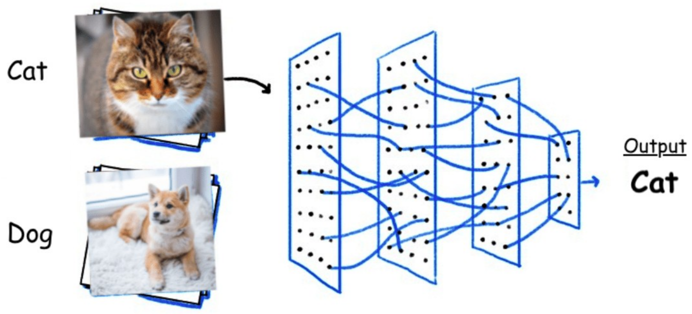
Module 2

Binary Classification

Binary Classification



Types of Classification Problems

- The 3 main classification problems are:

Binary Classification

- Spam
- Not spam

Multiclass Classification

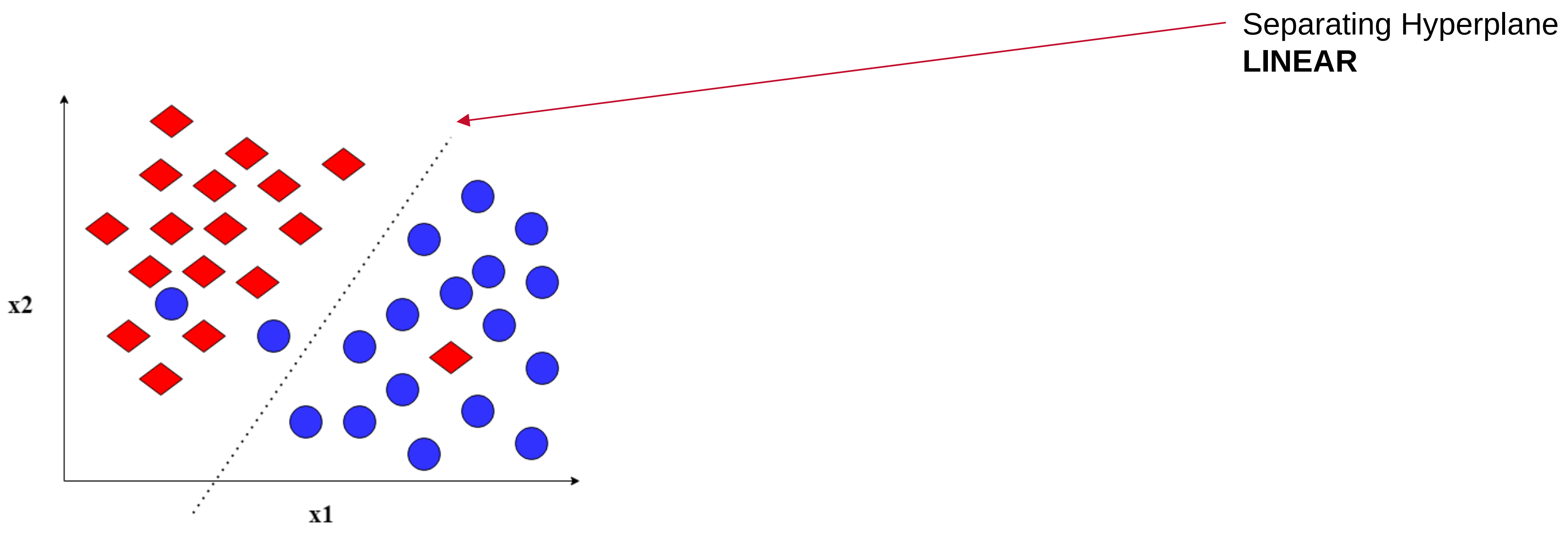
- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Multi-label Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

