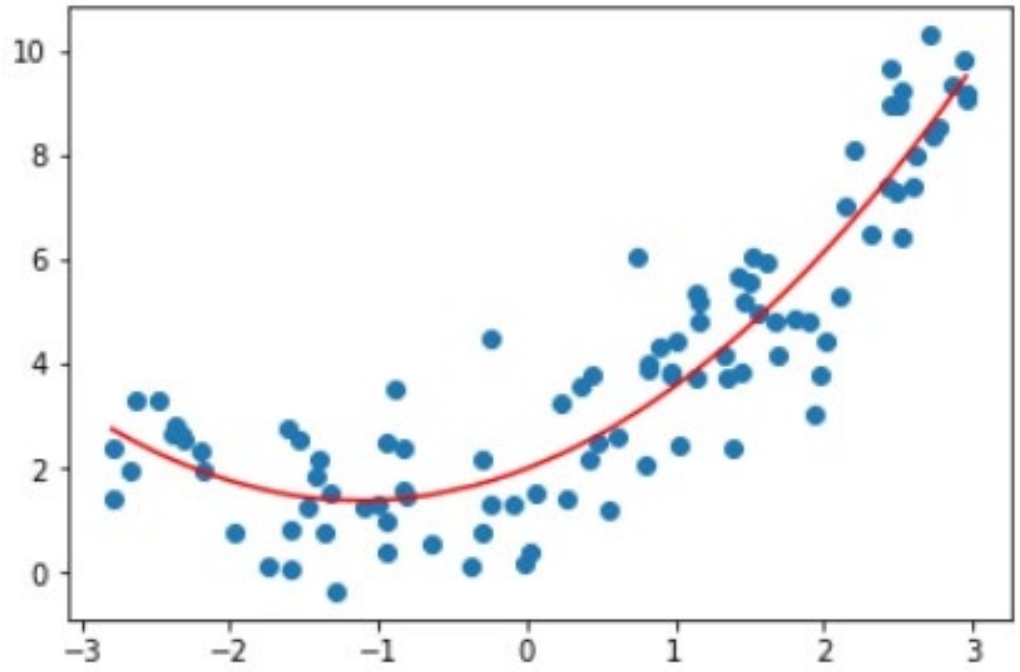
Lawrence Technological University Computer Science 1010

Regression Models and HPO

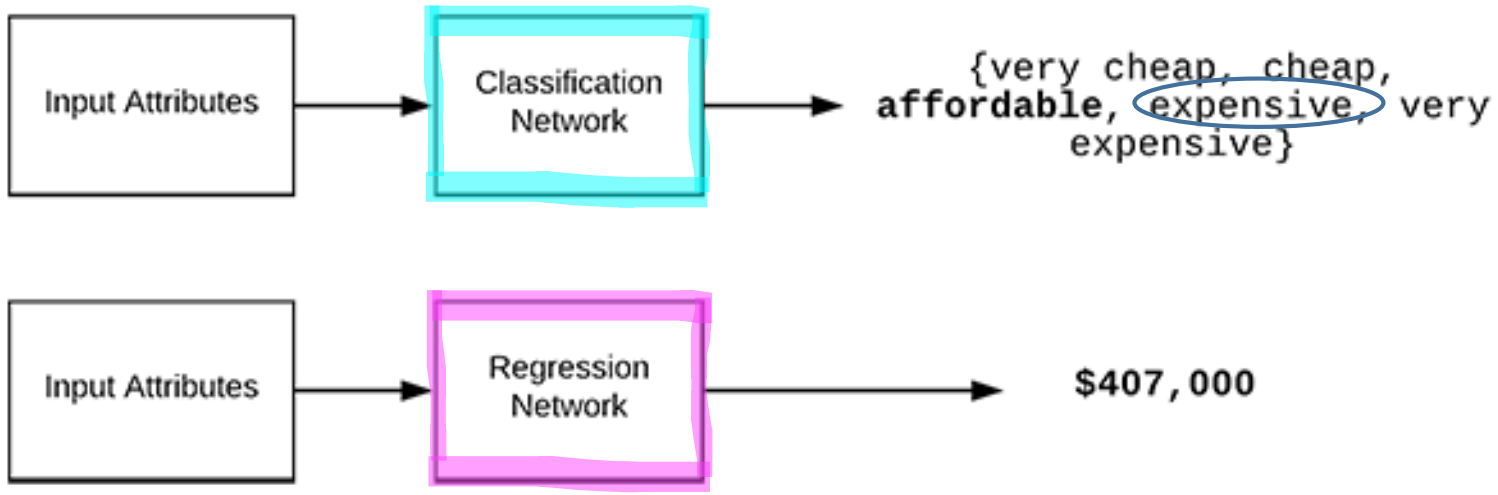
Additional Topics: Callback functions, Standardization, Pandas (How to load csv data into Keras), sklearn train_test_split(), and Numpy axis



