Building a Classification Model Using PyTorch

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Classification using neural networks

Softmax and cross-entropy loss

LogSoftmax and negative log likelihood loss

Extending the PyTorch Module base class to implement classification

Working with activation functions

Working with dropout

Softmax for Classification

Classification

Spam or ham?

Dog or pony?

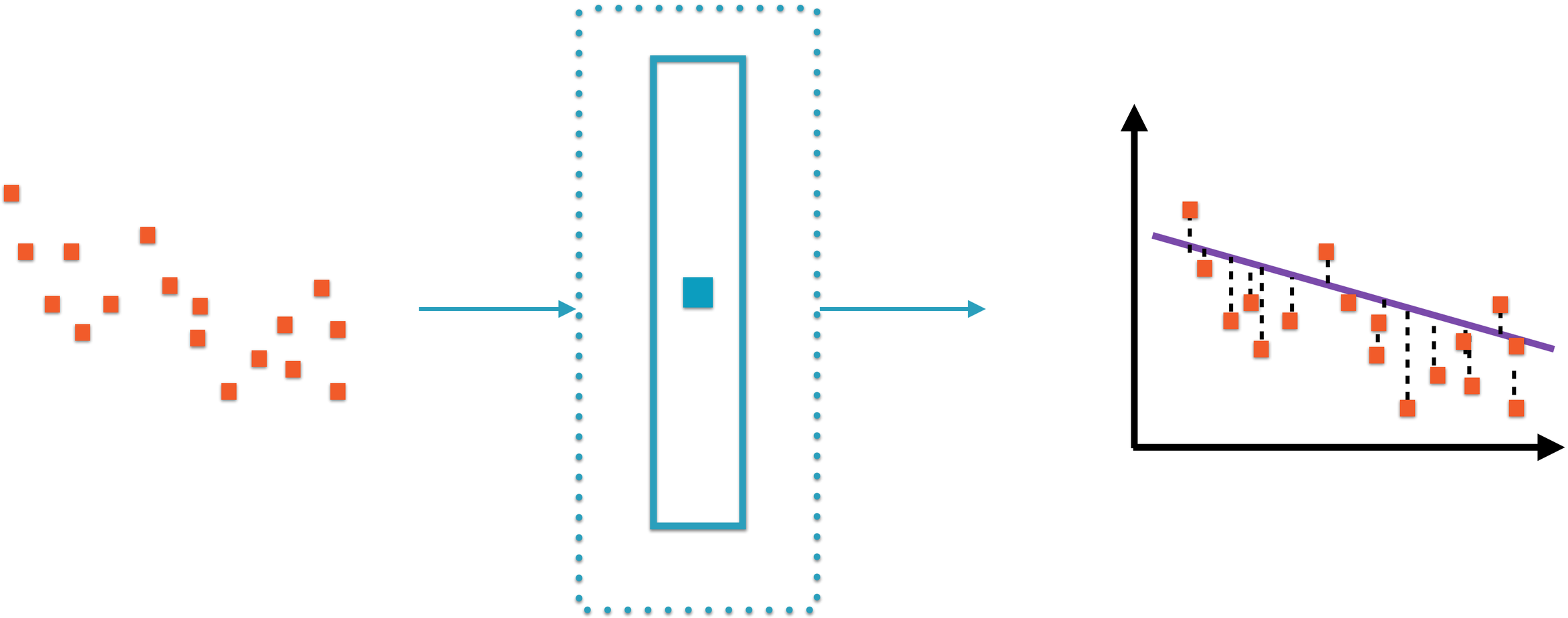
Which letter of the alphabet?

Buy, sell or hold?

