Machine Learning,
An introduction 2:

Scoring and Optimizing Neural Networks

Part 1

This lesson is based on the Standford Course:
CS231n: Convolutional Neural Networks for Visual Recognition

Lecture 3

http://cs231n.stanford.edu/syllabus.html

Questions from last time

- Size of NN:
  - One Input layer
  - One Output layer
  - For linearly separable problems, no NN required.
  - Rule of thumb:

    a =2-10 (arbitrary)

    N_h = Number of hidden neurons

    N_i = input...

    N_o = output...

$$N_h = \frac{N_S}{\left(a * \left(N_i + N_o\right)\right)}$$

or

$$N_h = \sqrt{\left(N_i * N_o\right)}$$

Or something between input and output neurons

Questions from last time

- Size of NN:
  - *In general:*
    - *Hidden Neurons between the size of the input layer and output layer*
    - *Too large a NN will memorize the dataset (a kind of overfitting)*
    - *Too small a NN will not be able to approximate the model*
    - *At the end of the day: trial and error / experience / recommended architectures*
    - *Alternative: automatic layer selection with genetic algorithms/ other automatic architecture generation*

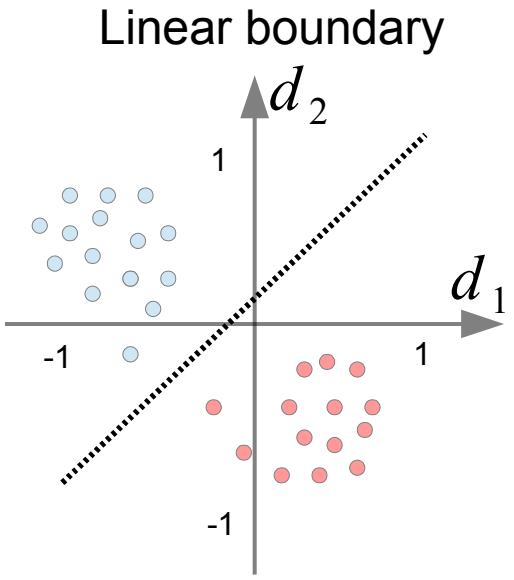
Questions from last time

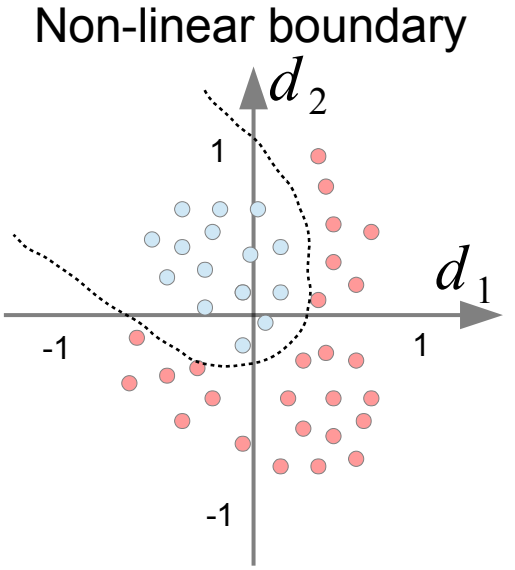
- Size of NN:
  - Number of layers*:
    - Zero: linearly seperable
    - One: can approximate any function that contains a continuous mapping from one finite space to another
    - Two: Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
  - In short:
    - The more layers the more complex boundaries can be modelled

* from: https://web.archive.org/web/20140721050413/http://www.heatonresearch.com/node/707

Questions from last time

- Size of NN:



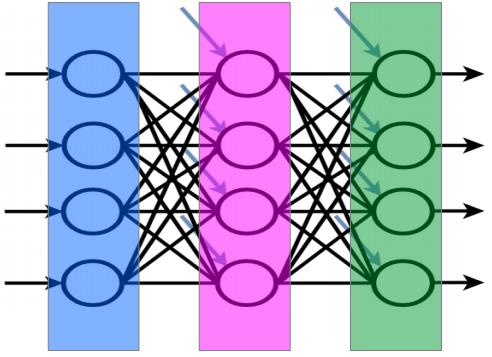


Questions from last time

- Size of NN:
  - This is only applicable to simple NN, Convolutional NN or specialized architectures are sometimes exempt from these rules.

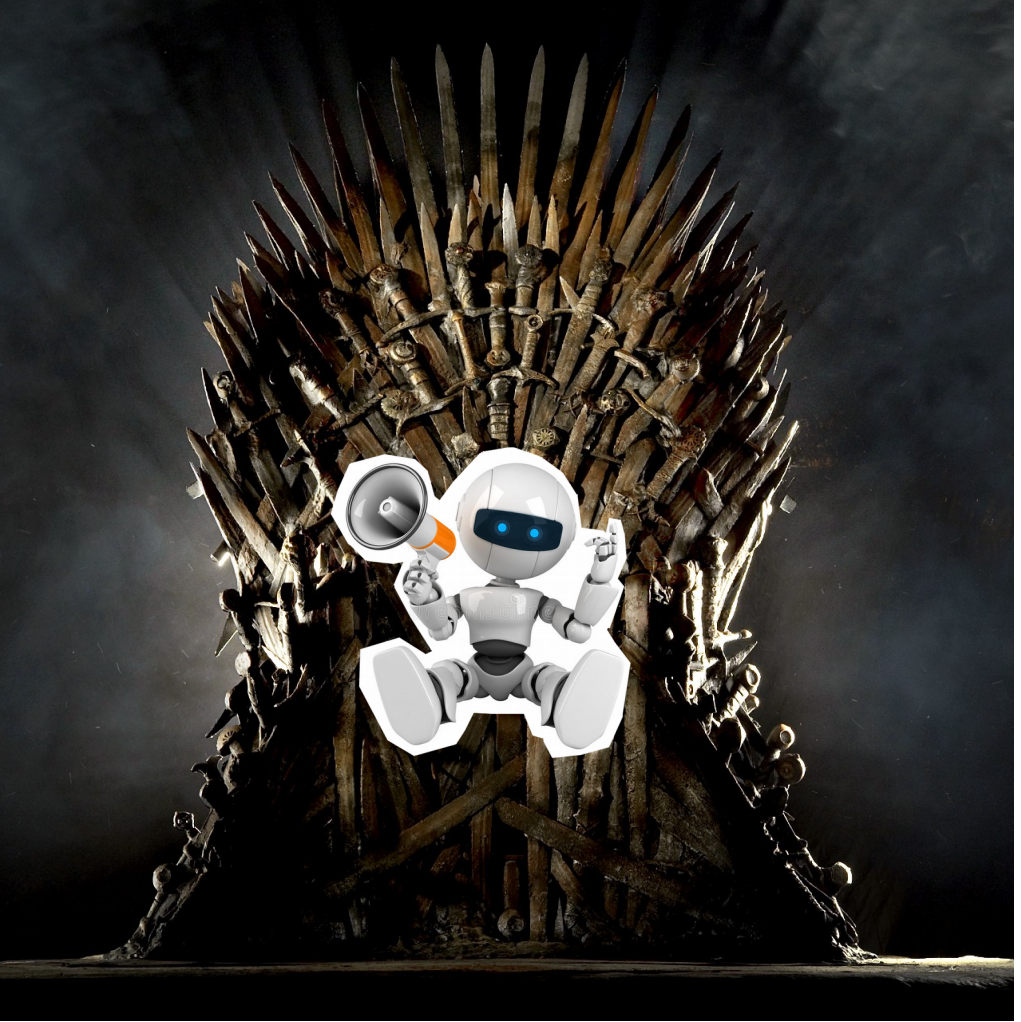
Questions from last time

- Size of NN:
  - One last concern:
    - Layer width (number of neurons in layer) easier to increase in regard to computational time due to parallelization. (We'll see how this works later)
    - Layer depth (number of layers) harder to increase due to serial processing



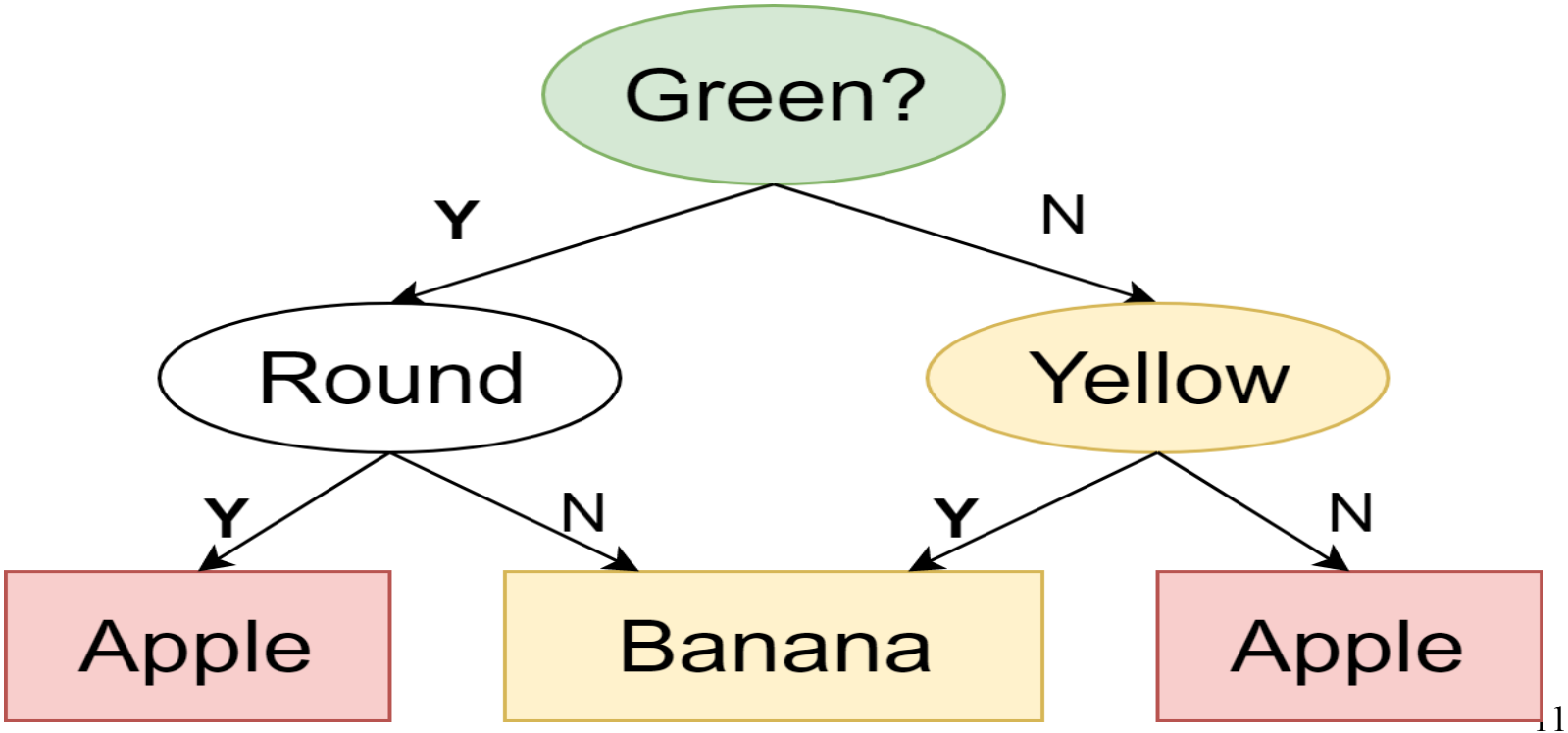
Questions from last time

- Bias:
  - We'll see next time :)
- Mean-Squared Error:
  - Used to calculate the **score** for a run of the NN. We'll talk about scores in a bit.

