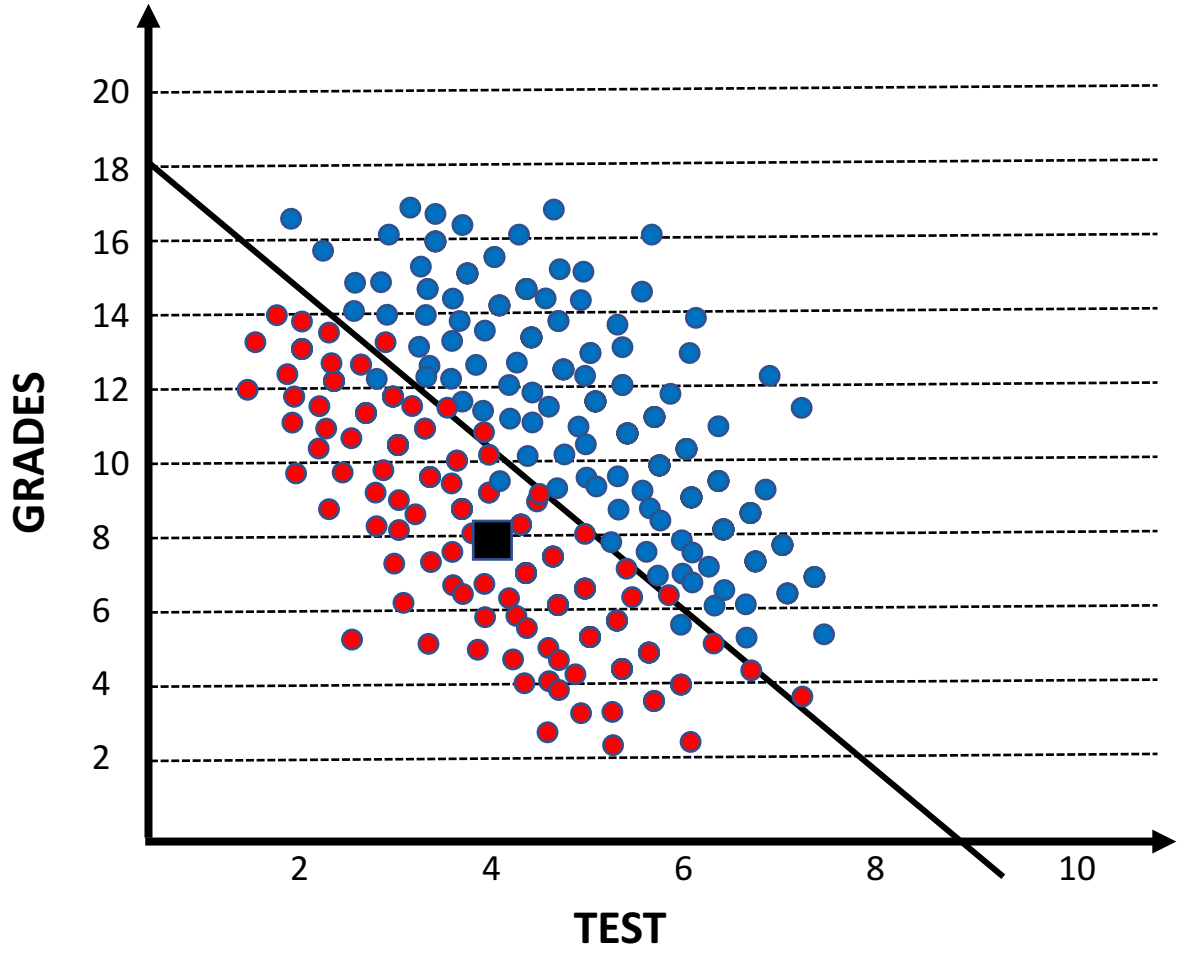
Perceptron

Linear Boundary / Classification



BOUNDARY: A LINE (WX + B)

$2X_1 + X_2 - 18 = 0$

X_1 = Test

X_2 = Grade

2*Test + Grade – 18 = Score

PREDICTION:

Score >= 0 Accept

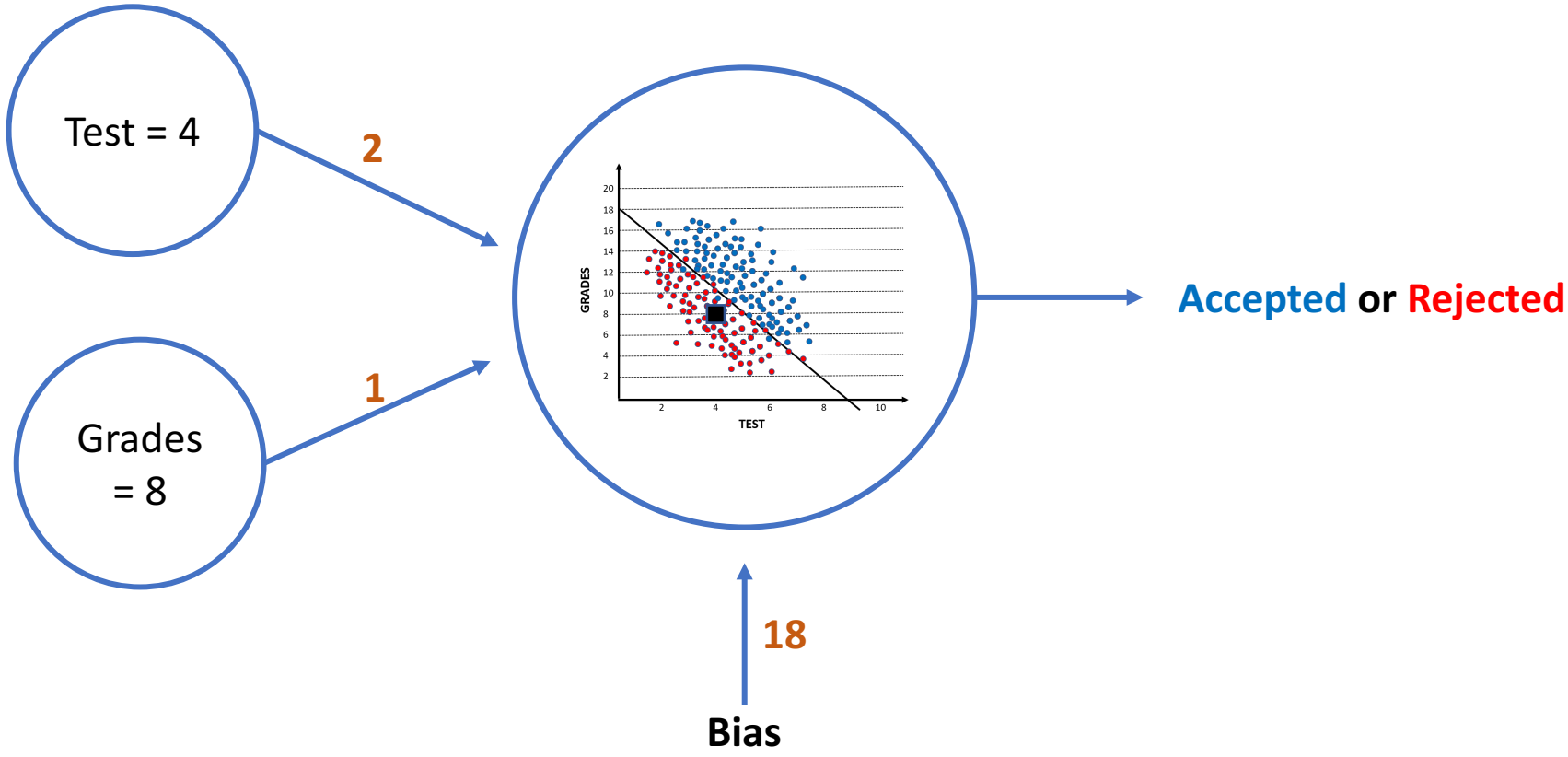
Score < 0 Reject

Example: What is prediction of a student who got 4 on the test and 8 on the grades?

Answer: 2 x 4 + 8 – 18 = 16 - 18 = -2

Perceptron

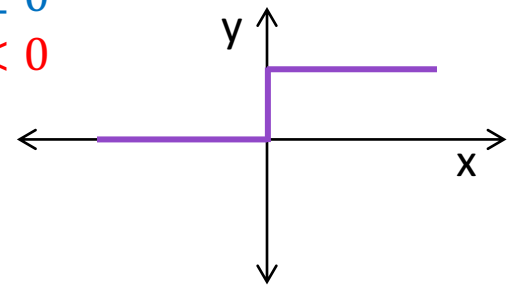
$$2X_1 + X_2 - 18 = 0$$

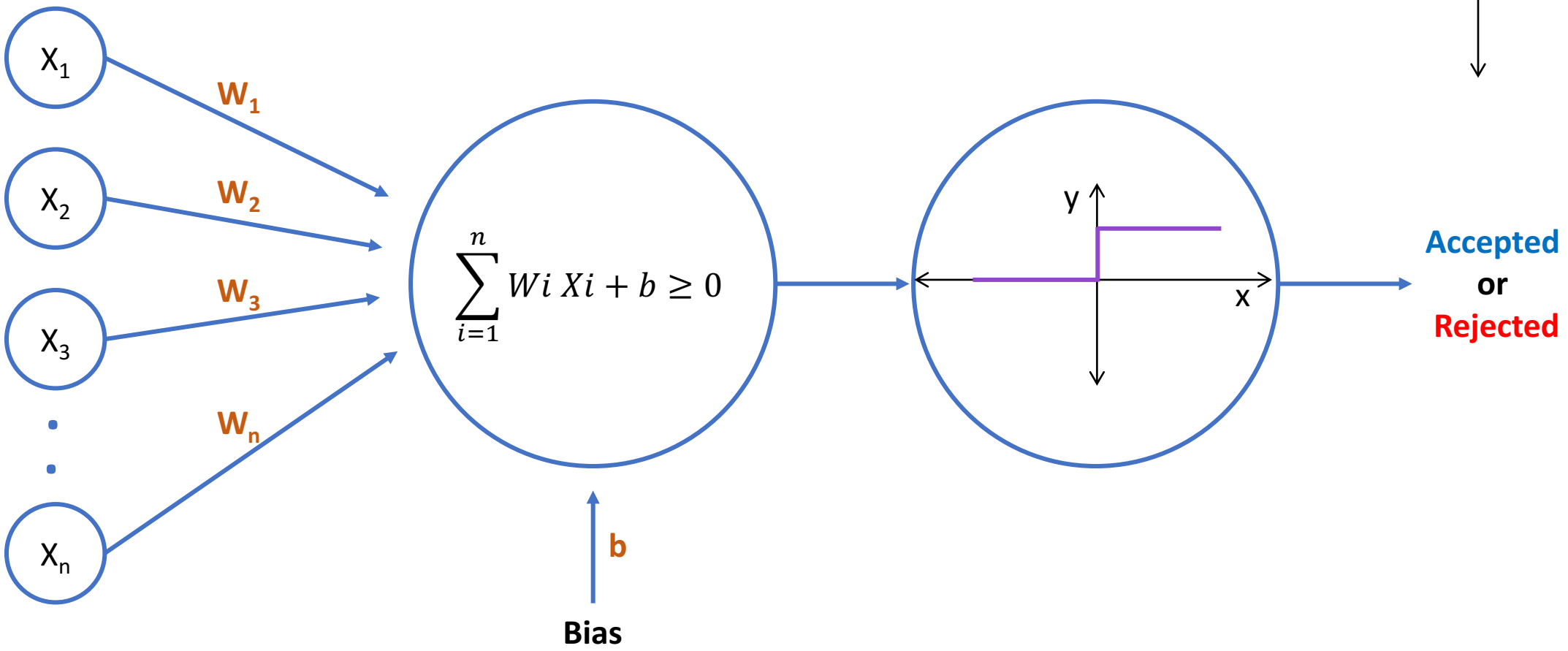


Perceptron

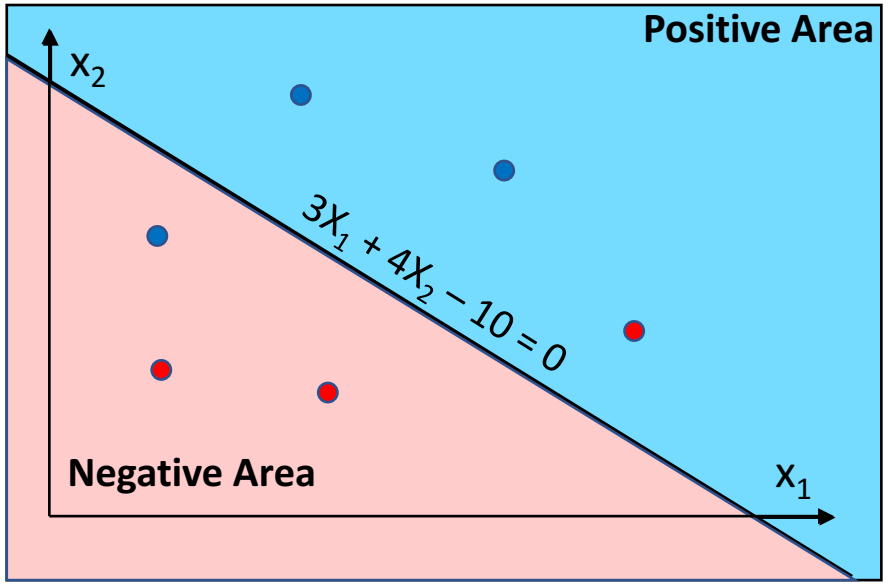
Step Function

$$Y = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$





How to select the weights and bias – called TRAINING the Perceptron



$3X_1 + 4X_2 - 10 = 0$
Misclassified Point: (8,3)

