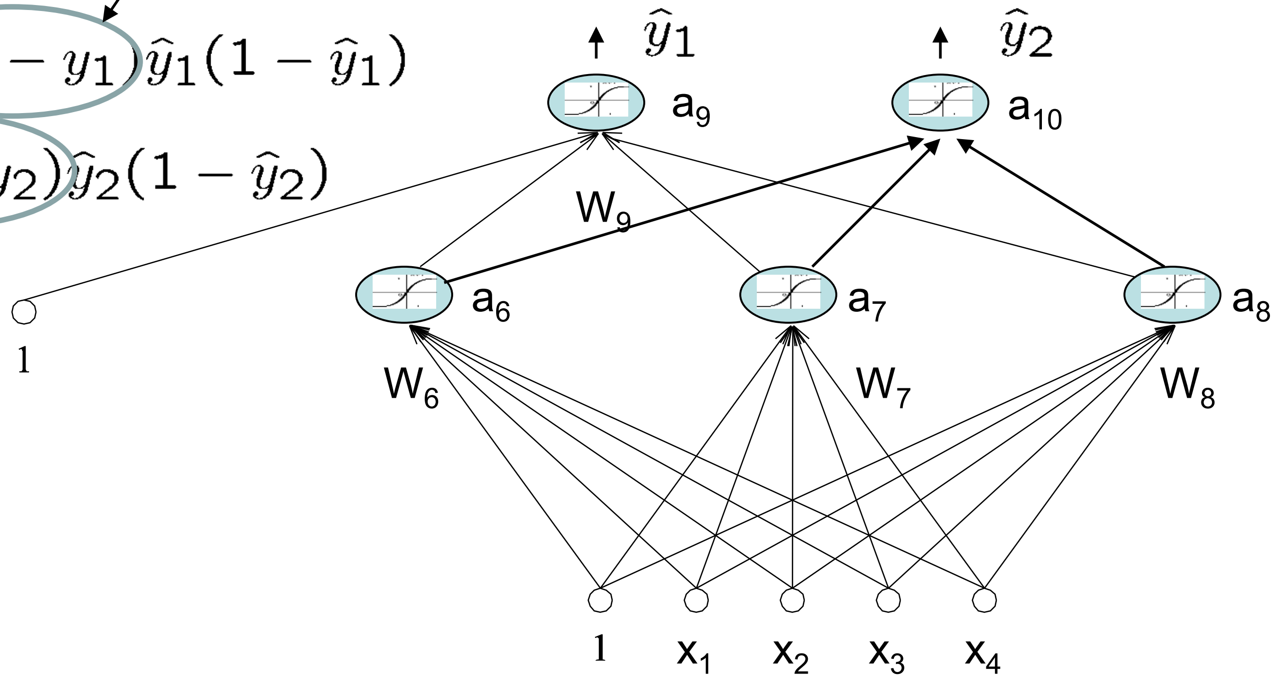
$$\frac{\partial J_i(W)}{\partial w_{6,2}} = \delta_6^i \cdot x_2^i.$$

$$\frac{\partial J_i}{\partial W_6} = \delta_6^i \cdot X, \qquad \frac{\partial J_i}{\partial W_7} = \delta_7^i \cdot X, \qquad \frac{\partial J_i}{\partial W_8} = \delta_8^i \cdot X$$

Networks with Multiple Output Units

$$\delta_9 = \overbrace{(\hat{y}_1 - y_1)}^{\frac{\partial L}{\partial \hat{y}_1}}\hat{y}_1(1 - \hat{y}_1)$$

$$\delta_{10} = \underbrace{(\hat{y}_2 - y_2)}_{\frac{\partial L}{\partial \hat{y}_2}}\hat{y}_2(1 - \hat{y}_2)$$



- We get a separate contribution to the gradient from each output unit.
- Hence, for input-to-hidden weights, we must sum up the contributions:

$$\delta_6 = a_6(1 - a_6)(w_{9,6}\delta_9 + w_{10,6}\delta_{10})$$

Backpropagation Training

- Initialize all the weights with small random values
- Repeat
  - For all training examples, do

