Deep learning study group

MEETUP #2

October 21, 2016

Housekeeping

- Q&A from last time
- Does the time work for everyone?
- News

Motivation #1

One might write an algorithm for identifying cats in images

Motivation #1

One might write an algorithm for identifying cats in images

Figure: *authors note: only fun I can get

Motivation #1

One might write an algorithm for identifying cats in images

Figure: *authors note: only fun I can get

How to do it?

Motivation #2

Lots of problems:

- *Viewpoint variation*. A single instance of an object can be oriented in many ways with respect to the camera
- *Scale variation*. Visual classes often exhibit variation in their size
- *Deformation*. Many objects of interest are not rigid bodies and can be deformed in extreme ways
- *Occlusion*. The objects of interest can be occluded
- *Illumination conditions*. The effects of illumination are drastic on the pixel level
- *Background clutter*. The objects of interest may blend into their environment, making them hard to identify
- *Intra-class variation*. The classes of interest can often be relatively broad, such as chair

Motivation #3

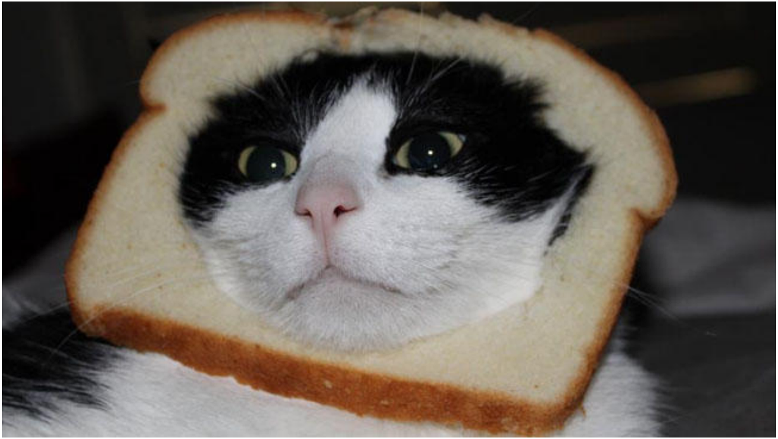


Figure: *I hope I can sneak this in. Broken cat

Not so obvious as writing an algorithm for, for example, sorting a list of numbers

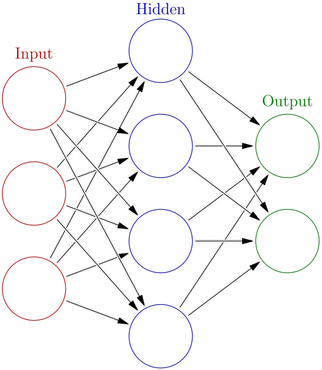
Not so obvious as writing an algorithm for, for example, sorting a list of numbers

Data driven aproach

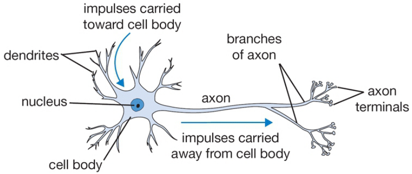
Not so obvious as writing an algorithm for, for example, sorting a list of numbers

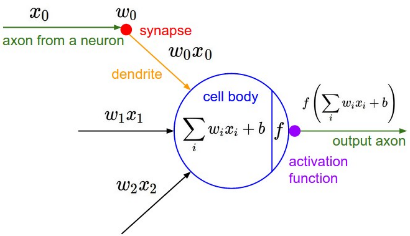
Data driven aproach

Artificial Neural Networks



History #1


impulses carried
toward cell body
dendrites
branches
of axon
axon
terminals
nucleus
axon
impulses carried
away from cell body
cell body


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
$w_2 x_2$

Perceptron #1

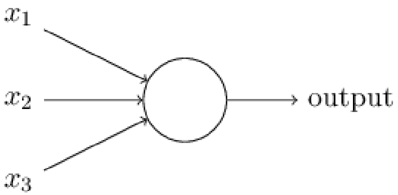


Figure: Perceptron with tree inputs x_1, x_2, x_3

Perceptron #2

In algebraic terms:

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \tag{1}$$

Where x_j are inputs and w_j are weights

Perceptron #3

Lets simplify it:

$$w \cdot x \equiv \sum_j w_j x_j$$

Perceptron #3

Lets simplify it:

$$w \cdot x \equiv \sum_j w_j x_j$$

$$output = \begin{cases} 0 & \text{if } w \cdot x \leq \text{threshold} \\ 1 & \text{if } w \cdot x > \text{threshold} \end{cases} \tag{2}$$

