Artificial Neural Networks

Beginning to understand what the heck is going on in there

Carl Skarbek

Statistics Café

December 5, 2018

Goals for today

- Understanding terminology

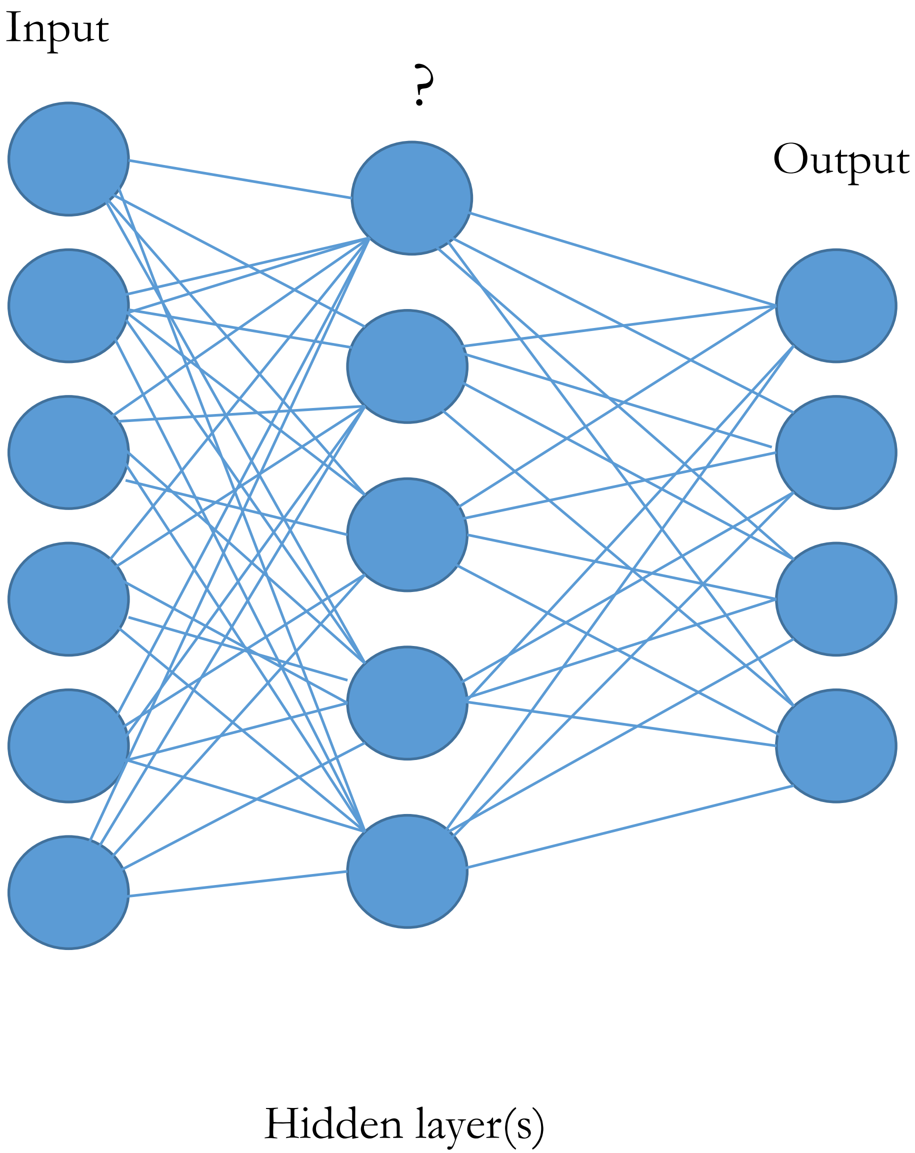
- What *is* an artificial neural network?

- What can we do with an artificial neural network?

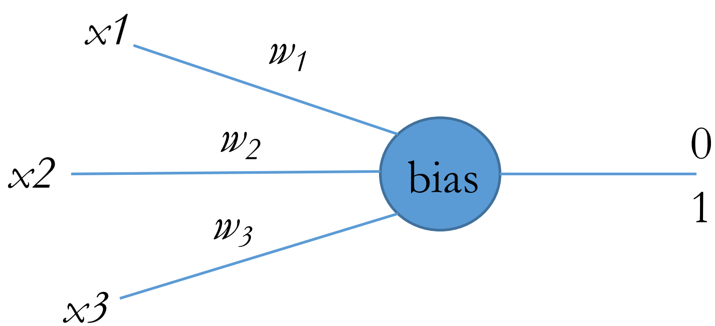
AI and ANNs

- In general, artificial intelligence seeks to make computers superior at doing things that humans are currently better at.

