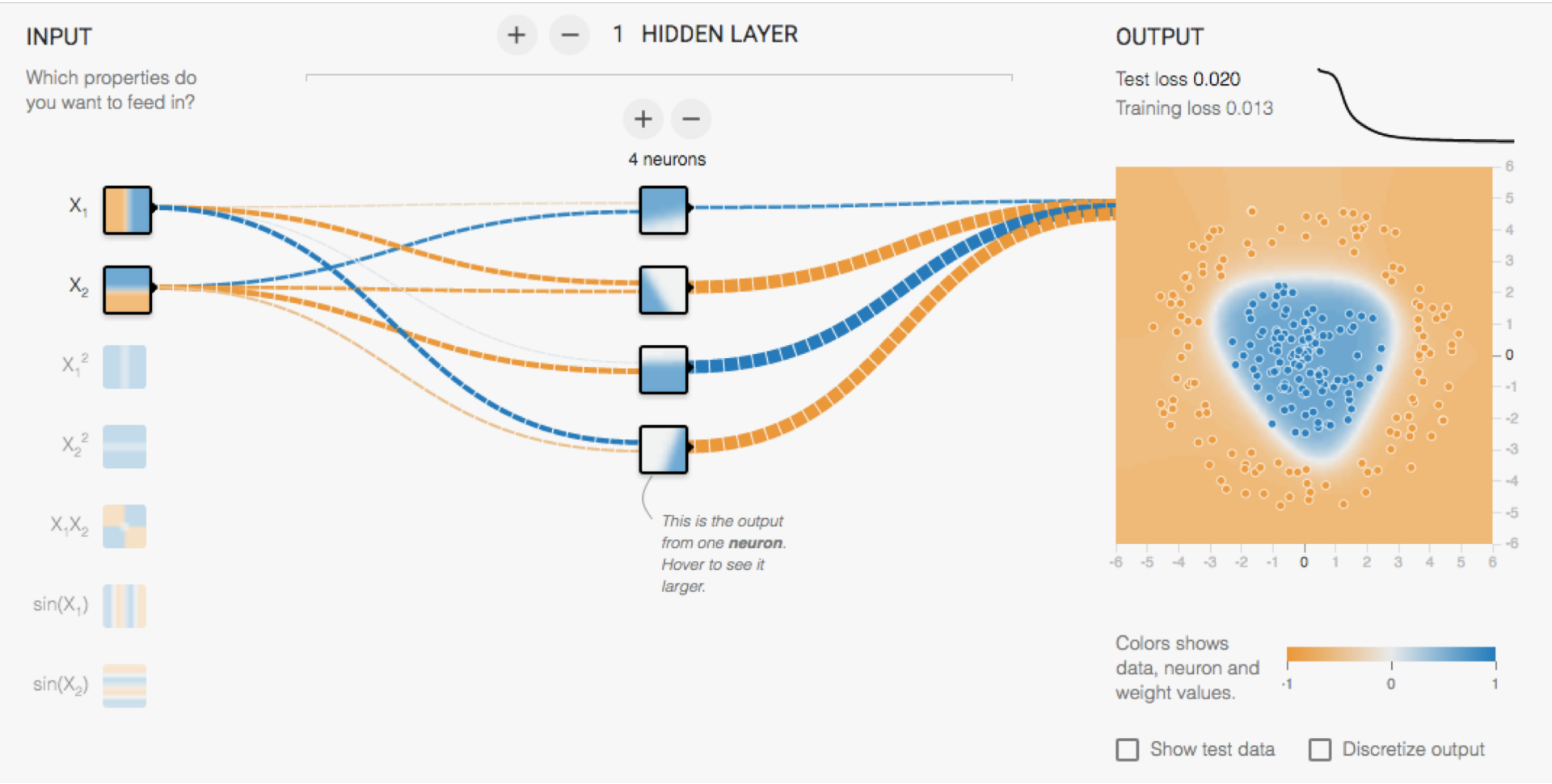
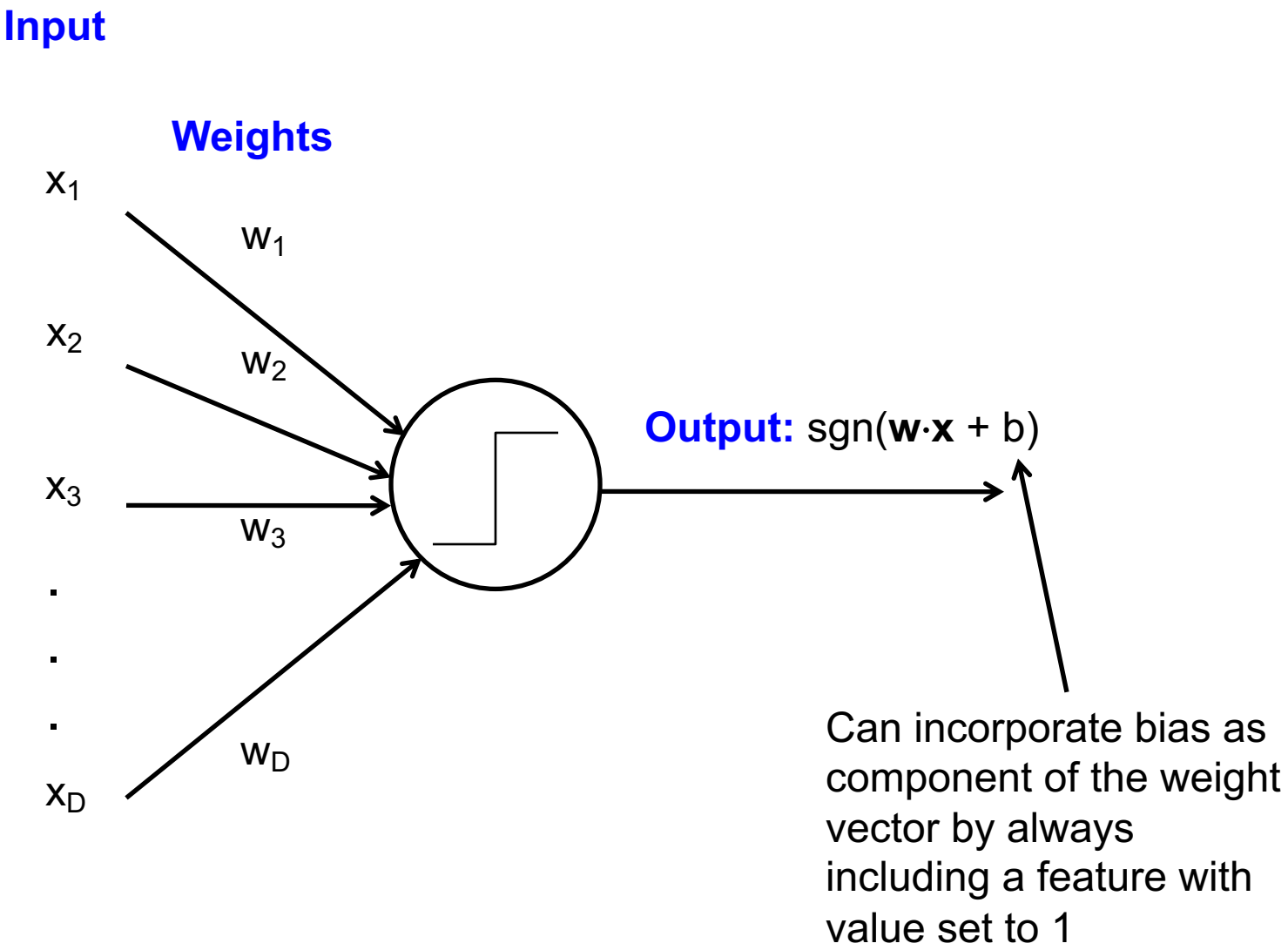
Neural networks

