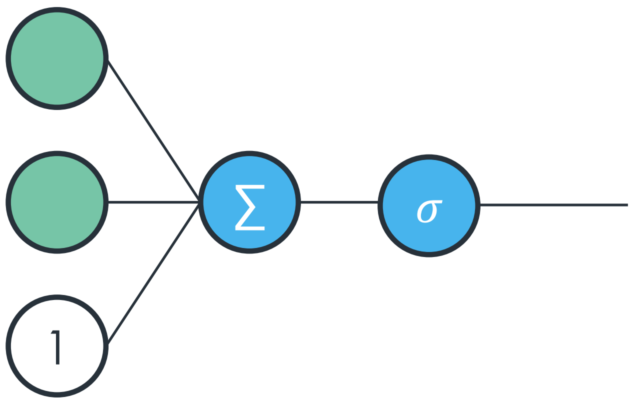
3. Fundamentals of Deep Learning

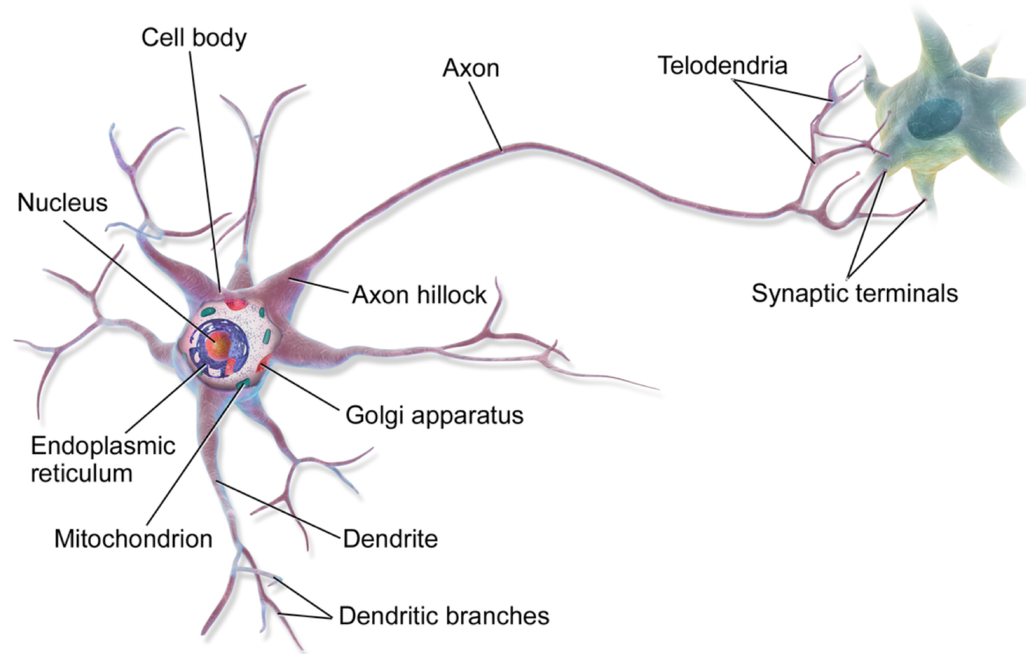
Perceptron

Feed-forward Neural Networks

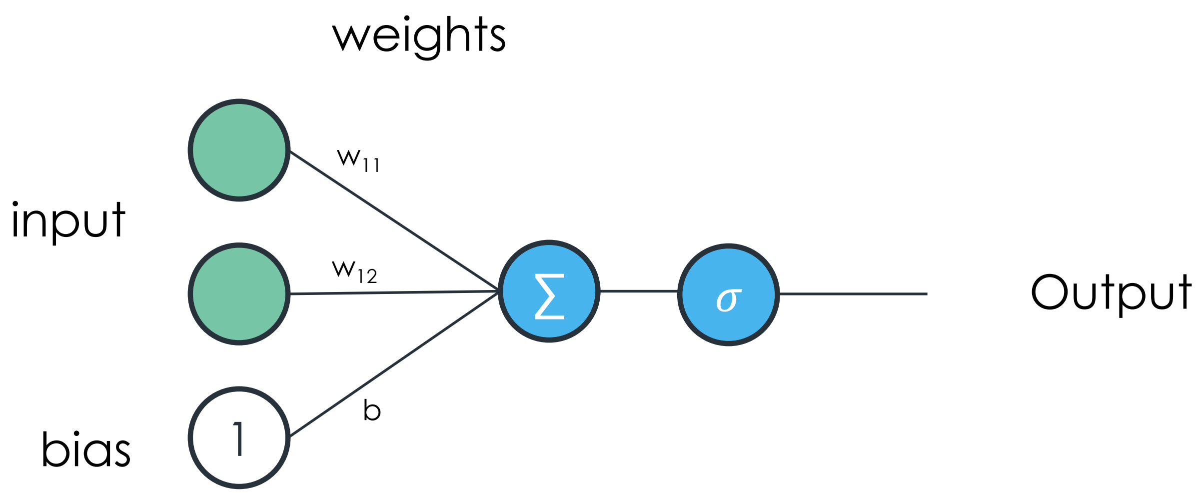
Architecture and Learning in Neural Networks

