

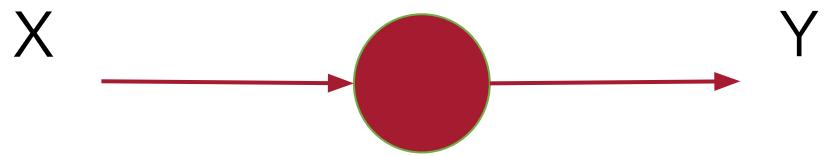


Machine Learning Fundamentals

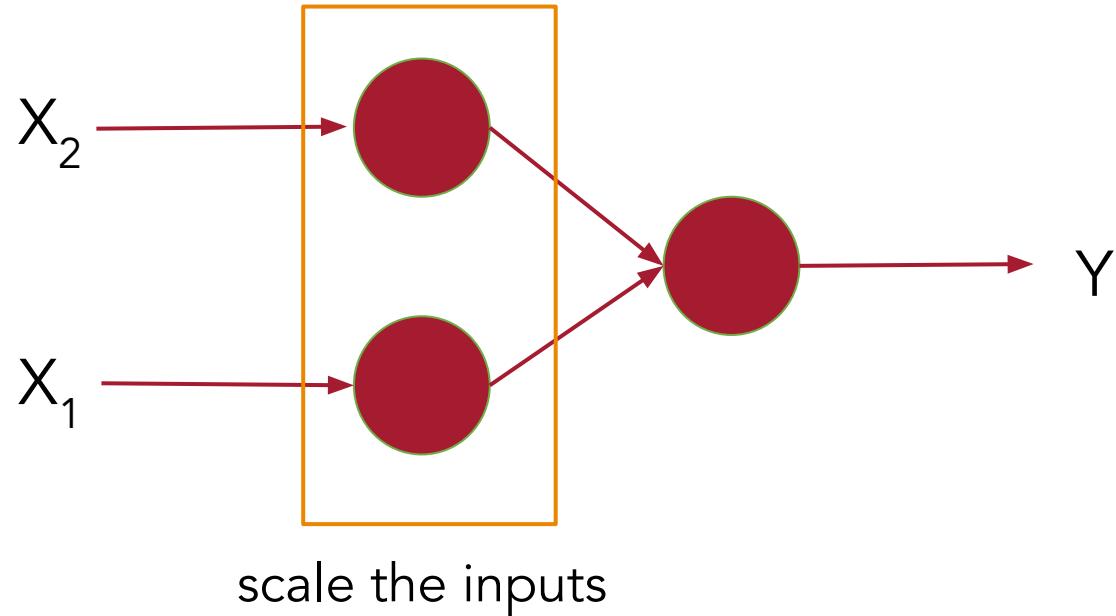
Workshop para América Latina y el Caribe (WALC)
Track 2 – Inteligencia Artificial Aplicada
November 14, 2023

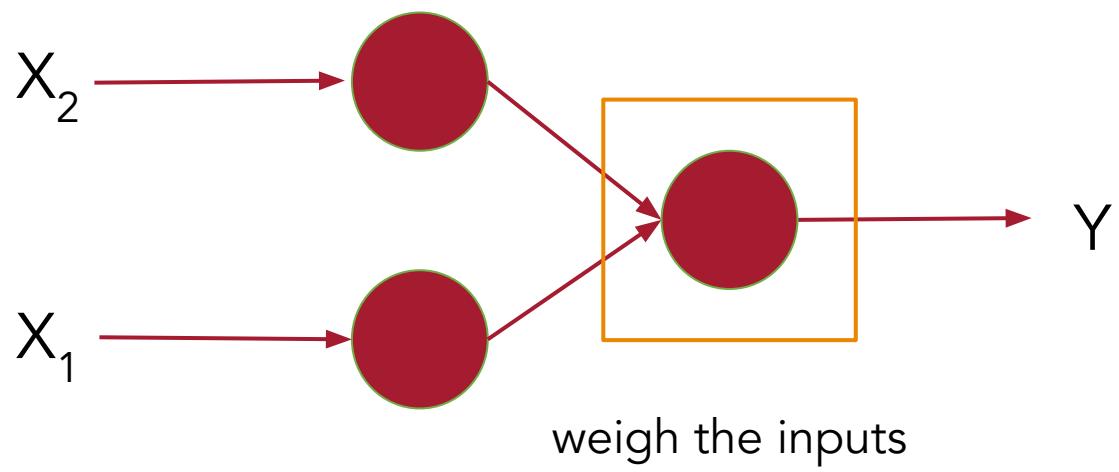
...a more
interesting
regression
application





What about more than one input?

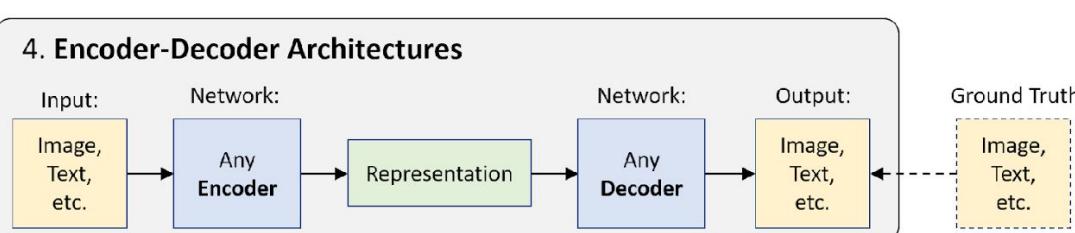
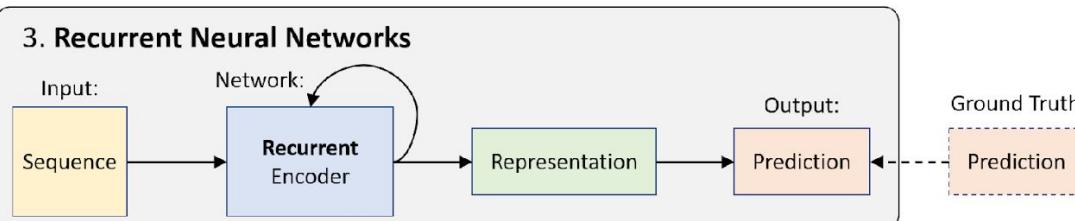
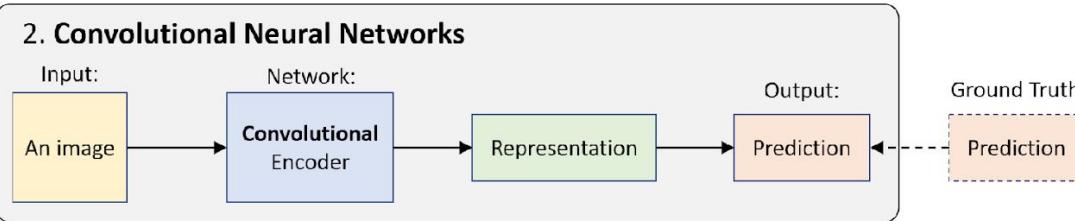
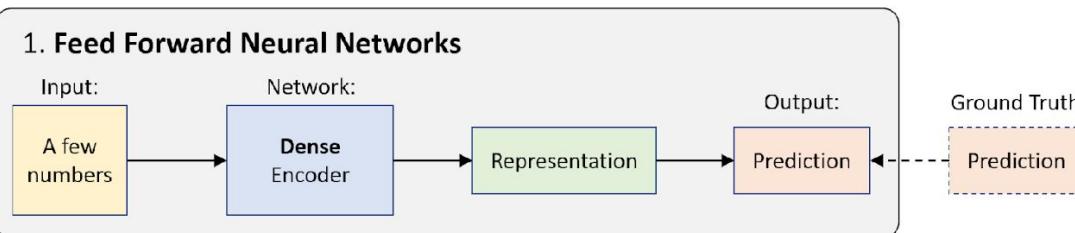




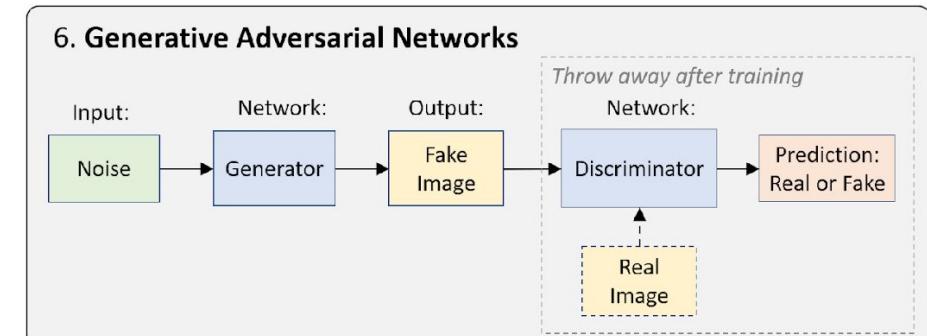
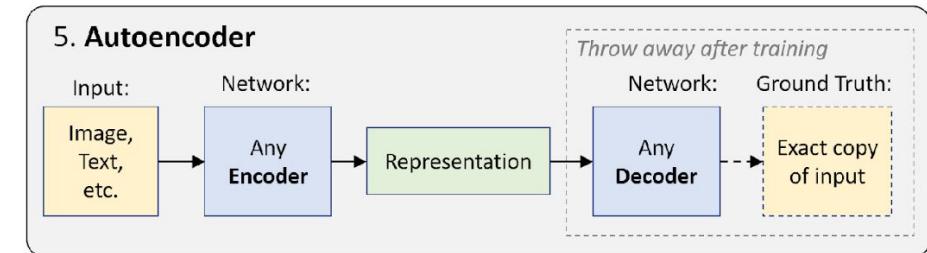
More inputs?

Machine Learning Types and Architectures

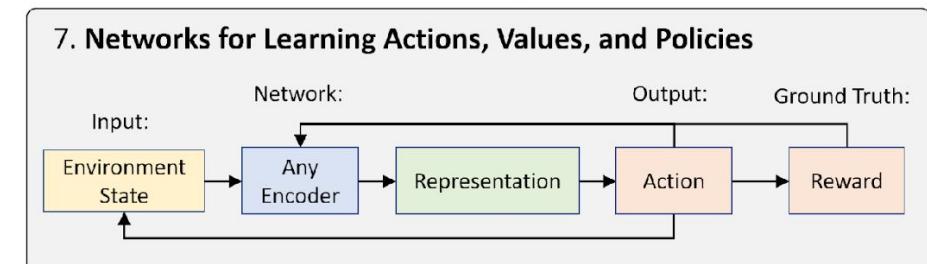
Supervised Learning



Unsupervised Learning

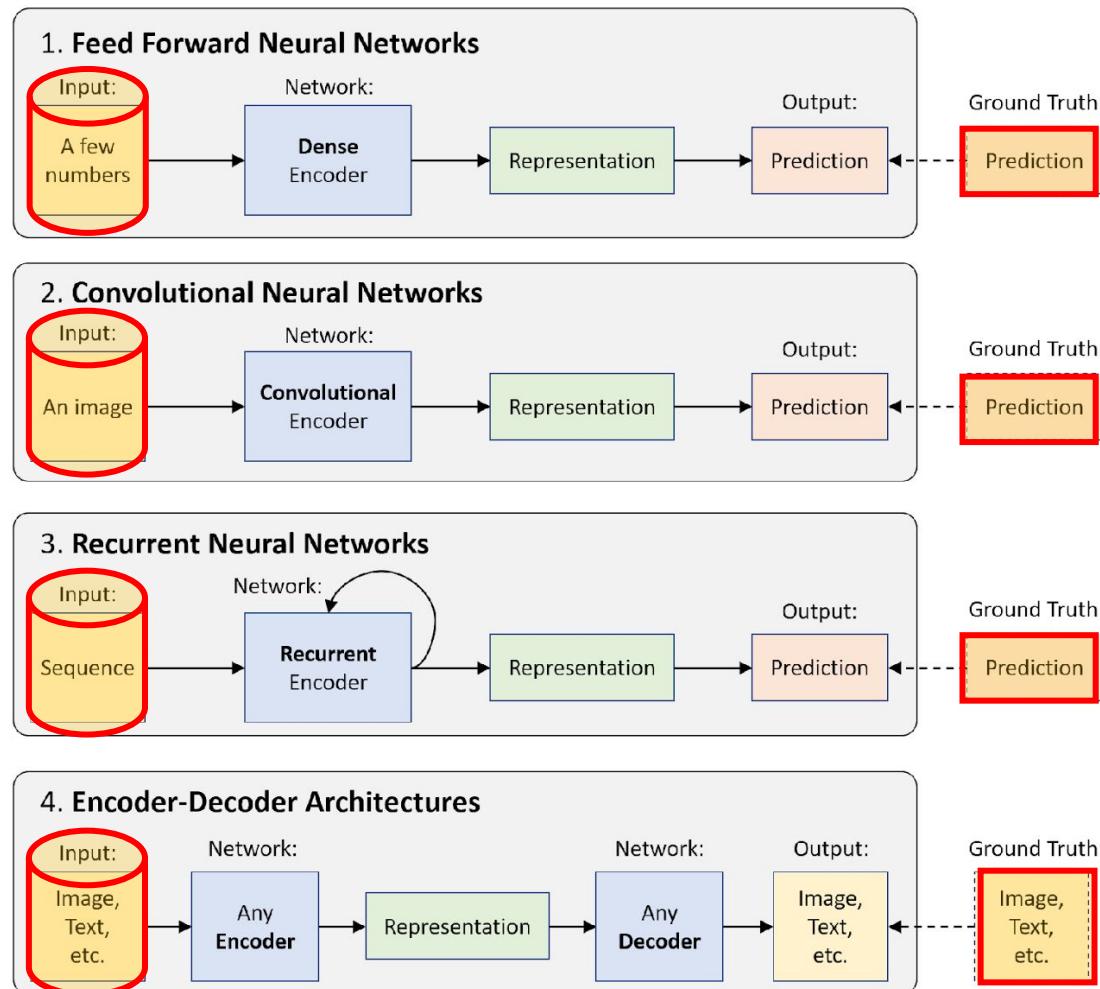


Reinforcement Learning



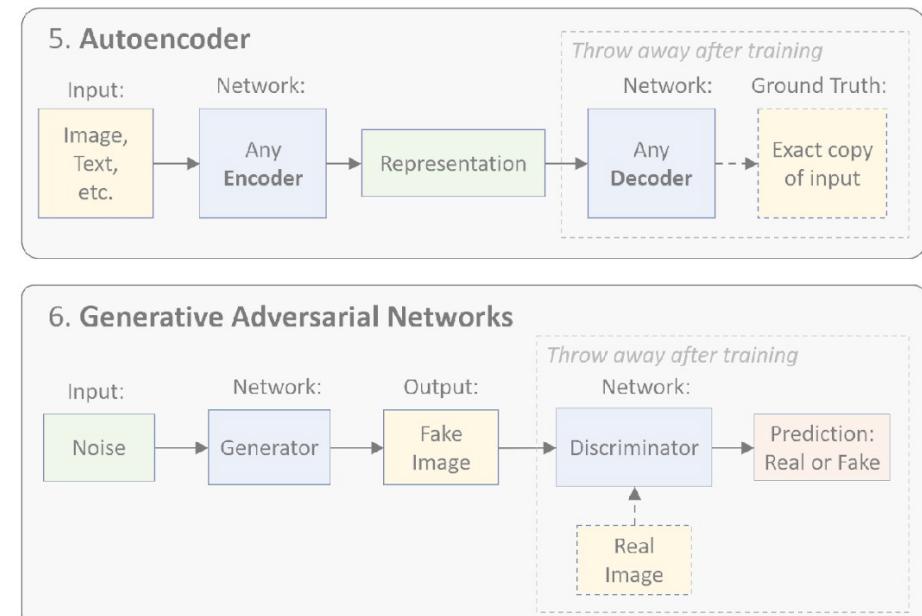
Machine Learning

Supervised Learning

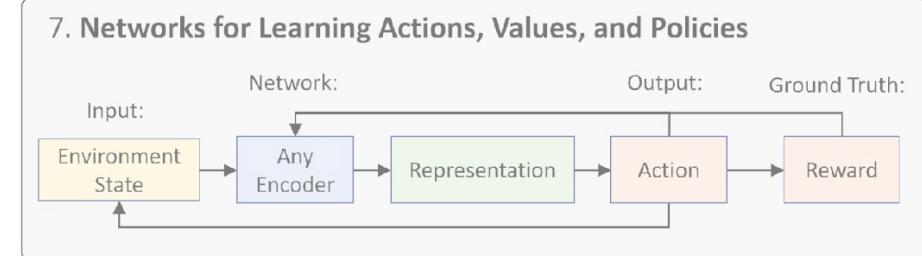


Training

Unsupervised Learning



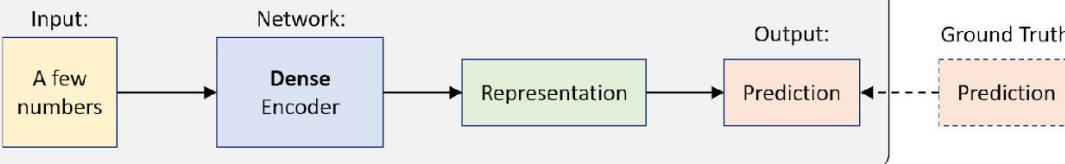
Reinforcement Learning



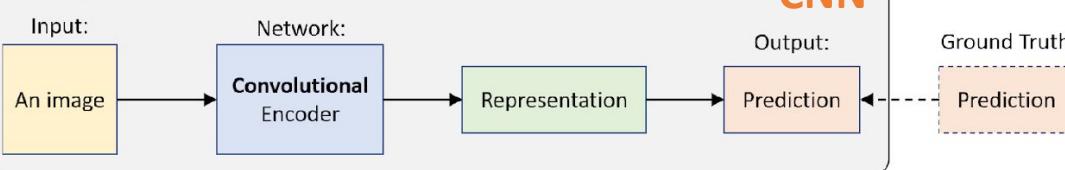
Machine Learning

Supervised Learning

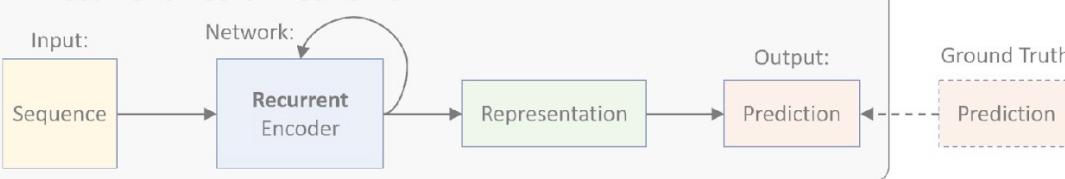
1. Feed Forward Neural Networks



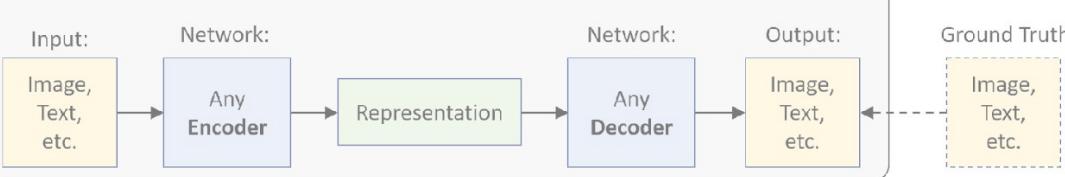
2. Convolutional Neural Networks



3. Recurrent Neural Networks



4. Encoder-Decoder Architectures

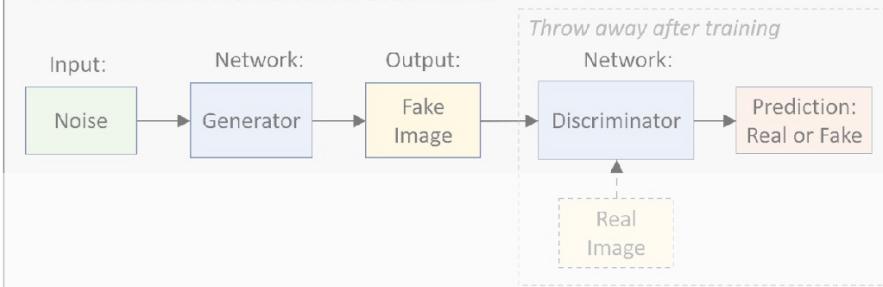


Unsupervised Learning

5. Autoencoder

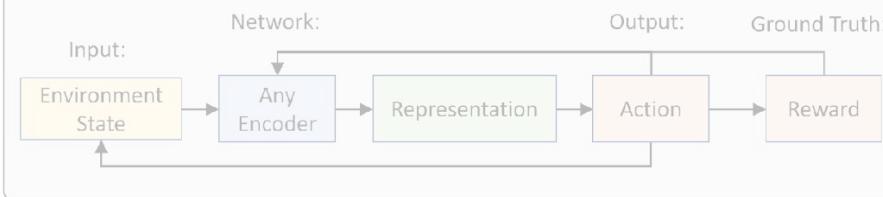


6. Generative Adversarial Networks



Reinforcement Learning

7. Networks for Learning Actions, Values, and Policies



Tiny Machine Learning

Supervised Learning

Regression

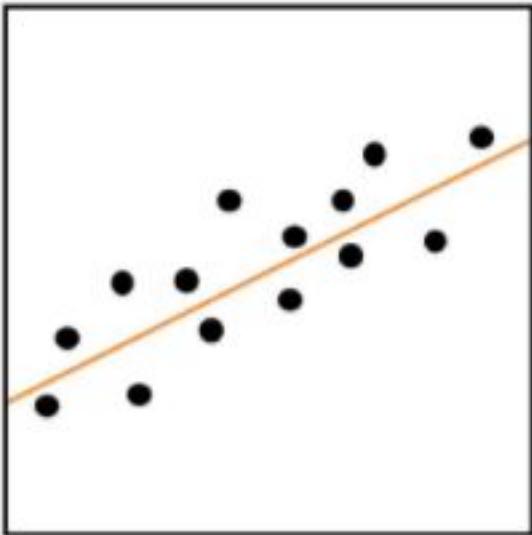
Classification

Tiny Machine Learning

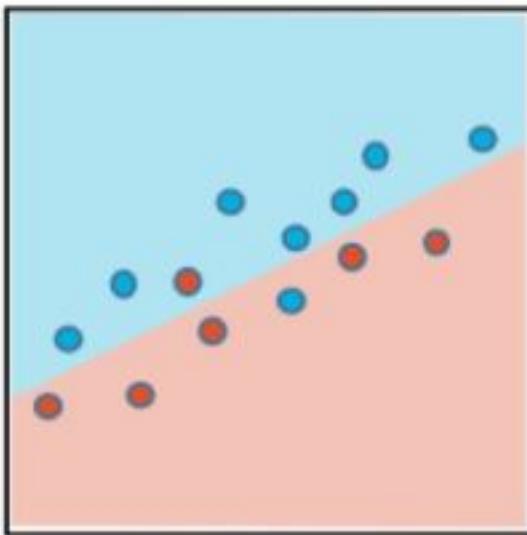
Supervised Learning

Regression

Classification



a) Regression



b) Classification

Regression

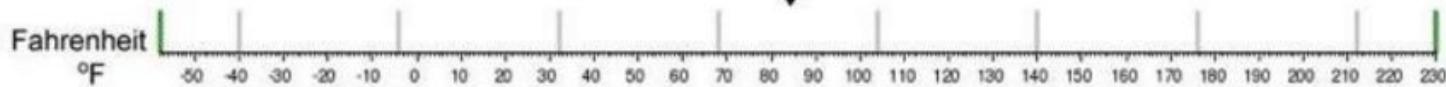


Regression

What is the temperature going to be tomorrow?