Output is one of many discrete categories

Categorical output

Linear Regression with One Neuron

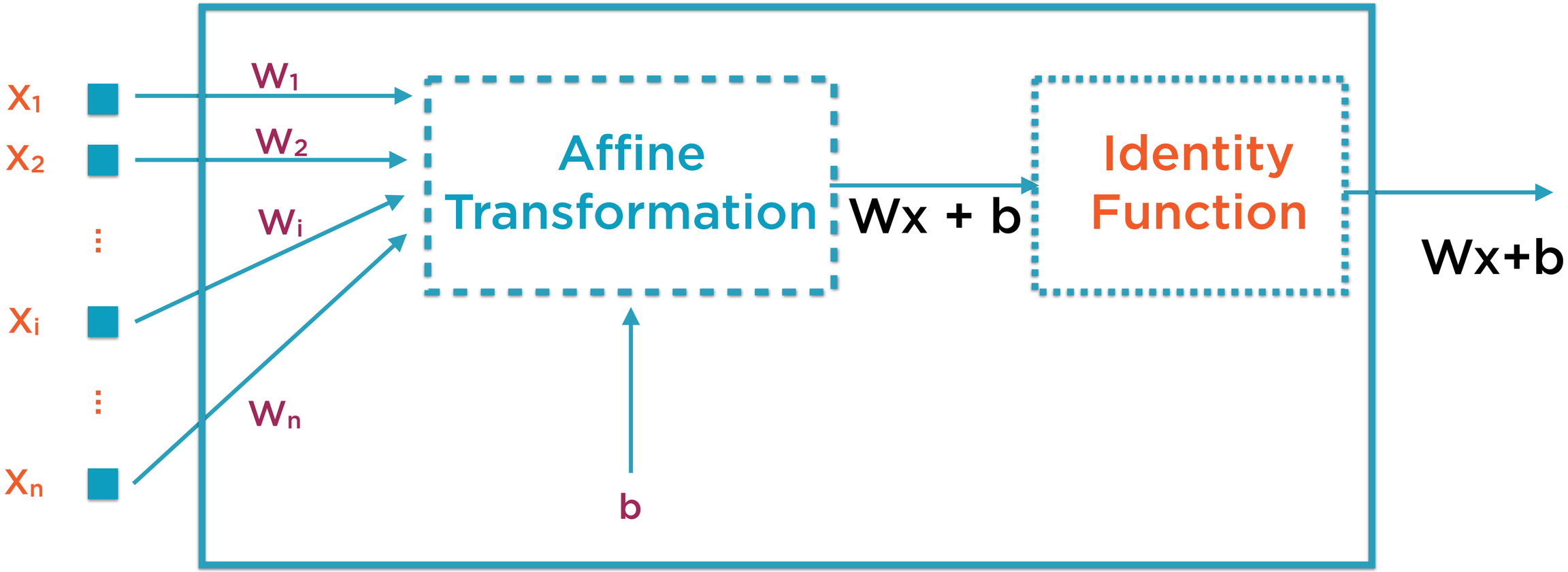


Set of Points

Single Neuron

Regression Line

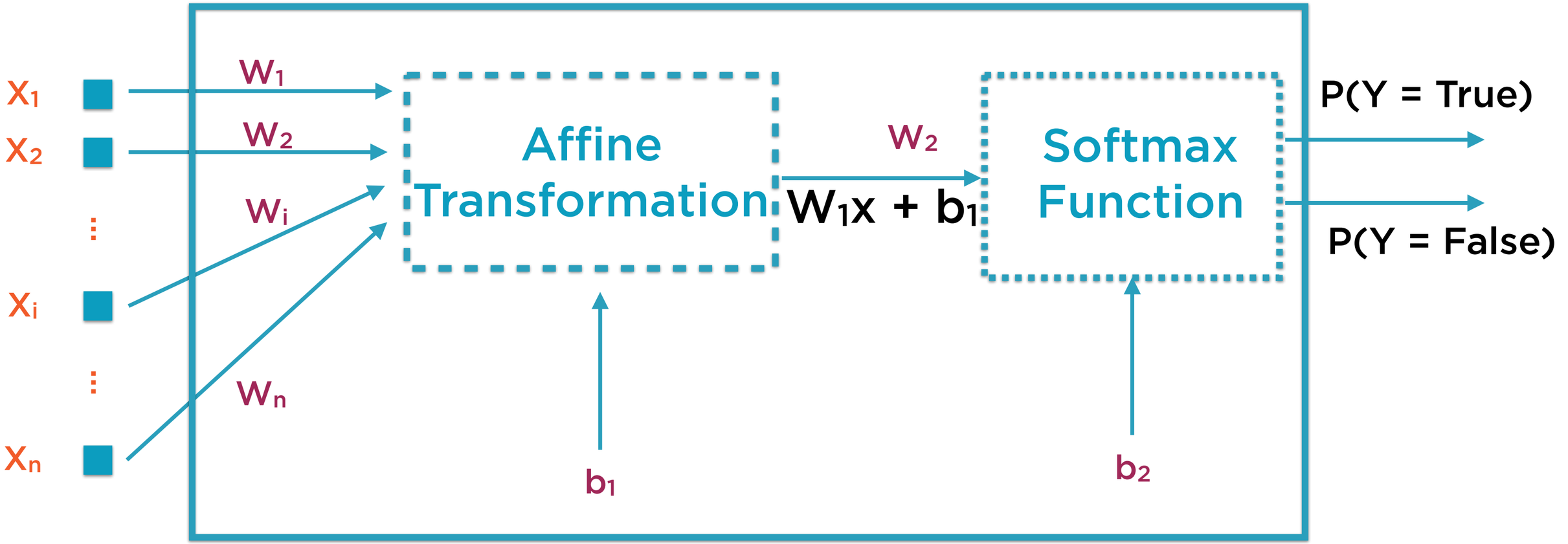
Linear Regression with One Neuron



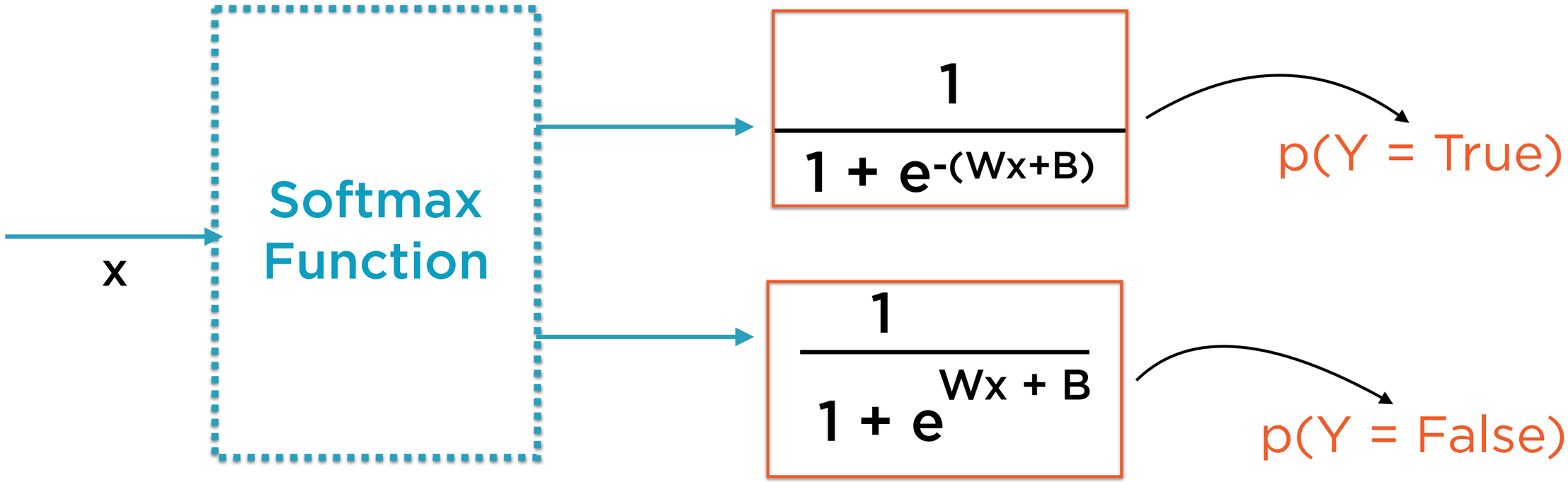
Linear Regression with One Neuron

Linear Classification with One Neuron

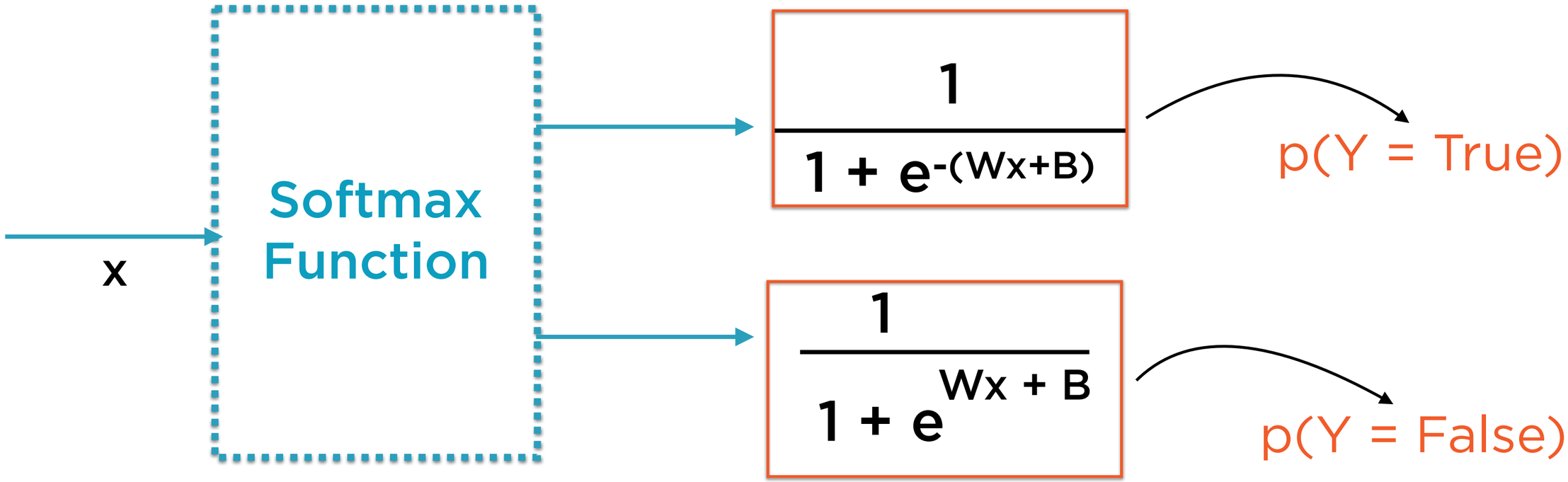
Linear Classification with One Neuron



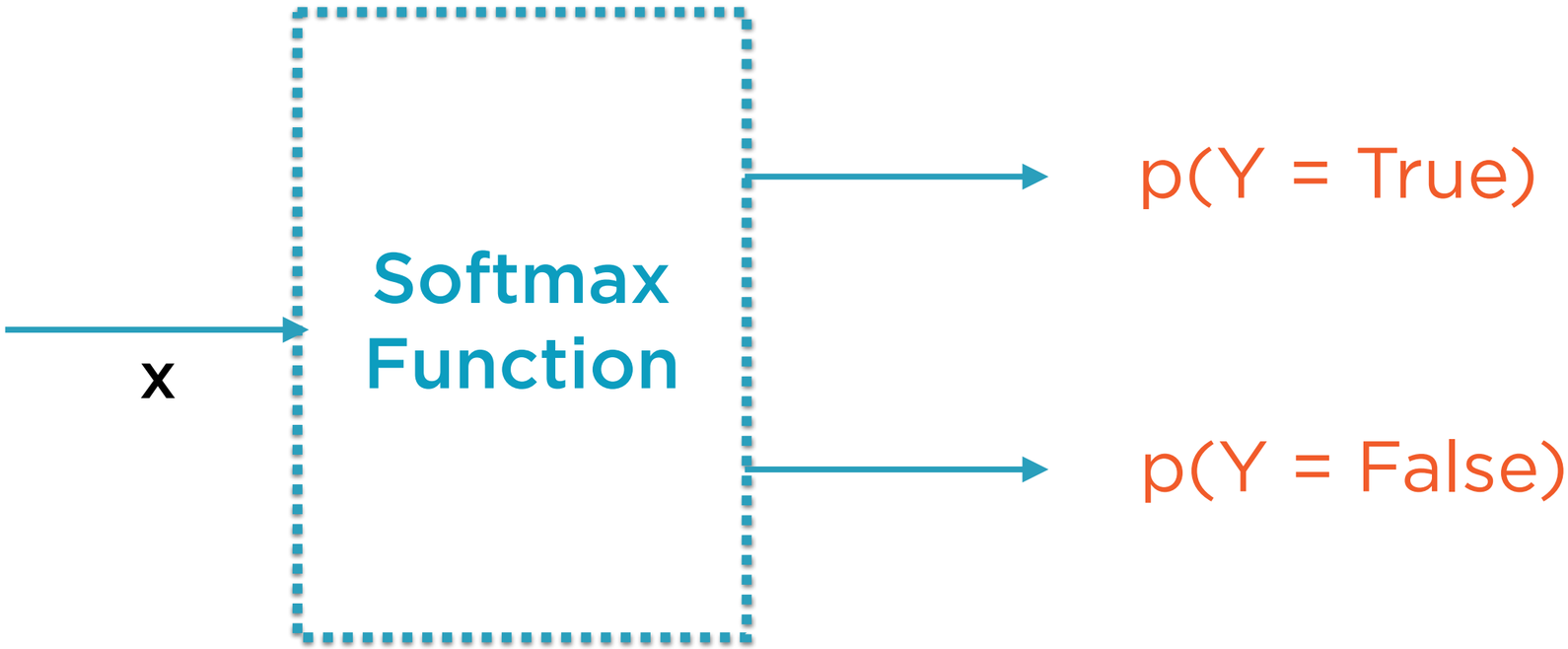
SoftMax for True/False Classification