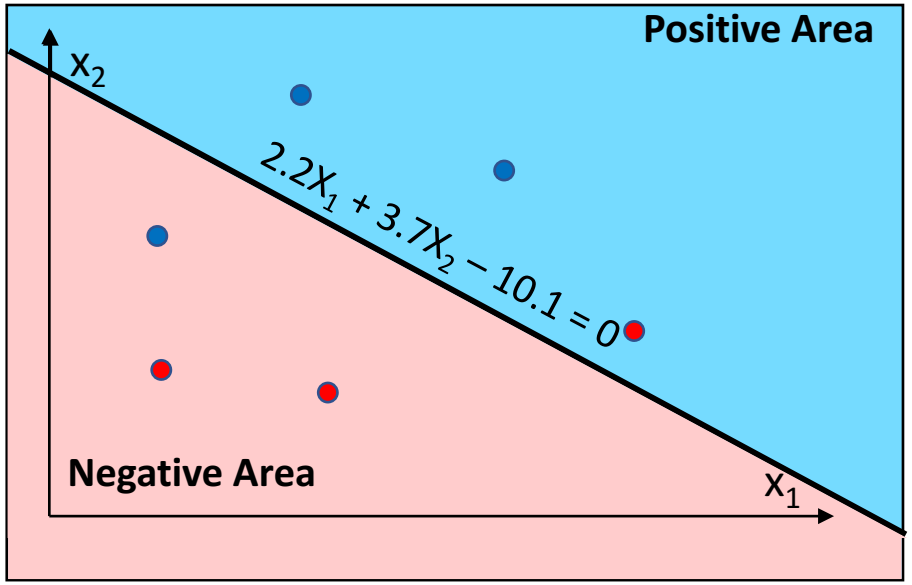
-	3	4	-10
	8	3	1
	-5	1	-11

$3X_1 + 4X_2 - 10 = 0$
Misclassified Point: (8,3)
Learning Rate = 0.1

-	3	4	-10
	8x0.1	3x0.1	1x0.1
	2.2	3.7	-10.1

New Line: $2.2X_1 + 3.7X_2 - 10.1 = 0$

How to select the weights and bias – called TRAINING the Perceptron



$3X_1 + 4X_2 - 10 = 0$
Misclassified Point: (8,3)

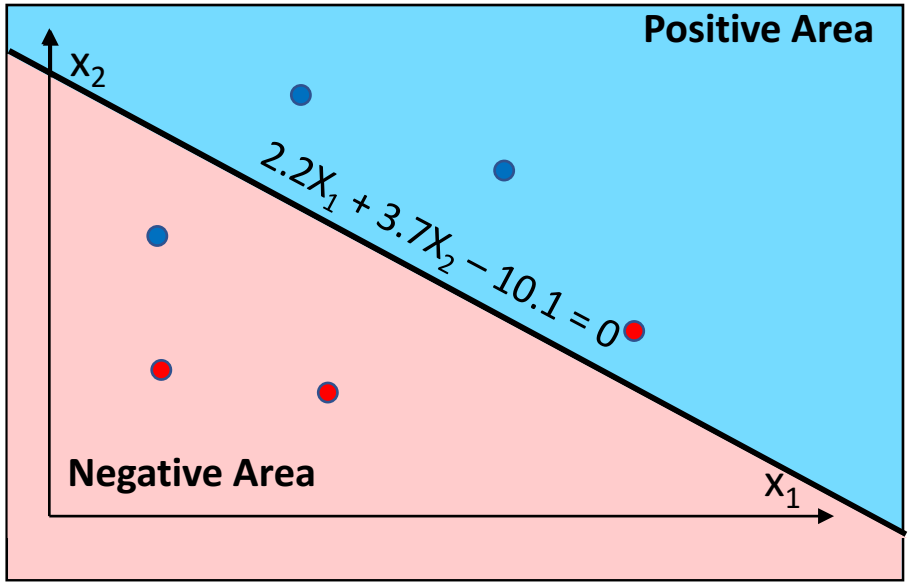
-	3	4	-10
	8	3	1
	-5	1	-11

$3X_1 + 4X_2 - 10 = 0$
Misclassified Point: (8,3)
Learning Rate = 0.1

-	3	4	-10
	8x0.1	3x0.1	1x0.1
	2.2	3.7	-10.1

New Line: $2.2X_1 + 3.7X_2 - 10.1 = 0$

How to select the weights and bias – called TRAINING the Perceptron

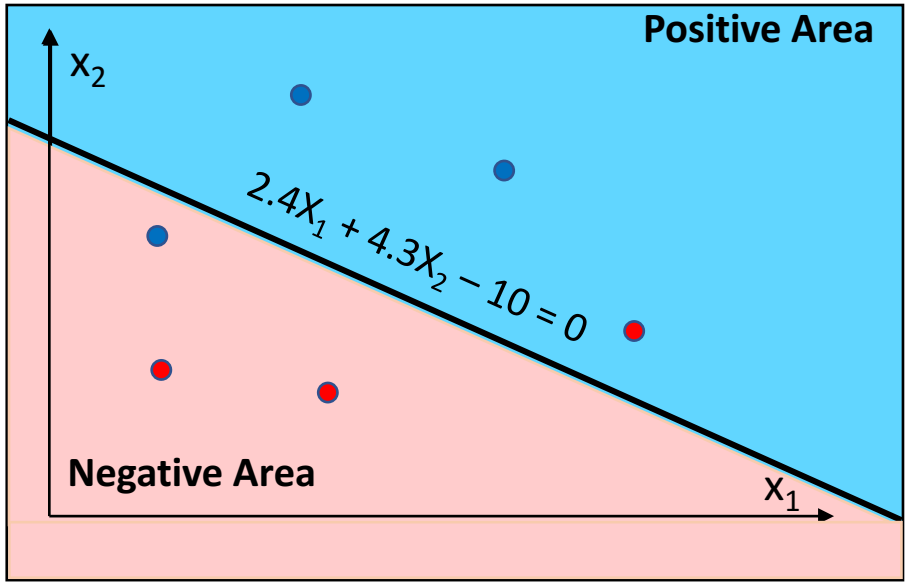


$2.2X_1 + 3.7X_2 - 10.1 = 0$
Misclassified Point: (2,6)
Learning Rate = 0.1

	2.2	3.7	-10.1
+	2x0.1	6x0.1	1x0.1
	2.4	4.3	-10

New Line:
$2.4X_1 + 4.3X_2 - 10 = 0$

How to select the weights and bias – called TRAINING the Perceptron

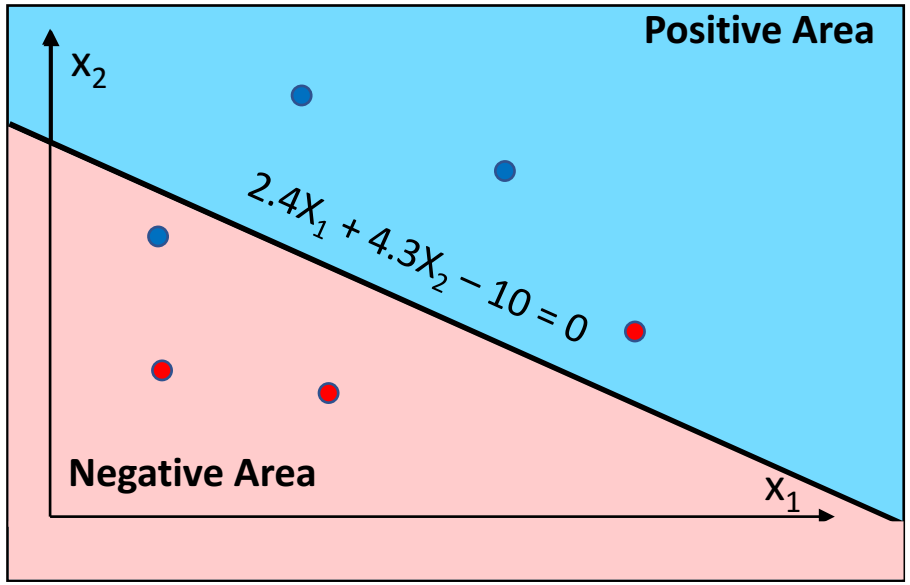


$2.2X_1 + 3.7X_2 - 10.1 = 0$
Misclassified Point: (2,6)
Learning Rate = 0.1

	2.2	3.7	-10.1
+	2x0.1	6x0.1	1x0.1
	2.4	4.3	-10

New Line:
$2.4X_1 + 4.3X_2 - 10 = 0$

Pseudocode for Training – Adjusting the Weights & Bias



1. Start with random weights: (W_1, … , W_n, b)

2. For every misclassified point (X_1, … ,X_n):
   - If **Prediction = 0**
     - For I = 1 … n
       - $W_i = W_i + \alpha X_i$
     - $b = b + \alpha$
   - If **Prediction = 1**
     - For I = 1 … n
       - $W_i = W_i - \alpha X_i$
     - $b = b - \alpha$

Perceptron – Basic Functions

- Step Function $Y' = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$
- Predict Function $\acute{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$
- Error Function $Error = y - \acute{y}$
- Update Weights Function $b \leftarrow b + \alpha \, . x . Error$

$$w_i \leftarrow w_i + \alpha . x_i \, . Error$$

Run Code

- **perceptron_1.py**

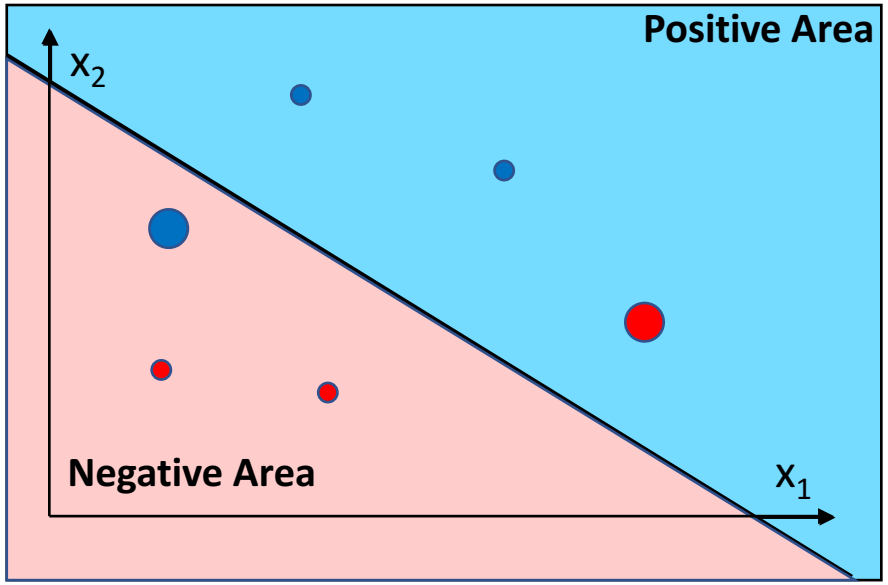
- How can we initialize **weights** and **bias**?

- How **learning rate** effects on model training?

Run Code

- **perceptron_2.py**
- What is **epoch**?
- What does epochs=100 do?
- Is our model better when we have high epoch?

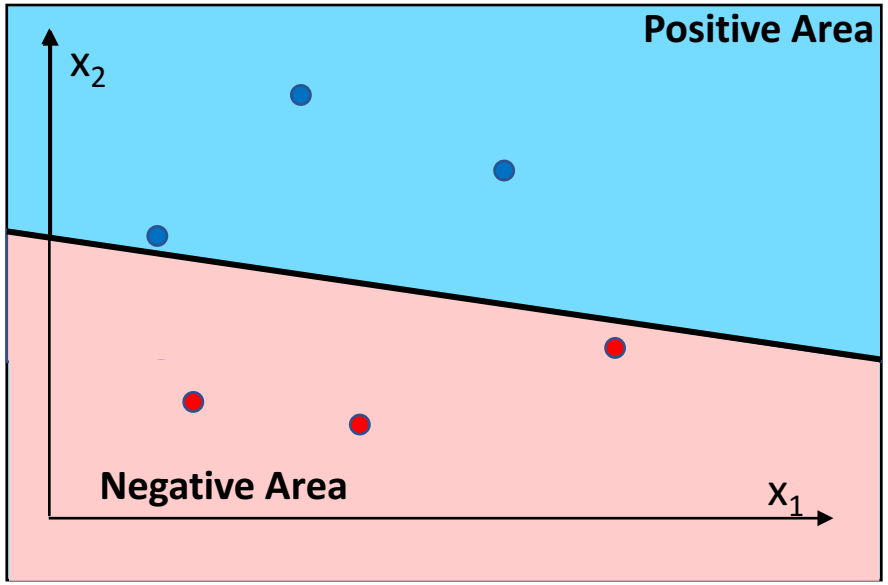
Error Function & Log-Loss Function



Error = + + + + +

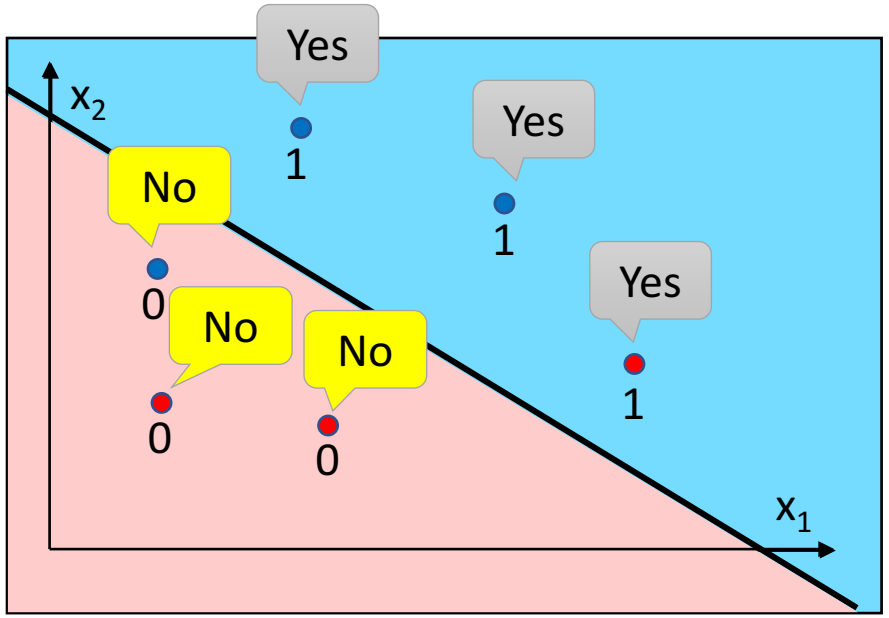
- What happened if error function is **discrete**?
- Error function should be **continues** and **differentiable**

