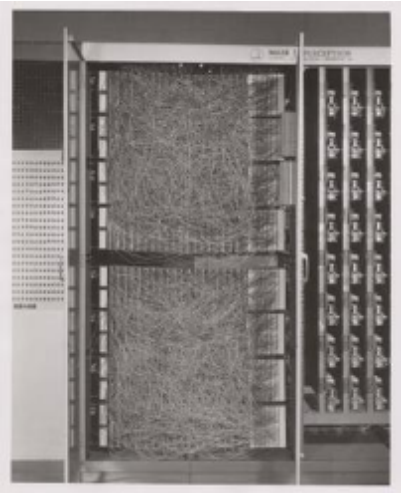
Perceptron Slides

Team DGMRW

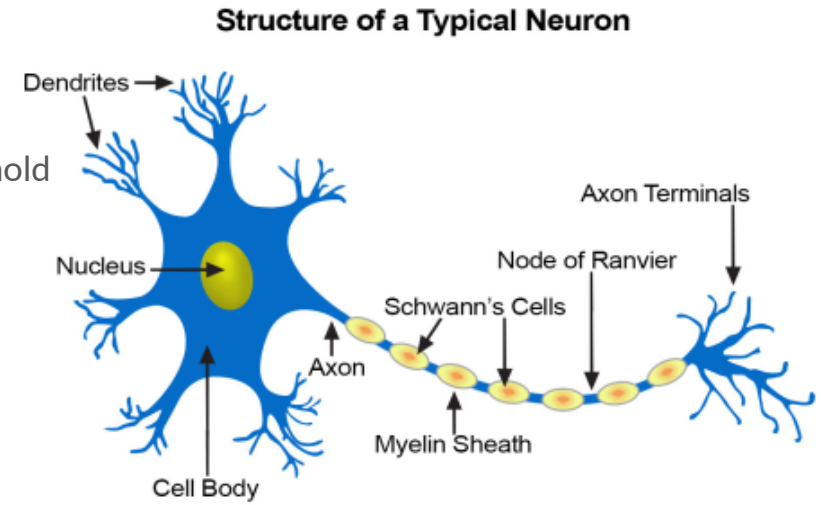
History of Perceptrons

- Invented by Frank Rosenblatt in 1958 for the Navy
- 5 ton machine that worked on punch cards
- Originally used for image recognition
- Laid foundation for neural networks and AI



Biological Influence

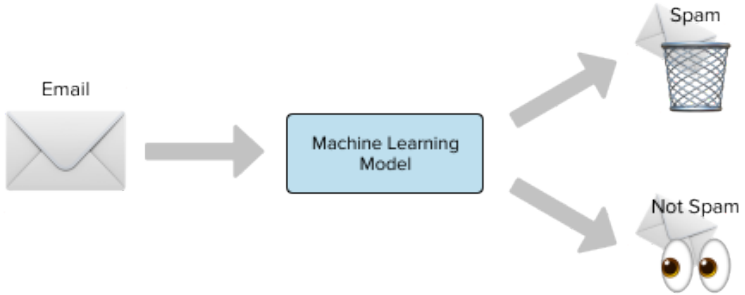
- Perceptron took heavy influence from neurons
- Neurons fire when sum of inputs exceed some threshold
- Perceptron acts like single neuron



Binary Linear Classification

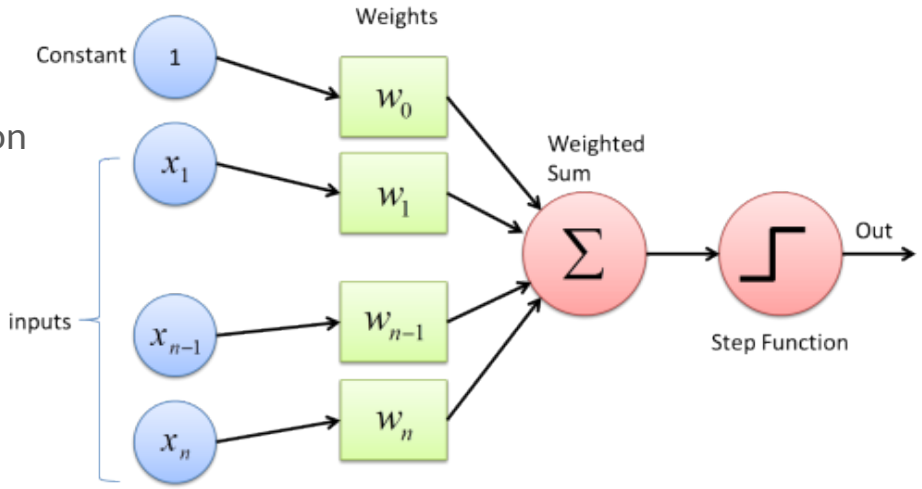
- Binary classification assigns a set of data (input) to one of two classes
- Examples: Email is spam or not, Picture is cat or dog
- Perceptrons take input data (either featurized or raw data) and output predicted label
- Use vector to represent input and scalar to represent output
- For classifying emails as spam or not, 1 may represent an email that is spam while 0 represents an email that is not

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$



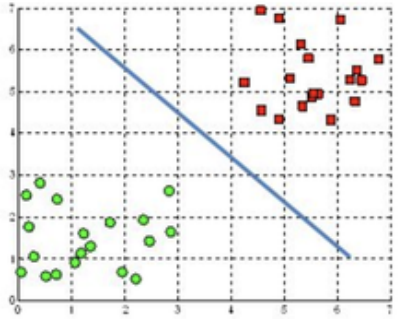
What are Perceptrons?