PREDICTION

84°



Classification

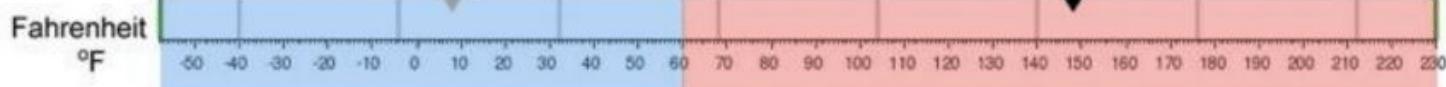


Classification

Will it be Cold or Hot tomorrow?

PREDICTION

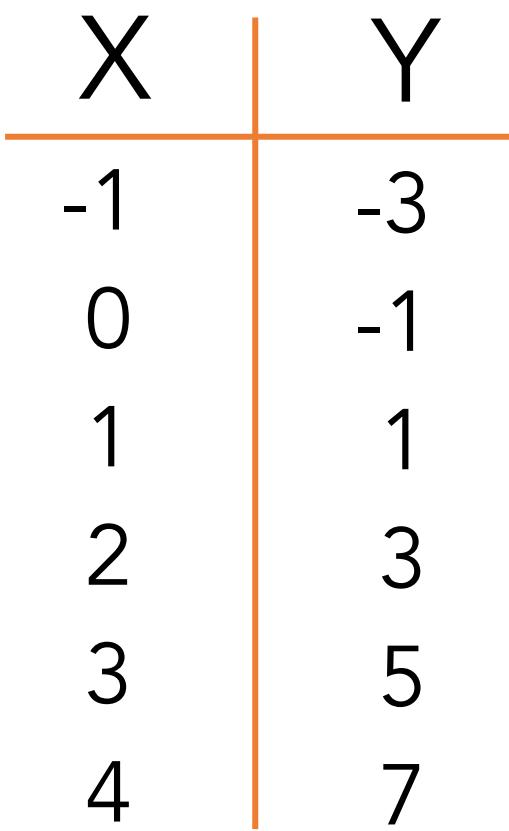
HOT



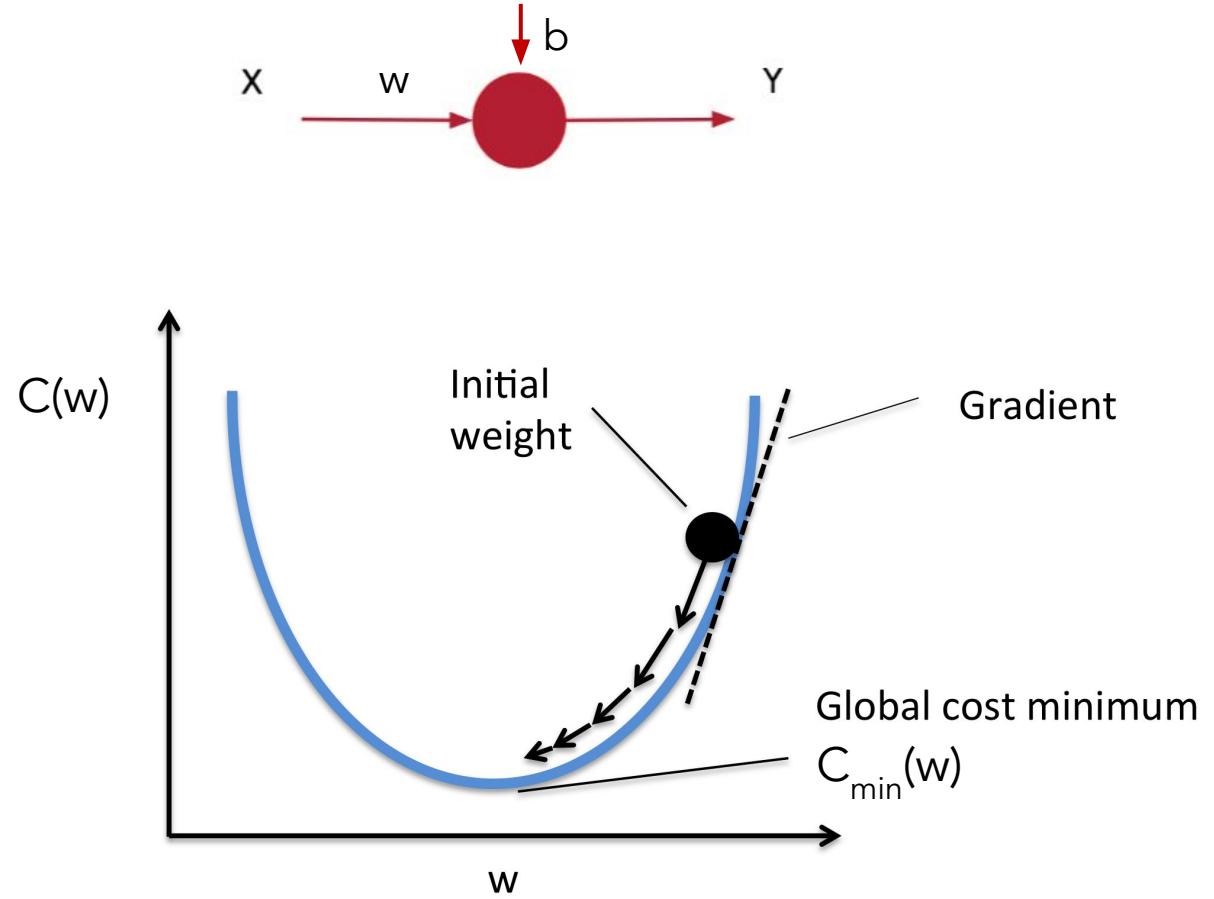
$X \rightarrow -1, 0, 1, 2, 3, 4$

$Y \rightarrow -3, -1, 1, 3, 5, 7$



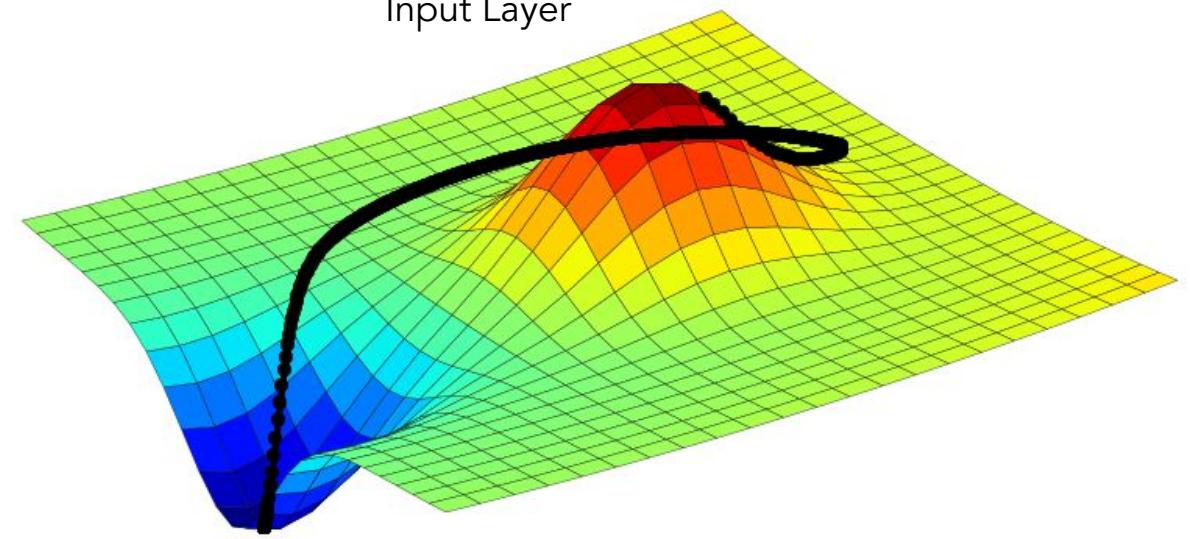
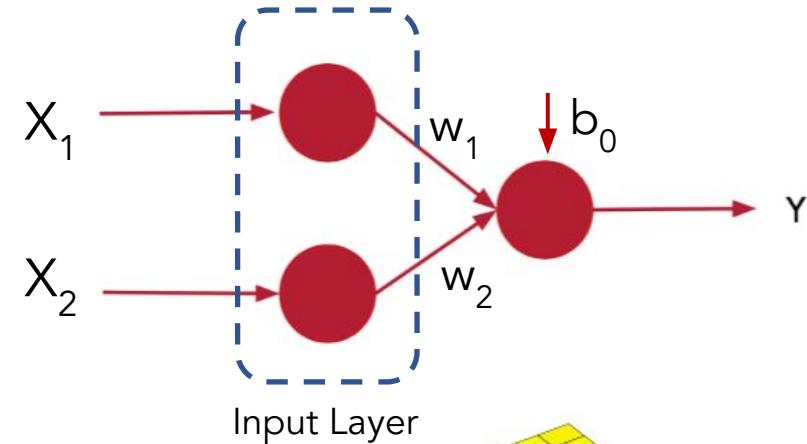


$$Y = wX + b$$



Cost Function

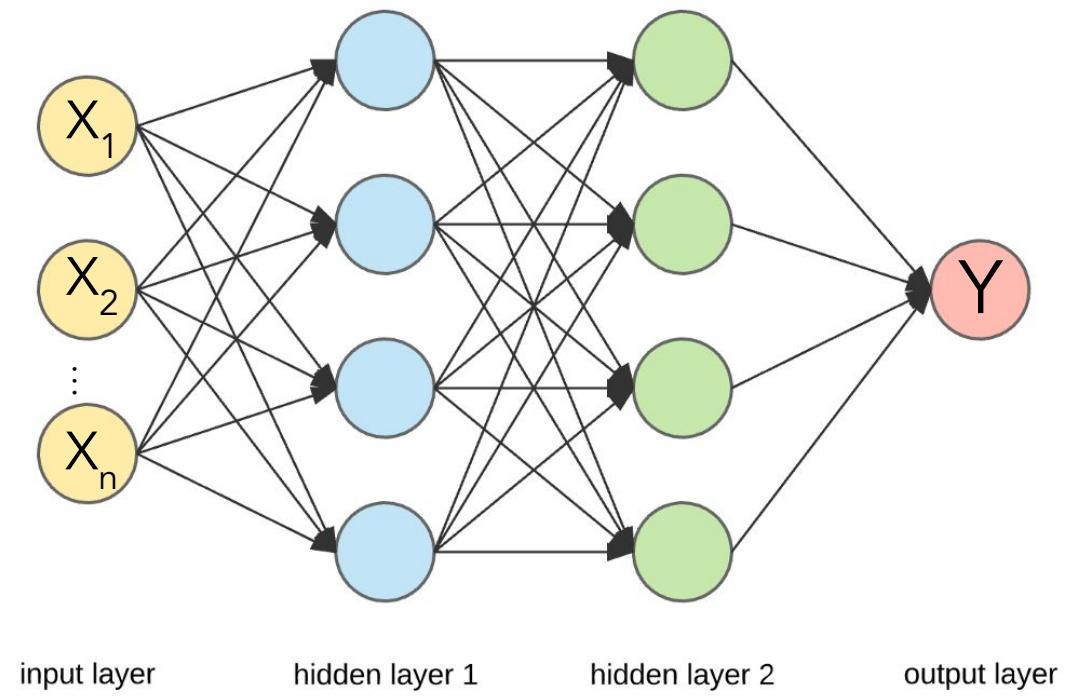
X_1	X_2	Y
-1	-8	-8
0	1	0
1	3	7
2	7	1
3	0	2
4	2	3



$$Y = w_1 X_1 + w_2 X_2 + b_0$$

Cost Function

X_1	X_2	...	X_n	Y
-1	-8		-81	-8
0	1		10	0
1	3		3	7
2	7		7	1
3	0		0	2
4	2		-7	3



$$Y = w_1 X_1 + w_2 X_2 + \dots + w_n X_n + b_0 + b_1 + \dots + b_n$$

Regression using DNN with TF2

Code Time!

[TF Boston Housing Regression.ipynb](#)



Machine Learning Workflow

Collect
Data

```
data = tf.keras.datasets.boston_housing  
  
(x_train, y_train), (x_test, y_test) = data.load_data()
```

- The Boston Housing dataset is taken from the StatLib library which is maintained at CMU.
- There are 506 samples (404:102), each one with 13 attributes (Features X_i from 0 to 12) of houses at different locations around the Boston suburbs in the late 1970s. The attributes themselves are defined in the StatLib website (as per capita crime rate in the area, number of rooms, distance from employment center, etc.).
- Target (Y) is the median values of the houses at a location (in USD 1,000).

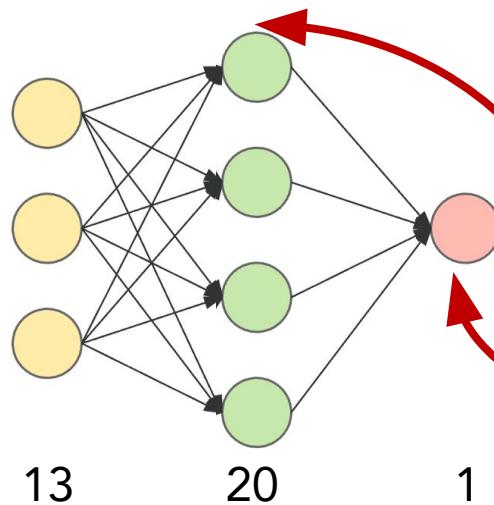
Machine Learning Workflow



```
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
scaler.fit(x_train)  
  
x_train_norm = scaler.transform(x_train)  
x_test_norm = scaler.transform(x_test)
```

Normalizing Data: We notice that values range varies depending on the type of the feature. If we are training a neural network, for various reasons it's easier if we treat all values as between 0 and 1 (or at least with similar ranges), a process called 'normalizing'. In this case, all features will be rescaled.

Machine Learning Workflow



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20,
        activation='relu',
        input_shape = [13]),
    tf.keras.layers.Dense(1)
])

model.compile(
    optimizer='adam',
    loss='mse',
    metrics=[ 'mae' ]
)
```

ADAM, a stochastic gradient descent method

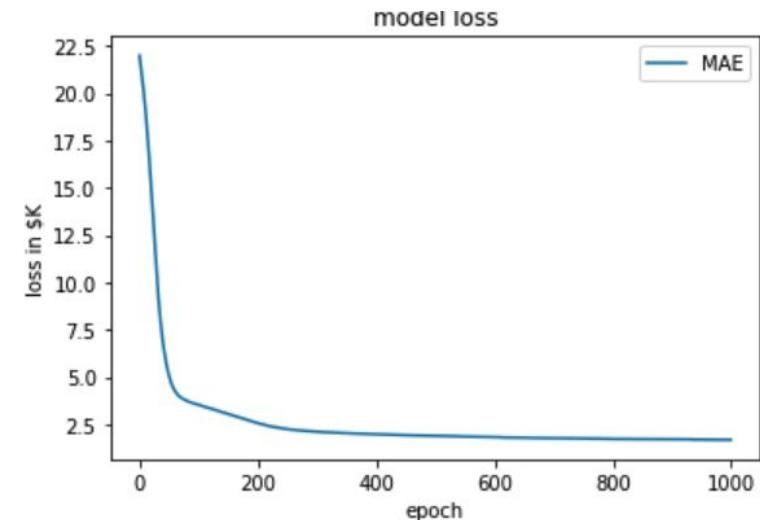
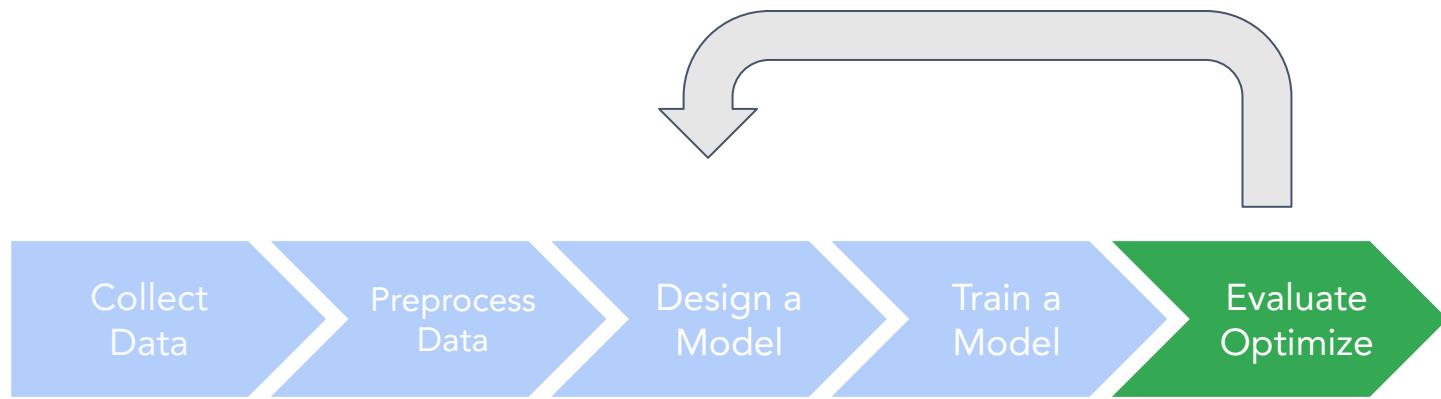
- Total parameters in the dense layer will be 13×20 weights (260) and 20 biases => 280.
- The output layer will be only one neuron that has one input for each output of the neurons in the previous layer (20) and 1 bias => 21.

Machine Learning Workflow



```
history = model.fit(  
    x_train_norm,  
    y_train,  
    epochs=1000,  
    verbose=0  
)
```

Machine Learning Workflow



```
tuner.results_summary()

Results summary
Results in ./untitled_project
Showing 10 best trials
Objective(name="val_loss", direction="min")

Trial 1 summary
Hyperparameters:
units: 30
Score: 16.199304580688477

Trial 0 summary
Hyperparameters:
units: 20
Score: 17.700151443481445

Trial 2 summary
Hyperparameters:
units: 10
Score: 18.69426727294922
```

```
train_eval = model.evaluate(x_train_norm, y_train)
print ("Training data MSE: {:.2}").format(train_eval[1])