- *Loosely* based on on the networks of neurons and synapses in the human brain
  - Human neural networks still mysterious
  - Scale of neurons/interconnections hard to replicate


Input
?
Output
Hidden layer(s)

Idea of artificial neurons: ‘perceptrons’



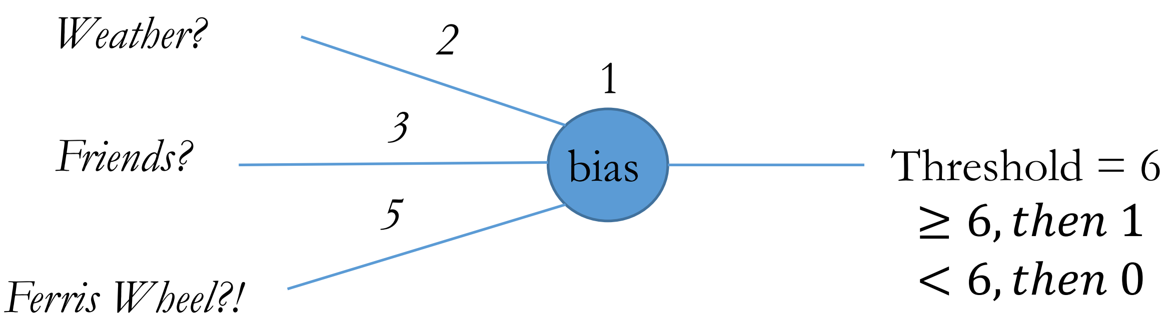
$$0 \text{ if } \sum_j x_j w_j + b \; < threshold$$
$$1 \text{ if } \sum_j x_j w_j + b \; \geq threshold$$

output

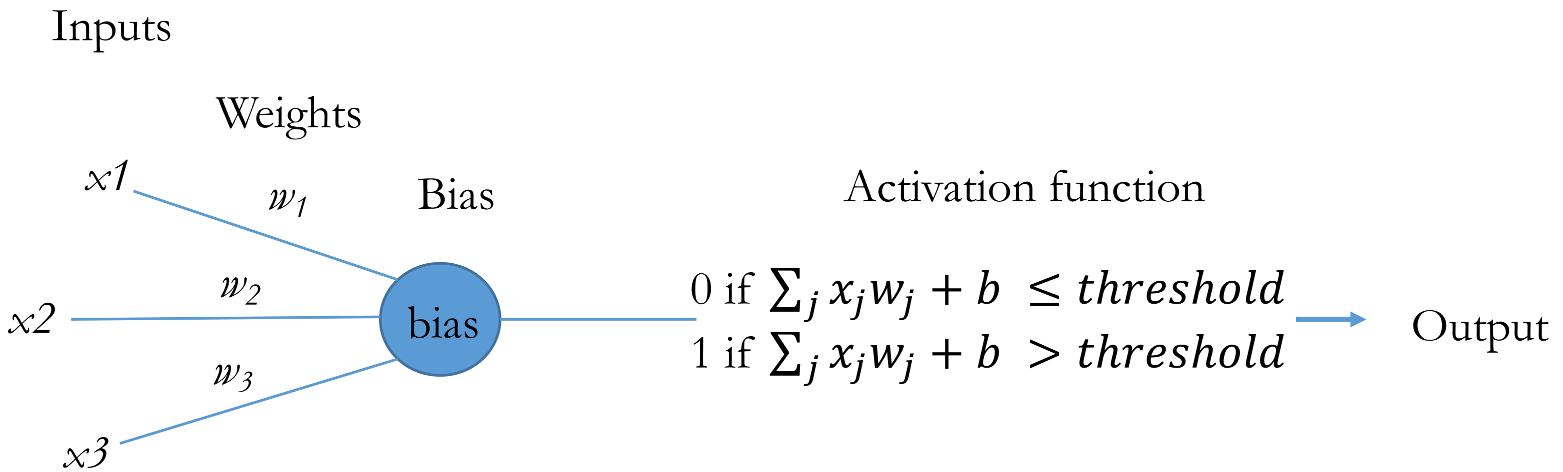
$$x_i w_i + x_{i+1} w_{i+1} \; + x_{i+2} w_{i+2} \ldots$$

Frank Rosenblatt, 1958

To go or not to go?
That is the question…



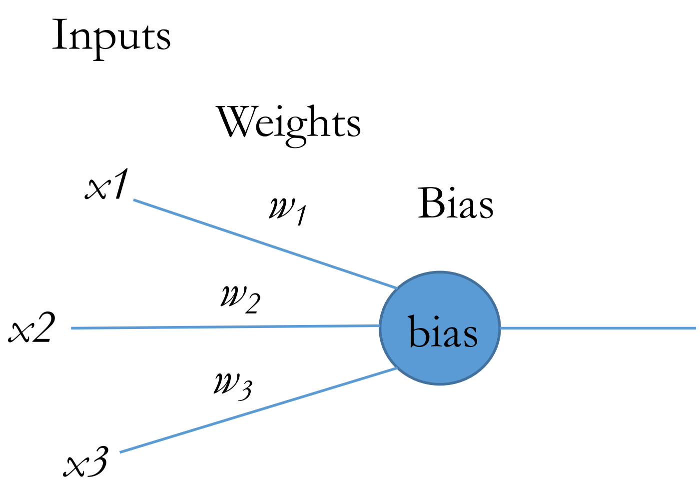
Idea of artificial neurons: 'perceptrons'



$$\begin{cases} 0 \text{ if } \sum_j x_j w_j + b \ \leq threshold \\ 1 \text{ if } \sum_j x_j w_j + b \ > threshold \end{cases}$$

How to adjust weights to get slight change in output?

