# Neural Networks:
Multi-layer perceptron

Objectives

- Neural net history
  - Biomimicry
  - AI Winter(s)
  - Enablers of current resurgence
- "Vanilla" neural networks: multilayer perceptrons
  - Parts of a neuron
  - Feed-forward
  - Backpropagation and gradient descent
- Provide examples of NN hyperparameters and training considerations
- Keras introduction (Python neural networks API)
- References

Neural Net motivation

Our brain: the ultimate parallel computer
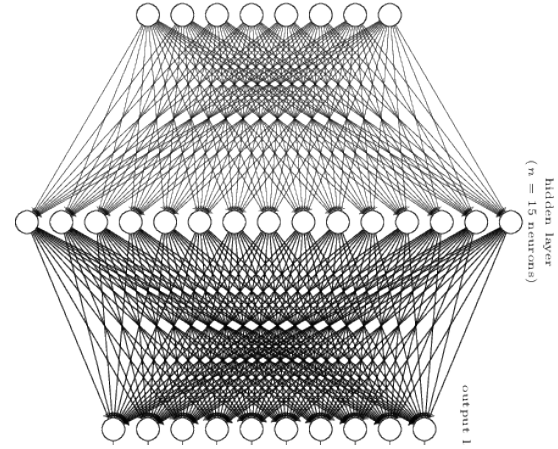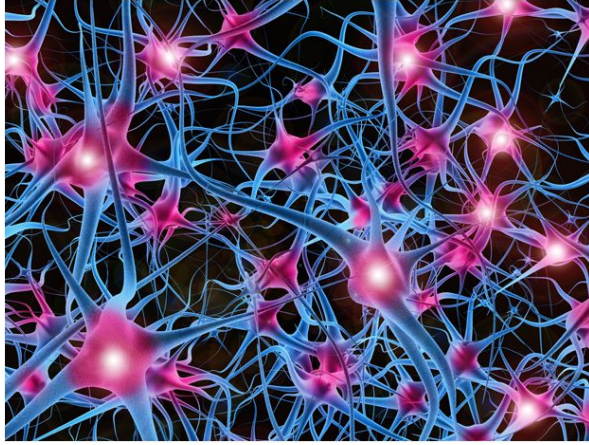
Number of parameters of biggest artificial neural net: 160,000,000,000 ([Digital Reasoning, 2015](#))

The number of neurons in your brain: 86,000,000,000
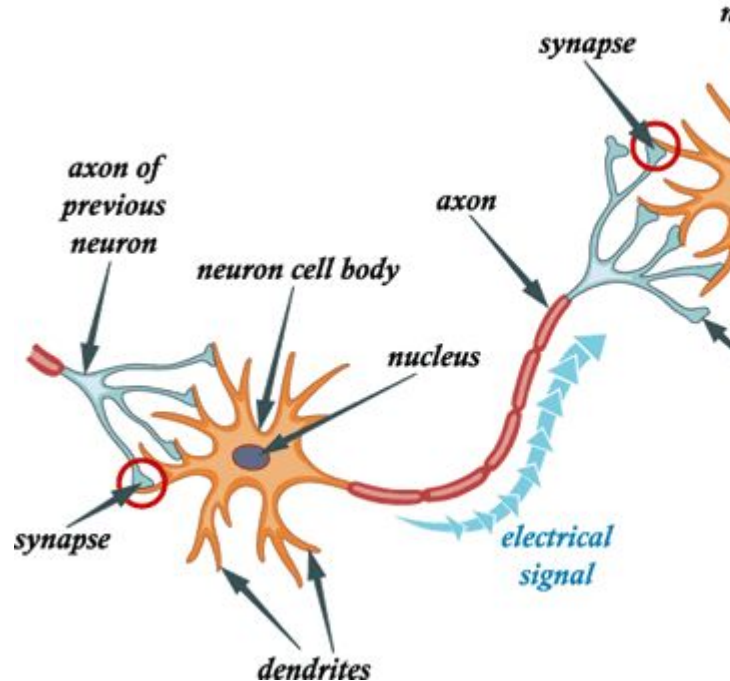The number of synapses in your brain: 1,000,000,000,000,000
The average number of synapses per neuron: 10,000

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

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.
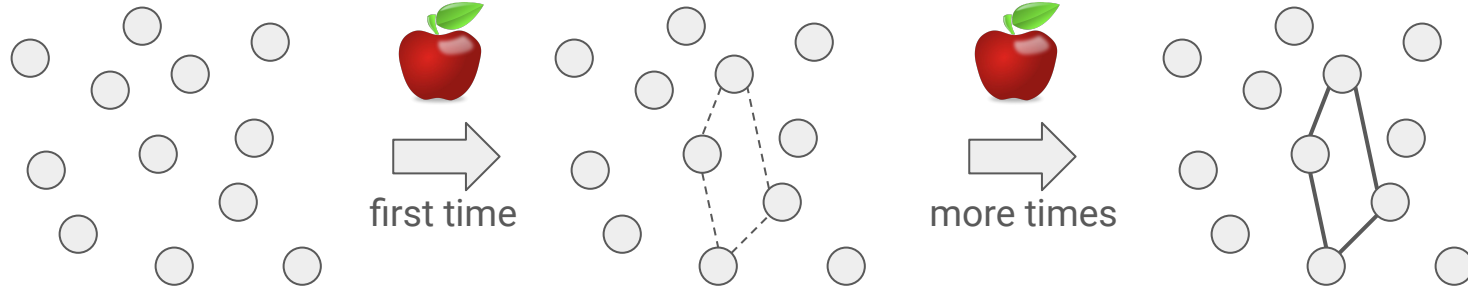
Improvements to MCP neuron

- 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."

  a.k.a.

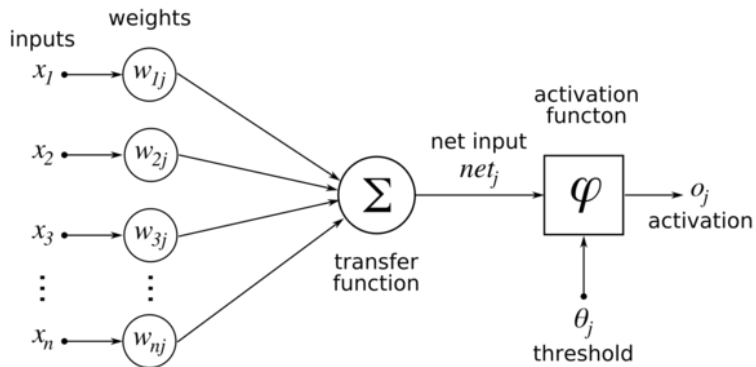 **"Cells that fire together wire together." - Hebb's rule**