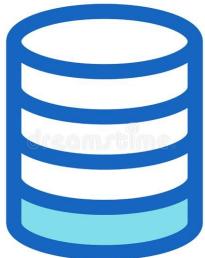
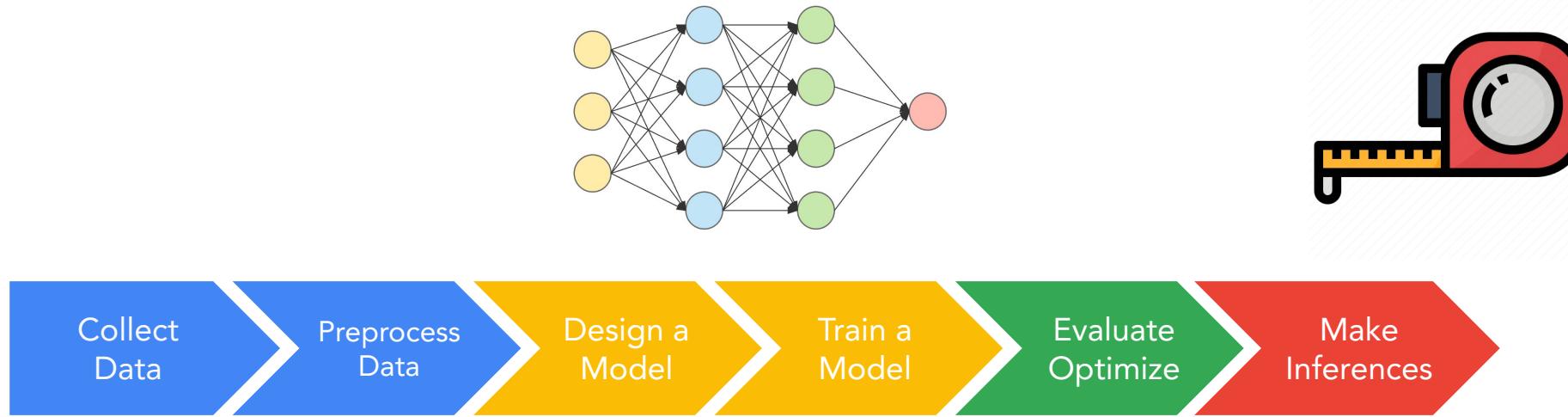
tuner.search(
    x_train_norm, y_train,
    epochs=500,
    validation_data=(x_test_norm, y_test))
```

Machine Learning Workflow



```
xt = np.array([1.1, 0., 9., 0., 0.6, 7., 92., 3.8, 4., 300., 21., 200, 19.5])
xt = np.reshape(xt, (1, 13))
xt_norm = scaler.transform(xt)
yt = model.predict(xt_norm)
```

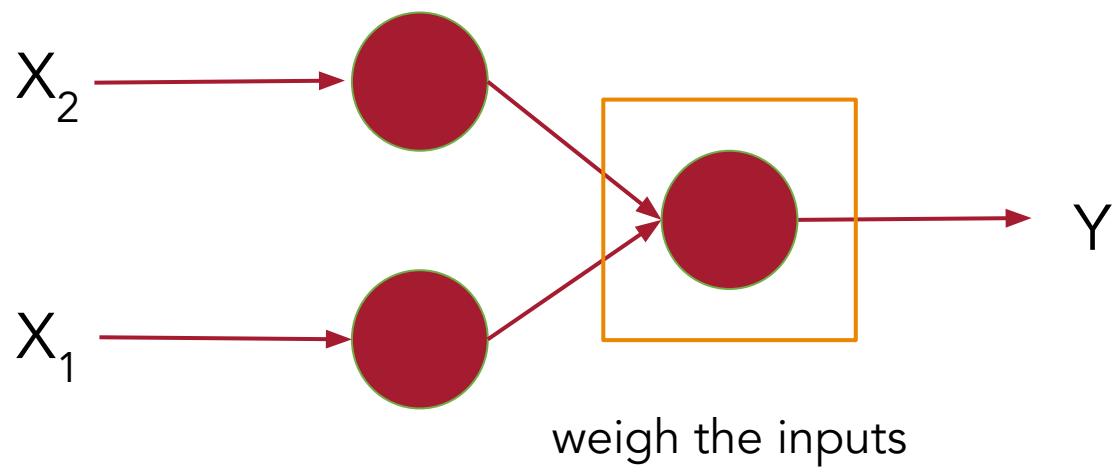
Machine Learning Workflow



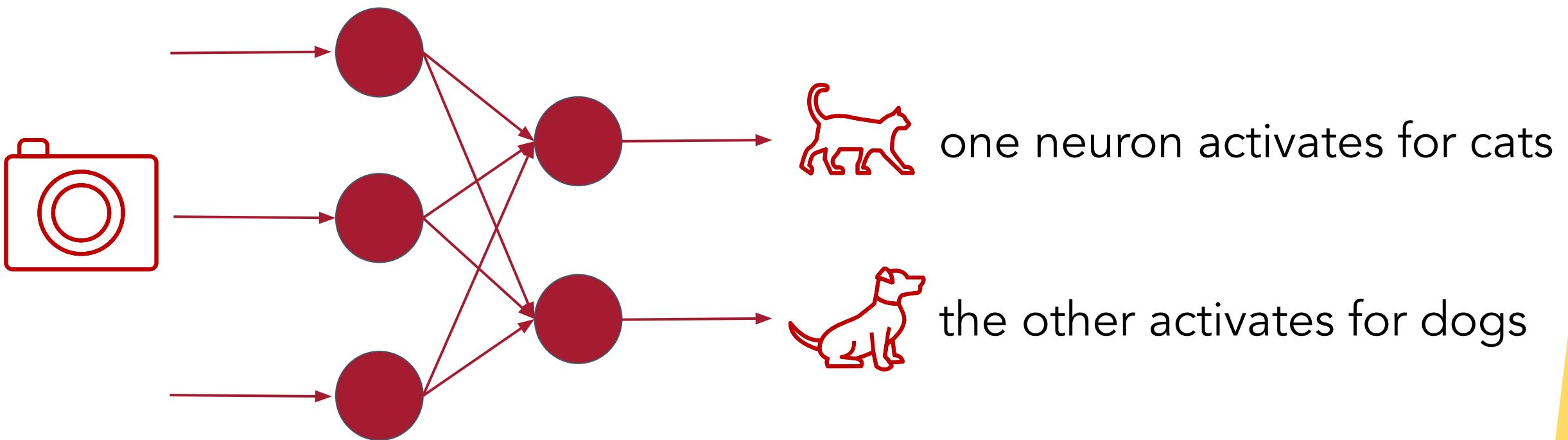


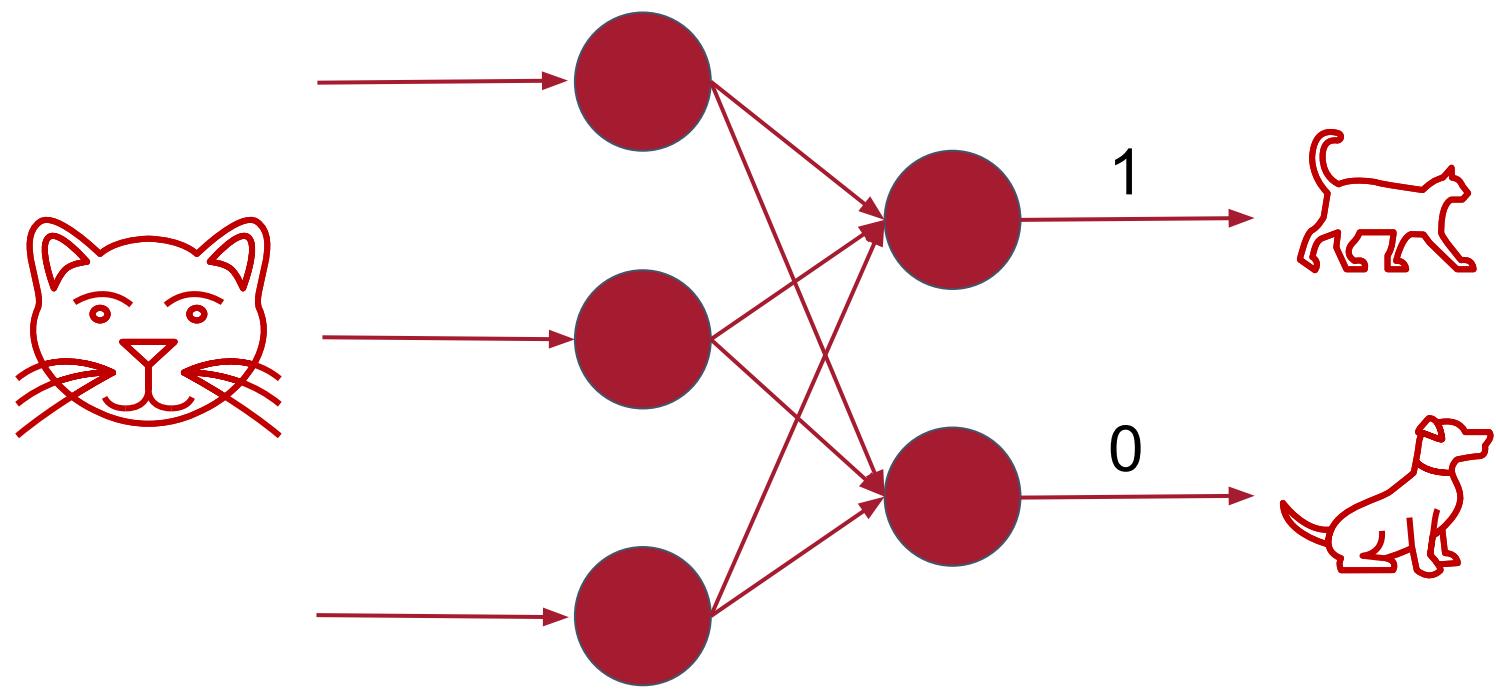
Now, Classification

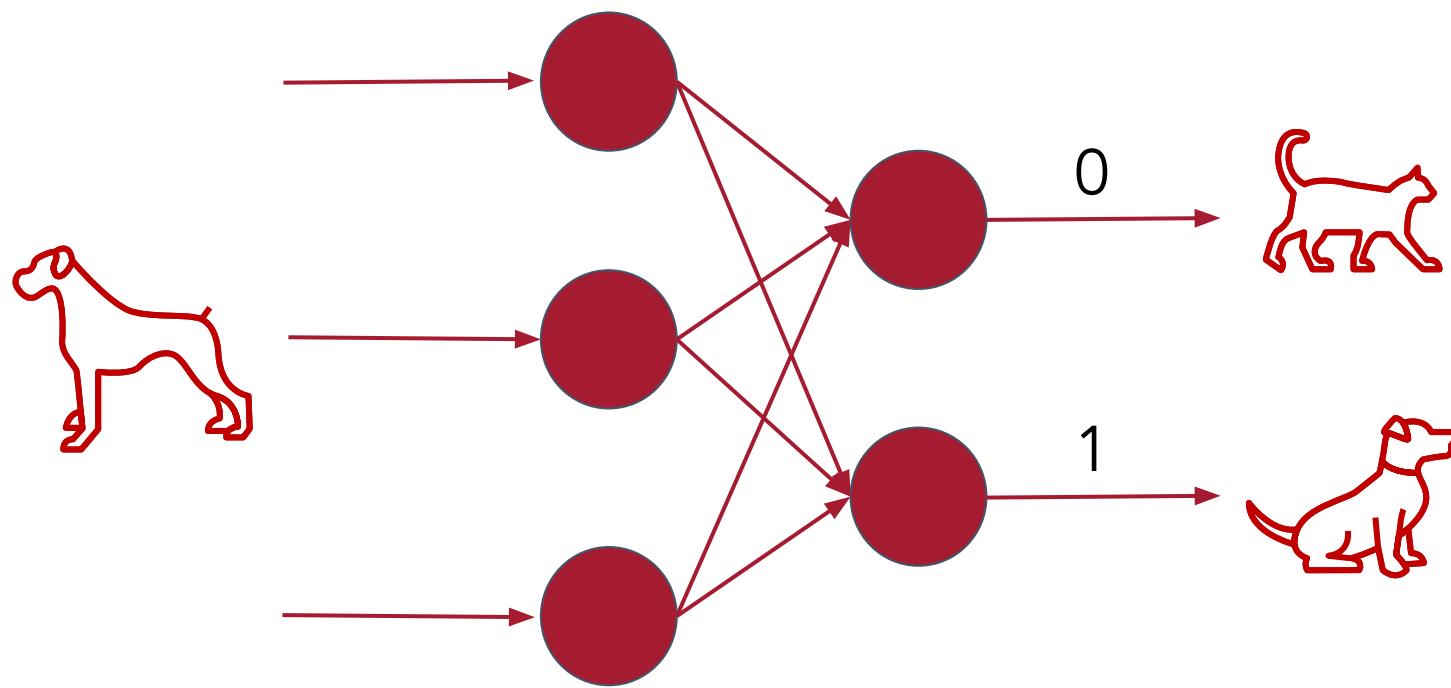




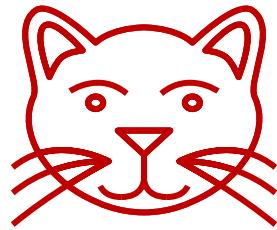
More inputs?





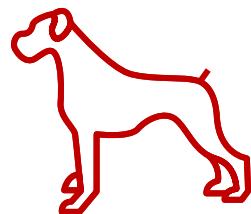


Data



Label

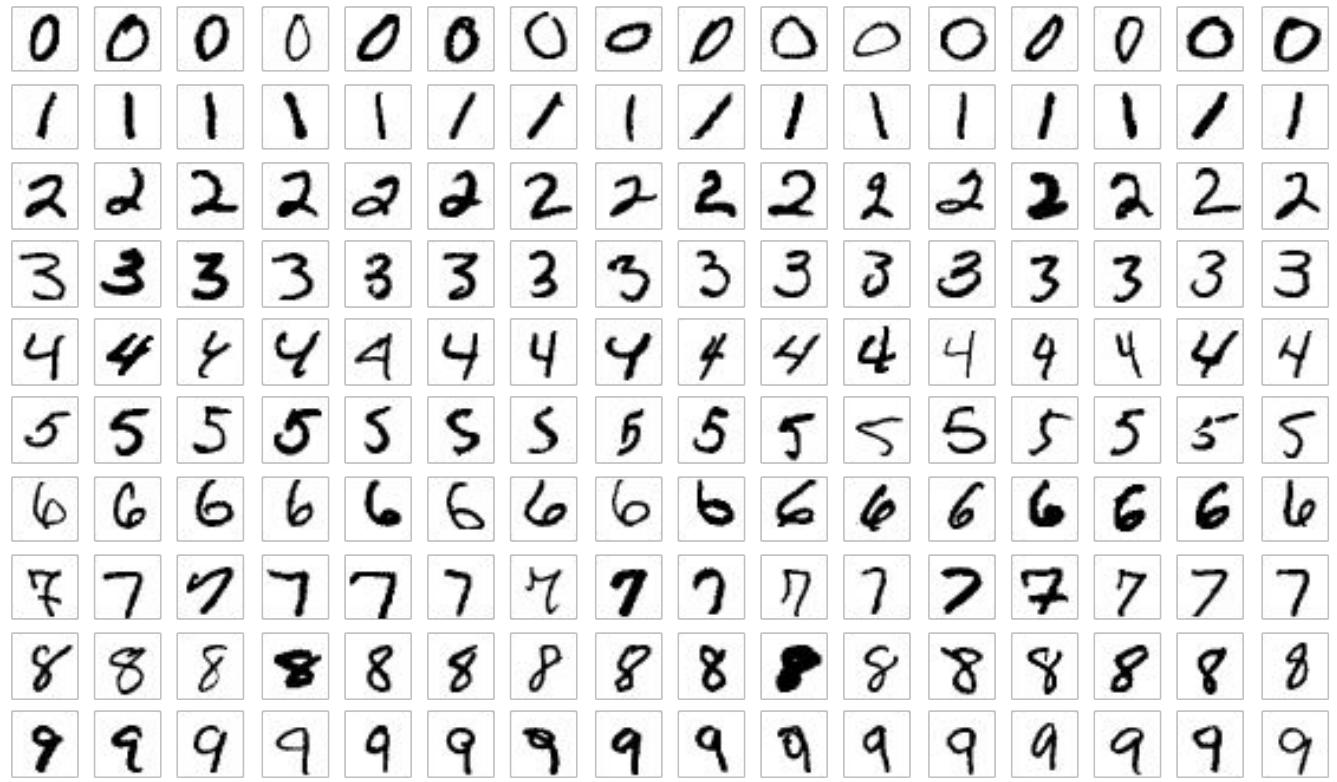
[1, 0]



[0, 1]

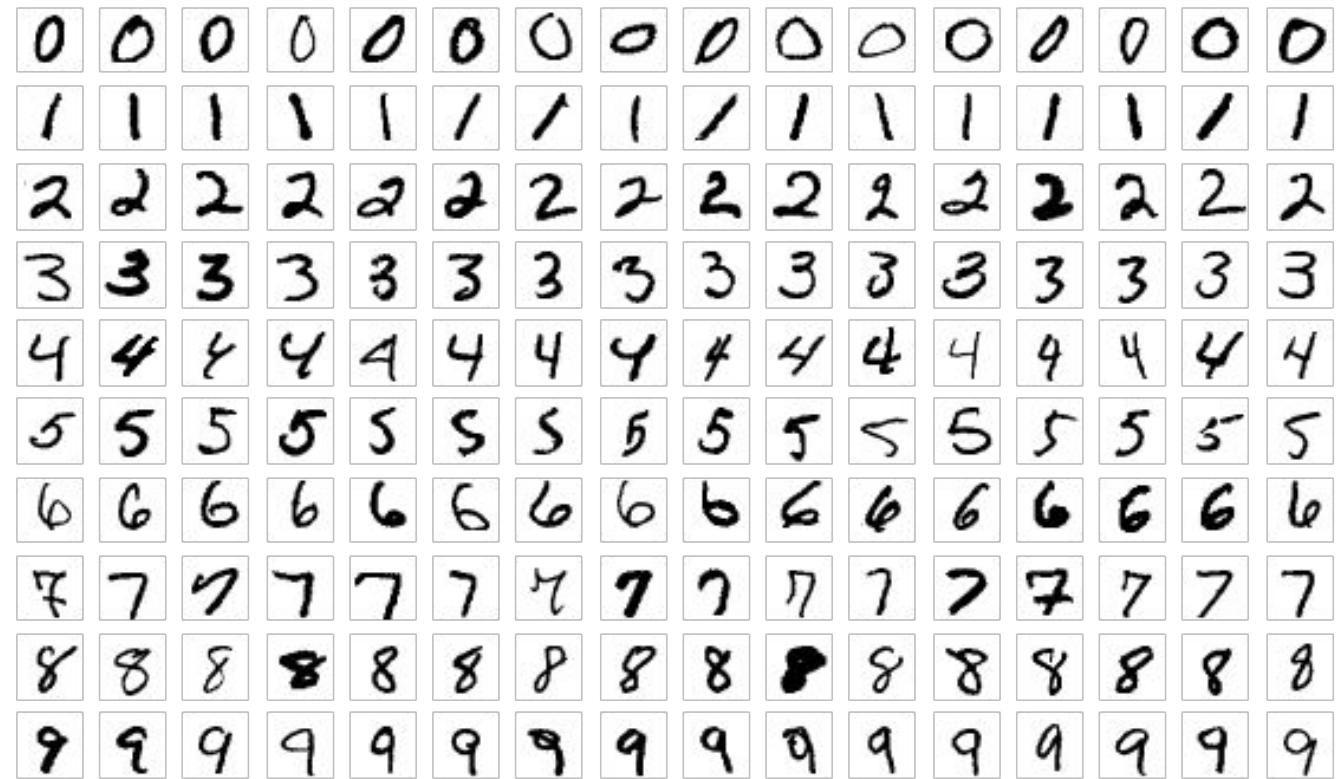
We can extend this example to other domains

-
- 0 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 - 1 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
 - 2 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
 - 3 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
 - 4 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
 - 5 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
 - 6 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
 - 7 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
 - 8 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
 - 9 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]



The **MNIST** (Modified National Institute of Standards and Technology database) is a large database of **handwritten digits** that is **commonly used for training** various image processing systems.

-
- 0 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 - 1 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
 - 2 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
 - 3 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
 - 4 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
 - 5 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
 - 6 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
 - 7 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
 - 8 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
 - 9 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]



60,000 Labelled Training Examples
10,000 Labelled Validation Examples

```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

Collect
Data

```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

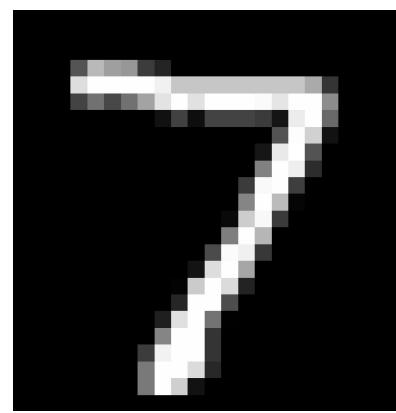
Collect
Data

```
import tensorflow as tf

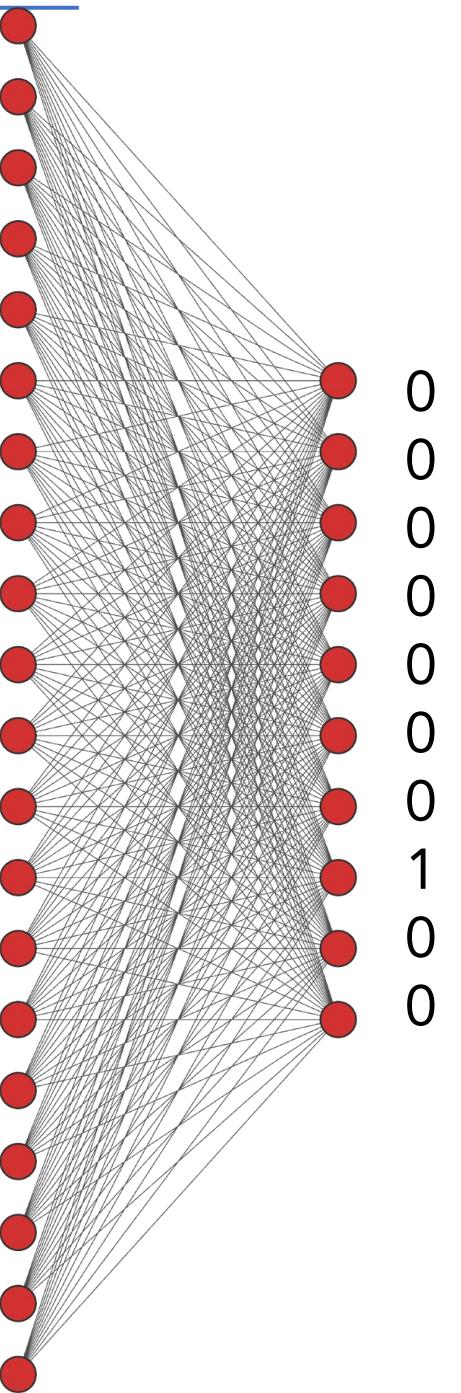
data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

