Tensorflow deep learning workshop:

Multi-layer perceptron (MLP)

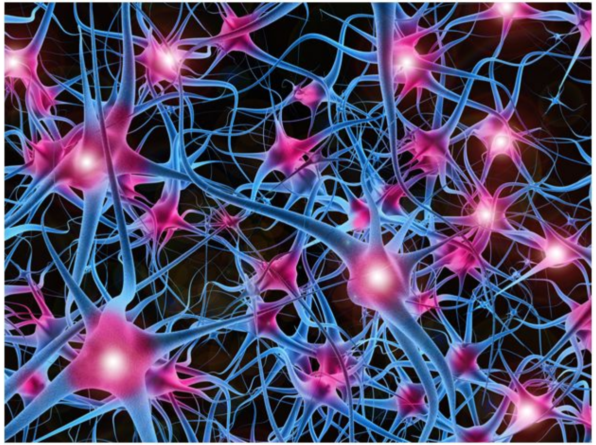
Objectives

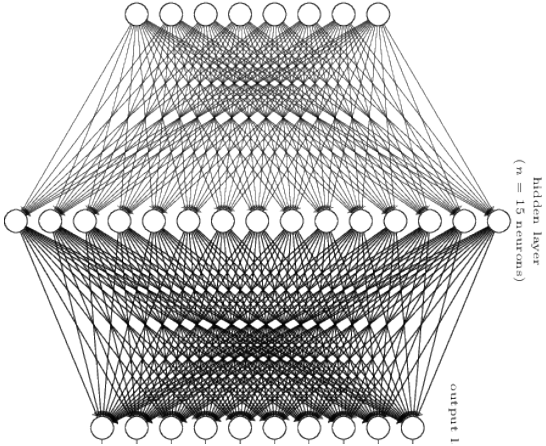
Objectives of this lecture:

- Neural net history
- “Vanilla” neural networks: multilayer perceptrons
  - Parts of a neuron
  - Feed-forward
  - Backpropagation and gradient descent
- Hyperparameters and Training
- References

Neural Net motivation

Our brain: the ultimate parallel computer





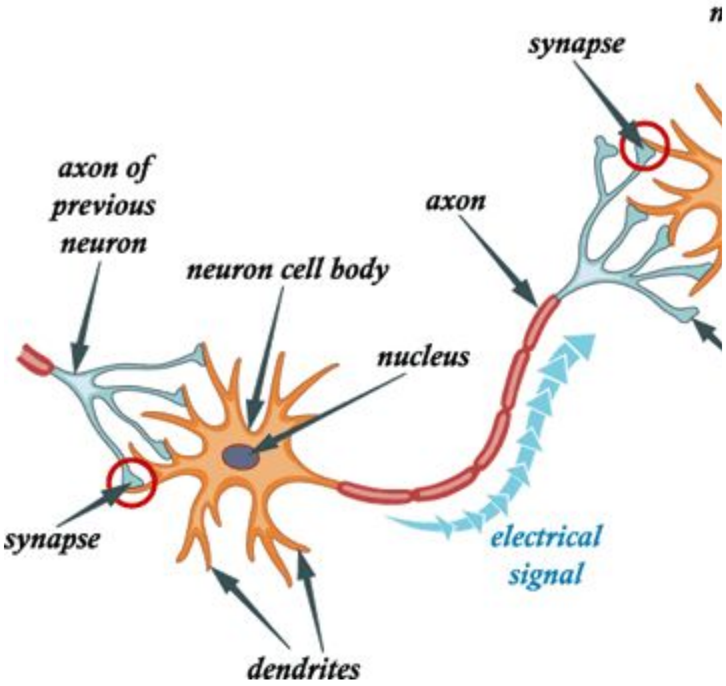
Number of parameters of biggest artificial neural net:	160,000,000,000 ([Digital Reasoning, 2015])
The number of neurons in your brain:	86,000,000,000
The number of synapses in your brain:	1,000,000,000,000,000
The average number of synapses per neuron:	10,000

If we want a computer to be able to perform the same tasks as our brain, we should look to how the brain works for inspiration.

Neural Net history

Biomimicry

McCulloch-Pitts
first neuron model in 1943



BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

I. *Introduction*

Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse.

Improvements to MCP neuron

- 1949, Donald Hebb, neuropsychologist

  "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

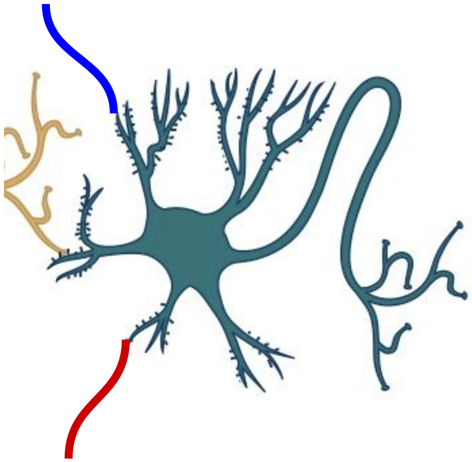
- 1957, Frank Rosenblatt invents the Perceptron
  - Initializes the weights* to random values (e.g. -1.0 to 1.0)
  - Weights change during supervised learning according to the delta rule, ~*(yi - yp)*. After a <u>certain number of training passes through the whole training set</u> (a.k.a. the number of *epochs*) stop changing the weights.
  - Implements a learning rate that affects how quickly weights can change.
  - Adds a bias to the activation function that shifts the location of the activation threshold and allows this location to be a learned quantity (by multiplying it by a weight).

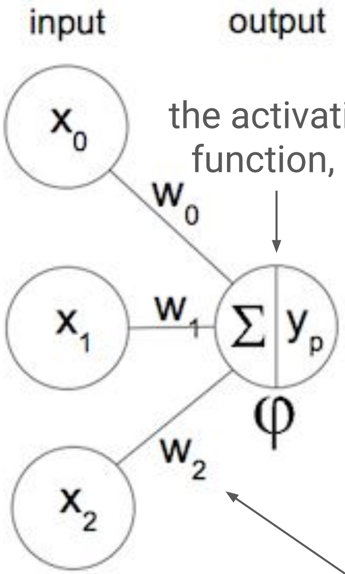
* weights are the quantified strength of a connection between neurons

How to pronounce epoch time?

Elizabeth Goldberg, B.A. in English, I know at least eight words. I've always **pronounced** it EE-pok. Merriam-Webster's website provides three possible pronunciations: ˈe-pək (EH-pik), ˈe-ˌpäk (EH-pok), and ˈē-ˌpäk (EE-pok). The first way is probably the most correct, considering it's Latin and Greek origins.

Perceptron





the activation function, φ

this neuron's activation based on the inputs, weights, and φ

neighboring neurons' activations

the strength (or weight) of the connection between each neighboring axon and this neuron. After training, the weights are fixed.

Inputs, X **Desired output, y**

X			y
x_0	x_1	x_2	y_t
1	0	0	0
1	1	1	1
1	0	1	0
1	1	0	1

Set-back: the XOR affair

- *Perceptrons, an introduction to computational geometry* (book by Minsky and Papert 1969)
  - From Wikipedia: *"critics of the book state that the authors imply that, since a single artificial neuron is incapable of implementing some functions such as the [XOR](XOR) logical function, larger networks also have similar limitations, and therefore should be dropped."*

AND

AND (A, B → Q)		A	
		0	1
B	0	0	0
	1	0	1

OR

OR (A, B → Q)		A	
		0	1
B	0	0	1
	1	1	1

XOR

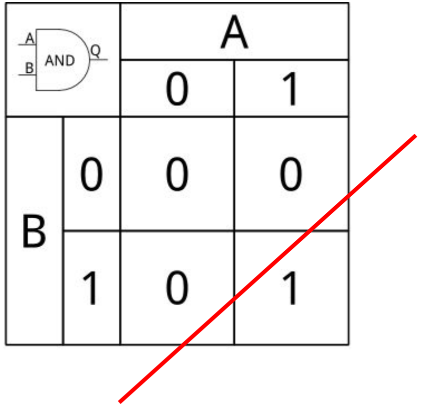
XOR (A, B → Q)		A	
		0	1
B	0	0	1
	1	1	0

?

