# Exam Prep Discussion 12

# Laplace Smoothing

- A technique to prevent overfitting when predicting probabilities from training data

# Laplace Smoothing

- A technique to prevent overfitting when predicting probabilities from training data
- Assumes that we have already seen $k$ extra instances of each observation

# Laplace Smoothing

- A technique to prevent overfitting when predicting probabilities from training data
- Assumes that we have already seen *k* extra instances of each observation

| X = x | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|
| Y = y | 1 | 1 | 1 | 1 | 0 | 0 |

- MLE prediction formula: $P(X = x|Y = y) = \dfrac{\text{\# of times X = x} \; + k}{\text{\# of times Y = y} \; + k|X|}$

# Laplace Smoothing

- A technique to prevent overfitting when predicting probabilities from training data
- Assumes that we have already seen *k* extra instances of each observation

| | | | | | | |
|---|---|---|---|---|---|---|
| **X = x** | 0 | 0 | 0 | 0 | 1 | 0 |
| **Y = y** | 1 | 1 | 1 | 1 | 0 | 0 |

- MLE prediction formula: $P(X = x|Y = y) = \dfrac{\text{\# of times X = x} \; + k}{\text{\# of times Y = y} \; + k|X|}$

  Without Laplace smoothing, P(X = 1 | Y = 1) = 0
  With Laplace smoothing with *k* = 2, P(X = 1 | Y = 1) = $\dfrac{0 + 2}{4 + 2*2}$ = ¼

# Laplace Smoothing

- A technique to prevent overfitting when predicting probabilities from training data
- Assumes that we have already seen *k* extra instances of each observation

| X = x | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|
| Y = y | 1 | 1 | 1 | 1 | 0 | 0 |

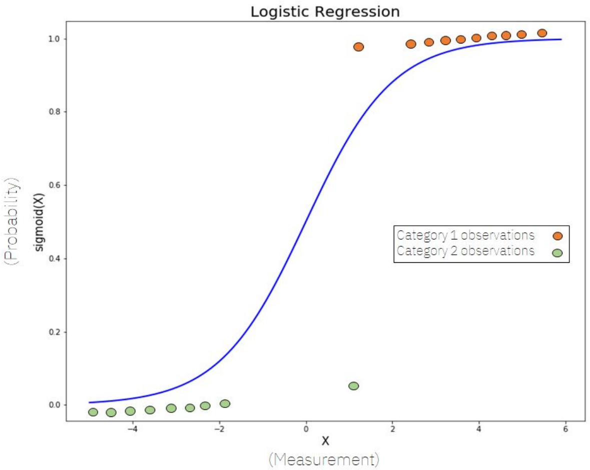
- MLE prediction formula: $P(X = x|Y = y) = \dfrac{\text{\# of times X = x} \; + k}{\text{\# of times Y = y} \; + k|X|}$

  Without Laplace smoothing, P(X = 1 | Y = 1) = 0
  With Laplace smoothing with *k* = 2, P(X = 1 | Y = 1) = $\dfrac{0 + 2}{4 + 2*2}$ = ¼

- *k* is a hyperparameter you can choose
  - A smaller *k* adheres to the true observation's distribution
  - A bigger *k* causes a more uniform distribution

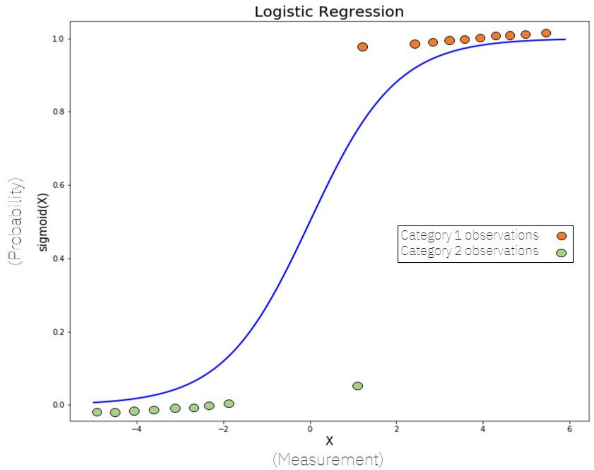
# Logistic Regression

- Use $\vec{w}^T\vec{x}$ to obtain a probability of a data point belonging to a class, using the sigmoid function



