

Multilayer Percetprons

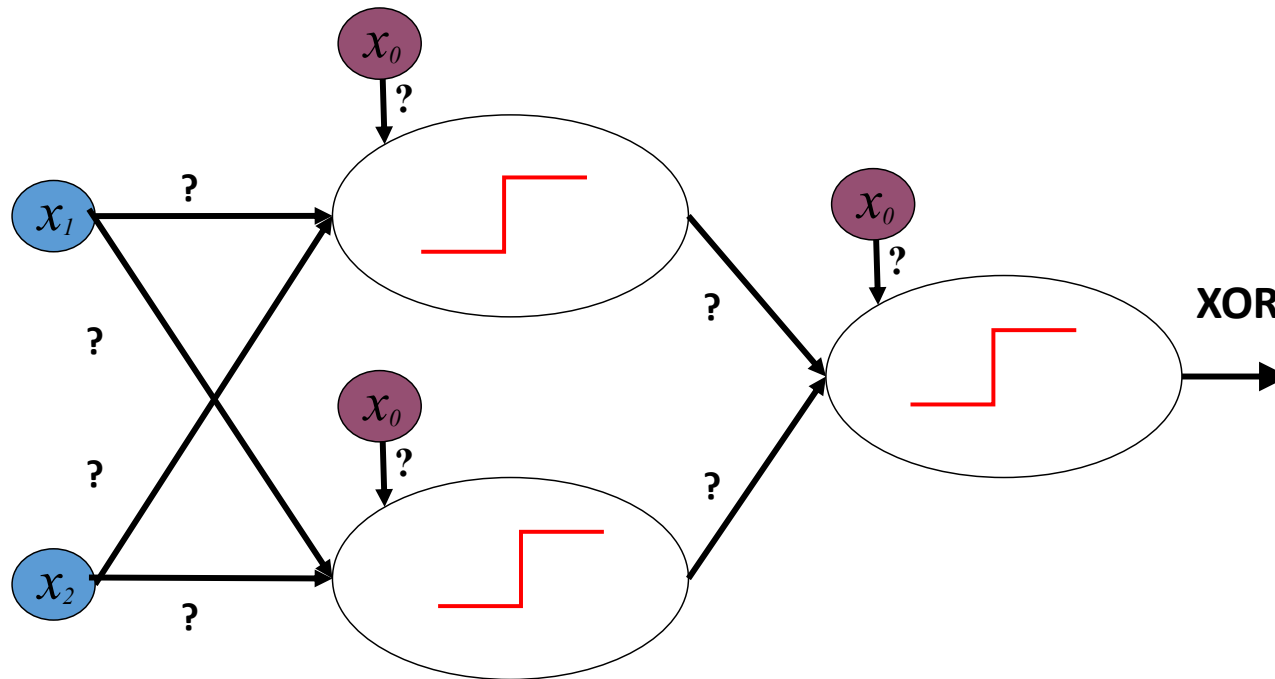
Bryan Pardo

Deep Learning

Northwestern University

Deep Learning: Bryan Pardo, Northwestern University, Fall 2020

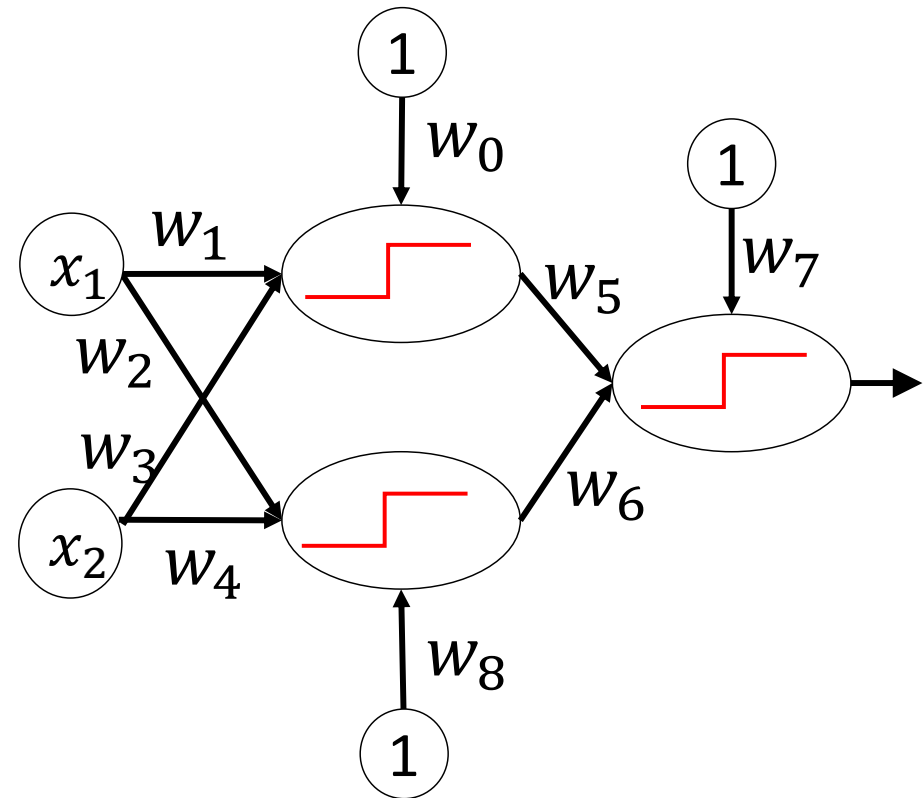
Combining perceptrons can make any Boolean function