Error Function & Log-Loss Function

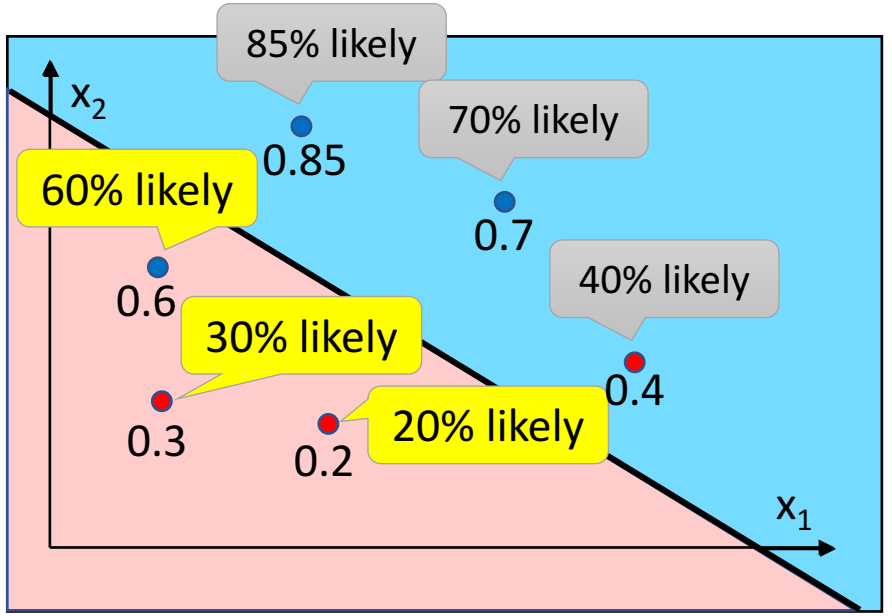


Error = ● + ● + ● + ● + ● + ●

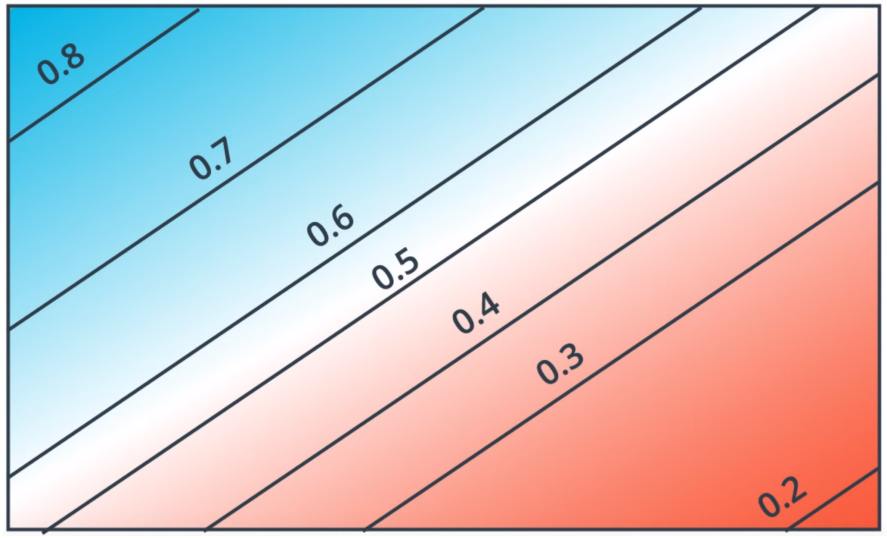
Predictions



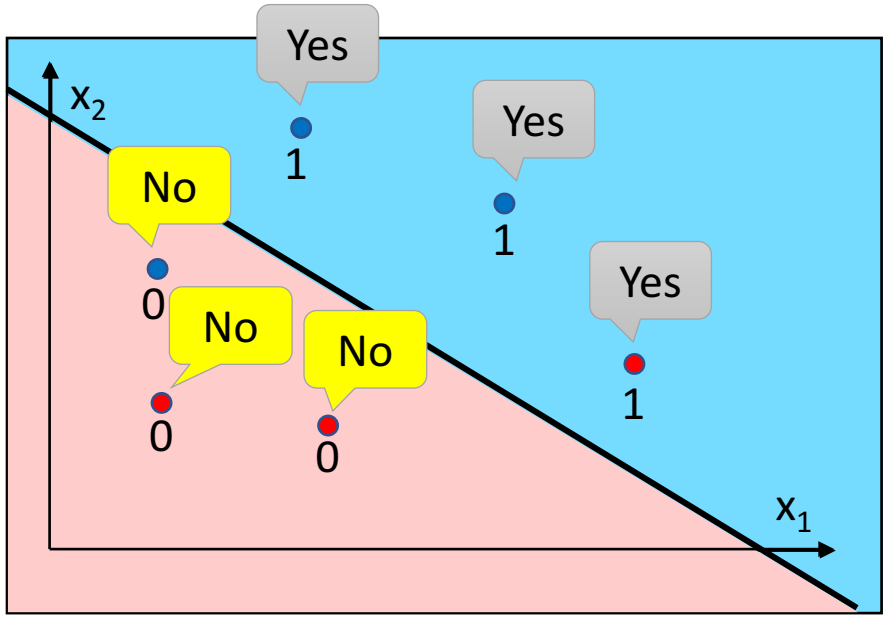
Discrete



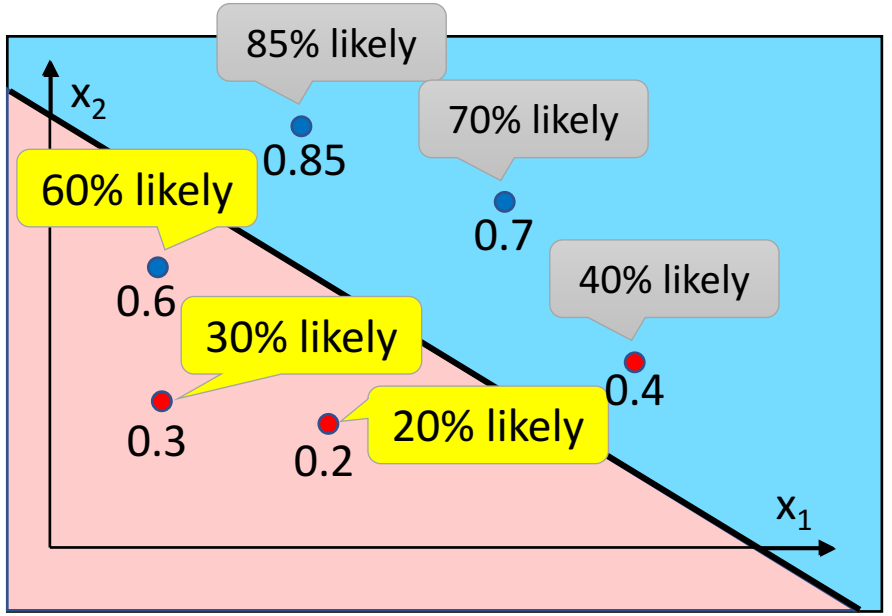
Continuous



Predictions & Activation Functions

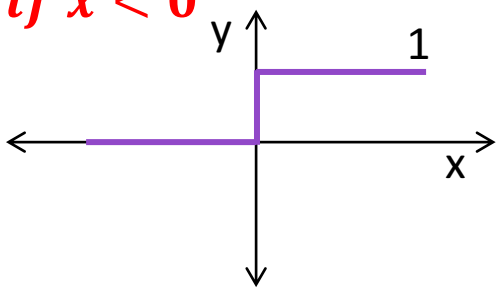


Discrete

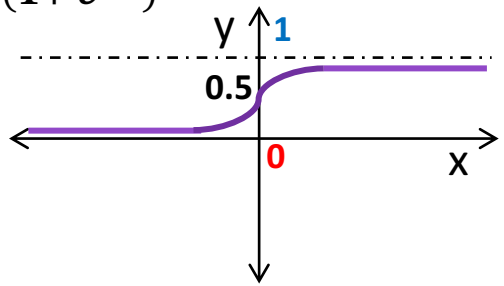


Continuous

Step Function Y = $\begin{cases} 1 \; if \; x \geq 0 \\ 1 \; if \; x < 0 \end{cases}$



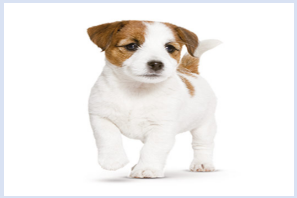
Sigmoid Function $Y = \frac{1}{(1+e^{-x})}$



Predictions & Activation Functions

Binary Classification

Sigmoid Function $Y = \frac{1}{(1 + e^{-x})}$

Animal	Value
	1
	0

Multi-Class Classification

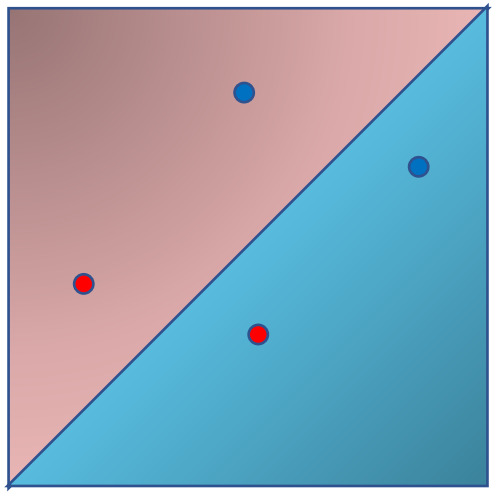
Softmax Function = $\frac{e^{Z_i}}{e^{Z_1} + e^{Z_2} + \ldots + e^{Z_n}}$

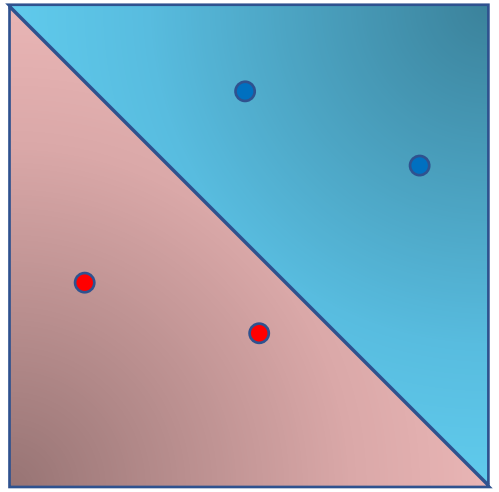
One-Hot Encoding

Animal	Dog	Cat	Duck
	1	0	0
	0	1	0
	0	0	1

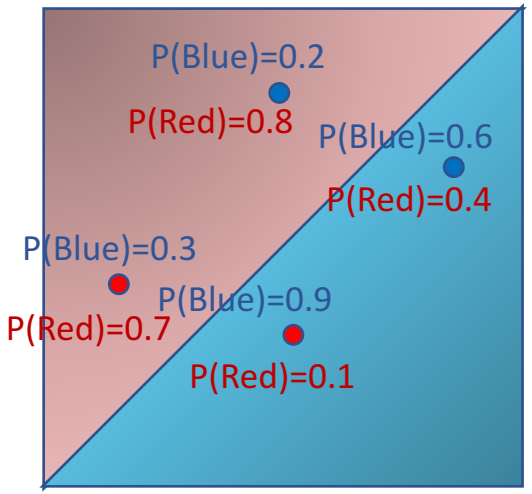
Maximum Likelihood

Which model is better?