  Repeated activity in response to a stimulus changes a neural network as it learns.
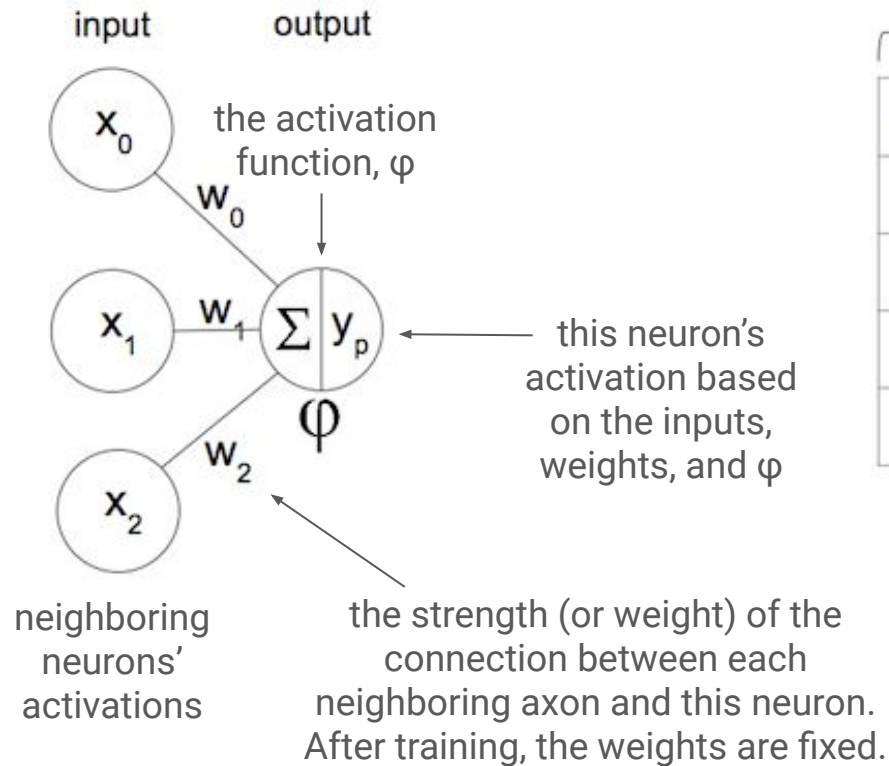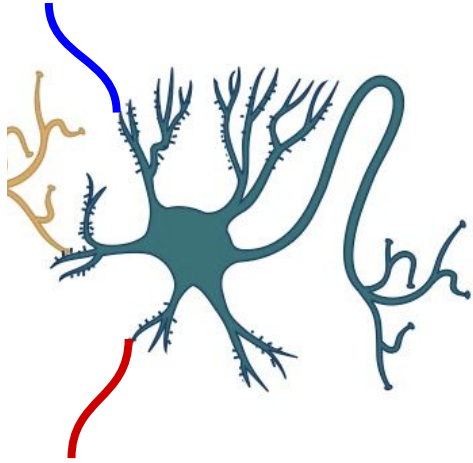

first time   more times

Neurons before learning about apple

Improvements to MCP neuron

- 1957, Frank Rosenblatt invents the Perceptron
  - Initializes the weights (synaptic strength) to random values (e.g. -1.0 to 1.0)

  - Weights change during supervised learning according to the delta rule, $\sim(yi - yp)$. After a <u>certain number of training passes through the whole training set</u> (a.k.a. the number of <u>epochs</u>) 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).

Perceptron

input   output

x_0

w_0

the activation function, φ

x_1 w_1 $\Sigma \mid y_p$

φ

w_2

x_2

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.

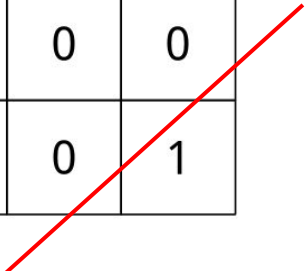| Inputs, X			Desired output, y
	X		y
x_0	x_1	x_2	y_t
1	0	0	0
1	1	1	1
1	0	1	0
1	1	0	1

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 logical function, larger networks also have similar limitations, and therefore should be dropped."*
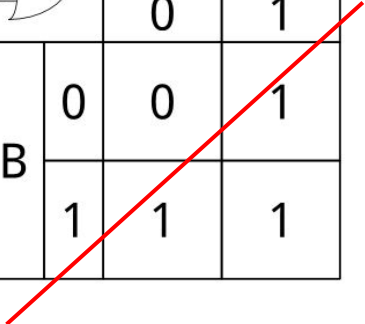
AND

A	AND	Q		A	
B				0	1
		0	0	0	0
B		1	0	0	1

OR

A	OR	Q		A	
B				0	1
		0	0	0	1
B		1	1	1	1

XOR

| A	XOR	Q		A	
B				0	1
		0	0	0	1
B		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 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 (multi-layer perceptron) networks.

AND

		A	
		0	1
B	0	0	0
	1	0	1

OR

		A	
		0	1
B	0	0	1
	1	1	1

XOR

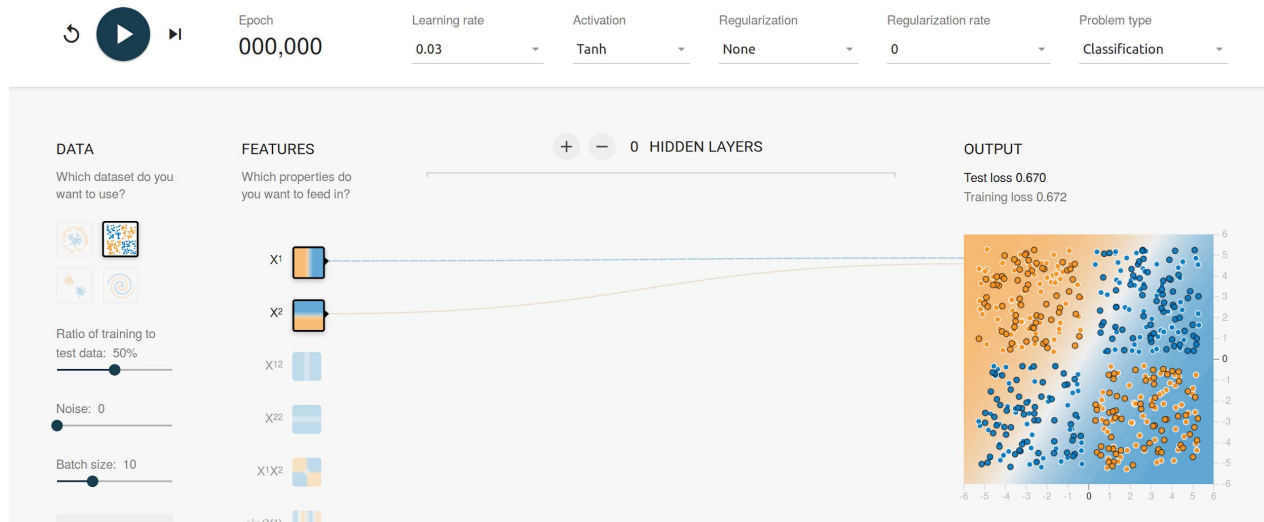|		A	
		0	1
B	0	0	1
	1	1	0

Prove it to yourself

"Single layer perceptron (no hidden layer) networks are limited to being linear classifiers. Not true of deeper MLP (multi-layer perceptron) networks."

https://playground.tensorflow.org/

How many hidden layers & neurons in the hidden layer are needed to learn XOR?