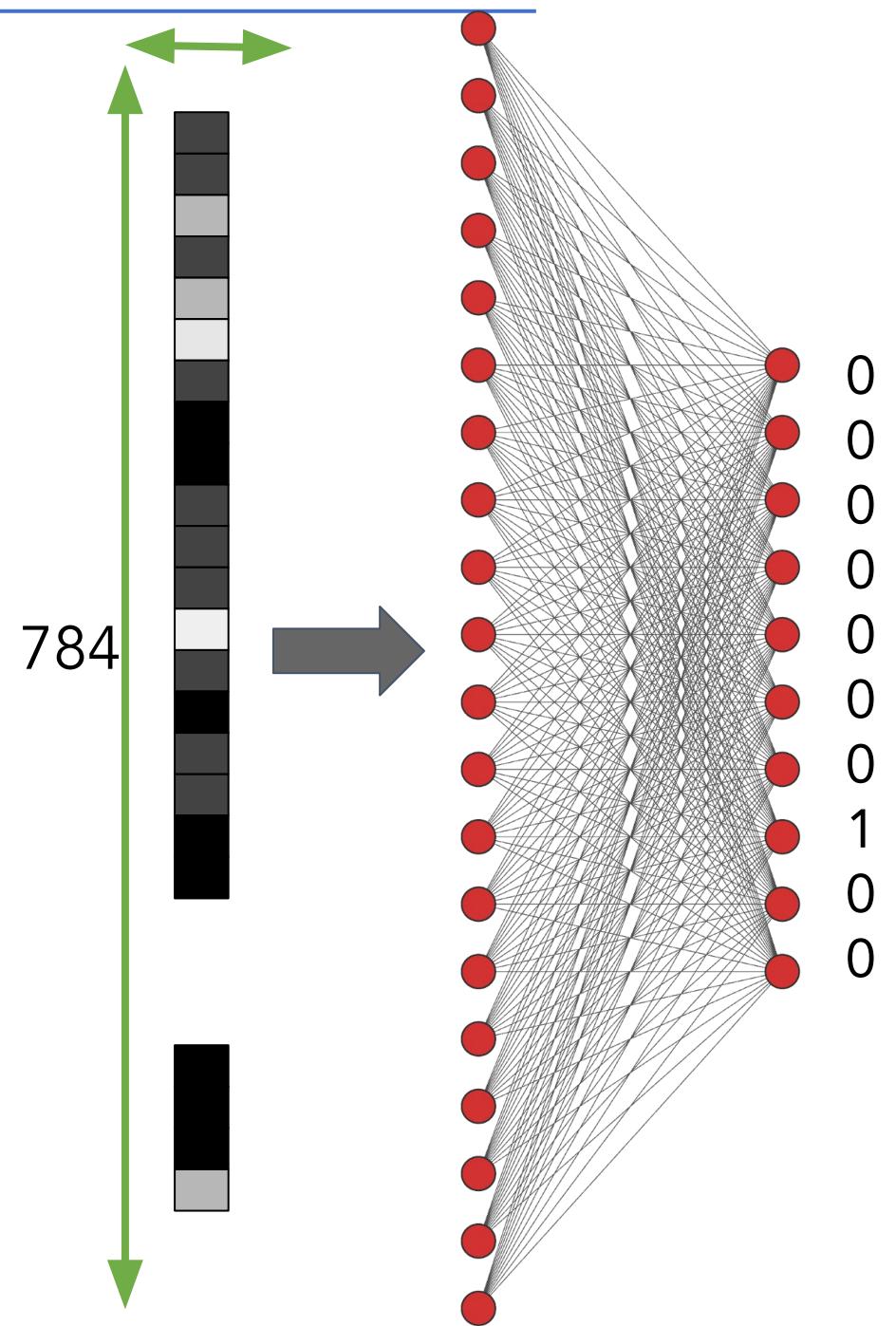
model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```



$28p$
 x



0
0
0
0
0
0
0
1
0
0



```
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

```

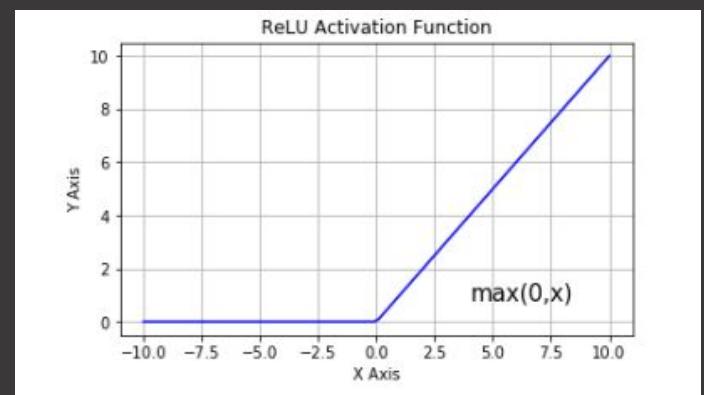
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

```



ReLU applies much-needed non-linearity into the model. Non-linearity is necessary to produce non-linear decision boundaries, so that the output cannot be written as a linear combination of the inputs.

Design a Model

```

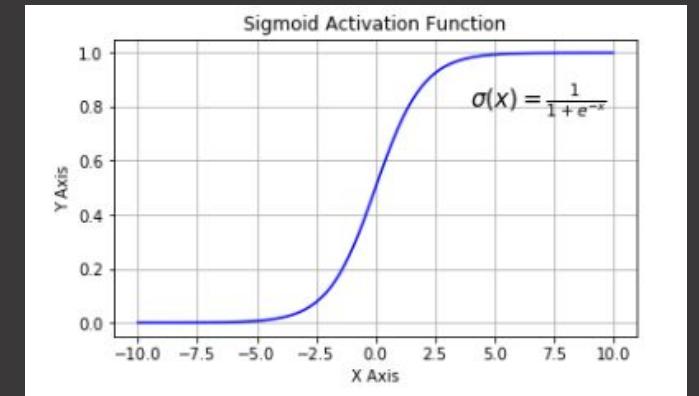
import tensorflow as tf

data = tf.keras.datasets.mnist
(training_images, training_labels), (val_images, val_labels) = data.load_data()

training_images = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential(
    [tf.keras.layers.Flatten(input_shape=(28,28)),
     tf.keras.layers.Dense(20, activation=tf.nn.relu),
     tf.keras.layers.Dense(10, activation=tf.nn.softmax) ])

```



SOFTMAX: Generalization of the logistic function (or Sigmoid) to multiple dimensions. A softmax operation serves a key purpose: making sure the Neural Network (in this case, a DNN) outputs sum to 1. Because of this, softmax operations are useful to scale model outputs into probabilities.

Design a Model

```
model.compile(optimizer='adam' ,  
              loss='sparse_categorical_crossentropy' ,  
              metrics=['accuracy'])
```

```
model.fit(training_images, training_labels, epochs=20)
```

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])  
  
model.fit(training_images, training_labels, epochs=20)
```

Mean Squared Error

$$MSE = \frac{1}{N} \sum (t_i - s_i)^2$$

Prediction
Ground Truth

Cross Entropy Loss

$$CE = - \sum_i^C t_i \log(s_i)$$

Classes
Prediction
Ground Truth {0,1}

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

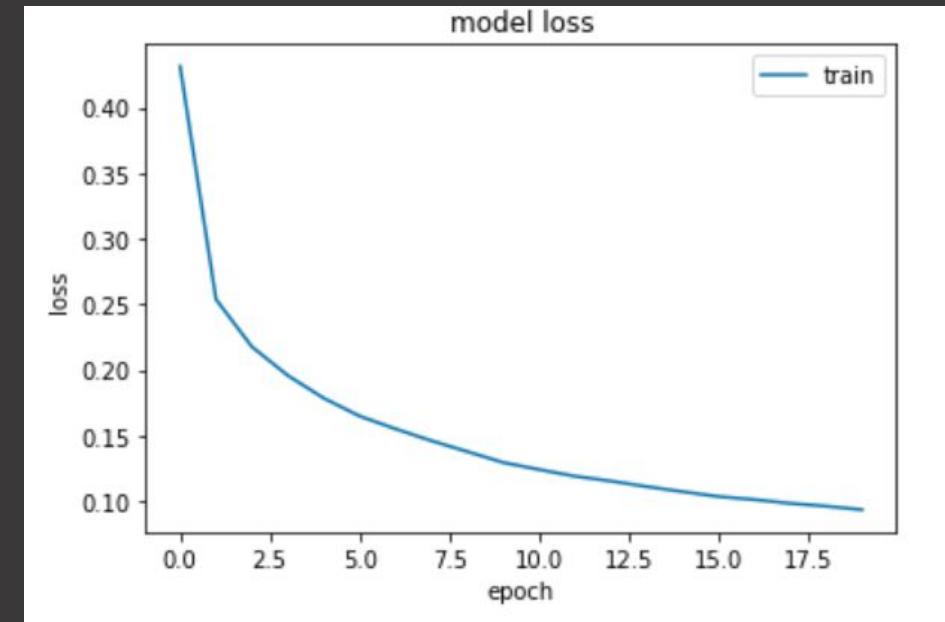
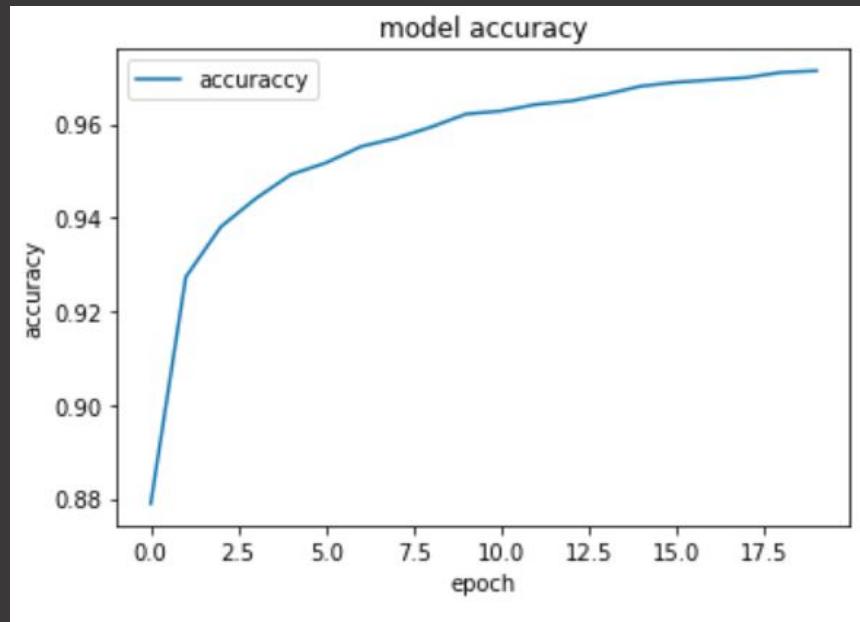
```
model.fit(training_images, training_labels, epochs=20)
```

A yellow arrow pointing to the right, containing the text "Train a Model".

Train a
Model

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

```
model.fit(training_images, training_labels, epochs=20)
```



Evaluate
Optimize

```
classifications = model.predict(val_images)

print(classifications[0])

print(test_labels[0])

[2.4921512e-09 1.3765138e-10 8.8281205e-08 1.0477231e-03 2.8455029e-12
4.0820678e-06 2.0070659e-16 9.9894780e-01 1.0296049e-07 2.9972372e-07]
7
```



Make
Inferences

a NN to classify the MNIST DB

colab.research.google.com
MNIST_NN.ipynb

```
▶ import tensorflow as tf
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (val_images, val_labels) = mnist.load_data()
training_images=training_images / 255.0
val_images=val_images / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(20, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, validation_data=(val_images, val_labels), epochs=20)
```

a NN to classify the MNIST DB

colab.research.google.com
MNIST_NN.ipynb

```
Epoch 9/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3555 - accuracy: 0.8724 - val_loss: 0.4090 - val_accuracy: 0.8516
Epoch 10/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3509 - accuracy: 0.8752 - val_loss: 0.4061 - val_accuracy: 0.8537
Epoch 11/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3452 - accuracy: 0.8768 - val_loss: 0.3980 - val_accuracy: 0.8580
Epoch 12/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3398 - accuracy: 0.8783 - val_loss: 0.4052 - val_accuracy: 0.8586
Epoch 13/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3355 - accuracy: 0.8798 - val_loss: 0.4160 - val_accuracy: 0.8533
Epoch 14/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3332 - accuracy: 0.8812 - val_loss: 0.3913 - val_accuracy: 0.8609
Epoch 15/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3279 - accuracy: 0.8818 - val_loss: 0.3971 - val_accuracy: 0.8588
Epoch 16/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3250 - accuracy: 0.8839 - val_loss: 0.3945 - val_accuracy: 0.8597
Epoch 17/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3221 - accuracy: 0.8839 - val_loss: 0.3985 - val_accuracy: 0.8578
Epoch 18/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3184 - accuracy: 0.8853 - val_loss: 0.3988 - val_accuracy: 0.8595
Epoch 19/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3158 - accuracy: 0.8857 - val_loss: 0.3984 - val_accuracy: 0.8578
Epoch 20/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3140 - accuracy: 0.8856 - val_loss: 0.4069 - val_accuracy: 0.8567
<keras.callbacks.History at 0x7fe50180b750>
```

a NN to classify the MNIST DB

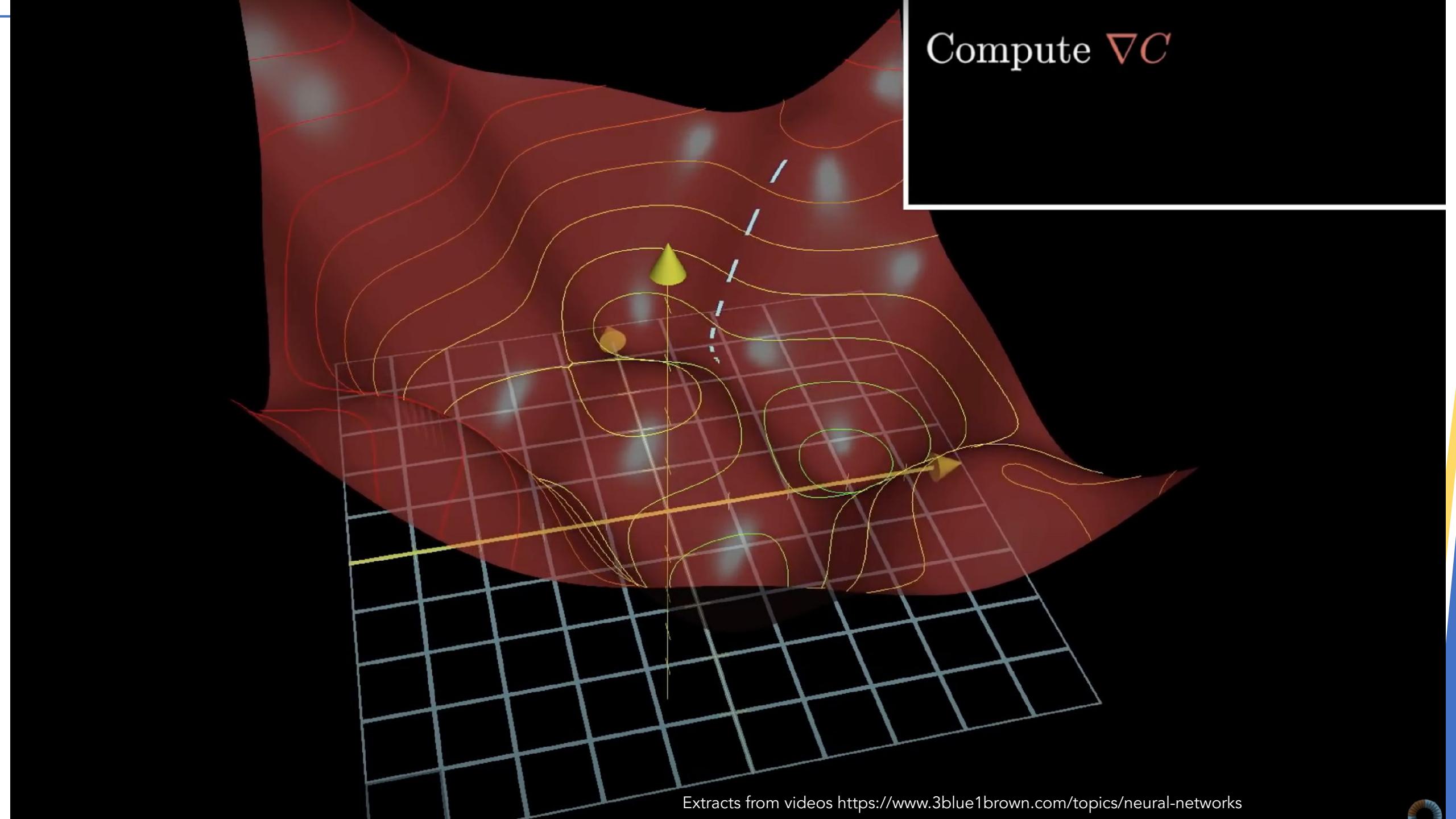
colab.research.google.com
MNIST_NN.ipynb

```
Epoch 19/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.3022 - accuracy: 0.8914 - val_loss: 0.3834 - val_accuracy: 0.8659
Epoch 20/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.2996 - accuracy: 0.8910 - val_loss: 0.3911 - val_accuracy: 0.8642
<keras.callbacks.History at 0x7f033e5f5bd0>
```