Classifier Models Output Probabilities

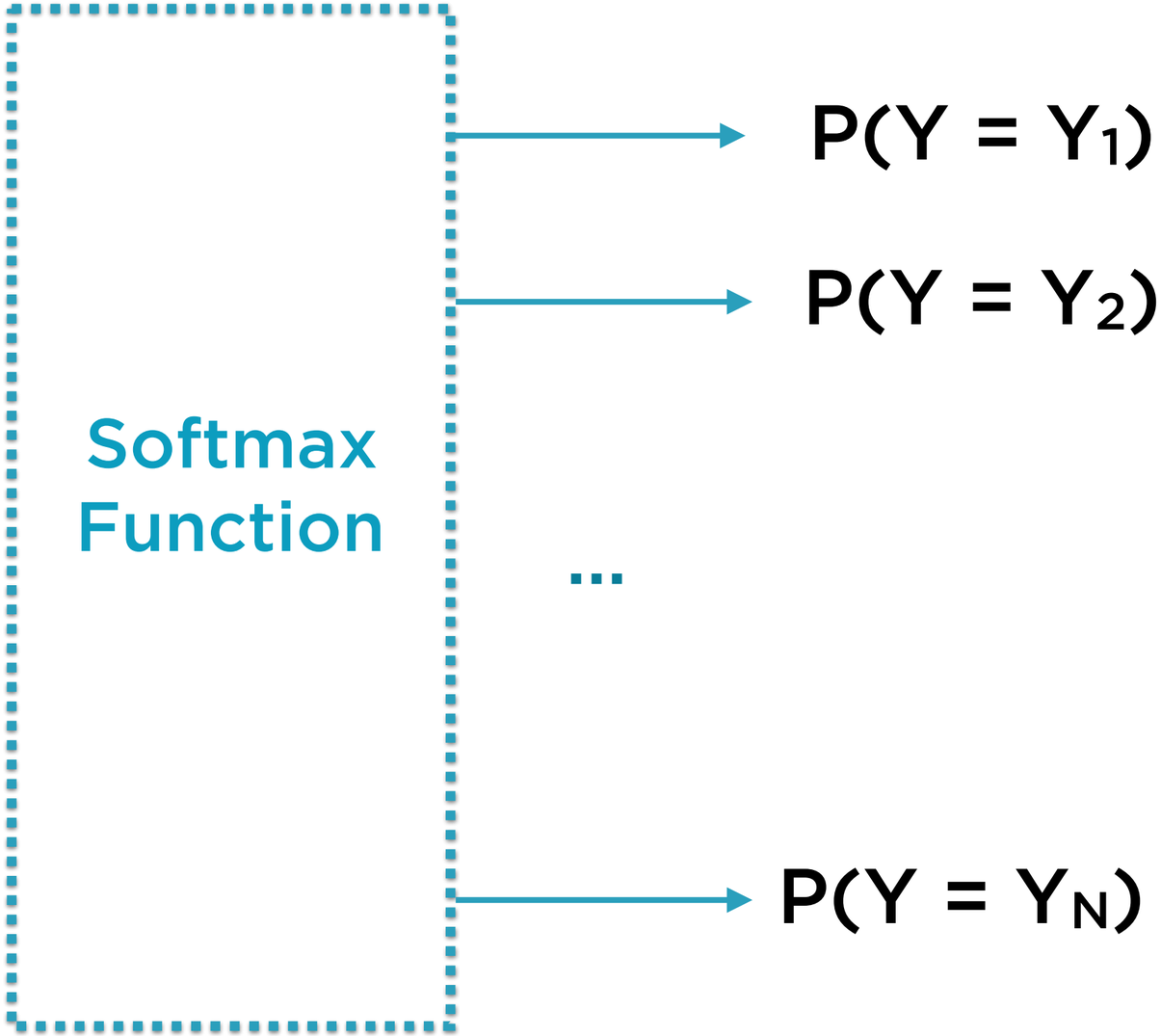


Classifier Models Output Probabilities

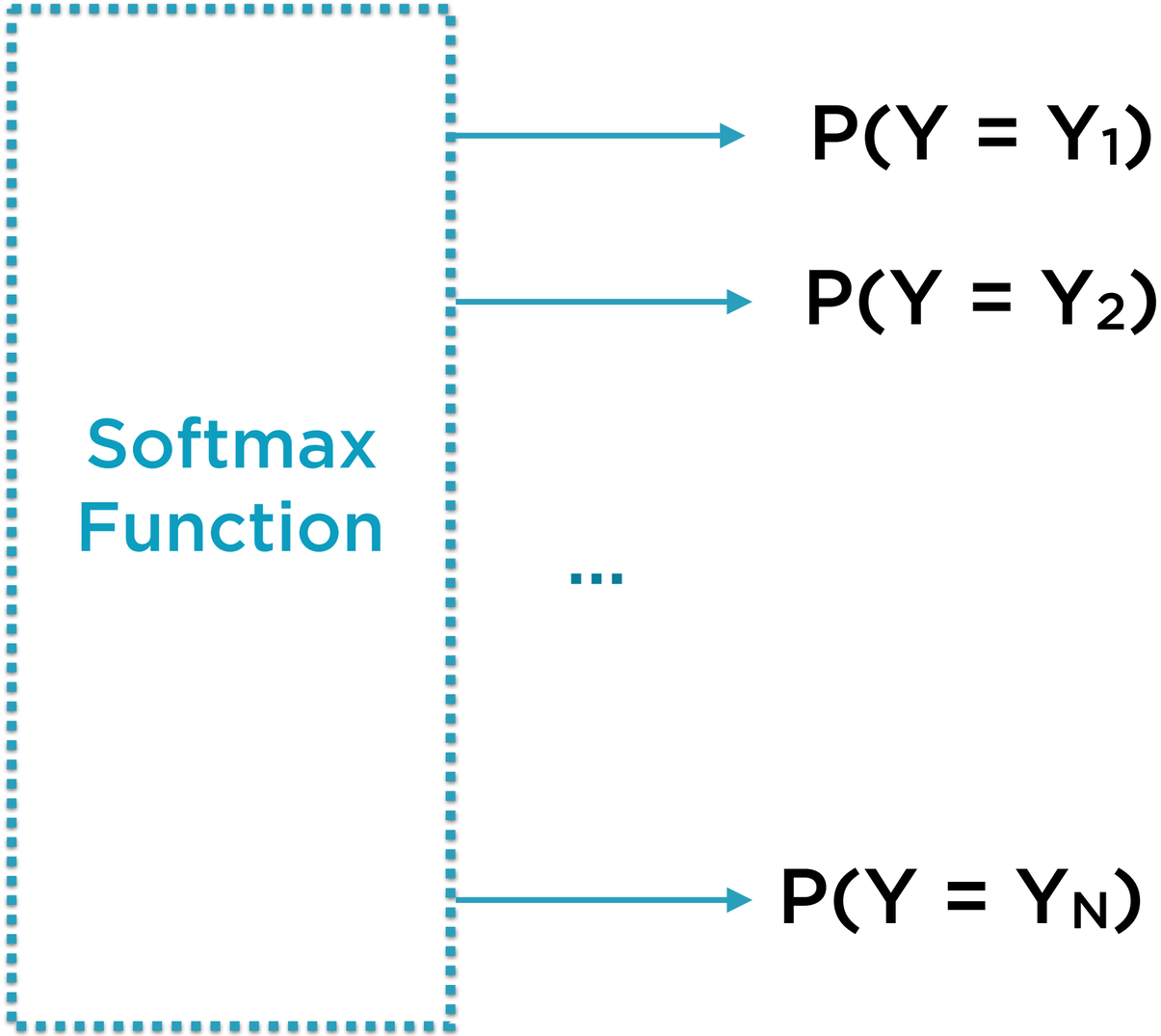


If p(Y = True) > p(Y = False) then output is classified as true

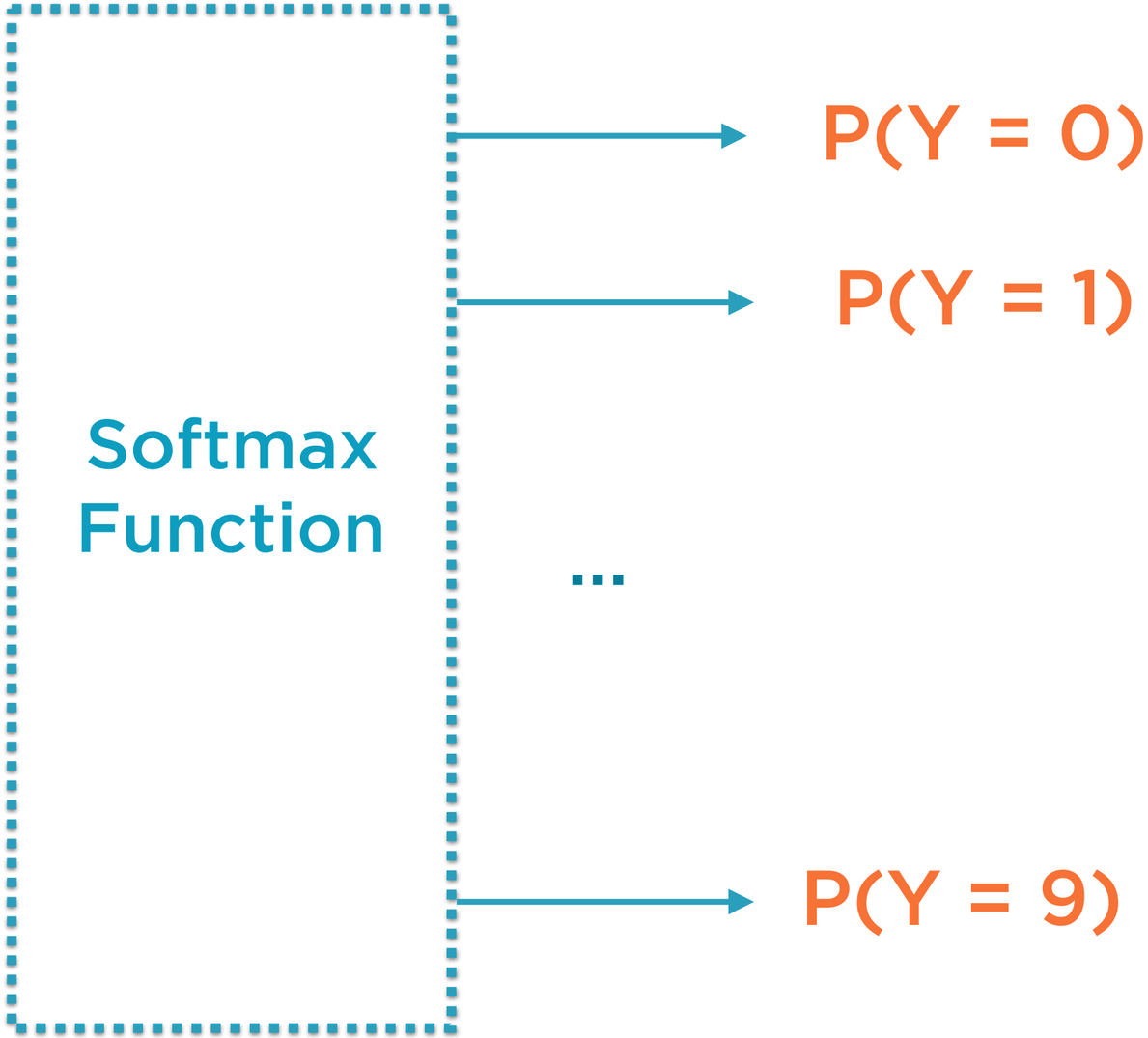
SoftMax N-category Classification



Prediction is Label with Highest Probability



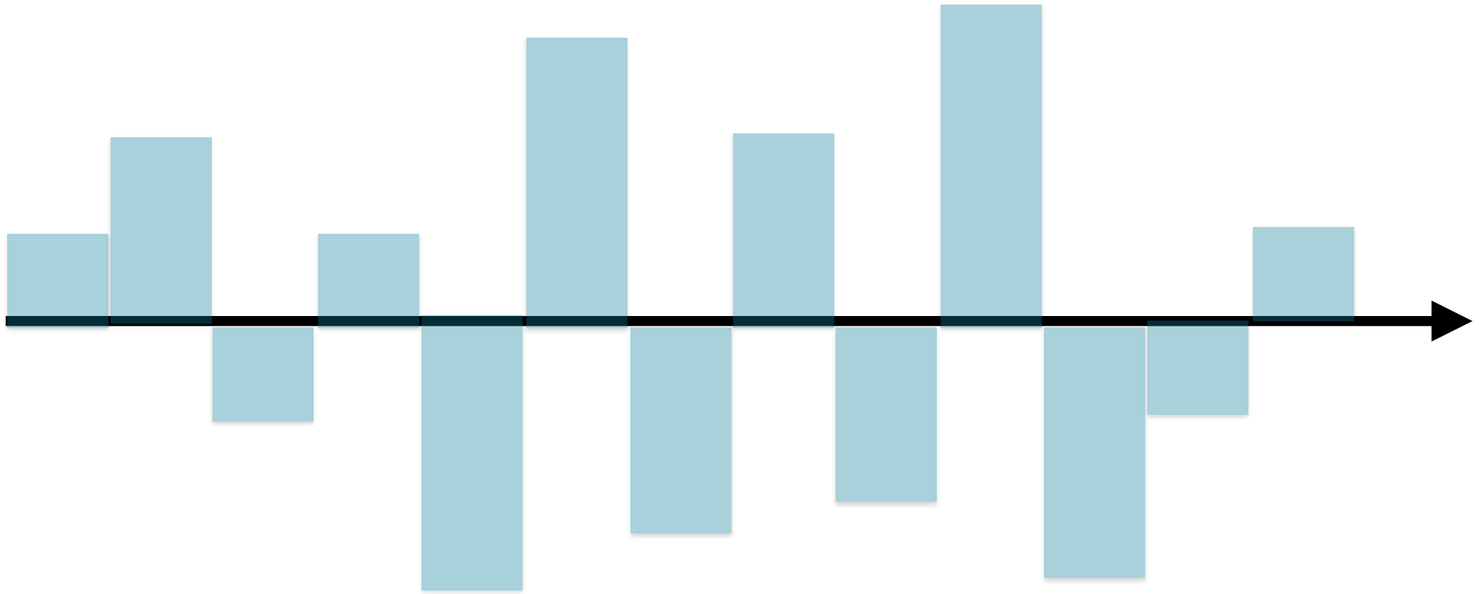
SoftMax for Digit Classification



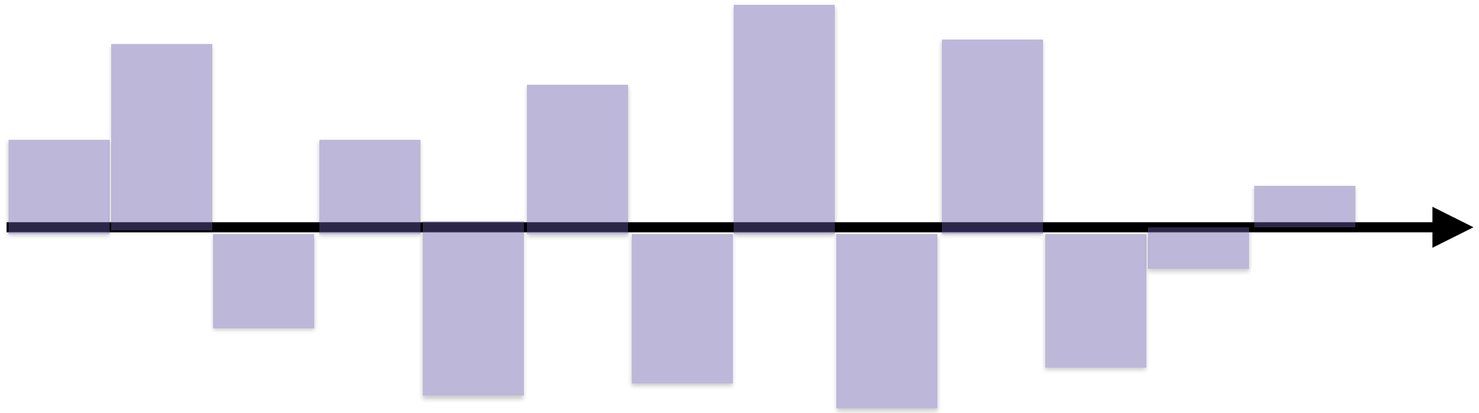
Cross entropy: Measure of how different two probability distributions are