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

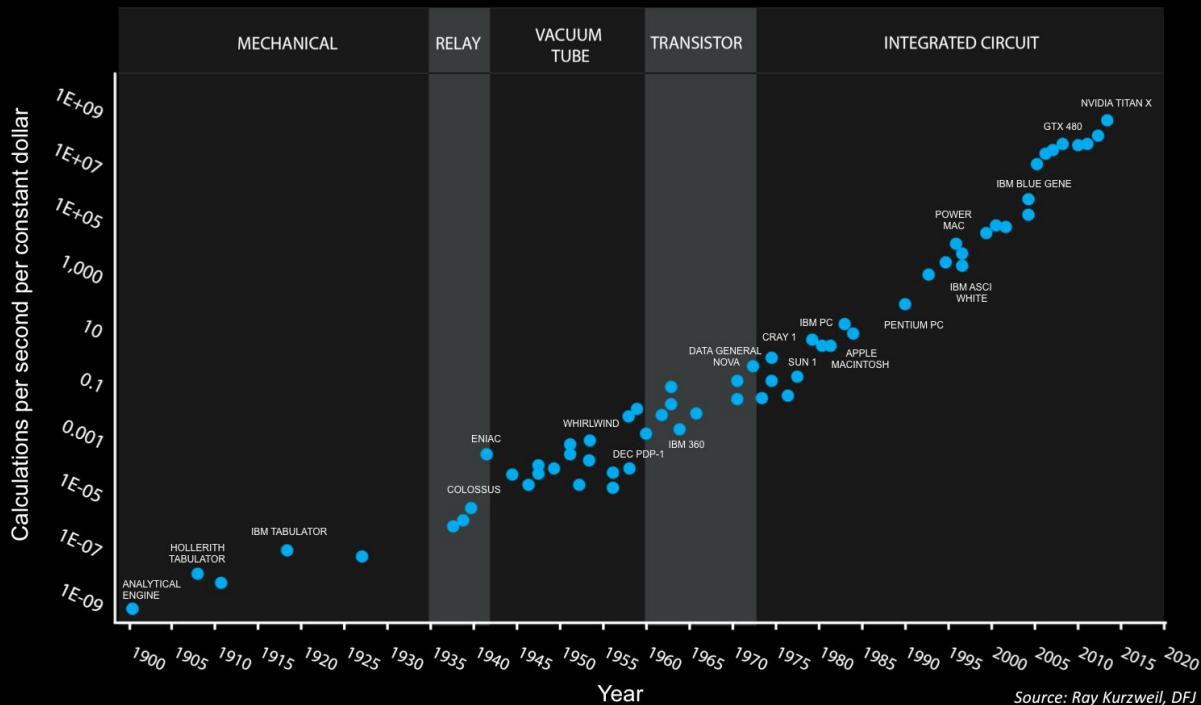19. ABSTRACT (Continue on reverse if necessary and identify by block number)

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

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

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.

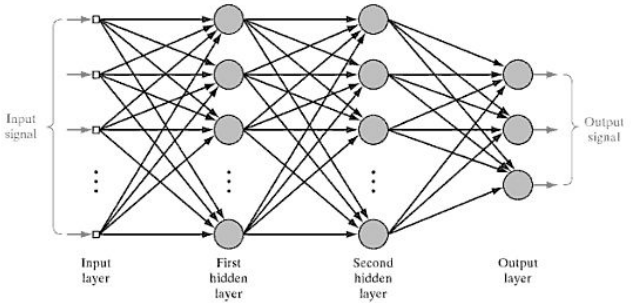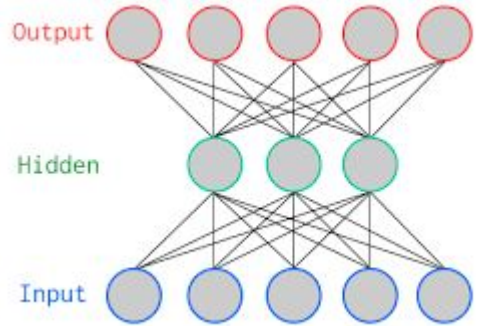http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

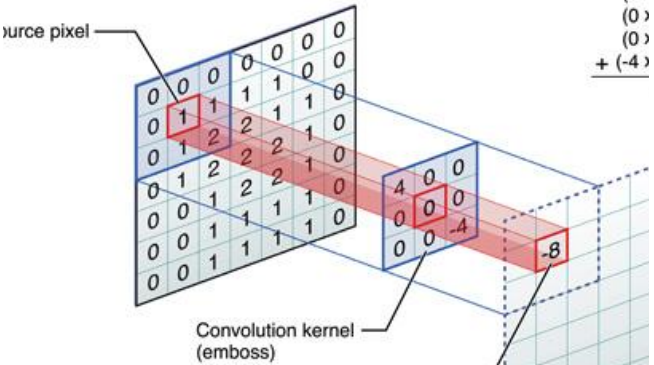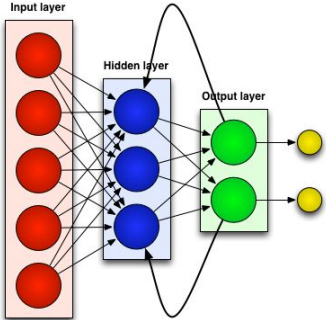# (Artificial) Neural networks

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."
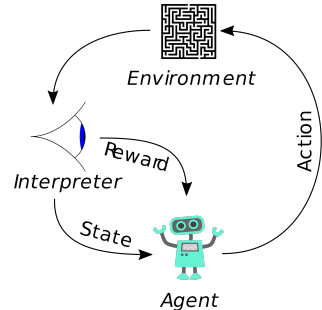
ANNs we will talk about

ANN	Description Past capstones	Picture of architecture
Multilayer perceptron (MLP)	Standard algorithm for supervised learning. Used for pattern, speech, image recognition. Smarter than Nate Silver Prediction Solar Power Plant Insolation	
Autoencoder	Used for unsupervised learning of efficient codings, usually for dimensionality reduction, though recently to make generative models. Tasty-Tweets Solving LSAT Puzzles with seq2seq Wedfuly Recommender	

ANNs we will talk about

| ANN	Description Past capstones	Picture of architecture
Convolutional neural network	Node connections inspired by visual cortex. Uses kernels to aggregate/transform information in network. Used for image, video recognition, nlp EyeNet, Curb Appeal, Wildflower Classifier ChefNet Brain Tumor Segmentation Detect Depression in Speech	
Recurrent neural network	Connections between nodes can be cyclical, giving the network memory. Used for sequences: handwriting, speech recognition, time series. RadStaffer GOT Book Generator ABC Music	

ANN topics we will talk about

Topic	Description	Picture
	Past capstones	
Reinforcement Learning	Use NNs feature extraction and prediction abilities to learn from an environment gitMunny - Crypto Trading with Deep Q Networks Sphero Navigation	
Transfer Learning	Leverage vetted NN architectures and pre-trained weights to get good performance on limited data Dog Breed Classifier	

"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)
  - Extrapolate to multilayer networks