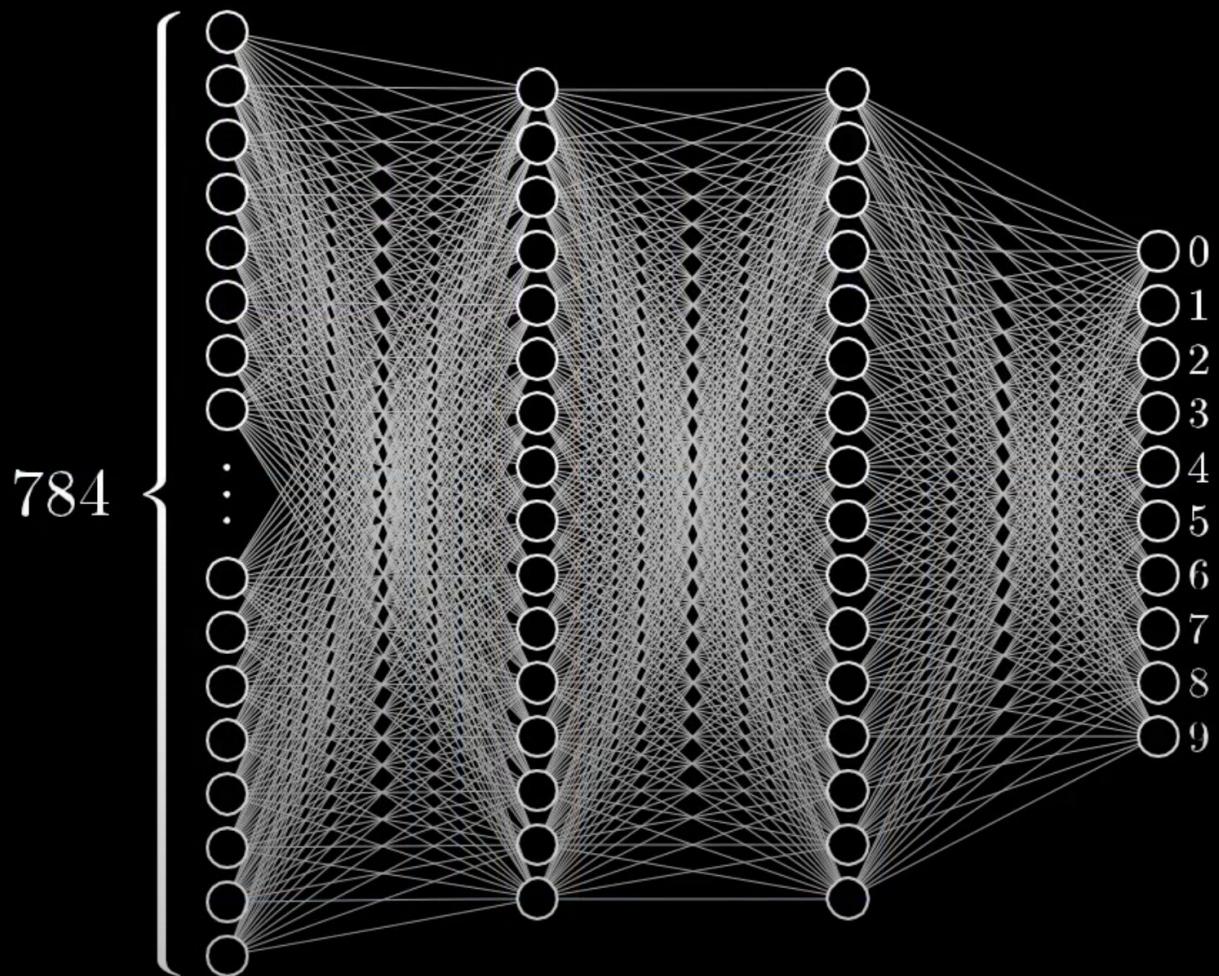
```
▶ model.evaluate(val_images, val_labels)

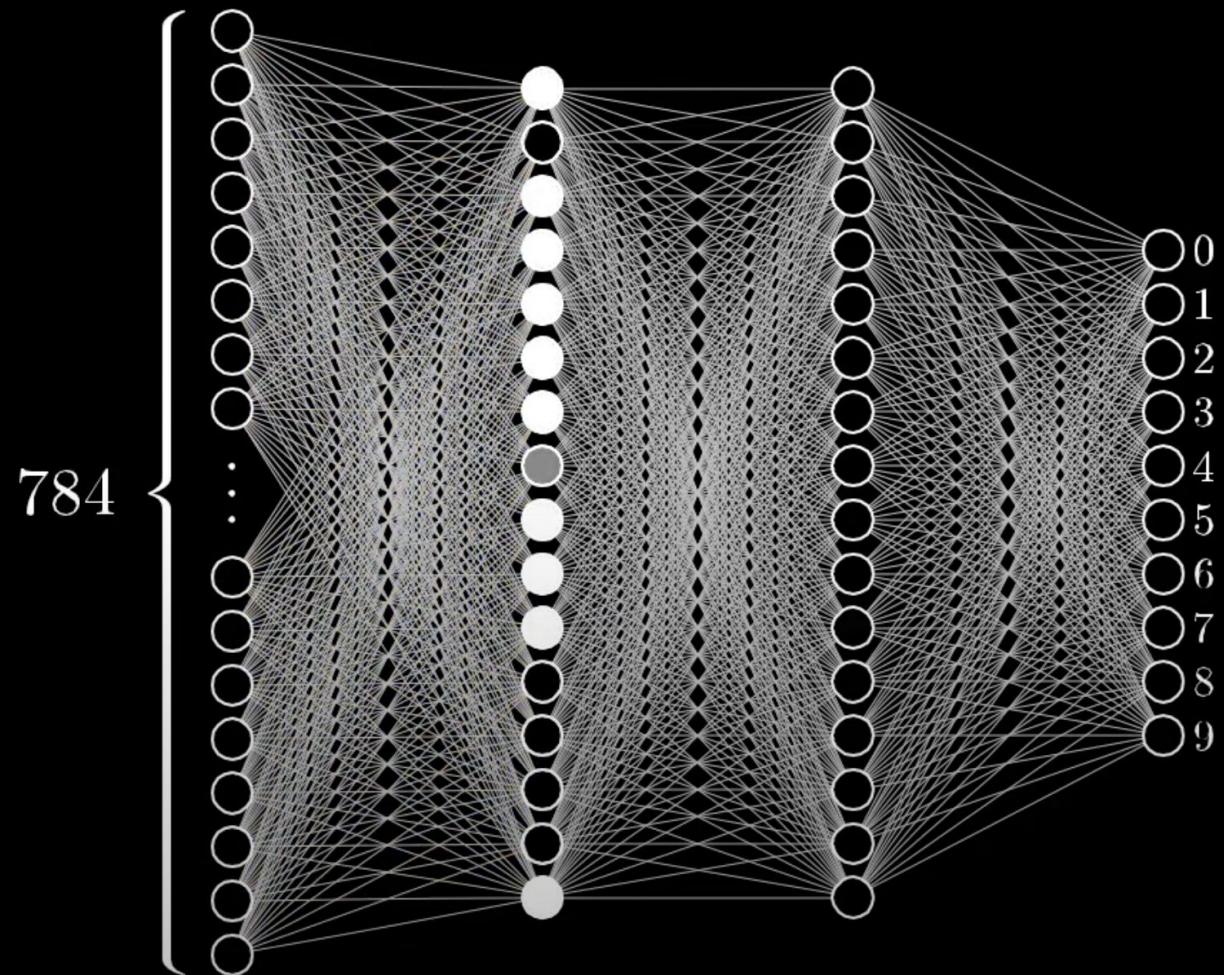
classifications = model.predict(val_images)
print(classifications[0])
print(val_labels[0])
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.3911 - accuracy: 0.8642
[5.2699960e-09 4.4460235e-10 2.9260536e-07 1.1081011e-04 1.4583268e-08
 8.1817927e-03 5.3513944e-09 5.8446459e-02 2.9248906e-05 9.3323141e-01]
```

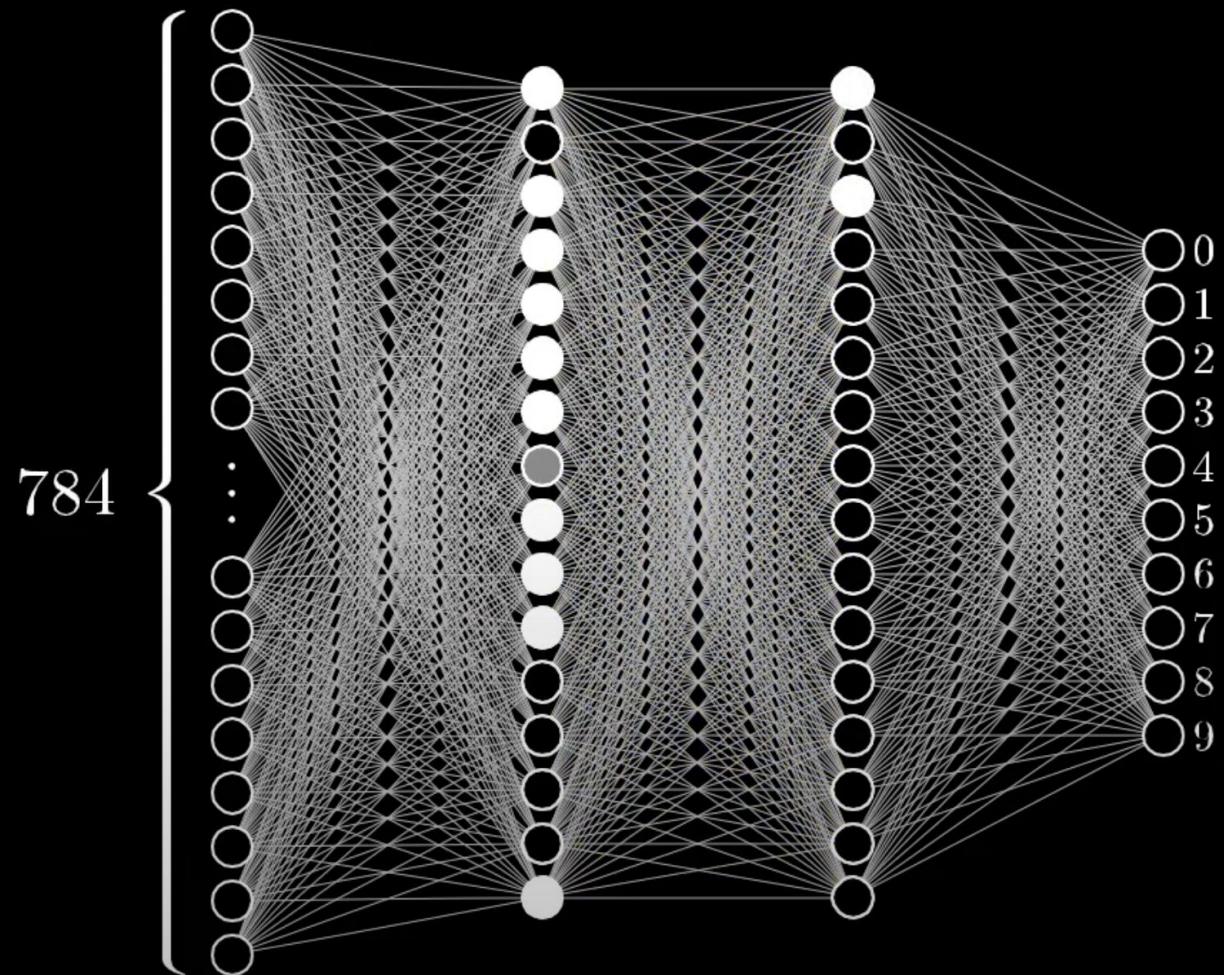


Compute ∇C



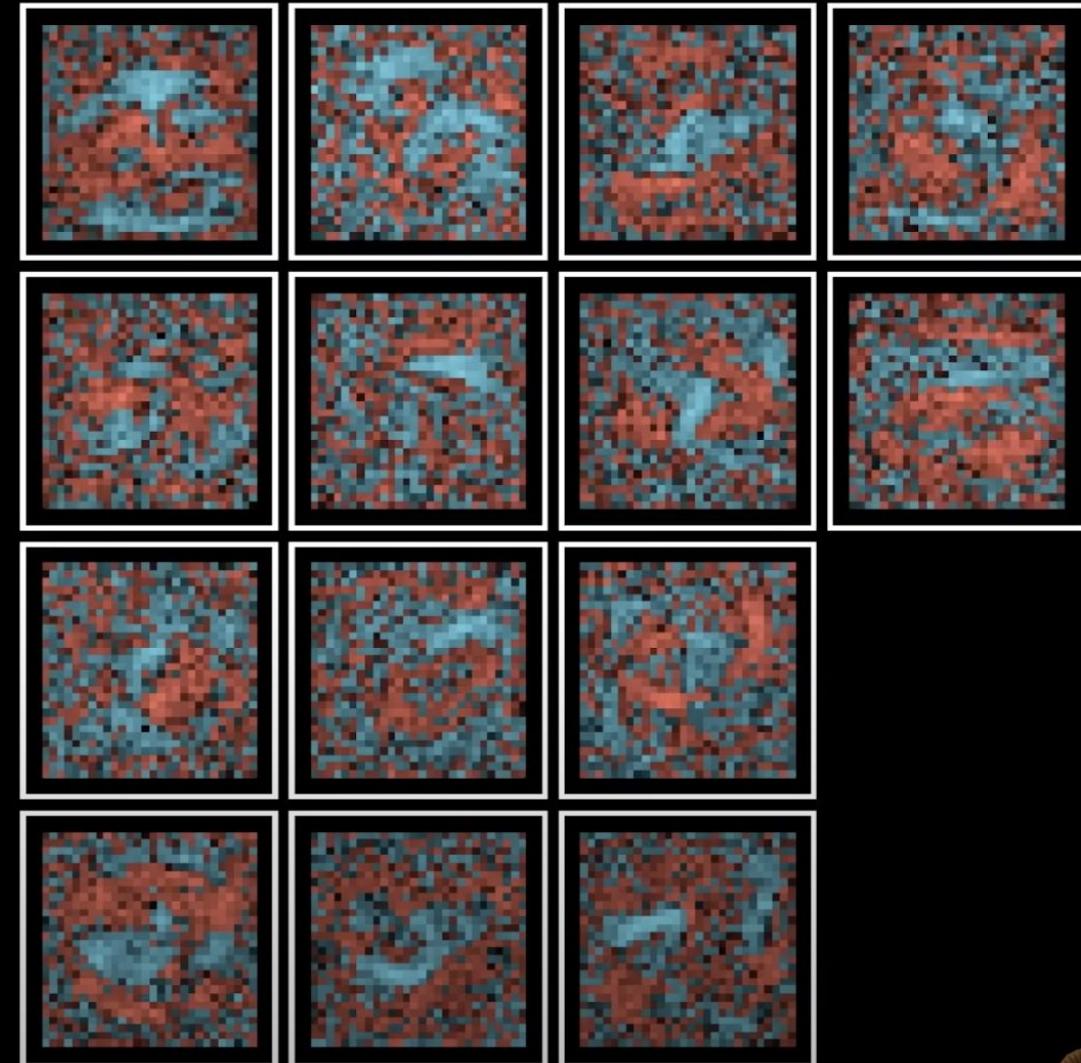
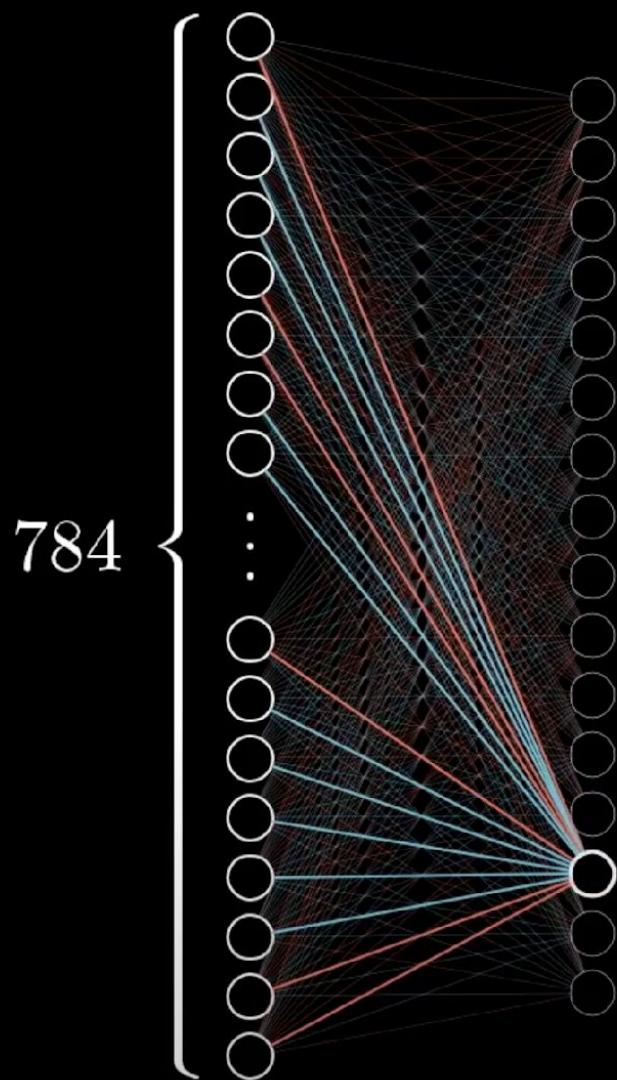


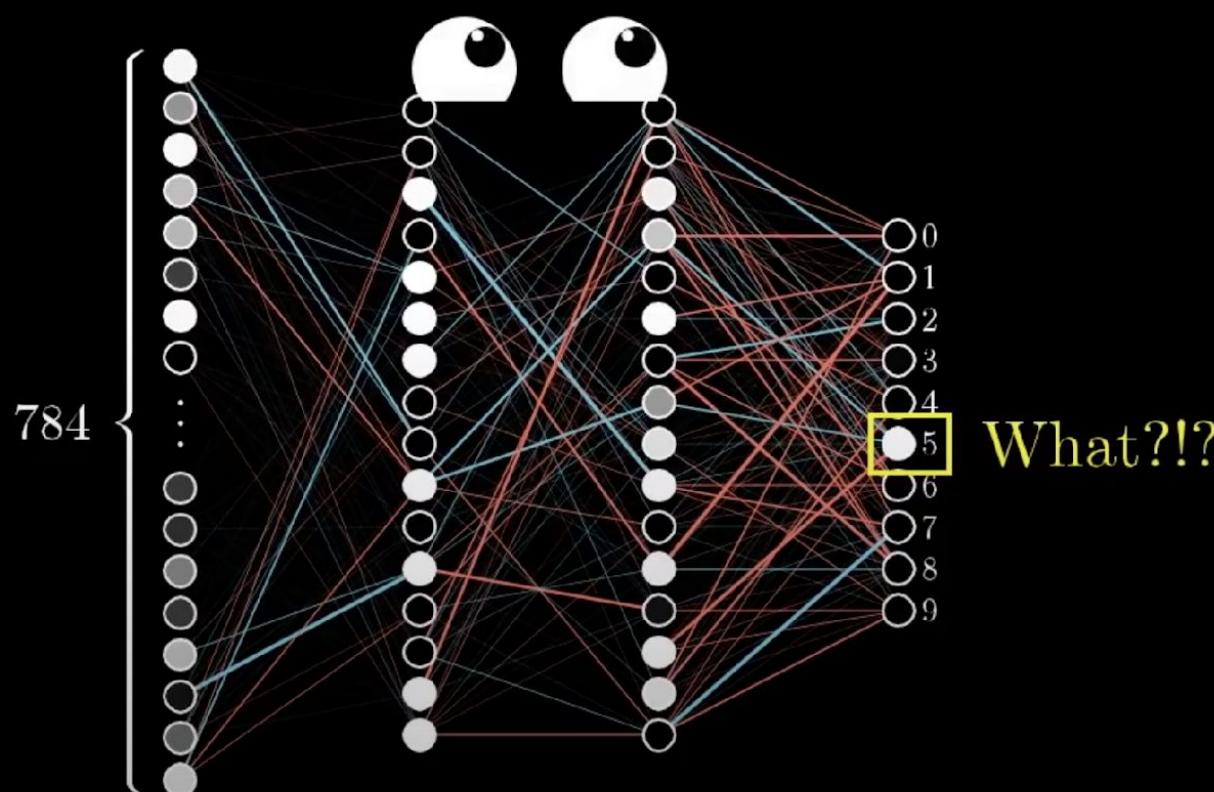
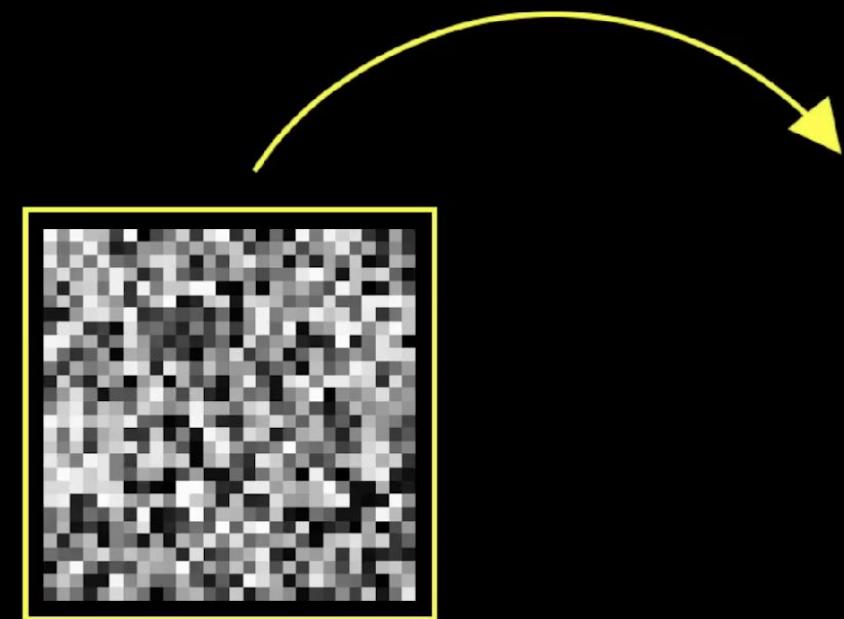
Extracts from videos <https://www.3blue1brown.com/topics/neural-networks>



Extracts from videos <https://www.3blue1brown.com/topics/neural-networks>

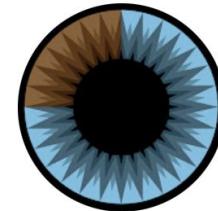
What second layer neurons look for





A very nice introduction to NN

- 3Blue1Brown playlist on Neural Networks
 - But what is a neural network?
 - Chapter 1 – Deep learning
 - <https://youtu.be/aircAruvnKk>
 - Gradient descent, how neural networks learn
 - Chapter 2 – Deep learning
 - <https://youtu.be/IHZwWFHWa-w>
 - What is backpropagation really doing?
 - Chapter 3 – Deep learning
 - <https://youtu.be/lIg3gGewQ5U>
 - (Optional) Backpropagation calculus
 - Chapter 4 – Deep learning
 - <https://youtu.be/tleHLnjs5U8>