Where x are inputs and w are weights

Perceptron #4

$$b \equiv -threshold$$

Perceptron #4

$$b \equiv -threshold$$

$$output = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{3}$$

Where x are inputs and w are weights

Multi Layered Perceptron

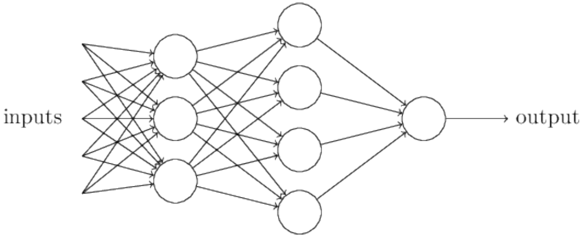


Figure: Multi Layered Perceptron

Sigmoid neuron #1

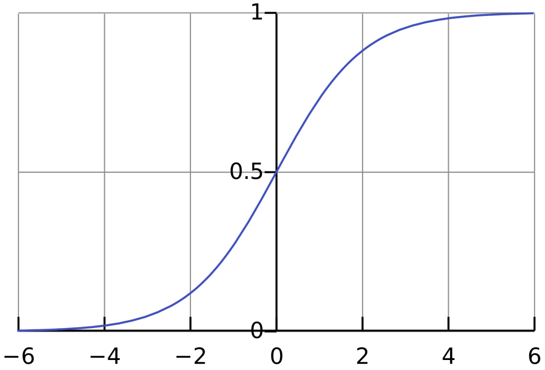


Figure: sigmoid/logistic function

Sigmoid neuron #2

$$\sigma(z) \equiv \frac{1}{1+e^{-z}} \tag{4}$$

Sigmoid neuron #2

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{4}$$

$$\sigma(z) \equiv \frac{1}{1 + exp(-\sum_j w_j x_j - b)} \tag{5}$$

Where x_j are inputs, w_j are weights and b is bias

Sigmoid neuron #3

$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b, \tag{6}$$

Softmax neuron #1

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}, \tag{7}$$

Figure: *Softmax* equation, where a_j^L is activation of the *jth* output neuron and z_j^l the weighted input for neuron j in layer L

Other activation functions

- tanh
- ReLU
- Softsign
- PReLU
- ArcTan
- ELU
- many, many more...

Cost functions #1

To quantify how well we're achieving this goal we define a *cost function*

Cost functions #1

To quantify how well we're achieving this goal we define a *cost function*

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{8}$$

Figure: *Mean squared error* - MSE (*quadratic* cost function), where $y(x)$ is desired output and a is the vector of outputs from the network when x is input

Cost functions #3

$$C(w,b) = -\frac{1}{n}\sum_{x}\left[y \ln a + (1-y)\ln(1-a)\right], \tag{9}$$

Figure: *Cross-entropy* cost function, where y is desired output and a is the vector of outputs from the network when x is input

MLP architecture #1

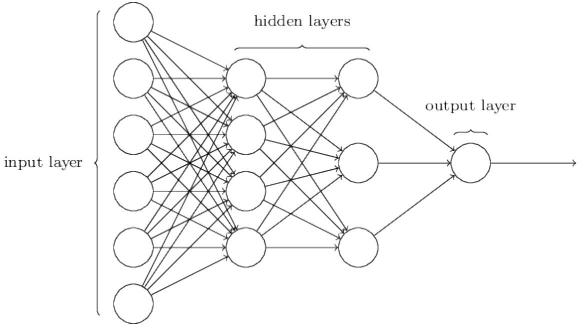


Figure: Usual Multi Layered Perceptron architecture

MLP architecture #2

$$O = F(F(w_1 \cdot x + b_1) \cdot w_2 + b_2) \tag{10}$$

Figure: Usualy found in reasearch papers

Training #1

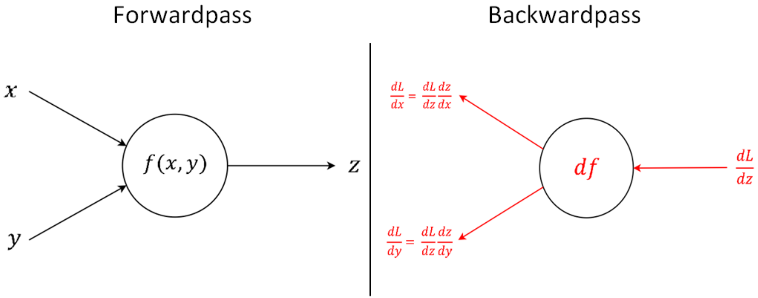


Figure: Forward pass and backpropagation for single gate

Training #2

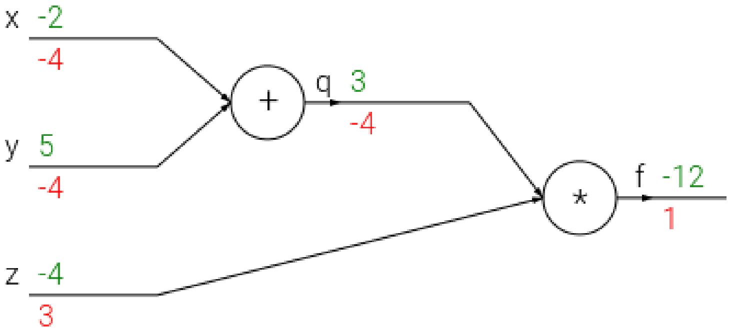


Figure: Example of forward pass and backpropagation for two simple gates, where green color indicates input to the network and red color is the gradiant flowing backwards