    Begin Epoch

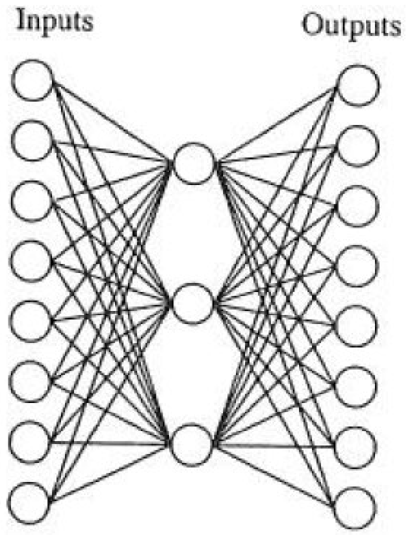
      For each training example do

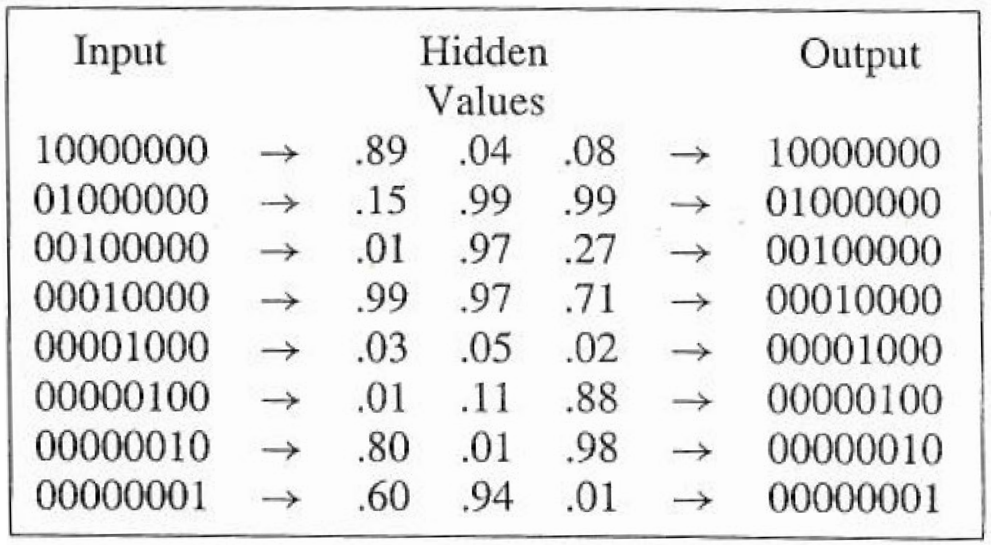
      - Compute the network output
      - Compute loss
      - Backpropagate this loss from layer to layer and adjust weights to decrease this loss using gradient descent

    End Epoch

Hidden layer representation

- Hidden nodes learn to discover useful intermediate representations
  - A intriguing property of multi-layer neural networks