Perceptron are loosely based on neurons





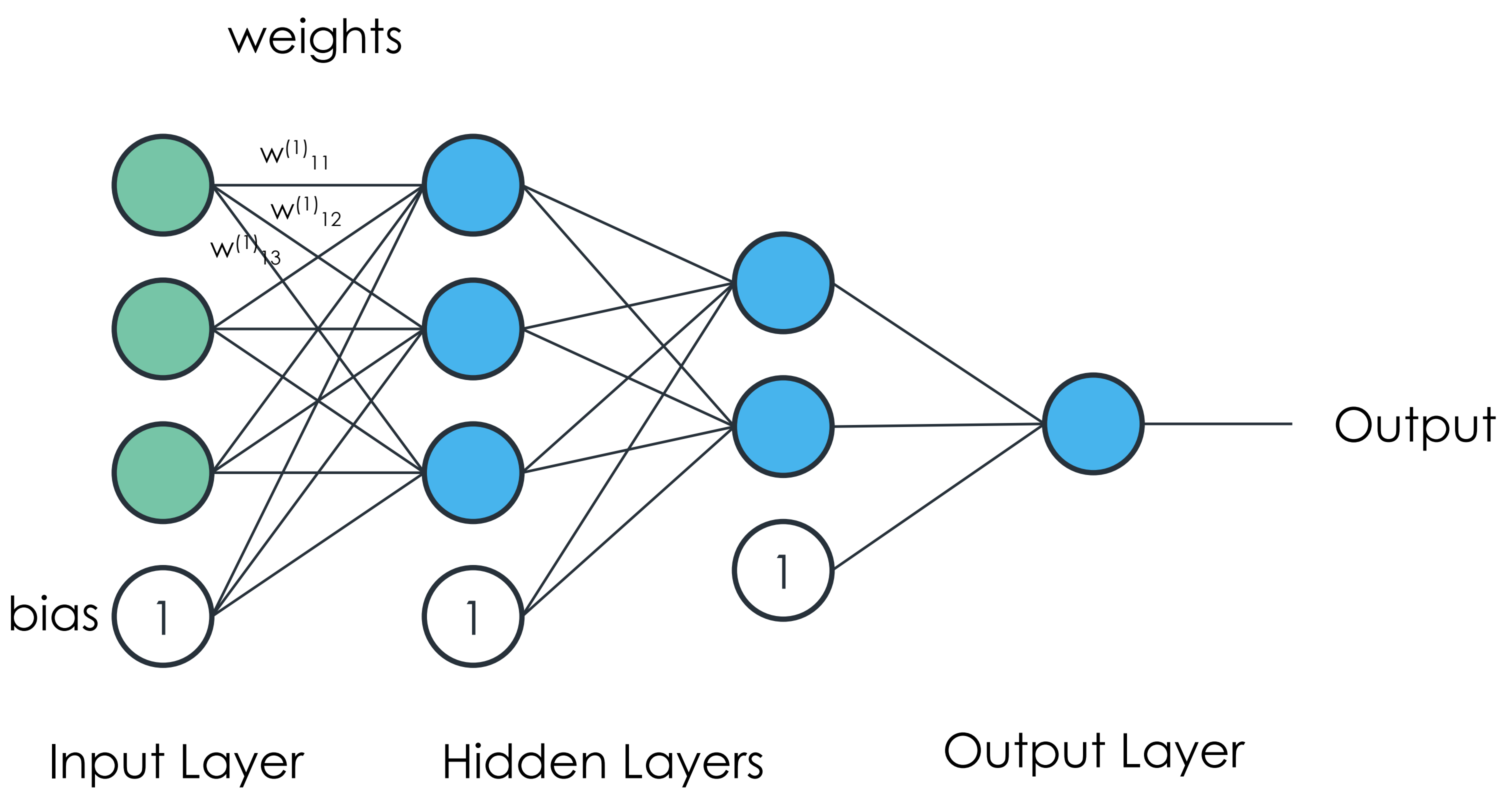
Perceptron



$$h = w_{11}x_1 + w_{12}x_2 + b$$

$$g = \sigma(h)$$

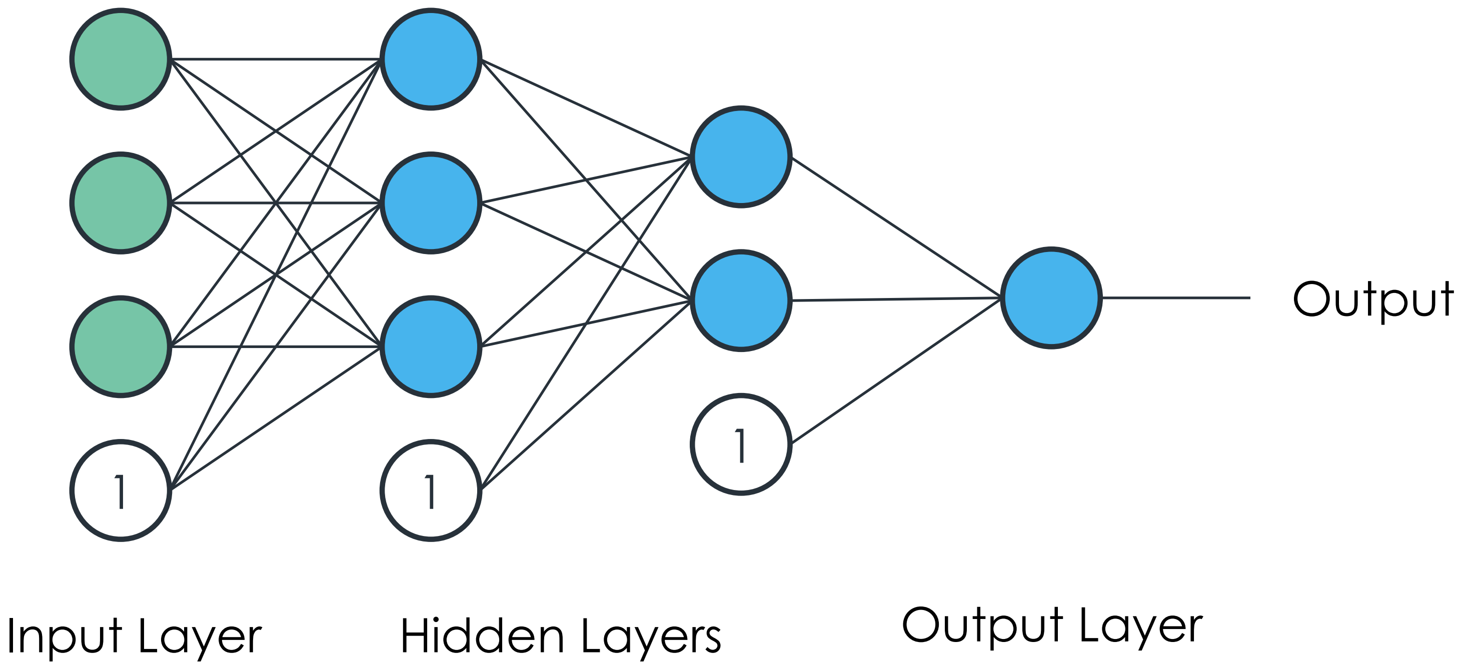
Neural Network



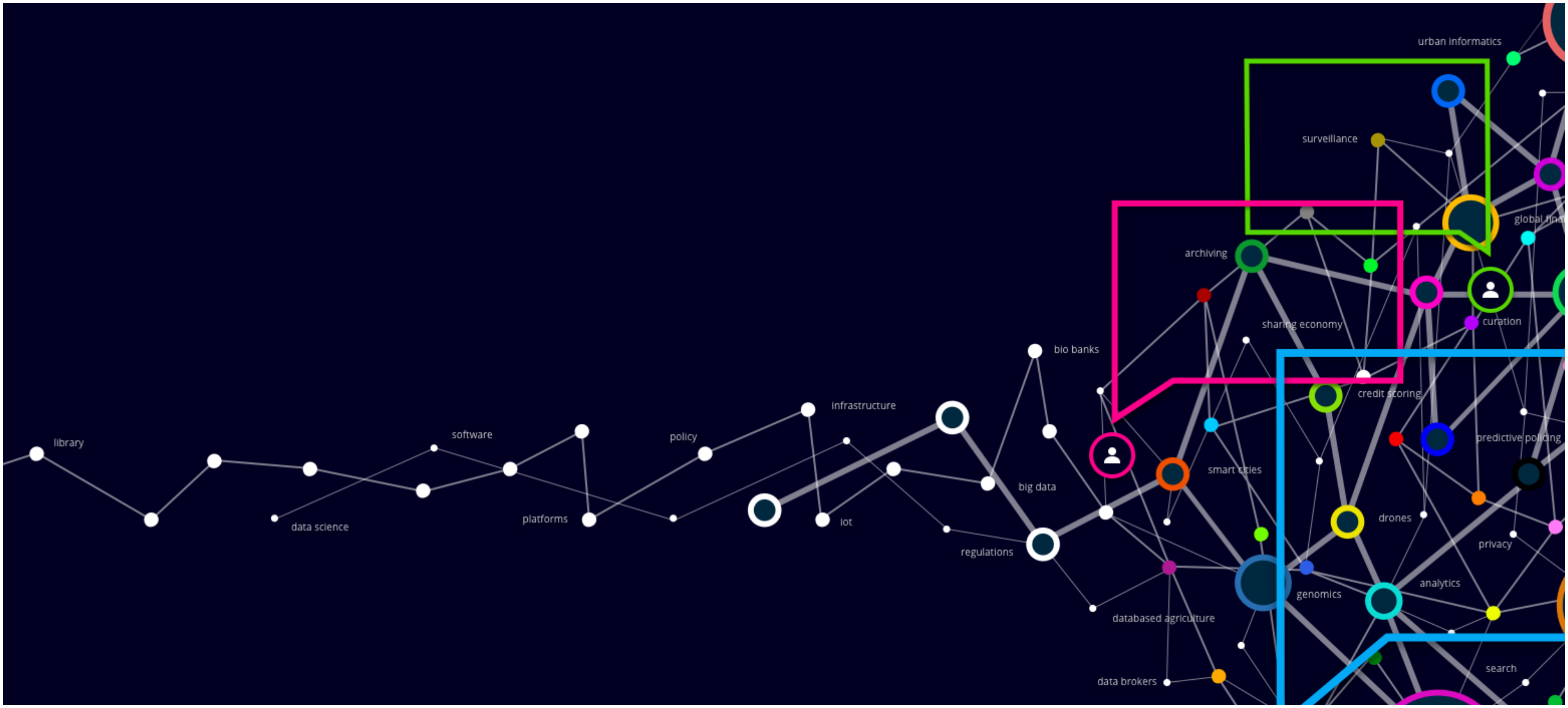
Building Neural Networks

- Frame the problem
  - What are you trying to predict?
  - What are the predictors ?
  - What known data do you have?
- Architect the network
- Train the network with known dataset (labeled data)
- Use the network to predict unseen data

How do you train a Neural Network?