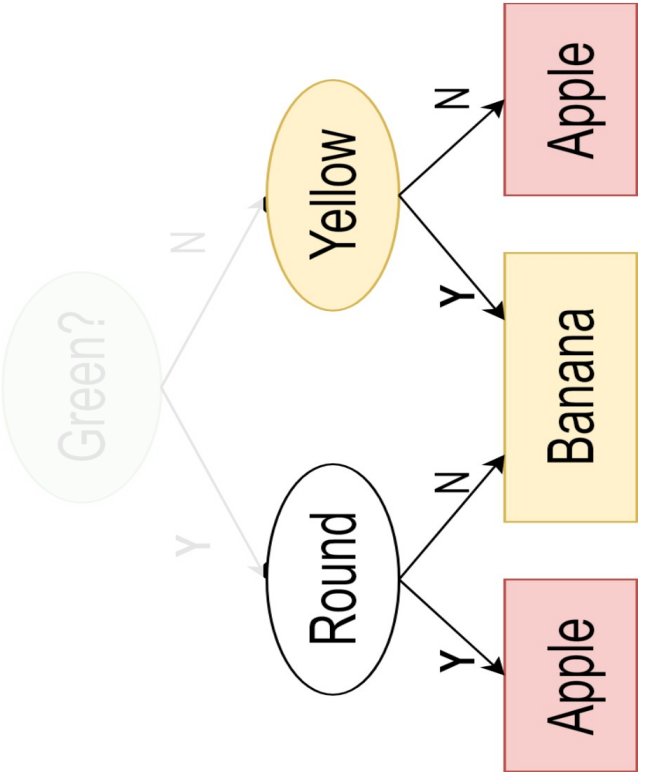


Previously on
~~Game of Thrones~~
Introduction to
Machine Learning

Banana or Apple?

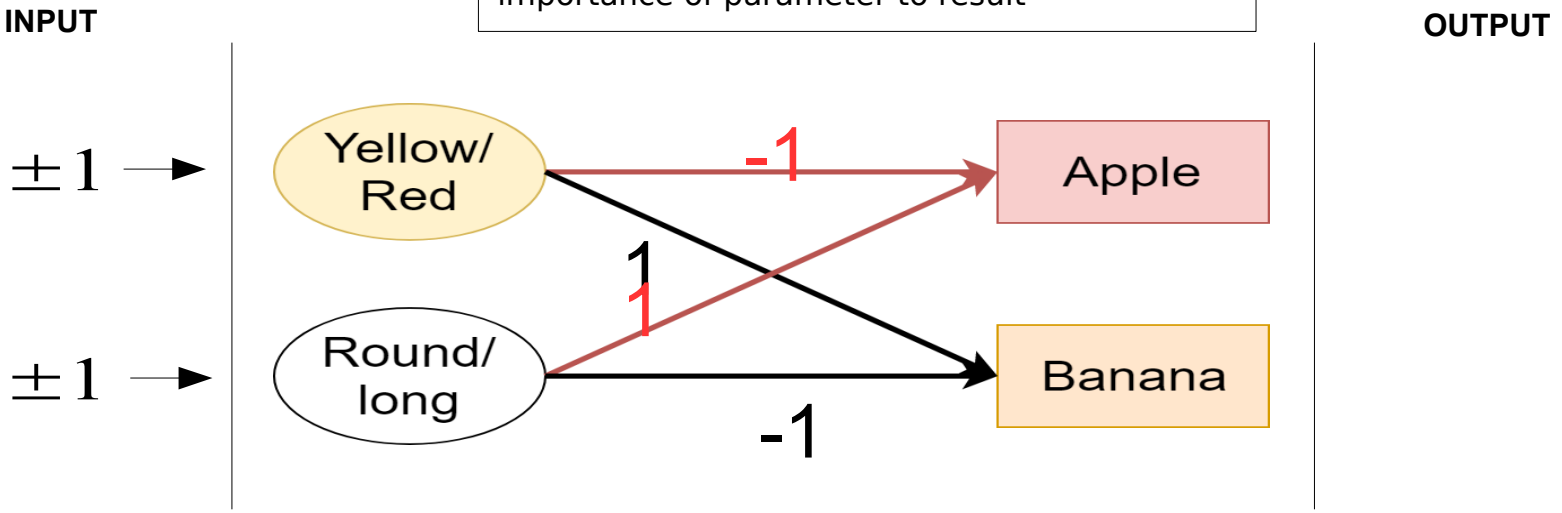


Banana or Apple?



Banana or Apple?

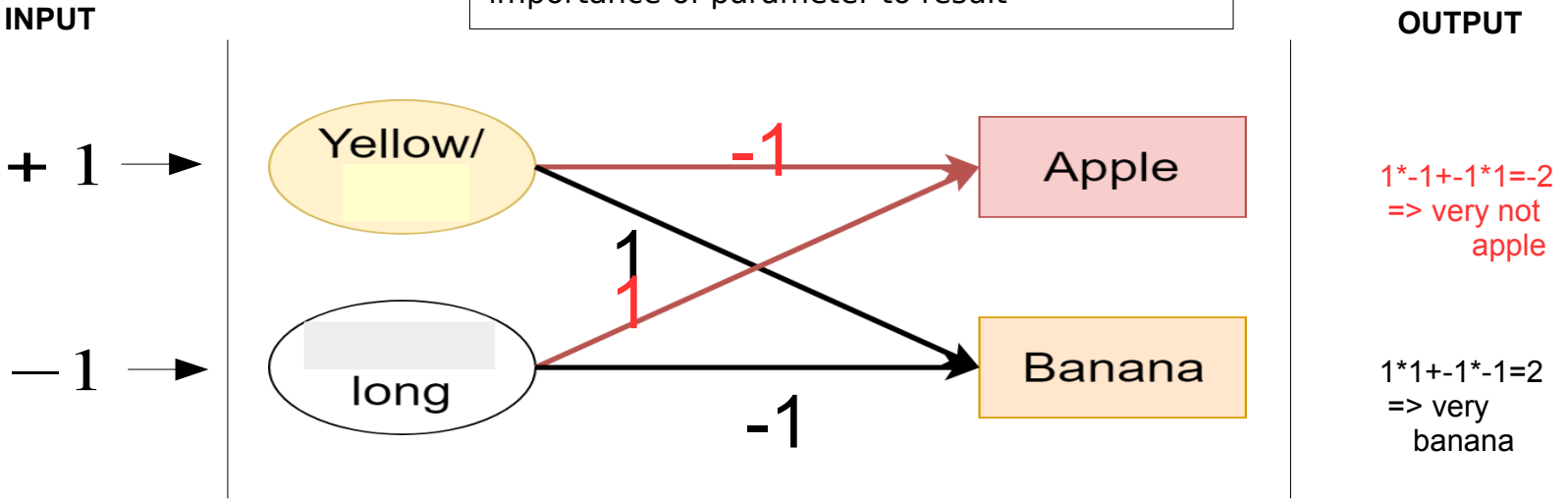
Strength of connection (Weight) shows importance of parameter to result



+1 is Yellow or Round like before,
-1 is Green or Long,
0 is lack of data.

Banana or Apple?

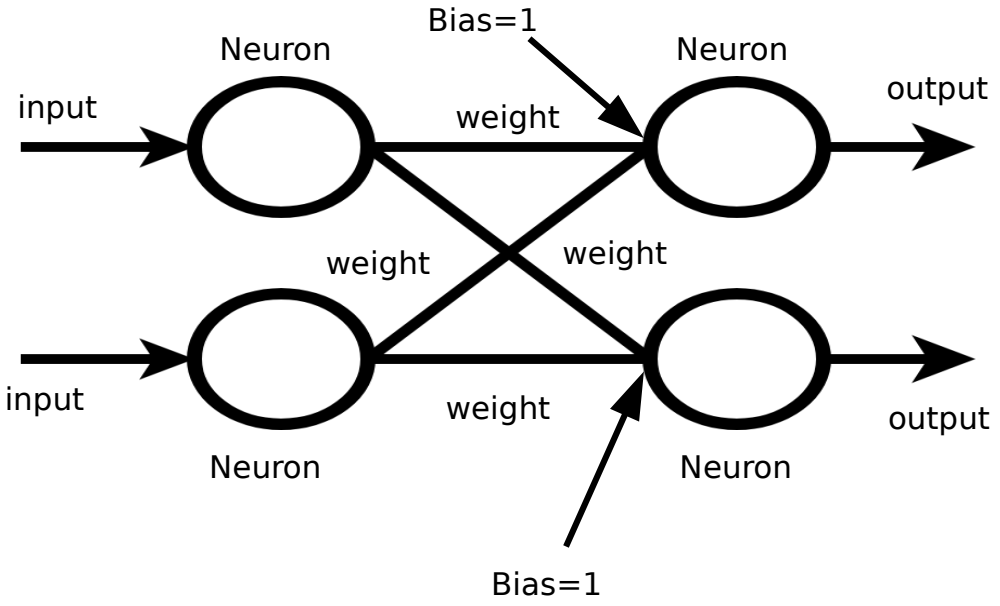
Strength of connection (Weight) shows importance of parameter to result