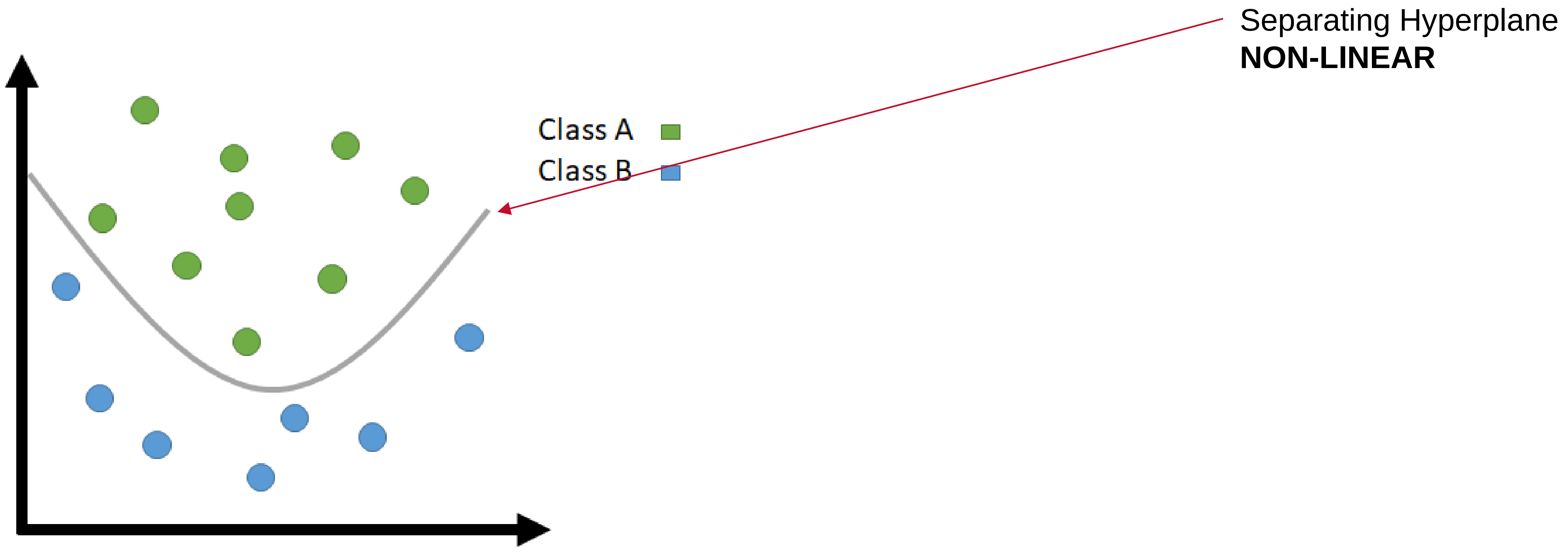
Binary Classification

- is the task of classifying the elements of a set into one of two groups (each called class).



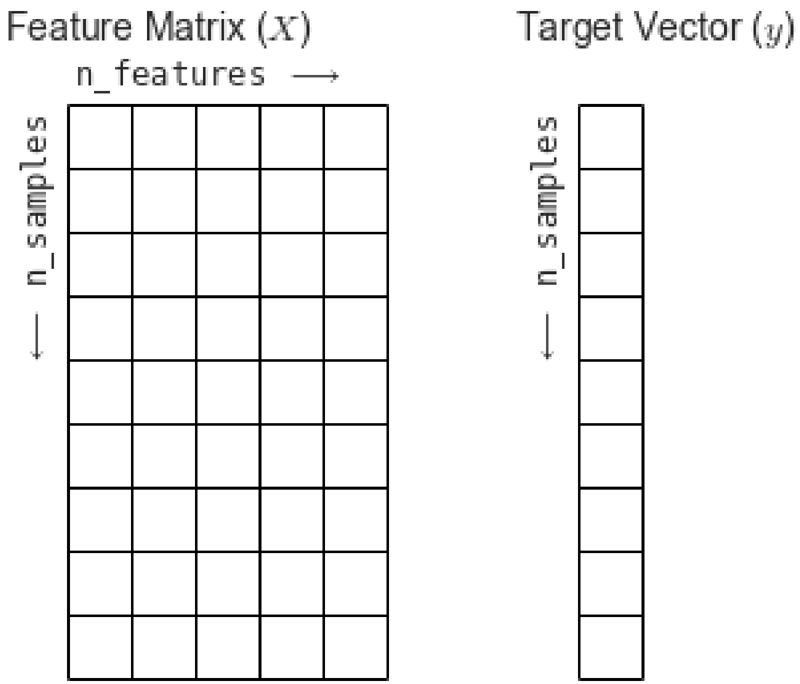
Binary Classification

- is the task of classifying the elements of a set into one of two groups (each called class).