...if you can set the weights & connections right

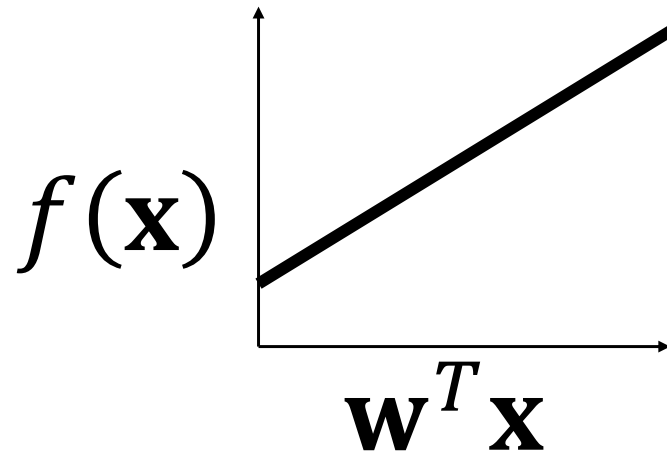
Problem with a step function: Assignment of error

- Stymies multi-layer weight learning
- Limits us to a single layer of units
- Thus, only linear functions
- You can hand-wire XOR perceptrons, but the system can't learn XOR with perceptrons