Sigmoid neurons



Activation function

$\sigma(w \cdot x + b) \rightarrow$ Output $\hat{y}$

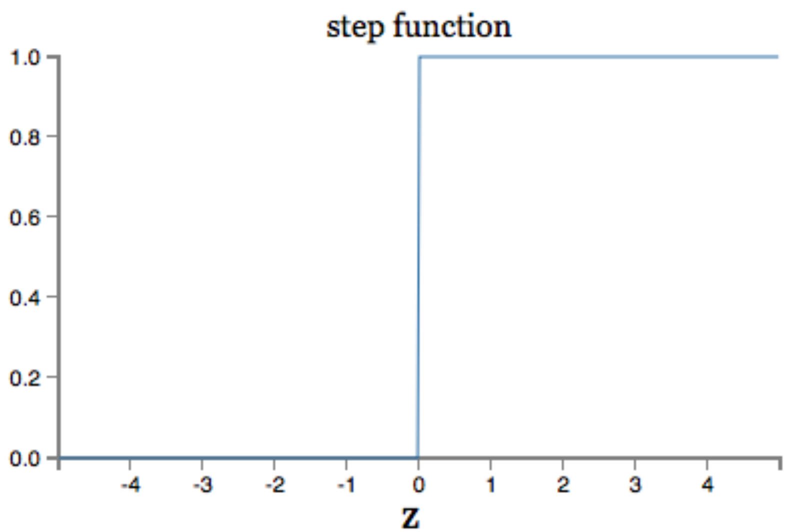
Inputs: any value between 0 and 1
Output: any value between 0 and 1

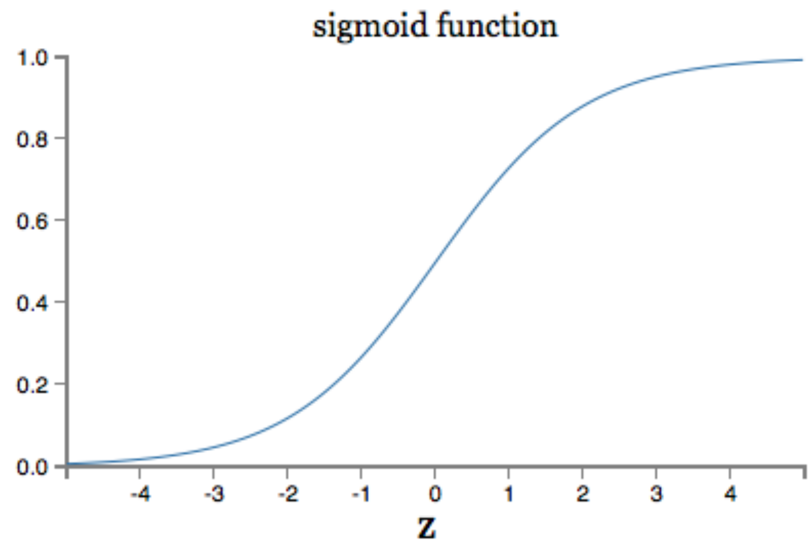
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

