Neural Networks
neural networks

• Show up everywhere
  • Machine translation
  • Image recognition
  • Video generation
  • …
• Many, many uses, and too much to cover in this course
• We will focus on neural networks used as classifiers
neural networks

• Basic classification problem for neural networks:
  
  • I have a set of labeled **training data**
  
  • Learn a **decision boundary** that separates the two classes of data
neural networks

- Basic classification problem for neural networks:
  - I have a set of labeled **training data**
  - Learn a **decision boundary** that separates the two classes of data
neural networks

• Basic classification problem for neural networks:
  • I have a set of labeled training data
  • Learn a decision boundary that separates the two classes of data
  • Given a new point
    • Classify it using the decision boundary you learned
neural networks

• Basic classification problem for neural networks:
  • I have a set of labeled training data
  • Learn a decision boundary that separates the two classes of data

• Given a new point
  • Classify it using the decision boundary you learned
what is this model?

• Suppose the decision boundary is a **straight line**

\[
\text{class} = \begin{cases} 
0 & \text{if } b + w_1 x + w_2 y \leq 0 \\
1 & \text{if } b + w_1 x + w_2 y > 0 
\end{cases}
\]

• How do we learn the parameters of this model?

• Can use a **perceptron**
A perceptron is the simplest form of a "neuron" like those used in neural networks.

- Weighted sum of inputs plus an **activation function** that translates that weighted sum into an output.

- Can view it graphically as a "node" with inputs, and weights.

\[
\begin{align*}
1.0 & \rightarrow b \\
x & \rightarrow w_1 \\
y & \rightarrow w_2 \\
\text{Output} & \\
\end{align*}
\]
A perceptron is the simplest form of a “neuron” like those used in neural networks.

- Weighted sum of inputs plus an activation function that translates that weighted sum into an output.

- Can view it graphically as a “node” with inputs, and weights.

- And can view the weighted sum as a linear algebra operation.
So how do we learn the weights?

Randomly initialize weights

Repeat the following on all inputs until convergence

- Run the perceptron on a training input to get a “predicted” output
- If the predicted output matches the real output (from the labels), leave the weights alone
- If it doesn’t match, move the weights in the direction of the input if the label is 1, away from the input if the direction is 0

input $i = (x_i, y_i)$
real output $= o_i$
predicted output $= p_i$

$$
\begin{align*}
    b^{(t+1)} &= b^{(t)} + (o_i - p_i) \cdot 1.0 \\
    w_1^{(t+1)} &= w_1^{(t)} + (o_i - p_i) \cdot x_i \\
    w_2^{(t+1)} &= w_2^{(t)} + (o_i - p_i) \cdot y_i \\
    \overline{w}^{(t+1)} &= \overline{w}^{(t)} + err \cdot \vec{i}
\end{align*}
$$
training a perceptron

• Does this always work?

• Guaranteed to converge if a **linear decision boundary** exists

• If no decision boundary exists, then algorithm will not converge
  
  • Perceptrons cannot learn non-linear decision boundaries

• **So what can?**
Does this always work?

Guaranteed to converge if a linear decision boundary exists

If no decision boundary exists, then algorithm will not converge

Perceptrons cannot learn non-linear decision boundaries

So what can?
training a perceptron

• Does this always work?

• Guaranteed to converge if a **linear decision boundary** exists

• If no decision boundary exists, then algorithm will not converge
  
  • Perceptrons cannot learn non-linear decision boundaries

• **So what can?**