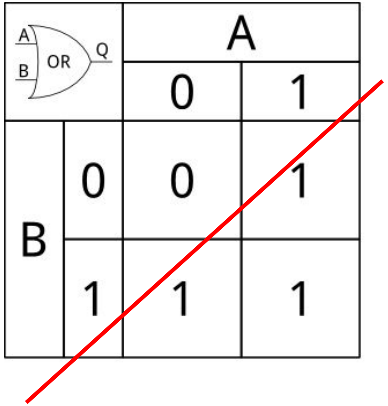
XOR affair solution: go deeper (multi-layer)

- *Perceptrons, an introduction to computational geometry* (book by Minsky and Papert 1969)
  - From Wikipedia: *"critics of the book state that the authors imply that, since a single artificial neuron is incapable of implementing some functions such as the [XOR](XOR) logical function, larger networks also have similar limitations, and therefore should be dropped."*
  - Clarification: single layer perceptron networks are limited to being linear classifiers. Not true of deeper MLP networks.

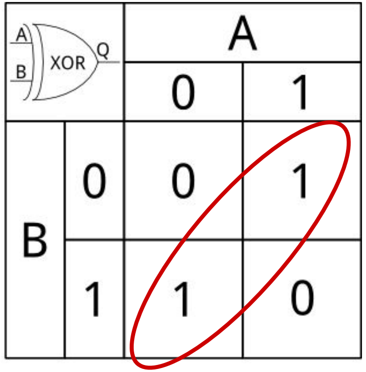
AND



OR

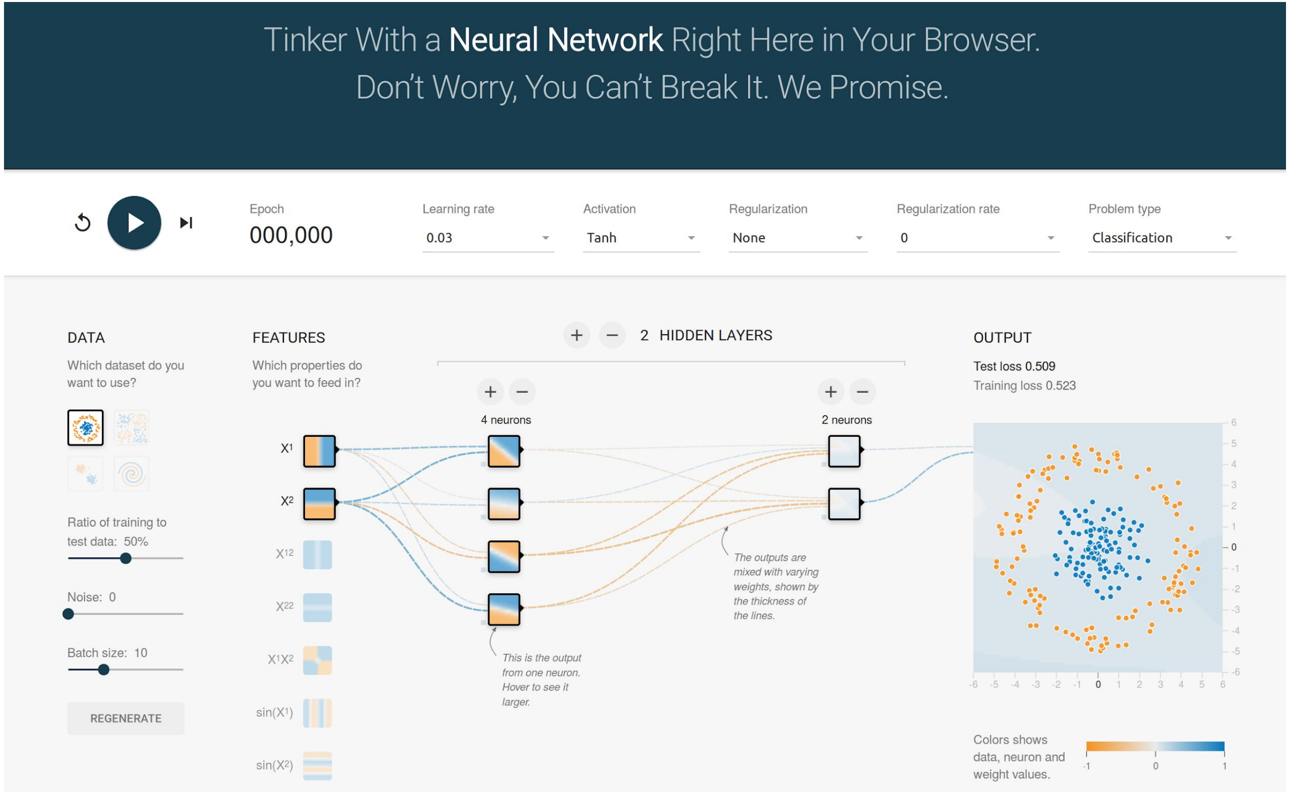


XOR



See for yourself!

https://playground.tensorflow.org/



Breakthrough for multi-layer networks

LEARNING INTERNAL REPRESENTATIONS BY ERROR PROPAGATION

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams

September 1985

ICS Report 8506

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM Mar 85 TO Sept 85	September 1985	34

16. SUPPLEMENTARY NOTATION To be published in J. L. McClelland, D. E. Rumelhart, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol 1. Foundations*. Cambridge, MA: Bradford Books/MIT Press.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
learning; networks; perceptrons; adaptive systems; learning machines; back propagation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper presents a generalization of the perceptron learning procedure for learning the correct sets of connections for arbitrary networks. The rule, called the generalized delta rule, is a simple scheme for implementing a gradient descent method for finding weights that minimize the sum squared error of the system's performance. The major theoretical contribution of the work is the procedure called error propagation, whereby the gradient can be determined by individual units of the network based only on locally available information. The major empirical contribution of the work is to show that the problem of local minima is not serious in this application of gradient descent.

[link to paper]