What's the difference between classification and regression?

- Classification: output variable is **categorical** (or discrete)
- Regression: output variable is **numerical** (or continuous)

- Example: Predicting House Prices

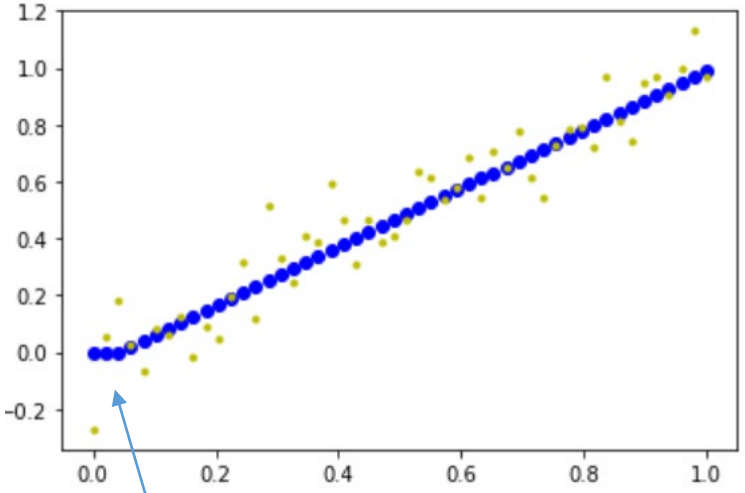


What is the similarity between classification and regression?