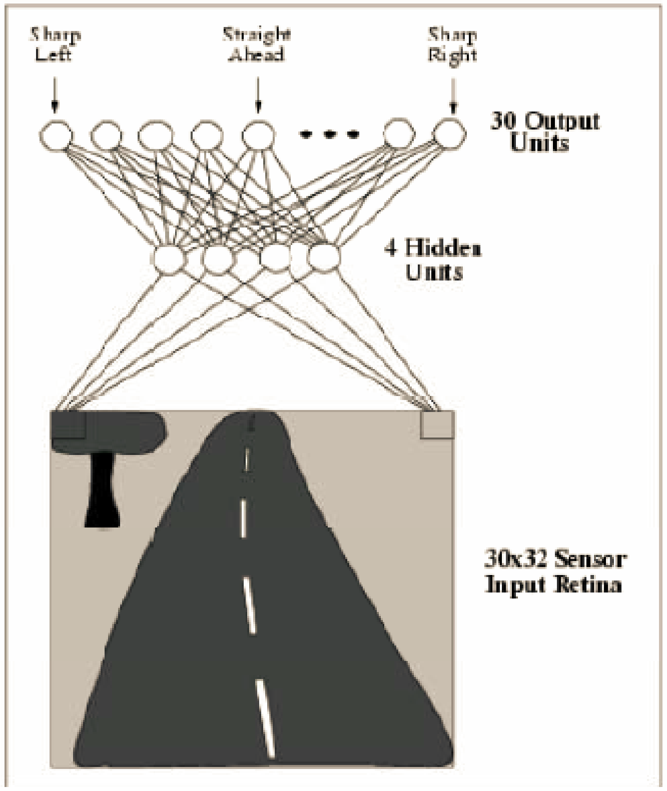


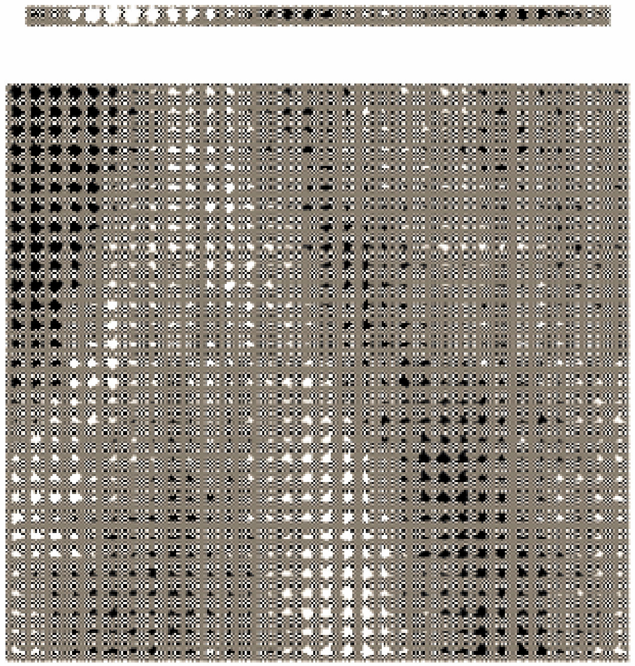


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

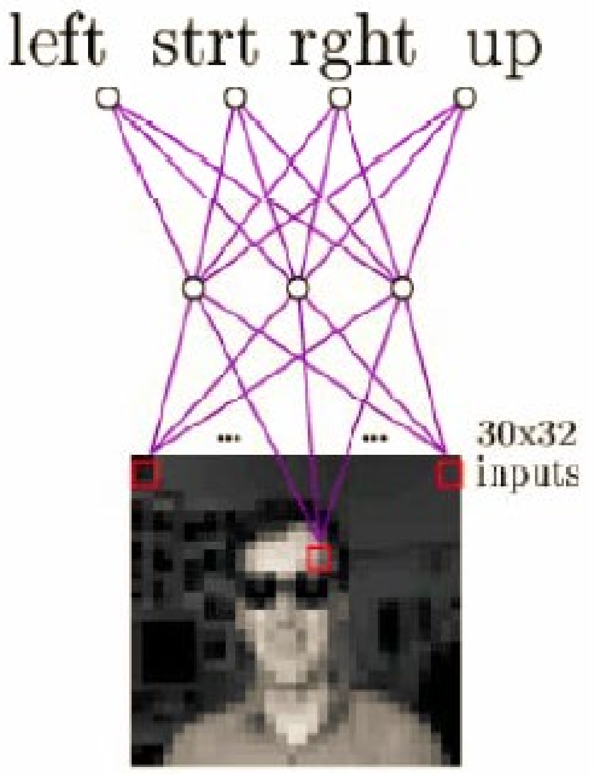
Example

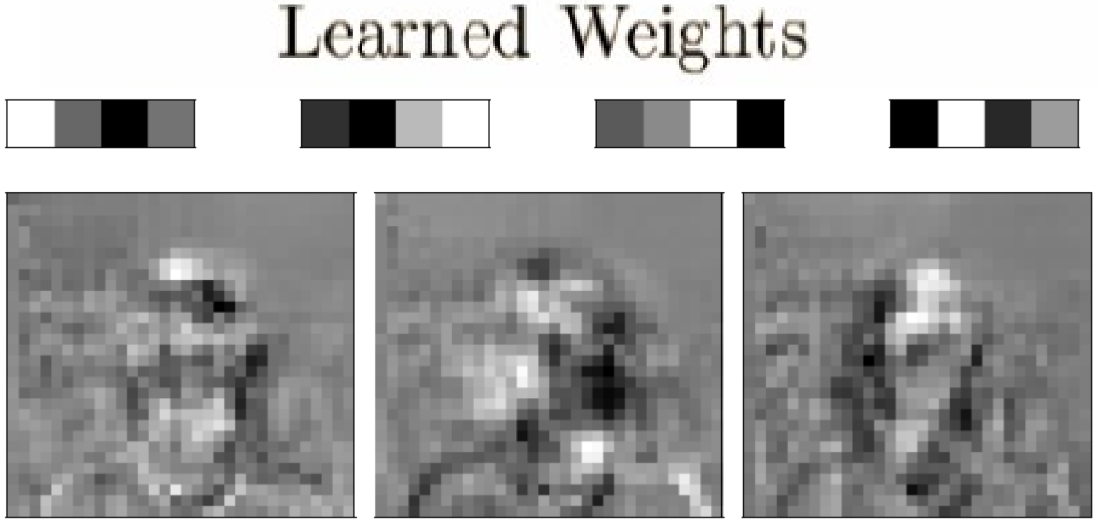
Neural net is one of the most effective methods when the data include complex sensory inputs such as images.





Example from Mitchell's ML book pp. 84





Example from Mitchell’s ML textbook pp. 113

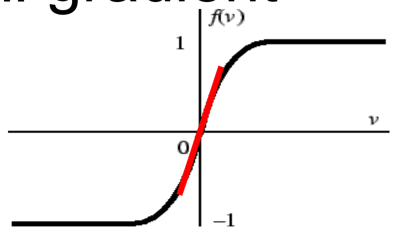
Typical input images

Remarks on Training

- Not guaranteed to convergence, may oscillate or reach a local minima.
- However, in practice many large networks can be adequately trained on large amounts of data for realistic problems, e.g.,
  - Driving a car
  - Recognizing handwritten zip codes
- Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of training
- Termination criteria can be:
  - Fixed number of epochs
  - Threshold on training set error
  - Increased error on a validation set
- To avoid local minima problems, can run several trials starting from different initial random weights and select the best according to the objective

Notes on Proper Initialization

- Start in the “linear” regions
  - keep all weights near zero, so that all sigmoid units are in their linear regions. This makes the whole net the equivalent of one linear threshold unit—a relatively simple function.
  - This will also avoid having very small gradient



- Break symmetry
  - If we start with all the weights equal, what would happen?
  - Ensure that each hidden unit has different input weights so that the hidden units move in different directions.