Back propagation

LEARNING INTERNAL REPRESENTATIONS
BY ERROR PROPAGATION

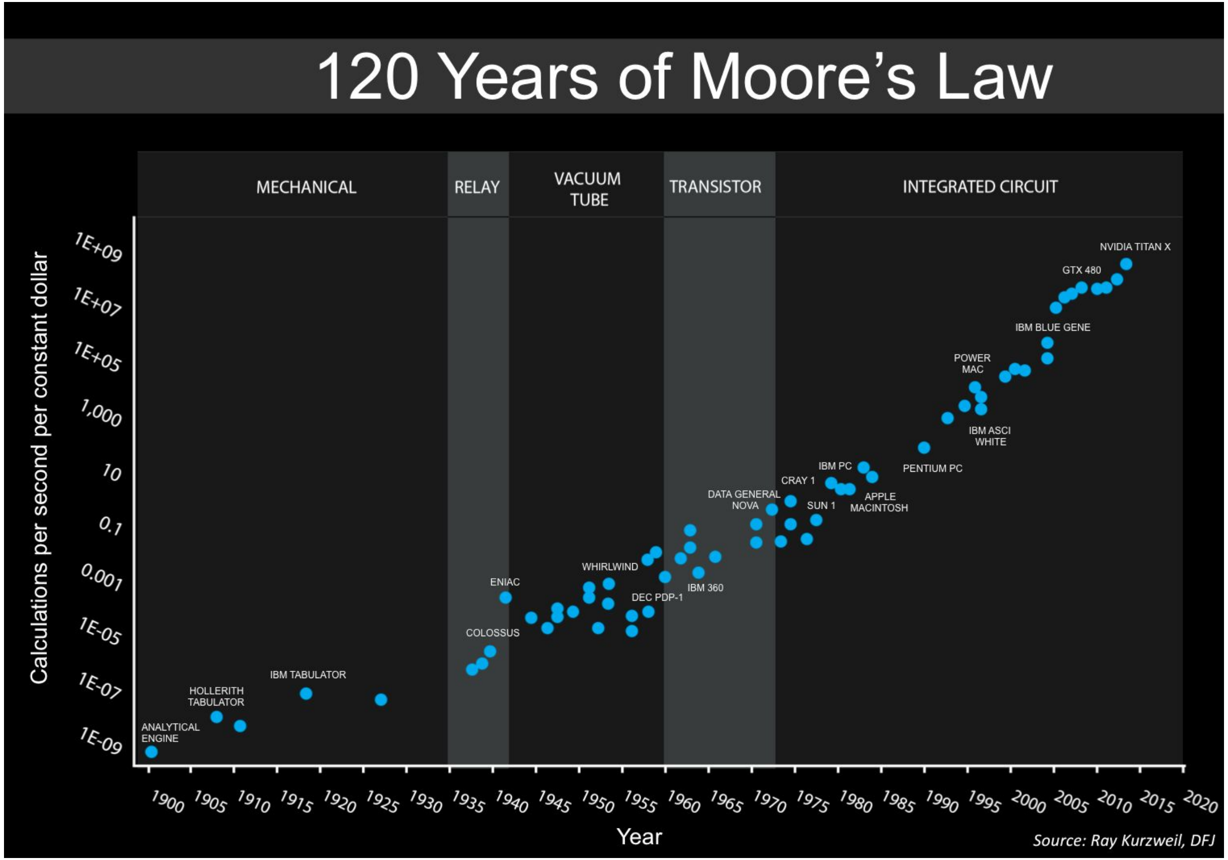
David E. Rumelhart, Geoffrey E. Hinton,
and Ronald J. Williams

September 1985

ICS Report 8506

This paper presents a generalization of the perceptron learning procedure for learning the correct sets of connections for arbitrary networks. The rule, called the generalized delta rule, is a simple scheme for implementing a gradient descent method for finding weights that minimize the sum squared error of the system's performance. The major theoretical contribution of the work is the procedure called error propagation, whereby the gradient can be determined by individual units of the network based only on locally available information. The major empirical contribution of the work is to show

Breakthrough: compute power



Steve Jurvetson, Moore's Law over 120 Years, CC BY 2.0

SPECIFICATIONS

GPU Architecture	**NVIDIA Pascal**
NVIDIA CUDA® Cores	**3584**
Double-Precision Performance	**4.7 TeraFLOPS**
Single-Precision Performance	**9.3 TeraFLOPS**
Half-Precision Performance	**18.7 TeraFLOPS**
GPU Memory	**16GB CoWoS HBM2 at 732 GB/s or 12GB CoWoS HBM2 at 549 GB/s**
System Interface	**PCIe Gen3**
Max Power Consumption	**250 W**
ECC	**Yes**
Thermal Solution	**Passive**
Form Factor	**PCIe Full Height/Length**
Compute APIs	**CUDA, DirectCompute, OpenCL™, OpenACC**

Signs of the current revolution

2009: Microsoft researcher Li Deng invited neural nets pioneer Geoffrey Hinton to visit. Impressed with his research, Deng's group experimented with neural nets for speech recognition. "We were achieving more than 30% improvements in accuracy with the very first prototypes."

2011 & 2012: Microsoft and Google introduced deep-learning technology into their commercial speech-recognition products.

2012: At a workshop in Florence, Italy, the founder of the annual ImageNet computer-vision contest announced that two of Hinton's students had identified objects with almost twice the accuracy of the nearest competitor. "It was a spectacular result," recounts Hinton, "and convinced lots and lots of people who had been very skeptical before."

But beware: an A.I. Winter may yet be coming. Hype precedes it.

http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

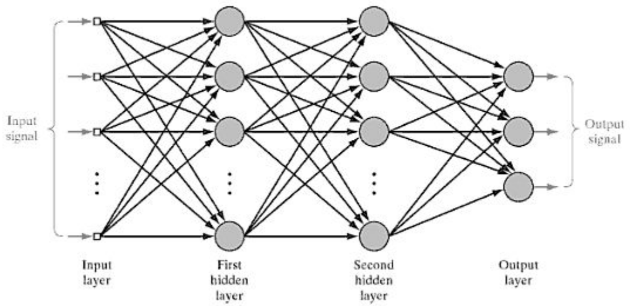
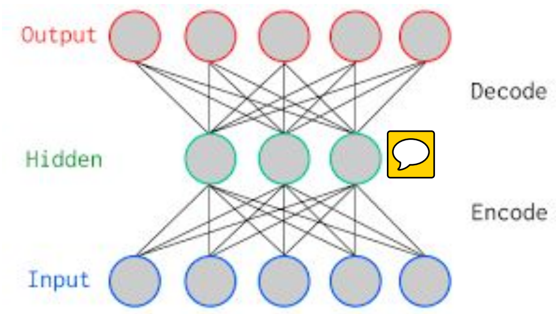
(Artificial) Neural networks

Paraphrased from Wikipedia:

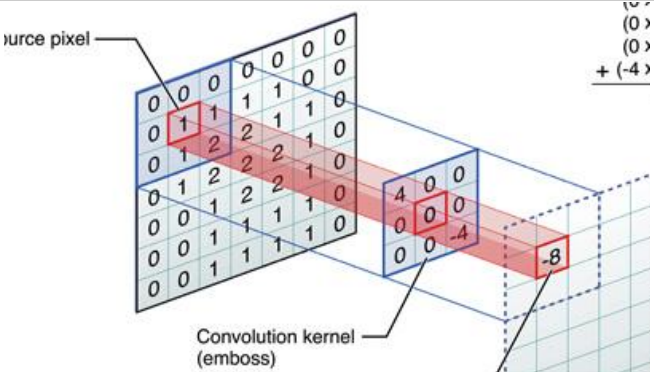
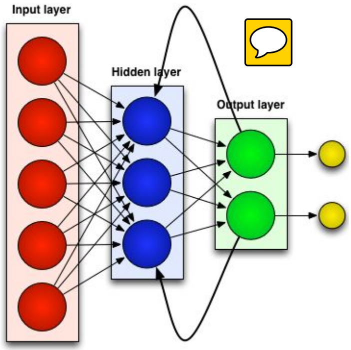
- *Artificial neural networks (ANNs) are a family of models inspired by biological neural networks*
- *Systems of interconnected 'neurons' which exchange messages…*
- *The connections have numeric weights that can be tuned based on experience, making neural nets … capable of learning.*

"Like other machine learning methods – systems that learn from data – neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are hard to solve using ordinary rule-based programming."

ANNs we will talk about in the workshop

ANN	Description	Picture of architecture
Multilayer perceptron (MLP)	Standard algorithm for supervised learning. Used for pattern, speech, image recognition.	
Autoencoder	Used for unsupervised learning of efficient codings, usually for dimensionality reduction, though recently to make generative models.	