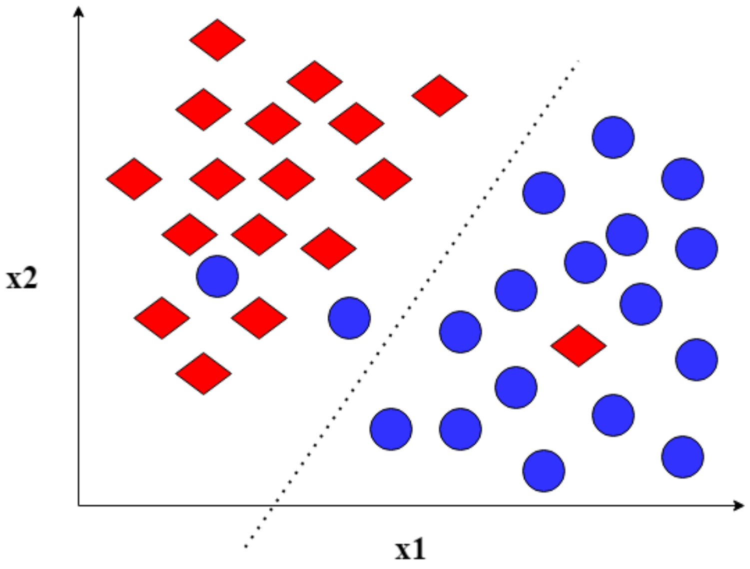
Binary Classification

- is the task of classifying the elements of a set into one of two groups (each called class).



Binary Classification - Propensities

- Linear and non-linear predictors output the probability of belonging to a particular class ($\hat{p}$)



Binary Cross-Entropy

- The most common **loss** function in binary classification

$$-(y\ \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$
1	0.775
0	0.116
0	0.039
0	0.070

How do we determine the **distance** between this two vectors?

Binary Cross-Entropy

- The most common **loss** function in binary classification

$$-(y\ \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$	
1	0.775	$-(1\ \log 0.775 + (1-1)\log(1-0.775)) = 0.2549$
0	0.116	
0	0.039	
0	0.070	

Binary Cross-Entropy

- The most common **loss** function in binary classification

$$-(y\ \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$
1	0.775
0	0.116
0	0.039
0	0.070

$$-(0\ \log 0.116 + (1-0)\log(1-0.116)) = 0.1233$$

Binary Cross-Entropy

- The most common **loss** function in binary classification

$$-(y \, \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$
1	0.775
0	0.116
0	0.039
0	0.070

$$-(0 \, \log 0.039 + (1-0)\log(1-0.039)) = 0.0398$$

Binary Cross-Entropy

- The most common **loss** function in binary classification

$$-(y\ \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$
1	0.775
0	0.116
0	0.039
0	0.070