Intuition: Low Cross Entropy

Y_{actual}



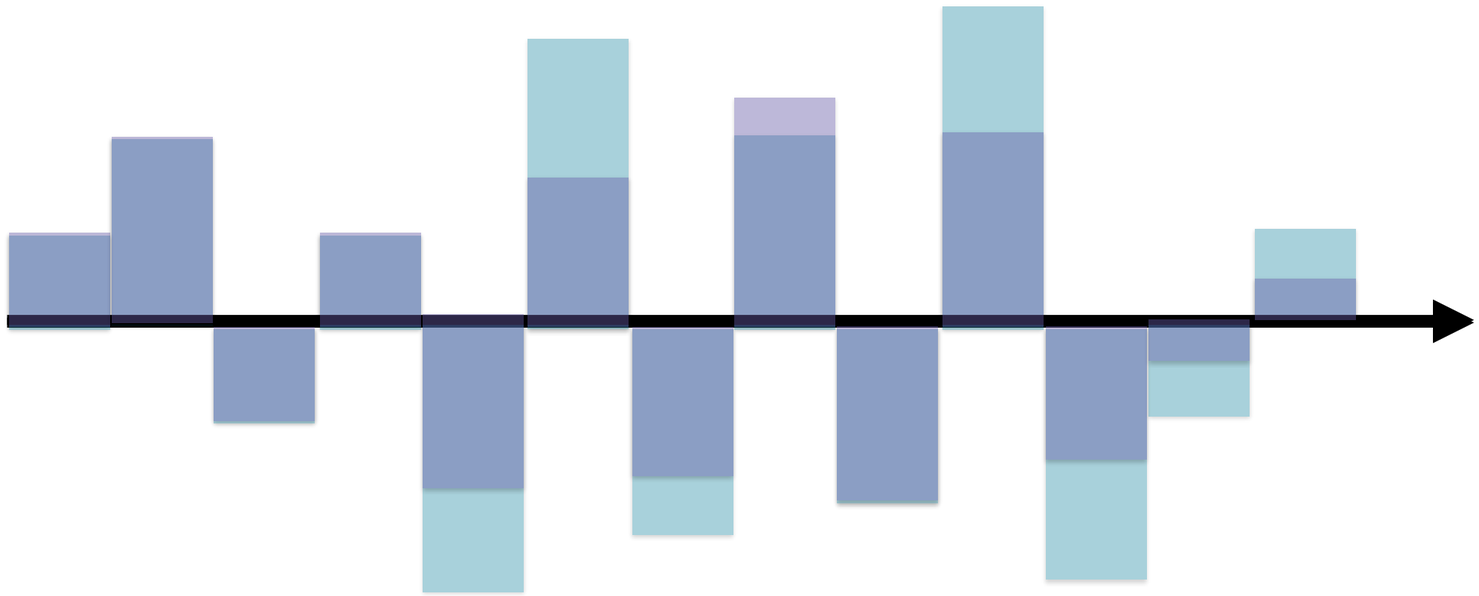
$Y_{predicted}$



Intuition: Low Cross Entropy

Y_{actual}

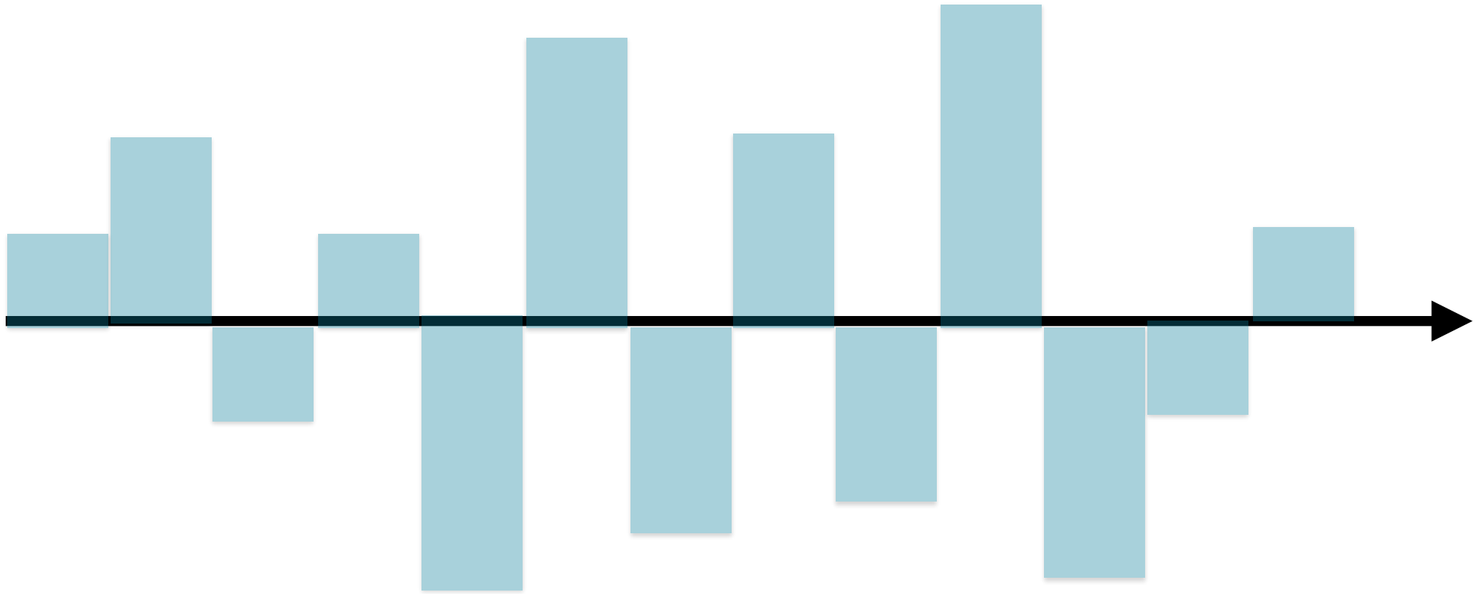
$Y_{predicted}$



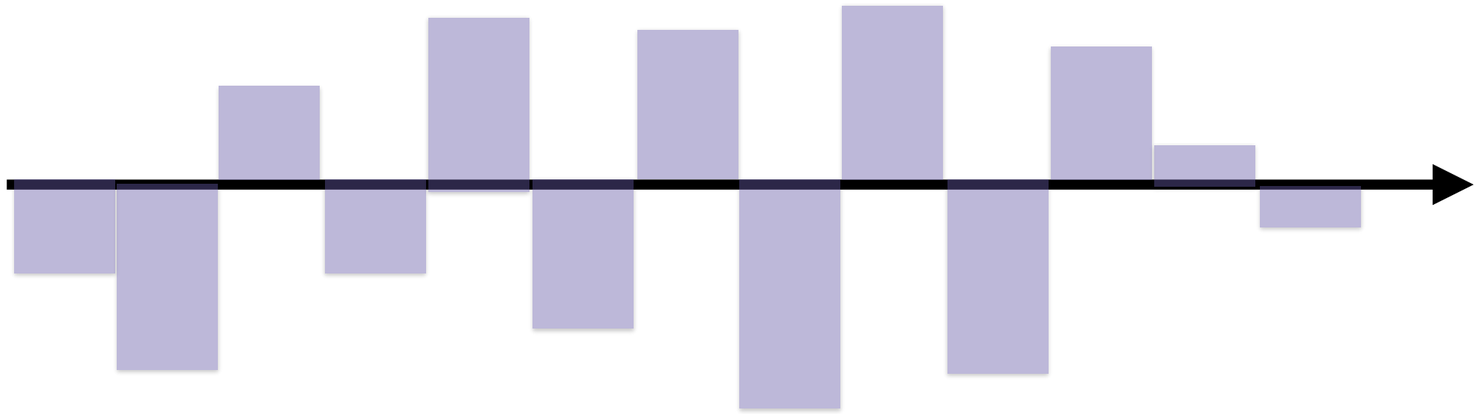
The labels of the two series are in-synch