- Gradient descent solution optimizers

- Multi-layer network

- MLP code-it-from-scratch pseudocode

A perceptron (neuron, neurode, node, unit)
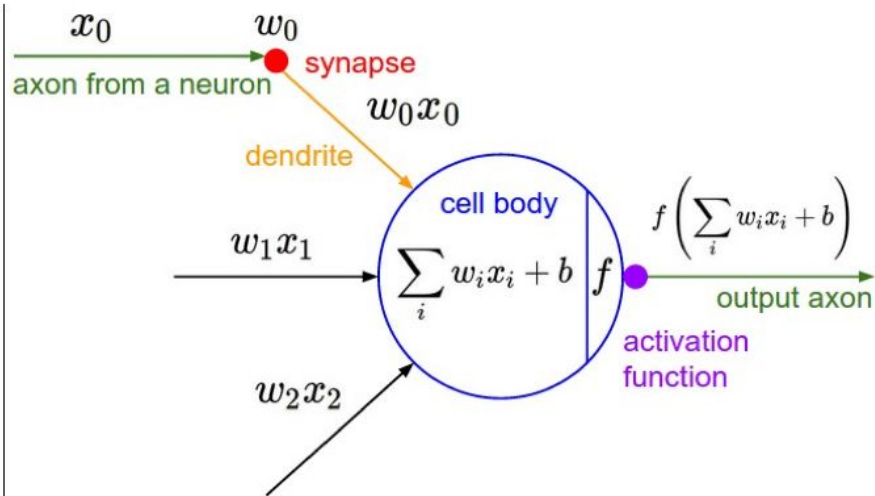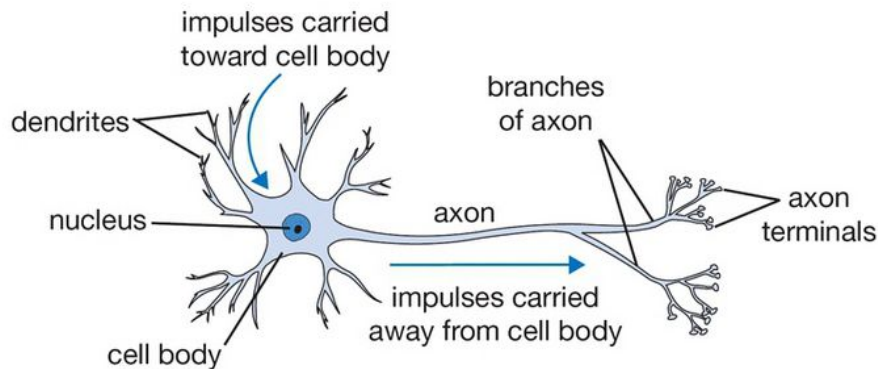
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

A perceptron (neuron, neurode, node, unit)
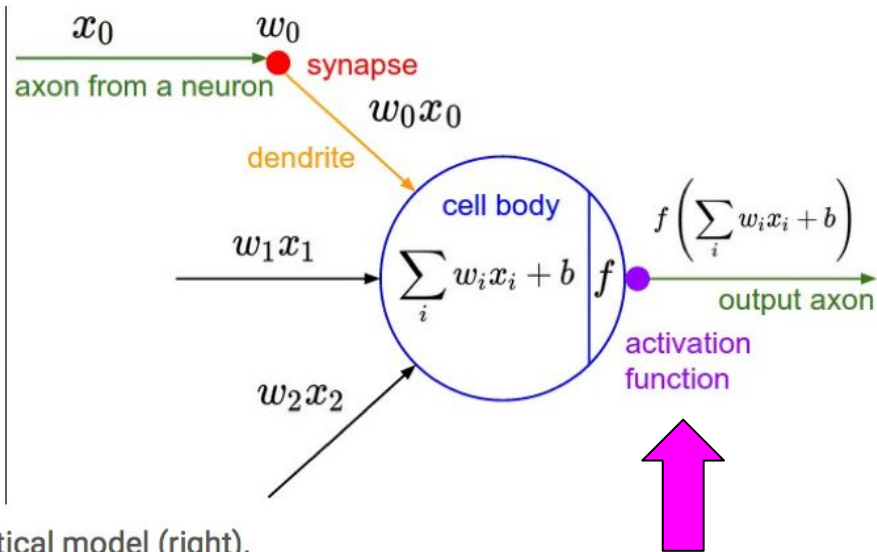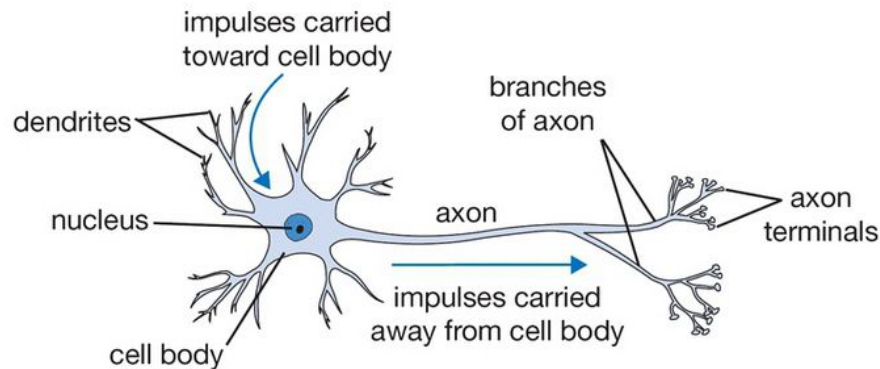
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Sigmoid activation function, and its derivative
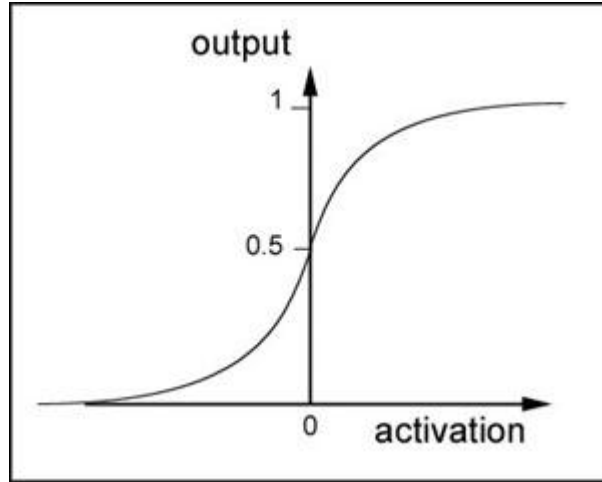
$$y = \frac{1}{1 + e^{-x}}$$

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

Sigmoid

$\sigma(x) = 1/(1 + e^{-x})$

ReLU
max(0,x)

tanh(x)

Leaky
ReLU
max(0.1x, x)

Your choice of an activation function for a given layer of your network should be informed by literature and your experience.
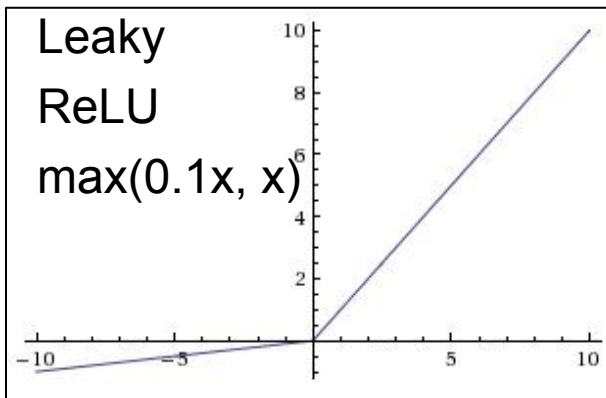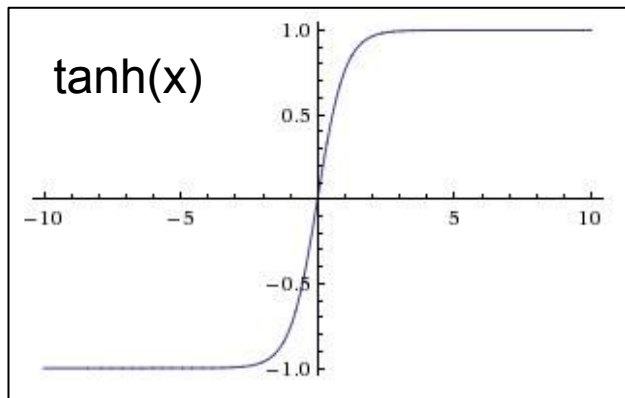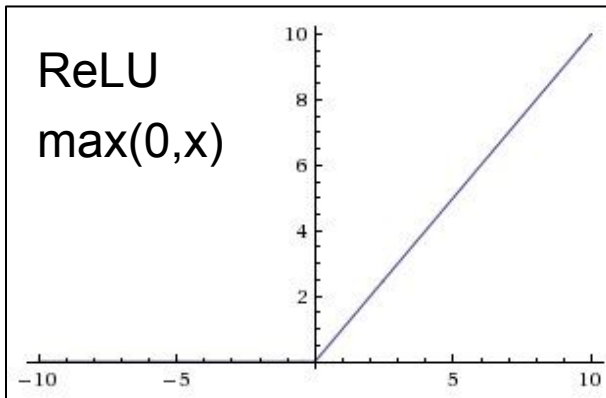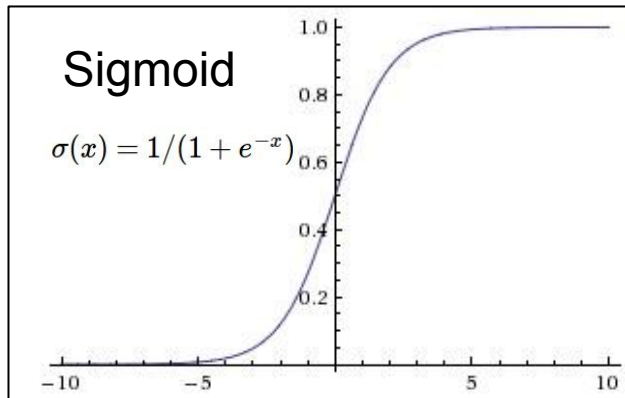
Questions to consider when selecting:

1) Benefit of zero mean?
2) Does derivative exist everywhere and does it give useful information?
3) Easy to compute?
4) Weight and bias initializations differ for these - take care!

Karpathy:
ReLU > Leaky ReLU > tanh (but be careful of ReLU learning rates)

and: **beware of sigmoid!** (exploding, vanishing gradient issues, slower to compute)

Computations

Feed forward
Calculate outputs from inputs
(prediction)

x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite

cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$ f
output axon
activation function
$w_2 x_2$

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

Computations - Feed forward (in 1 perceptron)

Given

Input, x

Weight, w

Product, $z = wx$

Activation function, φ

What is the prediction, y_p ?

Computations - Feed forward (in 1 perceptron)

Given

Input, x

Weight, w

Product, $z = wx$

Activation function, φ

What is the prediction, y_p ?

$$y_p = \varphi(z)$$

where

$$z = wx$$

Computations: Backpropagation (calculate the gradient)
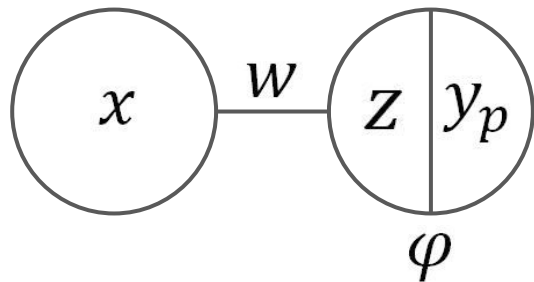
Given

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

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

$\dfrac{\partial E}{\partial w}$

Backpropagation (1)

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

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

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

Given

Cost function,

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

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

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

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

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

Backpropagation (2)

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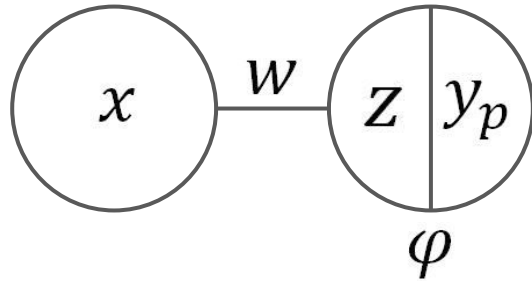
$$\frac{\partial E}{\partial w} = -(y_t - y_p) \cdot 1 \cdot \varphi(z)(1 - \varphi(z)) \cdot x$$

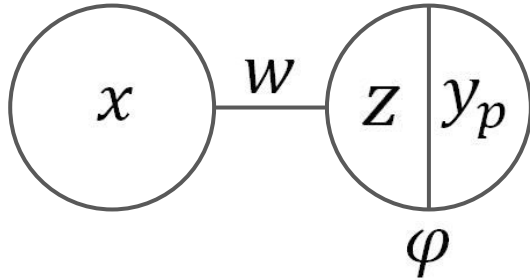$$\frac{\partial E}{\partial w} = -x(y_t - y_p)\varphi(z)(1 - \varphi(z))$$

Given

Cost function,

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

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

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

Computations: Gradient descent to update the weight
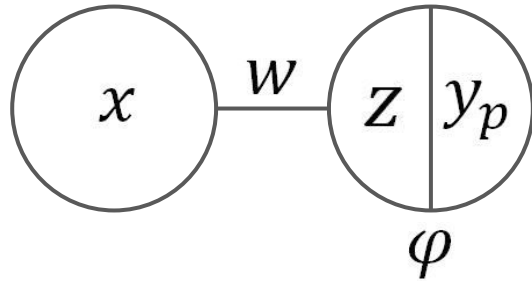
$$w = w - \alpha \frac{\partial E}{\partial w} \quad \text{or} \quad w = w + \Delta w$$

Given

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

Learning rate, α

How should the weight be updated? Δw

Computations: Gradient descent to update the weight
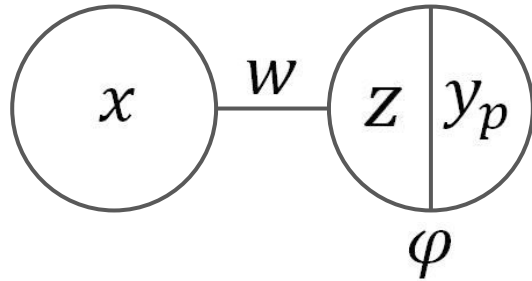
$$w = w - \alpha \frac{\partial E}{\partial w} \quad \text{or} \quad w = w + \Delta w$$

$$z = wx$$

Given

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

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

Learning rate, α

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

How should the weight be updated? $\quad \Delta w$

$$\Delta w = \alpha x(y_t - \varphi(wx))\varphi(wx)(1 - \varphi(wx))$$

Computations

Feed forward
Calculate outputs from inputs
(prediction)
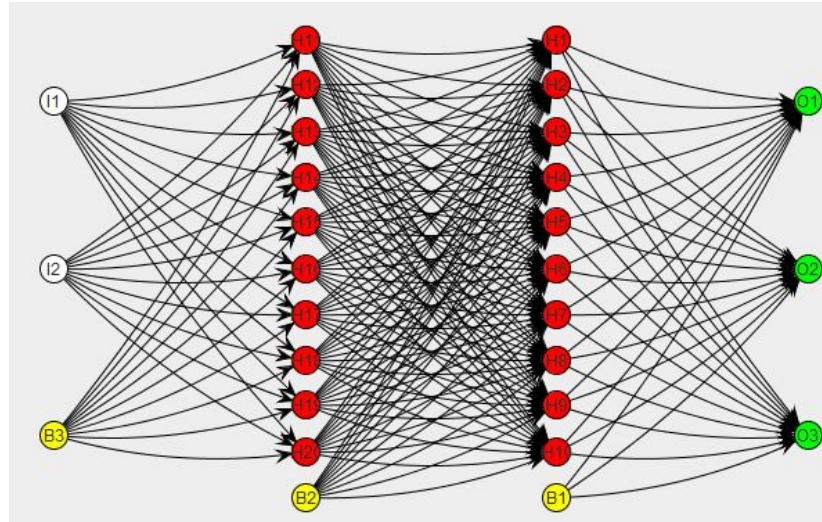
Back propagation
During training, change
weights so that predicted
output is closer to train output.

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.

The idea is the same for a single perceptron, or for a multi-layer network.