Maximum Likelihood

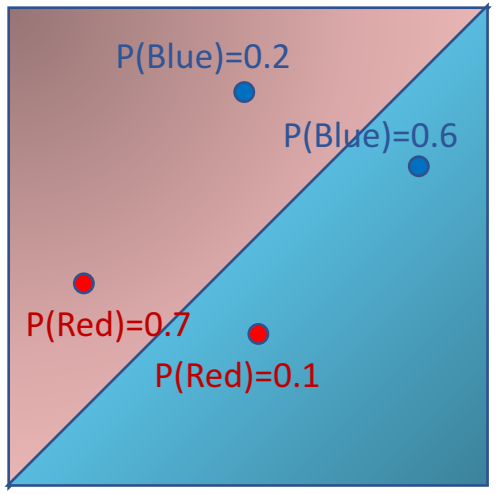


$$\hat{Y} = \sigma(Wx + b)$$

P (Blue) = $\sigma(Wx + b)$

P (Red) = 1 - P (Blue)

Maximum Likelihood



x

P(Blue)=0.2

P(Blue)=0.6

P(Red)=0.7

P(Red)=0.1

P(all)=0.0084

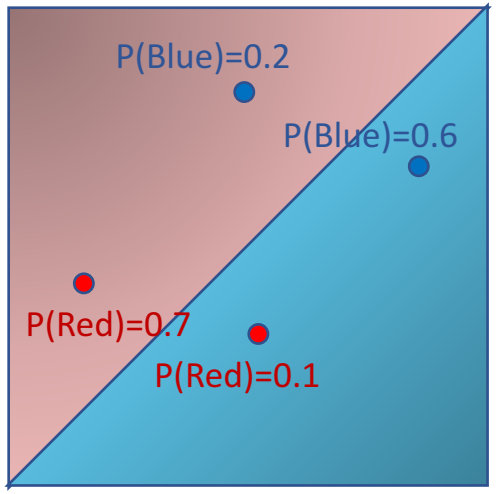
$$\hat{Y} = \sigma(Wx + b)$$

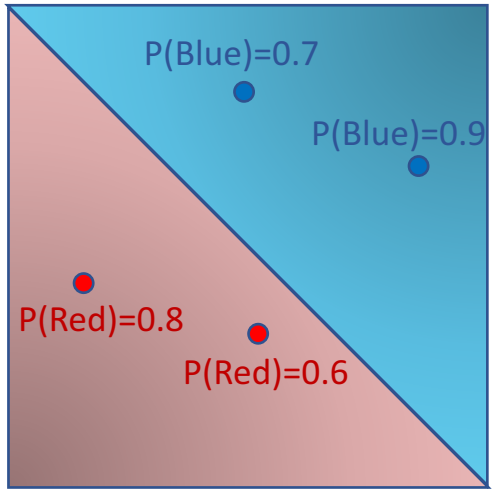
P (Blue) = $\sigma(Wx + b)$

P (Red) = 1 - P (Blue)

Maximum Likelihood

Goal: Maximize the Probability





0.6 x 0.2 x 0.1 x 0.7 = 0.0084

0.7 x 0.9 x 0.8 x 0.6 = 0.3024

ln(0.6) + ln(0.2) + ln(0.1) + ln(0.7)

ln(0.7) + ln(0.9) + ln(0.8) + ln(0.6)

(-0.51) + (-1.61) + (-2.3) + (-0.36) = -4.78

(-0.36) + (-0.1) + (-0.22) + (-0.51) = -1.19

- ln(0.6) - ln(0.2) - ln(0.1) - ln(0.7)

- ln(0.2) - ln(0.6) - ln(0.7) - ln(0.1)

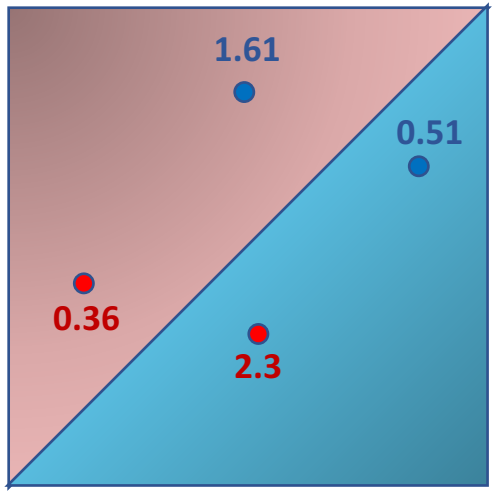
0.51 + 1.61 x+ 2.3 + 0.36 = 4.78

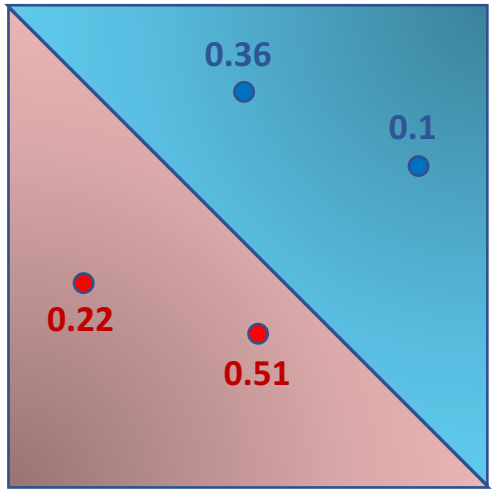
0.36 + 0.1 x+ 0.22 + 0.51 = 1.19

Cross Entropy

Maximum Likelihood

Goal: Maximize the Probability by minimize the cross entropy





0.6 x 0.2 x 0.1 x 0.7 = 0.0084

ln(0.6) + ln(0.2) + ln(0.1) + ln(0.7)

(-0.51) + (-1.61) + (-2.3) + (-0.36) = -4.78

- ln(0.6) - ln(0.2) - ln(0.1) - ln(0.7)

0.51 + 1.61 x+ 2.3 + 0.36 = 4.78

0.7 x 0.9 x 0.8 x 0.6 = 0.3024

ln(0.7) + ln(0.9) + ln(0.8) + ln(0.6)

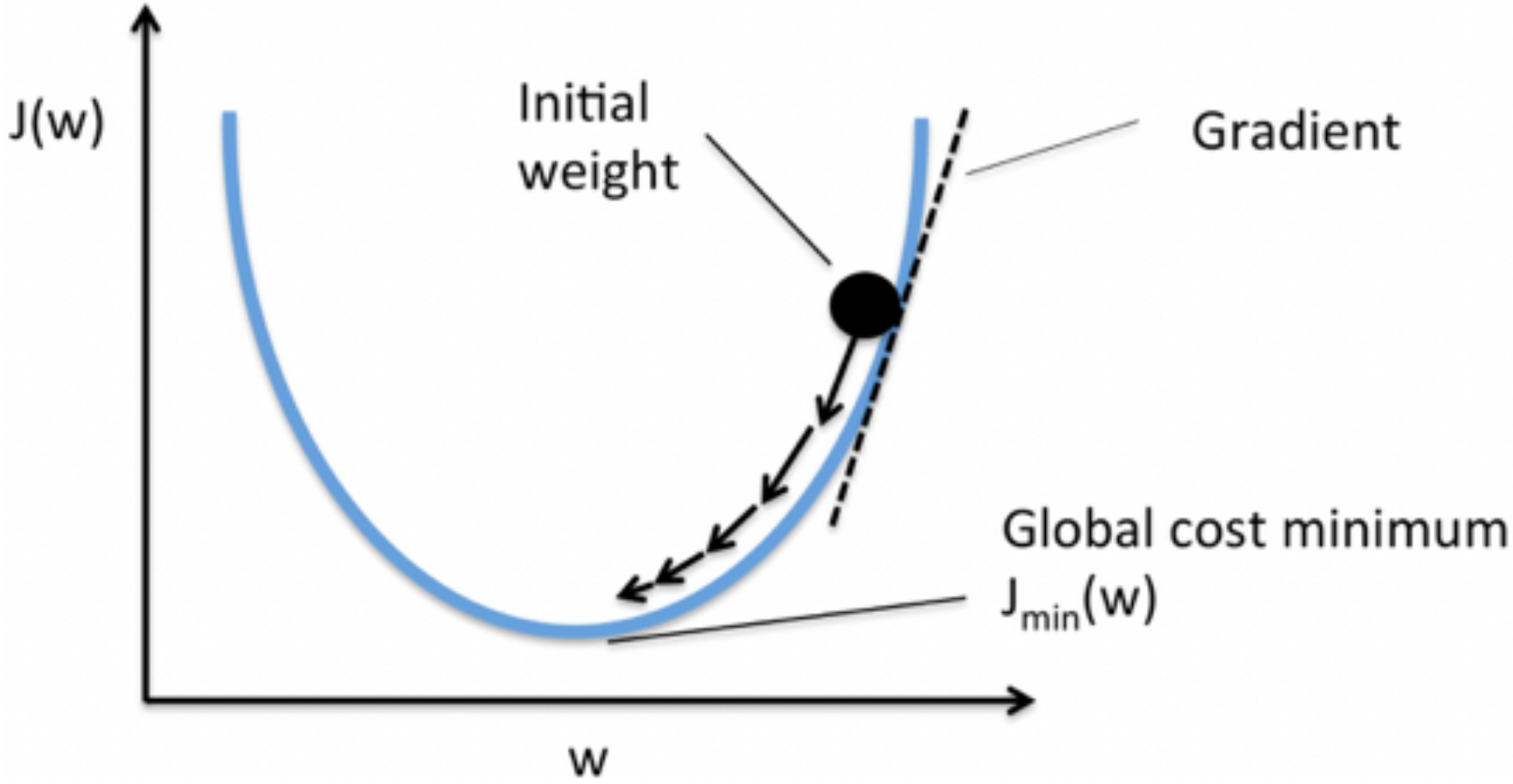
(-0.36) + (-0.1) + (-0.22) + (-0.51) = -1.19

- ln(0.2) - ln(0.6) - ln(0.7) - ln(0.1)

0.36 + 0.1 x+ 0.22 + 0.51 = 1.19

Gradient Decent

What are we optimizing?



Gradient Descent – Basic Functions

- Sigmoid Function $\sigma(x) = \frac{1}{1+ e^{-x}}$
- Predict Function $\acute{y} = \sigma(w_1 x_1 + w_2 x_2 + b)$
- Error Function (Cross Entropy) $Error(y, \acute{y})$ = -y log($\acute{y}$) – (1-y) log(1-$\acute{y}$)
- Update Weights Function $b \leftarrow b + \alpha \,.\, Error(y, \acute{y})$

  $w_i \leftarrow w_i + \alpha . x_i \,.\, Error(y, \acute{y})$

Run Code

- **gradient_descent.py**

- How gradient descent different from perceptron?

Run Code

- **perceptron_keras_scikit_learn.py**

- How can we evaluate the model with cross validation?

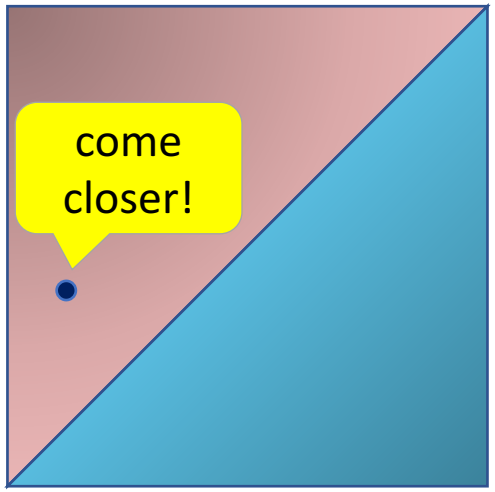
Perceptron vs Gradient Descent

Perceptron

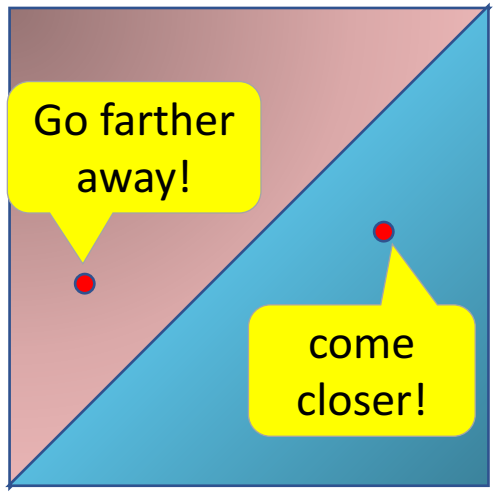
- Use step activation function
- $\acute{y}$ value is only 0 or 1
- If x_i is misclassified then only change its weight w_i

Gradient Descent

- Use sigmoid activation function
- $\acute{y}$ can be any value between 0 and 1
- Change all weights as $b \leftarrow b + \alpha . Error(y, \acute{y})$

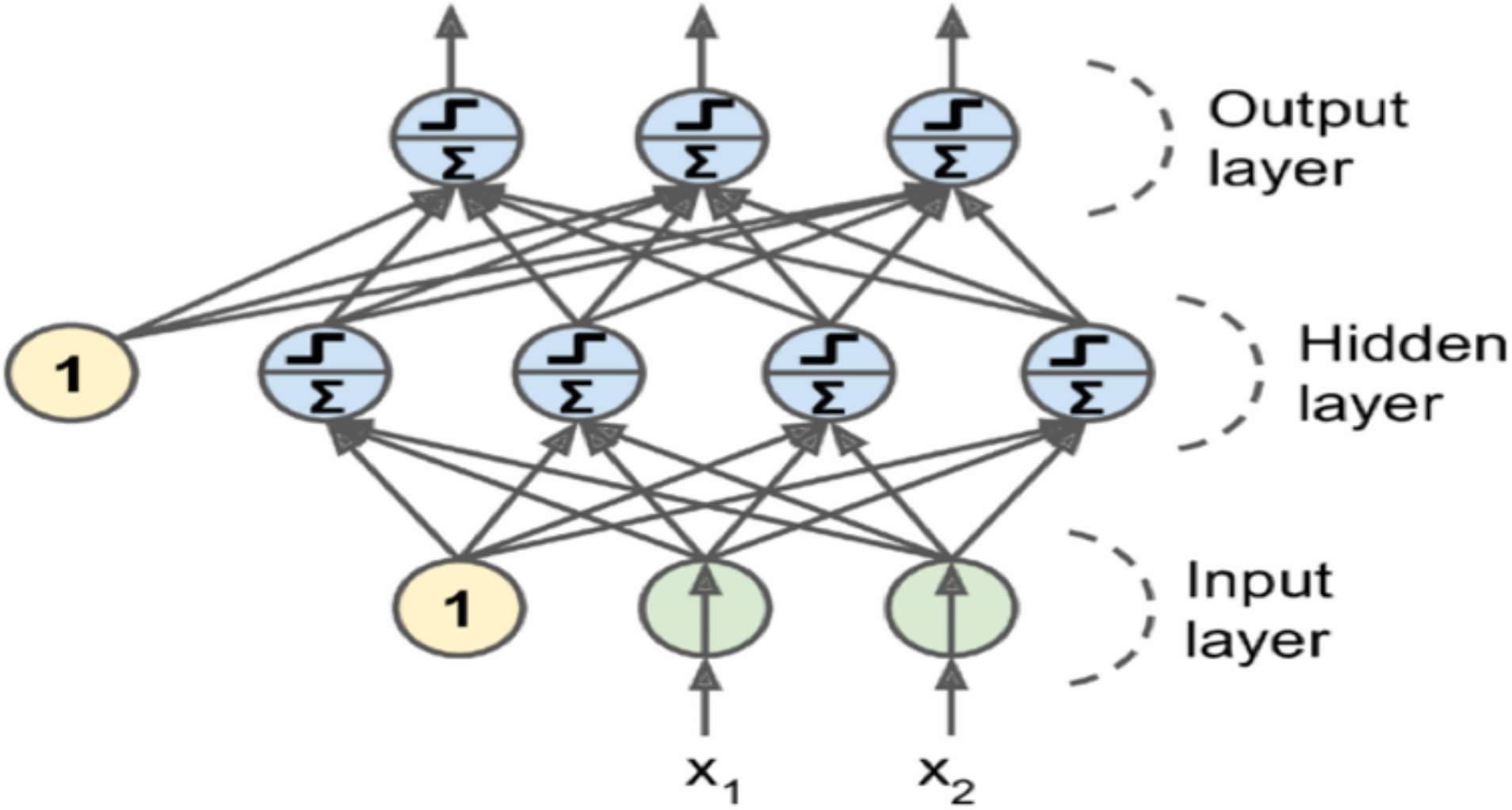


Incorrectly classified



Correctly classified

Two Layers of Perceptron



Cross Entropy

Binary Classification

- Cross-Entropy = $-\sum_{i=1}^{m} y_i \ \ln P_i + (1 - y_i) \ln(1 - P_i)$
- Keras has "**binary_crossentropy**" function

Multiclass Classification

- Cross-Entropy = $-\sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \ \ln P_{ij}$
- Keras has "**categorial_crossentropy**" **function.**

Run Code

- **gradient_descent_2.py**
- What is the data type of X_train?
- What does **Flatten** do?
- What does activation="**softmax**" do?
- Why use a loss function="**category_crossentropy**" at all?
- What does optimizer="**adam**" do?
- How can we get model performance (metrics)?
- How can we save best model (**ModelCheckPoint**)?