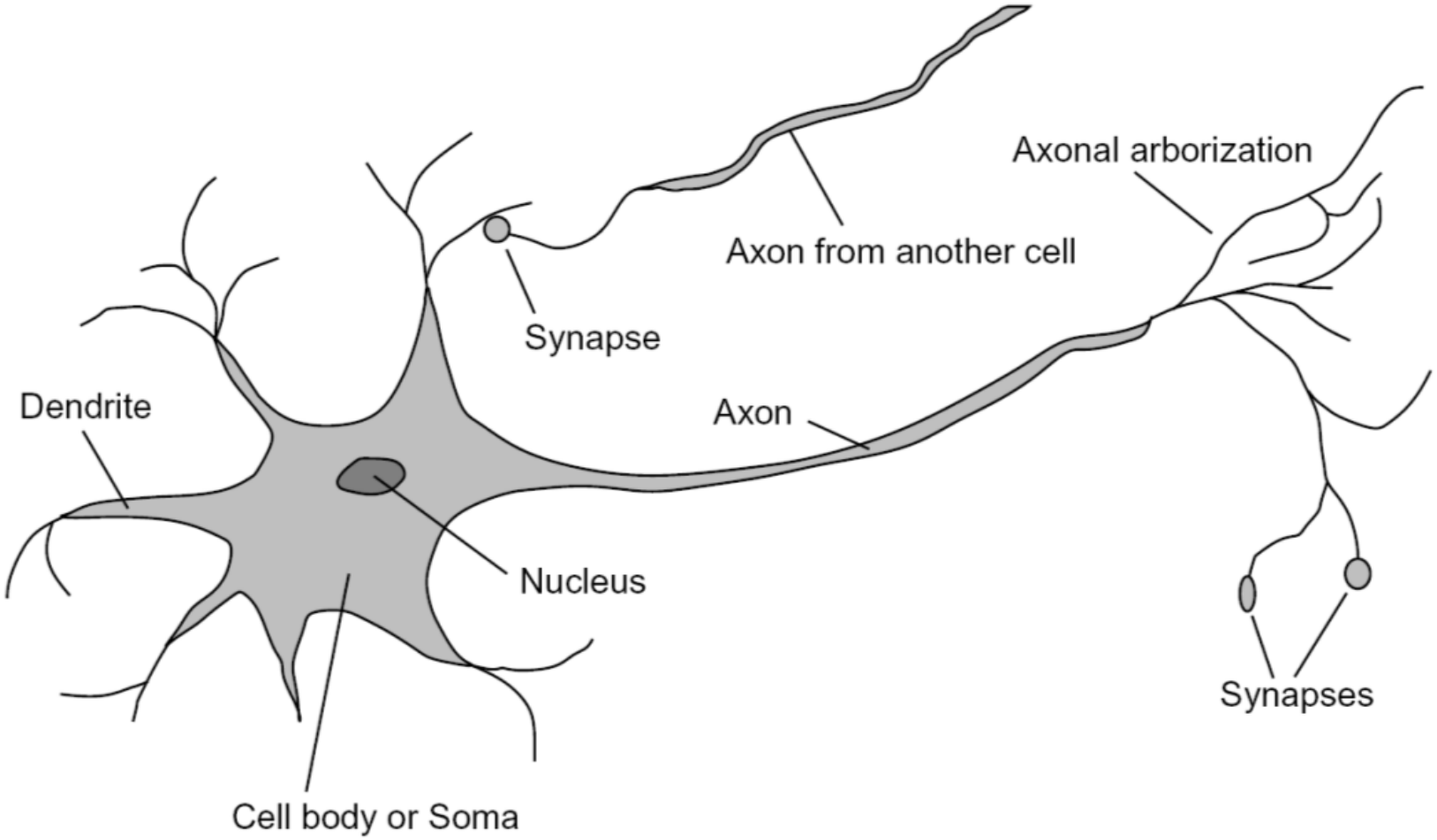


http://playground.tensorflow.org/

Perceptron


Input
Weights
x_1
w_1
x_2
w_2
x_3
w_3
w_D
x_D
Output: sgn(**w**·**x** + b)
Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Loose inspiration: Human neurons



Biological Neurons

- Human brain: 100, 000, 000, 000 neurons
- Each neuron receives input from 1,000 others
- Impulses arrive simultaneously
- Combined*
  - an impulse can either increase or decrease the possibility of nerve pulse firing
- If sufficiently strong, a nerve pulse is generated
- The pulse forms the input to other neurons.