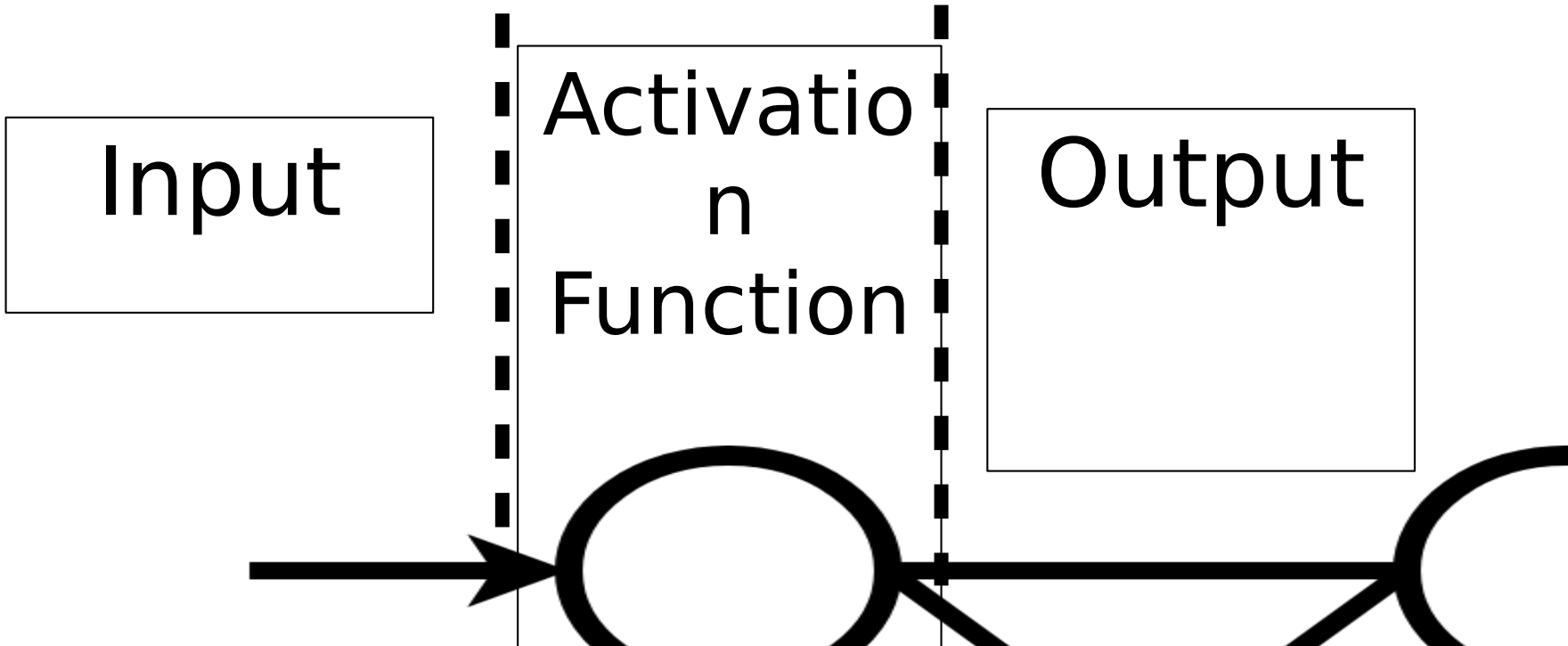
+1 is Yellow or Round like before,
-1 is Green or Long,
0 is lack of data.

Apple = Yellow * 0 + Round *1
Banana = Yellow * 1 + Round *0

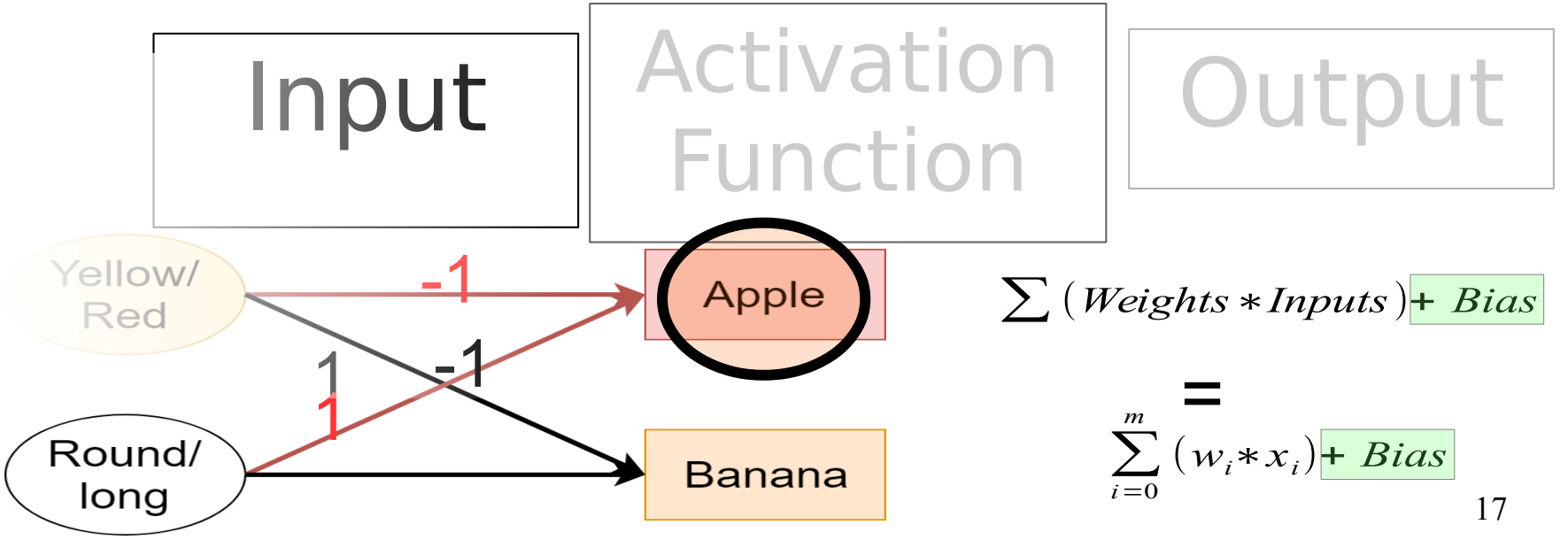
Multilayer Perceptrons



Multilayer Perceptrons: Neuron Components



Multilayer Perceptrons



Multilayer Perceptrons

*Image credits here

Input

Activation Function

$f\left(Inputs\right)$

Output

How about we cut and squeeze the output into a {0,1} range?

For x>0 : f(x) = x/2

For x<0 : f(x) = 0

Apple = -2 → =0

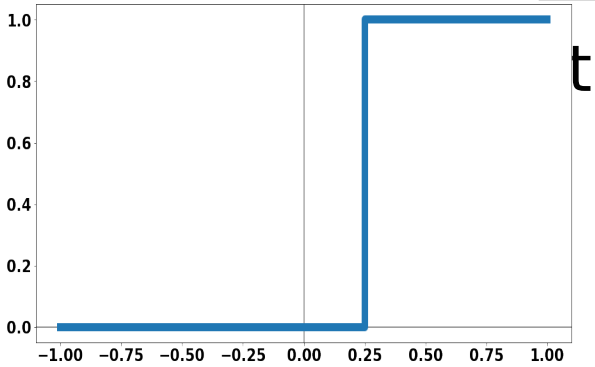
Banana = 2 → =1

Multilayer Perceptrons

Input

Activation Function

Output



tep function (Heaviside)

$$Bias = 1 * w_b$$

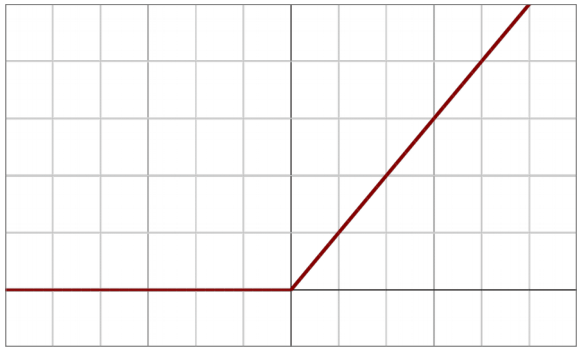
$$f(x) = \begin{cases} 0 \text{ for } x \leq 0 + 0.25 \\ 1 \text{ for } x > 0 + 0.25 \end{cases}$$

Multilayer Perceptrons

Input

Activation Function

Output



RELU (Rectified Linear Units)

$$f(x) = \begin{cases} 0 \text{ for } x \leqslant 0 \\ x \text{ for } x > 0 \end{cases}$$

1*-1+-1*1=-2
=> very not Apple
=> f(x)=0

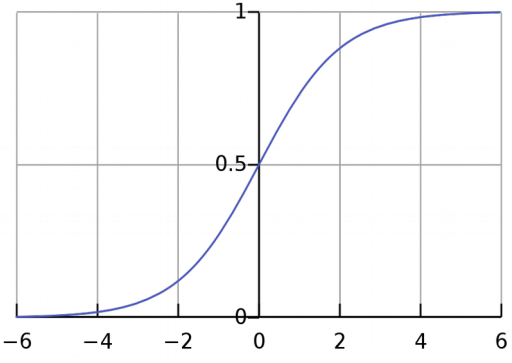
1*1+-1*-1=2
=> very Banana
=> f(x)=2

Multilayer Perceptrons

Input

Activation Function

Output



Sigmoid function (logistic function)

$$f(x) = \frac{1}{1 + e^{-x}}$$

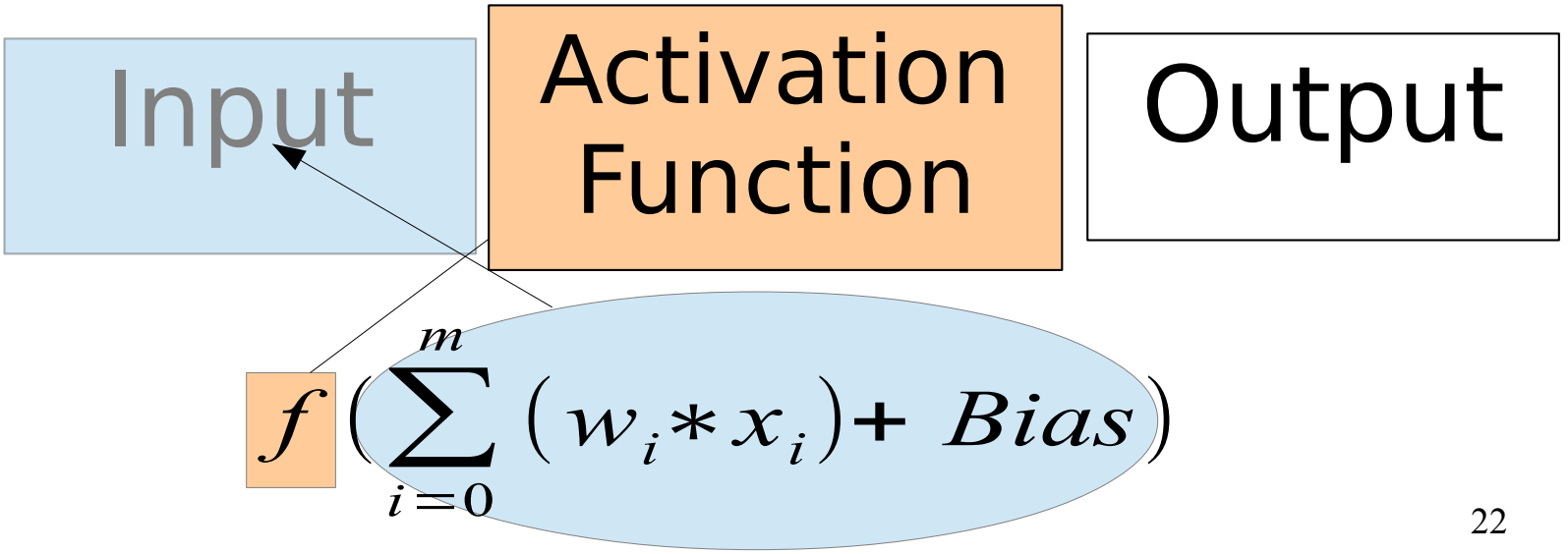
1*-1+-1*1=-2
=> very not Apple
=> f(x)=0.1192