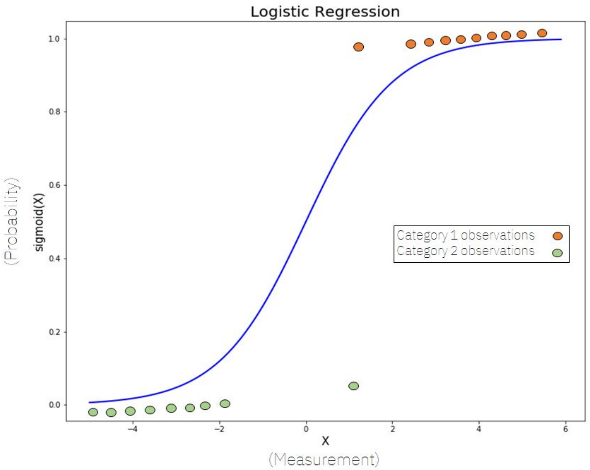
# Logistic Regression

- Use $\vec{w}^T\vec{x}$ to obtain a probability of a data point belonging to a class, using the sigmoid function
- Probability that the point belongs to the positive class: $P(y|x,w) = \dfrac{1}{1+e^{-w^T x}}$



# Logistic Regression

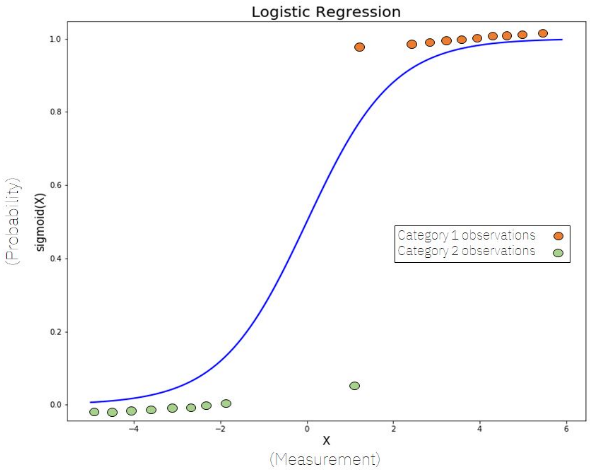
- Use $\vec{w}^T\vec{x}$ to obtain a probability of a data point belonging to a class, using the sigmoid function
- Probability that the point belongs to the positive class: $P(y|x,w) = \dfrac{1}{1+e^{-w^T x}}$
- Probability that the point belongs to the negative class: $P(y|x,w) = 1 - \dfrac{1}{1+e^{-w^T x}}$



# Logistic Regression

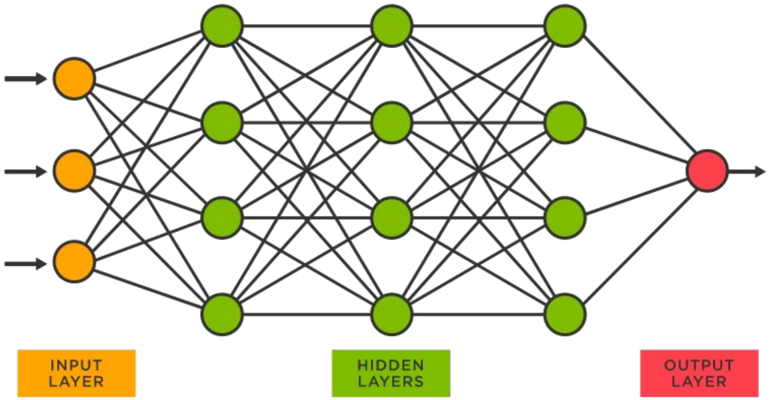
- Use $\vec{w}^T\vec{x}$ to obtain a probability of a data point belonging to a class, using the sigmoid function
- Probability that the point belongs to the positive class: $P(y|x,w) = \dfrac{1}{1+e^{-w^Tx}}$
- Probability that the point belongs to the negative class: $P(y|x,w) = 1 - \dfrac{1}{1+e^{-w^Tx}}$
- Goal: Find $\vec{w}$ that maximizes the probability of correct classification for all points

$$\vec{w}^* = \text{argmax}_w \prod_i P(y_i|x_i,w) = \text{argmax}_w \log \prod_i P(y_i|x_i,w)$$



# Neural Networks

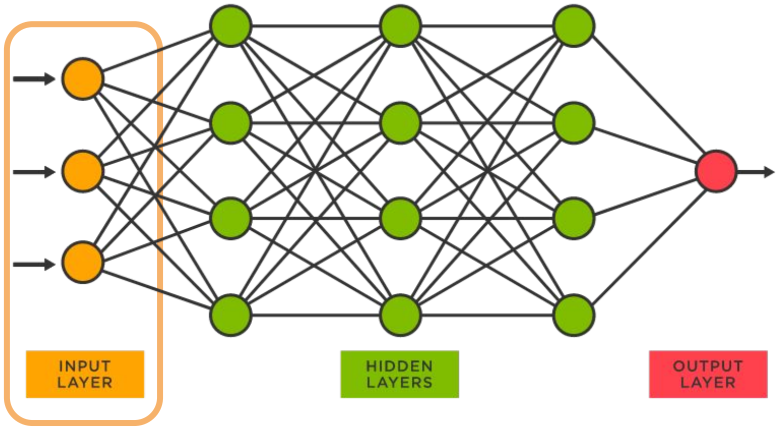
- ML algorithm that can learn complex functions to represent data and provide an output for the data
  - Attempts to simulate biological neural networks but does so poorly