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

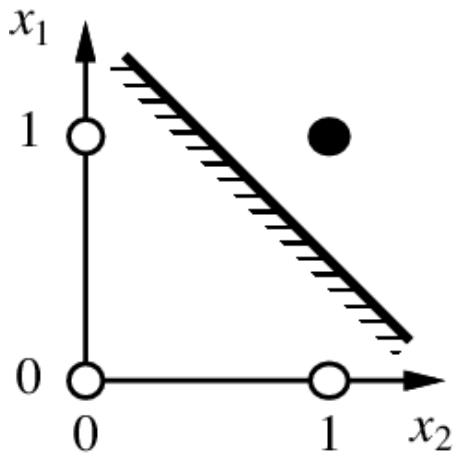
Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

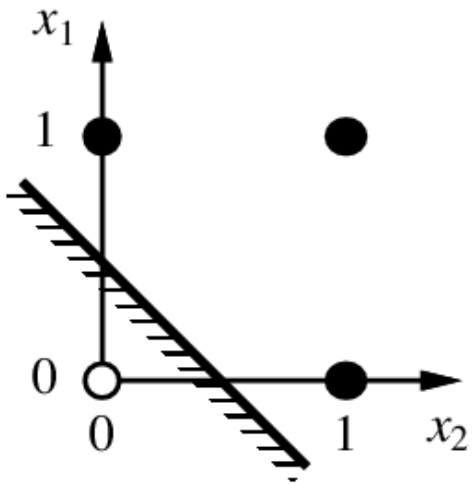
Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

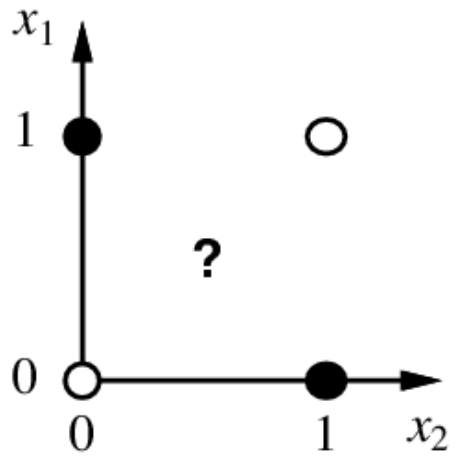
Linear separability



x_1 **and** x_2



x_1 **or** x_2



x_1 **xor** x_2

Perceptron training algorithm

- Initialize weights
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
  - Classify with current weights: $y' = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$
  - If classified incorrectly, update weights:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

  - α is a *learning rate* that should decay as a function of epoch t, e.g., 1000/(1000+t)

Perceptron update rule

$$y' = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(y - y' \right) \mathbf{x}$$

- The raw response of the classifier changes to

$$\mathbf{w} \cdot \mathbf{x} + \alpha \left(y - y' \right) \left\| \mathbf{x} \right\|^2$$

- If y = 1 and y' = -1, the response is initially negative and will be increased
- If y = -1 and y' = 1, the response is initially positive and will be decreased

Implementation details

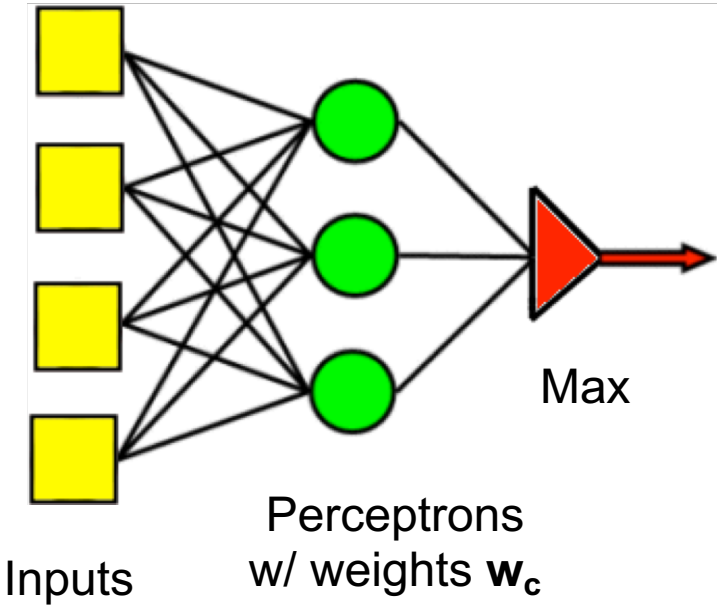
- Bias (add feature dimension with value fixed to 1) vs. no bias
- Initialization of weights
- Learning rate decay: as a function of 1/t
- Number of epochs (passes through the training data): monitor training and validation error
- In each epoch, cycle through training examples randomly

Convergence of perceptron update rule

- **Linearly separable data:** converges to a perfect solution
- **Non-separable data:** converges to a minimum-error solution assuming learning rate decays as O(1/t) and examples are presented in random sequence

Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $c = \text{argmax}_c \, \mathbf{w}_c \cdot \mathbf{x}$