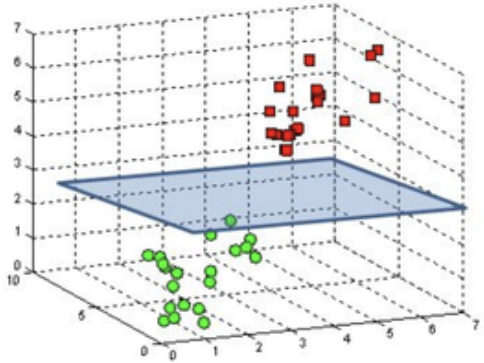
- Building blocks of neural networks
- Used primarily for binary linear classification
- Weighted sum of inputs + Step Function
- Finds dividing hyperplane to separate data



Weights

- One weight for each input
- “Training” the perceptron is finding the optimal weights for a given data set
- Weight vector represents separating hyperplane of data
- Dot product of weight vector and input is positive for one class or negative for the other

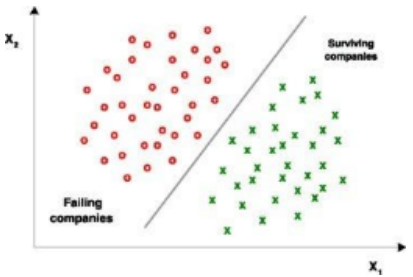




$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

Perceptron Learning Algorithm

- Iterative algorithm for finding weights
- “r” represents learning rate, can be tuned
- Only update weights if incorrect
- Perceptrons will only work if data is linearly separable
- Linear separability guarantees convergence
- Learning Algorithm does not have to find optimal weights
- Runtime of Learning



1. Initialize the weight vector $\mathbf{w}$ to the zero vector $\mathbf{0}$
2. For each training point $(\mathbf{x_i}, y_i)$ in dataset D:
   (a) Calculate the current classification using weights $\mathbf{w}$

$$y_i^* = \begin{cases} 1, \mathbf{w}^T\mathbf{x_i} \geq 0, \\ -1, \mathbf{w}^T\mathbf{x_i} < 0 \end{cases}$$

   (b) If the point is incorrectly classified, update weights using

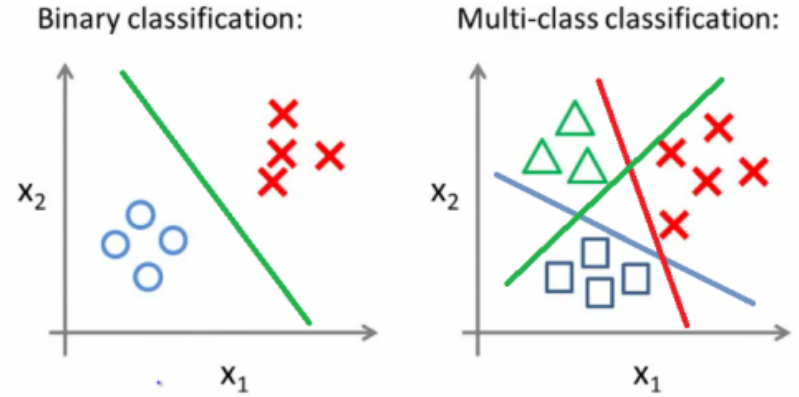
$$\mathbf{w}_{new} = \mathbf{w}_{old} + r * y_i^* * \mathbf{x_i}$$

3. Repeat step 2 until no weights are updated

Multiclass Classification

- Perceptron algorithm only works for binary case
- For multiclass classification, slightly modify perceptron
- One weight vector for every class
- New classification rule:

$$y = \underset{i=0,1,\ldots m}{\arg\max}\ \mathbf{w}_i^T \mathbf{x}$$



Multiclass Classification (cont.)

- Modified weight update algorithm
- Still only update weights if incorrect
  - Add to correct weight vector
  - Subtract from others (incorrect weights)
  - Hope that argmax correctly classifies next time

1. Initialize each weight vector $\mathbf{w}_j$ to the zero vector $\mathbf{0}$
2. For each training point $(\mathbf{x_i}, y_i)$ in dataset D:
   (a) Calculate the current classification

$$y_i^* = \arg\max_{i=0,1,\ldots m} \mathbf{w}_i^T \mathbf{x}$$

   (b) If the point is incorrectly classified:

      i. Update the desired weight vector $\mathbf{w}^*$ using

$$\mathbf{w}^*_{new} = \mathbf{w}^*_{old} + r * \mathbf{x_i}$$

      ii. Update the rest of the weight vectors using

$$\mathbf{w}_{new} = \mathbf{w}_{old} - r * \mathbf{x_i}$$

3. Repeat step 2 until no weights are updated

Beyond Perceptrons

- Linear separability assumption severely limits uses for perceptrons
- Can chain multiple perceptrons together to avoid this
  - Called Multilayer Perceptron
- Perceptron doesn't find optimal weights
  - Adding data points can shift hyperplane