Run Code

- **gradient_descent_load.py**

- How can we load saved model?

Run Code

- **model_saving.py**
- How to save a model as json file?
- How to save a model as yaml file?

Run Code

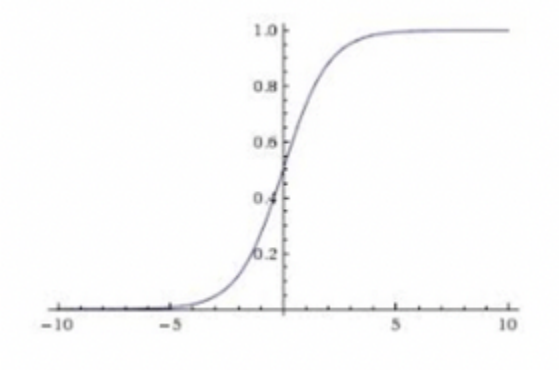
- **gradient_descent_predict.py**

- How can we make inference on unseen data?

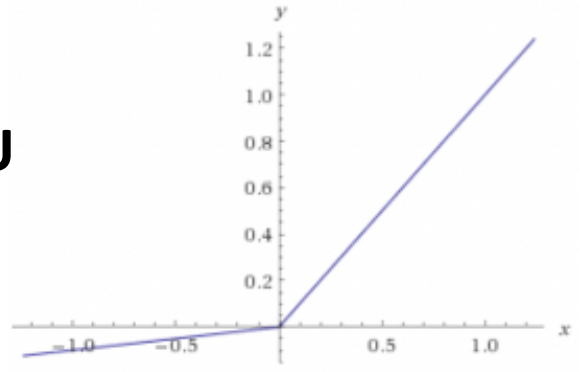
Multi Layers Networks

Activation Functions

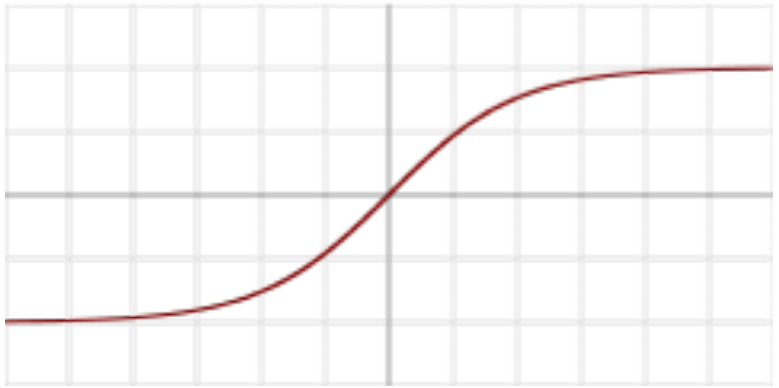
Sigmoid



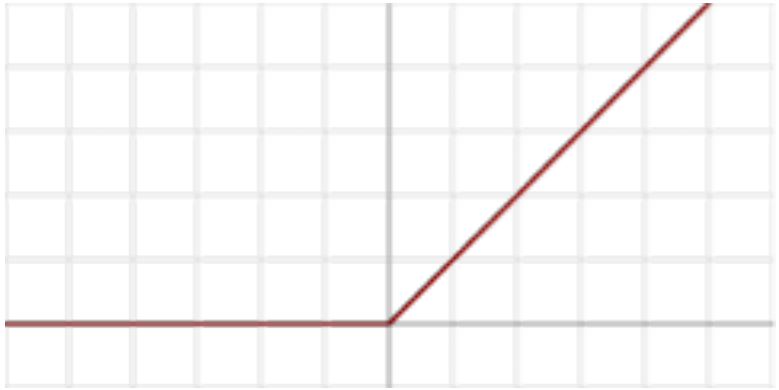
Leaky ReLU



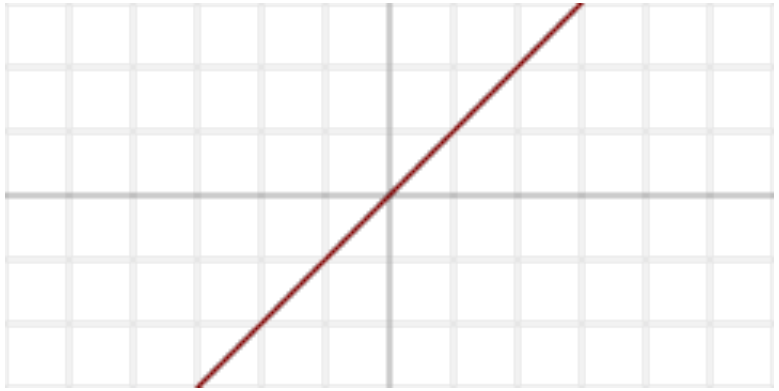
Tanh



ReLU



Linear



Loss Functions

- Binary Cross Entropy
- Categorical Cross Entropy
- Sparse Categorical Cross Entropy
- Mean Squared Error – MSE
- Mean Absolute Error – MAE
- Mean Absolute Percentage Error – MAPE
- Mean Squared Logarithmic Error – MSLE

Optimization Functions

- SGD – Stochastic Gradient Descent optimizer
  - Learning rate
  - Momentum
  - Decay
  - Nesterov momentum
- RMSProp
- Adagrad
- Adadelta
- Adam
- Adamax
- Nadam
- TFOptimize

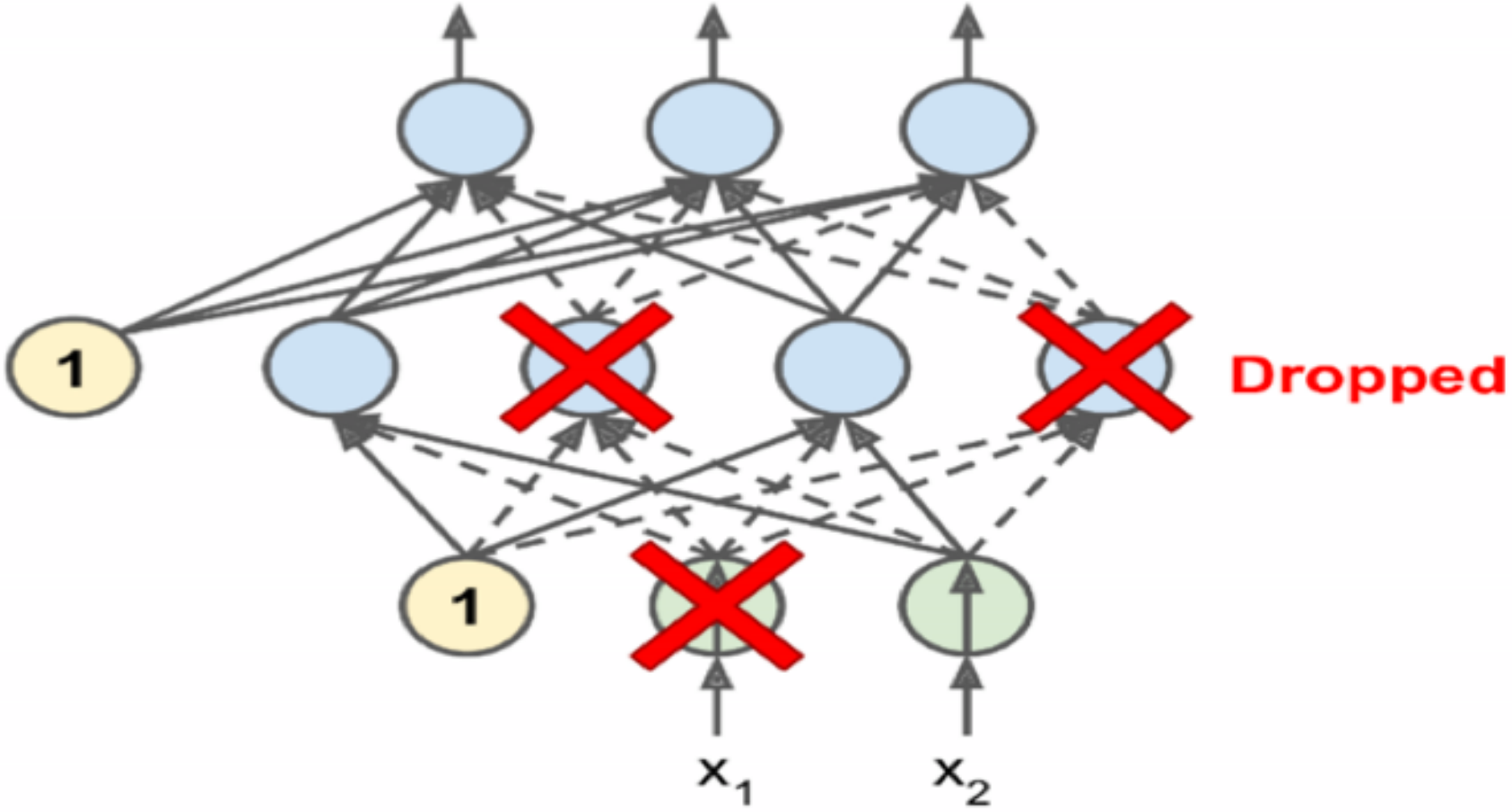
Metrics

- Accuracy
- MAE – Mean Absolute Error

Run Code

- **gradient_descent_multilayers.py**
- How did this change run-time and accuracy?
- Why do we use activation="**relu**" in one layer?
- How could we make a deeper model?

Dropout

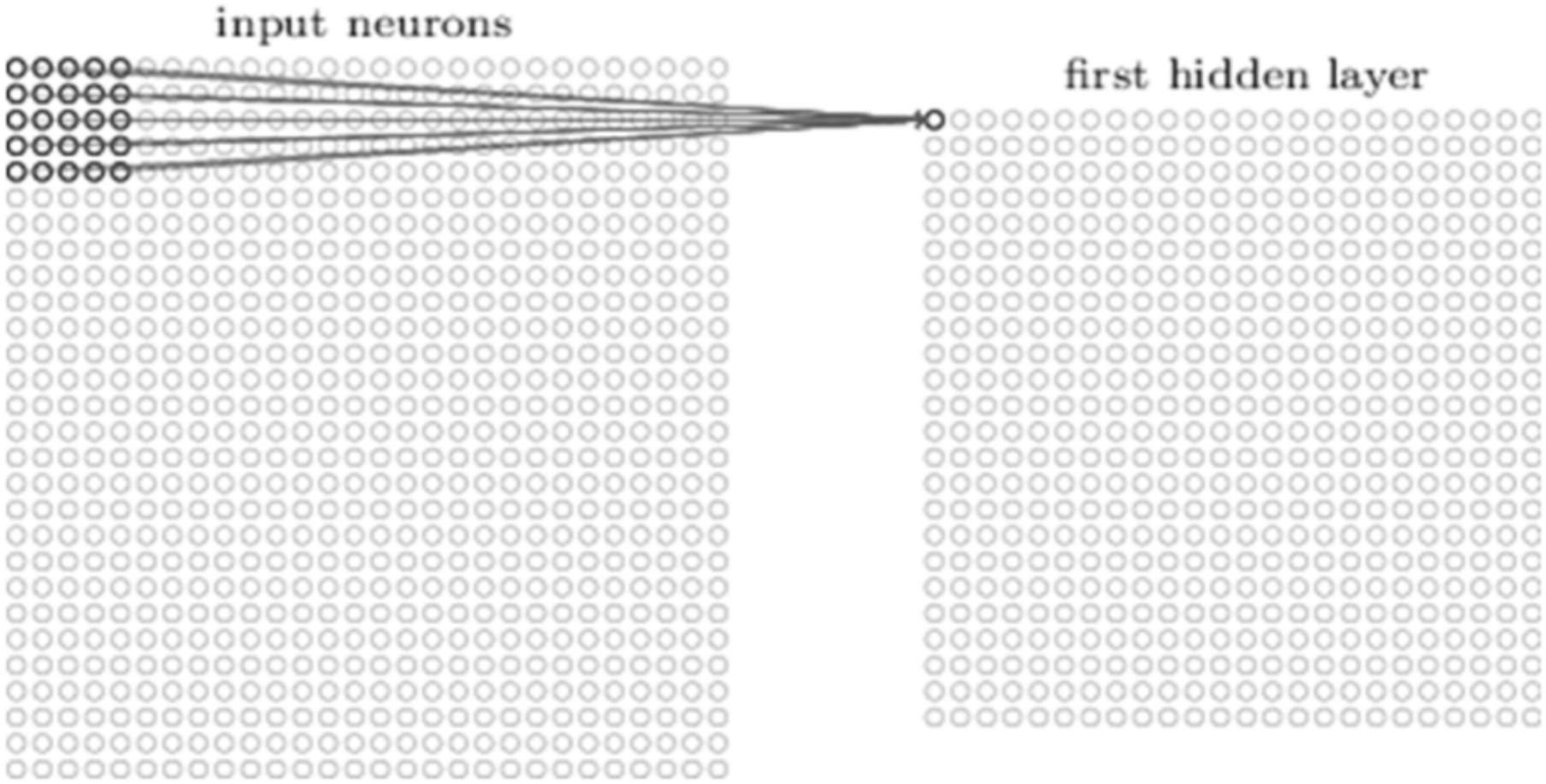


Run Code

- **gradient_descent_dropout.py**
- How does this change training accuracy and validation accuracy?
- When would we consider using dropout