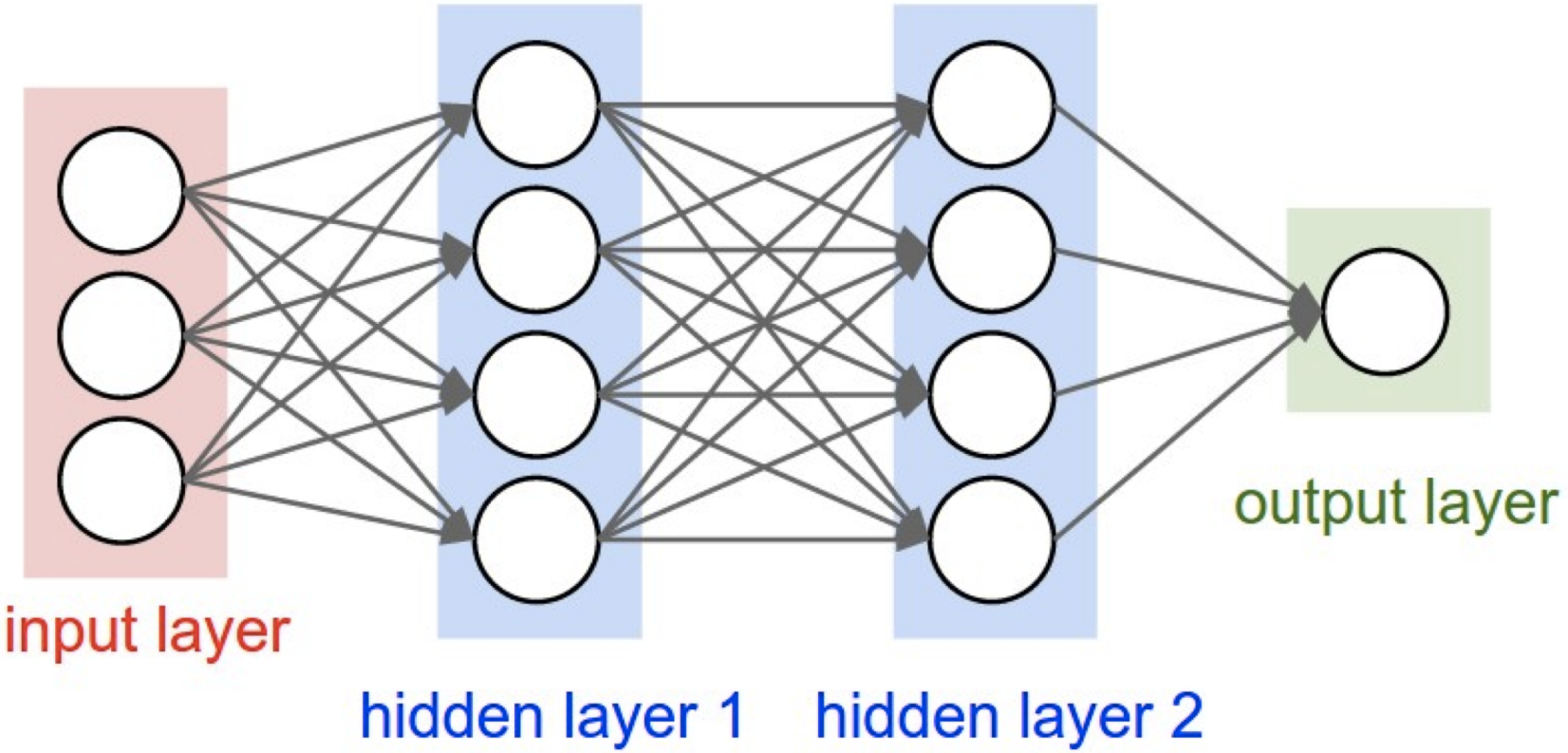


Multi-class perceptrons

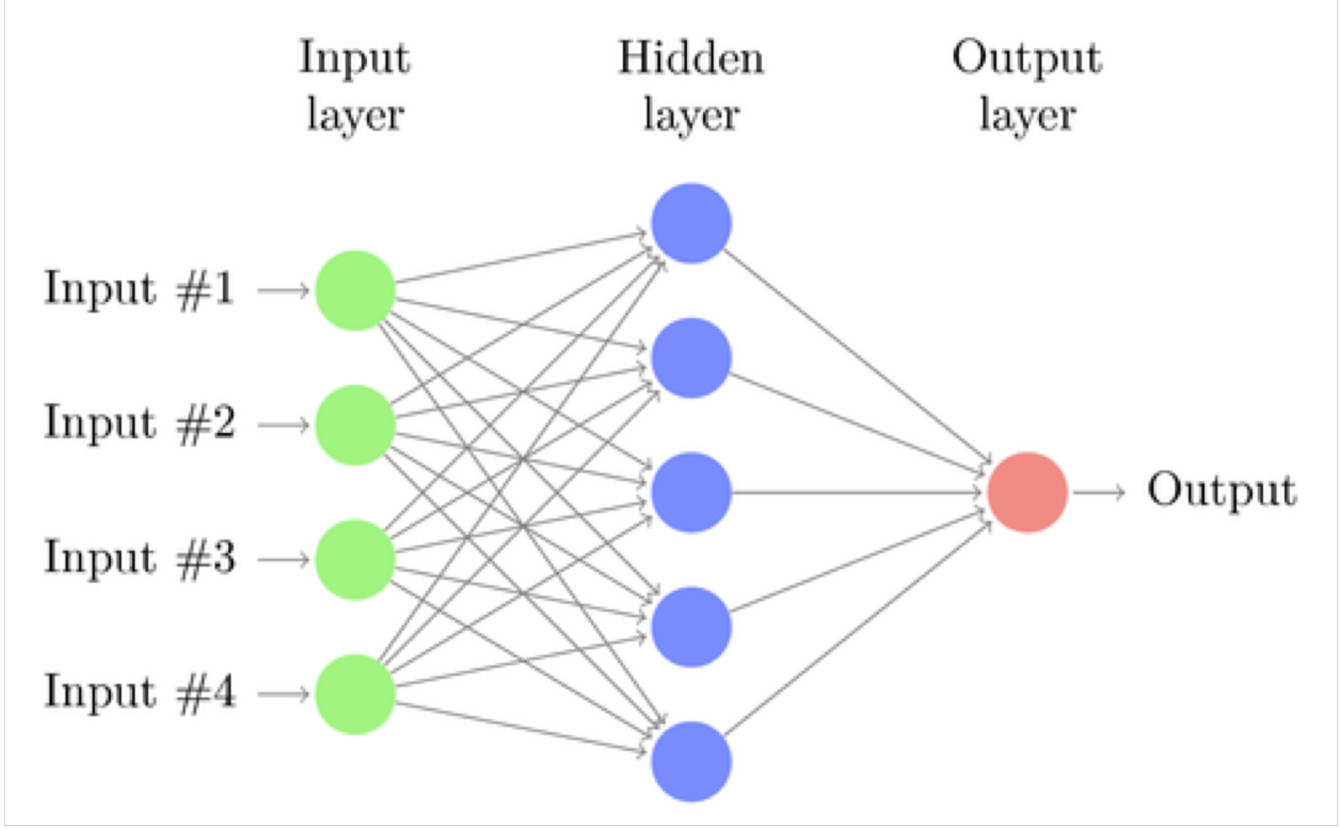
- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $c = \text{argmax}_c \, \mathbf{w}_c \cdot \mathbf{x}$
- Update rule: suppose example from class c gets misclassified as c'
  - Update for c: $\mathbf{w}_c \leftarrow \mathbf{w}_c + \alpha \mathbf{x}$
  - Update for c': $\mathbf{w}_{c'} \leftarrow \mathbf{w}_{c'} - \alpha \mathbf{x}$

How do we make nonlinear classifiers out of perceptrons?

- Build a multi-layer neural network!



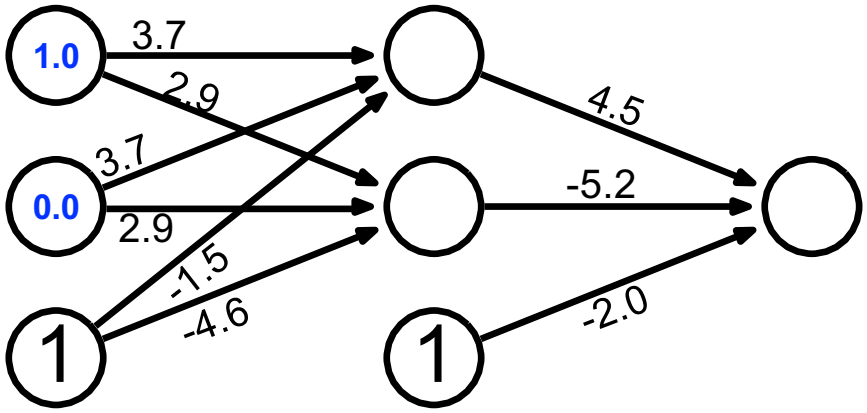
Network with a single hidden layer



Why Add Layers?