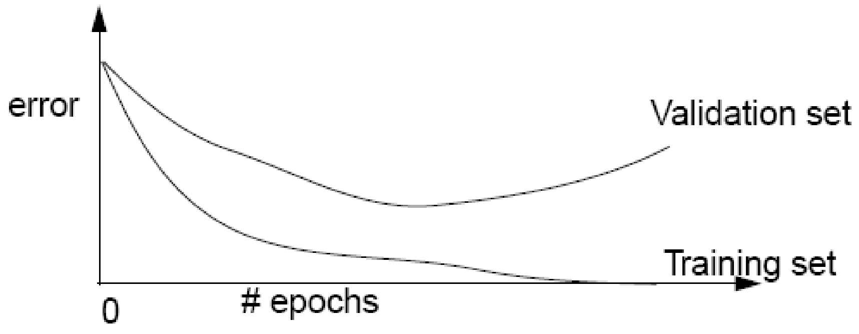
Batch, Online and Online with Momentum

- Batch. Sum up the gradient for a batch of examples and take a combined gradient step
- Online: Take a gradient step for each example
- Momentum: each update linearly combines the current gradient with the previous update direction to ensure smoother convergence

$$\Delta w_{i,j}(n) = \underbrace{\boxed{\eta \delta_j x_{i,j}}}_{\text{Current update}} + \underbrace{\alpha \Delta w_{i,j}(n-1)}_{\text{Previous update}}$$

Overtraining Prevention

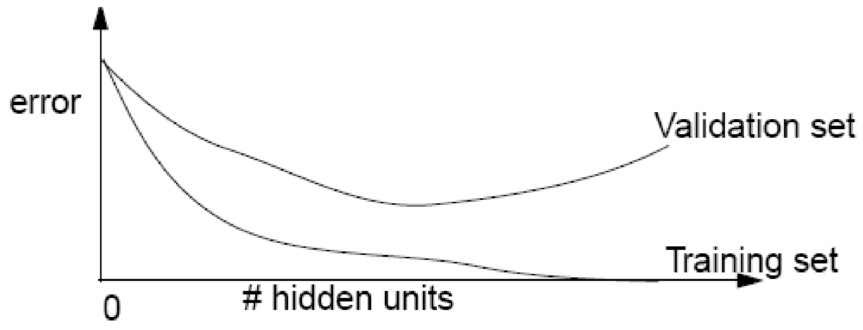
- Running too many epochs may overtrain the network and result in overfitting.



- Keep a validation set and test accuracy on the validation set after every epoch. Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond this.

Over-fitting Prevention

- Too few hidden units underfit the data and fail to learn the concept.
- Too many hidden units over-fit



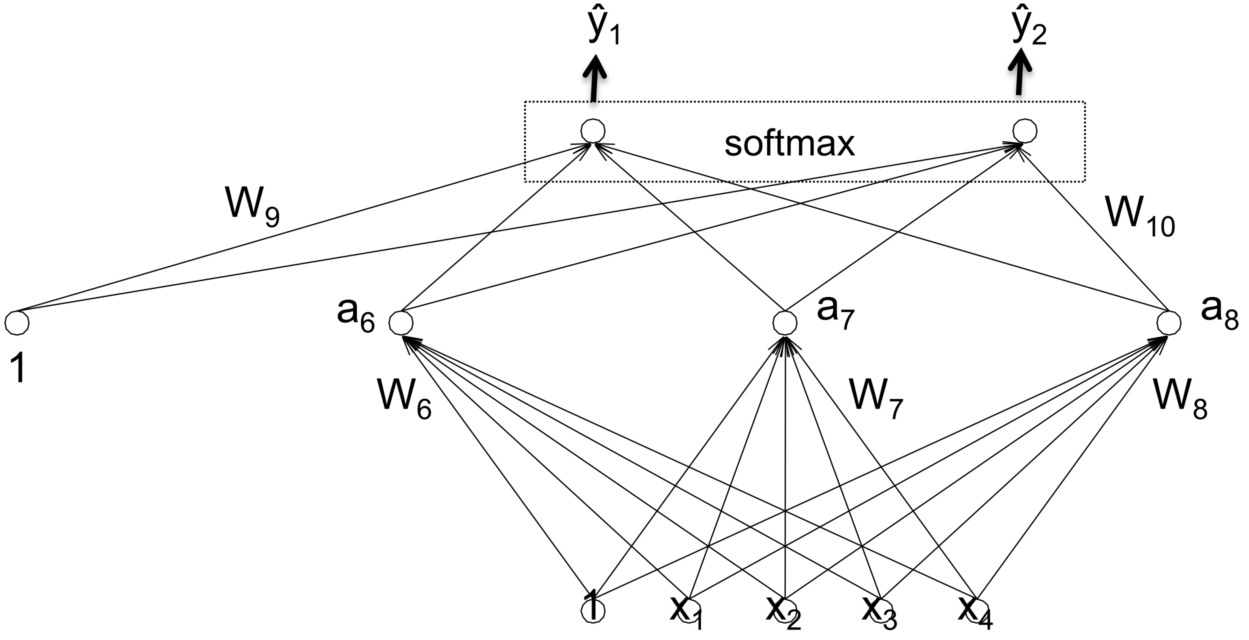
- Cross-validation can be used to decide the right number of hidden units.
- ***Weight decay*** multiplies all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less overfitting
  - Equivalent to including a regularization term on the weights

