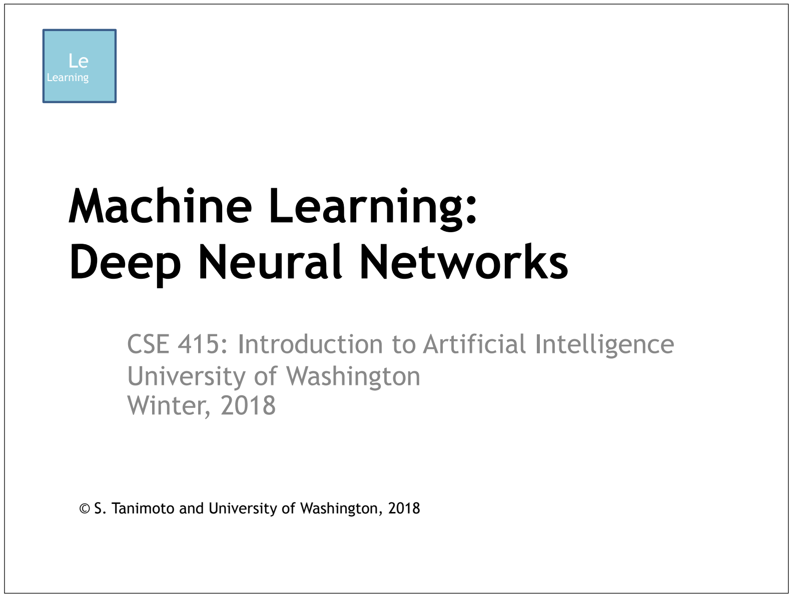
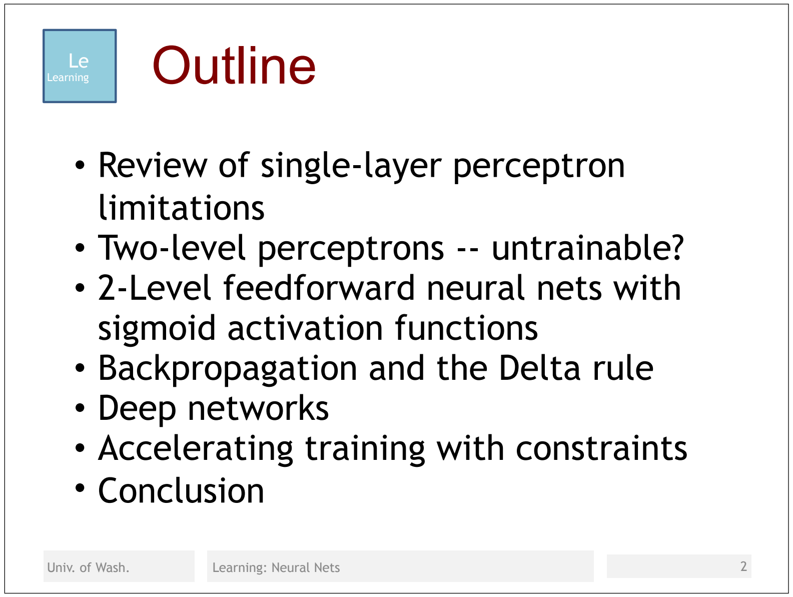


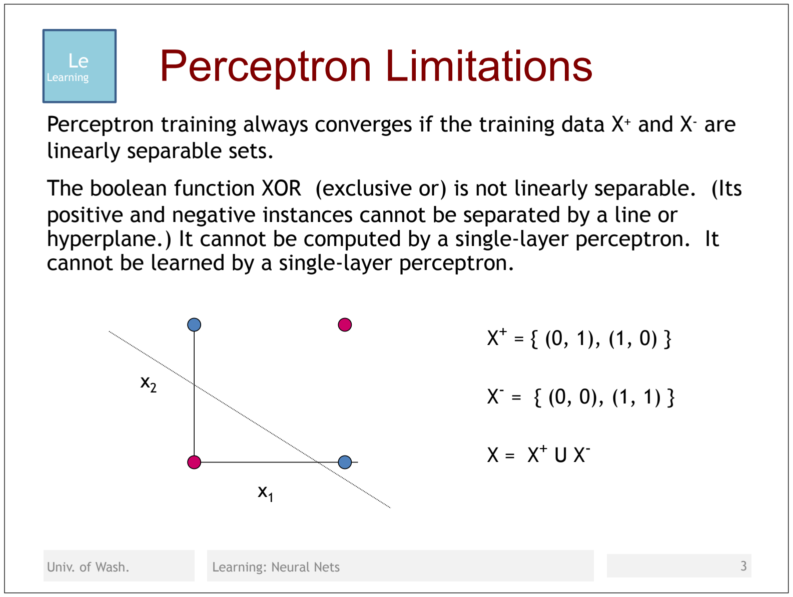
Machine Learning: Deep Neural Networks

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter, 2018



Outline

- Review of single-layer perceptron limitations
- Two-level perceptrons -- untrainable?
- 2-Level feedforward neural nets with sigmoid activation functions
- Backpropagation and the Delta rule
- Deep networks
- Accelerating training with constraints
- Conclusion