Linear Units & Delta Rule

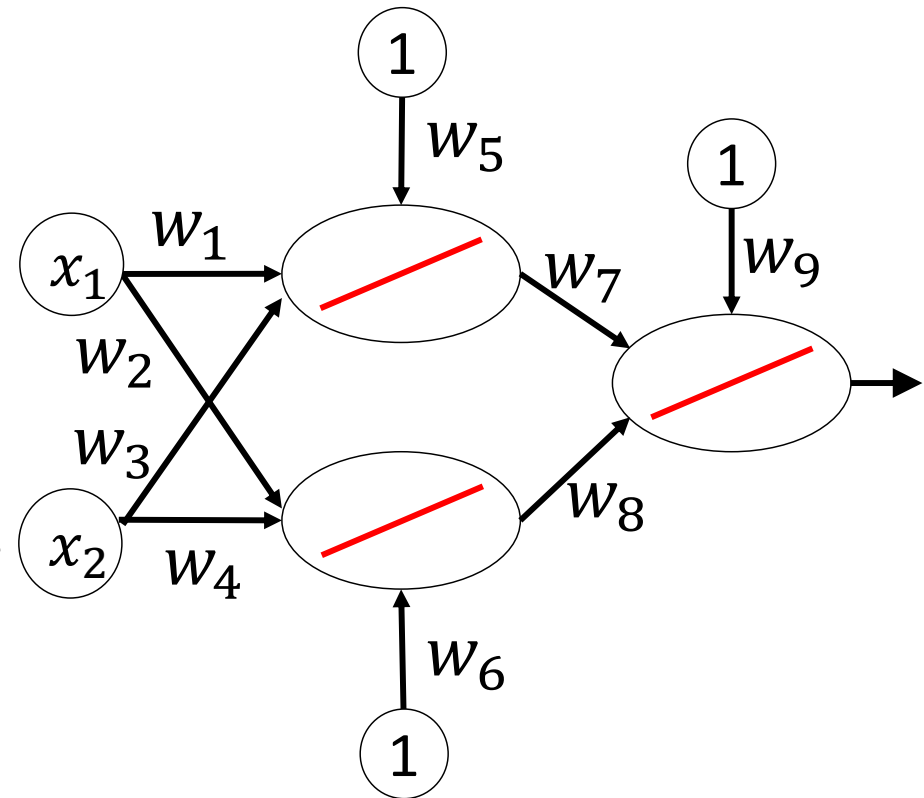
Solution: Remove the step function



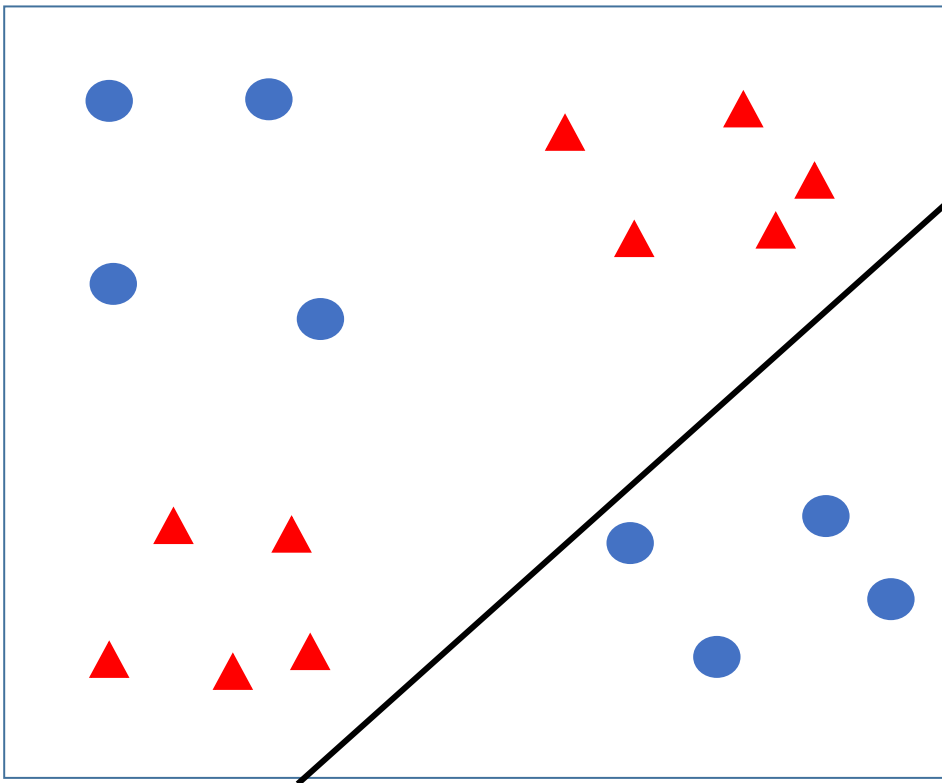
$$f(\mathbf{x}) = \sum_{i=0}^n w_i x_i = \mathbf{w}^T \mathbf{x}$$

Better & worse than a perceptron

- All changes in input result in changed output
- This gives us a gradient everywhere
- We can learn multiple layers of weights.
- **Combining linear functions only gives you linear functions**
- you can't represent XOR



Many linear units: Only linear decisions



This is XOR.

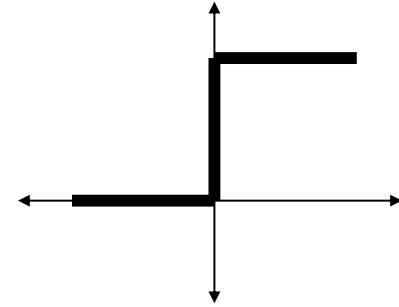
A multilayer
perceptron with
linear units
CANNOT learn XOR

The Sigmoid Unit

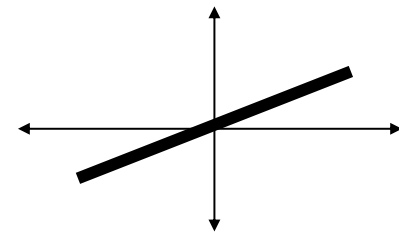
Rumelhart, David E., James L. McClelland, and PDP Research Group. Parallel distributed processing. Vol. 1. Cambridge, MA, USA:: MIT press, 1987.

Sigmoid (aka Logistic) function: best of both

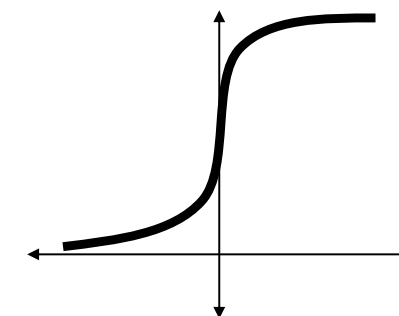
- Perceptron
$$f(x) = \begin{cases} 1 & \text{if } 0 < \sum_{i=0}^n w_i x_i \\ -1 & \text{else} \end{cases}$$