- *Each layer is a processing step*
- *Having multiple processing steps allows complex functions*
- *Metaphor: NN and computing circuits*
  - *computer = sequence of Boolean gates*
  - *neural computer = sequence of layers*
- Now we can calculate more complex (non-linear) functions

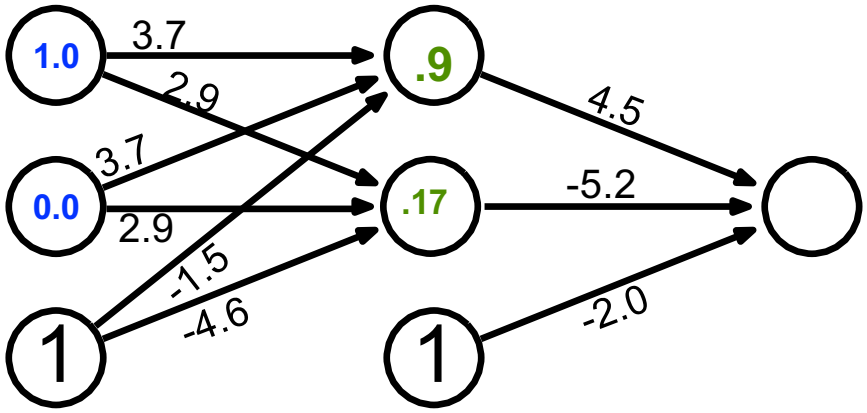
Simple Neural Net with 1 Hidden Layer



$$\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90$$

$$\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17$$

Simple Neural Net with 1 Hidden Layer

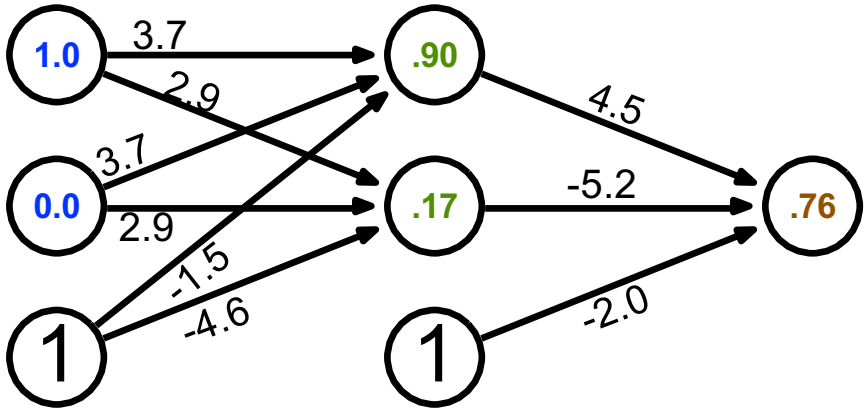


$$\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1 + e^{-2.2}} = 0.90$$

$$\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1 + e^{1.6}} = 0.17$$

*Example from Philipp Koehn, Johns Hopkins Univ

Computed Output

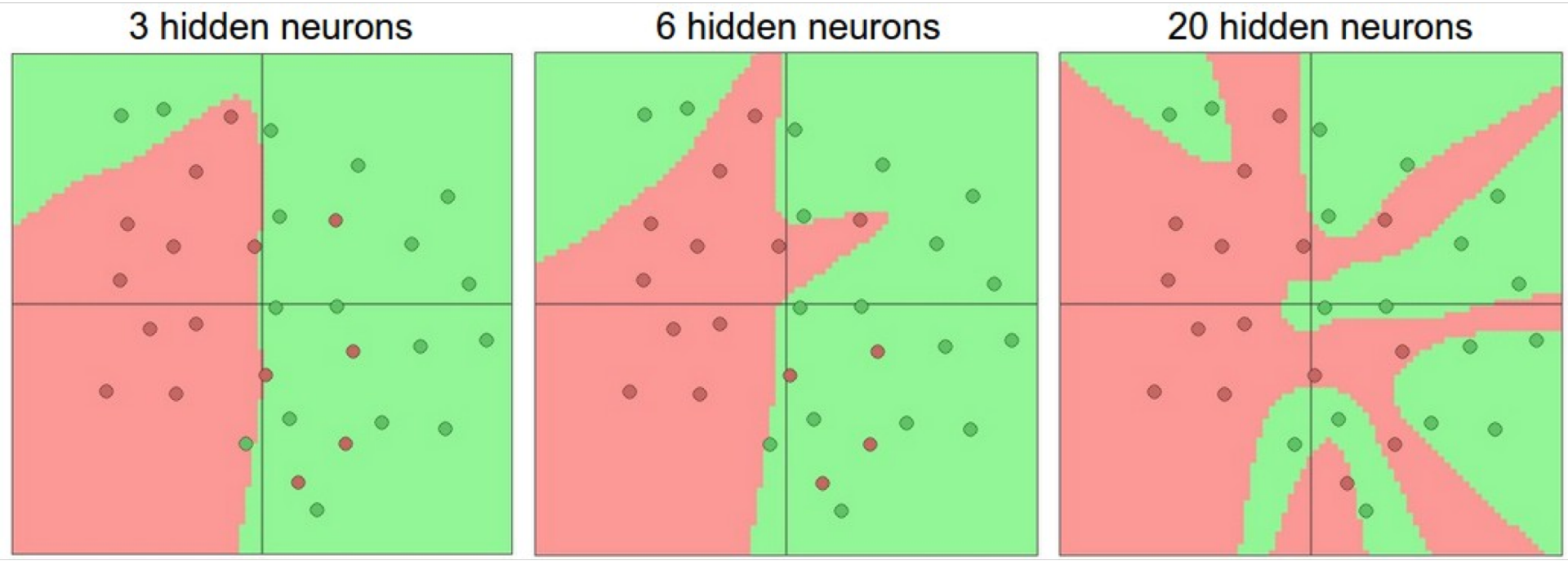


- Output unit computation

$$\text{sigmoid}(.90 \times 4.5 + .17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1 + e^{-1.17}} = 0.76$$

Network with a single hidden layer

- Hidden layer size and *network capacity*:



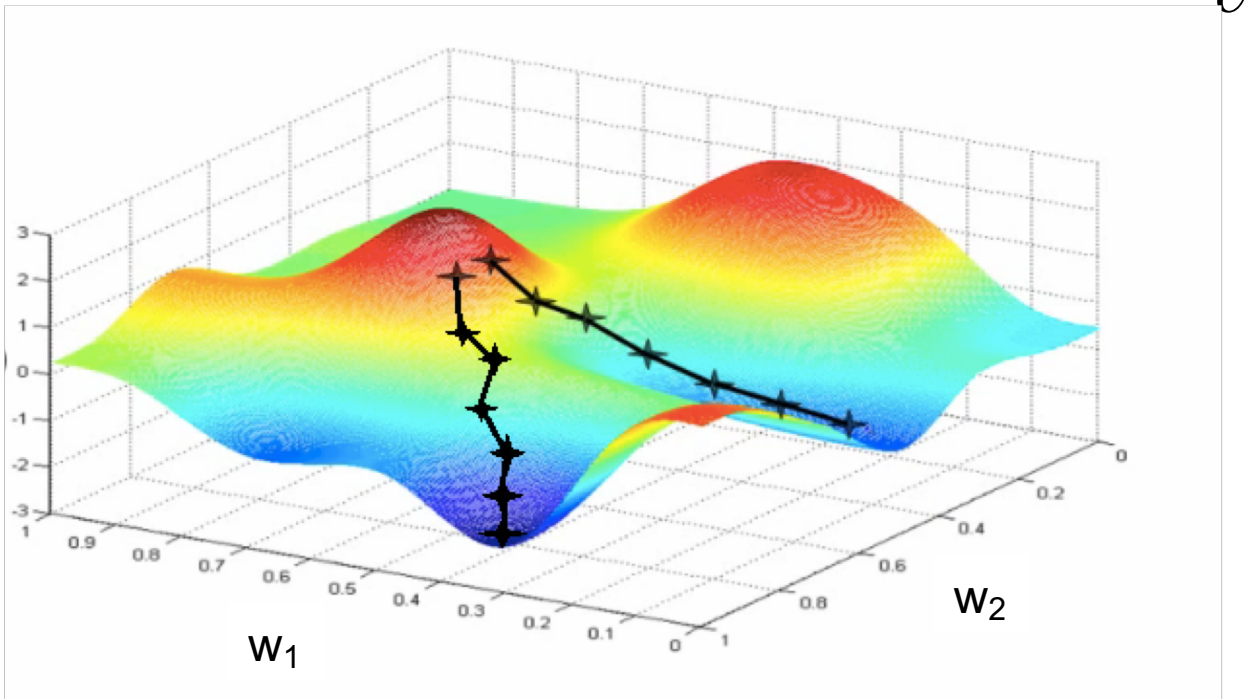
Source: http://cs231n.github.io/neural-networks-1/

Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



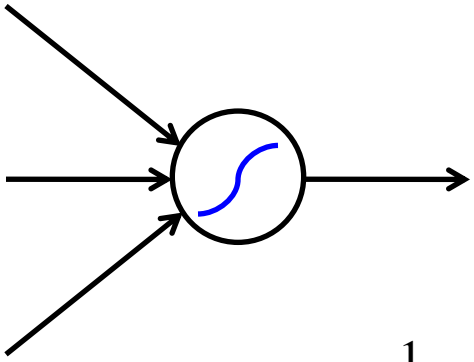
Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

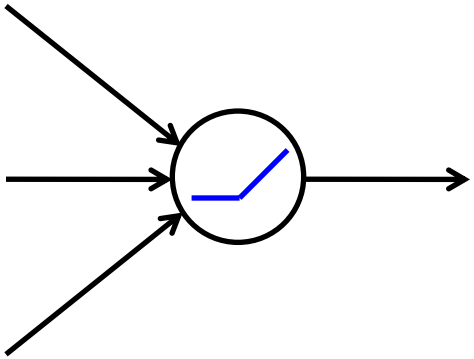
$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- This requires perceptrons with a differentiable nonlinearity



Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$



Rectified linear unit (ReLU): $g(t) = \max(0,t)$

Training of multi-layer networks

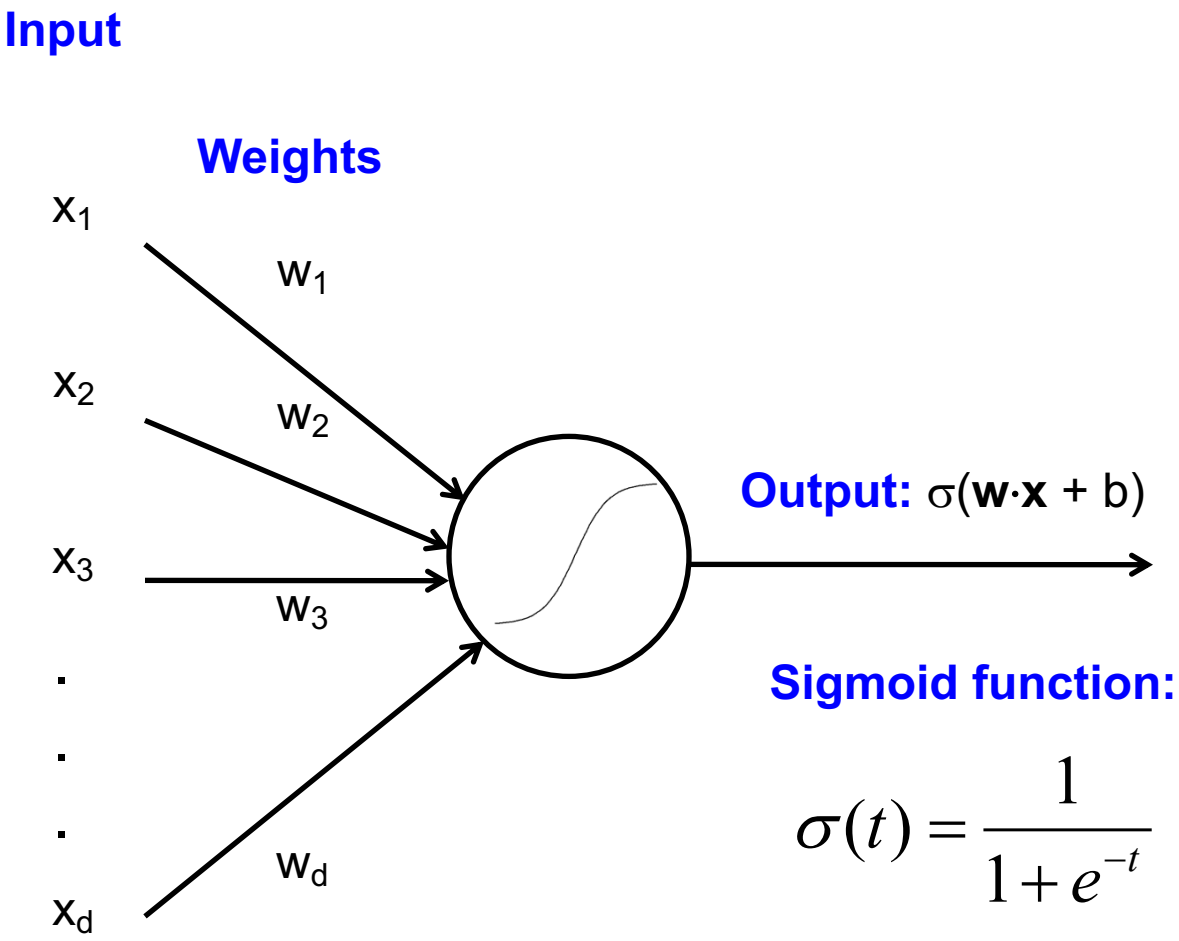
- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Perceptron with sigmoid nonlinearity