Intuition: High Cross Entropy

Y_{actual}



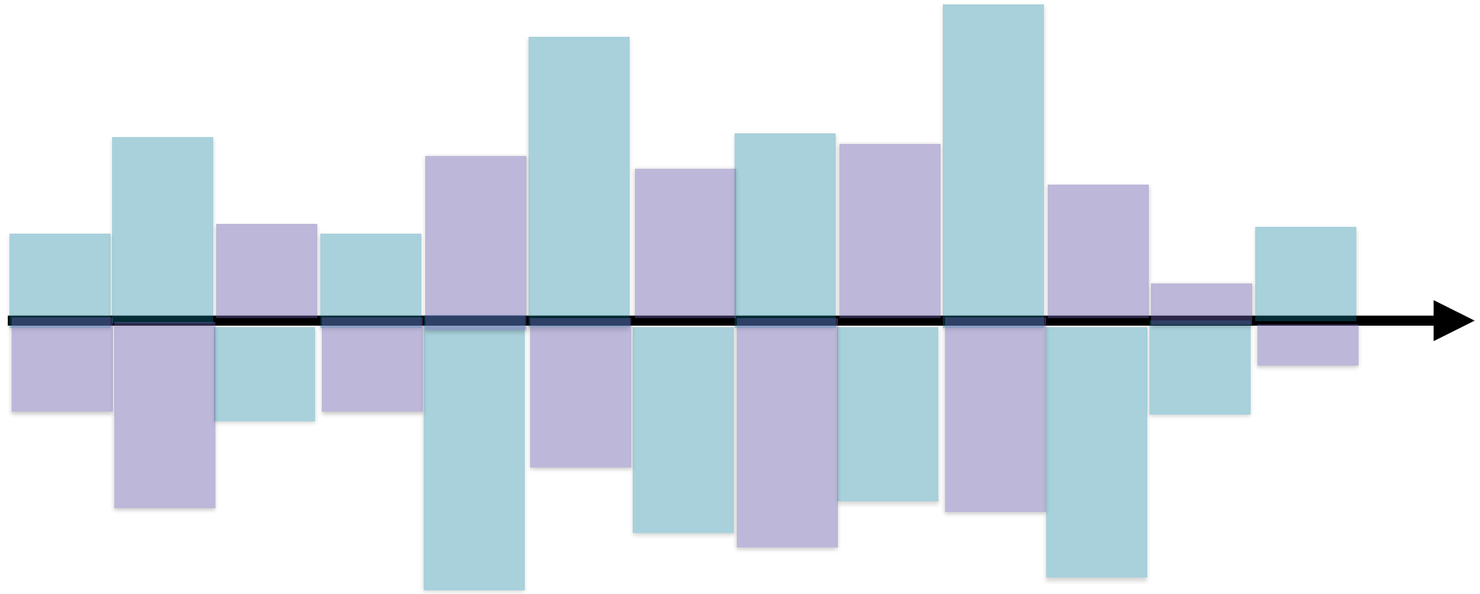
$Y_{predicted}$



Intuition: High Cross Entropy

Y_{actual}

$Y_{predicted}$



The labels of the two series are out-of-synch

Linear Classification as an Optimization Problem

Linear Classification as an Optimization Problem

Objective Function

Minimize cross-entropy between
Y_{actual} and $Y_{predicted}$

Linear Classification as an Optimization Problem

Objective Function

Minimize cross-entropy between Y_{actual} and $Y_{predicted}$

Constraints

Express relationship as an exponential one

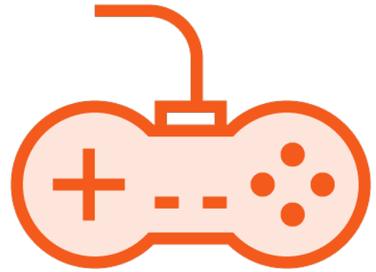
Linear Classification as an Optimization Problem

Objective Function

Minimize cross-entropy between Y_{actual} and $Y_{predicted}$

Constraints

Express relationship as an exponential one



Decision Variables

Find “best” values for parameters

Softmax or Log Softmax as Output Layer?

Softmax

Output layer of NN is Softmax

Slightly less stable

Log Softmax

Output layer of NN is Log Softmax

Slightly more stable, nicer properties

Softmax or Log Softmax as Output Layer?

Softmax

Use cross-entropy as loss function

Objective is to minimize cross-entropy

Need just 1 output layer (Softmax)

Log Softmax

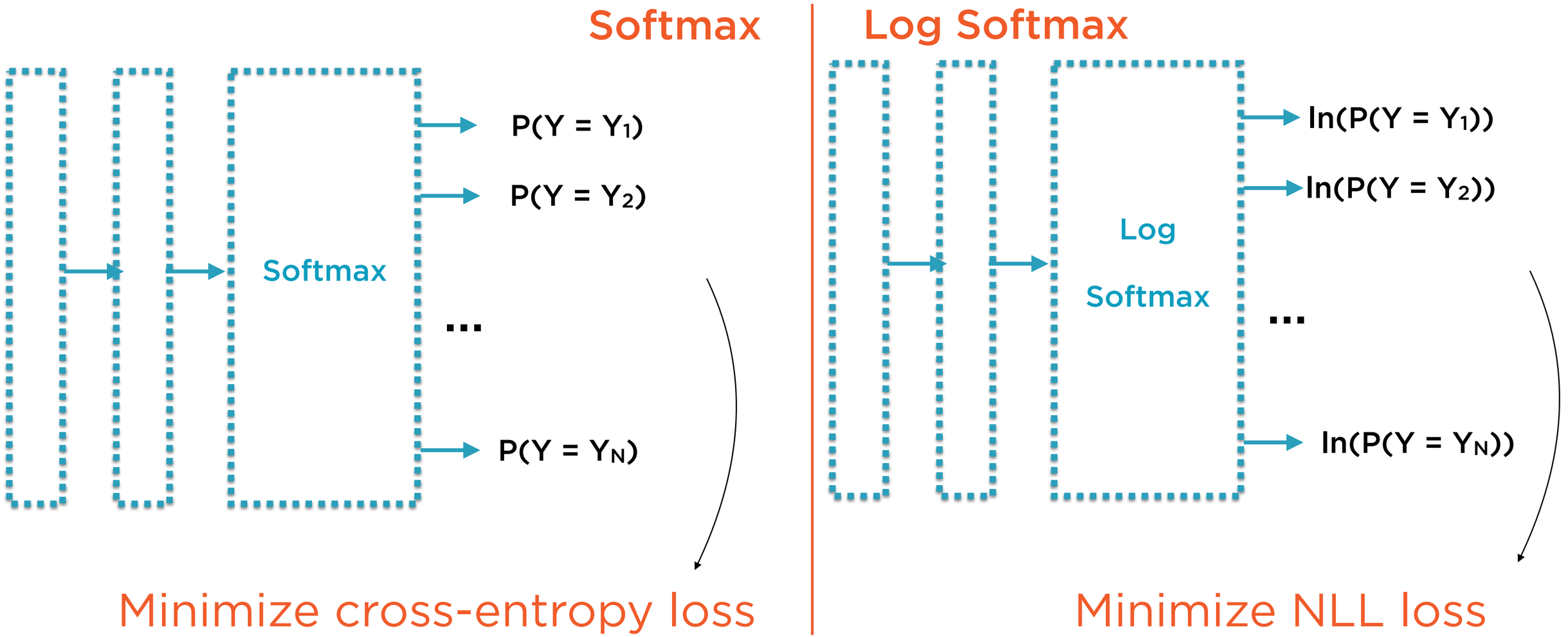
Use NLL (negative log likelihood)

Mathematically equivalent (almost)

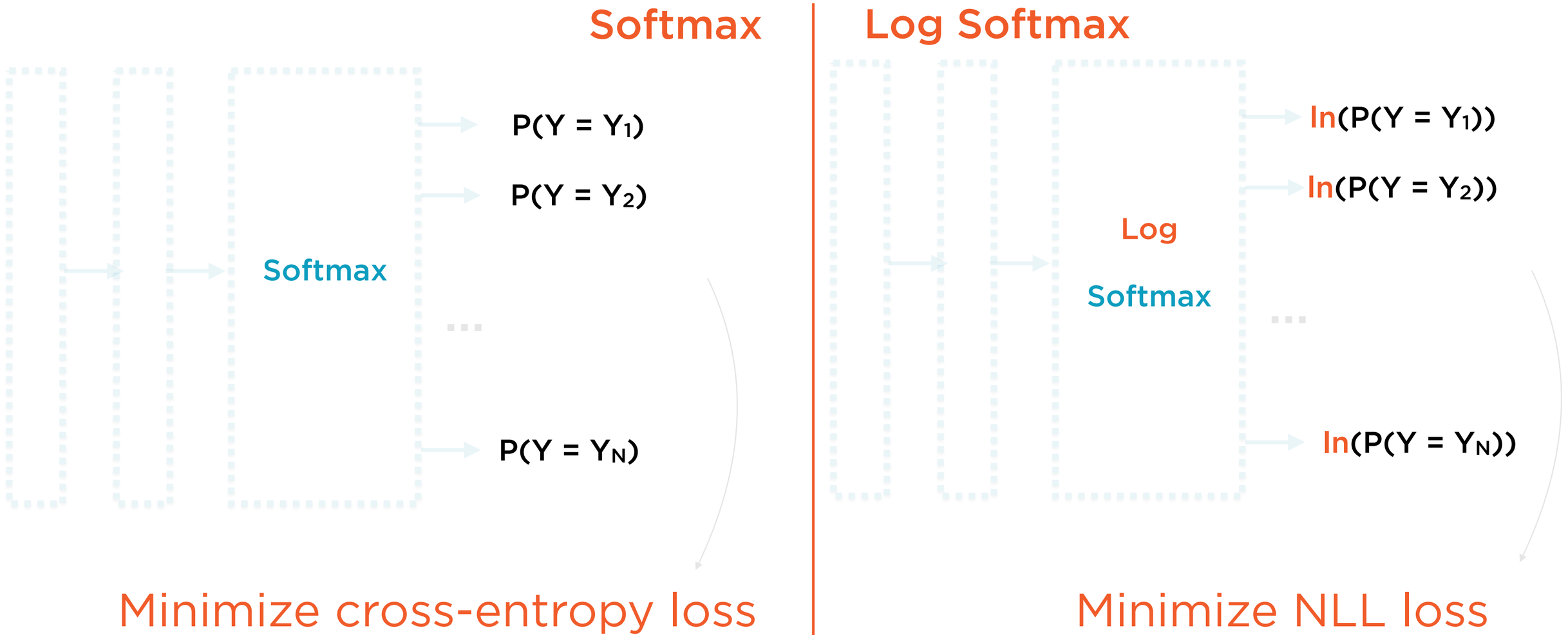
Might need additional output layer - to calculate log

But in PyTorch just use LogSoftMax

Softmax or Log Softmax as Output?



Softmax or Log Softmax as Output?



Using (Output Layer = LogSoftmax and Loss = NLL) is equivalent* to using (Output Layer = Softmax and Loss = Cross-entropy)

Activation Functions in Neurons

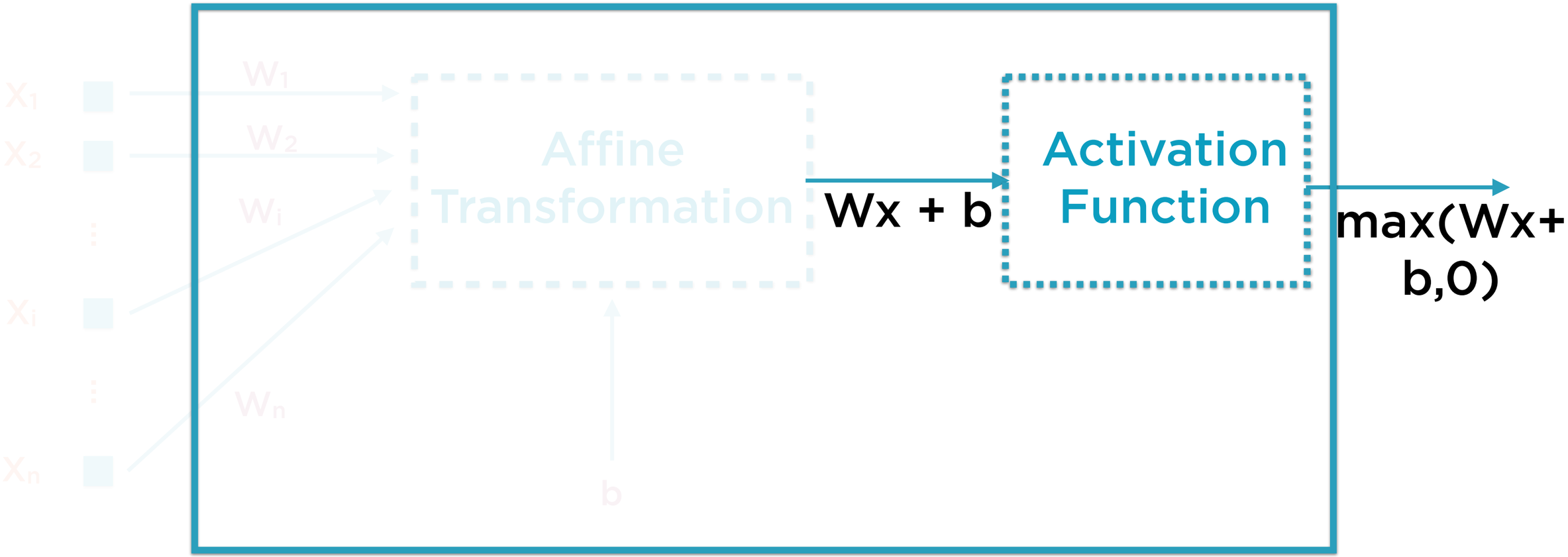
Operation of a Single Neuron

Each neuron only applies two simple functions to its inputs

Operation of a Single Neuron

The affine transformation alone can only learn linear relationships between the inputs and the output