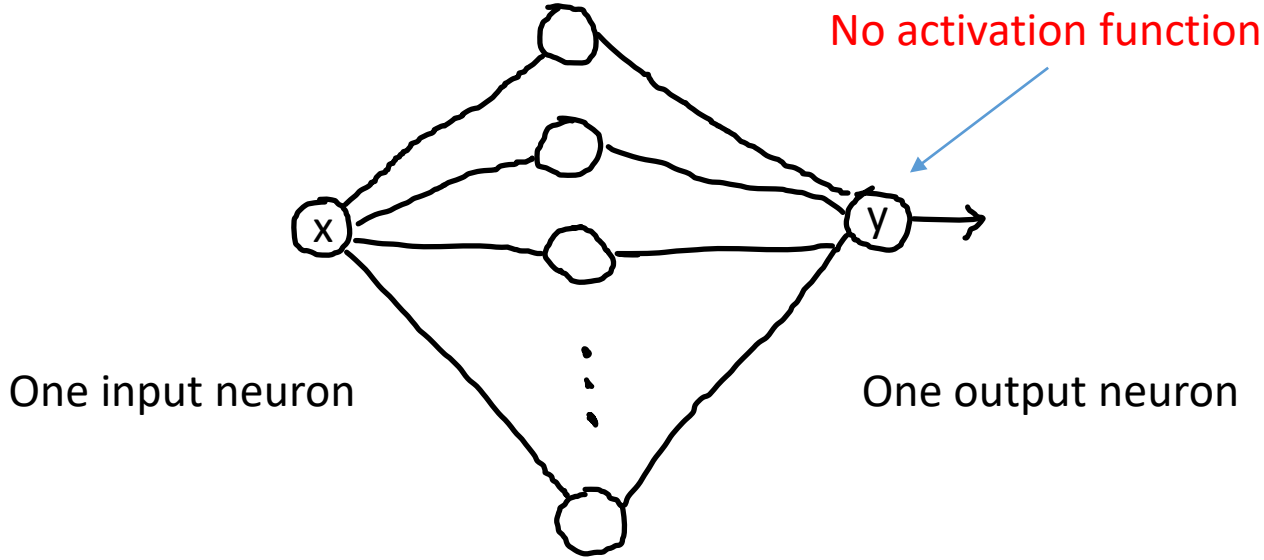
Supervised learning

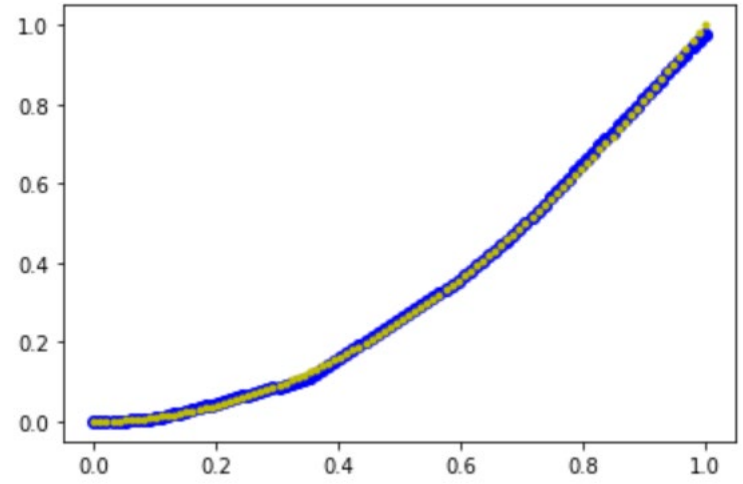
SimpleRegression.ipynb

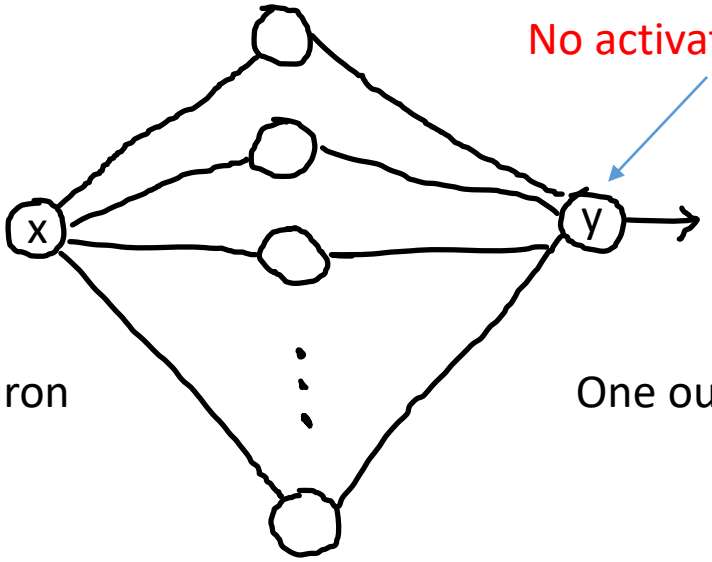
- Output can be any real value
- Linear regression
- Non-linear regression