ANNs we will talk about in the workshop

ANN	Description	Picture of architecture
Convolutional neural network	Node connections are inspired by visual cortex. Uses kernels to aggregate/transform information in network. Used for image, video recognition, natural lang. proc..	
Recurrent neural network	Connections between nodes can be cyclical, which gives the network memory. Used for sequences: handwriting, speech recognition, time series.	

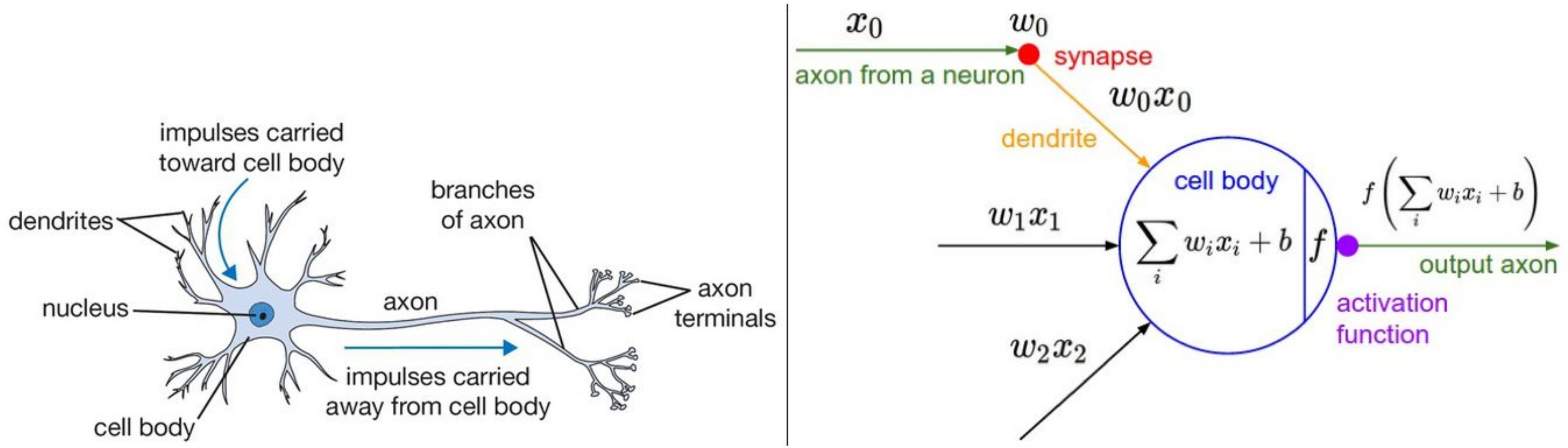
“Vanilla” neural network:

multilayer perceptron

The parts

- Computation unit: perceptron (neuron)
    - Input, weights, summation, activation function, prediction
- Use a single neuron to explain:
    - Feed-forward
    - Backpropagation (get gradients)
    - Gradient descent and its flavors (stochastic, batch, minibatch)
- Gradient descent solution optimizers
- Multi-layer network
- MLP code-it-from-scratch pseudocode

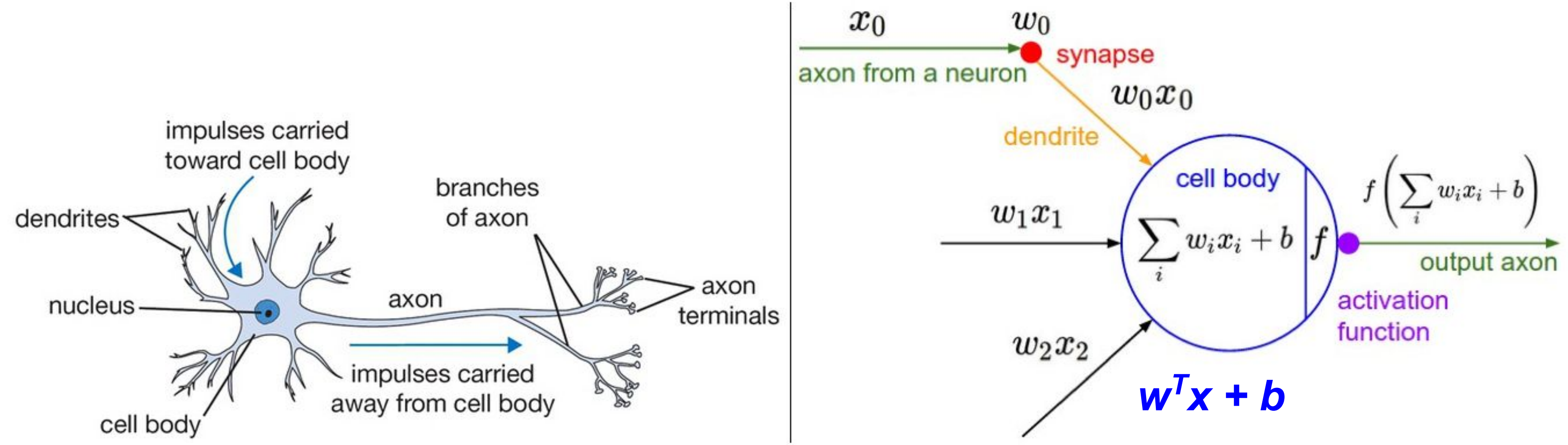
A perceptron (neuron, neurode, node, unit)



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

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

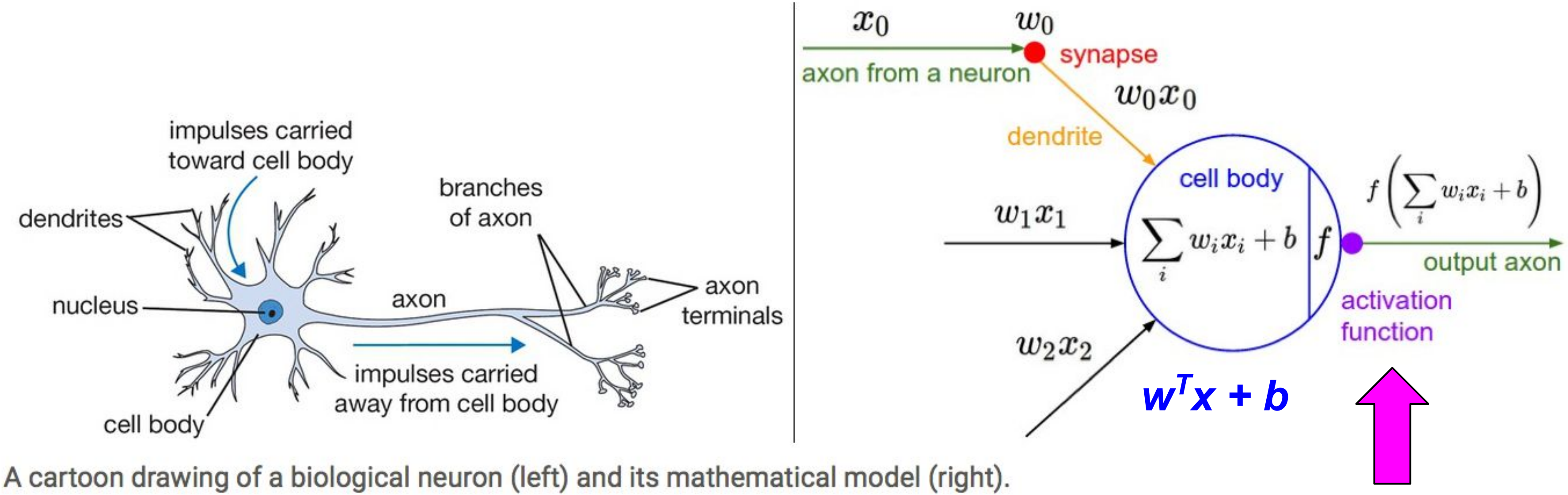
A perceptron (neuron, neurode, node, unit)