1*1+-1*-1=2
=> very Banana
=> f(x)=0.8807

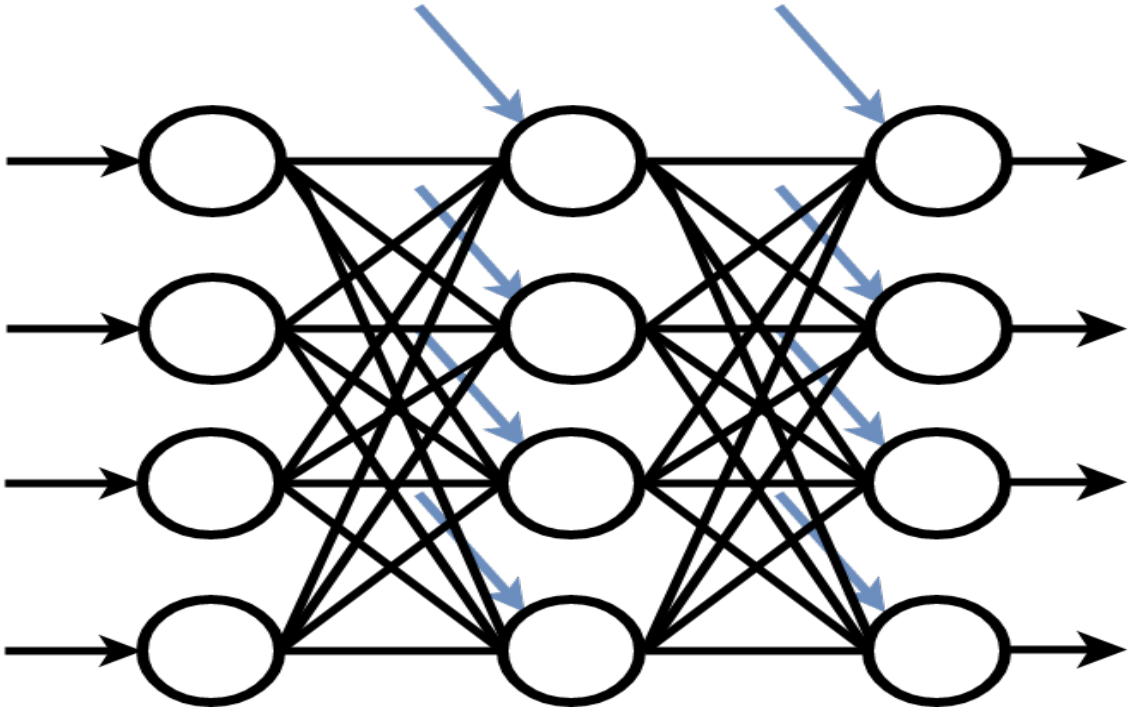
...aka Softstep..

...aka Logistic Curve..

Multilayer Perceptrons



Multilayer Perceptrons

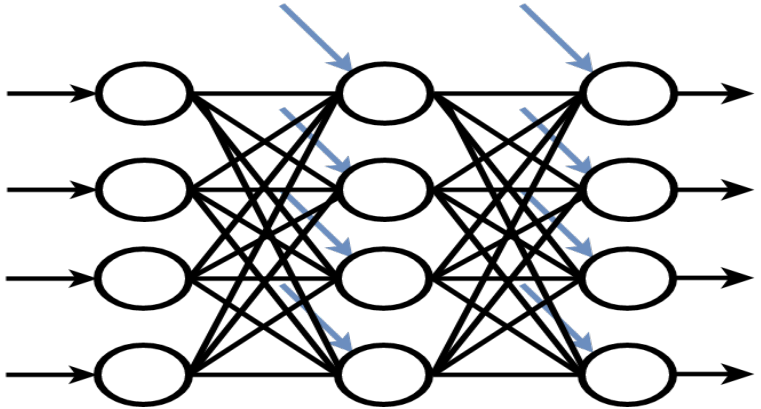


Where's the Advantage ?

Multilayer Perceptrons

cough cough
Forgot to write the weights last time...

$$f\left(\sum_{j=0}^{m} W_{i_2, j_2} * f\left(\sum_{i=0}^{m} w_{i,j} * x_{i,j} + Bias\right) + Bias\right)$$

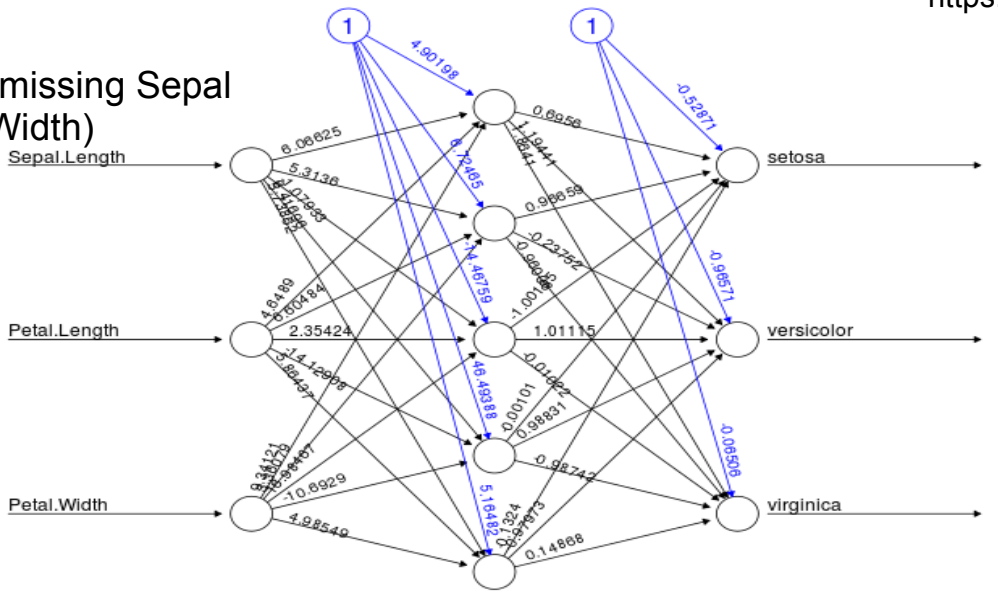


Where's the Advantage?

Multilayer Perceptrons

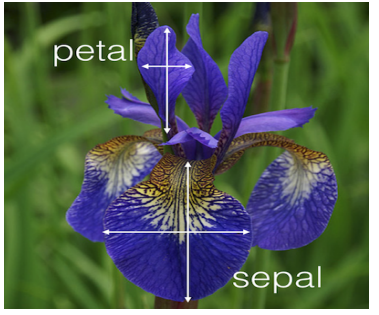
https://archive.ics.uci.edu/ml/datasets/Iris/

(missing Sepal Width)



http://www.learnbymarketing.com/tutorials/neural-networks-in-r-tutorial/

Famous Iris dataset which measured the Different Length of Iris petals and classifies them as one of 3 Species

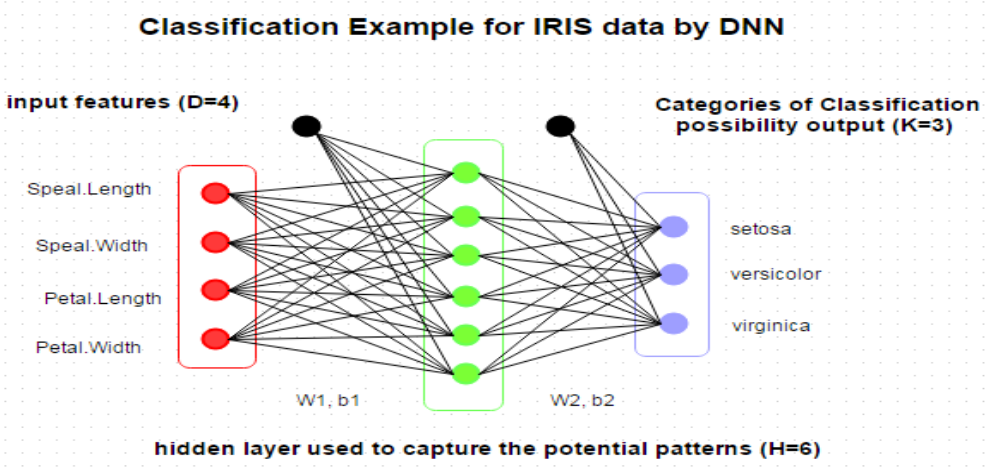


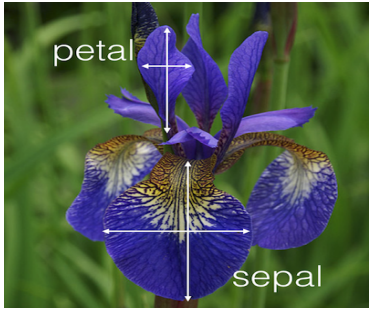
http://blog.kaggle.com/2015/04/22/scikit-learn-video-3-machine-learning-first-steps-with-the-iris-dataset/

Multilayer Perceptrons

https://archive.ics.uci.edu/ml/datasets/Iris/

Famous Iris dataset which measured the Different Length of Iris petals and classifies them as one of 3 Species





https://stats.stackexchange.com/questions/268202/backpropagation-algorithm-nn-with-rectified-linear-unit-relu-activation

http://blog.kaggle.com/2015/04/22/scikit-learn-video-3-machine-learning-first-steps-with-the-iris-dataset/

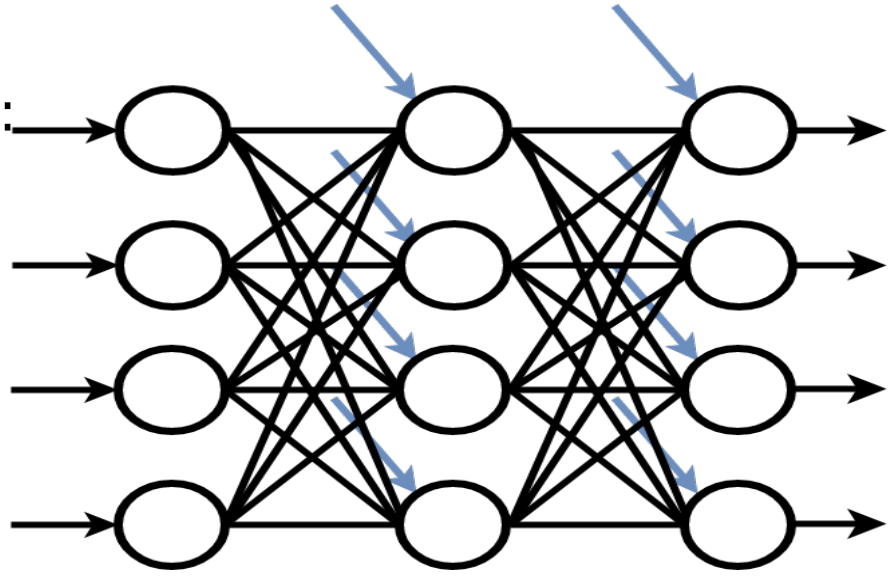
Today:

- Some Terminology
- Scoring!
- Loss function
- Regularization
- Gradient Descent

Type of Neural Network

- For now: Feedforward!
  Data flows in 1 direction:

- These are called
  Fully-Connected-Layers
  (FC-layer)
  Or “Dense” layers



Glossary

- Feature: a parameter
- Example: a collection of features

  e.g. Example: A picture, features: each pixel
- Score: prediction results of the model (NN, or some classifier) on the data
- Loss function: a measure of the 'badness' of the score compared to the actual result
- Optimizer: a method of minimizing the loss function

Glossary

- Score: prediction results of the model (NN, or some classifier) on the data

  [-2,2]==[apple, banana]

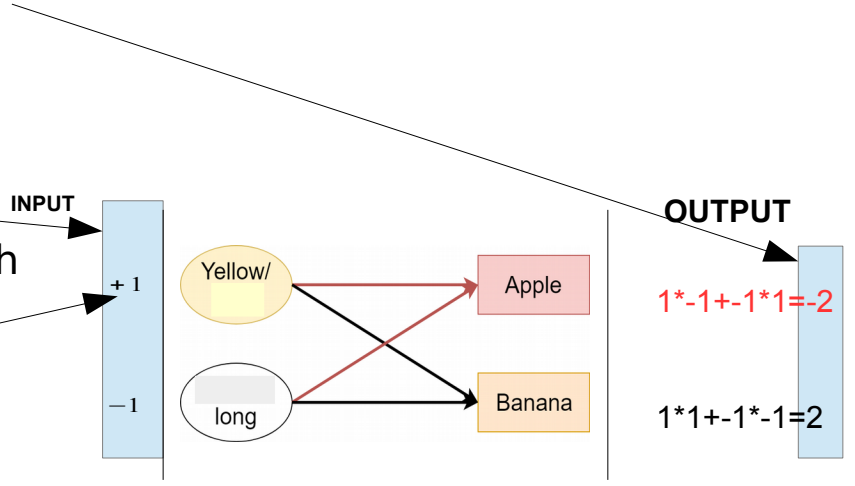
- Example: a collection of features

  e.g. Example: A picture, features: each pixel

  (Yellow & Long)

- Feature: a parameter

  (Yellow/Red)



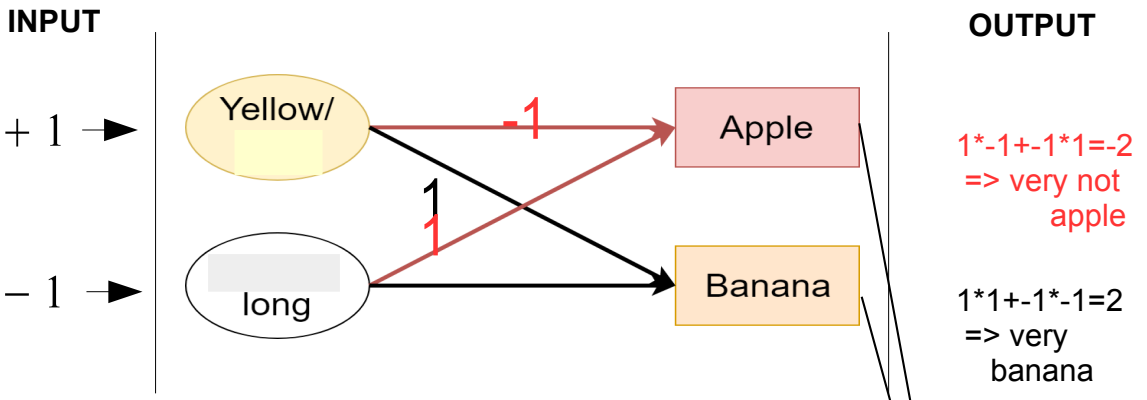
Types of Learning

- Supervised: You have data which includes inputs(features) AND outputs. i.e. values for for x and y in y(x)=bx
- Unsupervised : You have data which includes ONLY the inputs (features) i.e. values for x in y(x)=?

Some types of problems for Supervised Learning:

- Classification: pick a class from the features (Apple/Banana)
- Regression: numerical value from features (Value of Stock Market tomorrow)
- Transcription: input to text (example OCR)

Previously: We used a Linear Classifier



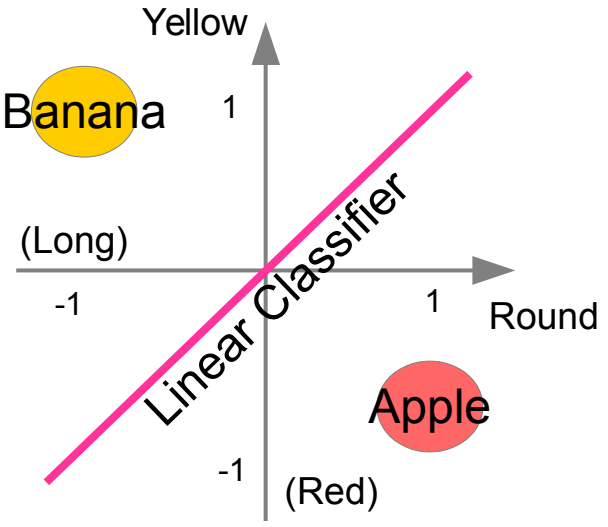
$$f(x, W) = W * x + b$$

Classifier 1: Apple
Classifier 2: Banana

Linear function: .... it's a line!

$$f(x, W) = W * x + b$$

Aka a polynomial function of 1st or 0th degree



How to quantify how good or bad a Neural Network/ ML model is [illegible]?

SCOOOOOOOOOOOOOOOOOOOOOOOOOORE!...ing

We now use our Linear Classifiers to predict 3 classes: Apple, Banana and Pear

	Prediction	Apple	Banana	Pear
Class				
Apple		**1.87**	-0.4	0.1
Banana		**1.3**	1.1	-0.3
Pear		0.56	1.35	**1.1**

$$f_{pear}(x, W) = W_{pear} * x_{pear} + b_{pear}$$

$$f_{banana}(x, W) = W_{banana} * x_{banana} + b_{banana}$$

$$f_{apple}(x, W) = W_{apple} * x_{apple} + b_{apple}$$

We have used our Linear Classifiers to predict 3 classes: Apple, Banana and Pear

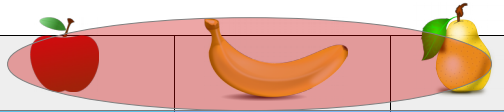
Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	1.35	**1.1**

The Scores for the Classifier. i.e. the predictions for each class

* images are in the public domain

We have used our Linear Classifiers to predict 3 classes: Apple, Banana and Pear

Dataset



Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	1.35	**1.1**

The Scores for the Classifier. i.e. the predictions for each class

Classes (labels)

* images are in the public domain

We have used our Linear Classifiers to predict 3 classes: Apple, Banana and Pear

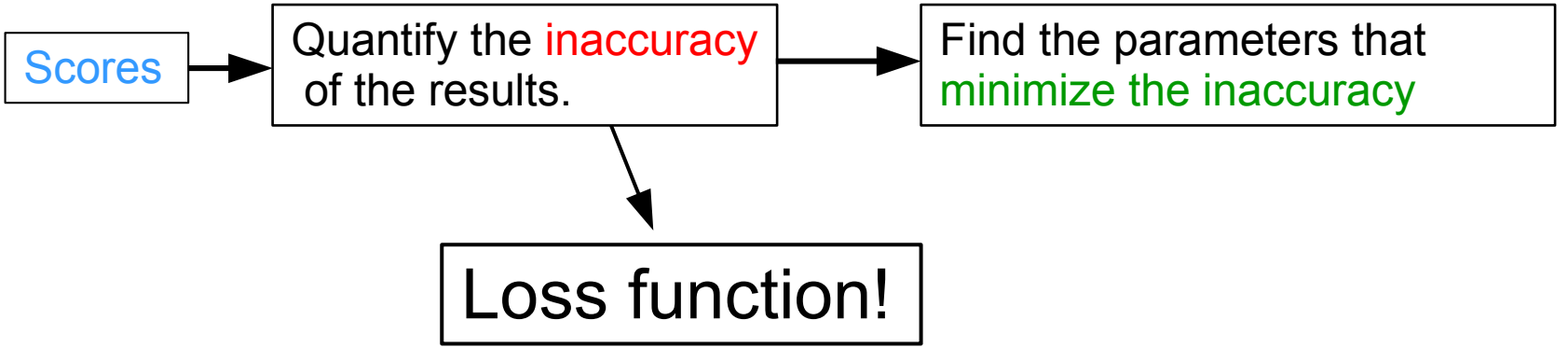
Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	1.35	**1.1**

The Scores for the Classifier. i.e. the predictions for each class

We need a way to quantify the inaccuracy of the results.

Then we need a way to find the parameters that minimize the inaccuracy of the results

We have used our Linear Classifiers to predict 3 classes: Apple, Banana and Pear



Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

Say that our dataset consists of i examples where
x_i is the ith image (example) and
y_i is the ith class label (Apple, banana, pear)

Scores

inaccuracy

minimize the inaccuracy

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

We have i examples:
so the loss should probably be the averaged sum over all examples

$$L = \frac{1}{N} \sum_i L_i$$

Say that our dataset consists of i examples where
x_i is the ith image (example) and
y_i is the ith class label (Apple, banana, pear)

Scores | inaccuracy | minimize the inaccuracy

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

Say that our dataset consists of i examples where
xi is the ith image (example) and
yi is the ith class label (Apple, banana, pear)

Scores | inaccuracy | minimize the inaccuracy

The loss should reflect the difference between the predicted value and the actual value

$$L = \frac{1}{N}\sum_i L_i$$

$$L_i = L_i\left(f\left(x_i, W\right), y_i\right)$$

classifier

Label for ith example From dataset

The predicted value f(x,W) is just the result of the classifier

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

The loss should reflect the difference between the predicted value and the actual value

$$L = \frac{1}{N} \sum_i L_i$$

$$L_i = L_i\left(f\left(x_i, W\right), y_i\right)$$

L is a function of the predicted and actual value for the ith example

f: our classifier
y_i: label for example i
L_i: Loss function for example i

Scores

inaccuracy

minimize the inaccuracy

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

The loss should reflect the difference between the predicted value and the actual value

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

f: our classifier
y_i: label for example i
L_i: Loss function for example i

L is a function of the predicted and actual value for the ith example

Scores

inaccuracy

minimize the inaccuracy

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

$$L = \frac{1}{N} \sum_i L_i$$

$$L_i = L_i(s, y_i)$$

We'll use the shorthand s for the scores: $s = f(x_i, W)$

f: our classifier
y_i: label for example i
L_i: Loss function for example i

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Scores

inaccuracy

minimize the inaccuracy

Loss Function

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

$$L_i = L_i(s, y_i)$$

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

We previously use a linear classifier, so we'll use the example of a linear SVM, which is equivalent to a NN with zero hidden layers.