Update rule for perceptron w/ sigmoid

- Define total classification error or loss on the training set:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j)\right)^2, \qquad f_{\mathbf{w}}(\mathbf{x}_j) = \sigma(\mathbf{w} \cdot \mathbf{x}_j)$$

- Update weights by *gradient descent*: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right) \sigma'(\mathbf{w} \cdot \mathbf{x}_j) \frac{\partial}{\partial \mathbf{w}} (\mathbf{w} \cdot \mathbf{x}_j) \right]$$

$$= \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right) \sigma(\mathbf{w} \cdot \mathbf{x}_j)(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_j))\mathbf{x}_j \right]$$

- For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - f(\mathbf{x})\right) \sigma(\mathbf{w} \cdot \mathbf{x})(1 - \sigma(\mathbf{w} \cdot \mathbf{x}))\mathbf{x}$$

Update rule for differentiable perceptron

- For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\sigma(\mathbf{w}\cdot\mathbf{x})(1-\sigma(\mathbf{w}\cdot\mathbf{x}))\mathbf{x}$$

- Compare with update rule with non-differentiable perceptron:

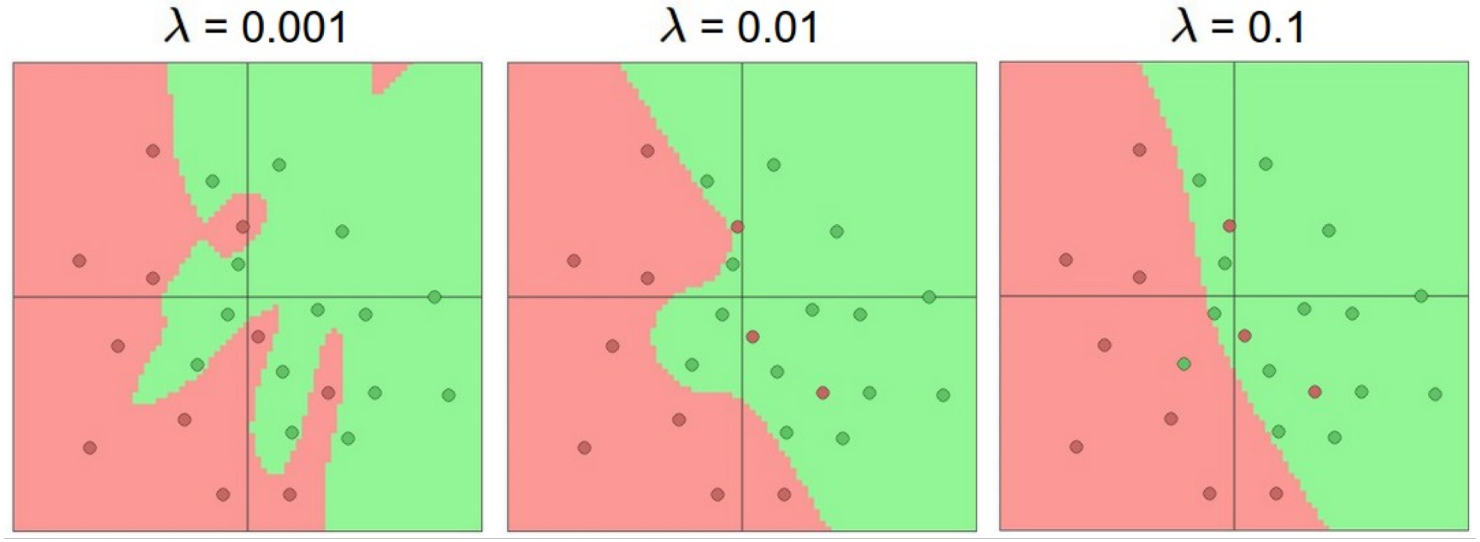
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\,\mathbf{x}$$

Regularization

- It is common to add a penalty on weight magnitudes to the objective function:

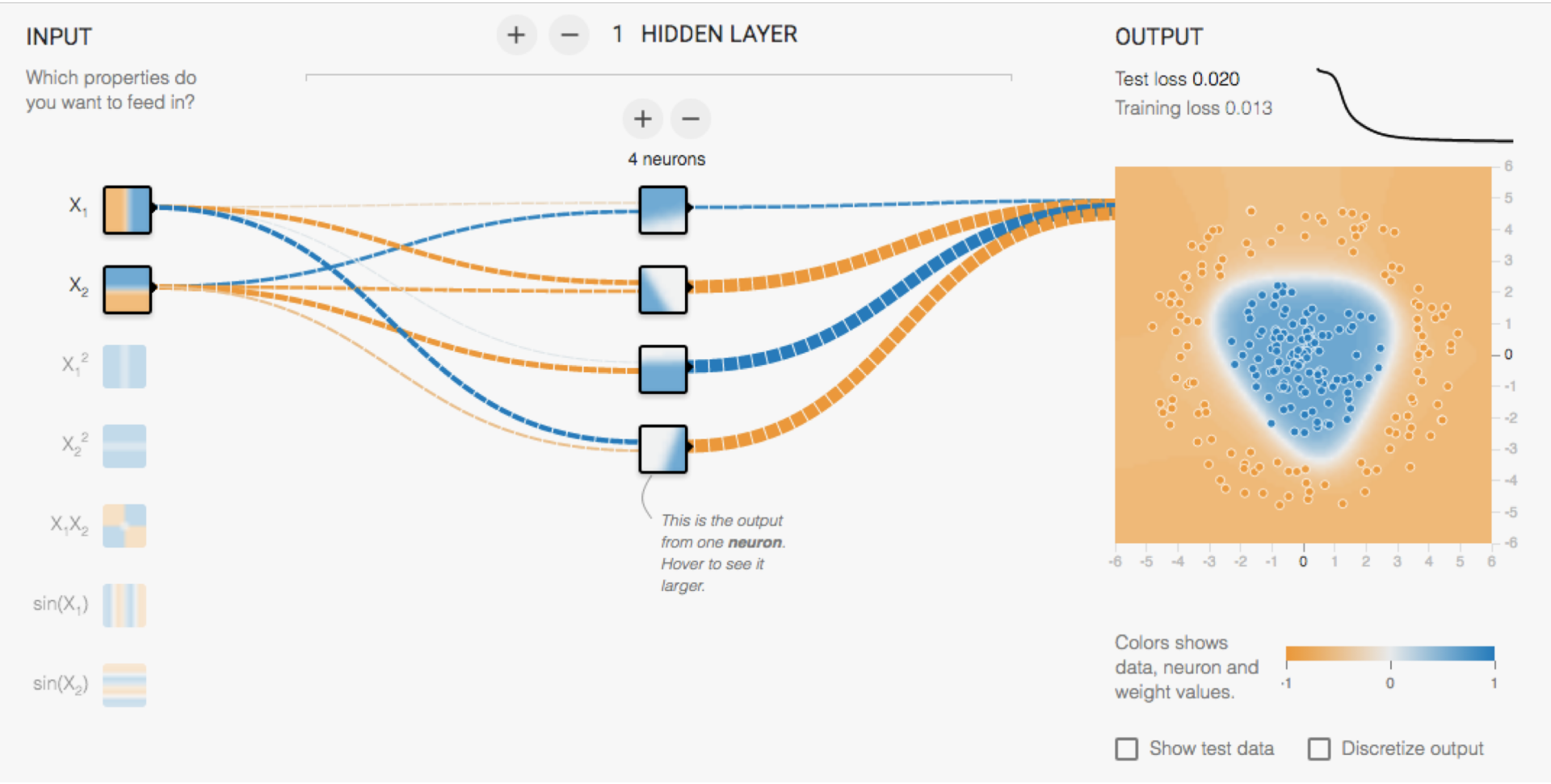
$$E(f) = \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2} \sum_j w_j^2$$

  - This encourages network to use all of its inputs “a little” rather than a few inputs “a lot”



Source: http://cs231n.github.io/neural-networks-1/

Multi-Layer Network Demo

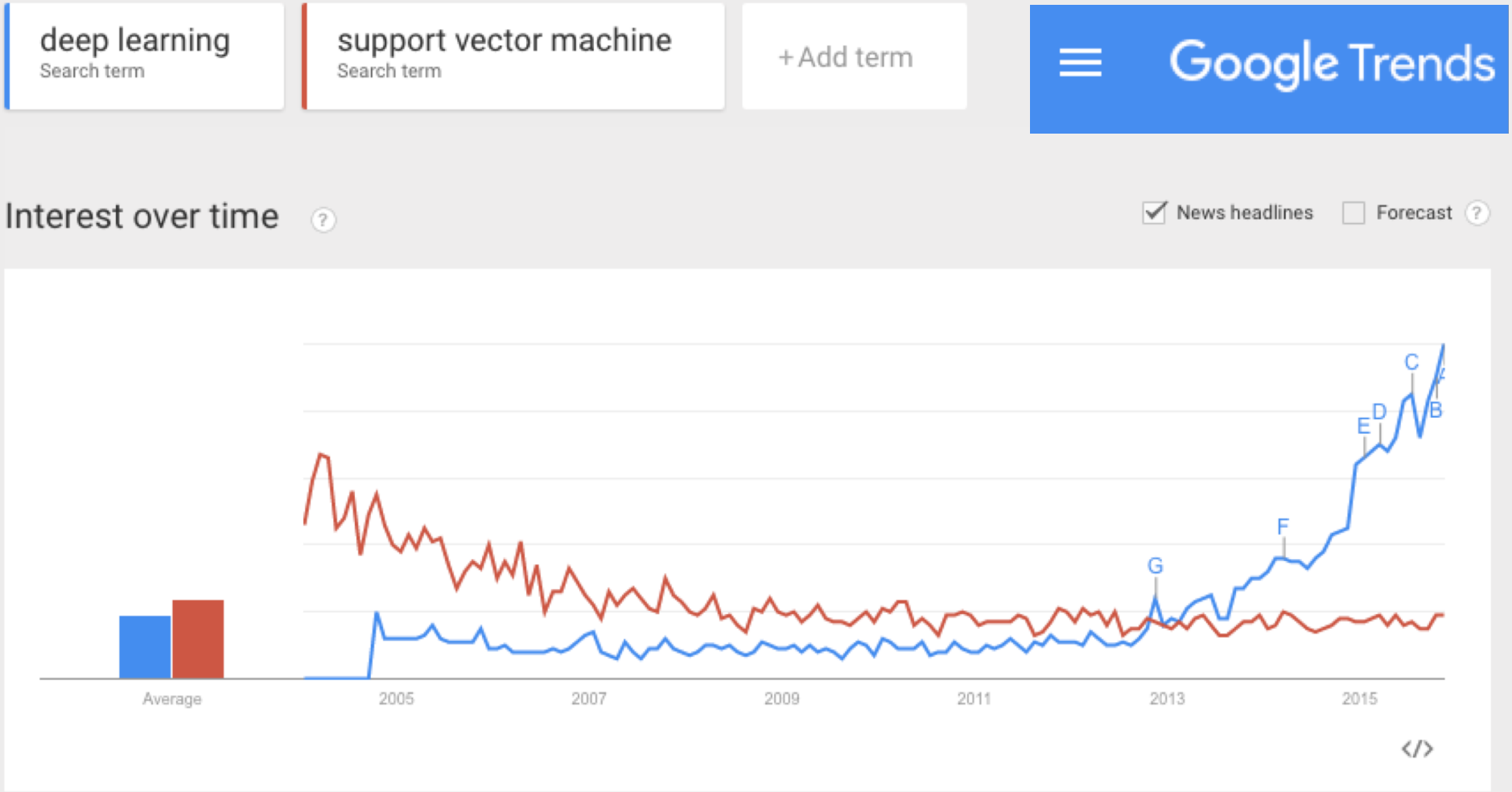


http://playground.tensorflow.org/

Neural networks: Pros and cons

- Pros
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- Cons
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power required to get good performance
  - The space of implementation choices is huge (network architectures, parameters)

Neural networks vs. SVMs
(a.k.a. “deep” vs. “shallow” learning)



Attribution

Slides originally developed by Svetlana Lazebnik based on content from Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 3rd edition. Slight modifications by Stephanie Schwartz.

Example (as noted) from Philipp Koehn at JHU