# Neural Networks

- ML algorithm that can learn complex functions to represent data and provide an output for the data
  - Attempts to simulate biological neural networks but does so poorly

- Input layer: data point *x* with *d* features, one for each node

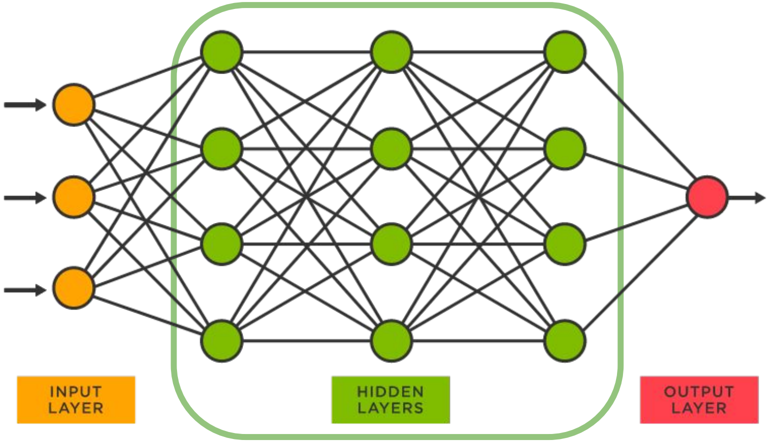


# Neural Networks

- ML algorithm that can learn complex functions to represent data and provide an output for the data
  - Attempts to simulate biological neural networks but does so poorly

- Input layer: data point *x* with *d* features, one for each node

- Hidden layers: apply linear functions and *nonlinear* activation functions to the features of the data



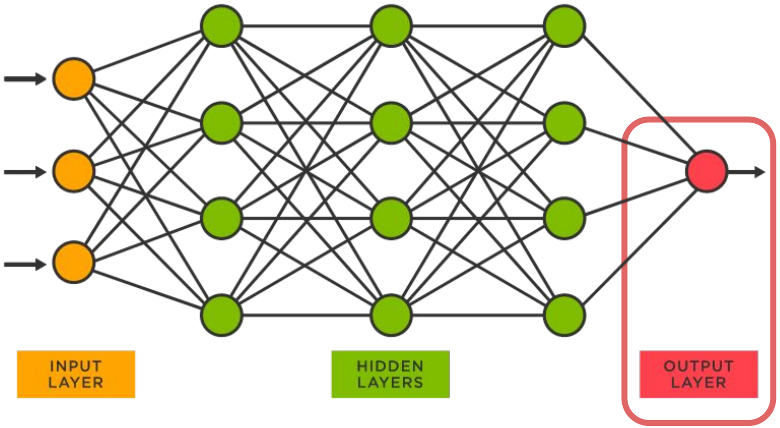
# Neural Networks

- ML algorithm that can learn complex functions to represent data and provide an output for the data
  - Attempts to simulate biological neural networks but does so poorly

- Input layer: data point *x* with *d* features, one for each node

- Hidden layers: apply linear functions and *nonlinear* activation functions to the features of the data

- Output layer: applies an activation function to produce the label/output value



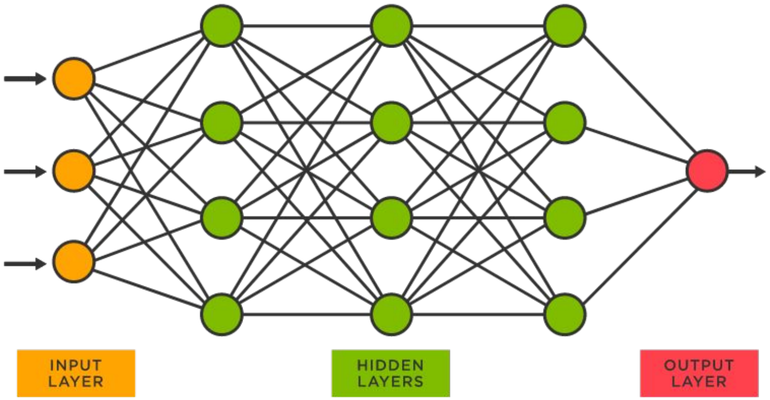
# Neural Networks

- ML algorithm that can learn complex functions to represent data and provide an output for the data
    - Attempts to simulate biological neural networks but does so poorly

- Input layer: data point *x* with *d* features, one for each node

- Hidden layers: apply linear functions and *nonlinear* activation functions to the features of the data

- Output layer: applies an activation function to produce the label/output value




$$\mathrm{softmax}(W^T(\mathrm{act}(W^T(x))))$$

# Activation Functions

- Sigmoid: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$
- Rectified Linear Unit (ReLU): $\mathrm{ReLU}(x) = \max(x, 0)$
- Tanh: $\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
- Softmax: $\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}}$