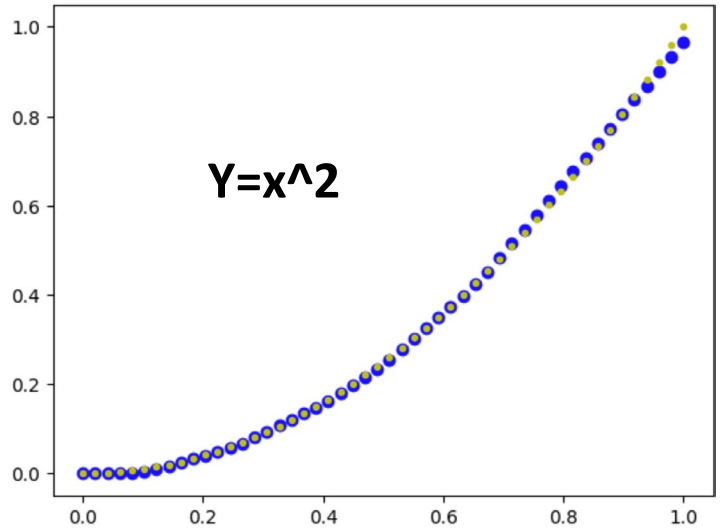


ReLU used



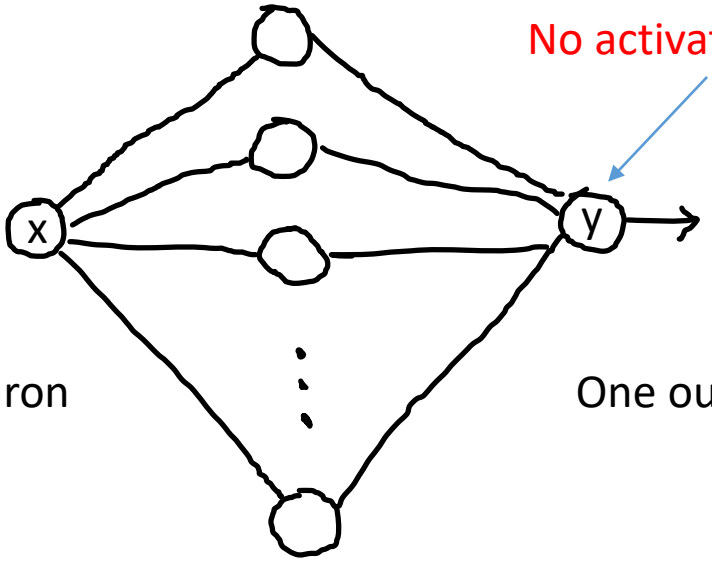


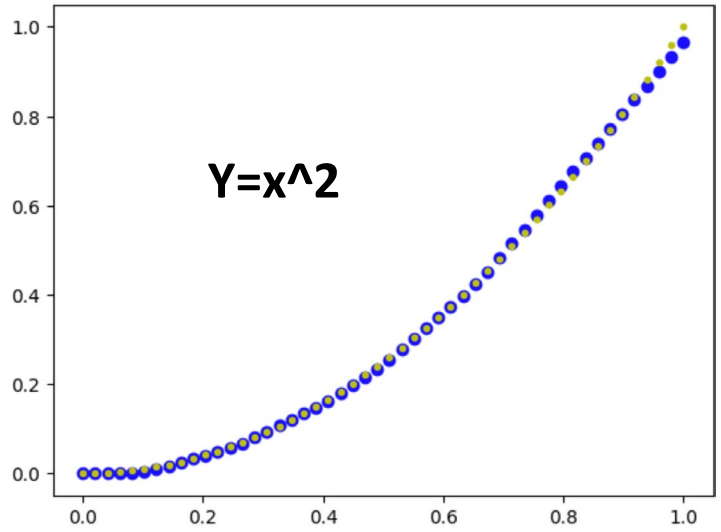





```
m3 = Sequential()
m3.add(Dense(32, input_dim=1, activation='relu'))
m3.add(Dense(1)) # Use no activation!
#m3.add(Dense(1, activation='sigmoid'))

m3.compile(optimizer=optimizers.SGD(lr=0.1), loss='mse')
```

OLD way






```
m3 = Sequential([
    Dense(32, input_dim=1, activation='relu'),
    Dense(1)
])
m3.compile(optimizer=optimizers.SGD(learning_rate=0.1), loss='mse')
m3.fit(X, Y3, epochs=500, batch_size=5, verbose = 0)
```