$$w = vector\ of\ weights$$
$$x = vector\ of\ inputs$$
$$b = bias$$

Perceptron

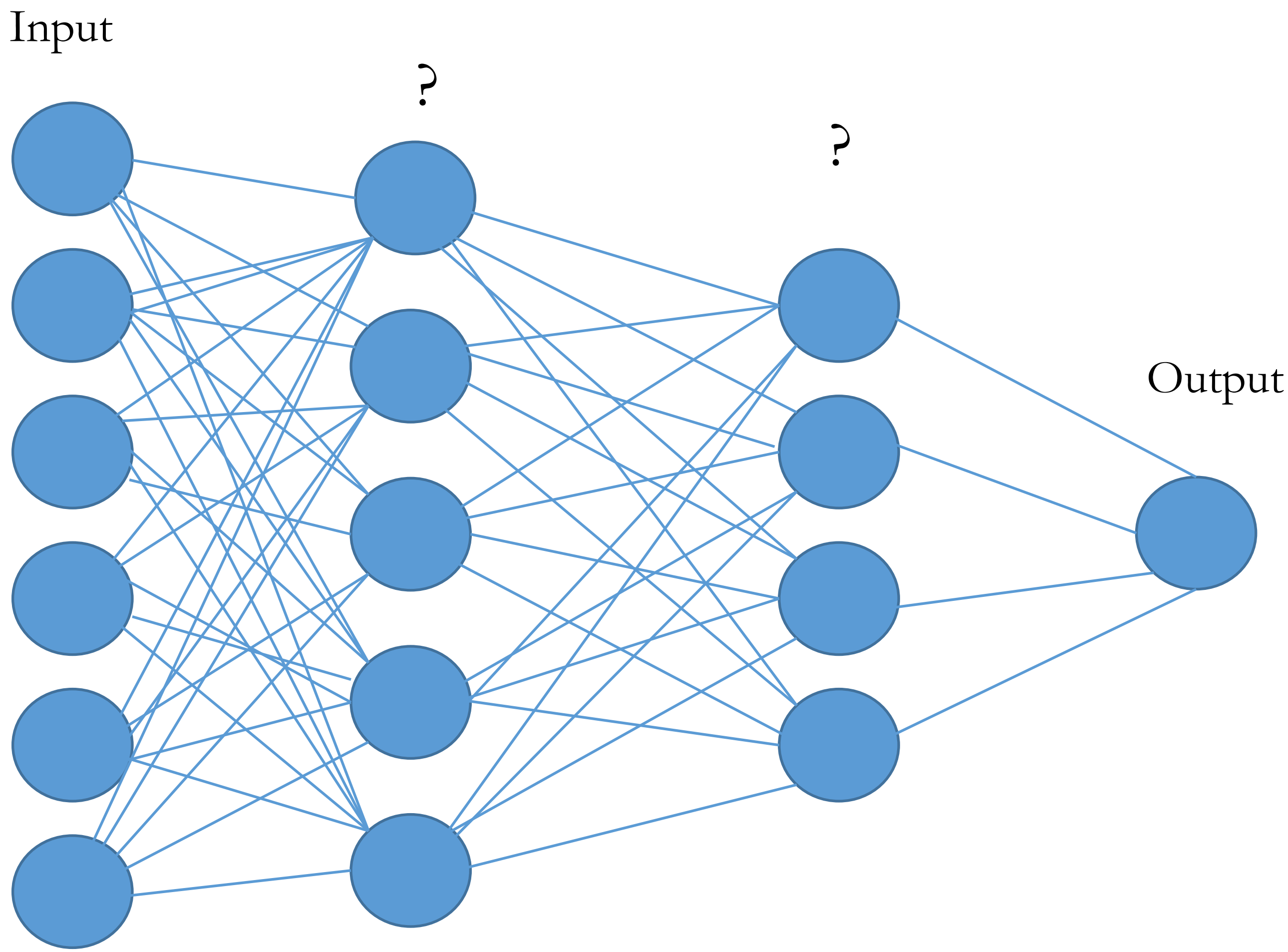
Sigmoid neuron





Not smooth

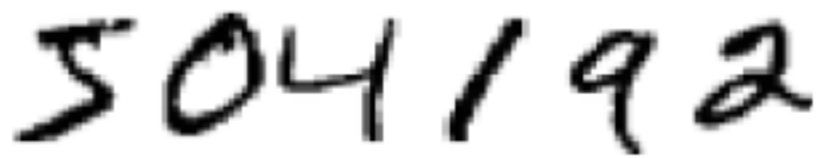
Very smooth indeed