$$-(0\ \log 0.070 + (1-0)\log(1-0.070)) = 0.0726$$

Binary Cross-Entropy

- The binary cross-entropy cost function is the average of the losses

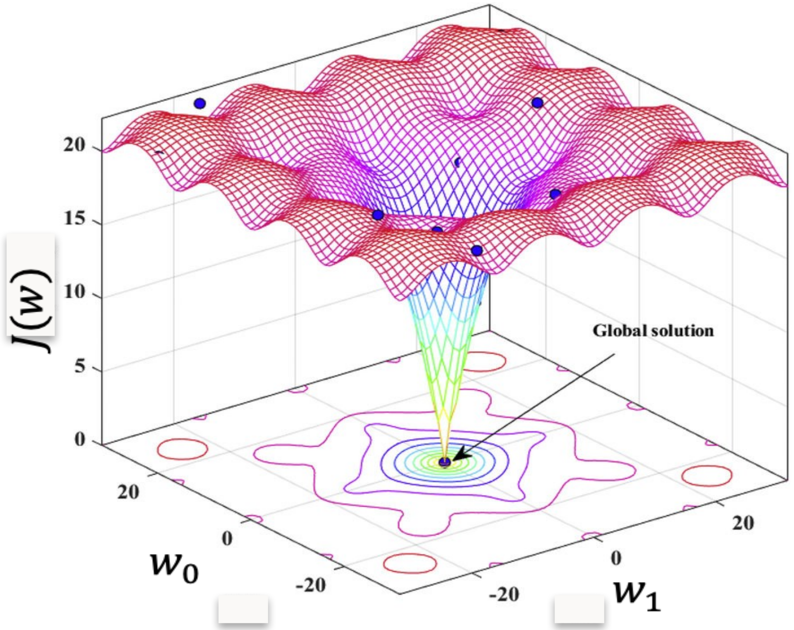
$$J(w) = -\frac{1}{n}\sum(y\ \log \hat{p} + (1-y)\log(1-\hat{p}))$$

y	$\hat{p}$	
1	0.775	0.2549 +
0	0.116	0.1233 +
0	0.039	0.0398 +
0	0.070	0.0726 = 0.4906/4 = 0.12265

Binary Cross-Entropy

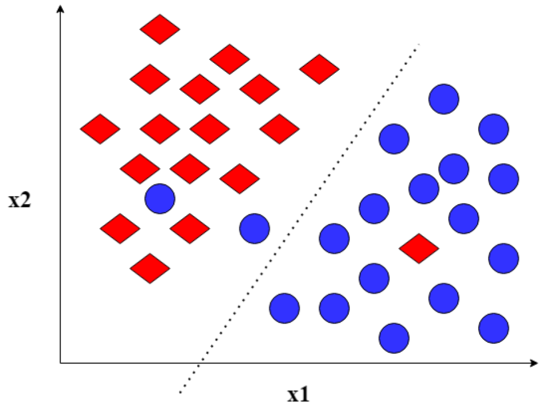
- The binary cross-entropy cost function is the average of the losses

$$J(w) = -\frac{1}{n}\sum(y\ \log\hat{p} + (1-y)\log(1-\hat{p}))$$



Example: **LOGISTIC REGRESSION**

$$\hat{p} = \frac{1}{1+\ e^{-(w_0+w_1x_1+\ \ldots)}} = \frac{1}{1+\ e^{Xw}}$$



Python

Metrics

- Used to determine if model is performing well

Scoring	Function	Comment
Classification		
'accuracy'	`metrics.accuracy_score`	
'balanced_accuracy'	`metrics.balanced_accuracy_score`	
'top_k_accuracy'	`metrics.top_k_accuracy_score`	
'average_precision'	`metrics.average_precision_score`	
'neg_brier_score'	`metrics.brier_score_loss`	
'f1'	`metrics.f1_score`	for binary targets
'f1_micro'	`metrics.f1_score`	micro-averaged
'f1_macro'	`metrics.f1_score`	macro-averaged
'f1_weighted'	`metrics.f1_score`	weighted average
'f1_samples'	`metrics.f1_score`	by multilabel sample
'neg_log_loss'	`metrics.log_loss`	requires `predict_proba` support
'precision' etc.	`metrics.precision_score`	suffixes apply as with 'f1'
'recall' etc.	`metrics.recall_score`	suffixes apply as with 'f1'
'jaccard' etc.	`metrics.jaccard_score`	suffixes apply as with 'f1'
'roc_auc'	`metrics.roc_auc_score`	
'roc_auc_ovr'	`metrics.roc_auc_score`	
'roc_auc_ovo'	`metrics.roc_auc_score`	
'roc_auc_ovr_weighted'	`metrics.roc_auc_score`	
'roc_auc_ovo_weighted'	`metrics.roc_auc_score`	

Metrics Review: Confusion Matrix

- The confusion matrix gives a summary of overall performance
- True Positives, True Negatives, False Positives and False Negatives

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

Metrics Review: Accuracy

- The accuracy metric measures number of obs predicted correctly
- It is one of the most important classification metrics

n=165	Predicted: NO	Predicted: YES	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

$$\text{accuracy} = \frac{n_{1,1} + n_{0,0}}{n}$$

Metrics Review: Recall (Sensitivity)