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 4, write some
pseudocode to do this!

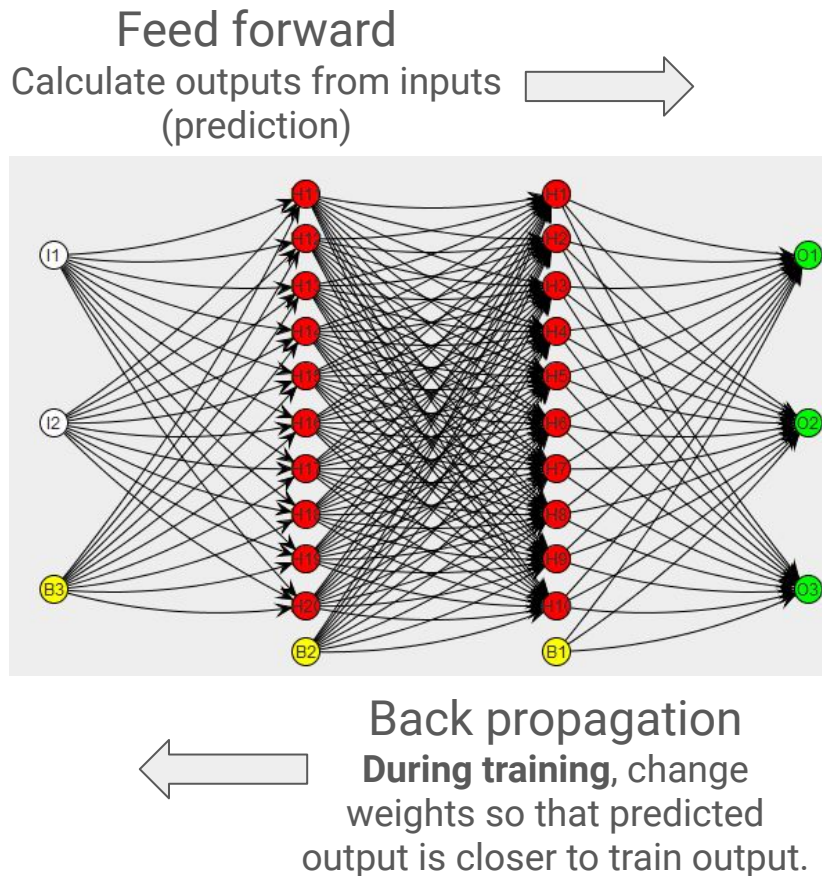# MLP pseudocode from scratch

```
# showing training using stochastic gradient descent
# 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
```
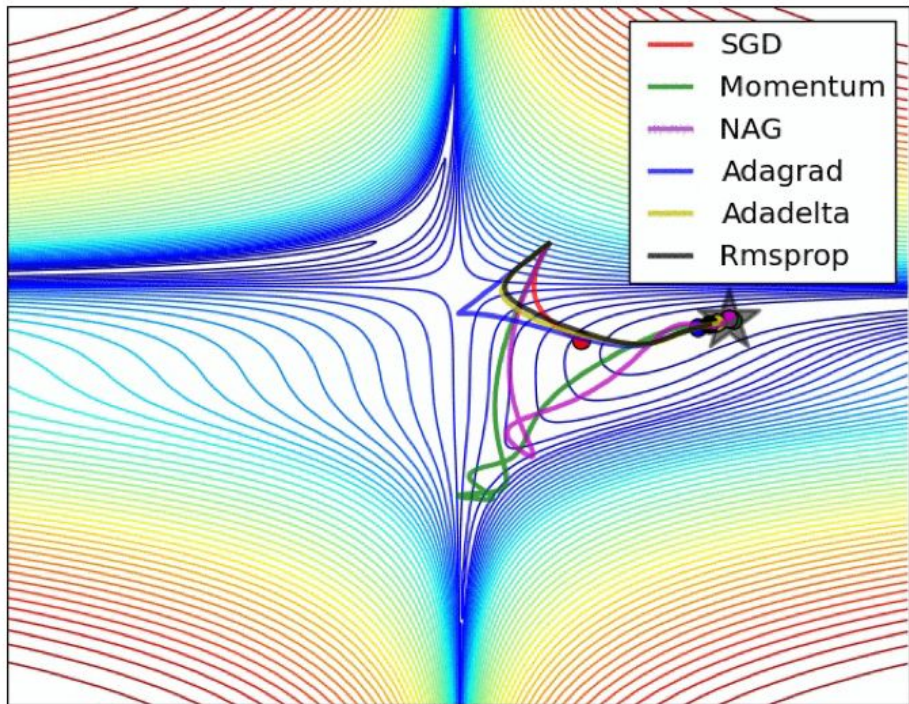
Stochastic

```
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:

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

Multilayer perceptron architecture

input layer

"hidden" layer

output layer

x_1

x_2

x_3

+1

Layer L_1

$a_1^{(2)}$

$a_2^{(2)}$

$a_3^{(2)}$

+1

Layer L_2

Layer L_3

$h_{W,b}(x)$
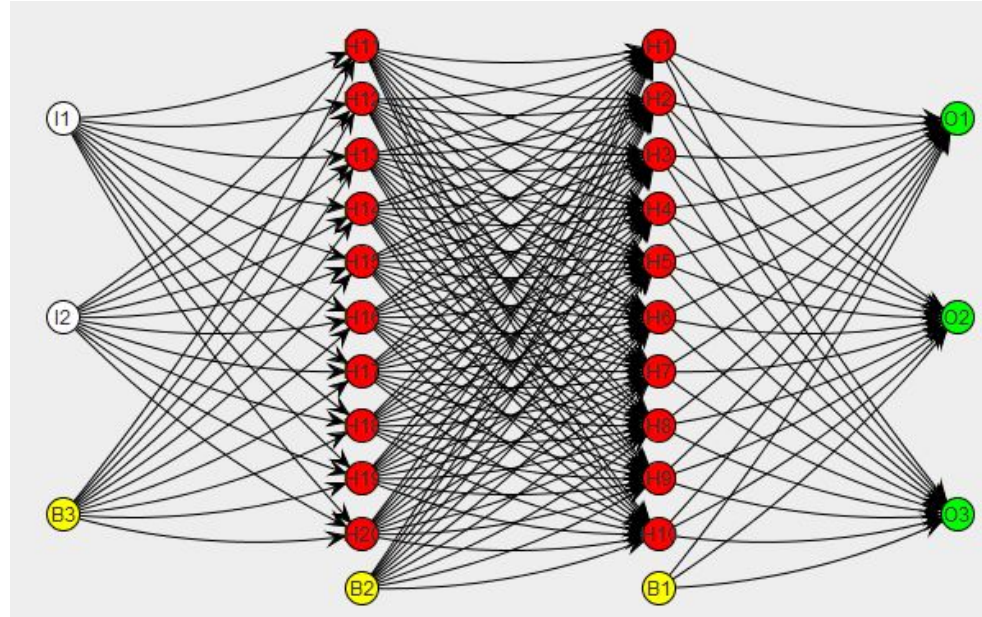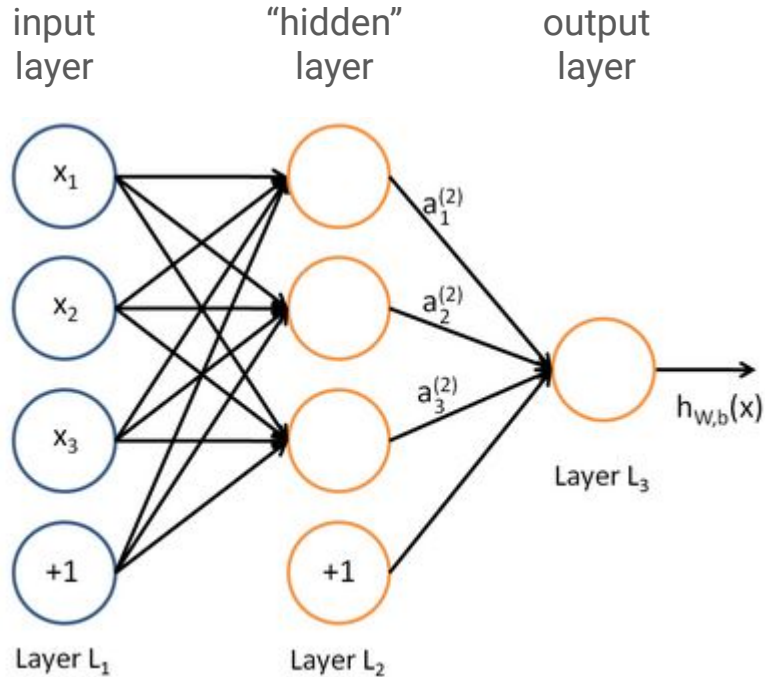
Training

Pre-processing

Standardized data
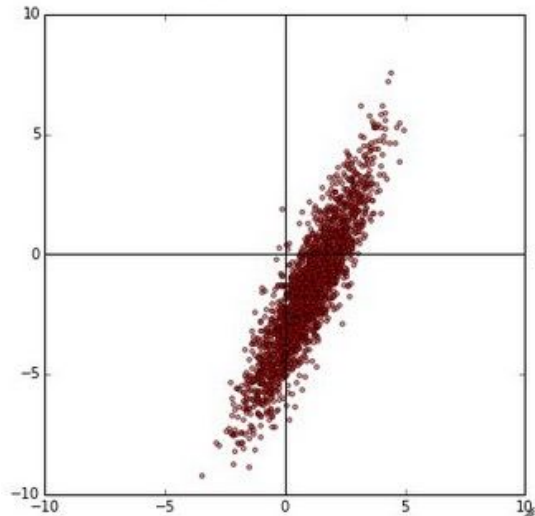
original data   zero-centered data   normalized data
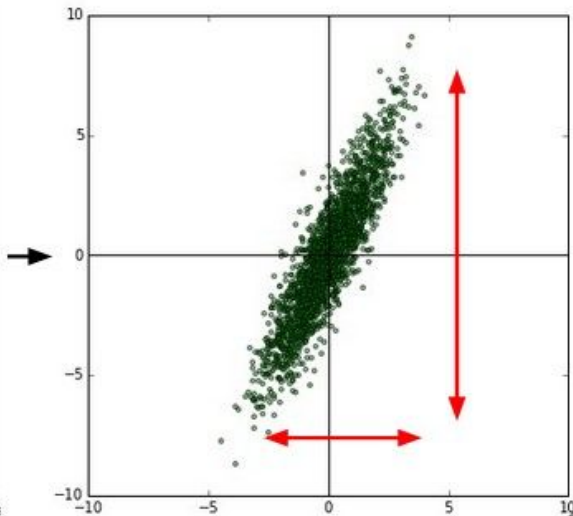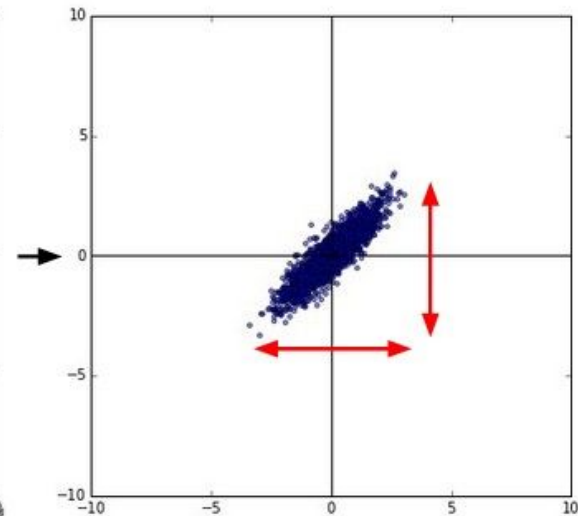
`X -= np.mean(X, axis = 0)` `X /= np.std(X, axis = 0)`

Karpathy advice for images: subtract the mean image only, can do it per channel

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
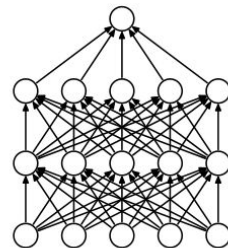
(a) Standard Neural Net   (b) After applying dropout.
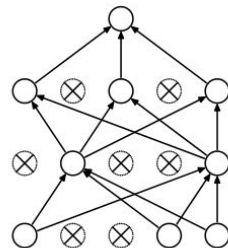
Defining a model - available hyperparameters

Yes, it's a lot of hyperparameters.

Defining a model - available hyperparameters

Yes, it's a lot of hyperparameters.

How do you figure out the right ones?

Defining a model - available hyperparameters

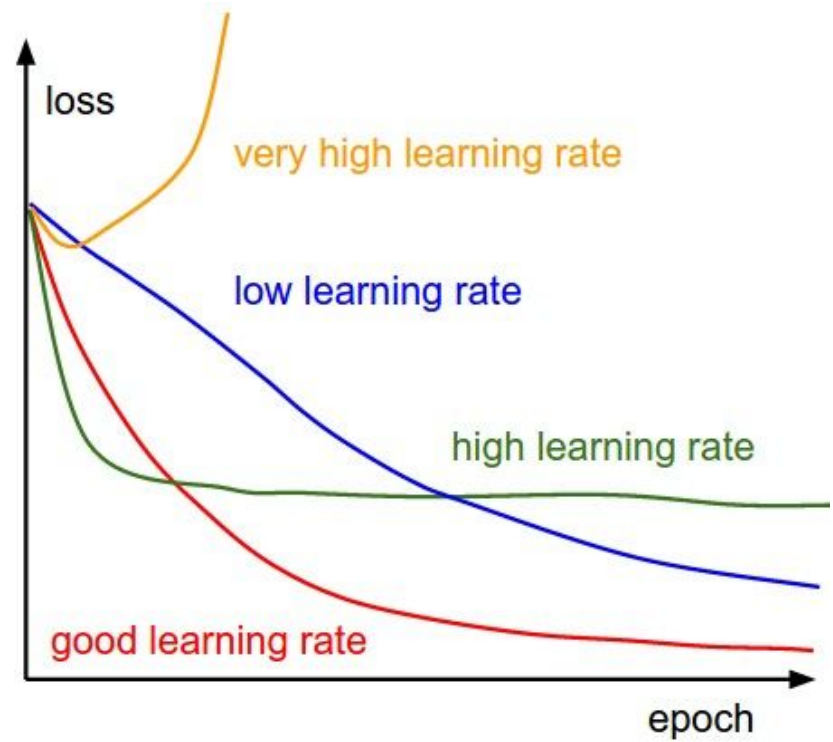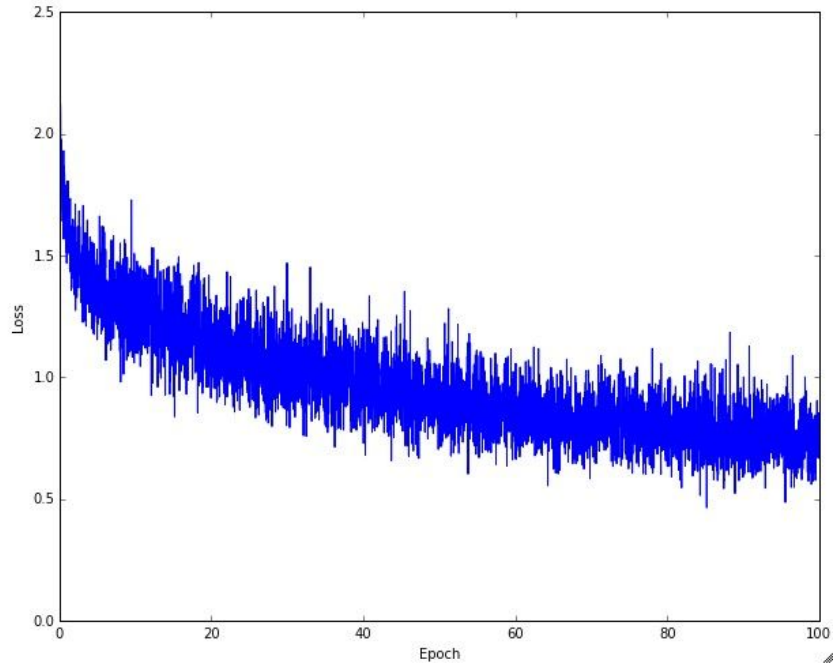Yes, it's a lot of hyperparameters.

How do you figure out the right ones?