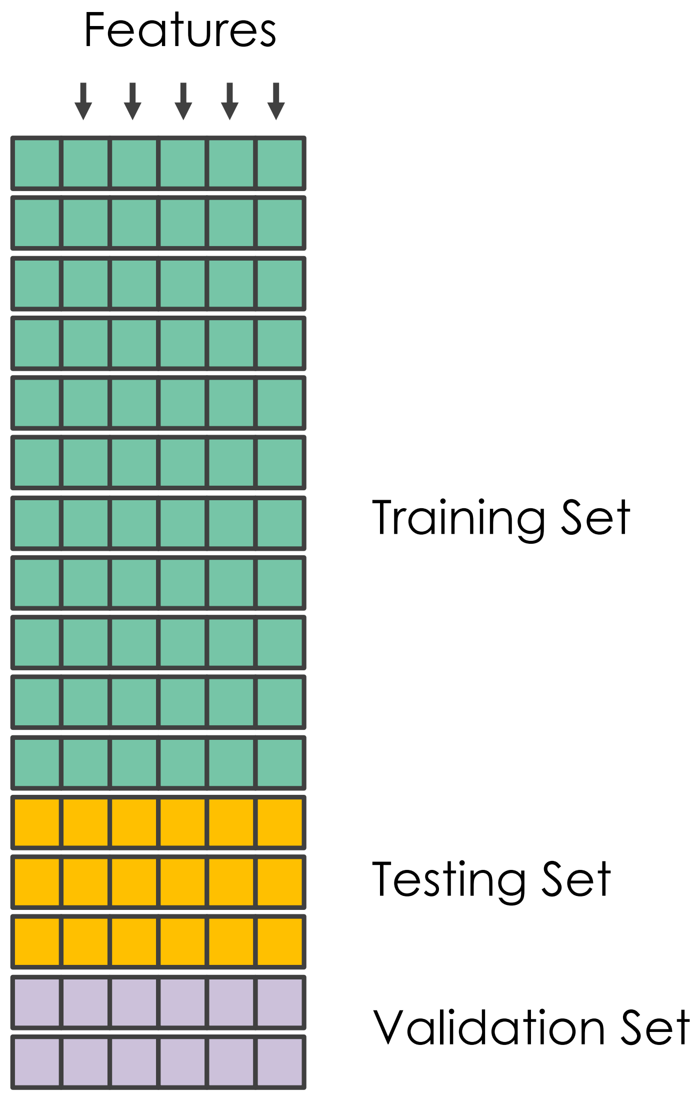


1. **Prepare the data** as input to feed into the neural network
2. **Feedforward** the input through the network, to get the output
3. Compare against the "correct" output,
   backpropagate to update weights
4. Repeat



Preparing Data

Labeled Training Data



Encoding Non-numeric input data

- What if the data has a column "Rainy Day" – Y/N.

How would we feed this to the network?

Rainy Day	encoded_rainy_day
Y	1
N	0

Encoding Non-numeric input data

- What if the data has a column "Countries" – "Canada", "USA", "Mexico".

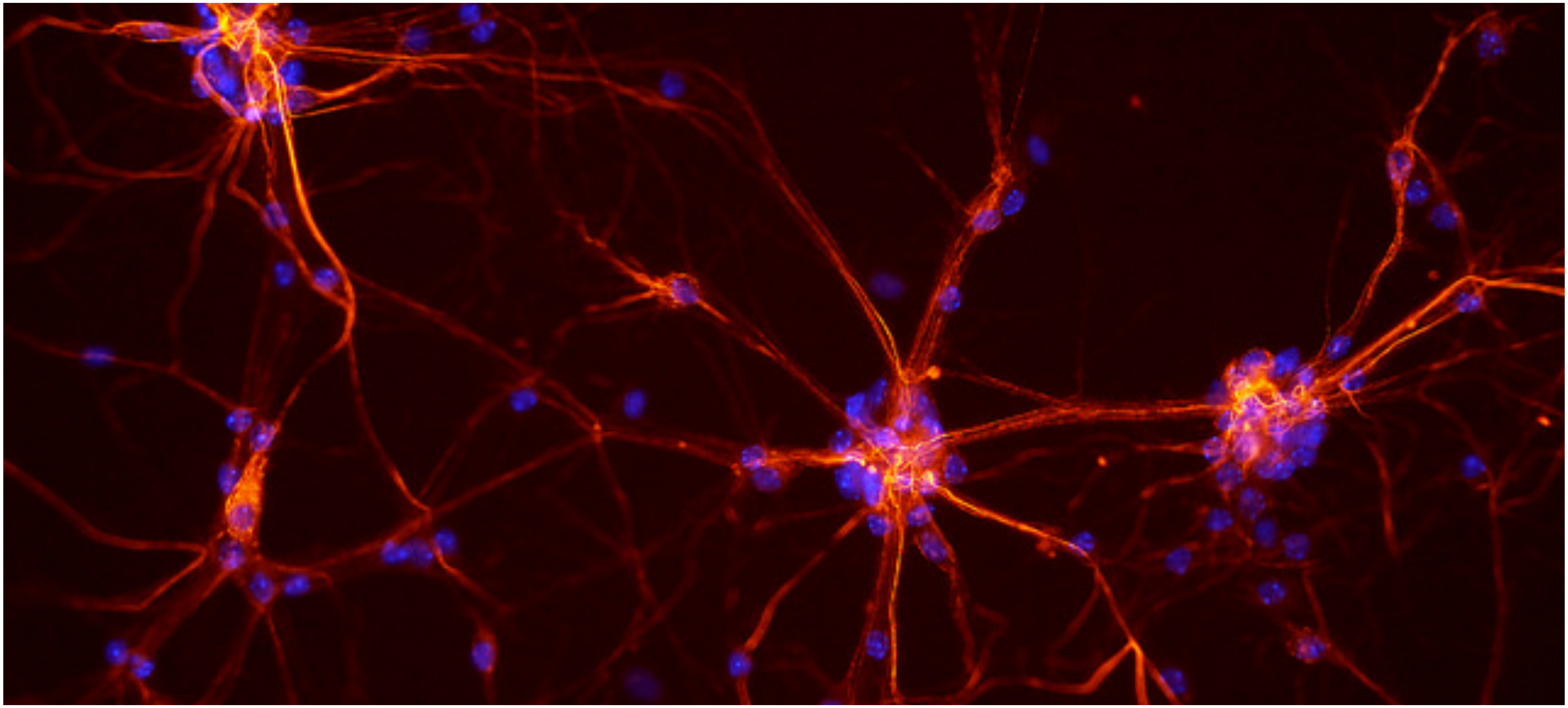
How would we feed this to the network?

Country	Country_Canada	Country_USA	Country_Mexico
Canada	1	0	0
USA	0	1	0
Mexico	0	0	1

This encoding is called One-hot encoding

Prepare the data

- Rules of thumb
  - Preserve existing relationship between features
  - Do not make nominal data into ordinal
  - Normalize data by scaling using mean/std-dev



Backpropagation & Training

Backpropagation

- Our objective is to reduce the error.

```
Assume some random weights
repeat
  calculate our model y=f(x)
  come up with a way to measure error
  adjust weights in order to minimize error
until error (accuracy level) is acceptable
```