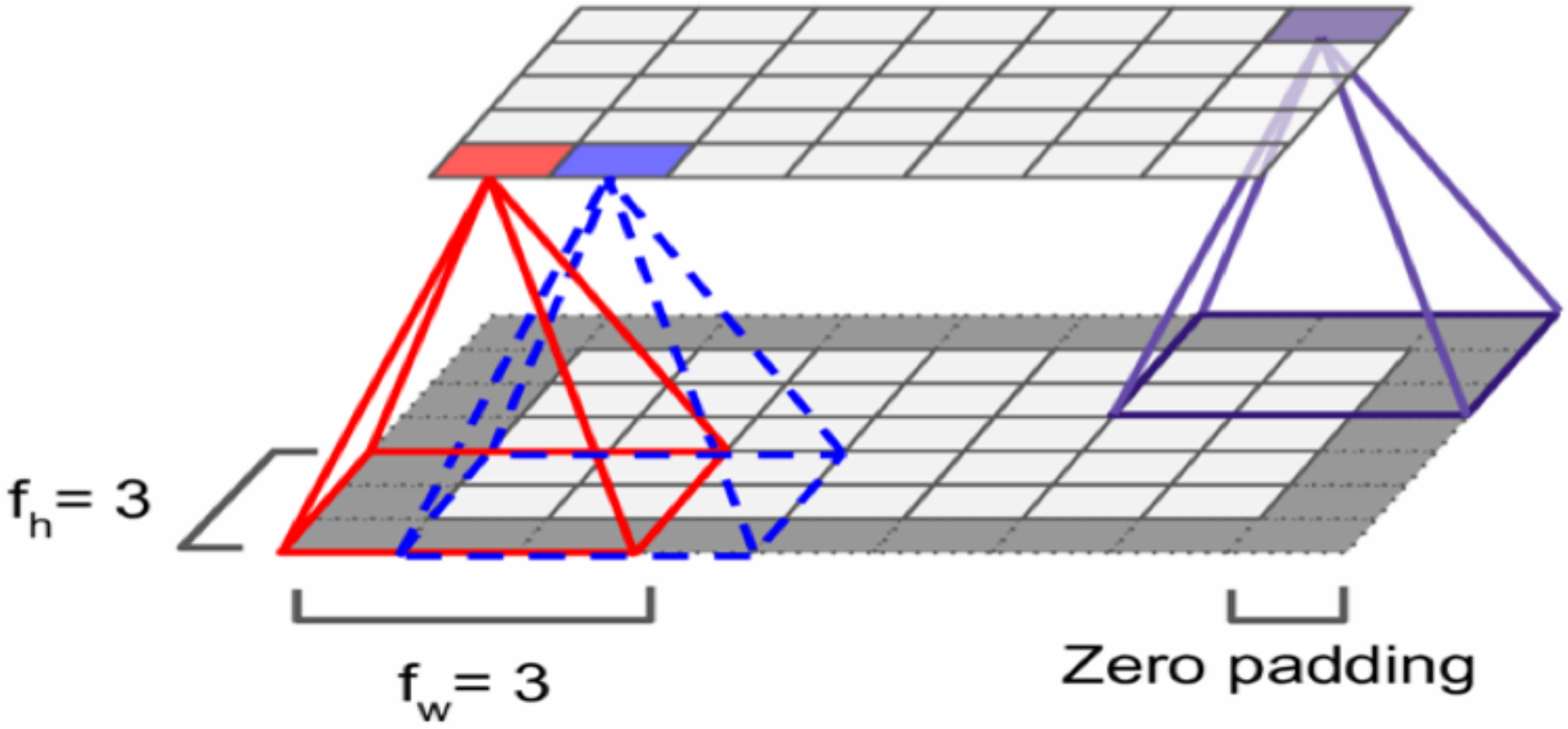
Convolutional Neural Net
CNN

Convolution



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

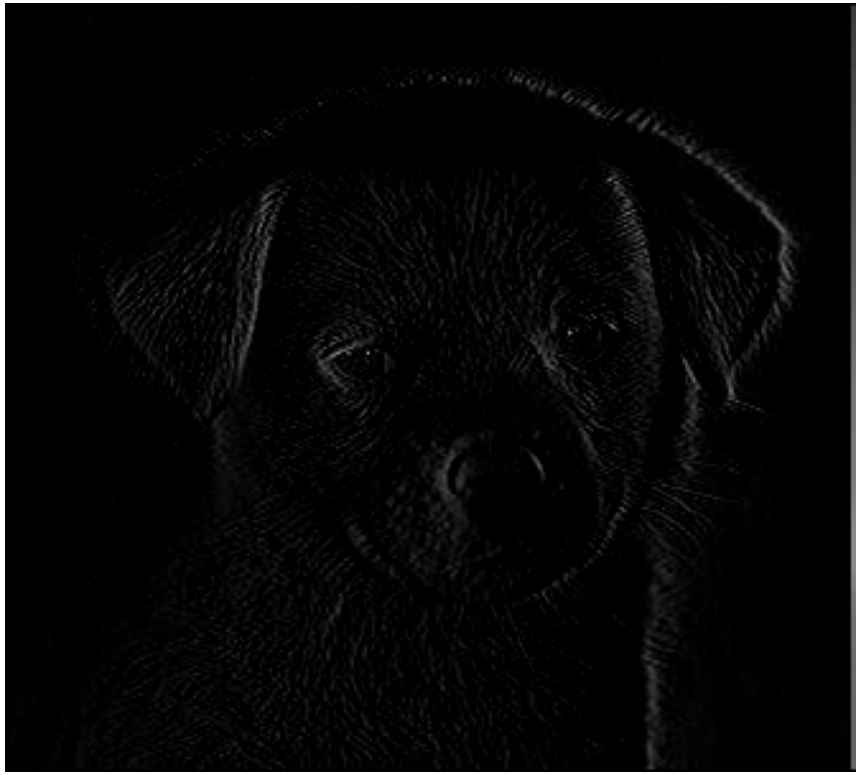
Convolutions



Run Code

- **convolution-demo.py**
- What happens if all the numbers are zeros?
- What happens if the numbers are large?
- What convolution would leave the image unaffected?

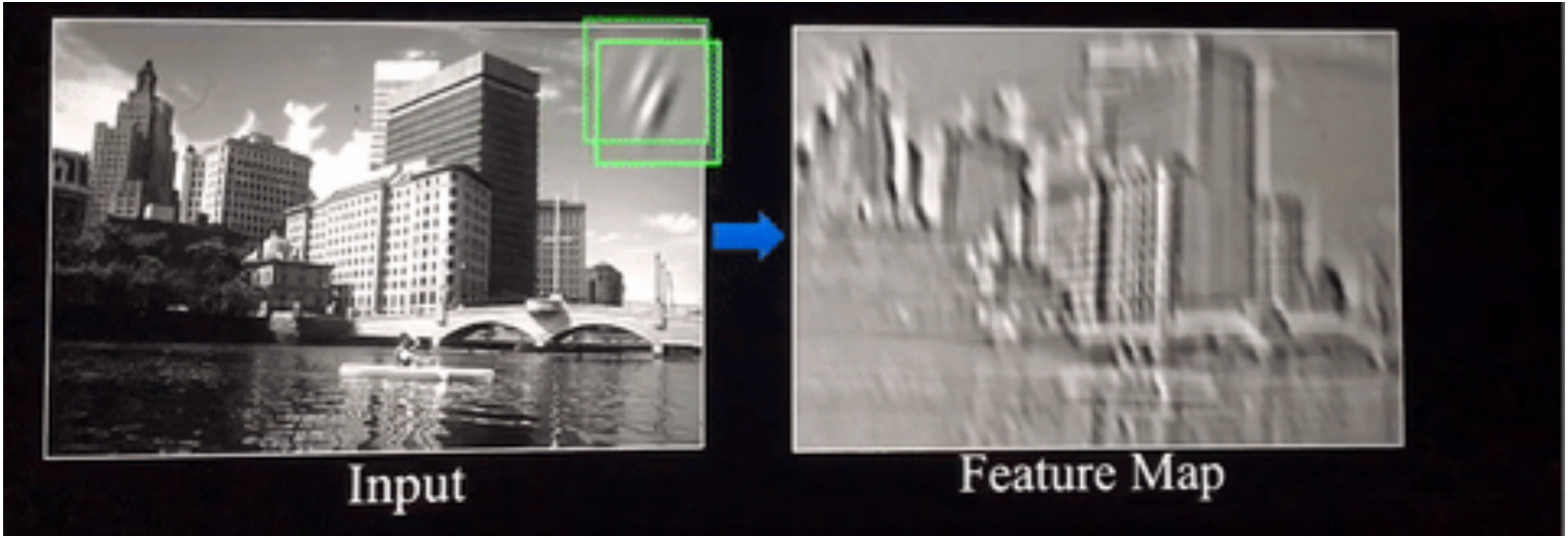
Convolutions



Horizontal edge

Vertical edge

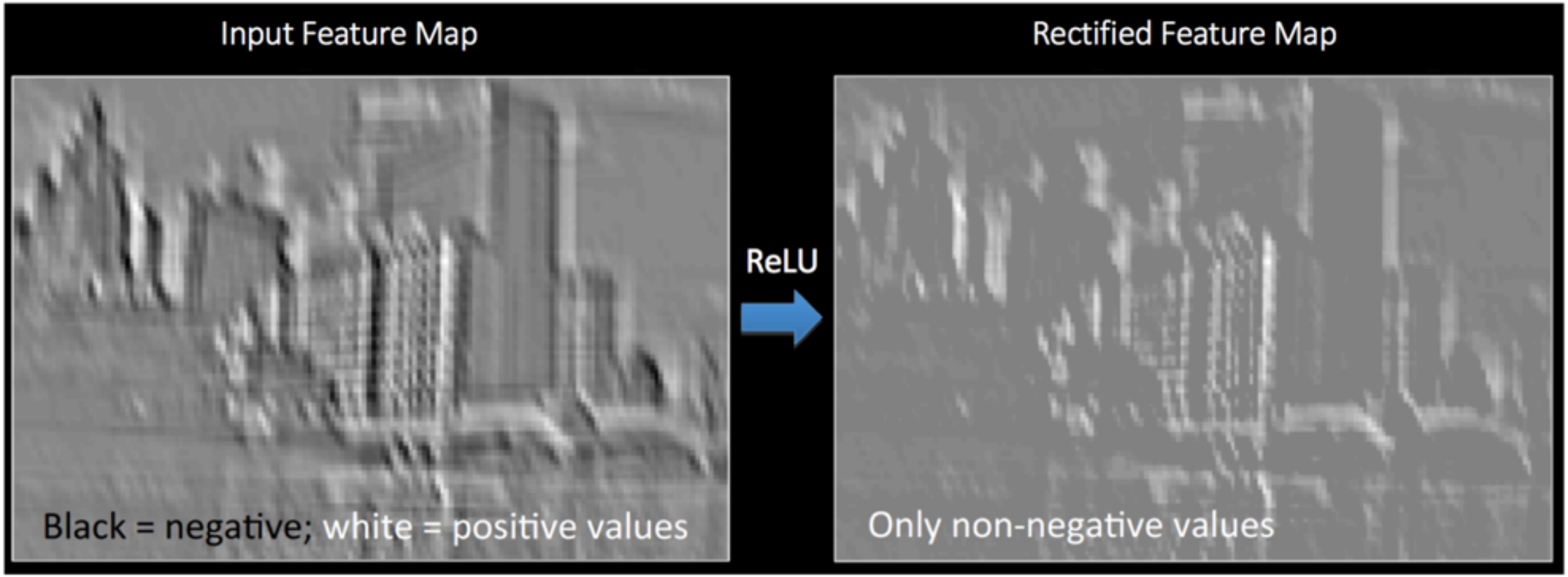
Convolutions



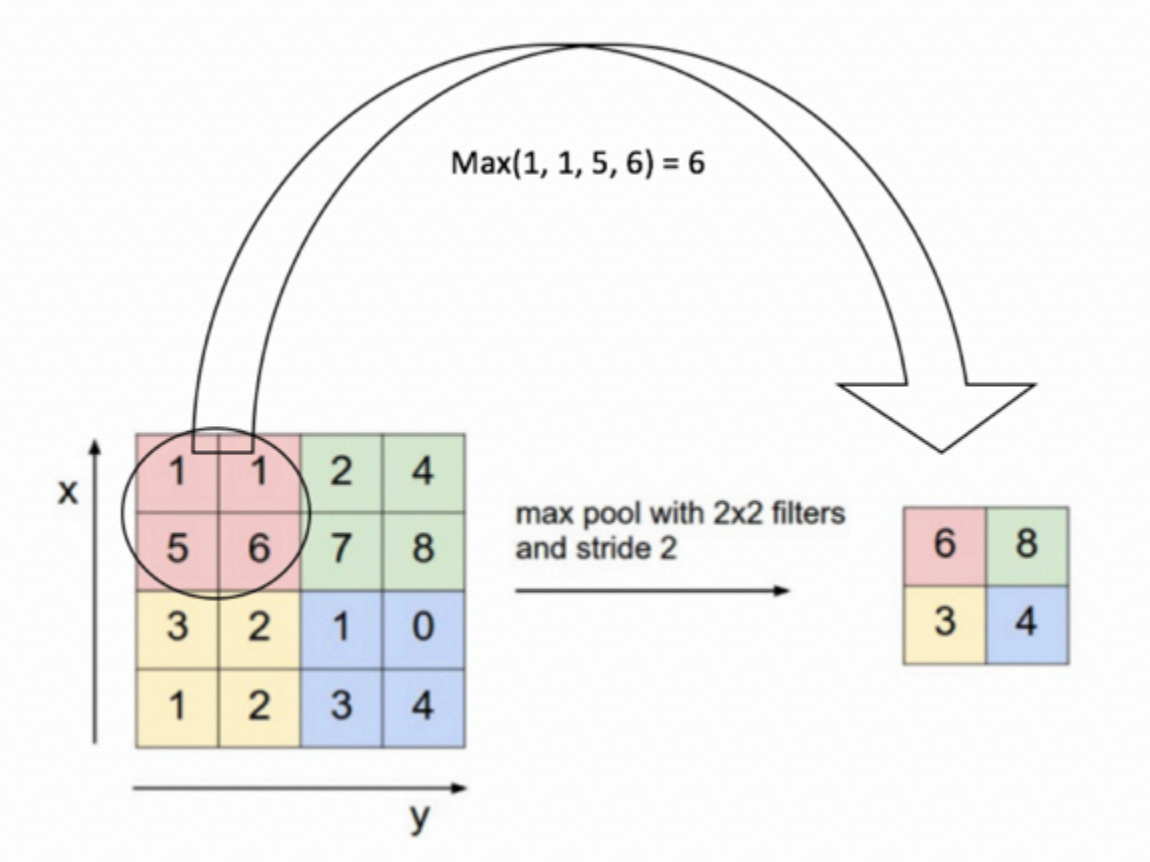
https://ujwlkarn.files.wordpress.com/2016/08/giphy.gif?w=748