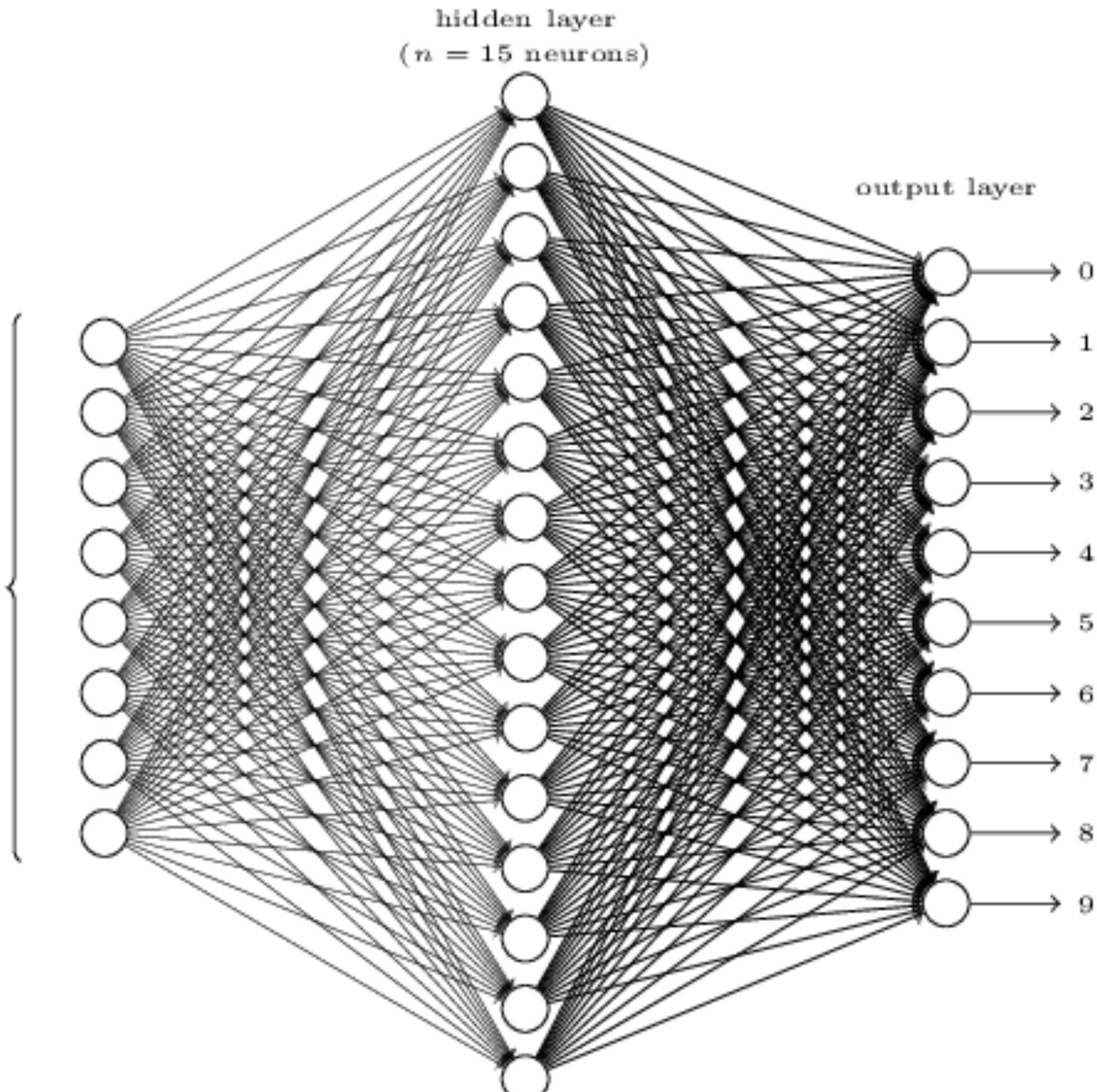
Hidden layer(s)
i.e. neither input nor output layer
How many/how many neurons?

Example: recognizing digits

- We want to classify some hand written digits in order to save them in a database