Perceptron Limitations

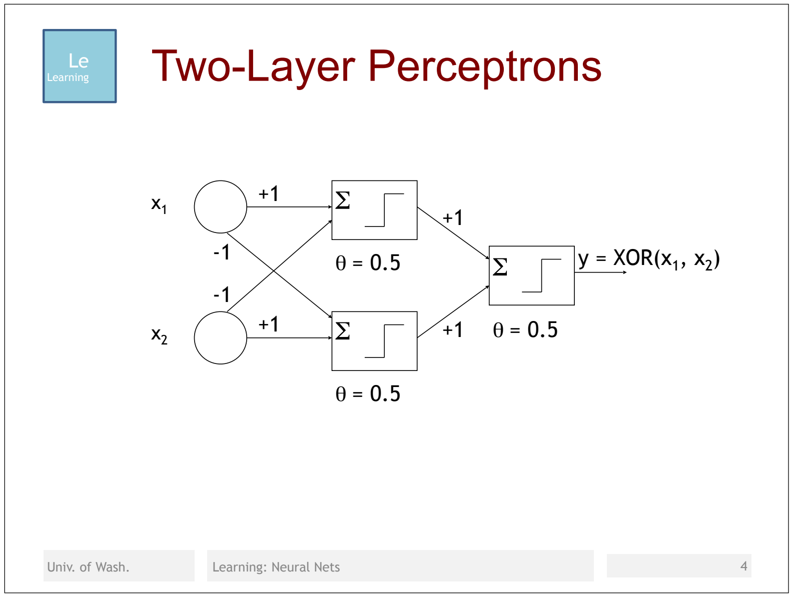
Perceptron training always converges if the training data X^+ and X^- are linearly separable sets.

The boolean function XOR (exclusive or) is not linearly separable. (Its positive and negative instances cannot be separated by a line or hyperplane.) It cannot be computed by a single-layer perceptron. It cannot be learned by a single-layer perceptron.

$X^+ = \{ (0, 1), (1, 0) \}$

$X^- = \{ (0, 0), (1, 1) \}$

$X = X^+ \cup X^-$



Two-Layer Perceptrons

Two-Layer Perceptrons (cont.)

Two-Layer perceptrons are computationally powerful.

However: they are not trainable with a method such as the perceptron training algorithm, because the threshold units in the middle level “block” updating information; there is no way to know what the correct updates to first-level weights should be.

Two-Layer Feedforward Networks with Sigmoid Activation Functions

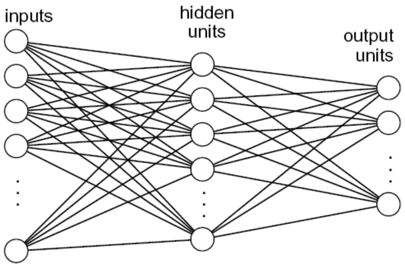
We get: the power of 2-level perceptrons,

plus

the trainability of 1-level perceptrons (well, sort of).

These are sometimes called (a) “backpropagation networks,” (because the training method is called backpropagation) and (b) “two-layer feedforward neural networks.”

Structure of a Backprop. Network



Hidden Node Input Activation

As with perceptrons, a weighted sum is computed of values from the previous level:

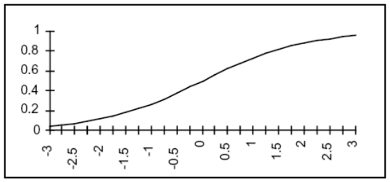
$$h_j = \sum_i w_{ij} x_i$$

However the hidden node does not apply a threshold, but a sigmoid function ...

Sigmoid Activation Functions

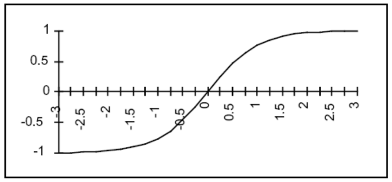
Instead of using threshold functions, which are neither continuous nor differentiable, we use a *sigmoid* function, which is a sort of smoothed threshold function.

$$g_1(h) = 1/(1 + e^{-h})$$



An Alternative Sigmoid Func.

$$g_2(h) = \tanh(h) = (e^h - e^{-h})/(e^h + e^{-h})$$



Sigmoid Function Properties

Both g_1 and g_2 are continuous and differentiable.

$$g_1(h) = 1/(1 + e^{-h})$$

$$g_1'(h) = g_1(h)\ (1 - g_1(h))$$

$$g_2(h) = \tanh(h) = (e^h - e^{-h})/(e^h + e^{-h})$$

$$g_2'(h) = 1 - g_2(h)^2$$

Training Algorithm

Each *training example* has the form $\langle X_i, T_i \rangle$, were X_i is the vector of inputs, and T_i is the desired corresponding output vector.