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

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

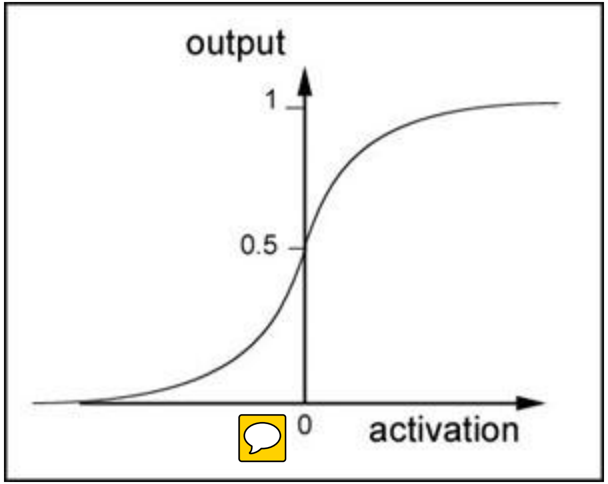
A perceptron (neuron, neurode, node, unit)



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

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

Sigmoid activation function, and its derivative



$$y = \frac{1}{1+e^{-x}}$$

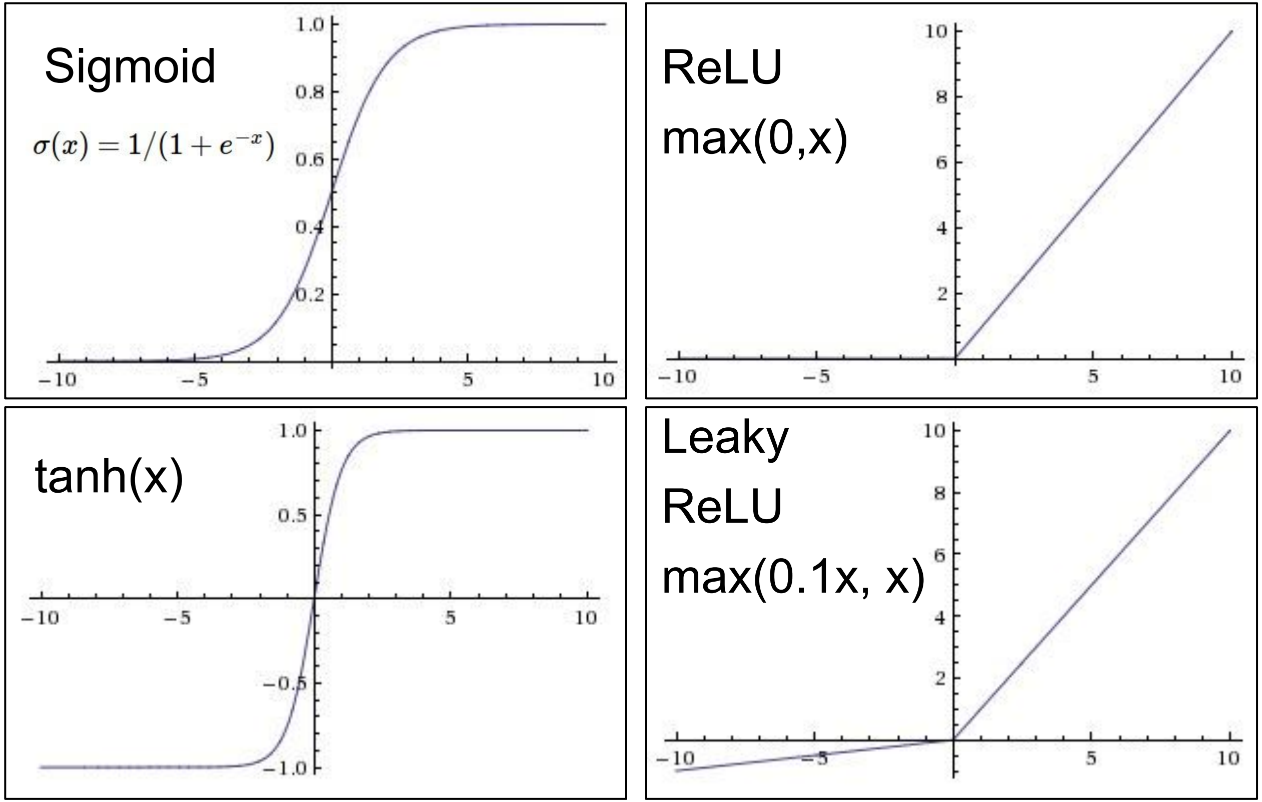
$$quotient\ rule:$$

$$derivate\left(\frac{f}{g}\right) = \left(\frac{f'g - g'f}{g^2}\right)$$

$$\frac{dy}{dx} = \frac{0(1+e^{-x}) - (-1 * e^{-x})}{(1+e^{-x})^2}$$

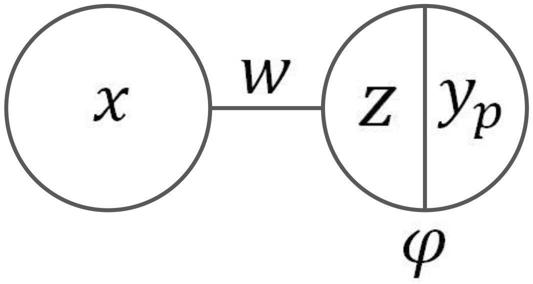
$$\frac{dy}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = (1-y)y$$

Activation functions - a subset



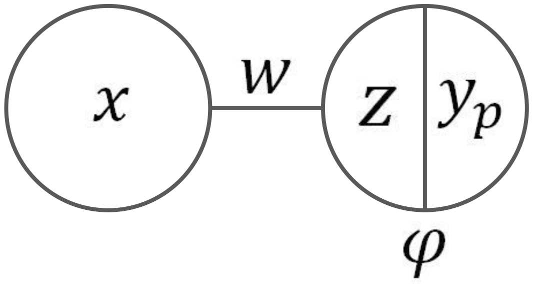
Your choice of an activation function for a given layer of your network should be informed by literature and your experience.

Computations - feed forward (treat like scalars)



$$z = w\,x$$
$$y_p = \varphi(z)$$

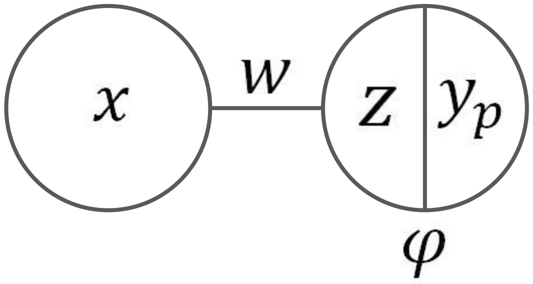
Computations - feed forward (treat like scalars)



$$z = w\,x$$
$$y_p = \varphi(z)$$

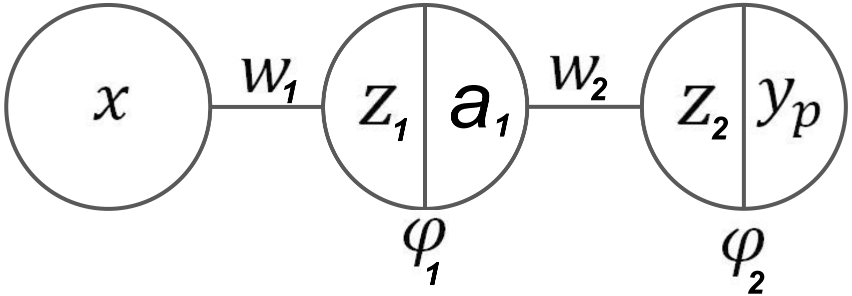
No hidden layers

Computations - feed forward (treat like scalars)