Training #3

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \tag{11}$$

Training #3

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2 \tag{12}$$

$$\tag{13}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \tag{14}$$

$$\tag{15}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \tag{16}$$

$$\tag{17}$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \tag{18}$$

Training #4

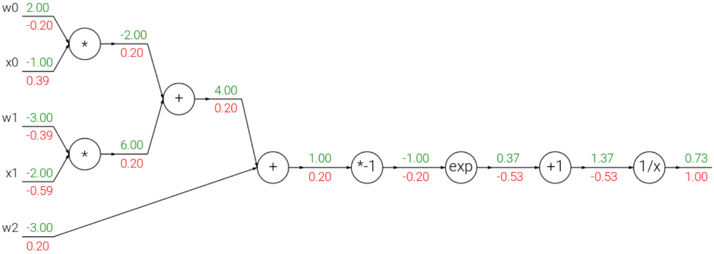


Figure: Computational graph of simple sigmoid

Training #5

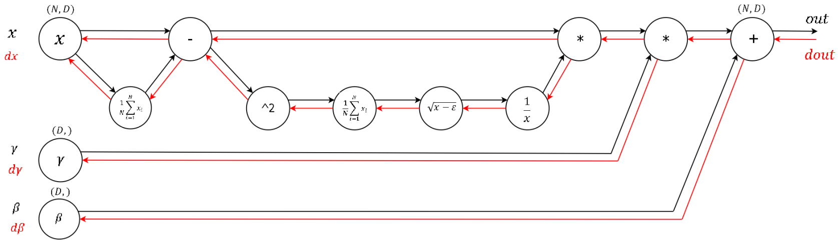
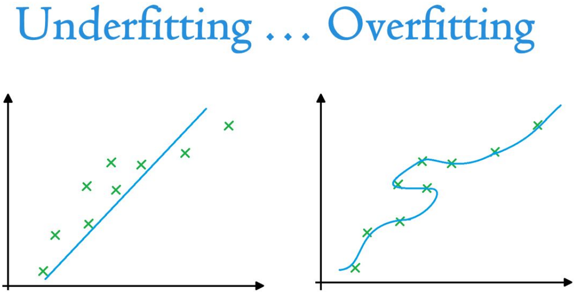


Figure: More complex computational graph

Underfitting & Overfitting #1



Underfitting & Overfitting #2

Test error suddenly goes up...

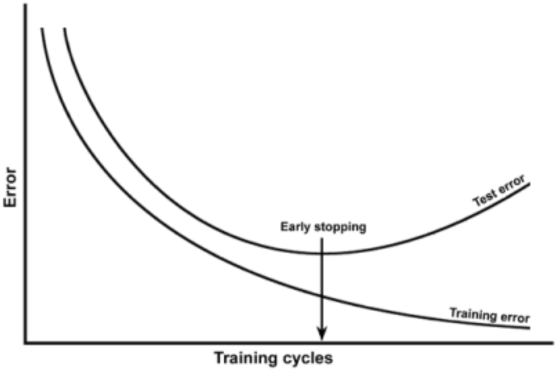


Figure: Overfitting with test and train error

Regularization #1

Are there methods that can prevent overfitting?

Regularization #1

Are there methods that can prevent overfitting?
Regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{19}$$

Figure: L2 regularization

$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \tag{20}$$

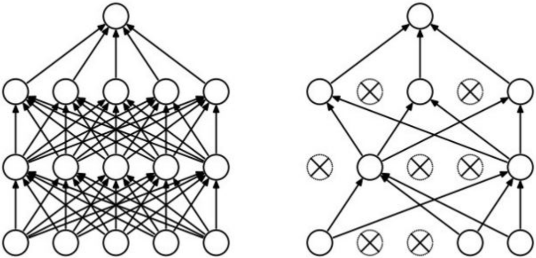
Figure: L1 regularization, where C_0 is a cost function, w are weights and λ is regularization parameter

Regularization #2

There are other ways to prevent overfitting:

Regularization #2

There are other ways to prevent overfitting:
Dropout



(a) Standard Neural Net

(b) After applying dropout.

References

- http://neuralnetworksanddeeplearning.com/chap1.html
- http://neuralnetworksanddeeplearning.com/chap2.html
- http://neuralnetworksanddeeplearning.com/chap3.html
- http://karpathy.github.io/neuralnets/
- http://cs231n.github.io/optimization-2/
- IRC server *freenode.net* - channel *#naiveneuron*

Next week

- Training schemes/methods(SGD, RMSProp, AdaGrad, Adam...) + Gentle intro to Convolutional Neural Networks.
- Let's do this collaboratively