Input/Output Coding

- Appropriate coding of inputs/outputs can make learning easier and improve generalization.
- Best to encode discrete multi-category features using multiple input units and include one binary unit per value
- Continuous inputs can be handled by a single input unit, but scaling them between 0 and 1
- For classification problems, best to have one output unit per class.
  - Continuous output values then represent certainty in various classes.
  - Assign test instances to the class with the highest output.
- Use target values of 0.9 and 0.1 for binary problems rather than forcing weights to grow large enough to closely approximate 0/1 outputs.
- Continuous outputs (regression) can also be handled by scaling to the range between 0 and 1

Softmax for multi-class classification

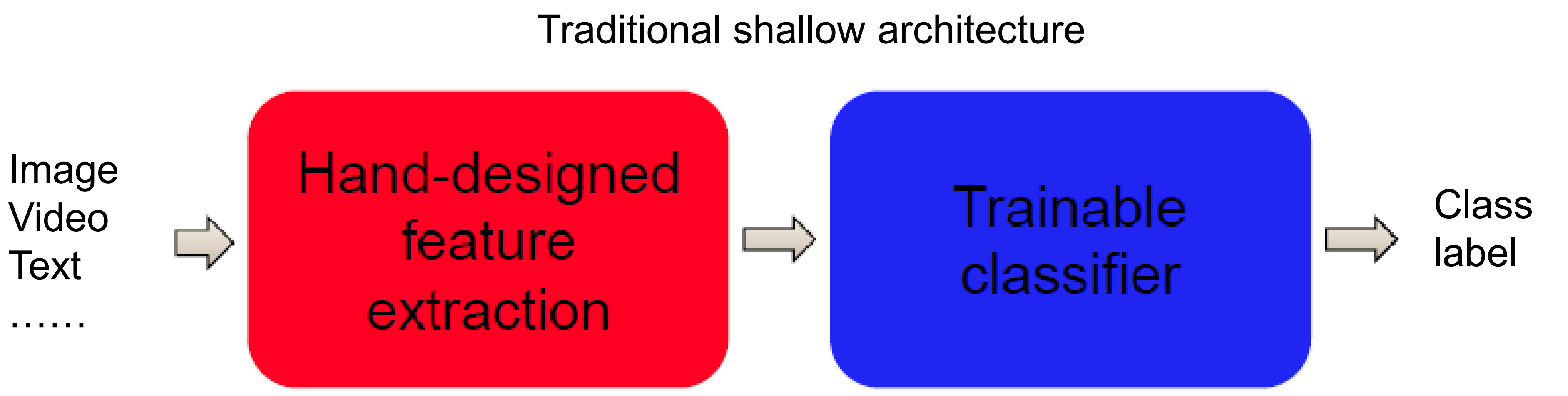
- For K classes, we have K nodes in the output layer, one for each class
- Let a_k be the output of the class-k node, i.e. $a_k = (w_k \cdot A)$, where A is the output of the hidden layer, and w_k is the weight vector leading into the class-k node
- We define: $P(y = k|\mathbf{x}) = \frac{\exp a_k}{\sum_{i=1}^{K} \exp a_i}$

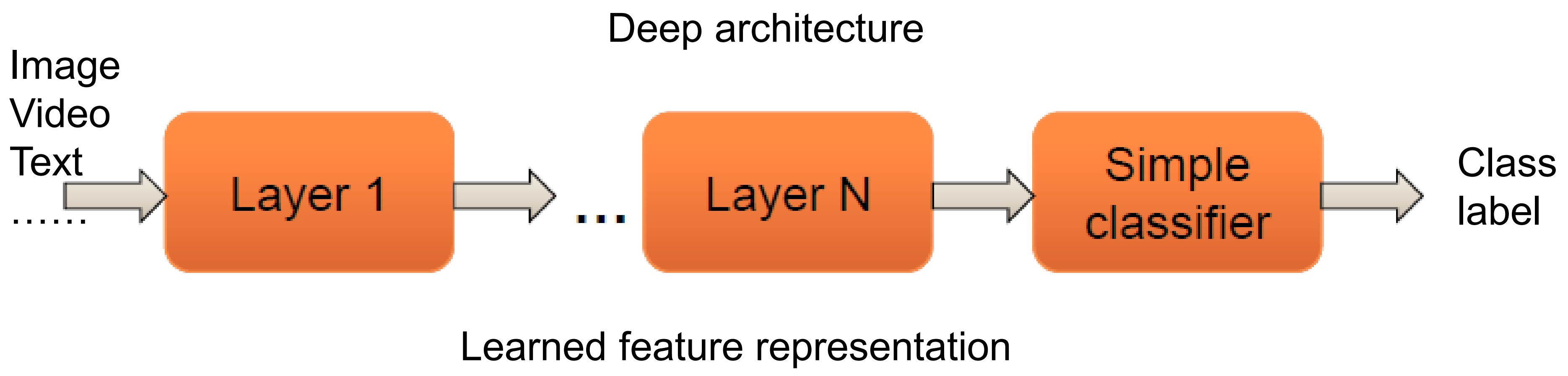


Recent Development

- A recent trend in ML is deep learning, which learns feature hierarchies from large amounts of unlabeled data
- The feature hierarchies are expected to capture the inherent structure in the data
- Can often lead to better classification when used the learned features to train with labeled data
- Neural networks provide one approach for deep learning