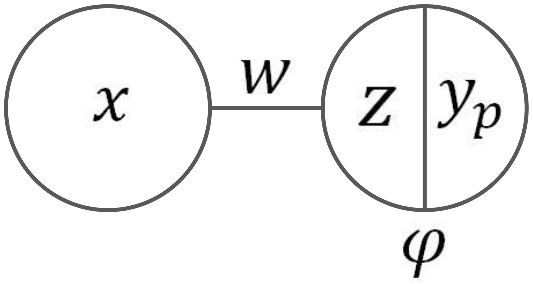
$$z = w\,x$$
$$y_p = \varphi(z)$$

No hidden layers



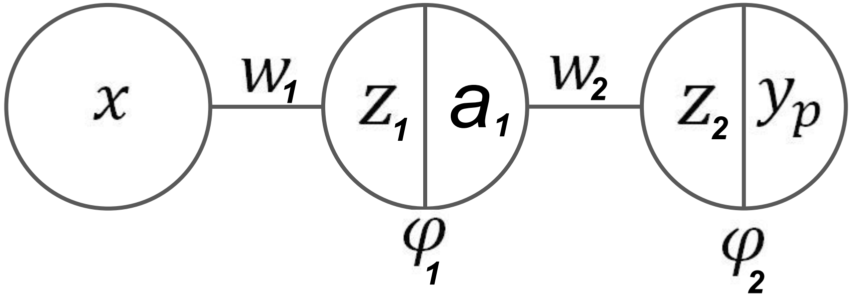
Single hidden layer

Computations - feed forward (treat like scalars)



$z = w\,x$
$y_p = \varphi(z)$

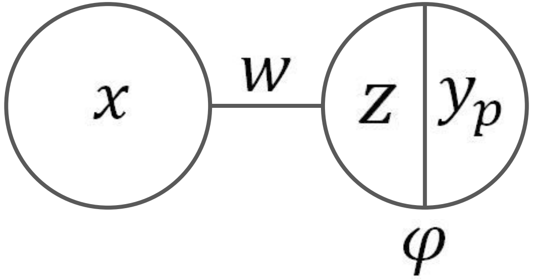
No hidden layers



Single hidden layer

$z_1 = w_1 x$ $\quad z_2 = w_2 a_1$
$a_1 = \varphi_1(z_1)$ $\quad y_p = \varphi_2(z_2)$

Computations - back propagation (scalars)



Back propagation

During training, change weights so that predicted output is closer to train output.

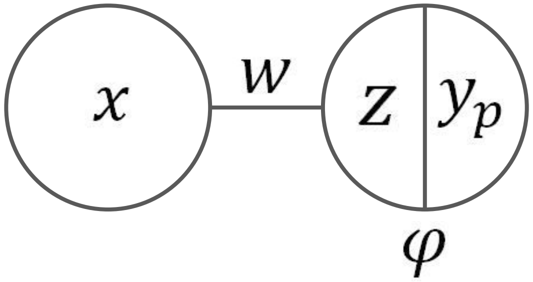
Given

Cost function, $E = \frac{1}{2}\left(y_t - y_p\right)^2$

What is the gradient of the cost function with respect to the weight?

$$\frac{\partial E}{\partial w}$$

Computations - back propagation (scalars)



Given

Cost function,

$$E = \frac{1}{2}\left(y_t - y_p\right)^2$$

What is the gradient of the cost function with respect to the weight?

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y_p}\frac{\partial y_p}{\partial \varphi}\frac{\partial \varphi}{\partial z}\frac{\partial z}{\partial w}$$

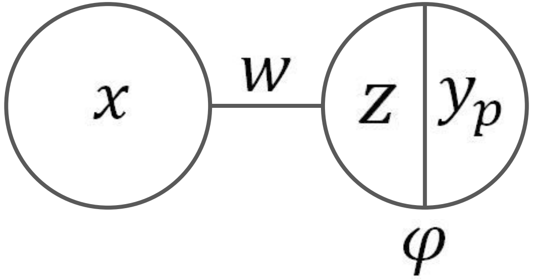
$$\frac{\partial E}{\partial y_p} = -\left(y_t - y_p\right)$$

$$\frac{\partial y_p}{\partial \varphi} = 1$$

$$\frac{\partial \varphi}{\partial z} = \varphi(z)\big(1 - \varphi(z)\big)$$ assuming sigmoid

$$\frac{\partial z}{\partial w} = x$$

Computations - back propagation (scalars)



$$\frac{\partial E}{\partial w} = -(y_t - y_p) \cdot 1 \cdot \varphi(z)\big(1 - \varphi(z)\big) \cdot x$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y_p}\frac{\partial y_p}{\partial \varphi}\frac{\partial \varphi}{\partial z}\frac{\partial z}{\partial w}$$

$$\frac{\partial E}{\partial w} = -(y_t - y_p) \cdot 1 \cdot \varphi(z)\big(1 - \varphi(z)\big) \cdot x$$

$$\frac{\partial E}{\partial w} = -x(y_t - y_p)\varphi(z)\big(1 - \varphi(z)\big)$$

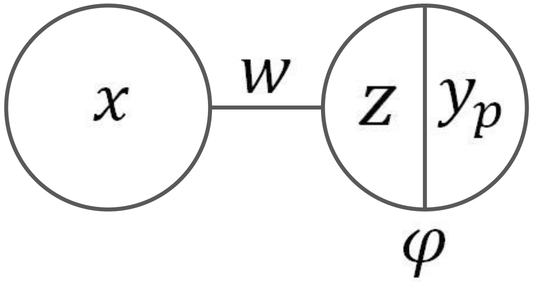
<u>Given</u>

Cost function,

$$E = \frac{1}{2}(y_t - y_p)^2$$

What is the gradient of the cost function with respect to the weight?

Computations - gradient descent (scalars)



$$z = wx$$

$$\frac{\partial E}{\partial w} = -x(y_t - \varphi(wx))\varphi(wx)\big(1 - \varphi(wx)\big)$$

Given

Gradient, $\frac{\partial E}{\partial w}$

$$\Delta w = -\alpha \frac{\partial E}{\partial w}$$

Learning rate, α

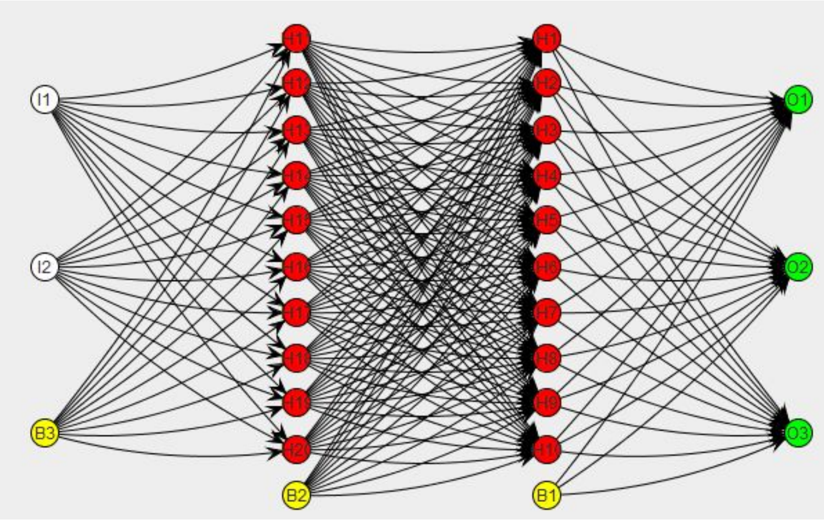
$$w = w + \Delta w$$

How should the weight be updated?