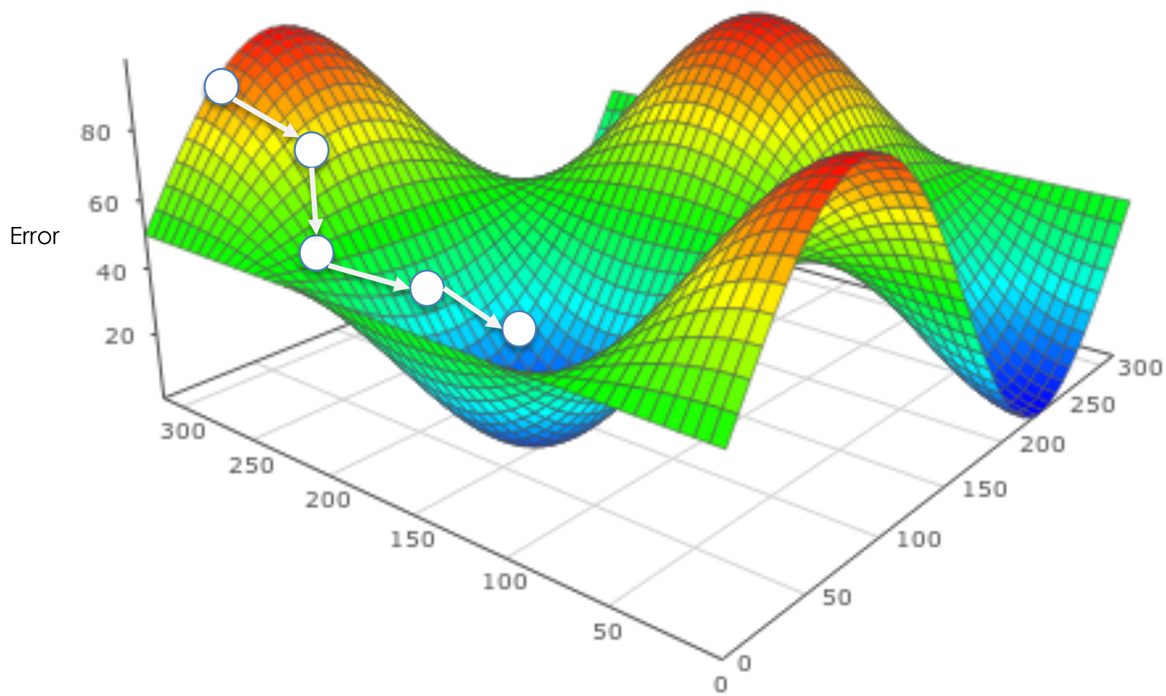
Objective function

- Our approach to measuring error and minimizing it, is called the objective function.

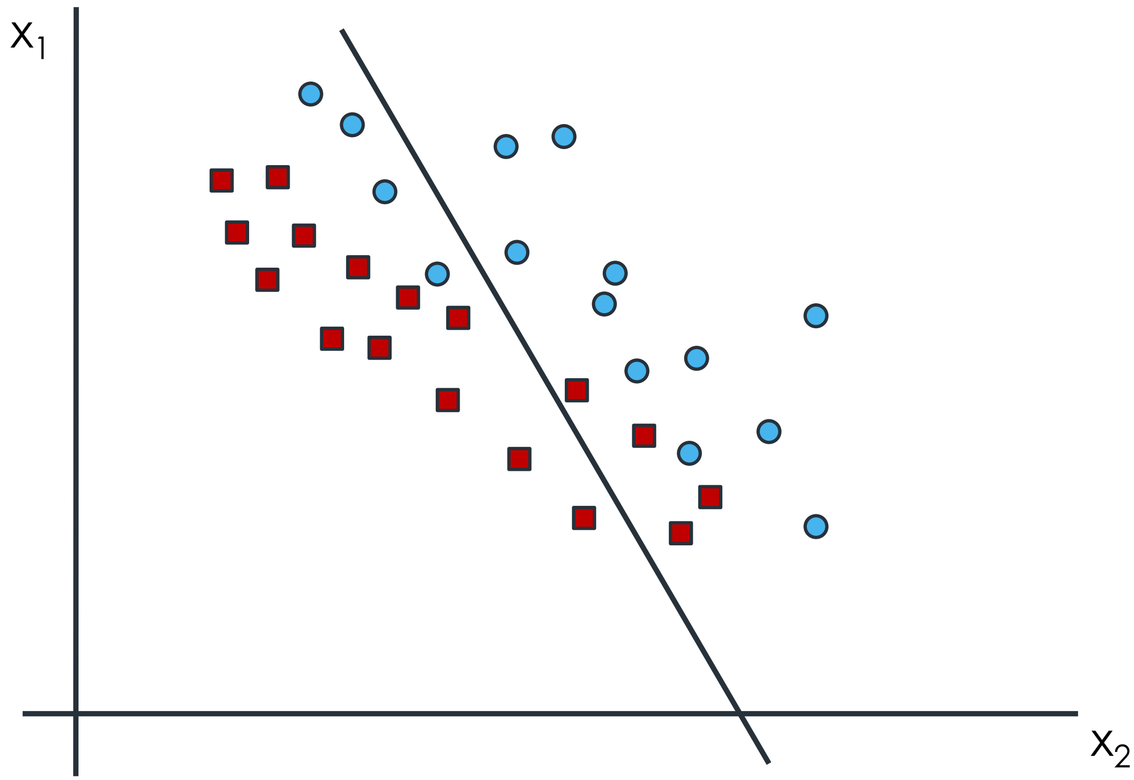
- Error is measured in many ways.
  This is referred to as **loss function** or **cost function**

- There some common ones,
  - Mean Square Error (MSE)
  - Mean Absolute Error
  - Logistic Error
  - Cross Entropy