- The fraction of all 1's that are classified (predicted) as 1's.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

$$\text{sensitivity} = \frac{n_{1,1}}{n_{1,1} + n_{1,0}}$$

Metrics Review: Precision

- The fraction of predicted values that are correct.

n=165	Predicted: NO	Predicted: YES	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

$$\text{precision} = \frac{n_{1,1}}{n_{1,1} + n_{0,1}}$$

F1 Score

- It is meant to capture both Recall and Precision

$$\text{F1 score} = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

n=165	Predicted: NO	Predicted: YES	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

Balanced Accuracy

- We use the balanced accuracy for imbalanced data

$$\text{balanced accuracy} = \frac{1}{2}\left(\frac{n_{0,0}}{n_{0,0}+n_{0,1}} + \frac{n_{1,1}}{n_{1,1}+n_{1,0}}\right)$$

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	$n_{0,0}$ TN = 50	$n_{0,1}$ FP = 10	60
Actual: YES	$n_{1,0}$ FN = 5	$n_{1,1}$ TP = 100	105
	55	110	

Python

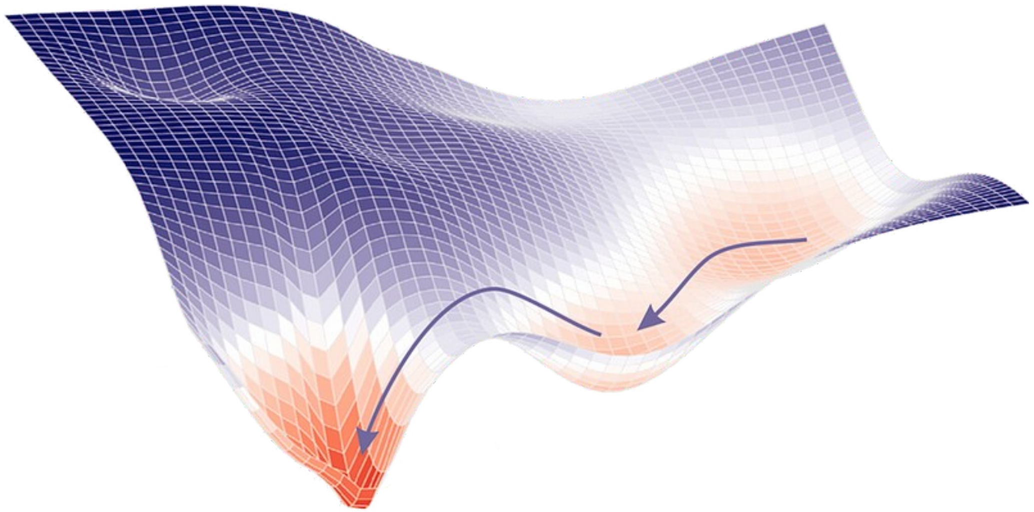
Common Cost Functions Neural Nets

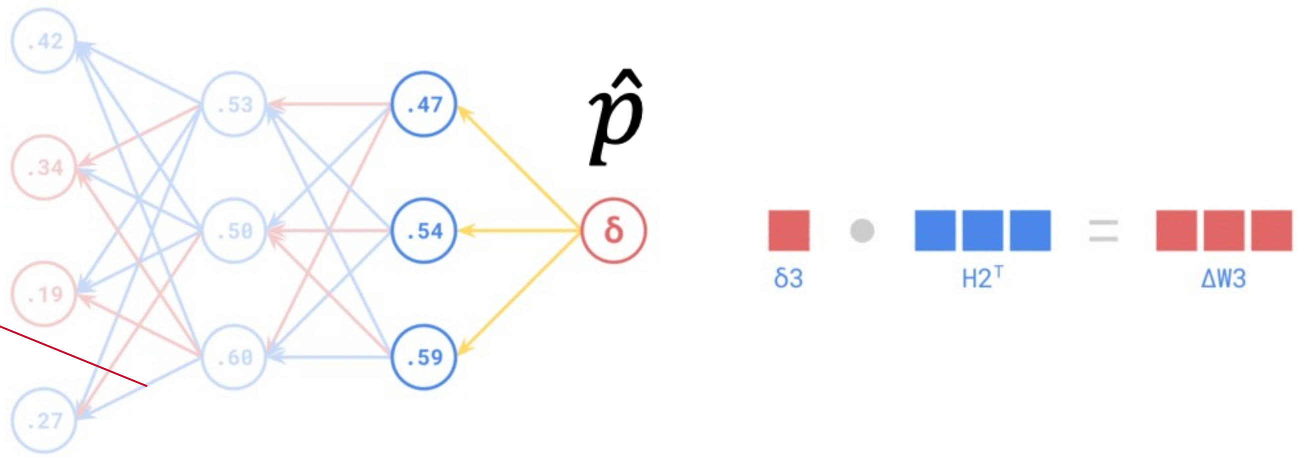
- Binary Cross-Entropy
  - Loss Function