Shallow vs Deep Architectures


Traditional shallow architecture
Image
Video
Text
……
Hand-designed feature extraction
Trainable classifier
Class label



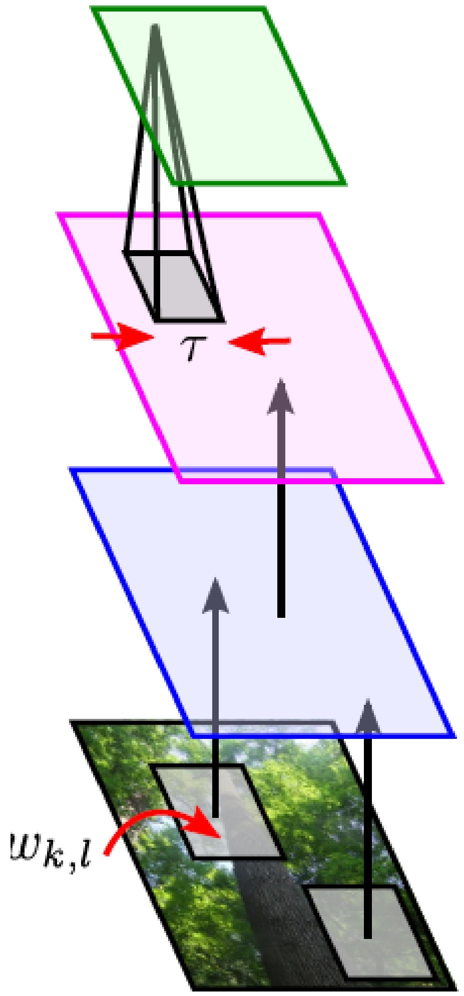
Convolutional neural networks

- A network architecture that has been extremely successful in handling visual, textual and audio data
- Current state of the art on many computer vision tasks

Key ideas behind convolutional neural networks

- Image statistics are translation invariant
  - Need to build translation invariance into the model
  - Tie parameters together in the network
  - Reduce number of parameters
- Low level features/patterns should be local
  - Network should have only local connectivity
  - Reduce # of parameters
- High-level features/patterns will be coarser
  - We can zoom out by subsampling and still capture the high level patterns well

Building blocks of CNN



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \textstyle\sum_{k,l} \boxed{w_{k,l}} z_{i-k,j-l}$$