An *epoch* is one pass through the training set, with an adjustment to the networks weights for each training example. (Use the “delta rule” for each example.)

Perform as many epochs of training as needed to reduce the classification error to the required level.

If there are not enough hidden nodes, then training might not converge.

Delta Rule

For each training example $\langle X_i, T_i \rangle$, Compute $F(X_i)$, the outputs based on the current weights.

To update a weight w_{ij}, add ∇w_{ij} to it, where

$$\nabla w_{ij} = \eta \, \delta_j \, F_j \qquad (\eta \text{ is the training rate.})$$

If w_{ij} leads to an output node, then use

$$\delta_j = (t_j - F_j) \, g'_j(h_j)$$

If w_{ij} leads to a hidden node, then use “backpropagation”:

$$\delta_j = g'_j(h_j) \sum_k \delta_k \, w_{kj}$$

The δ_k in this last formula comes from the output level, as computed a

Performance of Backpropagation

Backpropagation is slow compared with 1-layer perceptron training.

The training rate η can be set large near the beginning and made smaller in later epochs.

In principle, backpropagation can be applied to networks with more than one layer of hidden nodes, but this slows the algorithm much more.

Setting the Number of Hidden Nodes

The number of nodes in the hidden layer affects generality and convergence.

If too few hidden nodes: convergence may fail.

Few but not too few nodes: possibly slow convergence but good generalization

Too many hidden nodes: Rapid convergence, but “overfitting” happens.

Overfitting: the learned network handles the training set, but fails to generalize effectively to similar examples not in the training set.

Applications of 2-Layer Feedforward Neural Networks

These networks are very popular as trainable classifiers for a wide variety of pattern data.

Examples:

•Speech recognition and synthesis

•Visual texture classification

•Optical character recognition

•Control systems for robot actuators

Deep Neural Networks

By using multiple levels (i.e., L > 2) more complex recognition tasks can be learned.

However backpropagation becomes extremely slow.

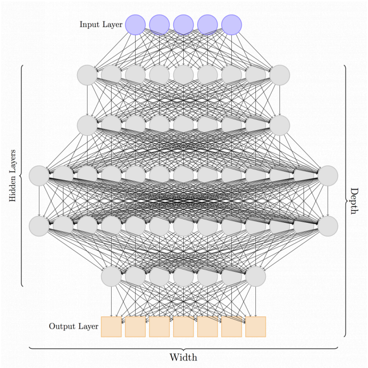


Illustration courtesy of A. Corbin at https://developingideas.me/deepneuralnetworkoverview/

Deep Convolutional Neural Networks

In some applications such as image understanding, full connectivity is not required between the input layer and the next layer. Rather, only a neighborhood around a pixel is relevant.

Not only that, the neighbor weights centered on one pixel can be identical to those around the other pixels.

This kind of constraint reduces the number of weights to be learned tremendously.

This typically means that the first level or first k levels implement forms of convolution, and hence the name.

Deep Convolutional Neural Networks

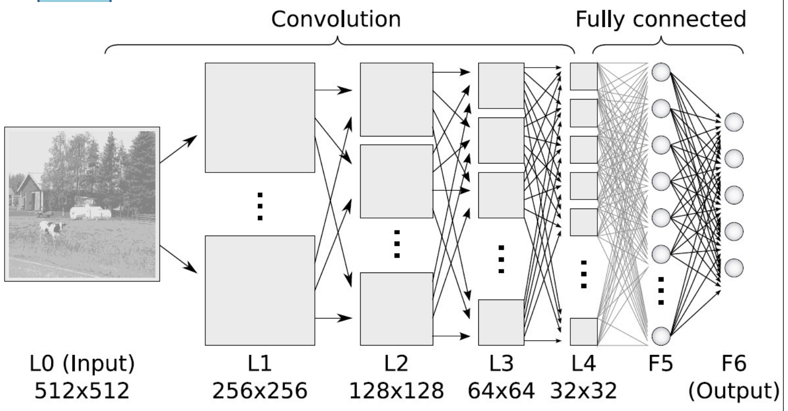


Illustration courtesy of Univ. of Bonn, at https://www.ais.uni-bonn.de/deep_learning/images/Convolutional_NN.jpg

Conclusion

- Neural networks have recently proven to be a powerful means to working systems for object classification.
- Large numbers of training examples are required, but using the web, and various archives, data is increasingly available.
- One issue with NN methods is that they are "opaque" and the weights and activations are typically difficult to understand in the context of the overall task.