3Blue1Brown •

@3blue1brown 5.09M subscribers 129 videos

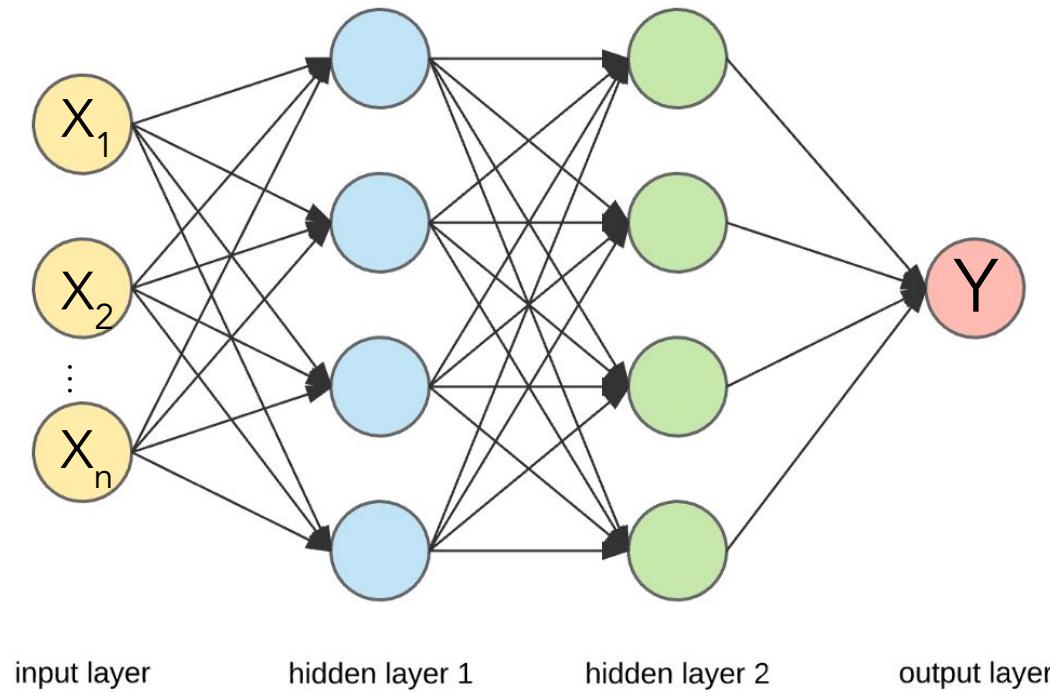
3Blue1Brown, by Grant Sanderson, is some combi



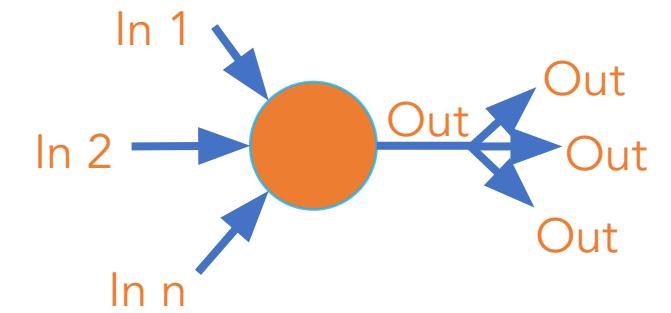
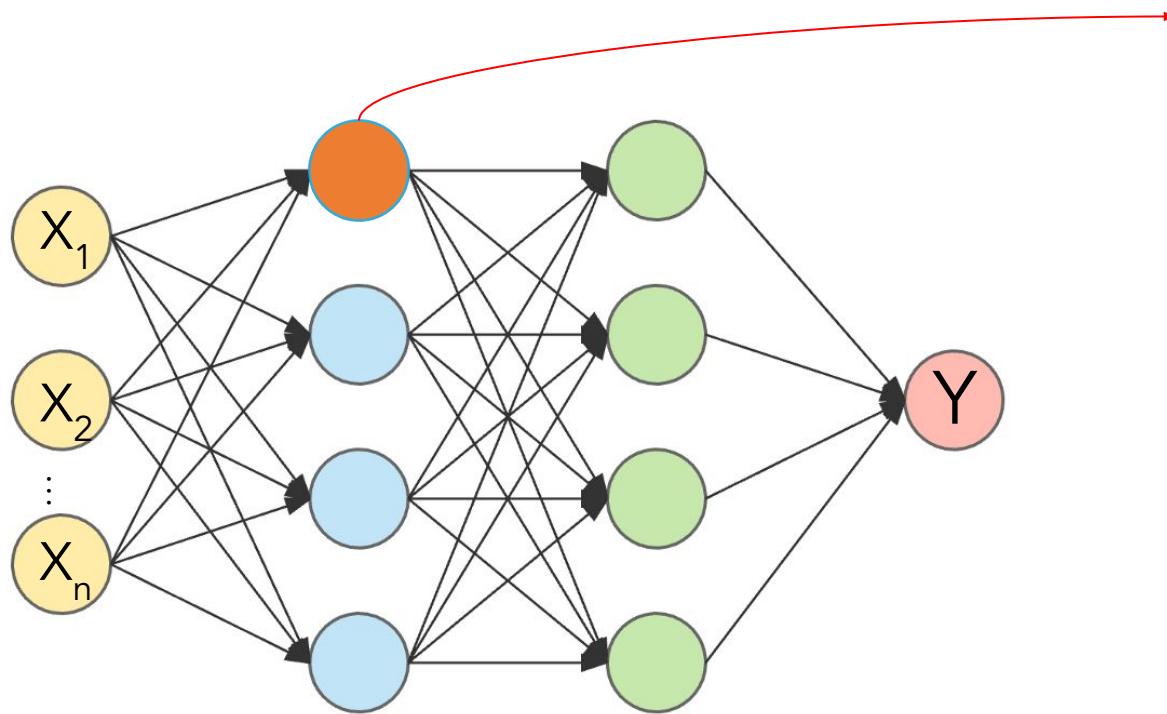
<https://www.3blue1brown.com/topics/neural-networks>

... in summary

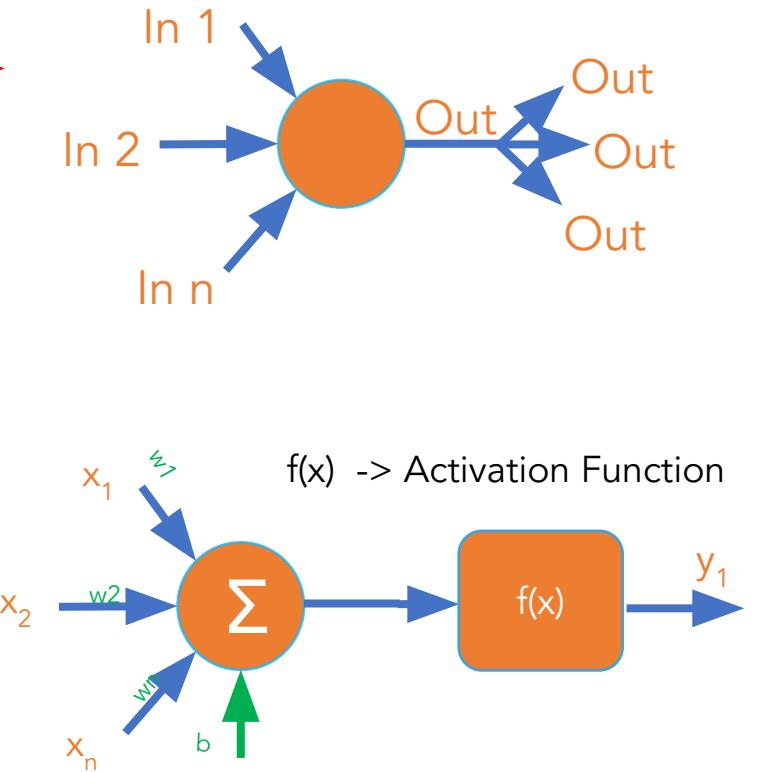
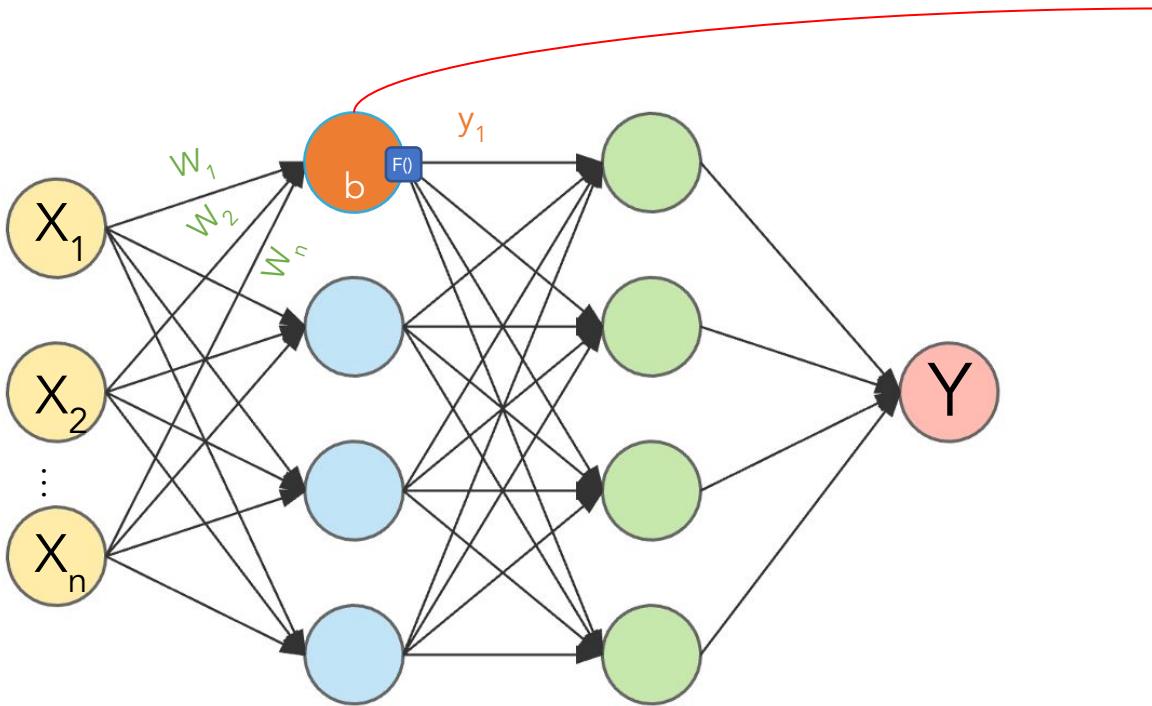
Dense Neural Networks – DNNs



DNNs

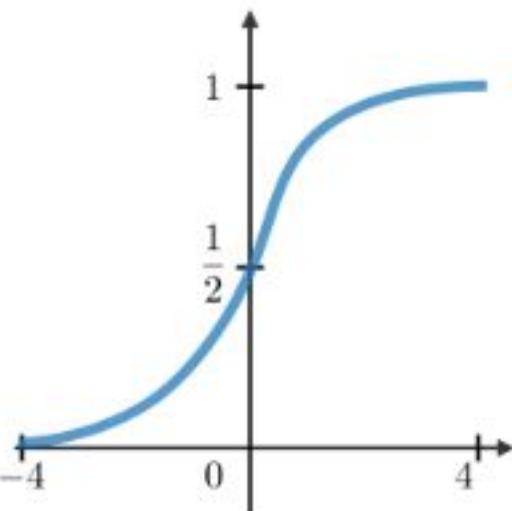
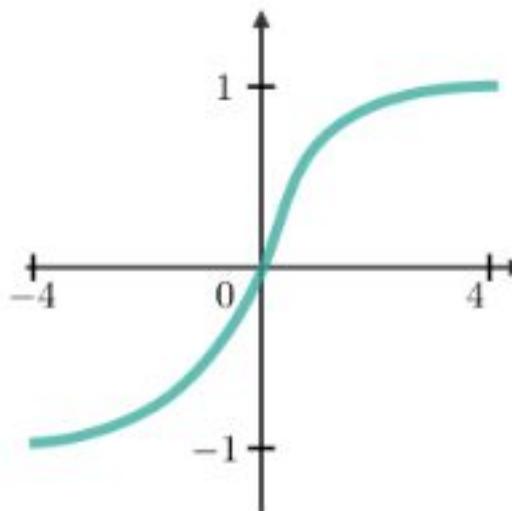
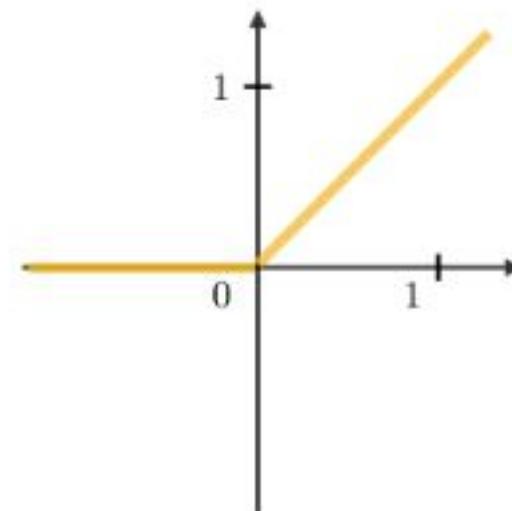


DNNs

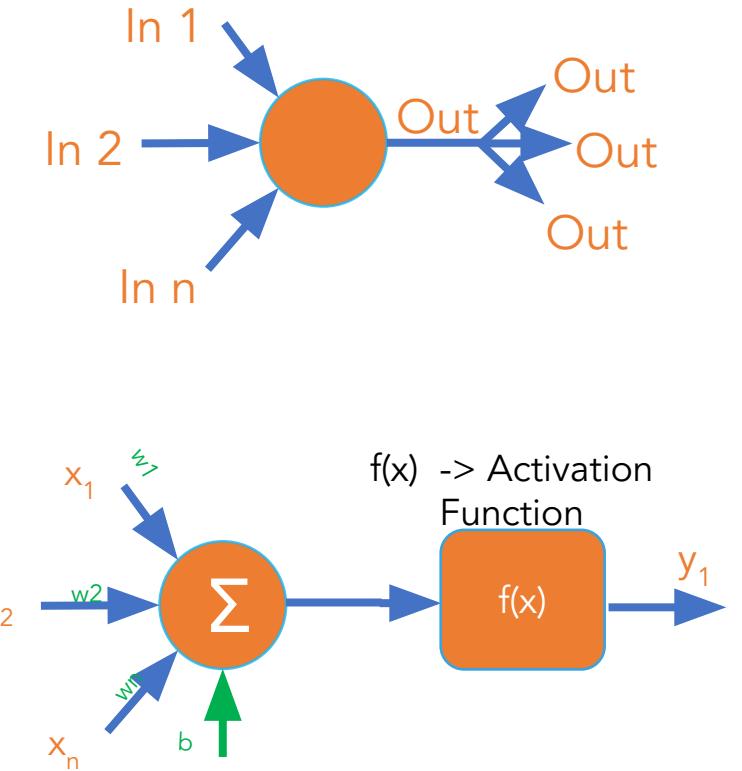
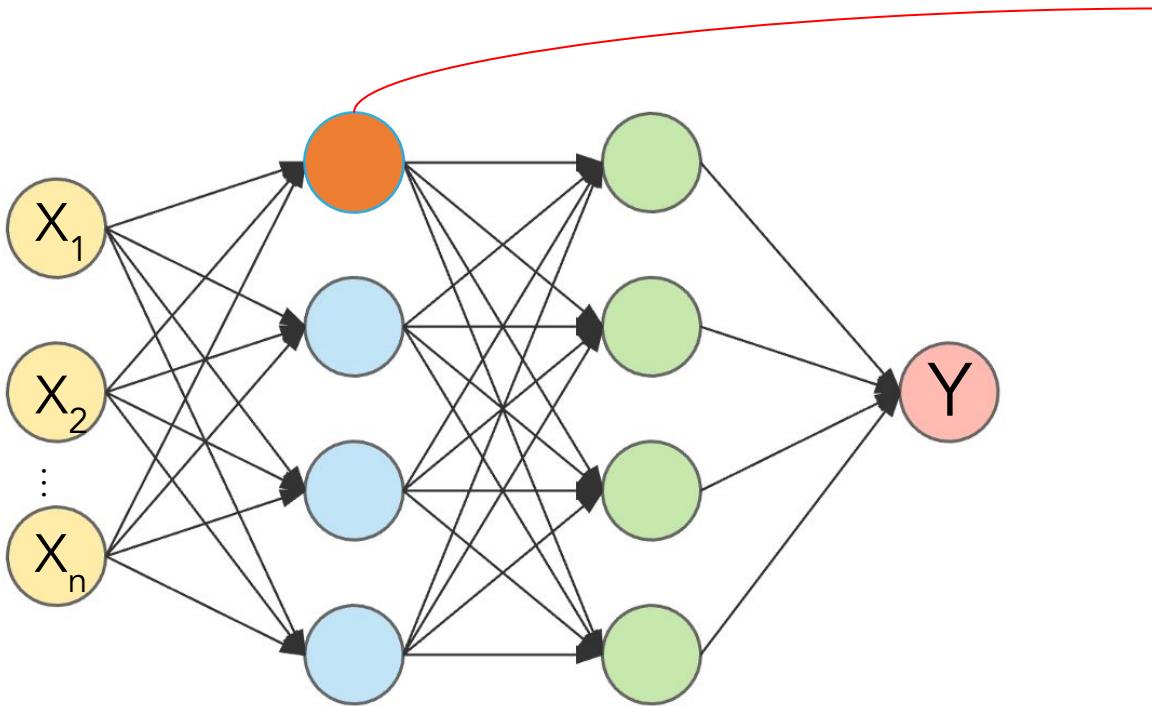


$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

DNNs – Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

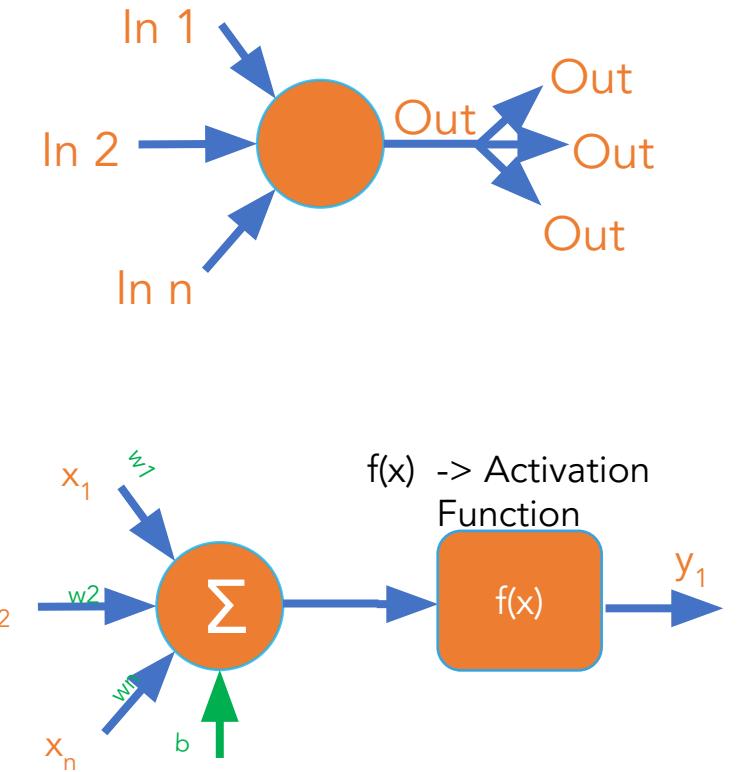
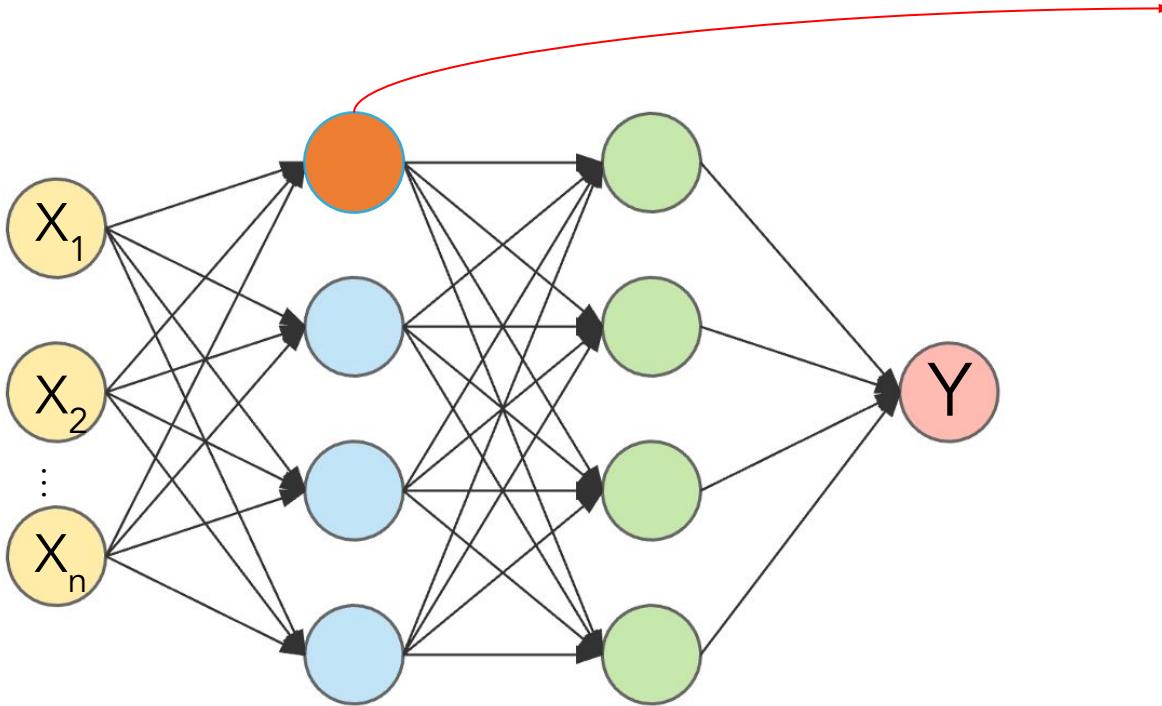
DNNs