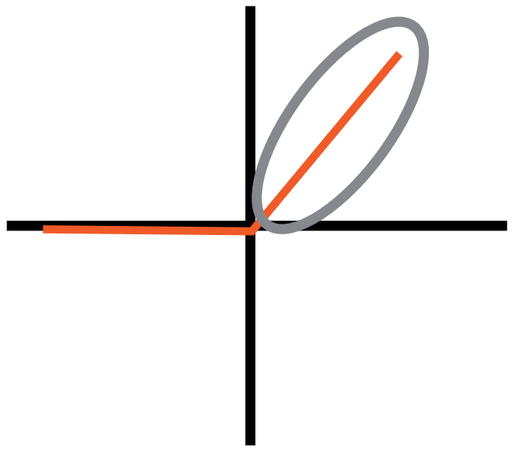
Operation of a Single Neuron



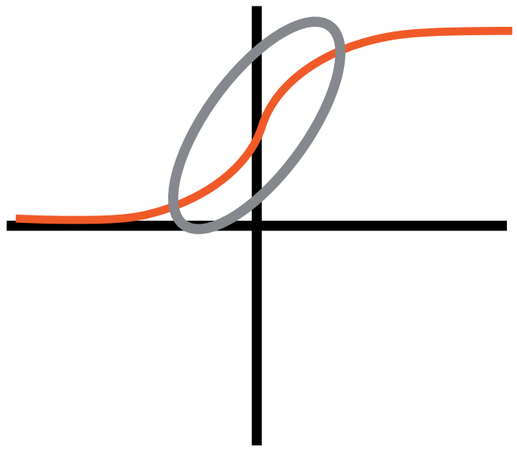
The combination of the affine transformation and the activation function can learn any arbitrary relationship

Common Activation Functions

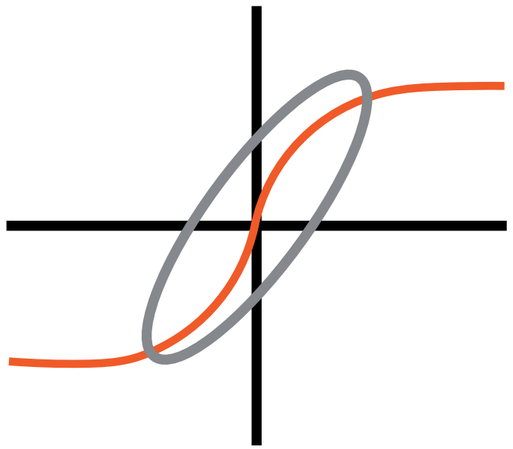
ReLU



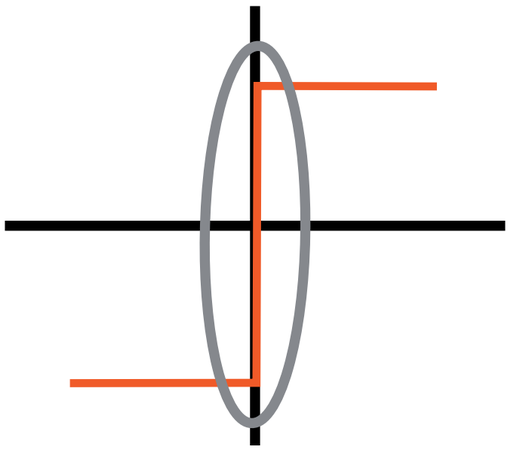
logit



tanh



step



Notice how activations functions have a gradient, this gradient allows them to be sensitive to input changes

Confusion Matrix in Evaluating Classifiers

Confusion Matrix

Predicted Labels

Actual Label

	Cancer	No Cancer
Cancer	**10 instances**	**4 instances**
No Cancer	**5 instances**	**1000 instances**

Confusion Matrix

Predicted Labels

Actual Label

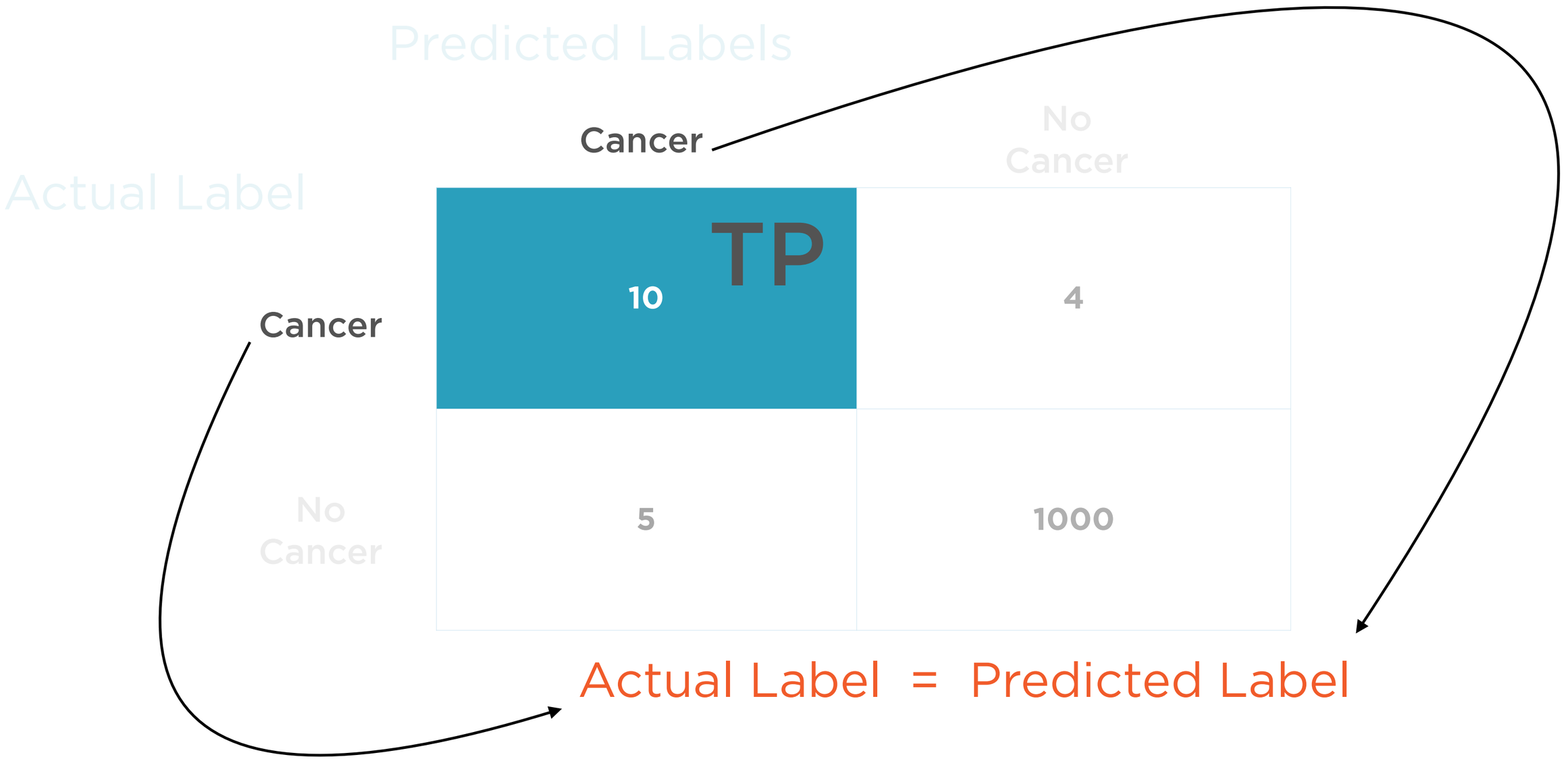
	Cancer	No Cancer
Cancer	**10** instances	**4** instances
No Cancer	**5** instances	**1000** instances