Computations (matrices)

Feed forward

Calculate outputs from inputs
(prediction)



Back propagation

During training, change
weights so that predicted
output is closer to train output.

Computations

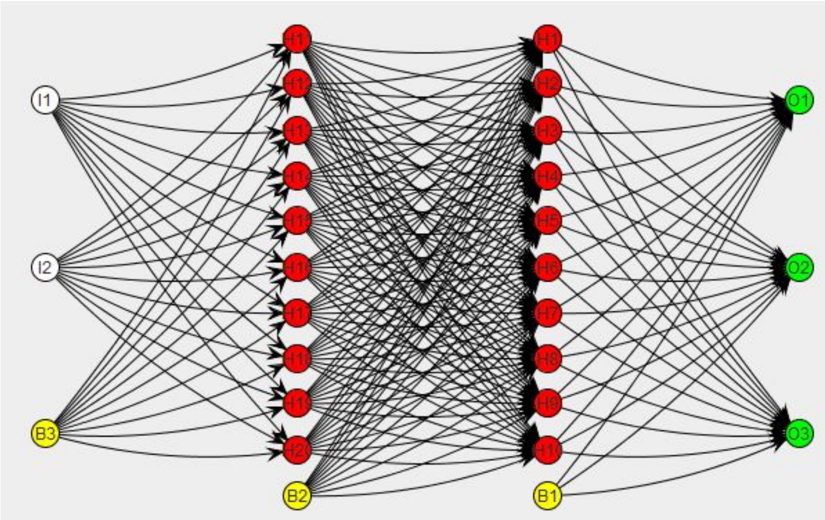
Goal: Minimize the error or loss function - RSS (regression), misclassification rate (classification) - by changing the weights in the model.

Back propagation, the recursive application of the chain rule back along the computational network, allows the calculation of the local gradients, enabling....

Gradient descent to be used to find the changes in the weights required to minimize the loss function.

Feed forward

Calculate outputs from inputs (prediction)



Back propagation

During training, change weights so that predicted output is closer to train output.

MLP pseudocode from scratch

In groups of 2, write some
pseudocode to do this!

MLP pseudocode from scratch

```
# showing stochastic
# training (attempt to converge on weights)
for a desired number of epochs:
    for each row of X, y in inputs, targets:
        Feed-forward to find:
            node activations
            prediction
        Calculate loss
        Backpropagate to find the gradient of the loss w.r.t. the weights
        Use gradient descent to update the weights
    print the training error

# now that the weights are trained
for all test data:
    Feed-foward to find the predictions
print the test error
```

Batch, Mini-Batch, and SGD in pseudocode

```
loop maxEpochs times
  for-each data item
    compute a gradient for each weight and bias
    accumulate gradient
  end-for
  use accumulated gradients to update each weight and bias
end-loop
```

```
loop maxEpochs times
  loop until all data items used
    for-each batch of items
      compute a gradient for each weight and bias
      accumulate gradient
    end-batch
    use accumulated gradients to update each weight and bias
  end-loop all item
end-loop
```

```
loop maxEpochs times
  for-each data item
    compute gradients for each weight and bias
    use gradients to update each weight and bias
  end-for
end-loop
```

[source]()

Batch, Mini-Batch, and SGD in pseudocode

Batch

```
loop maxEpochs times
  for-each data item
    compute a gradient for each weight and bias
    accumulate gradient
  end-for
  use accumulated gradients to update each weight and bias
end-loop
```

Mini-Batch

```
loop maxEpochs times
  loop until all data items used
    for-each batch of items
      compute a gradient for each weight and bias
      accumulate gradient
    end-batch
    use accumulated gradients to update each weight and bias
  end-loop all item
end-loop
```

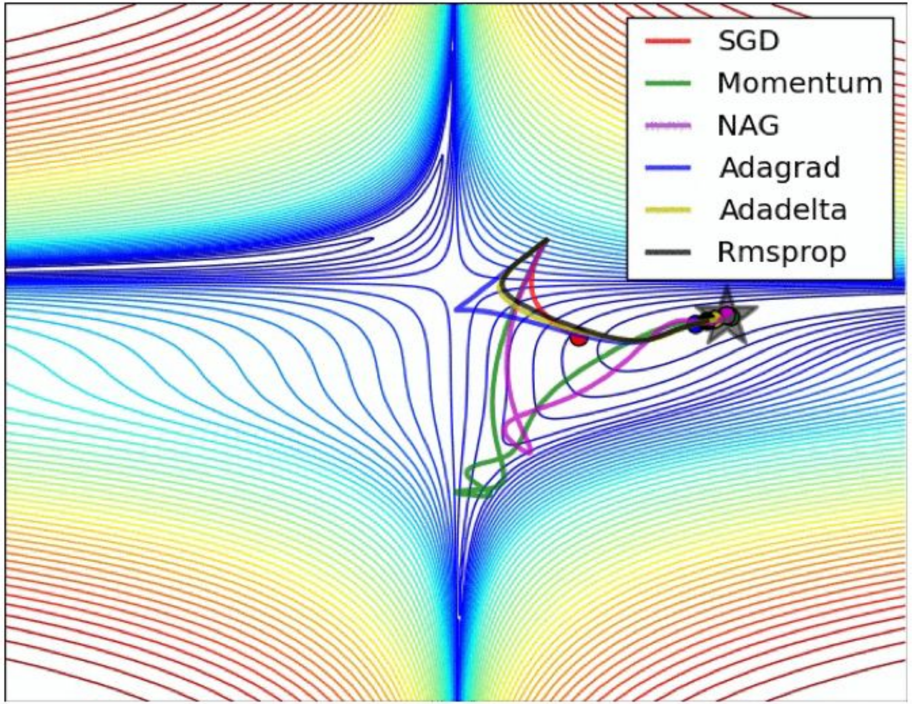
Stochastic

```
loop maxEpochs times
  for-each data item
    compute gradients for each weight and bias
    use gradients to update each weight and bias
  end-for
end-loop
```

source

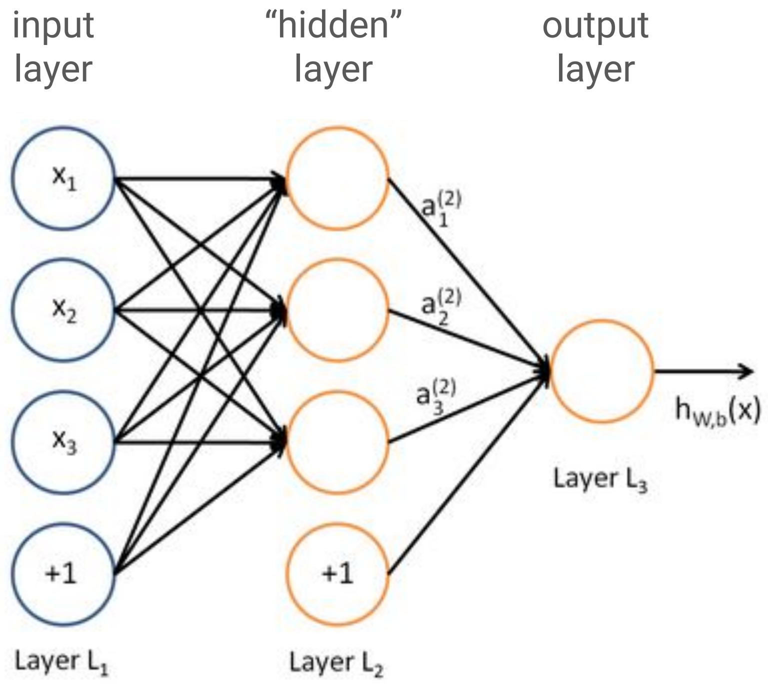
Ok - gradient descent, but how?