Activation Function

ReLU Activation



Max Pooling


Max(1, 1, 5, 6) = 6
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Run Code

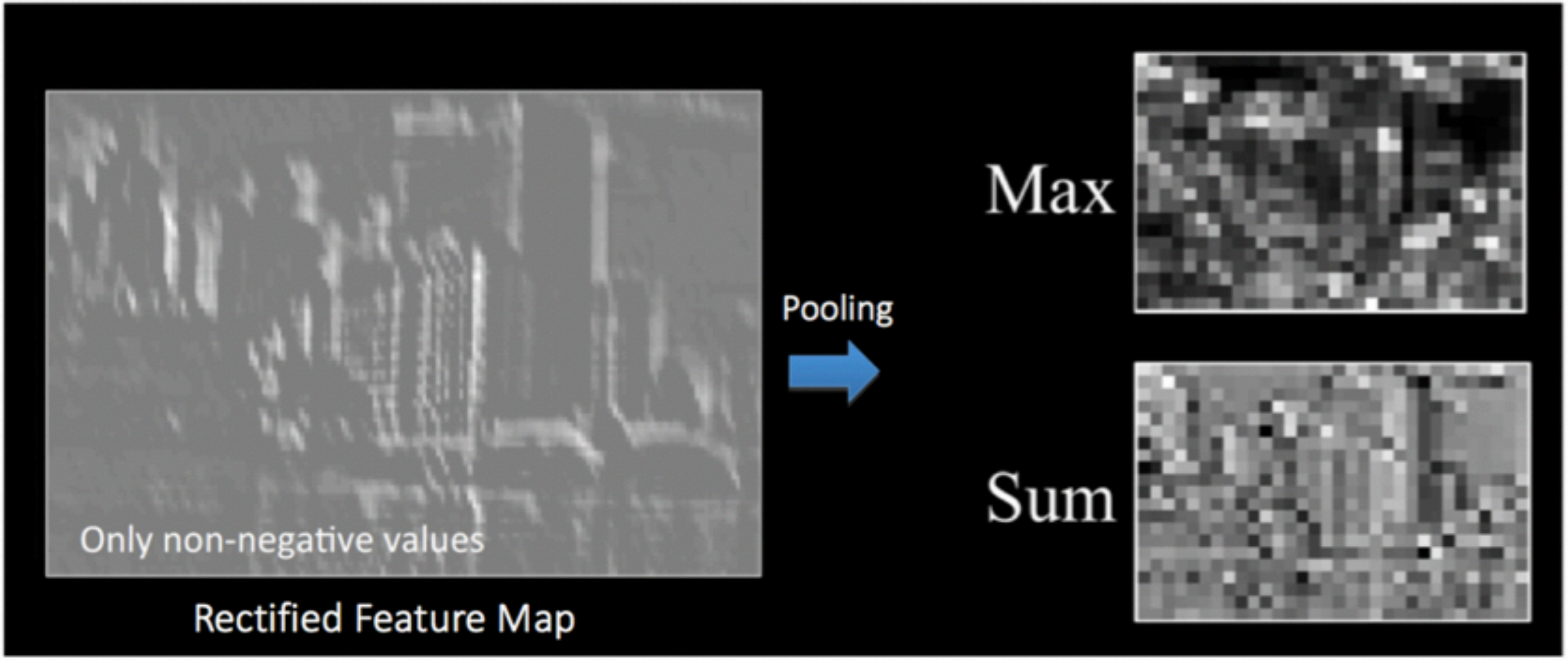
- **maxpool_demo.py**
- What is happening here?

Max Pooling

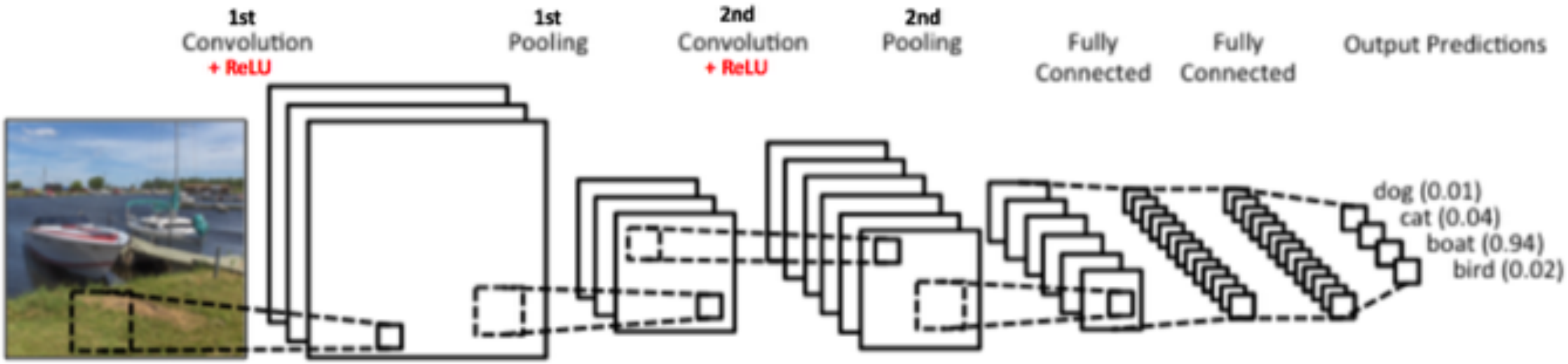
3 x 3

5 x 5

Max Pooling


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum

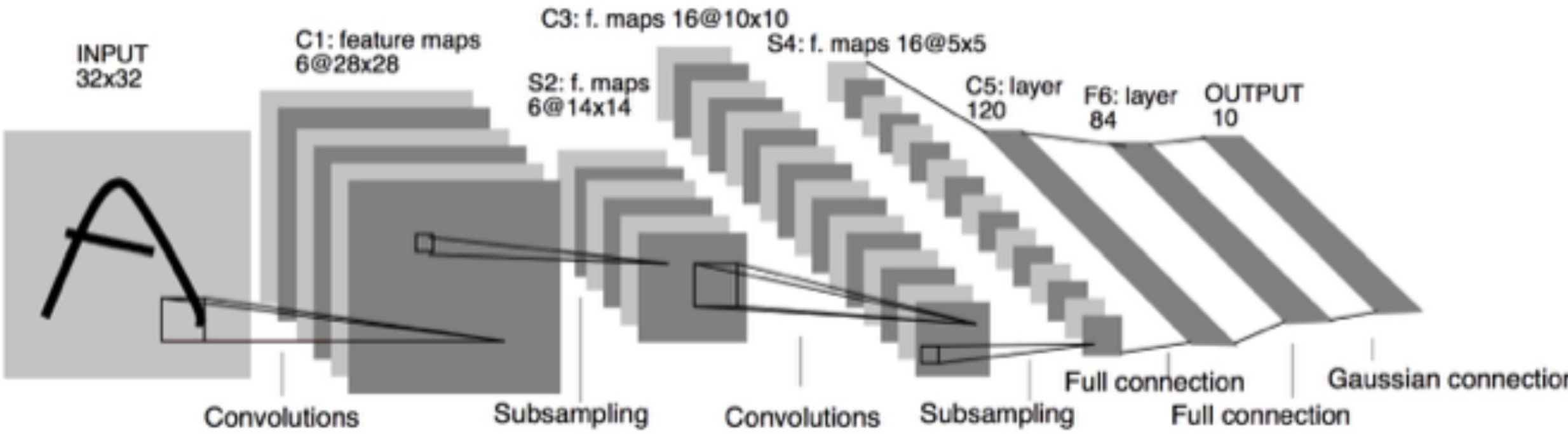
CNN Architecture



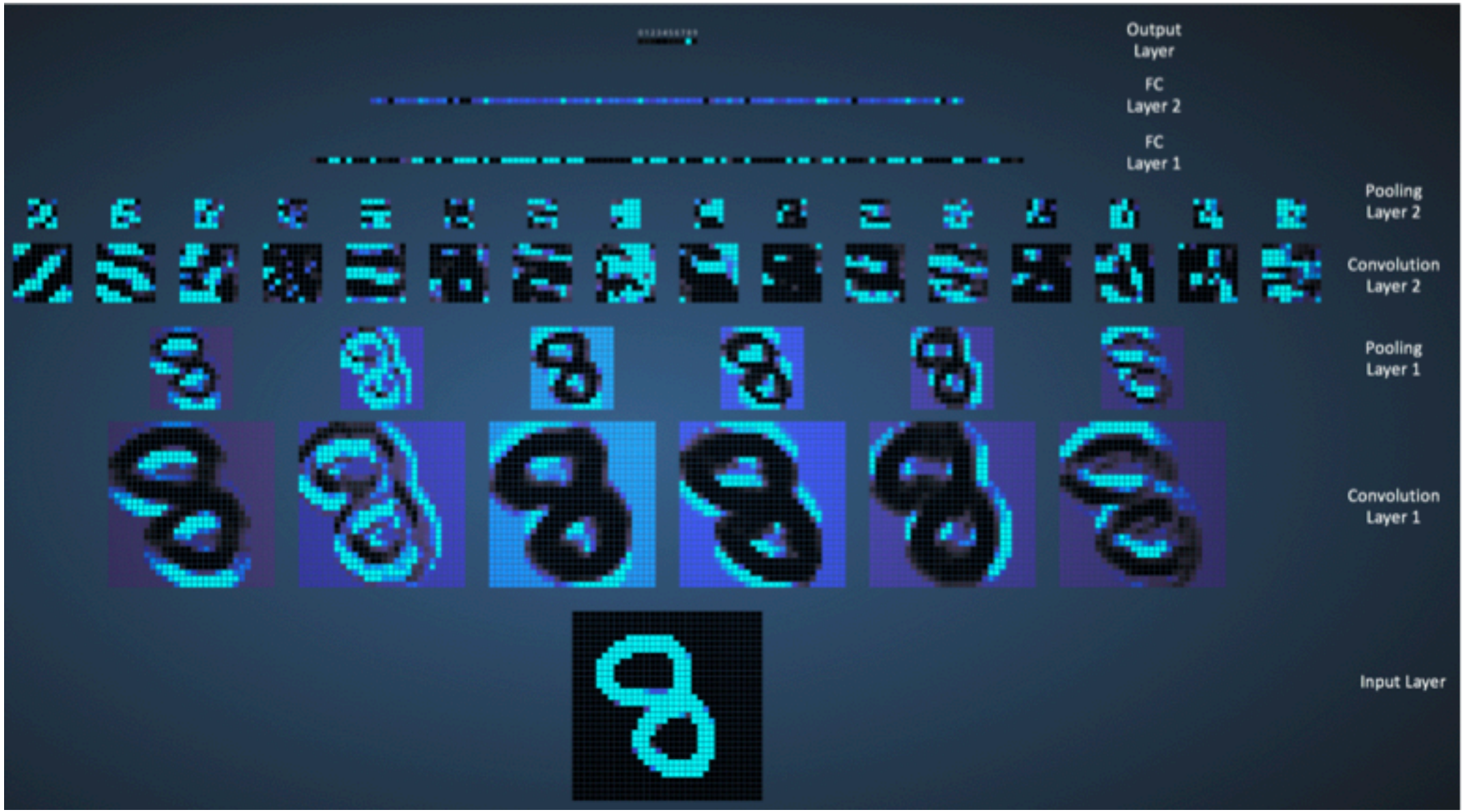
Run Code

- **cnn.py**
- What is happening here?
- Why am I reshaping the input data?
- How could I add more convolutional and pooling layers?