Confusion Matrix

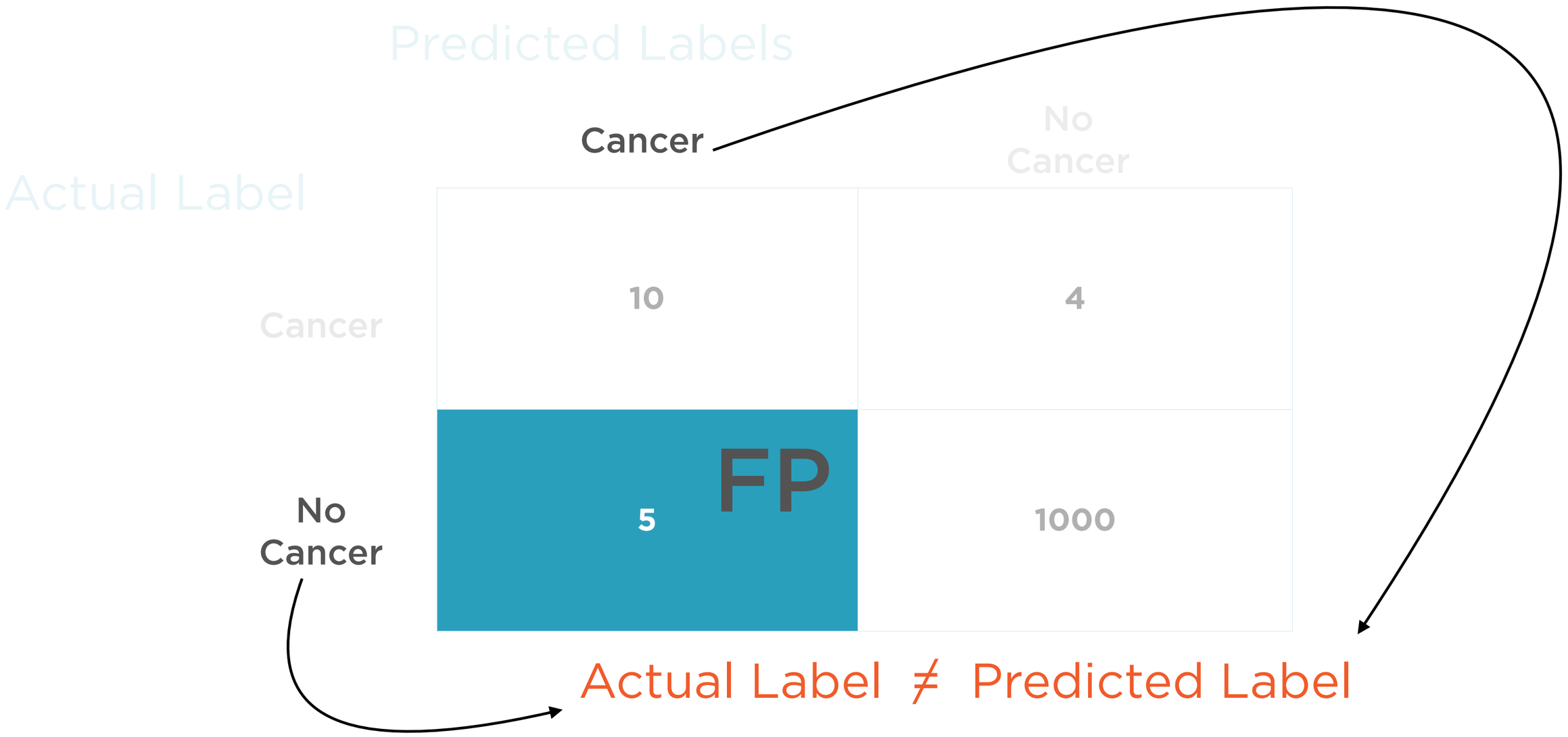
Predicted Labels

Actual Label	Cancer	No Cancer
Cancer	**10**	**4**
No Cancer	**5**	**1000**

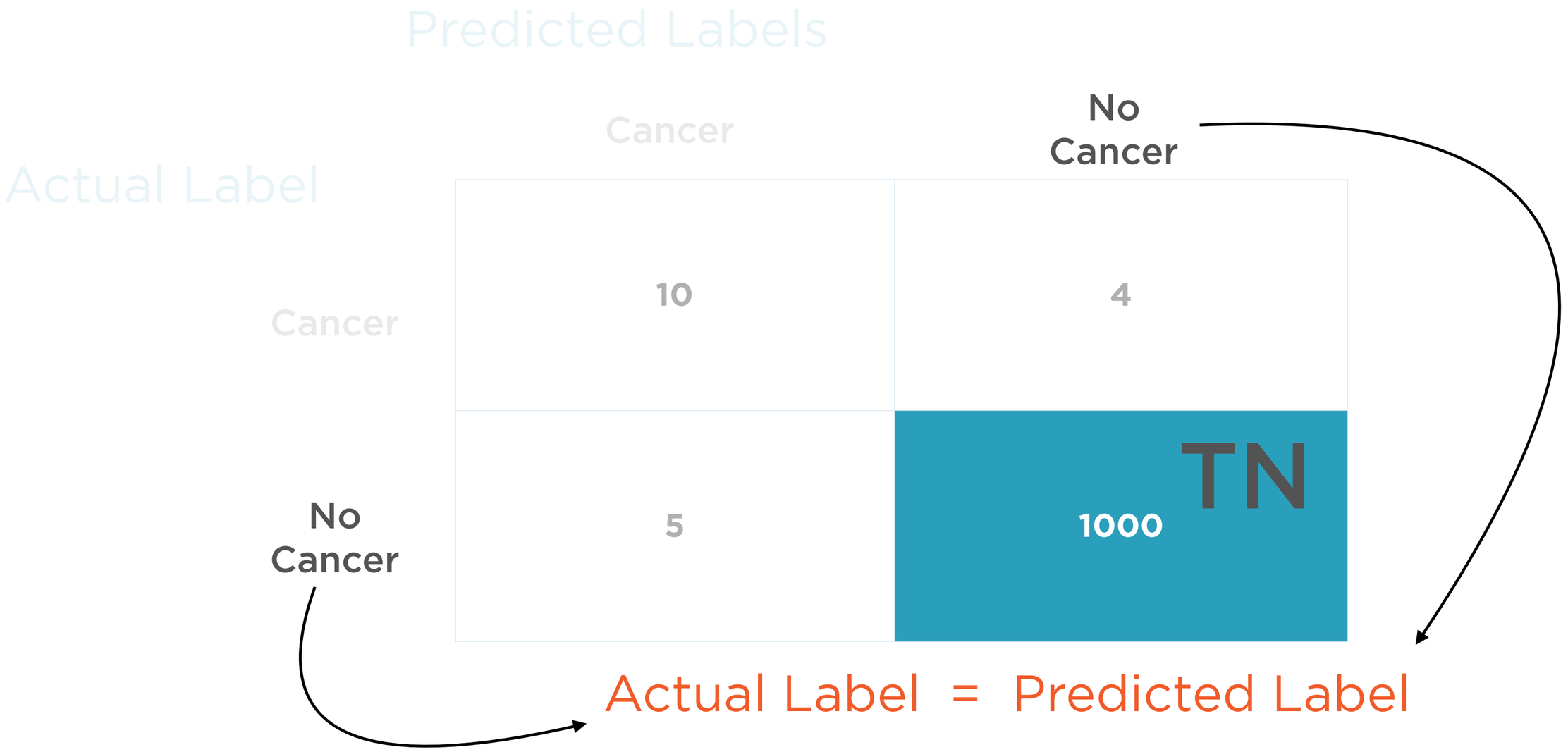
True Positive



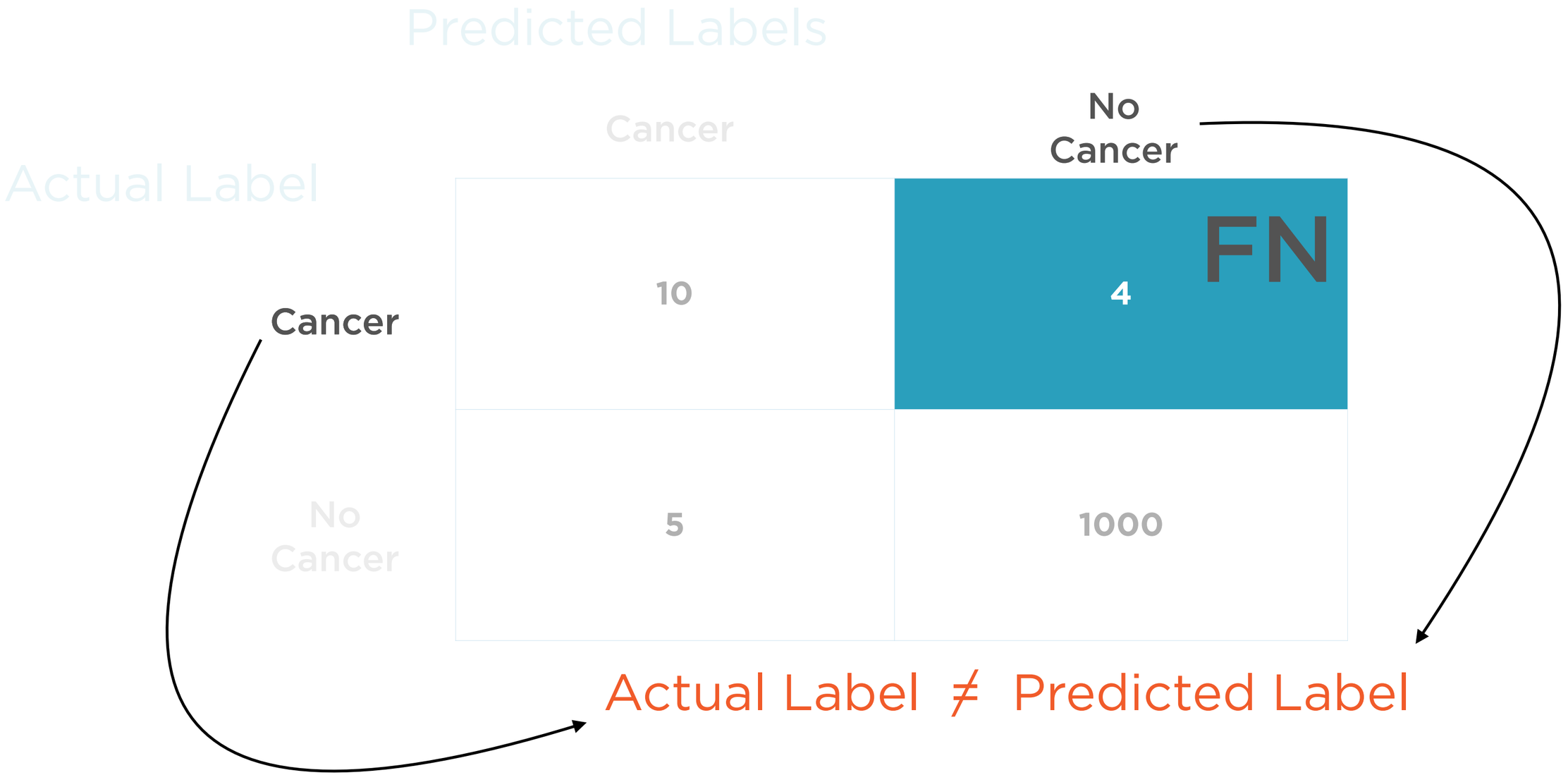
False Positive



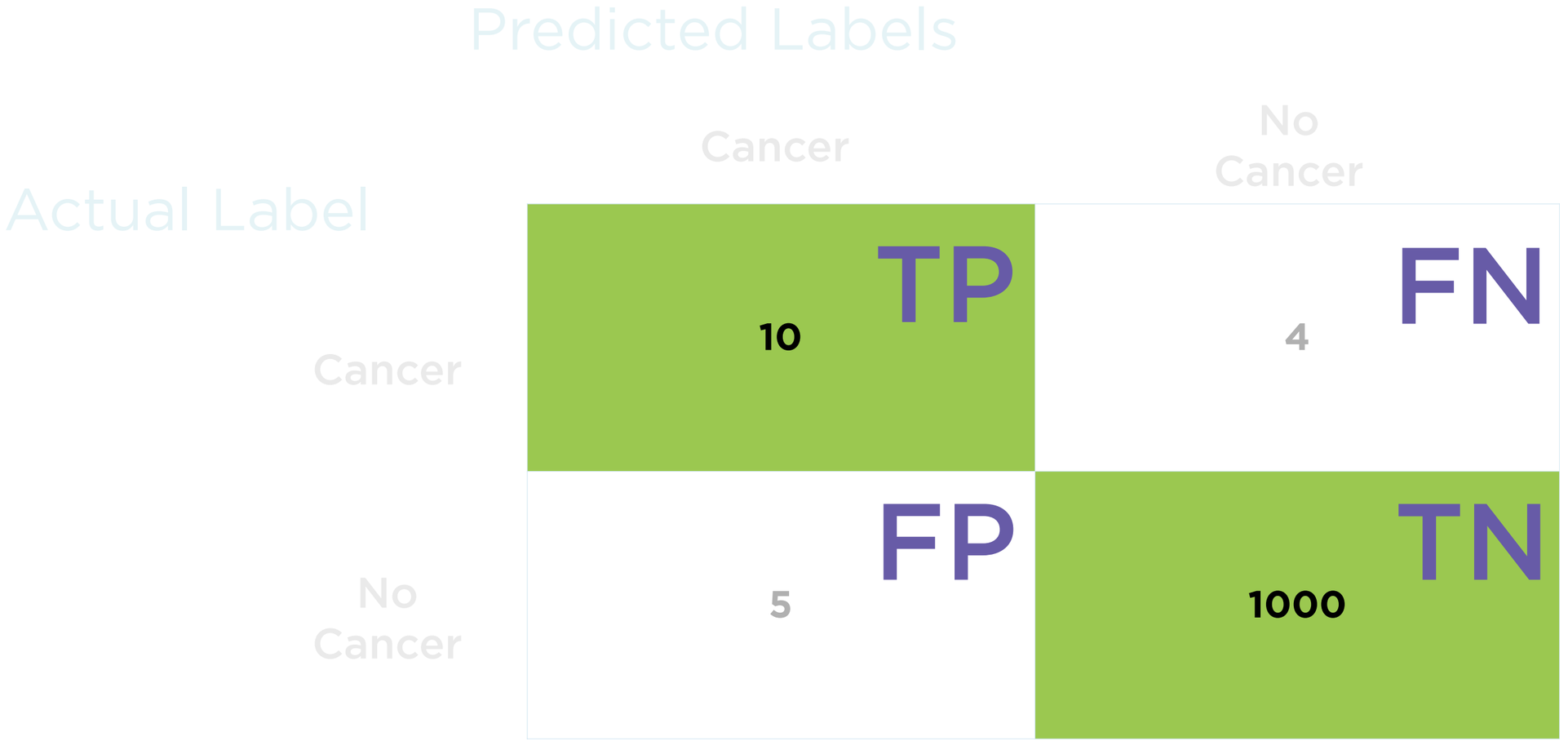
True Negative



False Negative

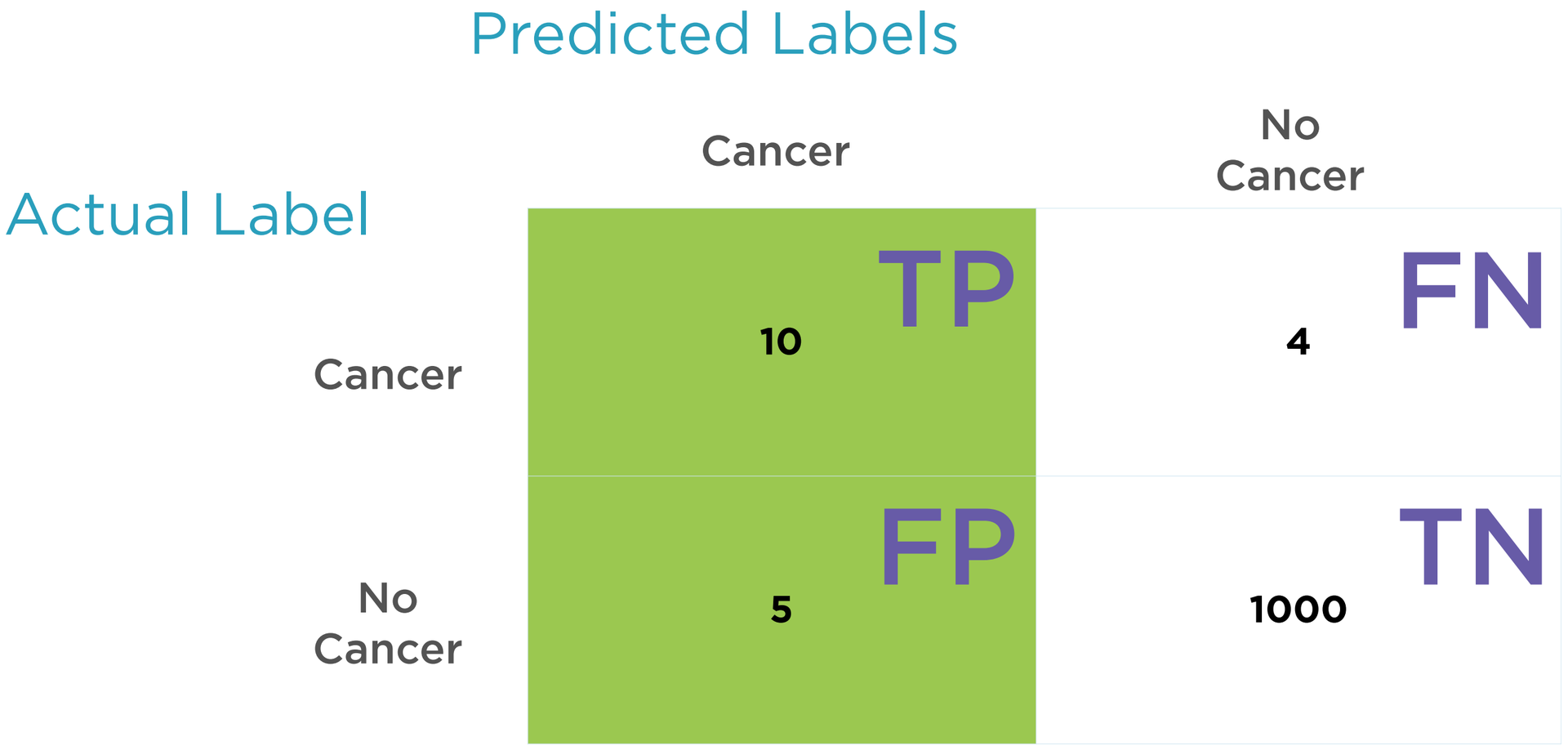


Accuracy



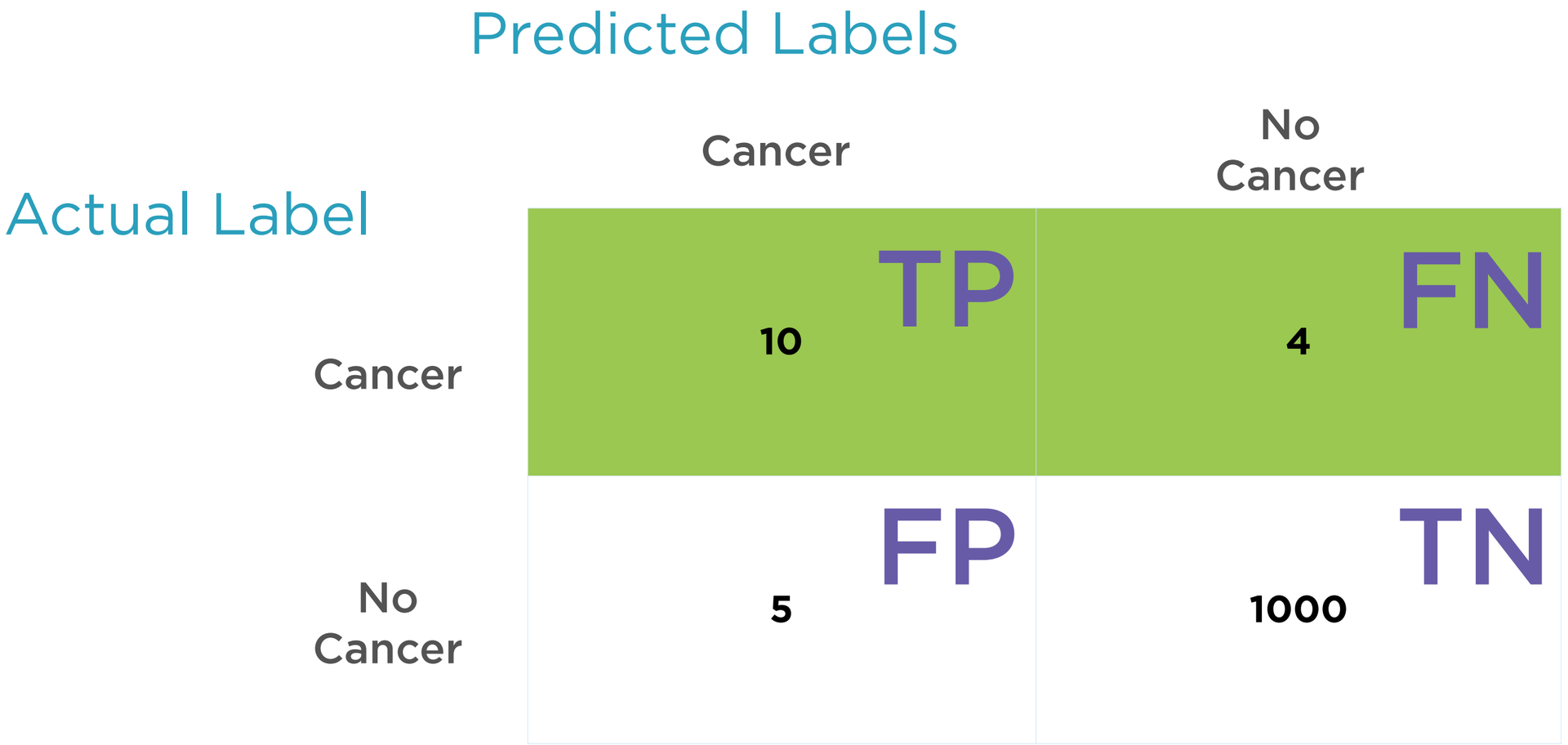
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Num Instances}} = \frac{1010}{1019} = 99.12\%$$

Precision