- Linear
$$f(x) = \mathbf{w}^T \mathbf{x} = \sum_{i=0}^n w_i x_i$$

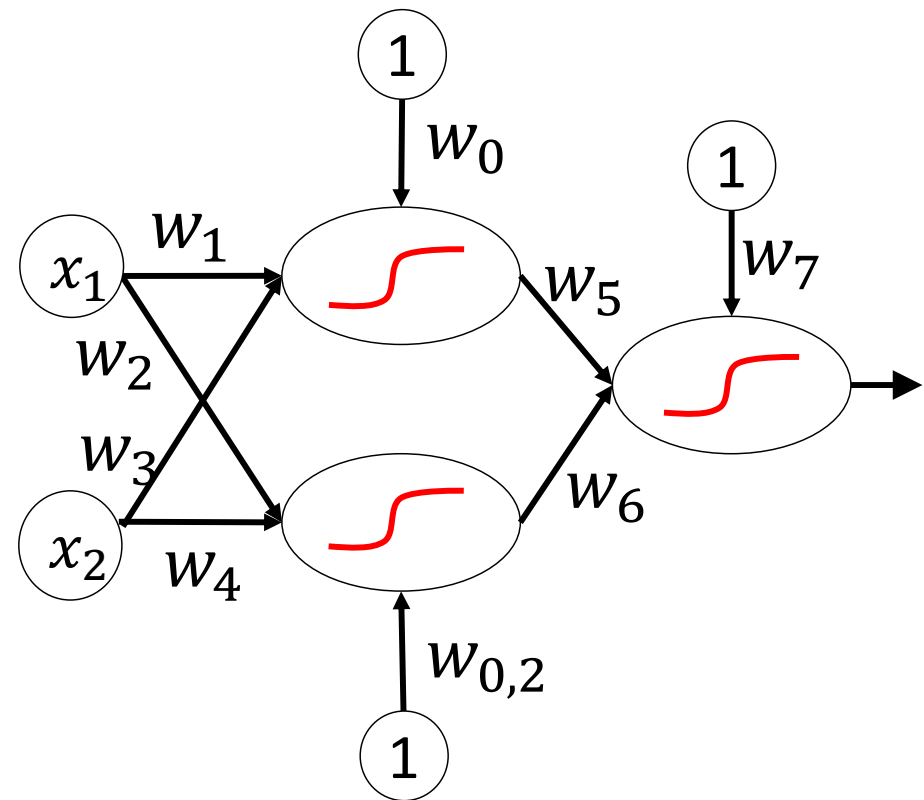


- Sigmoid
$$f(x) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$



A network of sigmoid units

- Small changes in input result in output
- This gives us a gradient everywhere
- We can learn multiple layers of weights.
- Combining layers gives non-linear functions



Sigmoid changes (almost) everything

Easy to differentiate

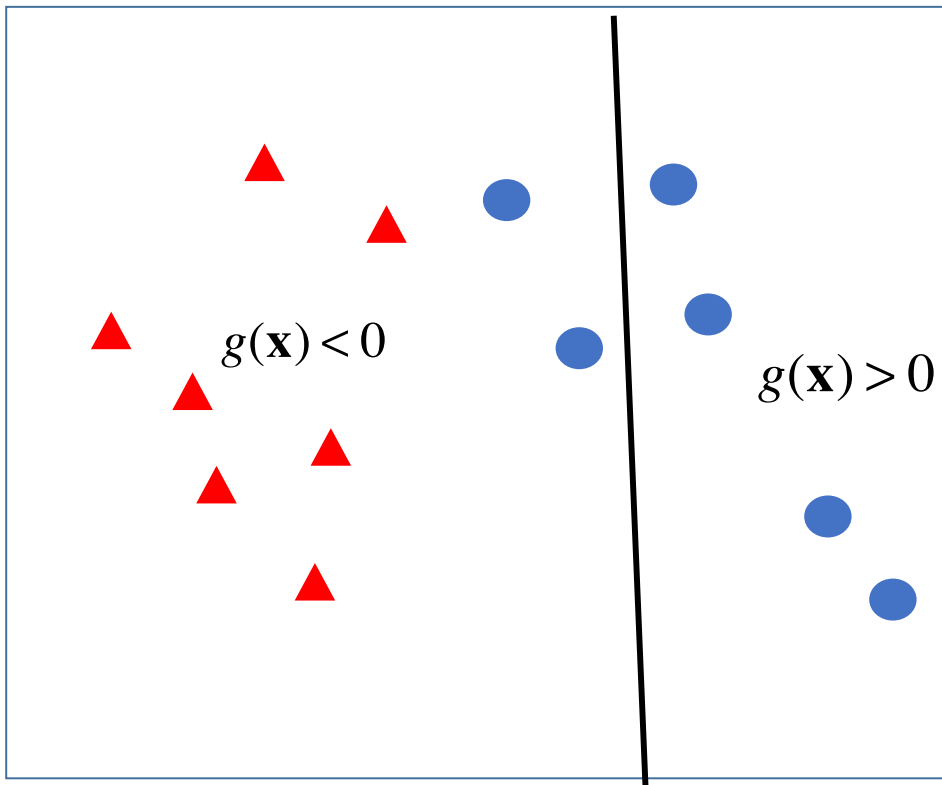
$$\sigma'(\mathbf{w}^T \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})(1 - \sigma(\mathbf{w}^T \mathbf{x}))$$

Gradient everywhere

This allows backpropagation of the gradient through multiple layers

Nonlinearity allows arbitrary nonlinear functions to be built by using multiple layers.