Minimizing Error



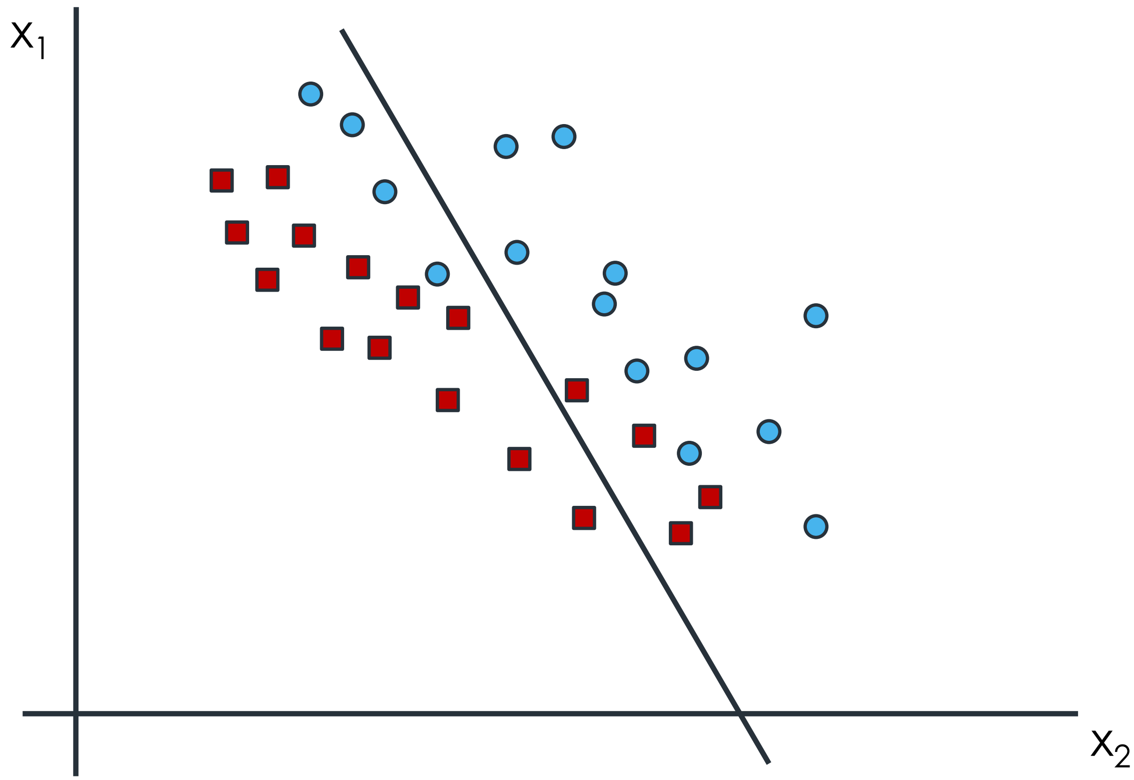
Classification



$$h = w_{11}x_1 + w_{12}x_2 + b$$

$$g = \sigma(h)$$

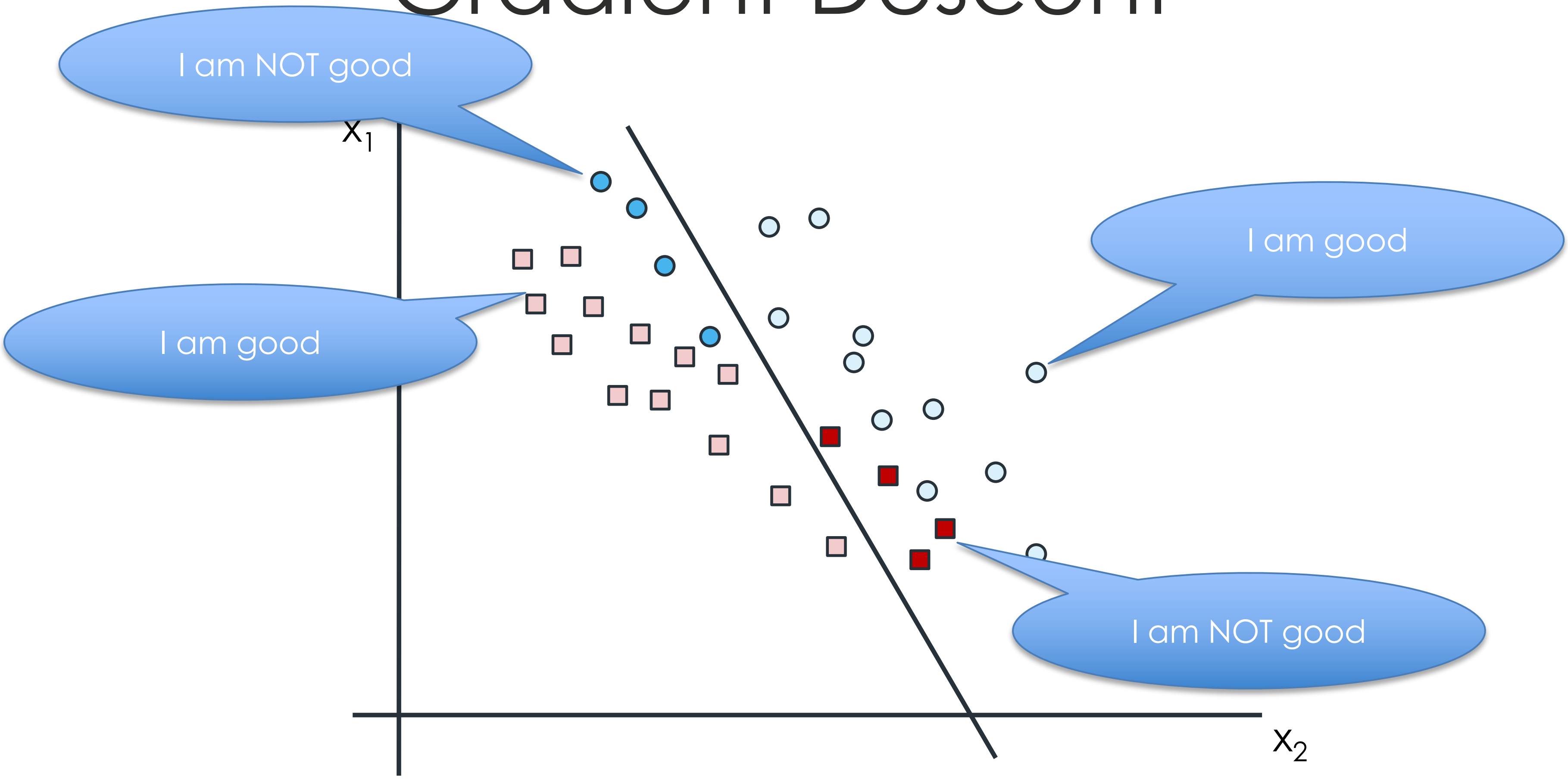
Classification



$$h = Wx + b$$

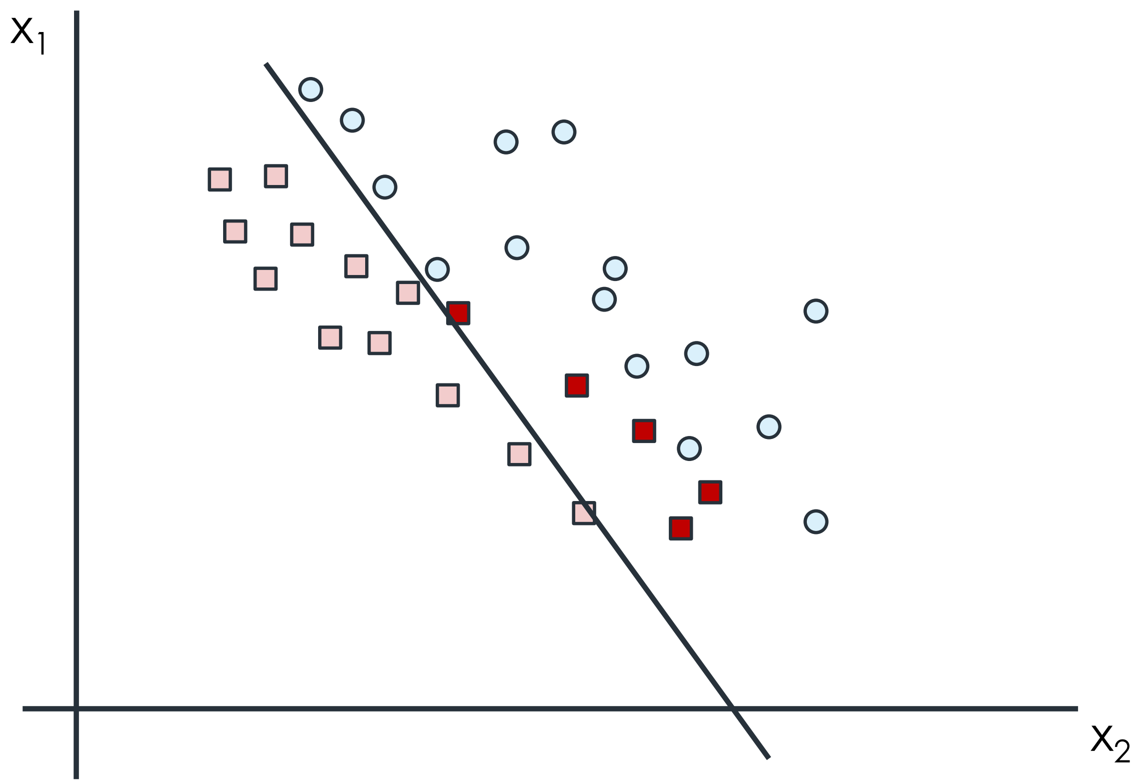
$$g = \sigma(h)$$

Gradient Descent



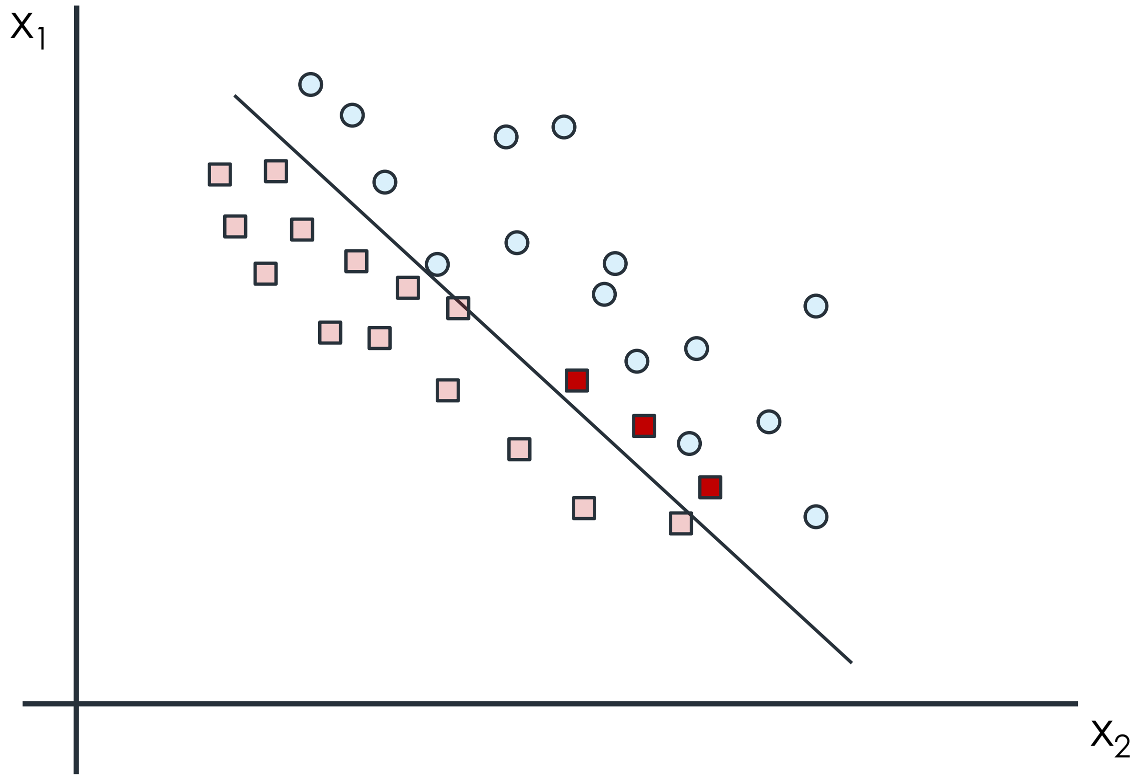
$$\hat{y} = \sigma(Wx + b)$$

Gradient Descent



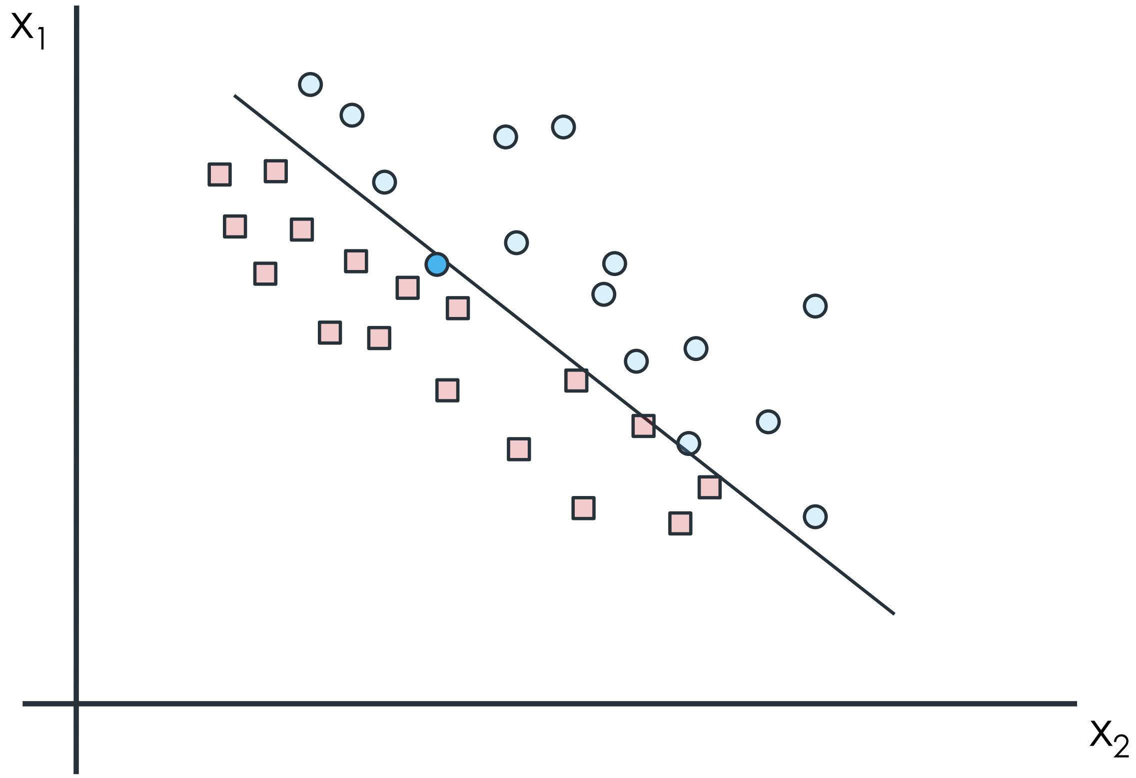
$$\hat{y} = \sigma(Wx + b)$$

Gradient Descent



$$\hat{y} = \sigma(Wx + b)$$

Gradient Descent