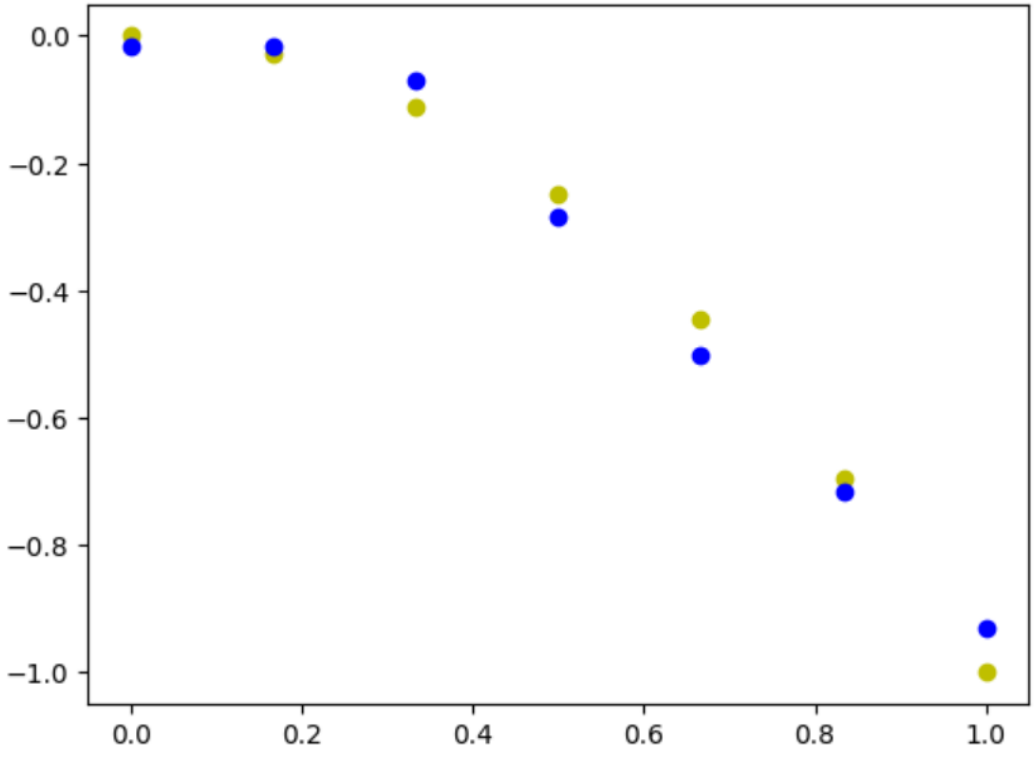
HPO of a Simple Regression Model Example: ES1115_HPO-x^2.ipynb

- Hyper parameters to optimize
  1. number of hidden neurons
  2. learning rate
  3. batch size