Example objective J : sum of squared errors

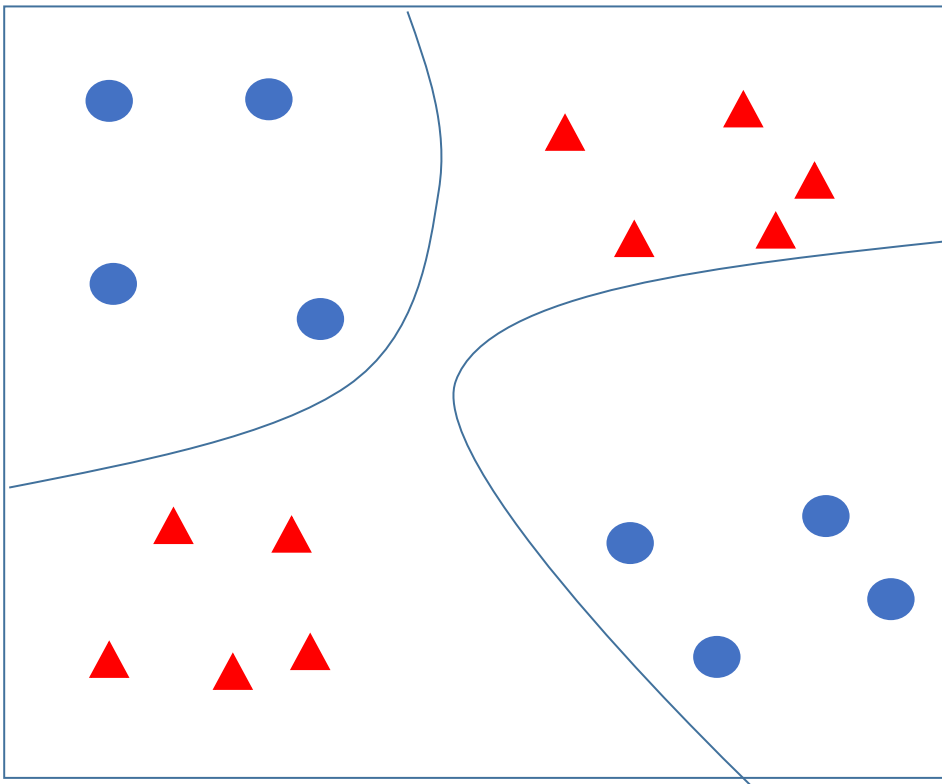


$$h(x) = f(x) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$$

$$SSE = \sum_i^n (y_i - h(\mathbf{x}_i))^2$$

Gradient non-zero everywhere!

Multilayer Perceptron with sigmoid units



This is XOR.

A multilayer perceptron with sigmoid units CAN learn XOR...or any other arbitrary Boolean function.

The promise of many layers

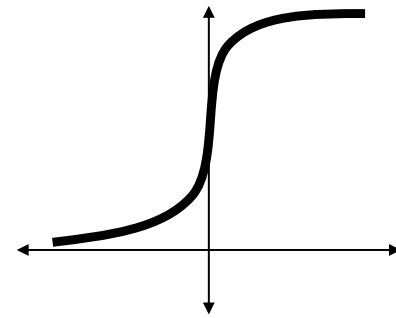
- Each layer learns an abstraction of its input representation (we hope)
-
- As we go up the layers, representations become increasingly abstract

- The hope is that the intermediate abstractions facilitate learning functions that require non-local connections in the input space (recognizing rotated & translated digits in images, for example)

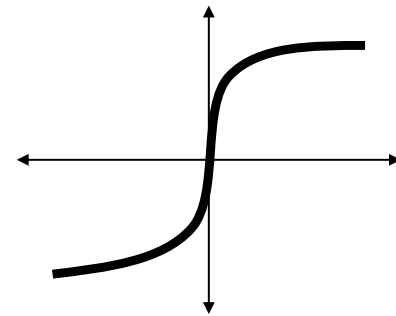
- Modern neural networks are up to 100 layers deep