1) Search the literature for similar use case and architecture to define a baseline model.

2) Once you have that, start varying the hyperparameters and check performance using **cross-validation.**

Monitor training process

Potential problems

- Instability, difficulty visualizing the training process, interpretability
- The vanishing (or exploding) gradient problem during backpropagation:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\alpha\left(\frac{\partial E}{\partial w_{ij}}\right)$$

$$\Delta w_{32} = -\alpha\left(\frac{\partial E}{\partial w_{32}}\right)$$

$$\frac{\partial E}{\partial w_{32}} \propto \varphi'(net_0)\cdot\ \varphi'(net_1)\cdot\varphi'(net_2)$$

The weights farthest away from the output can have very large, or very small, updates.
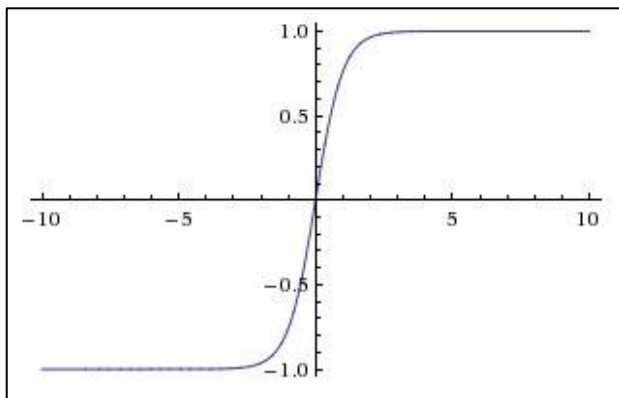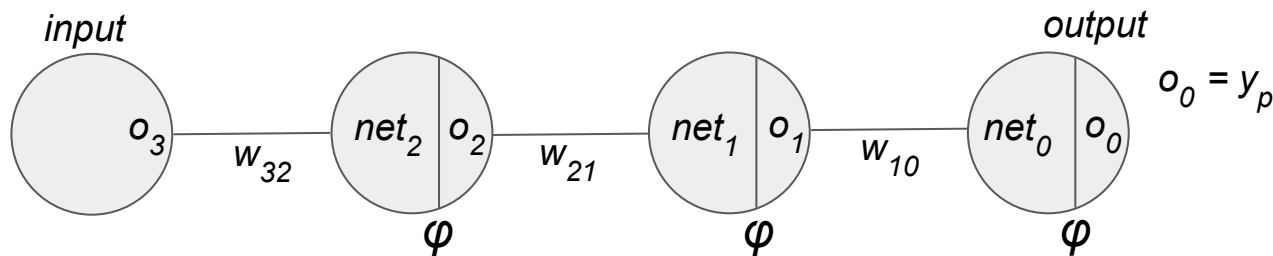
Implementing a neural network in Python using Keras

Keras

Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at Ke

Keras is compatible with: **Pyth**

Examples

Here are a few examples to get you started!

In the examples folder, you will also find example models for real datasets:

- CIFAR10 small images classification: Convolutional Neural Network (CNN) with realtime data augmentation
- IMDB movie review sentiment classification: LSTM over sequences of words
- Reuters newswires topic classification: Multilayer Perceptron (MLP)
- MNIST handwritten digits classification: MLP & CNN
- Character-level text generation with LSTM

...and more.

Quick overview of example code...

Compare
`3_layer_network.py`
to
`3_layer_keras.py`

X   H   y
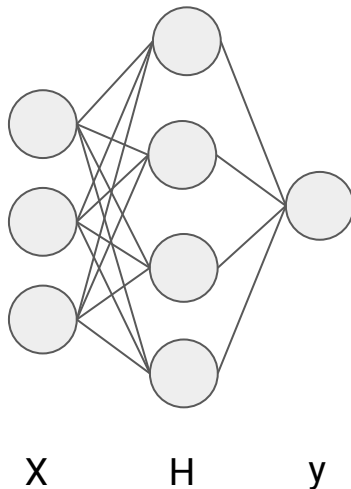
```
X = np.array([[0,0,1],
              [0,1,1],
              [1,0,1],
              [1,1,1]])