$$J(w) = -\frac{1}{n}\sum(y \log \hat{p} + (1-y)\log(1-\hat{p}))$$

  - Gradient

$$\nabla J(w) = \hat{p} - y$$

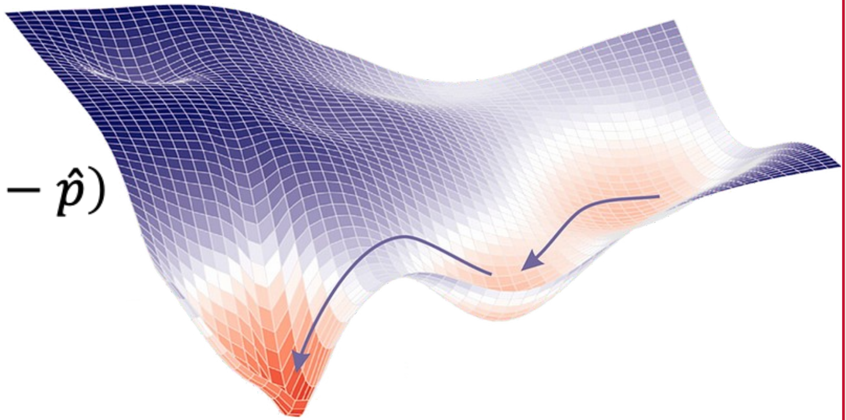




Common Cost Functions Neural Nets

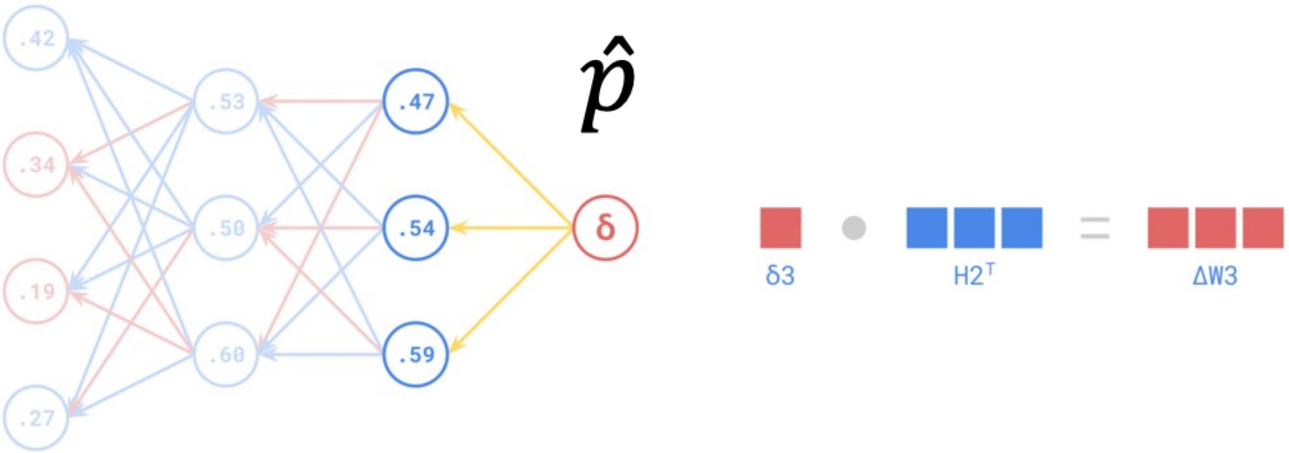
- Binary Focal Cross-Entropy
  - Loss Function

$$J(w) = -\frac{1}{n}\sum \alpha(1-\hat{p})^{\gamma} y \log(\hat{p}) + (1-\alpha)\hat{p}^{\gamma}(1-y)\log(1-\hat{p})$$