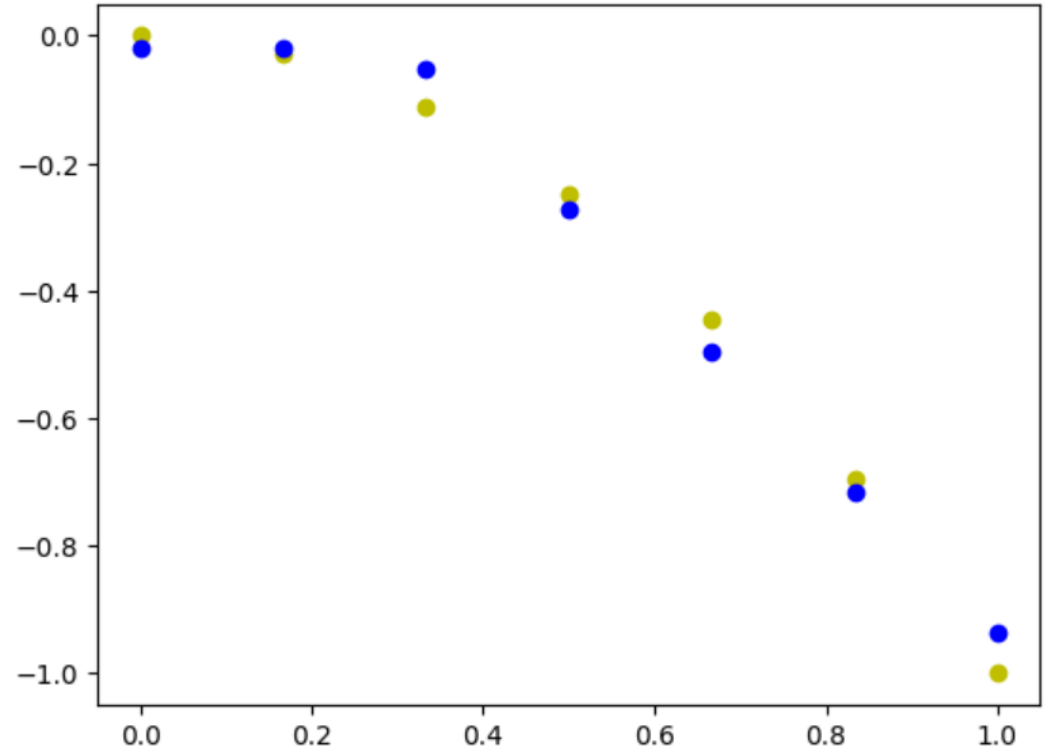
```
def func_eval(param):
  global m
  m = Sequential([
    Input(shape=(1,)),
    Dense(int(param[0]), activation='relu'),
    Dense(1)
  ])
  m.compile(optimizer=optimizers.SGD(learning_rate=param[1]), loss='mse')
  cb_list = [
    callbacks.EarlyStopping(monitor='val_loss', patience=5,
                            restore_best_weights=True),
  ]
  m.fit(X, y, epochs=400, batch_size=int(param[2]), verbose = 0,
        validation_data=(Xt, yt) , callbacks=cb_list)
  loss = m.evaluate(Xt, yt, verbose=0)
  return loss
```

Outputs of ES1115_HPO-x^2.ipynb

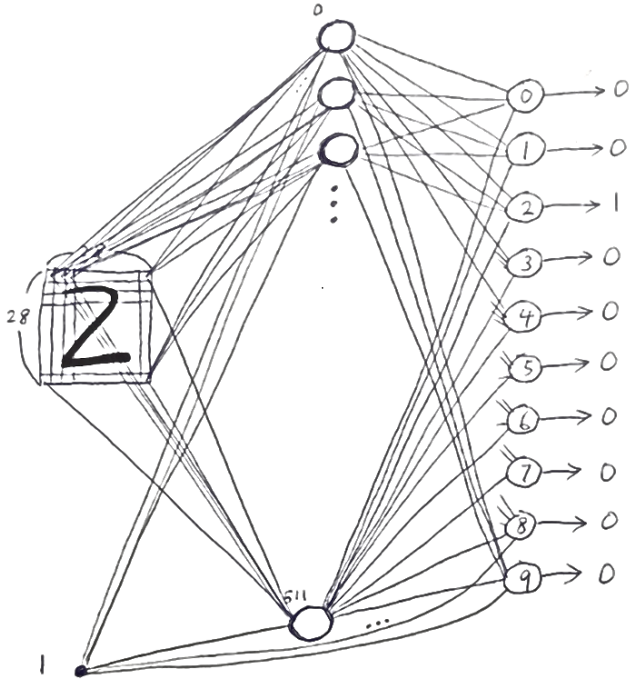
```
************************** Trial # = 1
Acceptable solution found after 3 generations:
#neurons=4, lr=0.056, bsize=4
Eval=0.001666784519329667
```