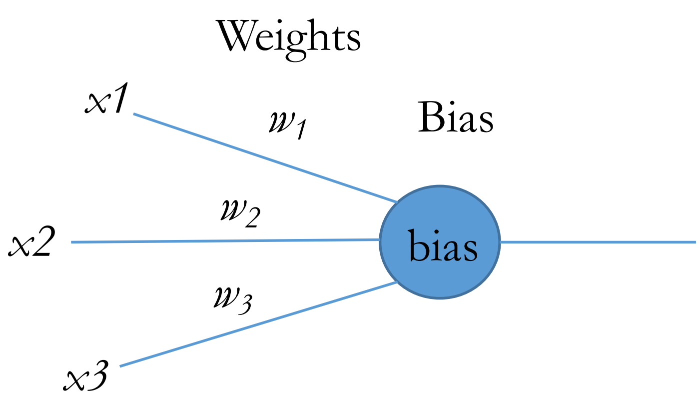


Example: recognizing digits



Nielsen 2015

Forward propagation



Activation function

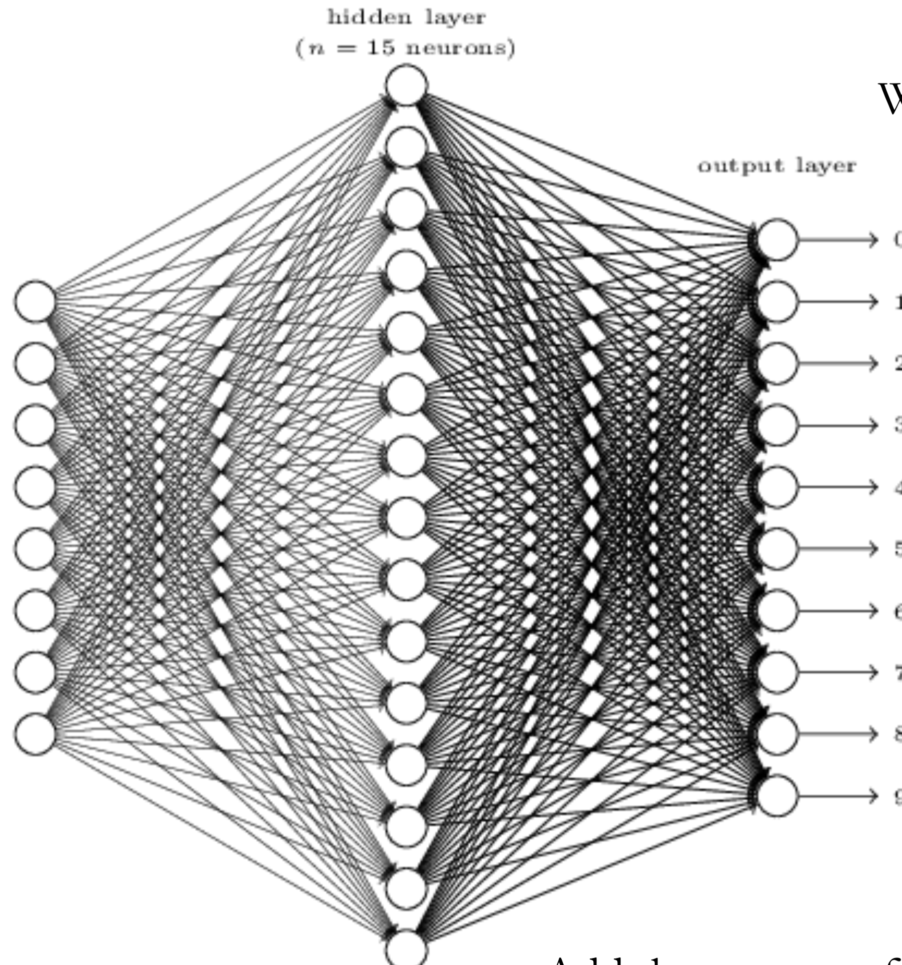
$\sigma(w \cdot x + b)$ → Output

$\hat{y}$

$$C(w,b) \equiv \frac{1}{2n}\sum_{x} \| y(x) - a \|^2$$

n = number of training inputs
a = vector of outputs when x is input
y(x) = expected output

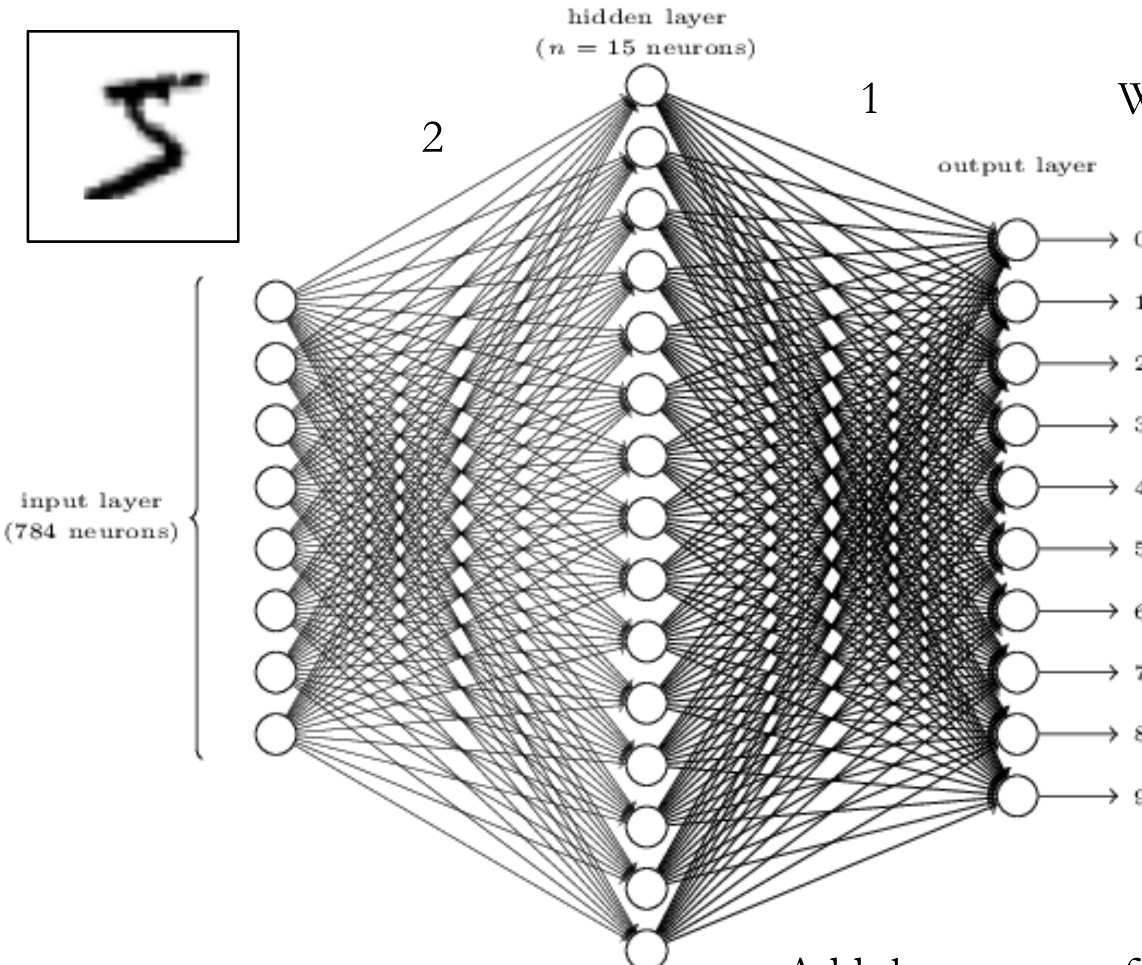
Cost function



	What we got	What we want
0	.38	0.0
1	.06	0.0
2	.11	0.0
3	.38	0.0
4	.21	0.0
5	.59	1.0
6	.22	0.0
7	.31	0.0
8	.69	0.0
9	.44	0.0