  - Gradient

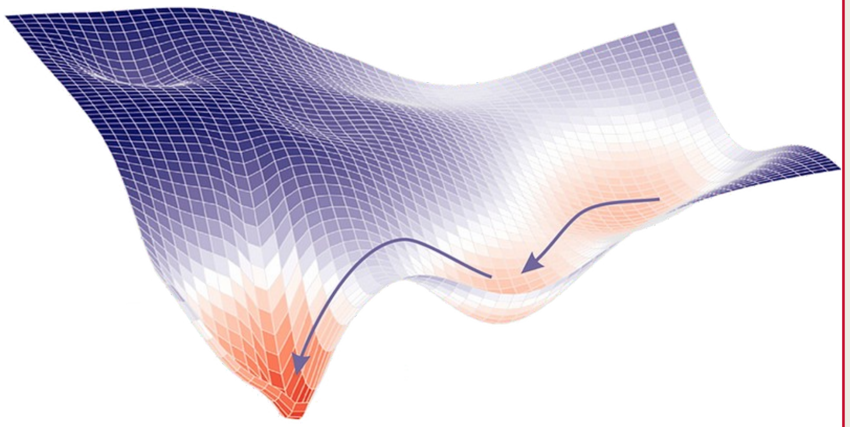
$$\nabla J(w) = (\gamma(1-y-\hat{p})\log(1-\hat{p}) + (1-y-\hat{p})\hat{p}(1-\hat{p})$$

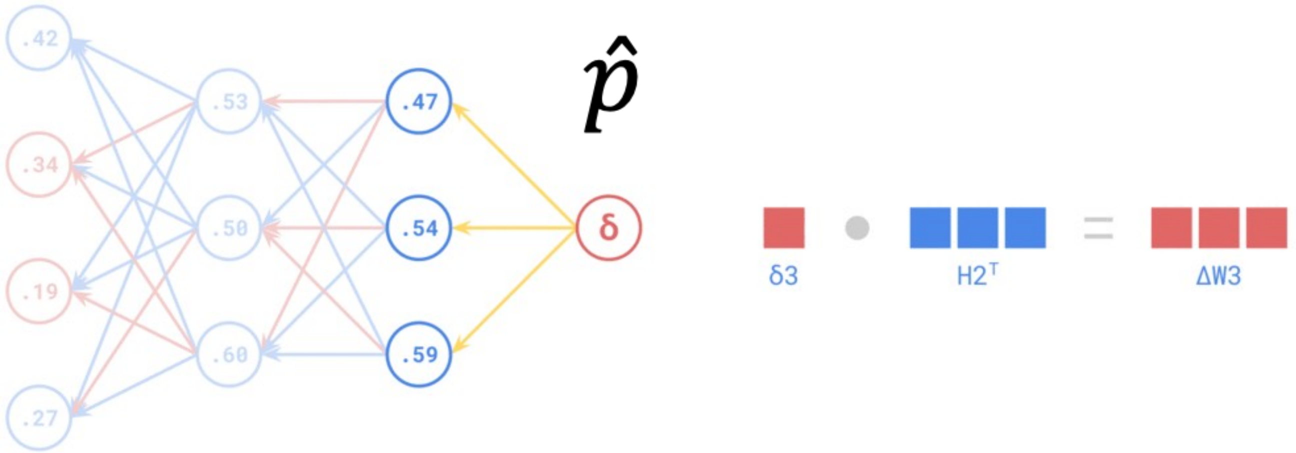


Activation Functions Neural Nets

- Common Activations on Layers
  - ReLu
  - Tanh
  - Sigmoid

- Output Layer Activation
  - Sigmoid
  - Tanh





Python