$$\hat{y} = \sigma(Wx + b)$$

This general approach is called Gradient Descent

Epochs

- Repeat the training exercise multiple times. Each cycle is called an epoch (pronounced 'epic')

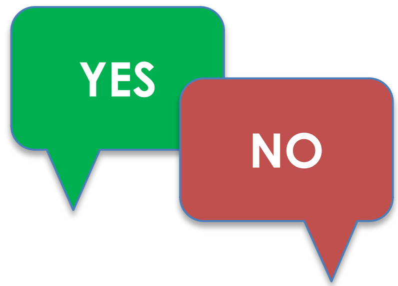
- Each epoch runs a training cycle that runs in memory. But, the dataset might be too large to fit in memory, so we break each epics into multiple batches. This can be organized as,

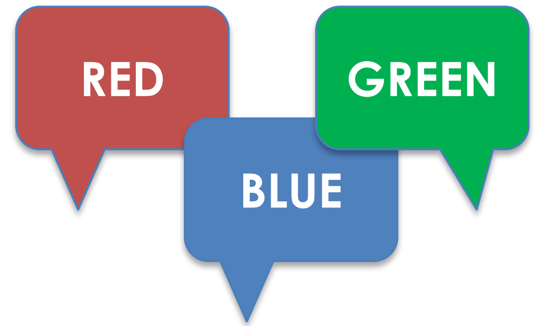
  - Batched Gradient Descent
    - Run all the data in each epoch