The multiclass* SVM loss is defined as:

$$L_i = \sum_{i \neq y_i} \begin{cases} 0 & if\ s_{y_i} \geqslant s_i + 1 \\ s_j - s_{y_i} + 1 & otherwise \end{cases}$$

$$= \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Scores

inaccuracy

minimize the inaccuracy

*we have multiple labels, Apple, Banana etc.

Loss Function for multiclass SVM

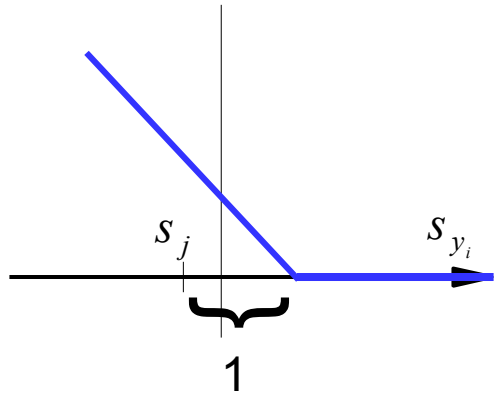
Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

$$L_i = \sum_{i \neq y_i} \begin{cases} 0 & if\ s_{y_i} \geqslant s_i + 1 \\ s_j - s_{y_i} + 1 & otherwise \end{cases}$$

$$= \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

$$L_i = L_i(s, y_i)$$

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$



Scores

inaccuracy

minimize the inaccuracy

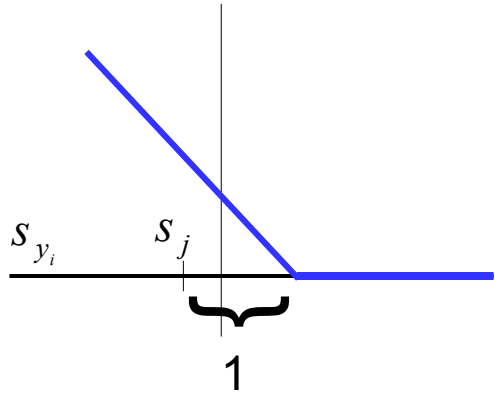
Loss Function for multiclass SVM

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**

This is called the **'Hinge' loss**:
It increases linearly as soon as the difference between the predicted and label score is higher than 1.
We'll see what that means on the next slide

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$



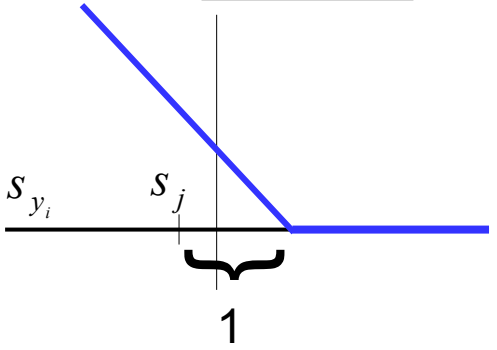
Scores

inaccuracy

minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	1.3	1.1	-0.3
Pear	0.56	**1.35**	**1.1**



For the first example (Apple):

$$L_i = \sum_{j \neq y_i} max\left(0, s_j - s_{y_i} + 1\right)$$

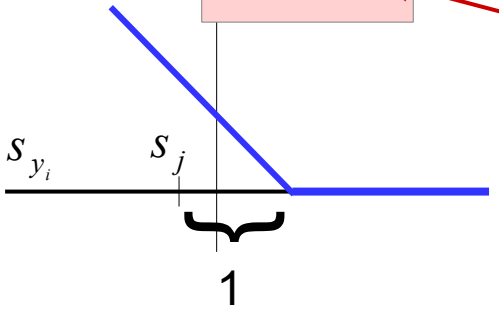
= max (0,1.3-1.87+1)
+max (0,0.56-1.87+1)
=max(0,0.33)+
max(0,-0.31)
=0.33+0
=0.33

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33		



For the first example (Apple):

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

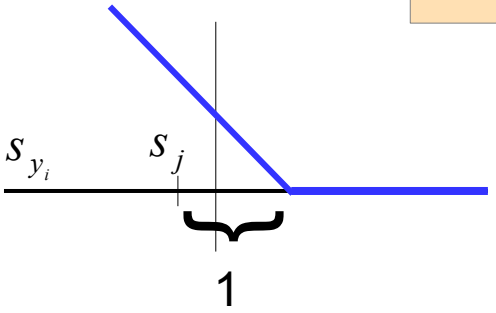
= max (0,1.3-1.87+1)
+max (0,0.56-1.87+1)
=max(0,0.33)+
max(0,-0.31)
=0.33+0
=0.33

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33		



Scores | inaccuracy | minimize the inaccuracy

For the second example (Banana):

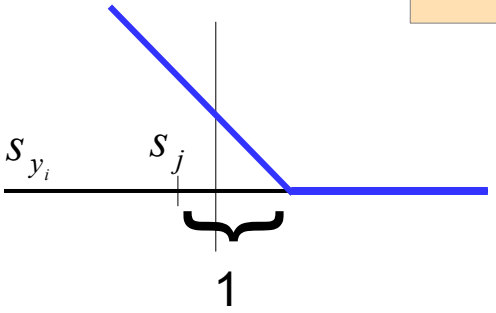
$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

= max (0,?+1)
+max (0,?+1)

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33		



For the second example (Banana):

$$L_{ban} = \sum_{j \neq y_{ban}} max\left(0, s_j - s_{y_{ban}} + 1\right)$$

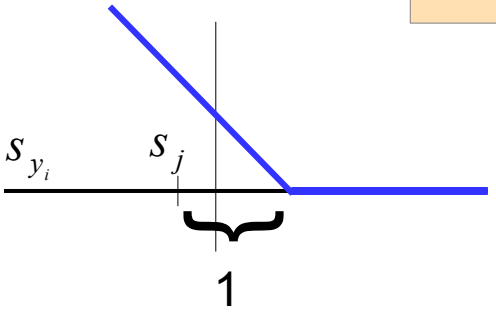
= max (0,-0.4-1.1+1)
+max (0,?+1)

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33		



For the second example (Banana):

$$L_{ban} = \sum_{j \neq y_{ban}} max\left(0, s_j - s_{y_{ban}} + 1\right)$$

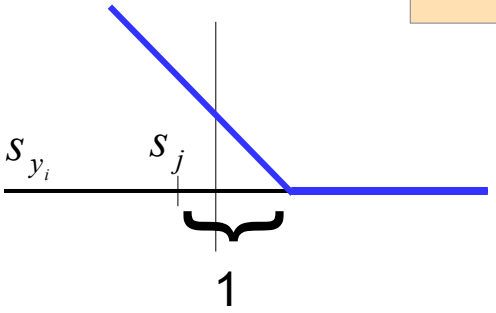
= max (0,-0.4-1.1+1)
+max (0,1.35-1.1.+1)

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33		



For the second example (Banana):

$$L_{ban} = \sum_{j \neq y_{ban}} max\left(0, s_j - s_{y_{ban}} + 1\right)$$

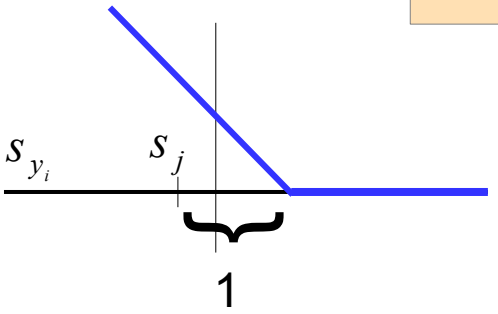
= max (0,-0.4-1.1+1)
+max (0,1.35-1.1.+1)

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33	1.25	



For the second example (Banana):

$$L_{ban} = \sum_{j \neq y_{ban}} max\left(0, s_j - s_{y_{ban}} + 1\right)$$

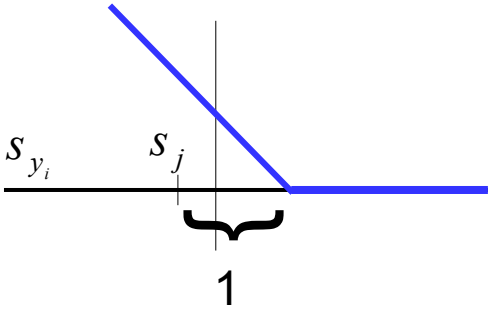
= max (0,-0.4-1.1+1)
+max (0,1.35-1.1.+1)
=max(0,-0,4)
+max(0,1.25)
=0+1.25
=1.25

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33	1.25	0



For the third example (Pear):

$$L_{pear} = \sum_{j \neq y_{pear}} max(0, s_j - s_{y_{pear}} + 1)$$

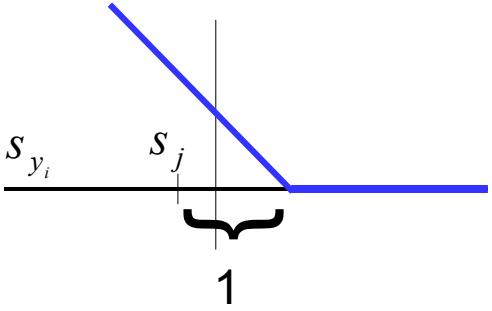
= max (0,0.1-1.1+1)
+max (0,-0.3-1.1.+1)
=max(0,0)
+max(0,-0.4)
=0+0
=0

The +1 means the loss function penalizes a difference of less than 1 between the correct score and the predicted score

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33	1.25	0



What if the Weights and therefore scores were doubled?