TanH: A shifted sigmoid

- Sigmoid $f(x) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x})}}$

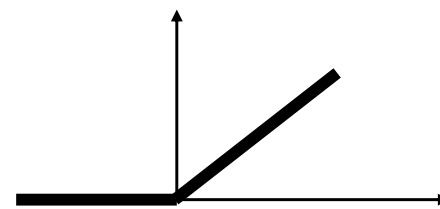


- TanH $f(x) = \frac{2}{1 + e^{-2(\mathbf{w}^T \mathbf{x})}} - 1$

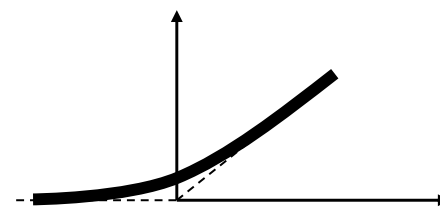


Rectified Linear Unit (ReLU) & Soft Plus :

- ReLU $f(x) = \max(0, \mathbf{w}^T \mathbf{x})$



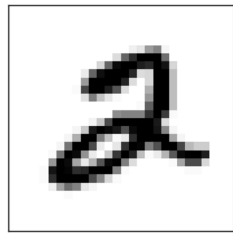
- Soft Plus $f(x) = \ln(1 + e^{\mathbf{w}^T \mathbf{x}})$



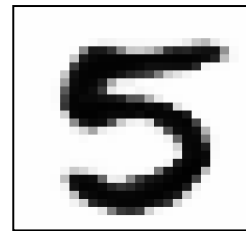
- Both can be combined in layers to make non-linear functions

“One Hot” Encoding

- A vector of values where a single element is 1 and all the rest are 0
- Common way to encode the true label, y , in a multi-class labeling problem
- Can be interpreted as a probability distribution



$y = 0010000000$



$y = 0000010000$

Probability distribution

- * Discrete random variable X represents some experiment.
- * $P(X)$ is the probability distributions over $\{x_1, \dots, x_n\}$,
the set of possible outcomes for X .
- * These outcomes are mutually exclusive.
- * Their probabilities sum to one: $\sum_{i=1}^n P(x_i) = 1$

Soft Max Function

- Turns an N-dimensional vector of real numbers into a probability distribution
- For a deep net, a_i is the output of the i th node in the output layer

$$p_i = \frac{e^{a_i}}{\sum_{j=1}^N a_j}$$

Cross Entropy Loss Function

Given: “true” distribution $y = \{y_1, y_2, \dots, y_k\}$ <-often a one-hot encoding
and estimated distribution $p = \{p_1, p_2, \dots, p_k\}$ <-soft max over the last layer

Define cross entropy loss between 2 distributions as

$$L(y, p) = - \sum_{i=1}^N y_i \log(p_i)$$

A common approach...

- Define labels with a one-hot vector encoding
- Make the last layer have n nodes for an n -way classification problem
- Apply soft max to the last layer
- Use a cross-entropy loss function
- The resulting derivative of the loss function is wonderfully simple:

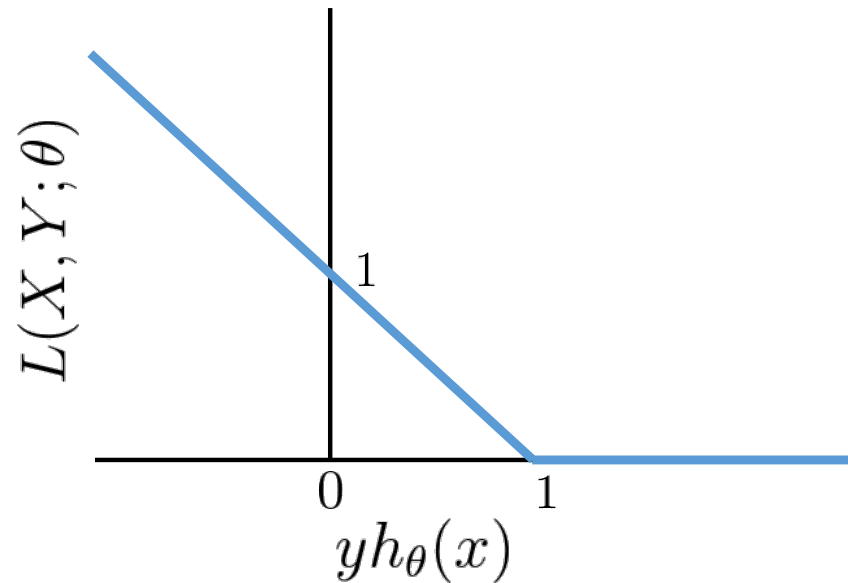
$$\frac{\partial L}{\partial a_i} = p_i - y_i$$

L is the loss, i is the index to a node, a is the output of the last layer, p is the softmax probability and y is the label.