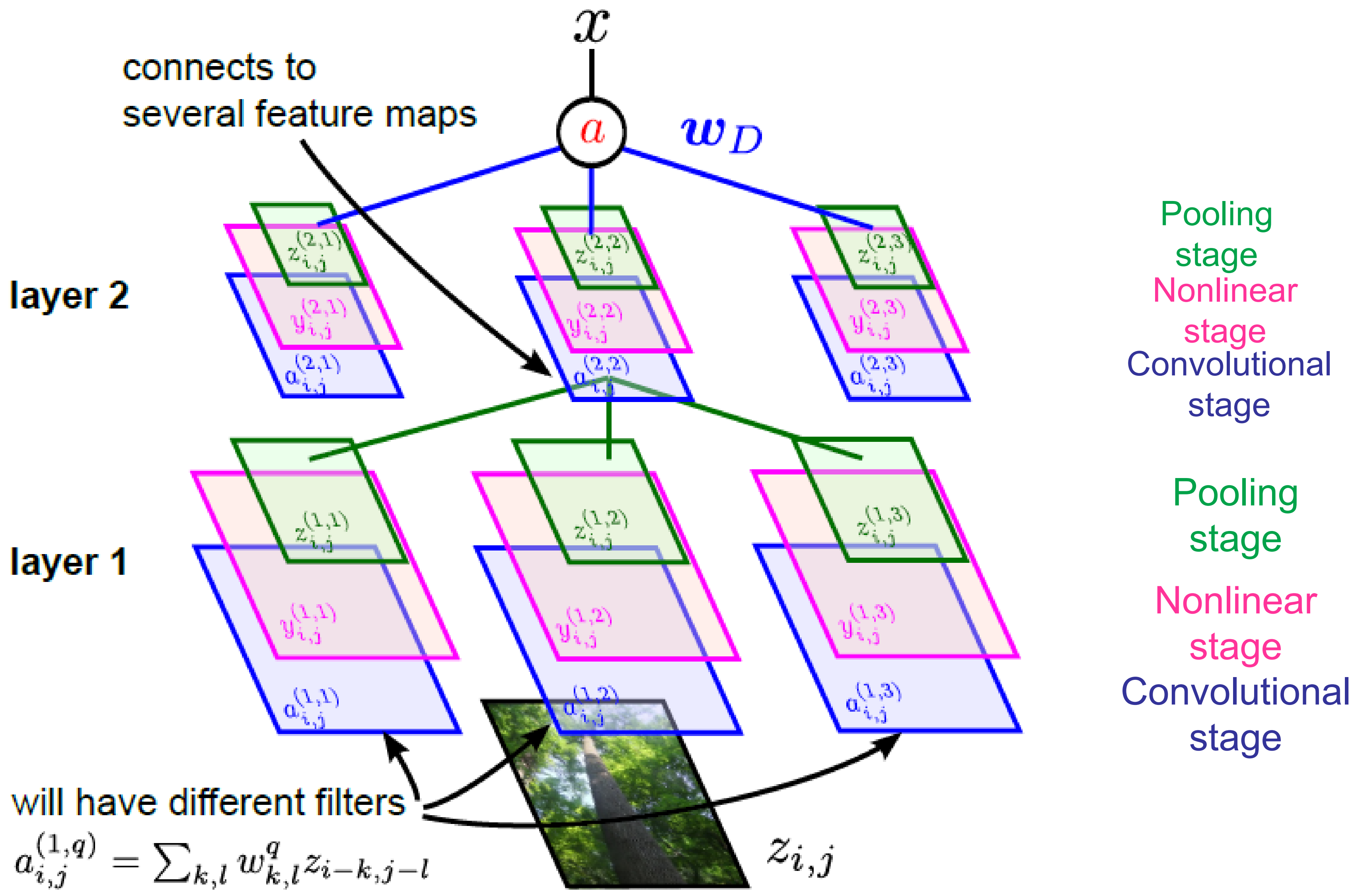
only parameters

convolutional stage

$z_{i,j}$

input image

Full CNN

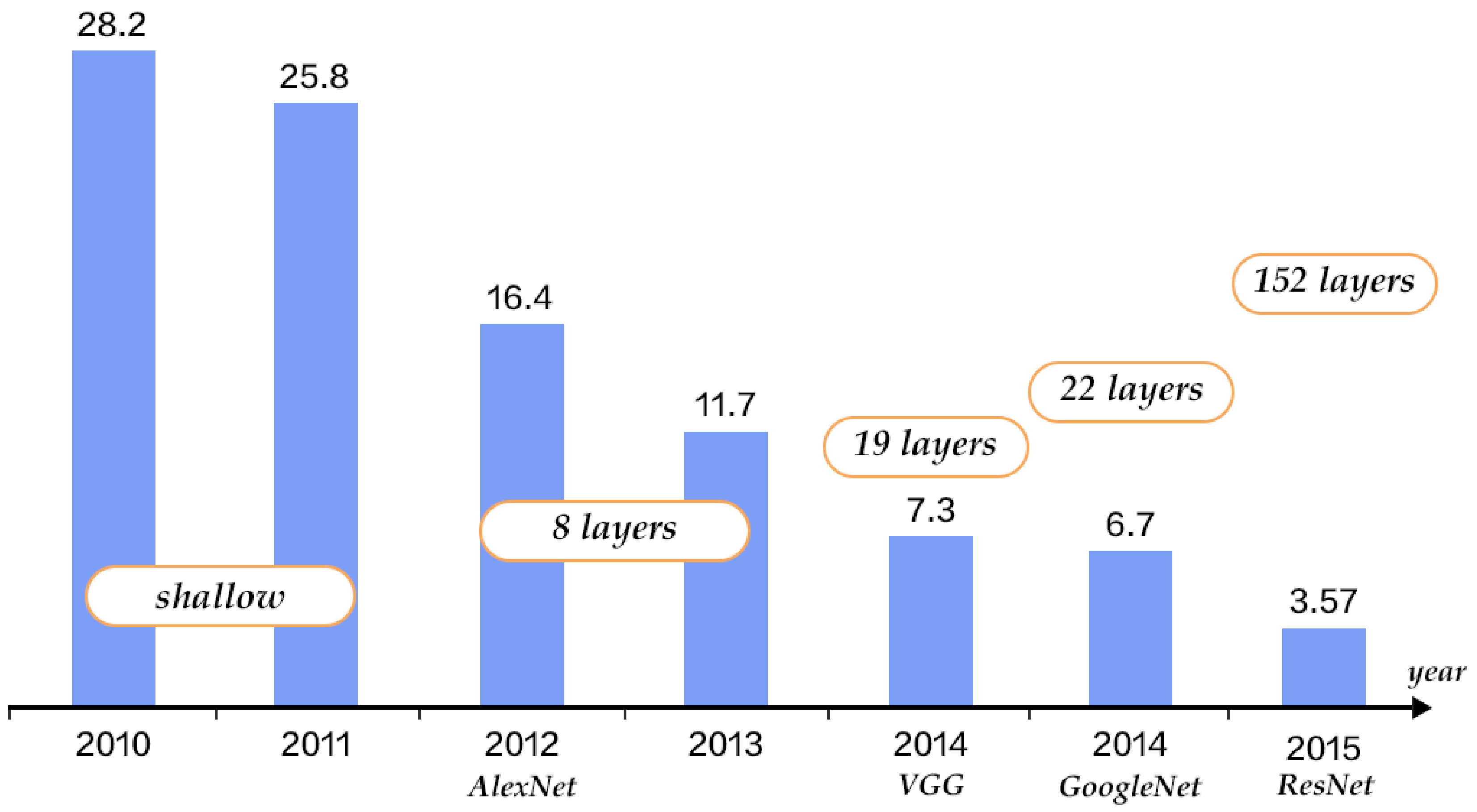


$$a^{(1,q)}_{i,j} = \sum_{k,l} w^q_{k,l} z_{i-k,j-l}$$

Training

- back-propagation for training
- data-augmentation: include shifted, rotations, mirroring, locally distorted versions of the training data
  - Often improves performance substantially
- typical numbers:
  - 5 convolutional layers, 3 fully connected layers in the top
  - 500,000 neurons
  - 50,000,000 parameters
  - 1 week to train (GPUs)