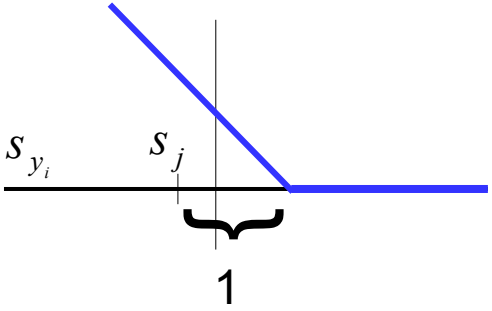
$$L_{pear} = \sum_{j \neq y_{pear}} max(0, s_j - s_{y_{pear}} + 1)$$

= max (0,0.1-1.1+1)
+max (0,-0.3-1.1.+1)
=max(0,0)
+max(0,-0.4)
=0+0
=0

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.2
Banana	**1.3**	1.1	-0.6
Pear	0.56	**1.35**	**2.2**
Loss	0.33	1.25	0



What if the Weights and therefore scores were doubled?

$$L_{pear} = \sum_{j \neq y_{pear}} max(0, s_j - s_{y_{pear}} + 1)$$

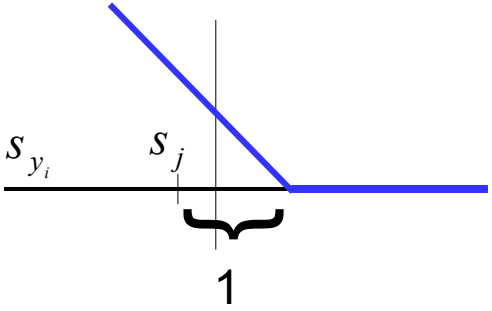
= max (0,0.2-2.2+1)
+max (0,-0.6-2.2.+1)
=max(0,-2)
+max(0,-1.8)
=0+0
=0

It's the same score!
So any multiples of weights that result in 0 will still result in zero.

Scores | inaccuracy | minimize the inaccuracy

Loss Function for multiclass SVM: 'Hinge' Loss

Class\Pred.			
Apple	**1.87**	-0.4	0.2
Banana	**1.3**	1.1	-0.6
Pear	0.56	**1.35**	**2.2**
Loss	0.33	1.25	0



What if the Weights and therefore scores were doubled?

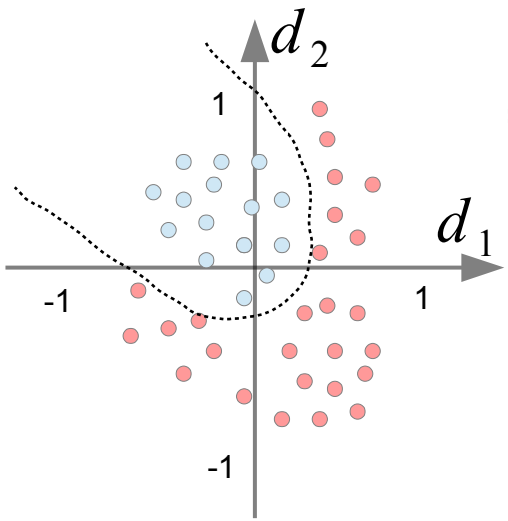
This could result excessively large weights

It's the same score!
So any multiples of weights that result in 0 will still result in zero.

Scores | inaccuracy | minimize the inaccuracy

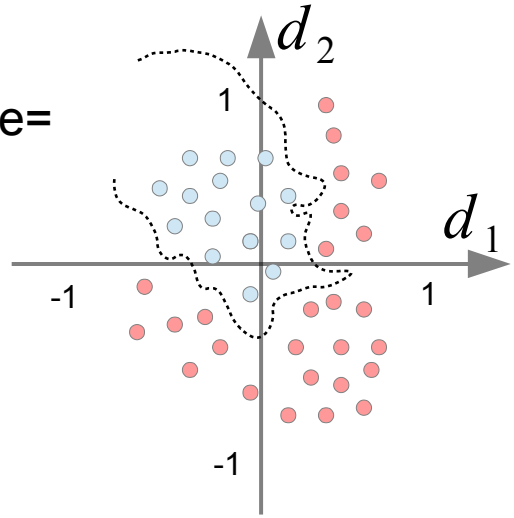
Loss Function for multiclass SVM: 'Hinge' Loss

Remember:
Non-linear boundary



=Same score=

Large Weights:
Overfitting



Scores | inaccuracy | minimize the inaccuracy

Loss Function Regularization

Solution: Regularization!

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Scores | inaccuracy | minimize the inaccuracy

Loss Function Regularization

Solution: Regularization!

$$L(W) = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

λ Is a scaling parameter (how strong do we want The regularization to be)

$R(W)$ Is a function of the weights

Scores | inaccuracy | minimize the inaccuracy

Loss Function Regularization

Solution: Regularization!

$$L(W) = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

Some regularization functions:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Loss increases with increasing Weights, keeping weights low

Dropout (removing weights)

Other...

Scores

inaccuracy

minimize the inaccuracy

Other Loss Functions

- Mean Squared Loss (SVMs)
- Softmax Loss (Logistic function that normalizes loss)

  Use Is as always problem specific
- L2 loss (not the same as L2 regularization)
- L1 loss (not the same as L2 regularization)
- Etc...

Optimization!

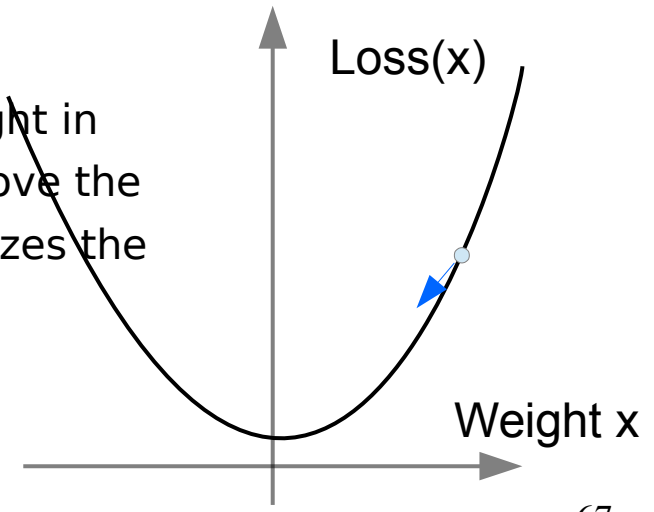
- Now that we can quantify how good our model is, we try to find a method which allows us to change the weights so that the loss function is minimized

Optimization!

- Now that we can quantify how good our model is, we try to find a method which allows us to change the weights so that the loss function is minimized:
  - Random Search?
    - We keep randomizing all the weights, and keep the best configuration.
    - Not a good method...

Optimization!

- Now that we can quantify how good our model is, we try to find a method which allows us to change the weights so that the loss function is minimized:
  - Gradient Descent!
    - We calculate the slope for each weight in regards to the loss function, then move the weight in the direction which minimizes the loss



Optimization via Gradient Descent

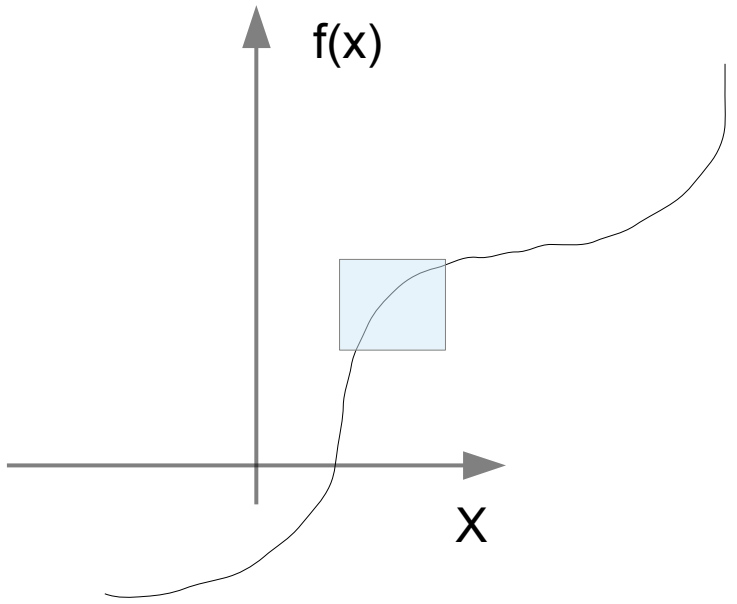
- The derivative of a function in 1 dimension is

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

The Gradient is the vector of all the partial derivatives for each dimension [df(x1,x2,x3...)/dx1, df(x1,x2,x3...)/dx2...]

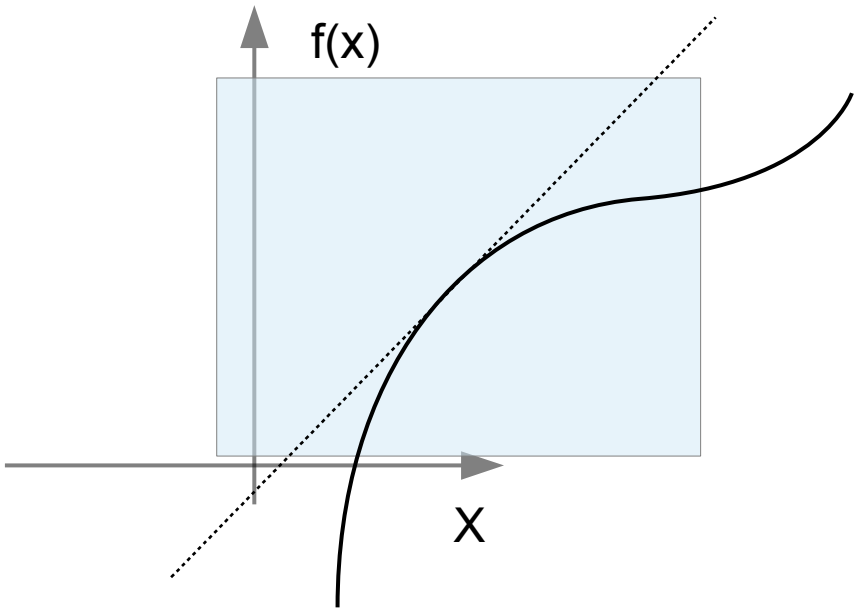
the negative gradient shows the direction that minimizes the function