Hinge Loss

$$L_H(X, Y; \theta) = \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i h_\theta(x_i))$$

- Loss only happens if the data is on the wrong side of the line.



Hinge Loss

The loss function:

$$L_H(X, Y; \theta) = \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i h_\theta(x_i))$$

The gradient of the loss function is...

$$\nabla_\theta L_H(X, Y; \theta) = \frac{1}{N} \sum_{i=1}^N [[1 - y_i h_\theta(x_i) > 0]] (-y_i \nabla_\theta h_\theta(x_i))$$

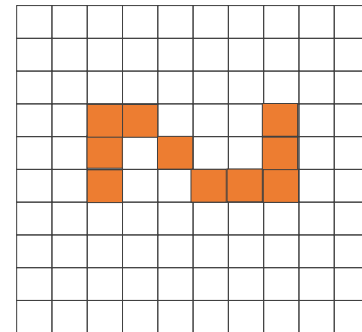
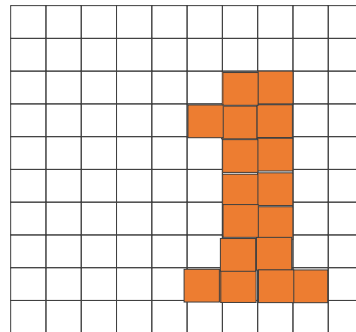
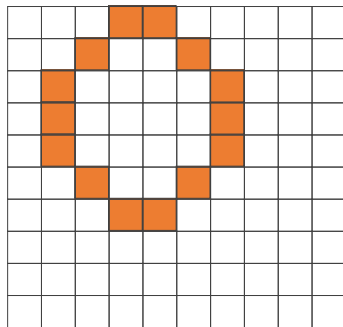
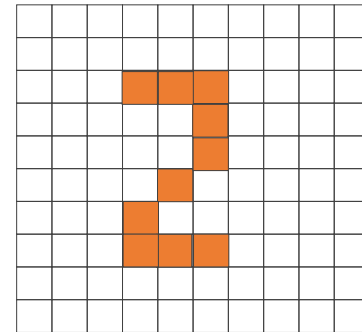
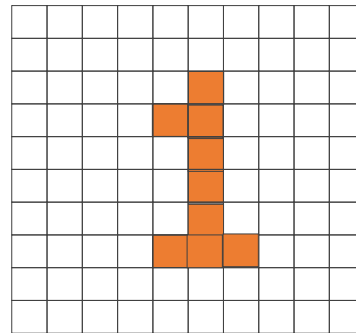
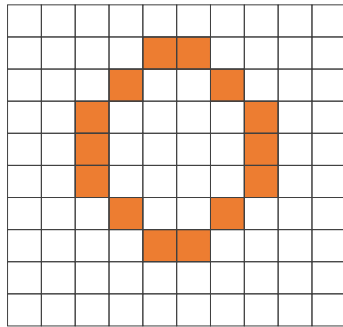
For example, if we use a linear model $h_\theta(x_i) = \theta^T x_i$,

$$\nabla_\theta L_H(X, Y; \theta) = \frac{1}{N} \sum_{i=1}^N [[1 - y_i \theta^T x_i > 0]] (-y_i x_i)$$