LeNet Architecture 1994



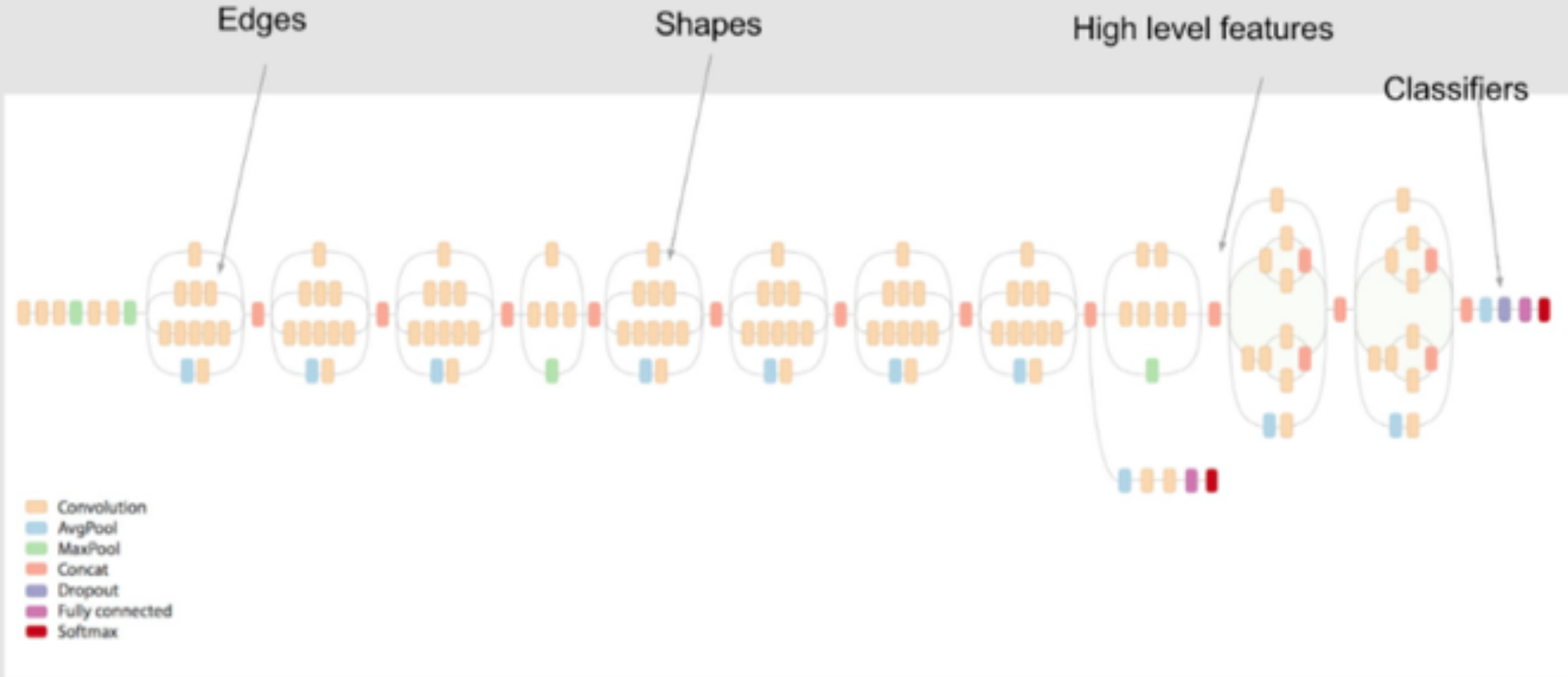
CNN


Output Layer
FC Layer 2
FC Layer 1
Pooling Layer 2
Convolution Layer 2
Pooling Layer 1
Convolution Layer 1
Input Layer

Run Code

- **LeNet.py**

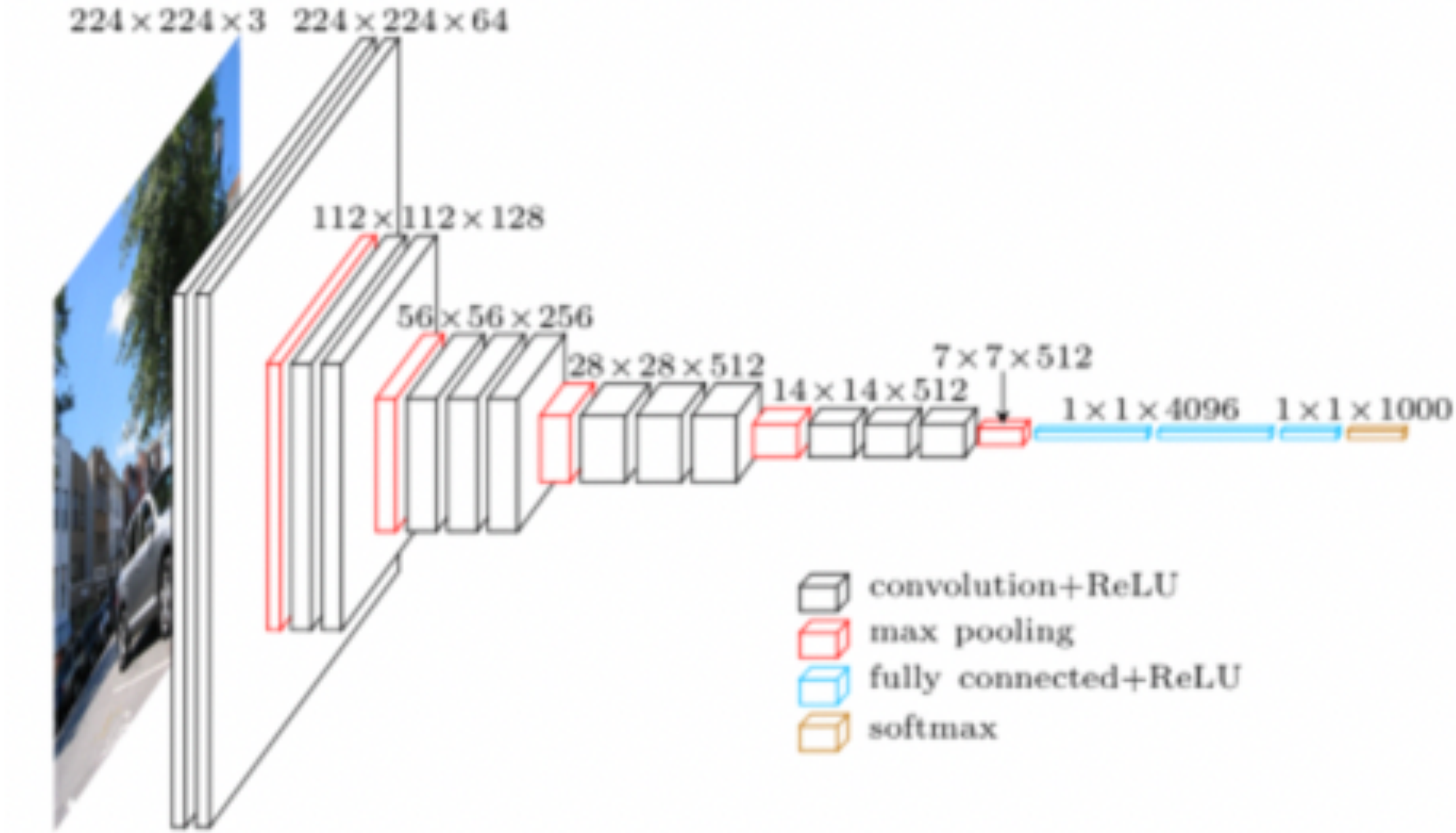
What does the layers learn?



Takeaways

- What is the difference between MLPs and CNNs?
- When might one or the other be more useful?

VGG Architecture 2014



Run Code

- **vgg_inspect.py**
- **How many parameters does this model have?**
- **Why doesn’t it overfit the training data?**

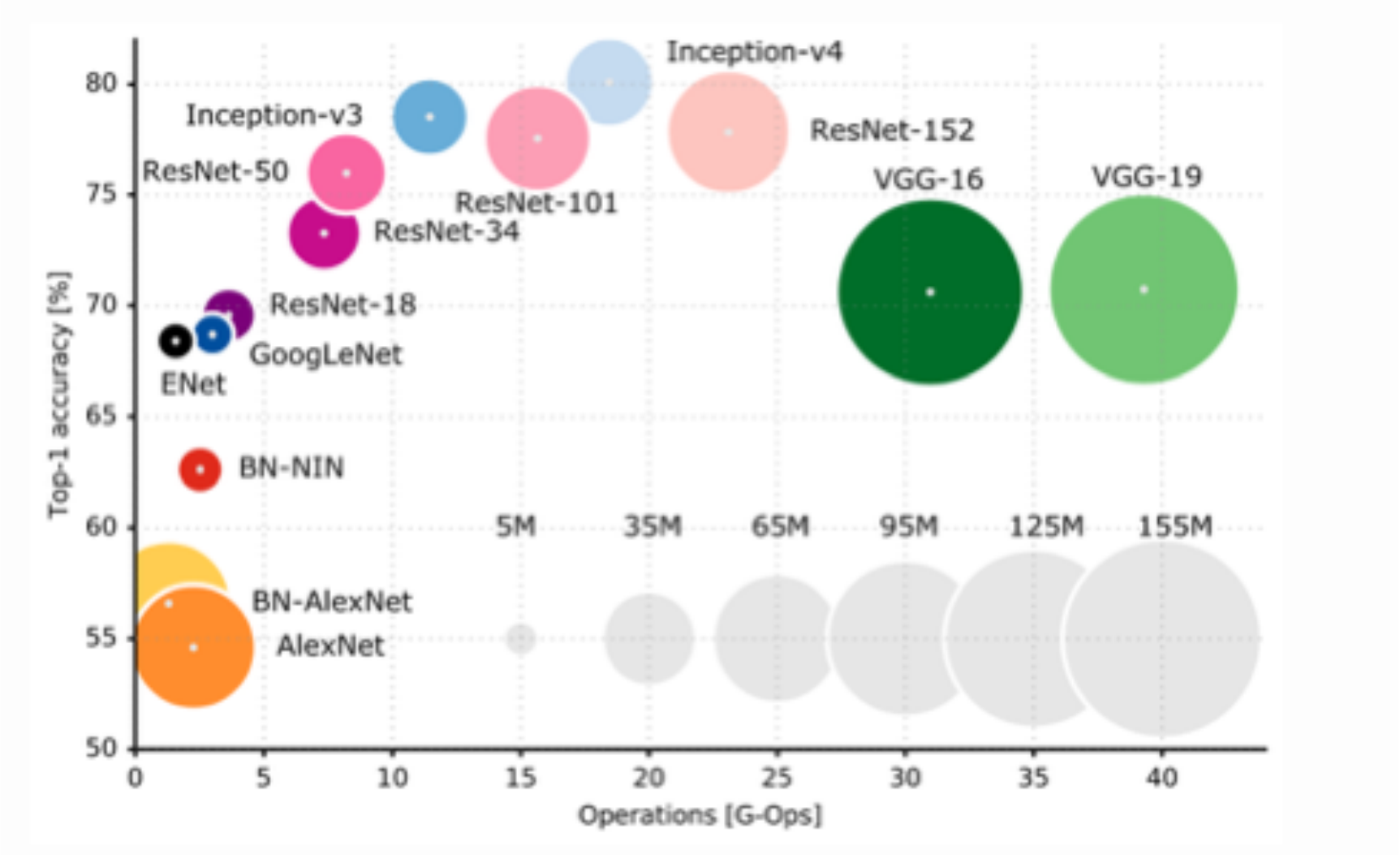
Run Code

- **resnet50_inspect.py**

Run Code

- **inception_inspect.py**

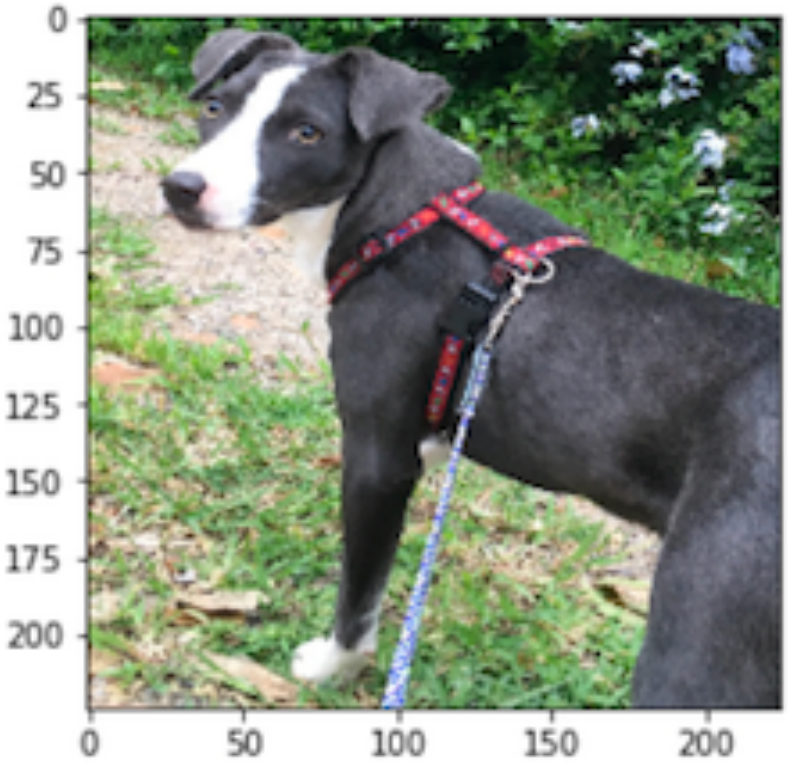
Overview of Neural Networks



Project- Dog Breed Classification

Given an image of a dog, the algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



Project- Dog Breed Classification

Step 0: Import Datasets

Step 1: Detect Humans

Step 2: Detect Dogs

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)

Step 5: Create a CNN to Classify Dog Breeds (using Transfer Learning)

Step 6: Write your Algorithm

Step 7: Test Your Algorithm

Project- Dog Breed Classification

- Git clone https://github.com/humayun/udacity_nanodegree/tree/master/p2_dog_project
- https://github.com/humayun/udacity_nanodegree/blob/master/p2_dog_project/dog_app.ipynb