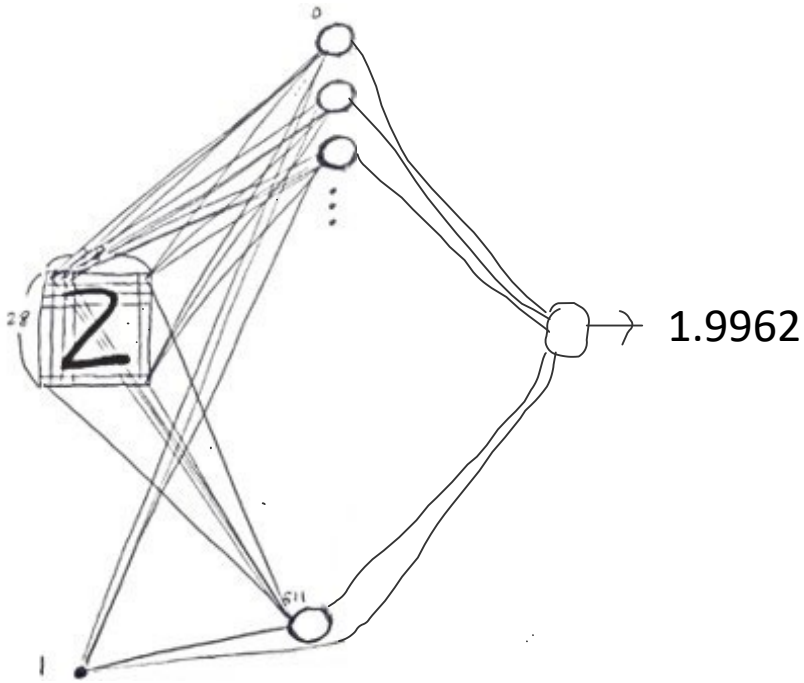

```
************************** Trial # = 2
Acceptable solution found after 7 generations:
#neurons=4, lr=0.316, bsize=10
Eval=0.0016347834607586265
```



Example: MNIST Handwritten Digit Recognition as a Regression Problem



Classification



Regression

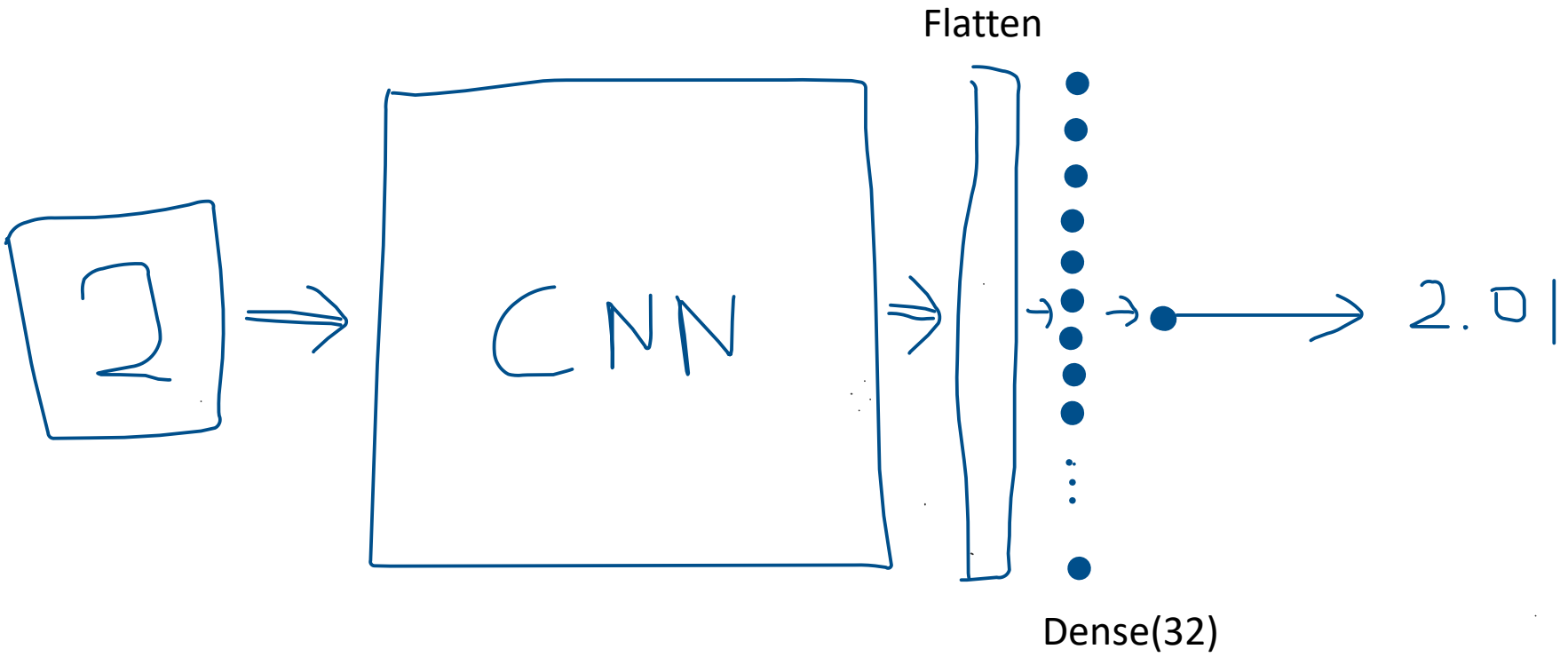
Example: MNIST Handwritten Digit Recognition as a Regression Problem using DNN, MNIST_DNN_regression.ipynb