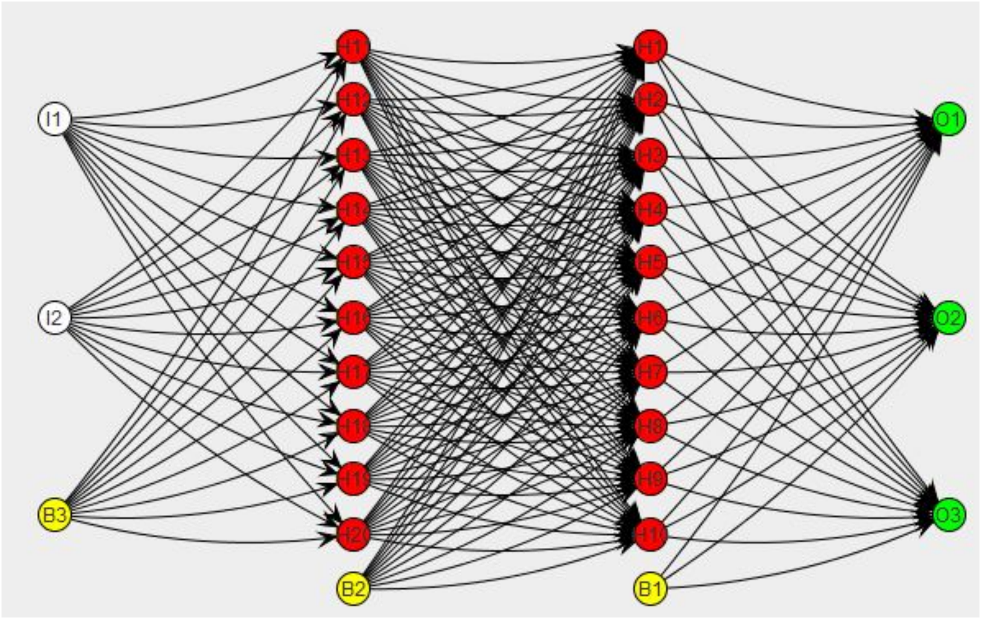
Different gradient descent optimizers: http://cs231n.github.io/neural-networks-3/



See also:
https://keras.io/optimizers/

Multilayer perceptron architecture



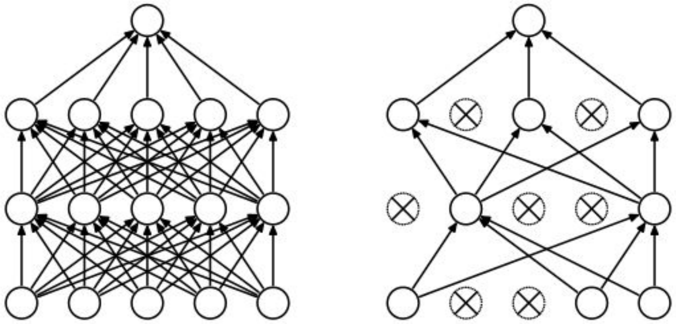


http://ufldl.stanford.edu/wiki/index.php/Neural_Networks

http://www.codeproject.com/KB/recipes/477689/jmsl-7.png

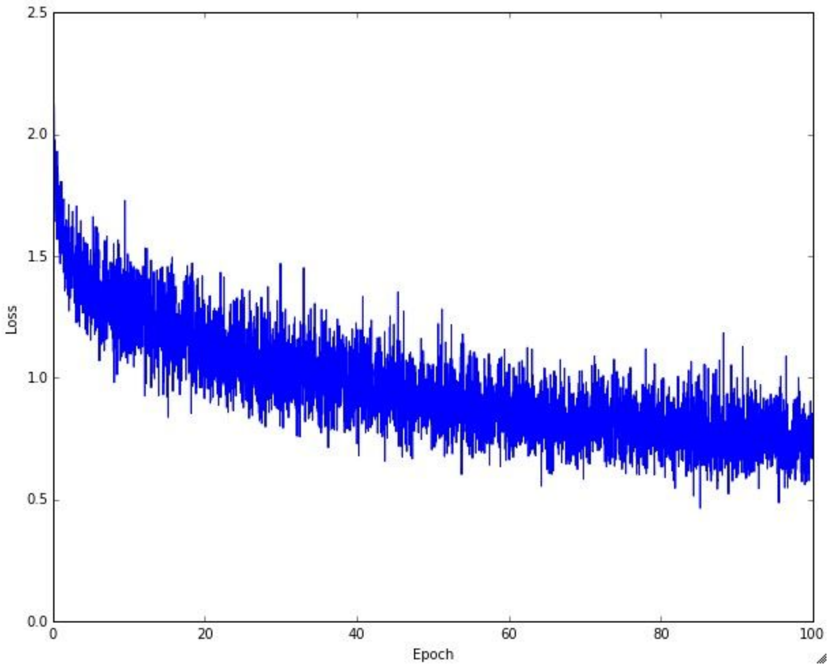
Defining a model - available hyperparameters

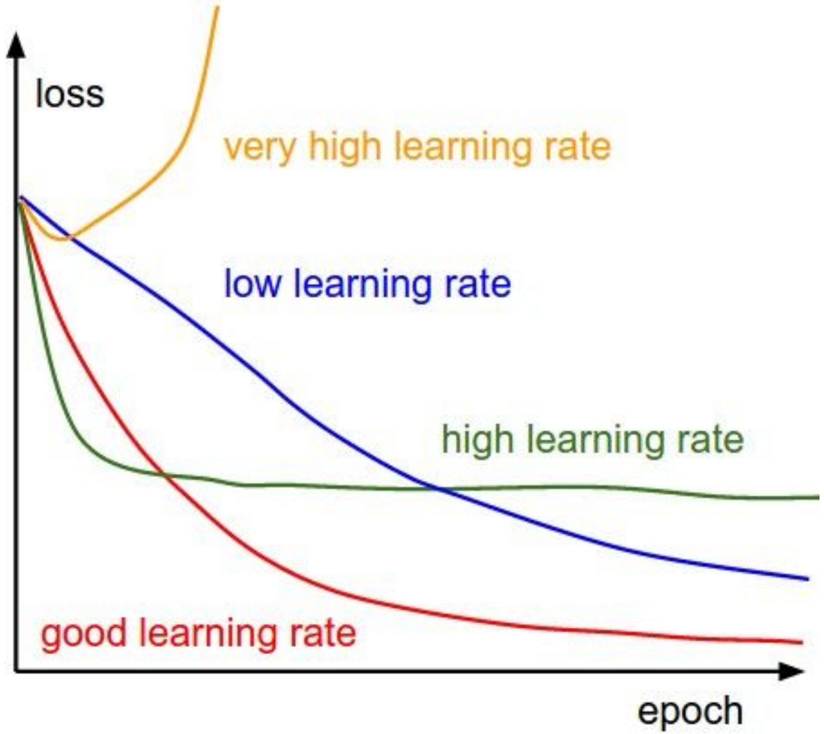
- Most are particular to your application: read the literature and start with something that has been shown to work well.
- Structure: the number of hidden layers, the number of nodes in each layer
- Activation functions
- Weight and bias initialization (for weights Karpathy recommends Xavier init.)
- Training method: Loss function, learning rate, batch size, number of epochs
- Regularization: Likely needed! (NN very complex models that will overfit)
    - weight decay (L1 & L2, see here), early stopping, dropout
- Random seed



(a) Standard Neural Net (b) After applying dropout.

Monitor training process





References

- http://cs231n.stanford.edu/ - Andrej Karpathy slides, lectures on youtube
- http://playground.tensorflow.org Visualize neural network training online
- A.I. vs. Machine Learning vs. Deep Learning