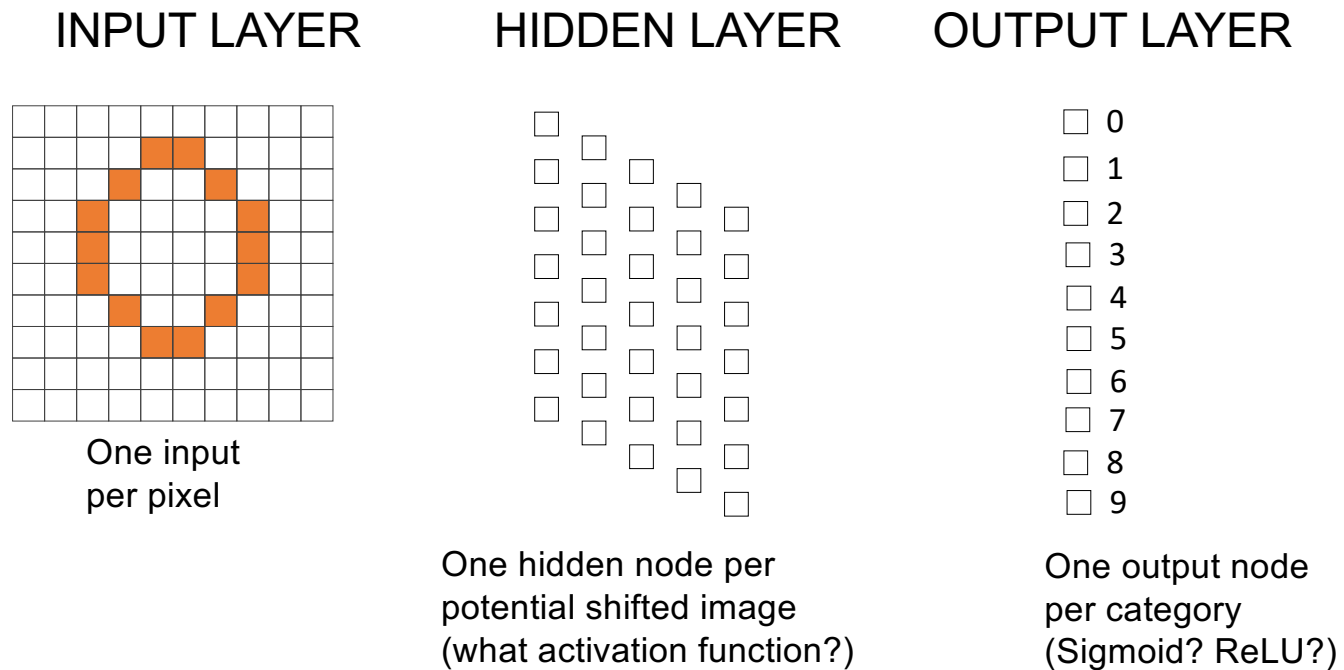
Design choices

- Define the function you want to learn
- Determine an encoding for the data
- Pick a network architecture
 - Number of layers (between 3 and 100)
 - Activation functions function (tanh,ReLU, linear)
 - Select how units connect within and between layers
- Pick a gradient descent algorithm
- Pick regularization approach (e.g. dropout)

Classifying images of digits

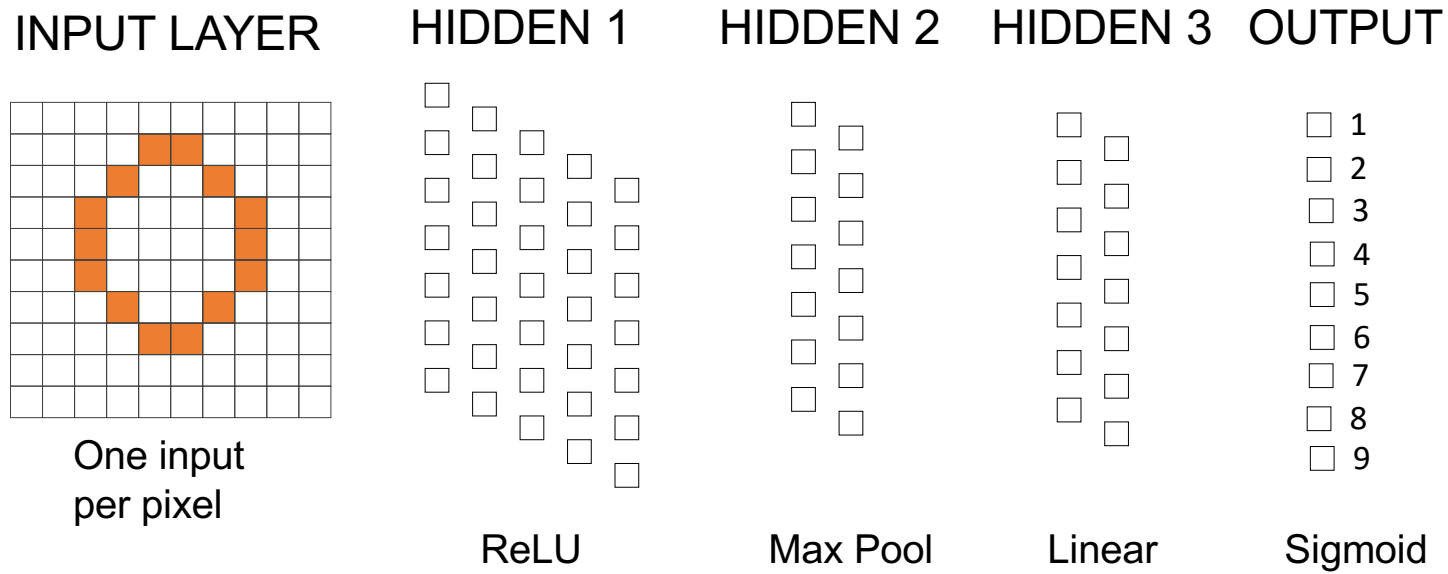


A possibility: one fully-connected hidden layer



Each node is connected to EVERY node in the prior layer (it is just too many lines to draw)

Another possibility



HUGE DESIGN SPACE!