  - Stochastic Gradient Descent
    - Run random subset of data in each epoch

Converting Discrete Errors into Continuous Errors

- Discrete





- Continuous

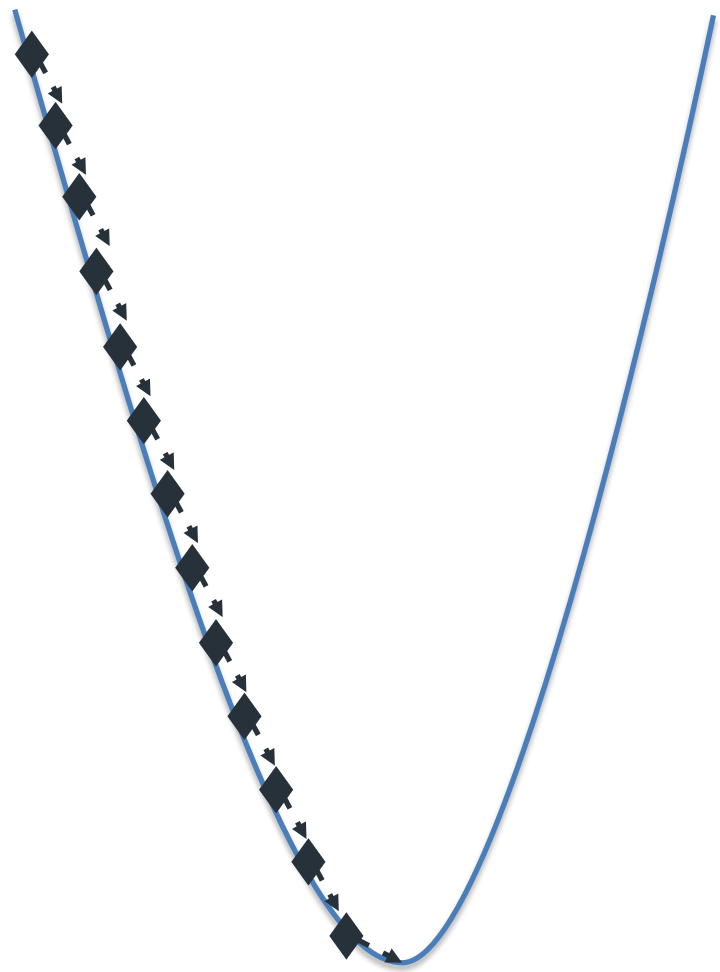
P(Yes) = 85%

P(Red) = 0.4
P(Blue) = 0.5
P(Green) = 0.1

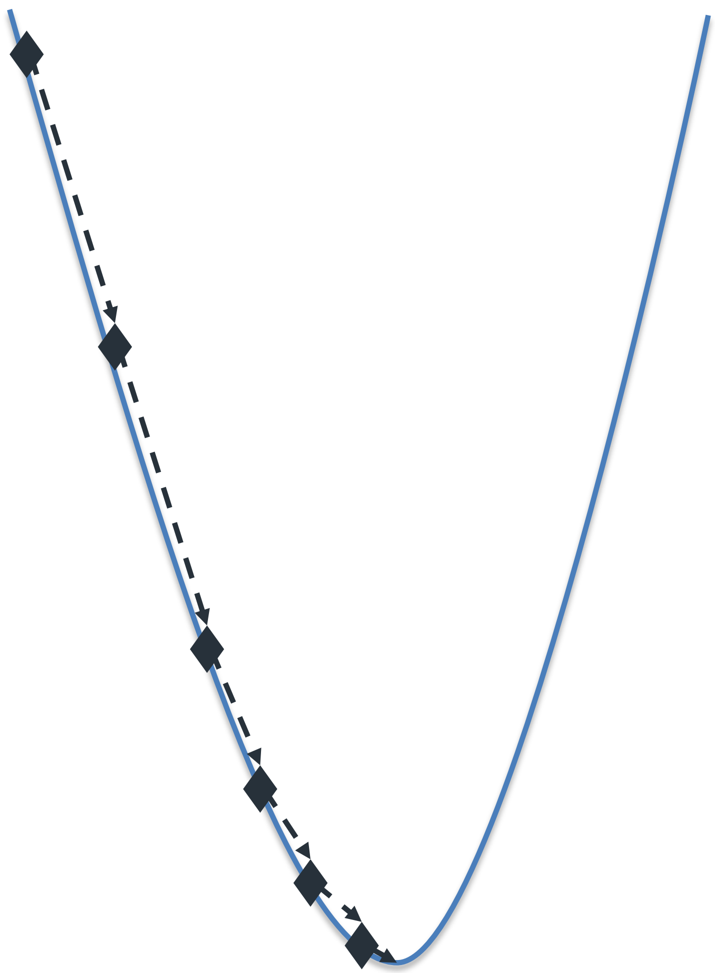
Common Error Functions

- Cross-entropy
  - Good of using with probabilities
  - Works well for classification problems
- Mean-Squared Error
  - Good for continuous predictions

Gradient Descent is a slow process

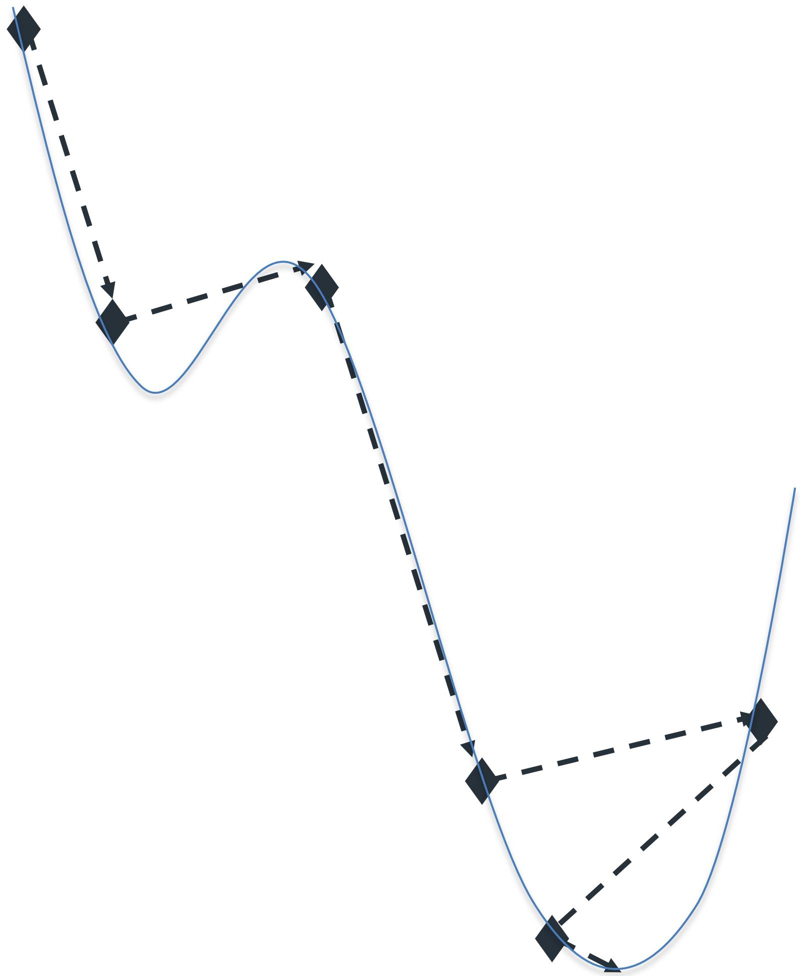


Adapting Learning Rate



- Make big jumps farther away from minima
- Make smaller (surgical) moves closer to a minima