Add the squares of the differences and average over all training data to get the ”total cost” of the network

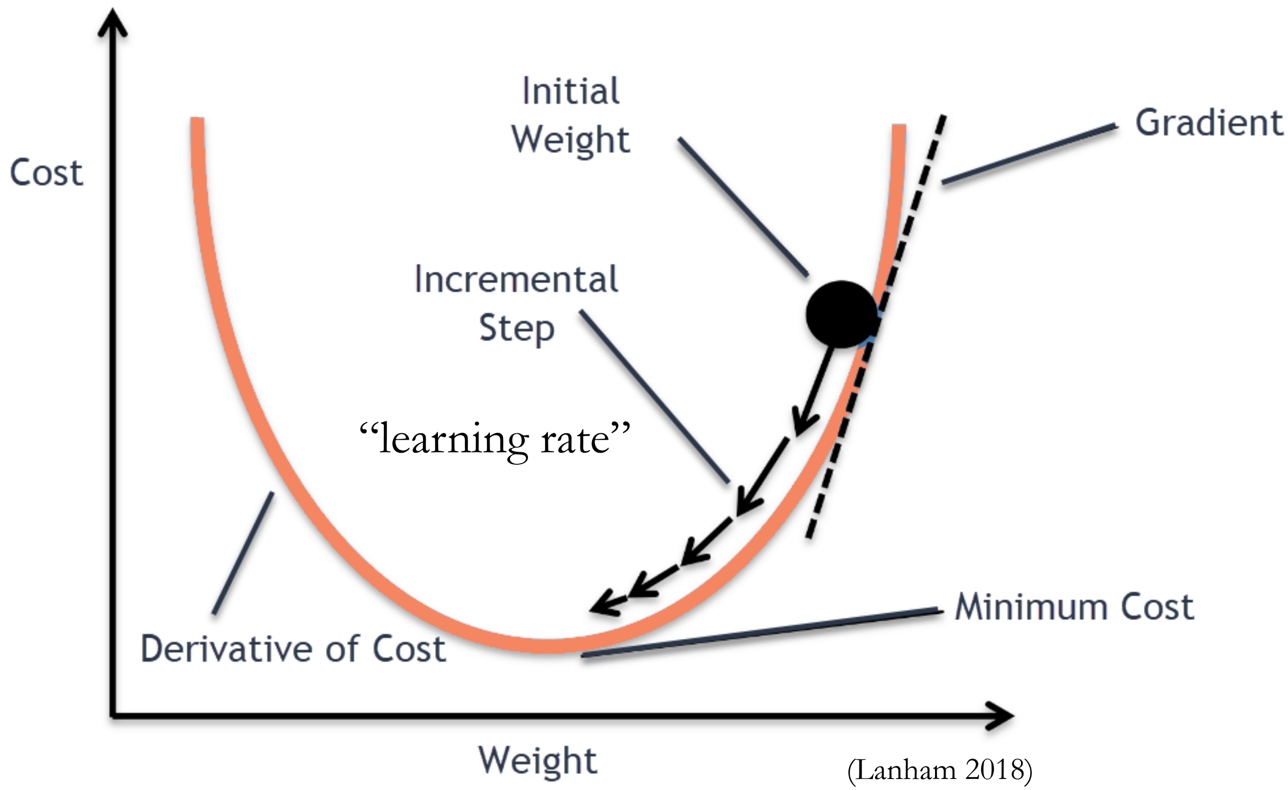
Back(ward) propagation



What we got	What we want
.38	0.0
.06	0.0
.11	0.0
.38	0.0
.21	0.0
.59	1.0
.22	0.0
.31	0.0
.69	0.0
.44	0.0

Add the squares of the differences and average over all training data to get the "total cost" of the network