Parameters to be found during
training, to reach minimum error

$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

DNNs

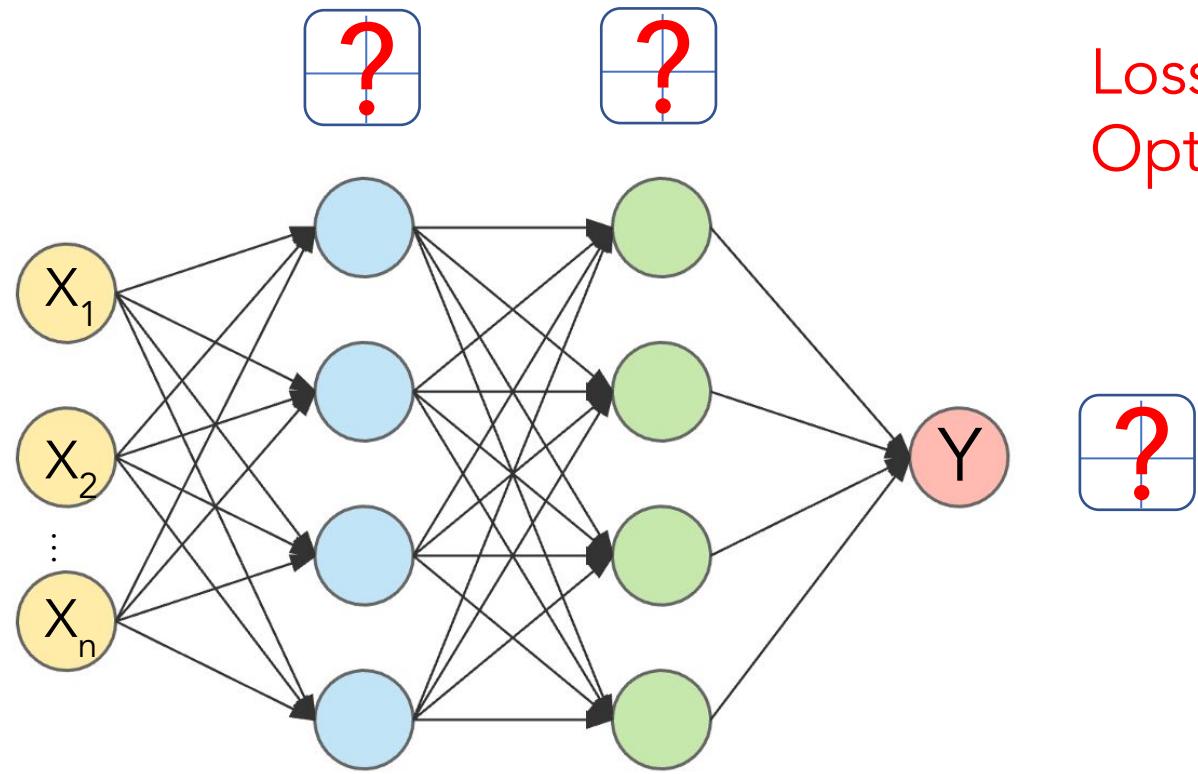


- Error Measurement (Loss)
- Optimization

Parameters to be found during training, to reach minimum error

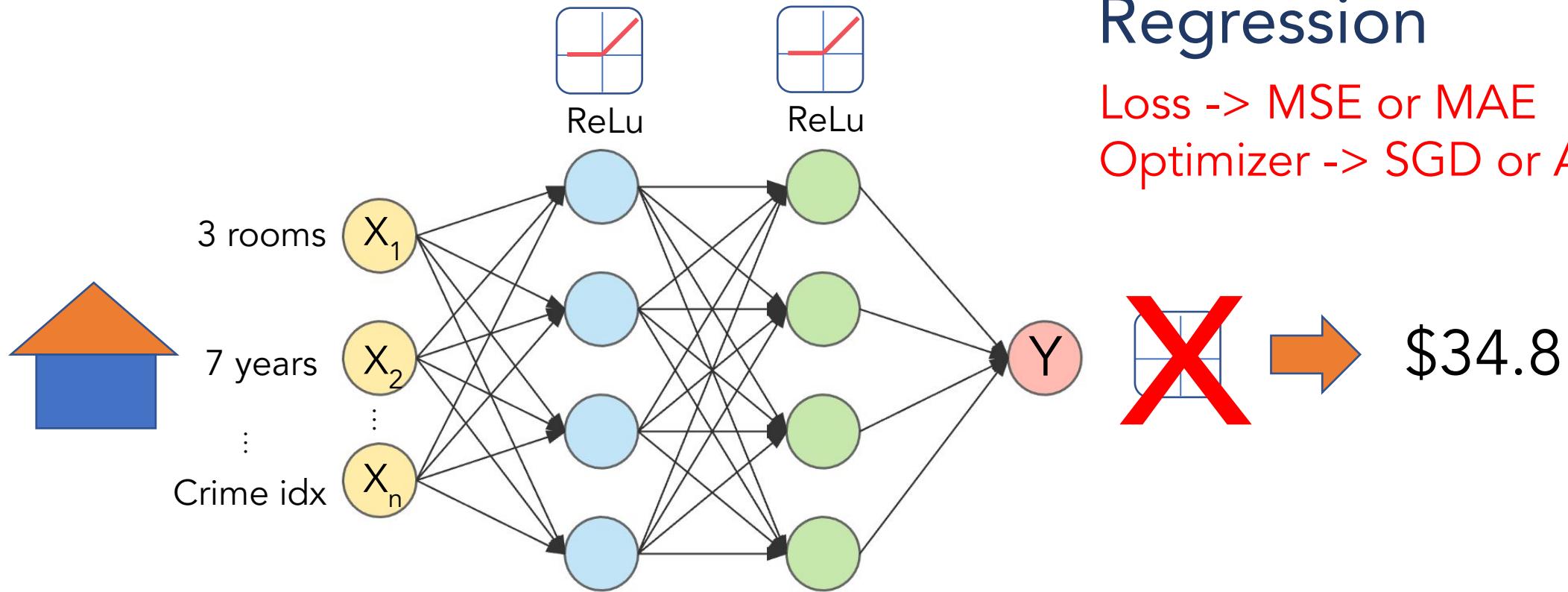
$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

DNNs



Loss -> ?
Optimizer -> ?

DNNs



Regression

Loss -> MSE or MAE

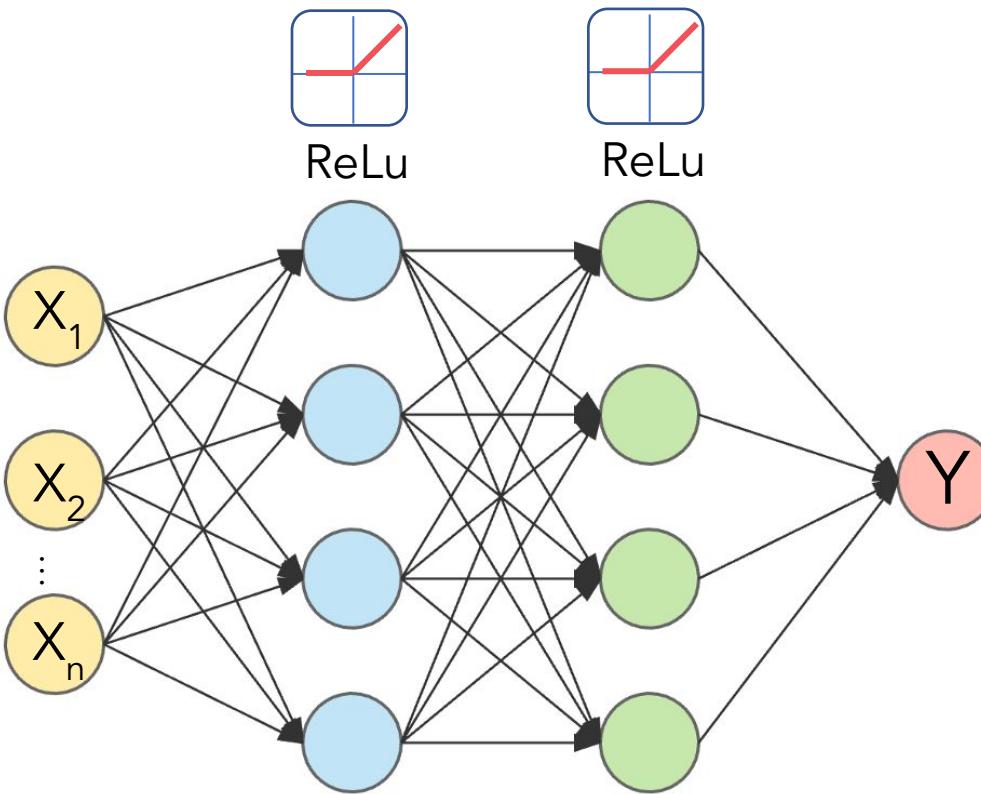
Optimizer -> SGD or Adam

DNNs



(28,28)

Flatten
(784)



Binary Classification

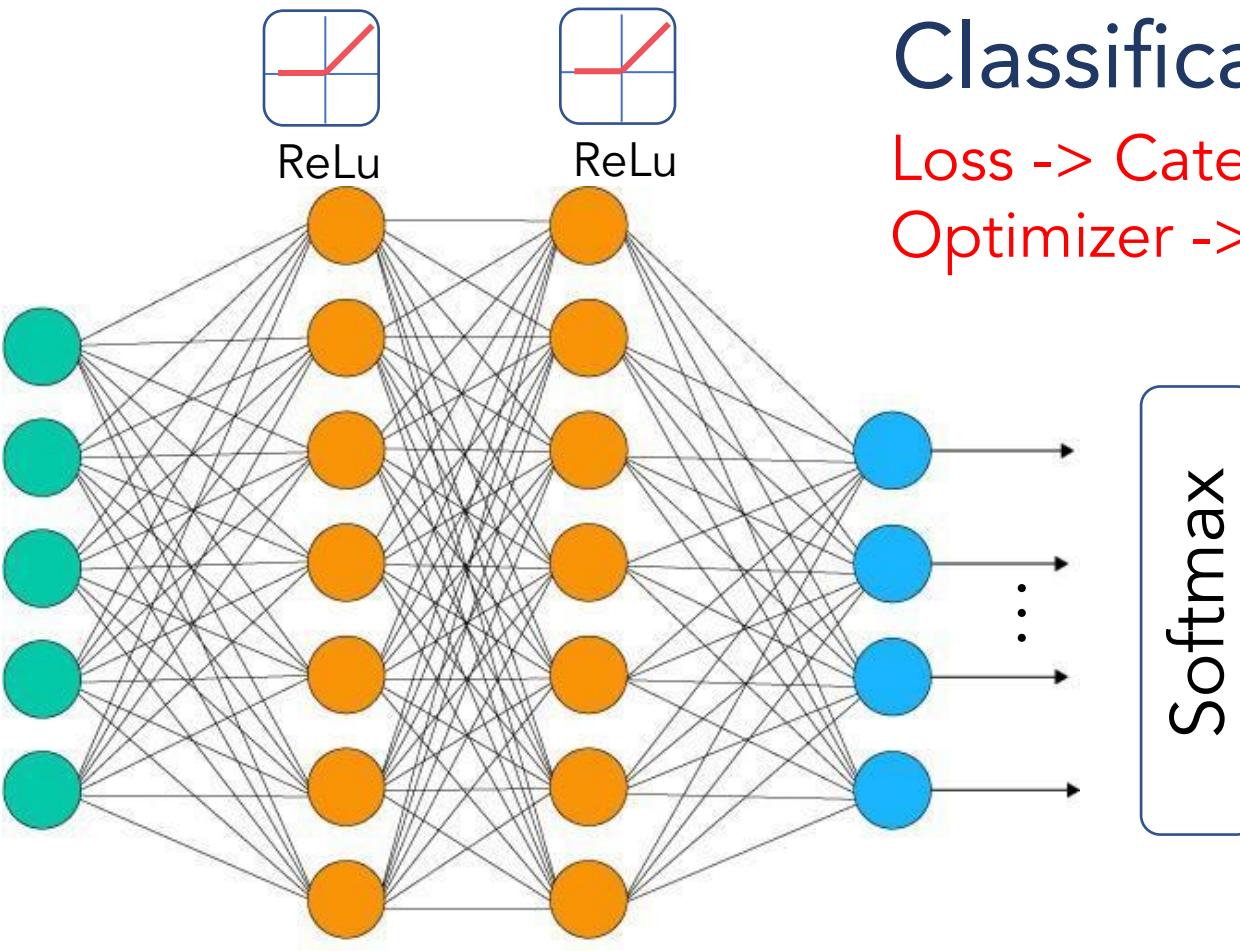
Loss -> Binary Crossentropy
Optimizer -> SGD or Adam

0: Cat
1: Dog

DNNs



Flatten



Multi-class Classification

Loss -> Categorical Crossentropy *

Optimizer -> SGD or Adam

* or "Sparse Categorical Crossentropy" if label is 1, 2, 3, ...

and some issues?





Steps to take

- Get as many examples of shoes as possible
- Train using these examples
- Profit!





Steps to take

- Get as many examples of shoes as possible

- Train using these examples

- Profit!

Training accuracy: .920

Training accuracy: .935

Training accuracy: .947

Training accuracy: .961

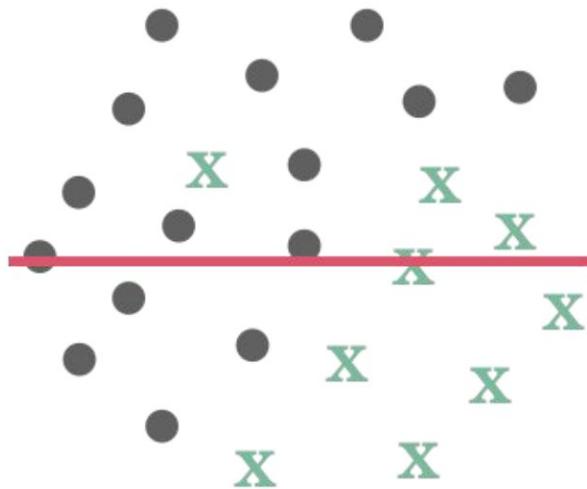
Training accuracy: .977

Training accuracy: .995

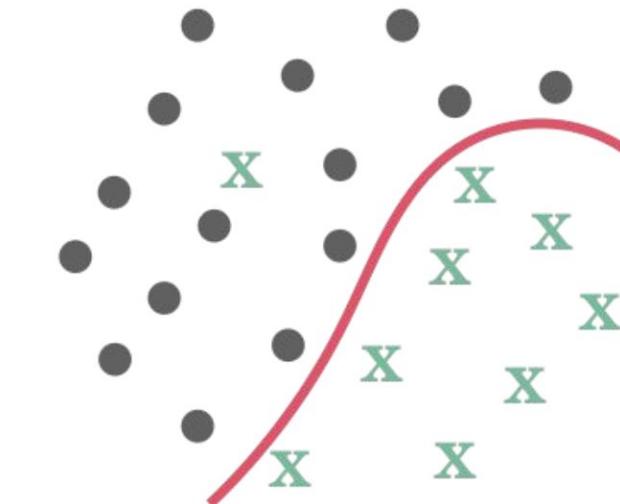
Training accuracy: 1.00

Data

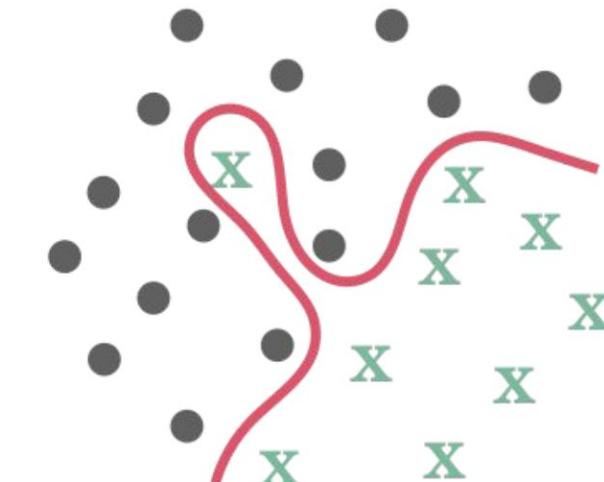
The network '**sees**' everything. Has no context for measuring how well it does with data it has never previously been exposed to.



Underfitting



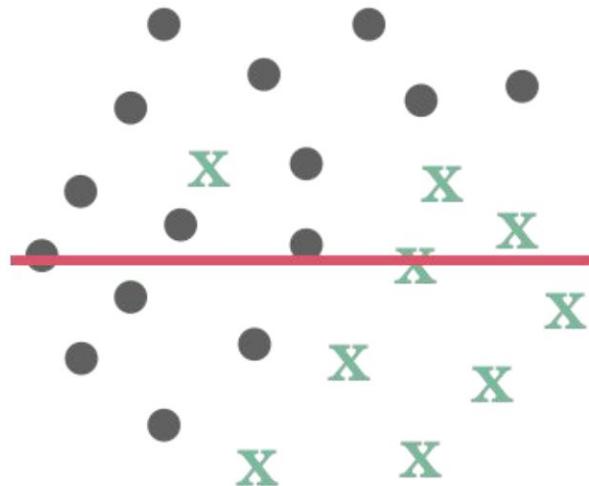
Desired



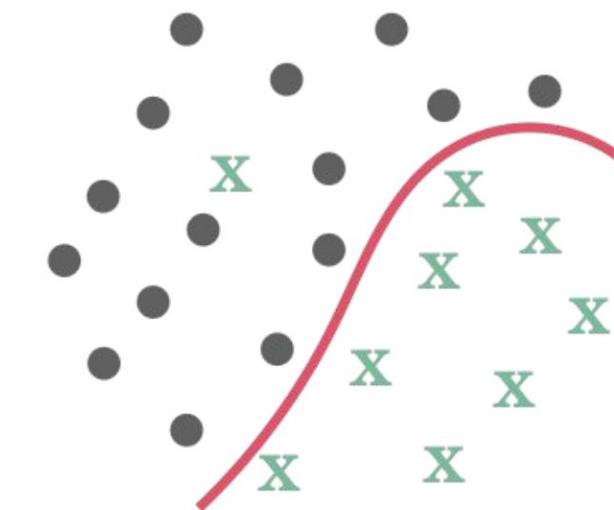
Overfitting

Correct vs. Overfit Model

Model fitting refers to the accuracy of the model's underlying function as it attempts to analyze data with which it is not familiar.

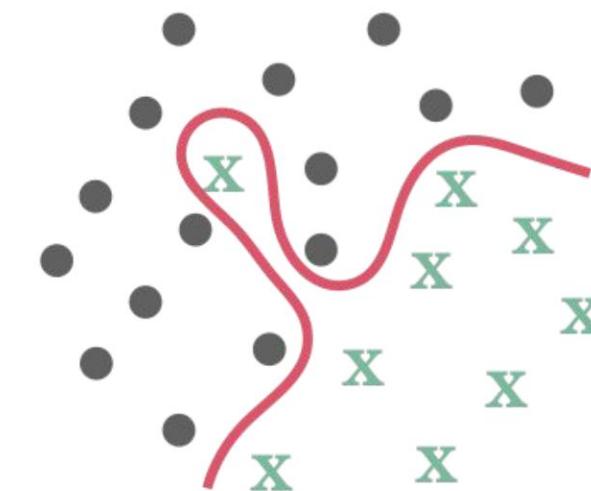


Underfitting



Desired

Underfitting and **overfitting** are common problems that degrade the quality of the model, as the model fits either not well enough or too well.



Overfitting



Data

Validation Data

The network '**sees**' a subset of **your data**. You can use **the rest** to measure its performance against previously unseen data.

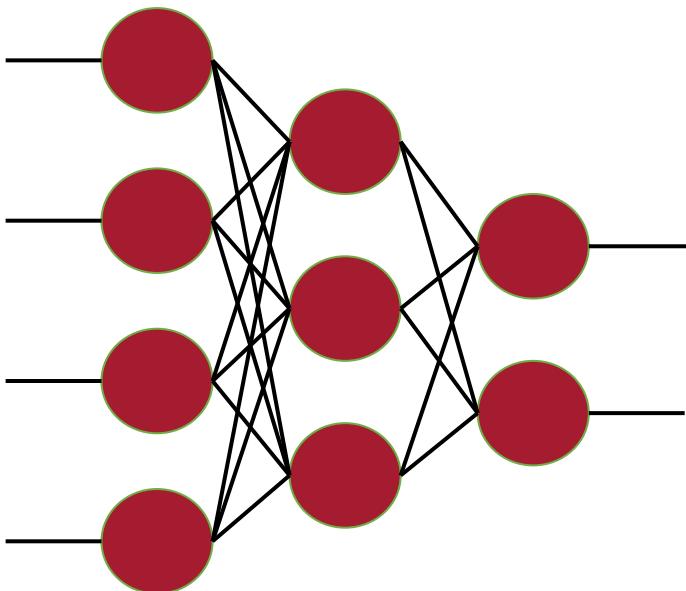
Data

Validation Data

Test Data

The network '**sees**' a subset of **your data**. You can use an **unseen subset to measure its accuracy while training (validation)**, and then **another subset to measure its accuracy after it's finished training (test)**.