Adapting with Momentum

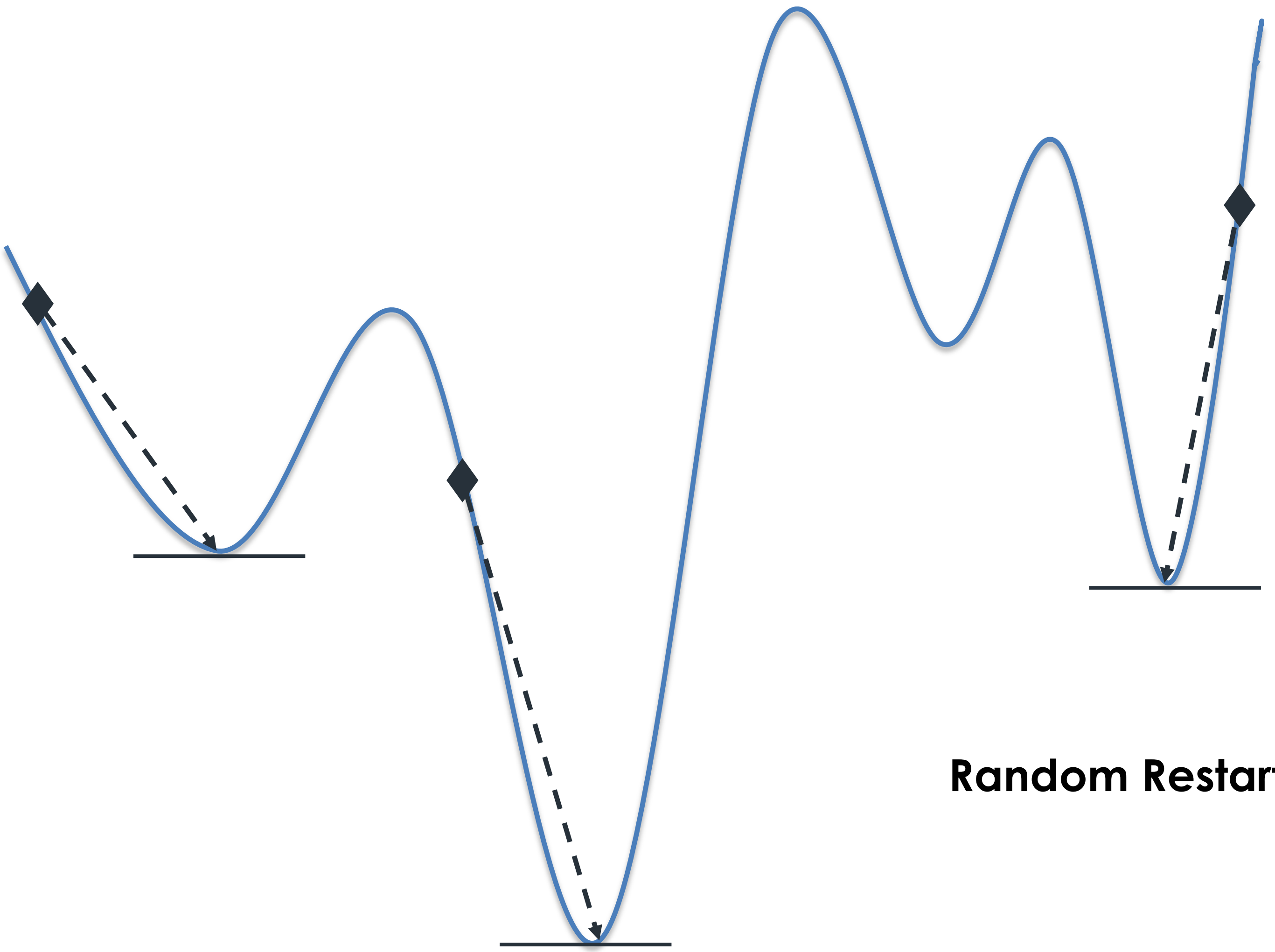


- Adding momentum can help the function cross small dips

Common optimizers

- Some common optimizers include,
  - AdamOptimizer
  - RmsPropOptimizer
  - AdaGrad

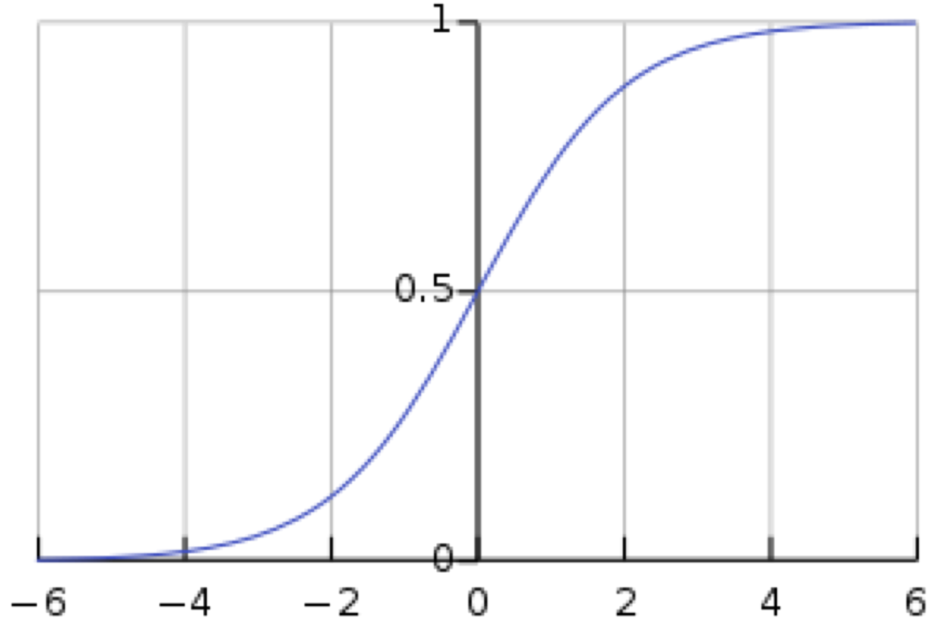
Avoiding Local Minima



Random Restarts

Activation - Sigmoid

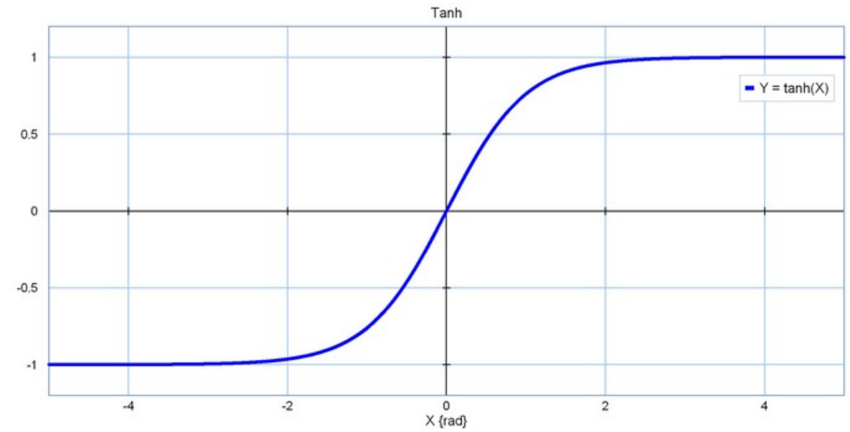
- Sigmoid – the original Activation function
- Makes the model non-linear
- Vanishing Gradient with Sigmoid – Away from the mean, sigmoid tangents are nearly flat



Better activation functions

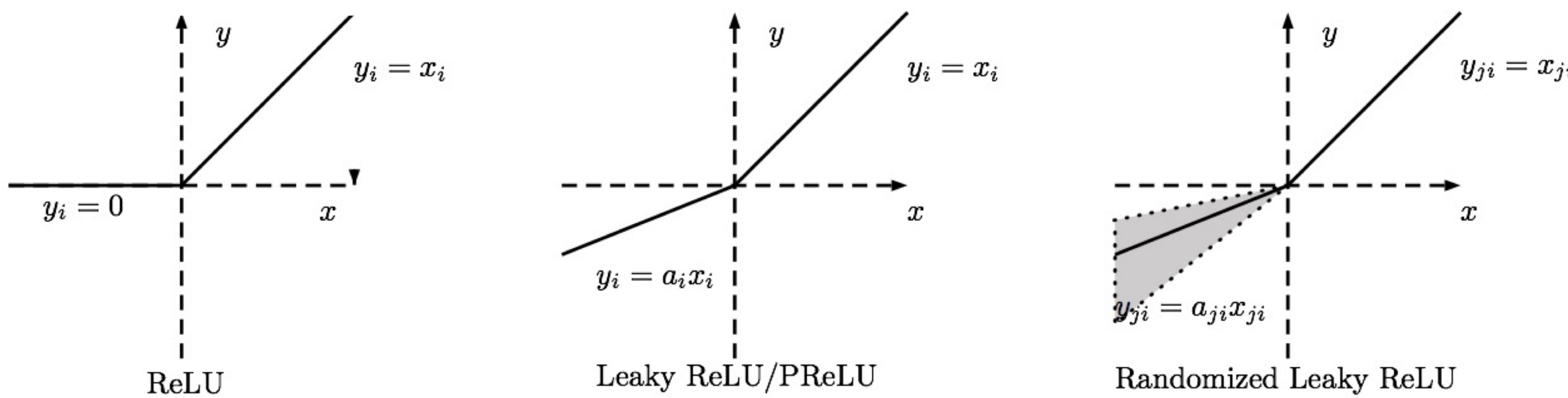
Hyperbolic Tangents (tanh)

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Rectified Linear Unit (relu)

$$relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$



Regularization

- Penalize Large Weights
- Make the activation function shallow, so GD works better

L1:

$$ErrorFunction = -\frac{1}{m}\sum_{i=1}^{n}[y_i\, p_i + (1 - y_i)\,(1 - p_i)] + \lambda(|w_1| + |w_2| + \ldots |w_n|)$$

L2:

$$ErrorFunction = -\frac{1}{m}\sum_{i=1}^{n}[y_i\, p_i + (1 - y_i)\,(1 - p_i)] + \lambda(w_1^2 + w_2^2 + \ldots w_n^2)$$

Layers

- Dense

  Fully Connected Layer

- Dropout

  To reduce overfitting. Without dropout, it is likely that the dominant nodes train more than the non-dominant nodes. Dropout gives other nodes a chance. Each node has a probability it will be dropped in a particular epoch during training.

- Flatten
- Convolutional

  Apply filters to convert large shallow datasets, into smaller deeper datasets.

- Pooling

  Reduce size of dataset, by reducing resolution across local cells

- Locally Connected Layers
- BasicRNNCell
- LSTM …

Hands-on – Bike Rental Prediction