Optimization via Gradient Descent



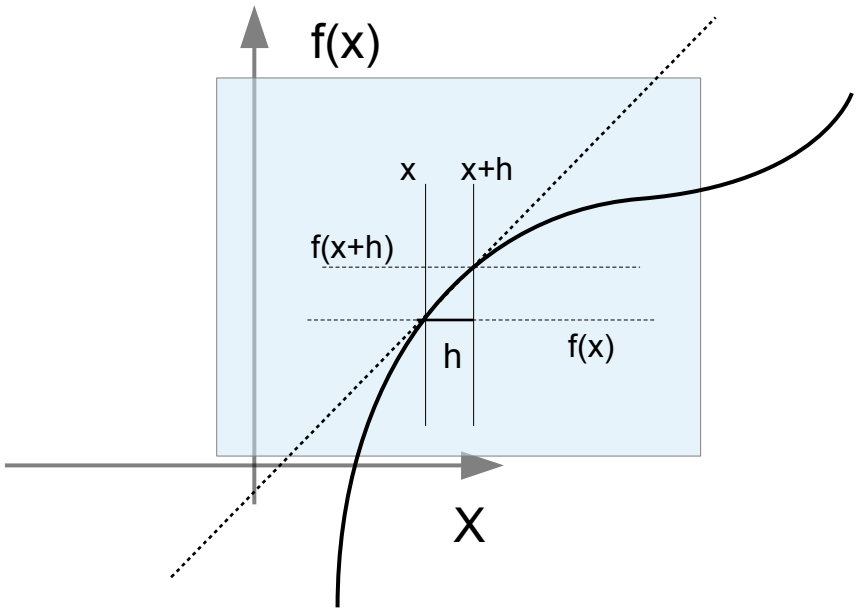
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Optimization via Gradient Descent



$$Slope = \frac{vertical\ change}{horizontal\ change}$$

Optimization via Gradient Descent



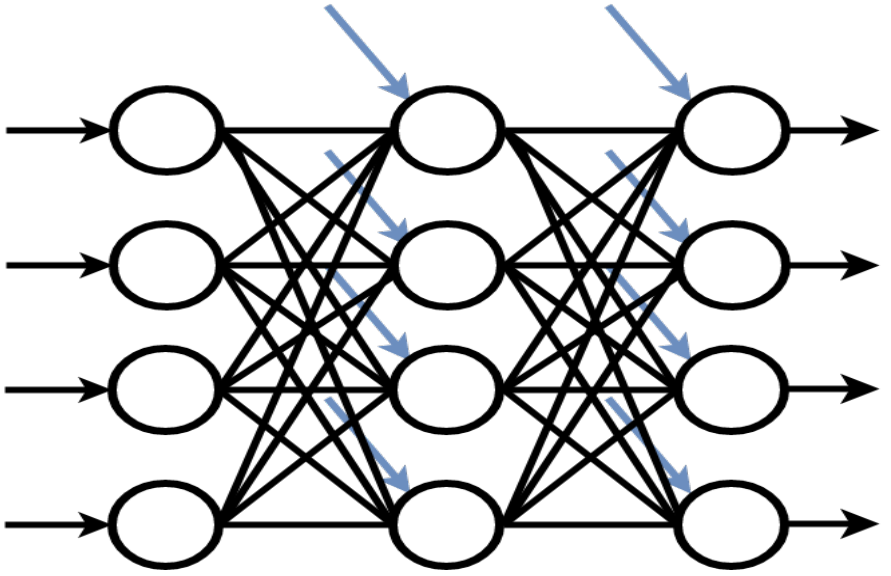
$$Slope = \frac{vertical\ change}{horizontal\ change}$$

$$Slope = \frac{f(x+h) - f(x)}{h}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

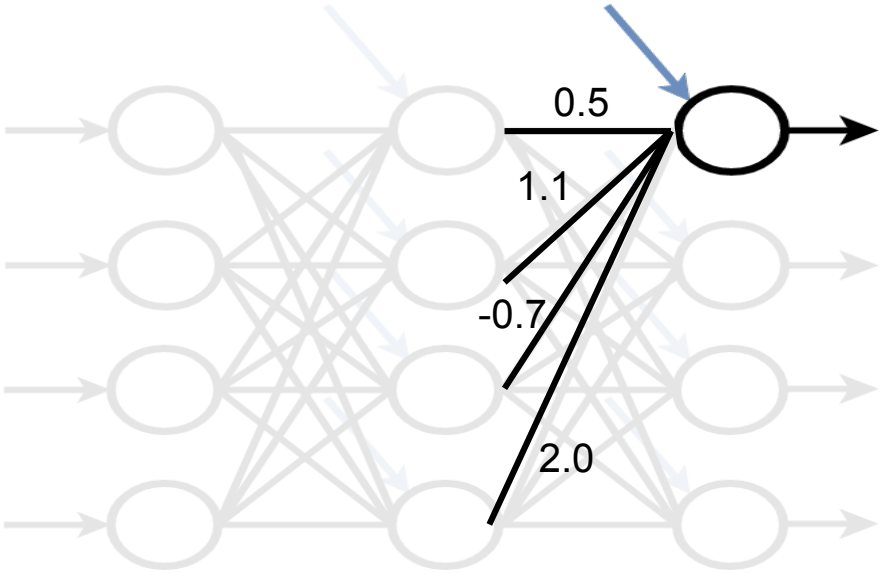
Optimization via Gradient Descent

Current W



Optimization via Gradient Descent

Current W
0.5
1.1
-0.7
2.0



Optimization via Gradient Descent

Current W	W+h	Gradient dW
0.5	0.5	
1.1	1.1	
-0.7	-0.7	
2.0	2.0	
Loss: 1.25347		

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

h=0.0001

Optimization via Gradient Descent

Current W	W+h	Gradient dW
0.5	0.5+0.0001	-2.5
1.1	1.1	
-0.7	-0.7	
2.0	2.0	
Loss: 1.25347	Loss: 1.25322	

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{1.25322 - 1.25347}{0.0001}$$

$$= -2.5$$

Optimization via Gradient Descent

Current W	W+h	Gradient dW
0.5	0.5	-2.5
1.1	1.1+0.0001	-1.3
-0.7	-0.7	
2.0	2.0	
Loss: 1.25347	Loss: 1.25334	

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{?-?}{?}$$

Optimization via Gradient Descent

Current W	W+h	Gradient dW
0.5	0.5	-2.5
1.1	1.1+0.0001	-1.3
-0.7	-0.7	
2.0	2.0	
Loss: 1.25347	Loss: 1.25334	

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{1.25334 - 1.25347}{0.0001}$$

$$= -1.3$$

Optimization via Gradient Descent

Current W	W+h	Gradient dW
0.5	0.5	-2.5
1.1	1.1+0.0001	-1.3
-0.7	-0.7	Etc...
2.0	2.0	
Loss: 1.25347	Loss: 1.25334	

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

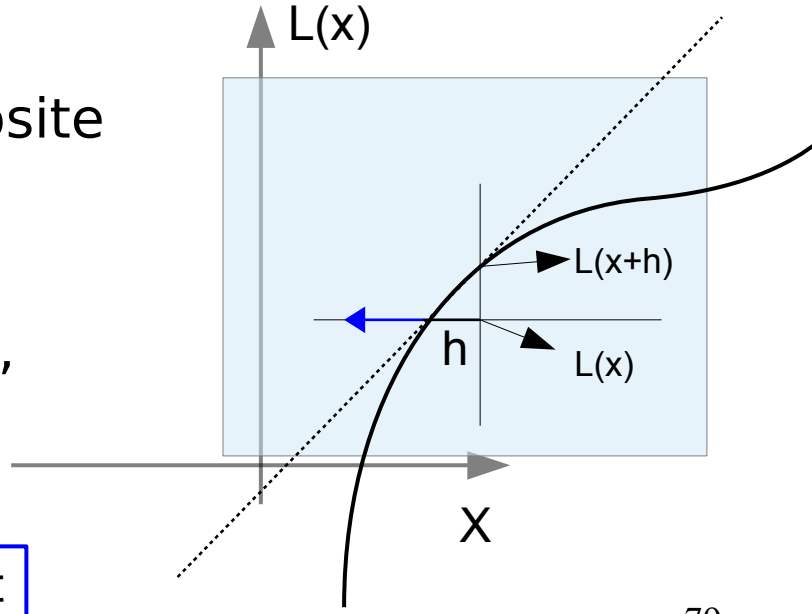
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - 1.25347}{0.0001}$$

Numerical Gradient

Optimization via Gradient Descent

1) Calculate Loss
2) Calculate Gradient
3) Move Parameter x in the opposite direction of the gradient (gradient points “uphill”) by a distance called the “step size”
4) repeat

Weight += - step size * gradient



Optimization via Gradient Descent

- That is very slow, we have to compute each weight independently

  → We use the analytical gradient!

  This is calculated using calculus on the expressions for Loss and Weights *hand waving* e.g. f(x) = x^2, f'(x) = 2x

- The analytical gradient is much faster, and exact, but can cause errors
- The numerical gradient is approximate, slow, but easy to write and it's hard to calculate wrong.

  → Use analytical gradient, but check for errors using

Gradient Descent

- https://docs.google.com/file/d/0Byvt-AfX75o1ZWxMRkxrUFJ2ZUE/preview
- https://docs.google.com/file/d/0Byvt-AfX75o1NndHNjVoVU1RRzQ/preview

Optimization via Gradient Descent

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

$$\nabla_w L(W) = \frac{1}{N} \sum_i \nabla_w L_i(f(x_i, W), y_i) + \lambda \nabla_w R(W)$$

'Nabla' operator: it denotes the gradient

i might be too big

Optimization via Stochastic Gradient Descent

$$\nabla_w L(W) = \frac{1}{N} \sum_i \nabla_w L_i(f(x_i, W), y_i) + \lambda \nabla_w R(W)$$

'Nabla' operator: it denotes the gradient

i might be too big

If you have many examples, the summing over i might take to long.
Solution: ***Stochastic Gradient Descent***

This means you only use a smaller subsample of your dataset, called a **minibatch** of 32/64/128/other examples to get an almost correct gradient.

DEMO

- http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

Summary

- The Model predicts the scores,

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	1.35	**1.1**

Summary

- The Model predicts the scores,
- The scores plug into the Loss function

$$L = \frac{1}{N} \sum_i L_i\left(f\left(x_i, W\right), y_i\right)$$

Summary

- The Model predicts the scores,
- The scores plug into the Loss function which
- Gives us the Loss for each class

Class\Pred.			
Apple	**1.87**	-0.4	0.1
Banana	**1.3**	1.1	-0.3
Pear	0.56	**1.35**	**1.1**
Loss	0.33	1.25	0

Summary

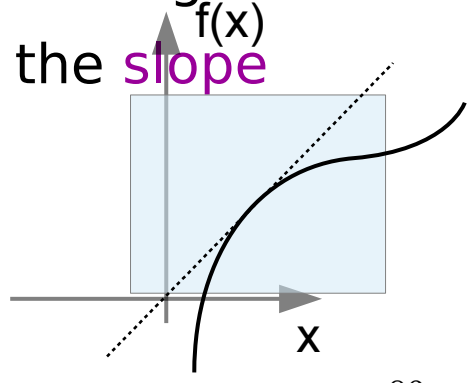
- The Model predicts the scores,
- The scores plug into the Loss function which
- Gives us the Loss for each class
- The Loss can be Regularized to avoid overfitting

$$L(W) = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + \lambda R(W)$$

Summary

- The Model predicts the scores,
- The scores plug into the Loss function which
- Gives us the Loss for each class
- The Loss can be Regularized to avoid overfitting
- Using the Loss function we can calculate the slope (Gradient)

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Summary

- The Model predicts the scores,
- The scores plug into the Loss function which
- Gives us the Loss for each class
- The Loss can be Regularized to avoid overfitting
- Using the Loss function we can calculate the slope (Gradient)
- Adjust weights in the opposite direction of the gradient

$$weights += -stepSize * weightsGrad$$

Summary

- The Model predicts the scores,
- The scores plug into the Loss function which
- Gives us the Loss for each class
- The Loss can be Regularized to avoid overfitting
- Using the Loss function we can calculate the slope (Gradient)
- Adjust weights in the opposite direction of the gradient
- Profit!

Questions?