y = np.array([[0],
              [1],
              [1],
              [0]])
```

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

- Goodfellow et. al. Deep Learning Book (online)

Objectives

- Neural net history
  - Biomimicry
  - AI Winter(s)
  - Enablers of current resurgence
- "Vanilla" neural networks: multilayer perceptrons
  - Parts of a neuron
  - Feed-forward
  - Backpropagation and gradient descent
- Provide examples of NN hyperparameters and training considerations
- Keras introduction (Python neural networks API)
- References

Appendix

Activation function derivatives

	Propagation	Back-propagation
Sigmoid	$y_s = \dfrac{1}{1+e^{-x_s}}$	$\left[\dfrac{\partial E}{\partial x}\right]_s = \left[\dfrac{\partial E}{\partial y}\right]_s \dfrac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\dfrac{\partial E}{\partial x}\right]_s = \left[\dfrac{\partial E}{\partial y}\right]_s \dfrac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\dfrac{\partial E}{\partial x}\right]_s = \left[\dfrac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min\left(-1, \max(1, x_s)\right)$	$\left[\dfrac{\partial E}{\partial x}\right]_s = \left[\dfrac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$

http://www.marekrei.com/blog/26-things-i-learned-in-the-deep-learning-summer-school/

Loss function derivatives

	Propagation	Back-propagation
Square	$y = \frac{1}{2}(x-d)^2$	$\frac{\partial E}{\partial x} = (x-d)^T \frac{\partial E}{\partial y}$
Log $\quad c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1+e^{cx}} \frac{\partial E}{\partial y}$
Hinge $\quad c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \; \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax $\quad c = 1 \dots k$	$y = \log\left(\sum_k e^{x_k}\right) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = \left(e^{x_s} / \sum_k e^{x_k} - \delta_{sc}\right) \frac{\partial E}{\partial y}$
MaxMargin $\quad c = 1 \dots k$	$y = \left[\max_{k \neq c}\{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

http://www.marekrei.com/blog/26-things-i-learned-in-the-deep-learning-summer-school/