Data



Validation Data

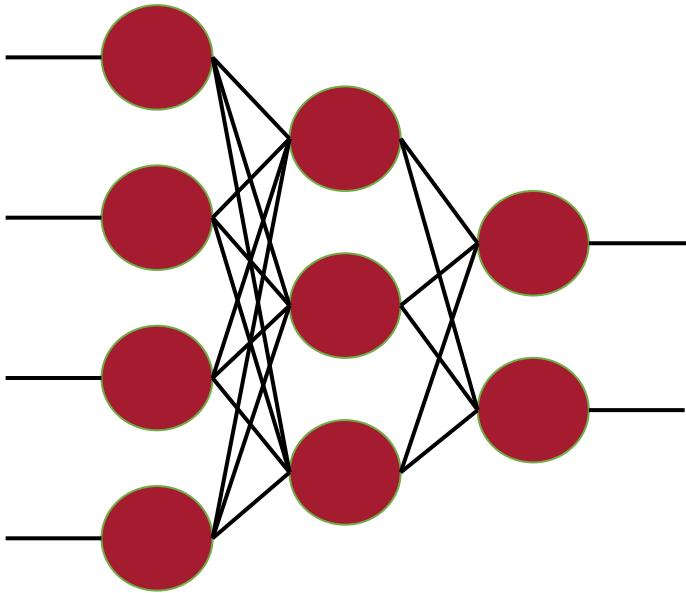
Accuracy:
0.999

Test Data

Accuracy:
0.920

Accuracy:
0.800

Data



Accuracy:
0.999

Validation Data

Accuracy:
0.920

Test Data

Accuracy:
0.800

Data

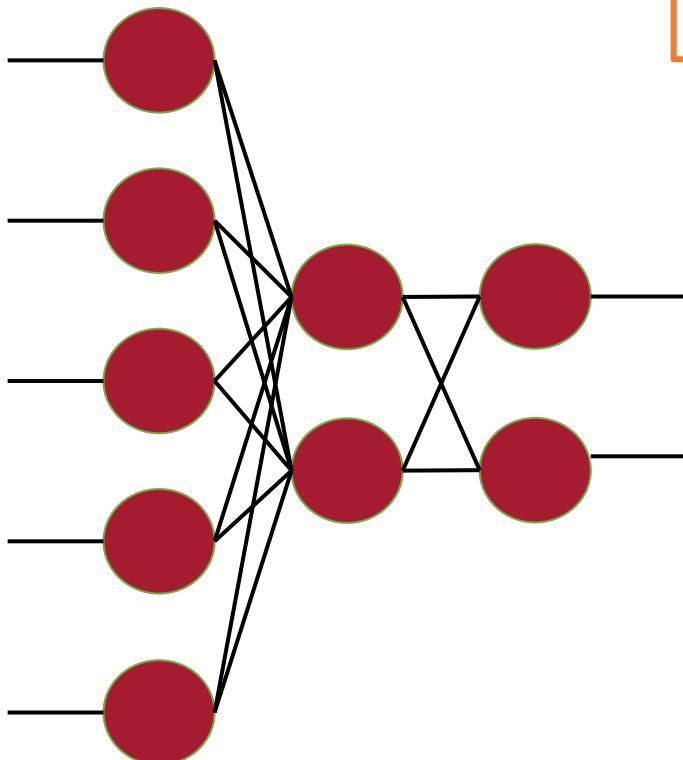
Validation Data

Test Data

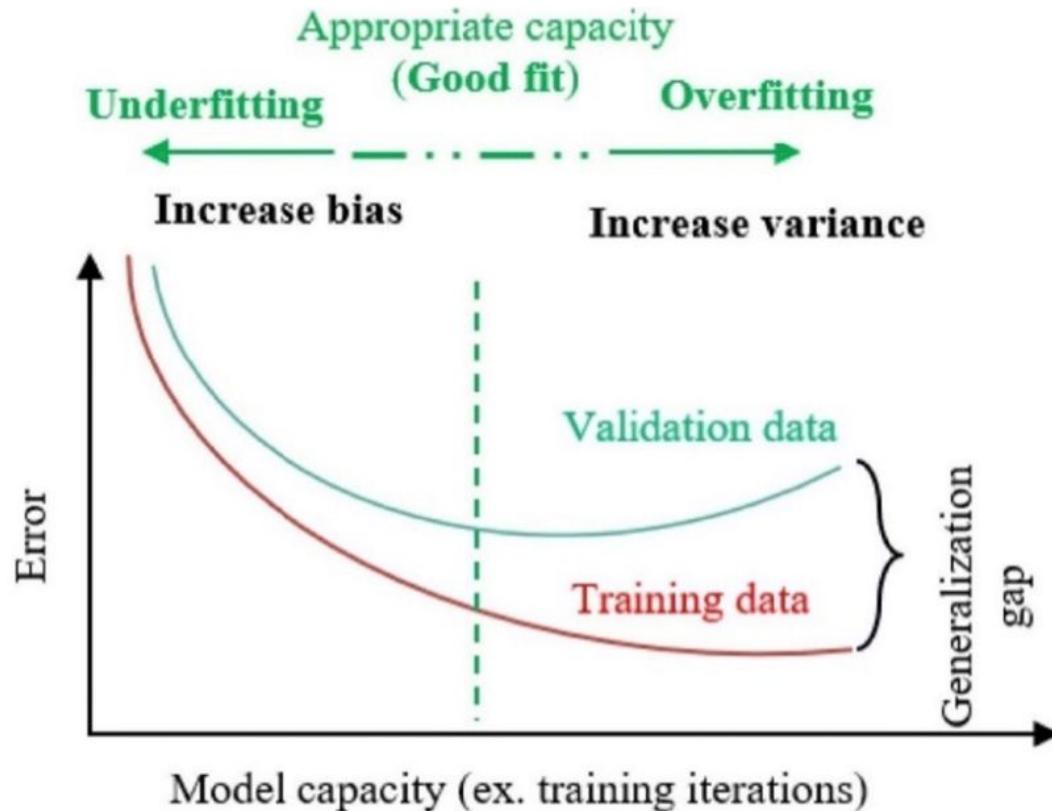
Accuracy:
0.942

Accuracy:
0.930

Accuracy:
0.925

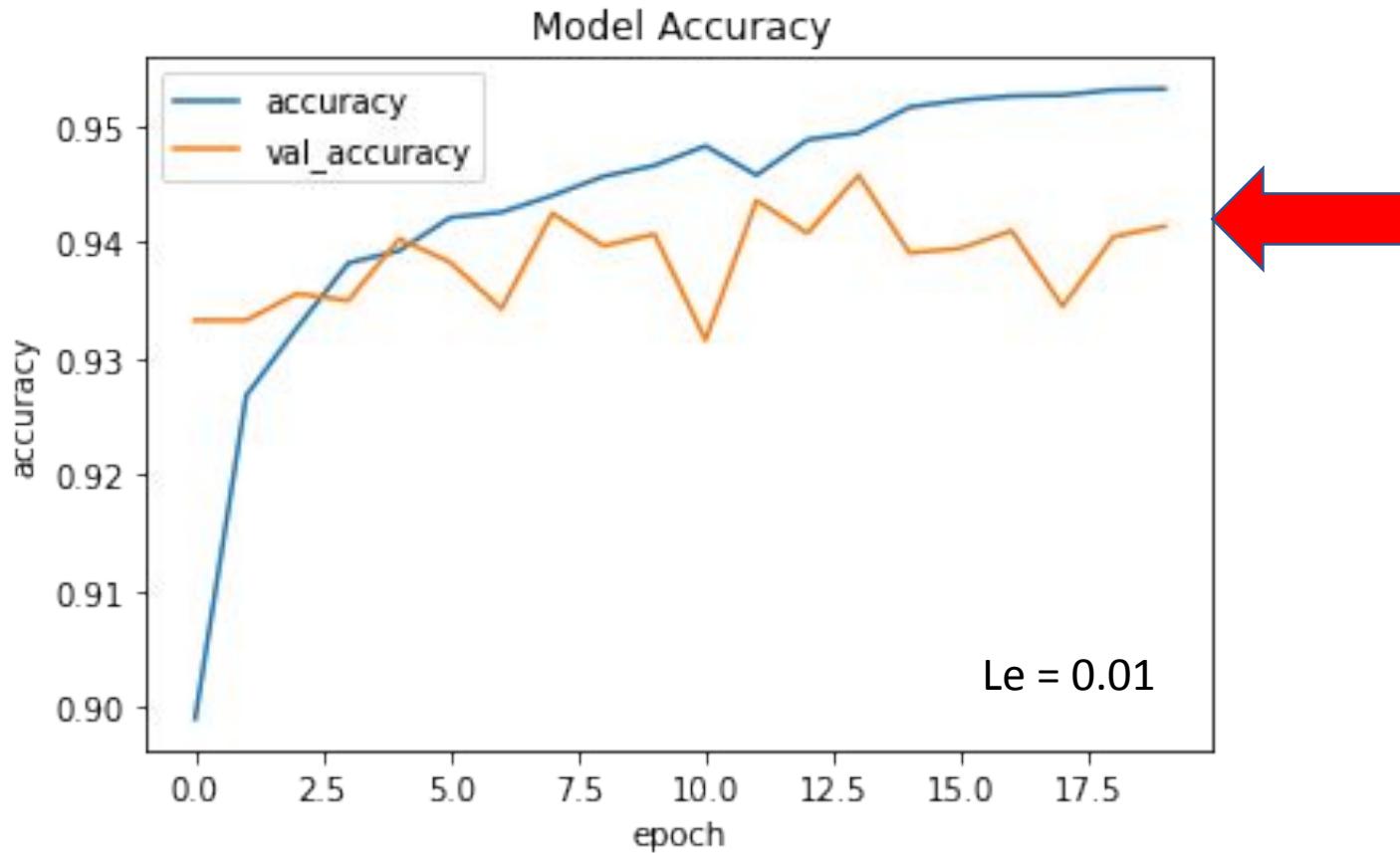


Prevent Overfitting and Imbalanced Data



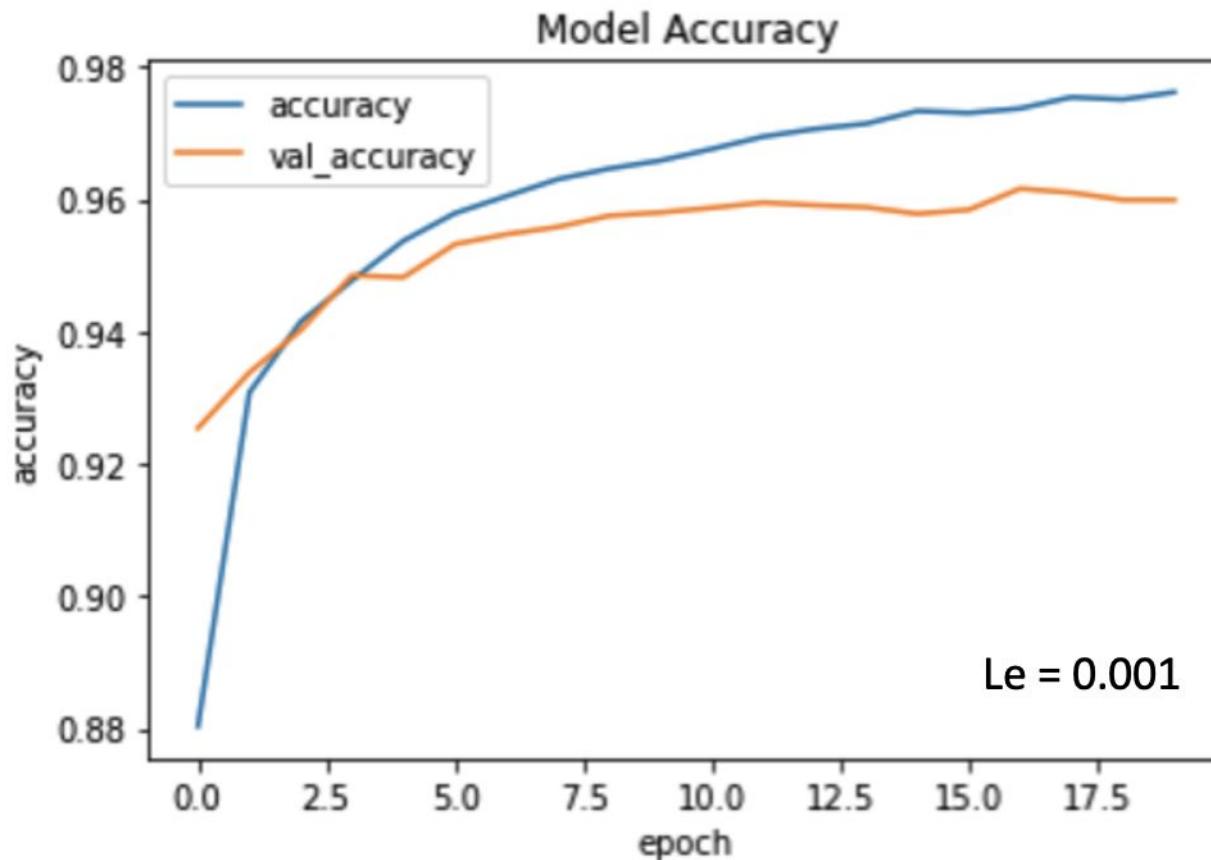
Training loss	Validation loss	Situation	Solution
High	High	Underfitting	Increase capacity
Low	High	Overfitting	Decrease capacity (shrink or regularization techniques)
Low	Low	Good fit	Run test
High	Low	Unlikely	Debug

Prevent Overfitting and Imbalanced Data



If validation accuracy seems unstable, could be that **Learning Rate** is high (try to reduce it).

Prevent Overfitting and Imbalanced Data



If validation accuracy goes down (or became stable), even if train accuracy goes up, means that probably the model is **overfitting**. In this case the training (epochs) should terminate.

In summary

Model	Train Accuracy	Test Accuracy
A	99,9%	95%
B	87%	87%
C	99,9%	45%

Test accuracy should be lower than train accuracy, but **how much less accurate?**

Model A is better than model B because it has a higher test accuracy, regardless its difference with the train accuracy.

Model C is a clear case of overfitting as the train accuracy is very high but the test accuracy isn't anywhere near as high.

This **distinction is subjective**, but comes from knowledge of your problem and data, and **what magnitudes of error are acceptable**.

In summary

Training Data -> Used to train **model parameters**

Validation Data -> Used to determine what **model hyperparameters** to adjust (and re-training)

Test Data -> Used to get **model final performance metric**

Classification Model Performance Metrics



Class = [1]

actual = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]



Class = [0]

prediction = [0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]



Confusion Matrix

		predicted condition	
		Cat [1]	Dog [0]
true condition	12 pictures, 8 of cats and 4 of dogs	Cat [1]	Dog [0]
	Cat [1]	6	2
Dog [0]		1	3

Confusion Matrix

		predicted condition	
		Cat [1]	Dog [0]
true condition	Cat [1]	True Positive (TP)	False Negative (FN) (type II error)
	Dog [0]	False Positive (FP) (Type I error)	True Negative (TN)

12 pictures, 8 of cats and 4 of dogs

6 2

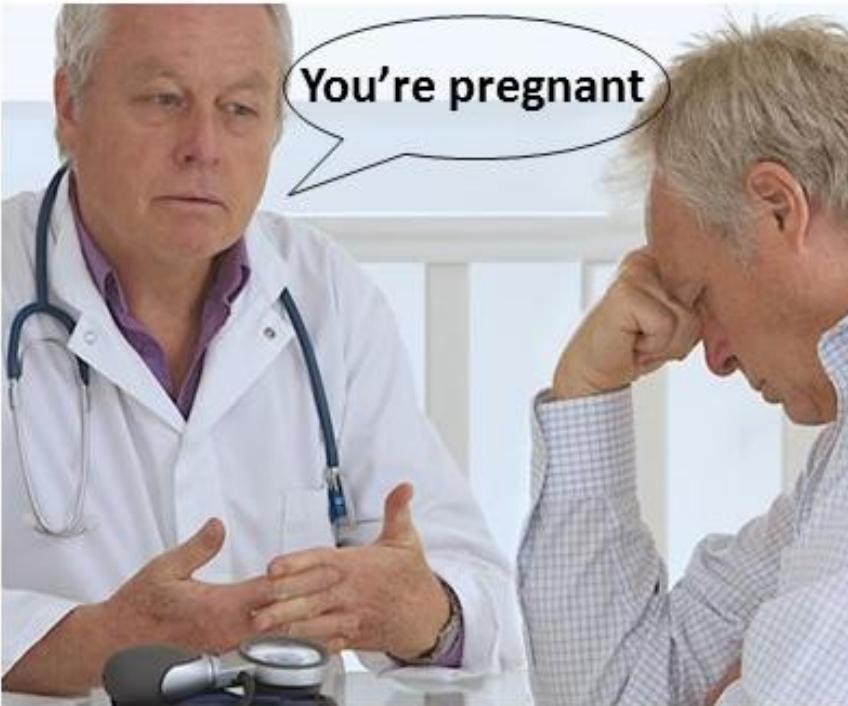
1 3

Confusion Matrix

		predicted condition	
		prediction positive (PP)	prediction negative (PN)
total population (P + N)			
true condition	condition positive (P)	True Positive (TP)	False Negative (FN) (type II error)
	condition negative (N)	False Positive (FP) (Type I error)	True Negative (TN)

Confusion Matrix

Type I error (false positive)



Type II error (false negative)



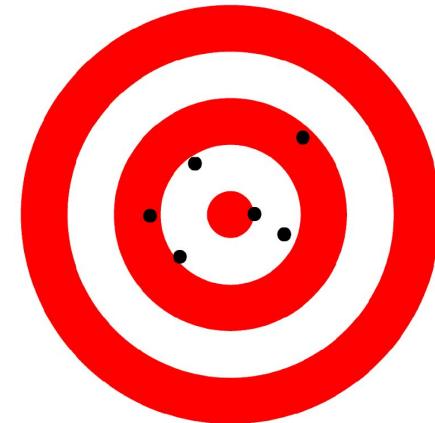
Precision vs. Accuracy

In a set of measurements:

- Accuracy is closeness of the measurements to a specific value
- Precision is the closeness of the measurements to each other.



High Precision, High Accuracy



Low Precision, High Accuracy



High Precision, Low Accuracy



Low Precision, Low Accuracy

Accuracy , Precision and Recall

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{(\text{P} + \text{N})} = \frac{\text{TP} + \text{TN}}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})} = \frac{6 + 3}{(6 + 3 + 1 + 2)} = \frac{9}{12} = 0.75$$

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} = \frac{6}{(6 + 1)} = \frac{6}{7} = 0.86$$

$\frac{\text{Total Positive}}{\text{Total Predict Positive}}$

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})} = \frac{6}{(6 + 2)} = \frac{6}{8} = 0.75$$

$\frac{\text{Total Positive}}{\text{Total Actual Positive}}$

(or Sensitivity)

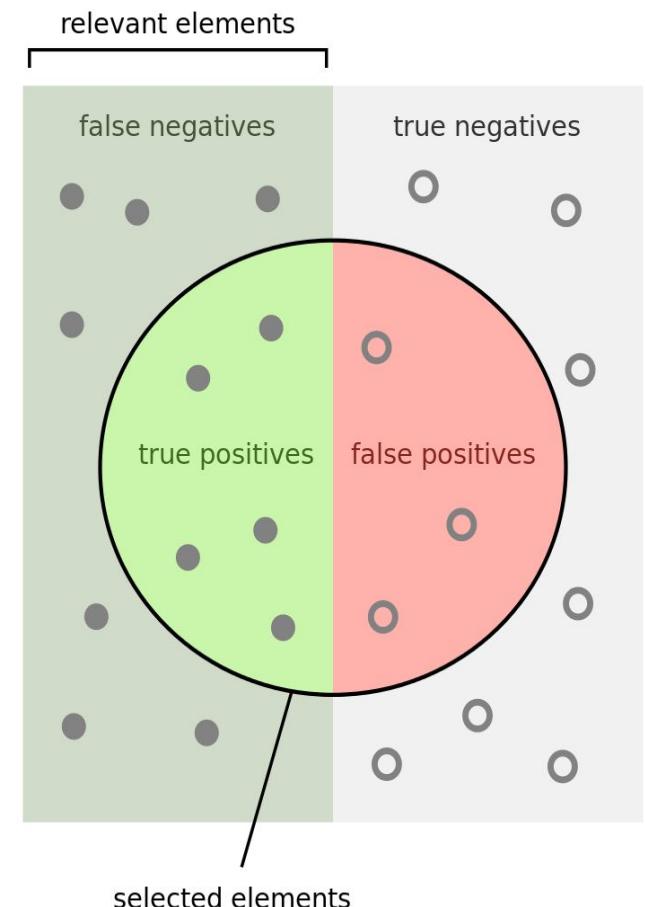
F1-Score

$$F1 = \frac{2 \times (\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

$$F1 = \frac{2 \times (0.86 * 0.75)}{(0.86 + 0.75)} = \frac{2 \times 0.65}{1.61} = 0.80$$

The F1-score is a way of combining the precision and recall of the model

<https://en.wikipedia.org/wiki/F-score#Formulation>



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

```
1 from sklearn.metrics import classification_report
```

```
1 actual = [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]  
2 prediction = [0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1]
```

```
1 target_names = ['Dogs', 'Cats']
```

```
1 print(classification_report(actual, prediction, target_names=target_names))
```

	precision	recall	f1-score	support
Dogs	0.60	0.75	0.67	4
Cats	0.86	0.75	0.80	8
accuracy			0.75	12
macro avg	0.73	0.75	0.73	12
weighted avg	0.77	0.75	0.76	12

Gracias!

Prof. Diego Méndez Chaves, Ph.D

Associate Professor - Electronics Engineering Department
Director of the Master Program in Internet of Things
Director of the Master Program in Electronics Engineering
email: diego-mendez@javeriana.edu.co