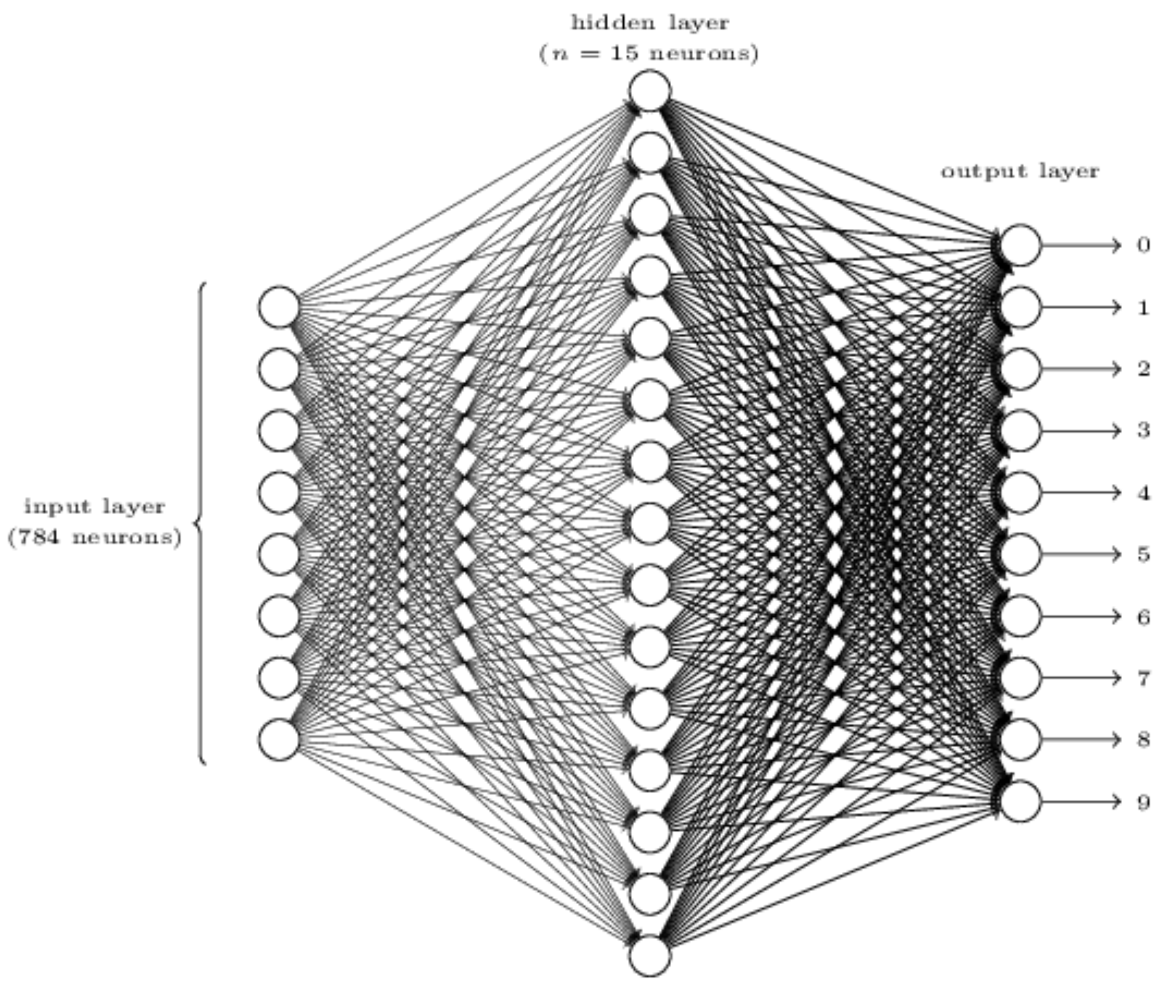
Gradient descent: key to learning



(Lanham 2018)

*Stochastic gradient descent more common, as it is less computationally costly, and in some cases may help avoid landing in local minima.

Check accuracy with test data



Things you should be aware of /open questions

- There are other types of layers than just input, output and hidden:
  - dropout, convolutional, pooling, and recurrent layers.
- Also other types of neurons than the ones shown here
  - Rectified linear unit (ReLU) – popular in DNNs
- All local minima are global minima (Kawaguchi 2016)
  - At least in feed forward DNNs…How does this really work?
- How can we determine the correct number of hidden layers or neurons in our network?

Examples of using ANNs

- Automatic translation
- Spam email filters
- Speech recognition
- Facial recognition
- Coming up with new species names…

Terminology

- Artificial neurons -> nodes that hold numbers designating the "activation" level of the neuron
- Connections -> akin to synapses in the human brain, transmit signals from one neuron to the other
- Weights -> each connection carries a weight that shows how much influence the input has on the output
- Bias -> term added to weighted sum of inputs for shifting the activation function
- Hidden layers -> Layers of neurons that are neither input nor output
- Training data -> input data with known outcome used to train and fine tune the neural network
- Cost Function -> function used to find the difference between output result and training data. Used to adjust weights/biases
- Gradient Descent -> method of minimizing the cost function by slowly descending to local minima.

Sources

- Michael A. Nielsen. “Neural Networks and Deep Learning (Determination Press, 2015)
- Kawaguchi, Kenji. "Deep learning without poor local minima." In *Advances in Neural Information Processing Systems*, pp. 586-594. 2016.
- Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5, no. 2 (1994): 157-166.
- Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65, no. 6 (1958): 386.