Precision = Accuracy when classifier flags cancer

Recall



Recall = Accuracy when cancer actually present

Demo

Building a classification model using a custom neural network

Summary

- Classification using neural networks
- Softmax and cross-entropy loss
- LogSoftmax and negative log likelihood loss
- Extending the PyTorch Module base class to implement classification
- Working with activation functions
- Working with dropout

Related Courses

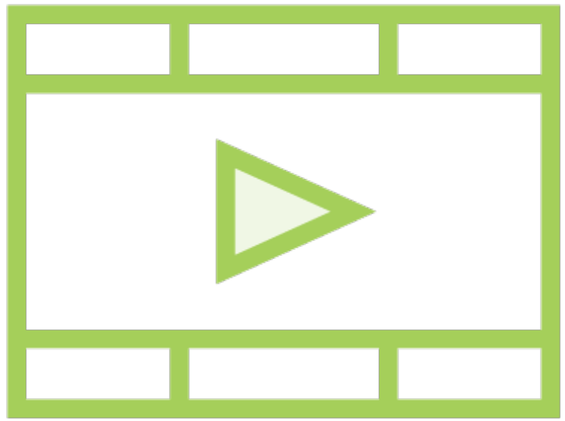


Image Classification with PyTorch

Natural Language Processing with PyTorch

Deploying PyTorch Models