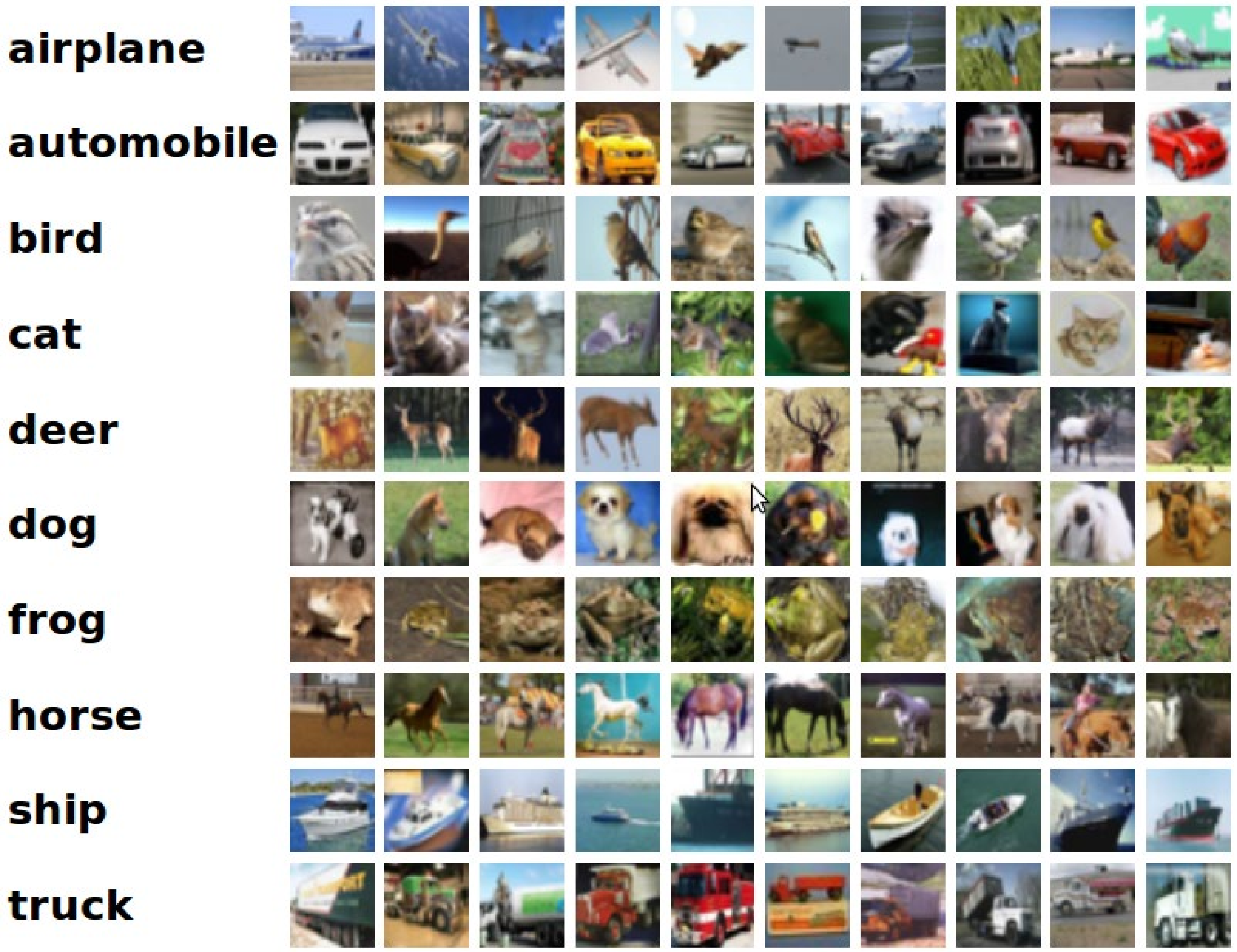
ImageNet Large Scale Vision Recognition Challenge



Error rate of the top performer in each year and corresponding network complexity

Demo

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html



CIFAR 10 dataset: 50,000 training images, 10,000 test images

Summary

- That’s a basic intro
- There are many many types of deep learning architectures – autoencoders, Convolutional netoworks, recurrent networks …
- Various packages: Pytorch, Tensorflow …
- Tremendous impact in vision, speech and natural language processing
- Very fast growing area, to learn more, take the deep learning class next term