- Note that “mae” is used for the metrics, unlike “accuracy” in classification problem.
- model.evaluate() returned mae because the program explicitly set metrics=['mae'] in model.compile().

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Train the model
model.fit(x_train, y_train, epochs=20, batch_size=32, validation_data=(x_val, y_val))


# Evaluate the model on the test set
test_loss, test_mae = model.evaluate(x_test2, y_test)
print(f'Test Mean Squared Error: {test_loss}, Test Mean Absolute Error: {test_mae}')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2872 - mae: 0.1493
Test Mean Squared Error: 0.28721147775650024, Test Mean Absolute Error: 0.1493115872144699
```

Example: MNIST Handwritten Digit Recognition as a Regression Problem using CNN, MNIST_CNN_regression.ipynb

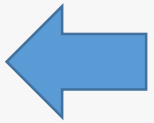


MNIST_CNN_regression.ipynb

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)  # One output neuron for regression
])
```

No activation function!

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

# Train the model with callback
cb_list = [
    callbacks.EarlyStopping(monitor='val_loss', patience=4,
                            restore_best_weights=True),
]
model.fit(x_train, y_train, epochs=20, batch_size=32,
          validation_data=(x_val, y_val),
          callbacks=cb_list,
          verbose=1)
```



Callbacks

Self-Assignment

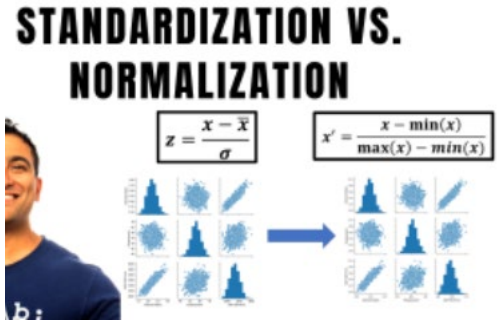
- HPO for **CNNs** such as MNIST Handwritten Digit Recognition as a Regression Problem
- Note that there are more Hyper Parameters for CNN models!!

Normalization vs Standardization

Feature Scaling is an important step to take prior to training of machine learning models to ensure that features are within the same scale. Two basic techniques:

- Normalization is conducted to make feature values range (usually) from **0 to 1**. (sometimes [-1, 1])
- Standardization is to transform the data to have a mean of zero and standard deviation of 1. Standardization is also known as Z-score normalization in which properties will have the behavior of a standard normal distribution.

https://youtu.be/bqhQ2LWBheQ



Min-max normalization to make feature values range from 0 to 1.

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79

Q: Student A scored 1800 on SAT, and B scored 24 on ACT.

Which student performed better relative to other test-takers?

Z-score Standardization

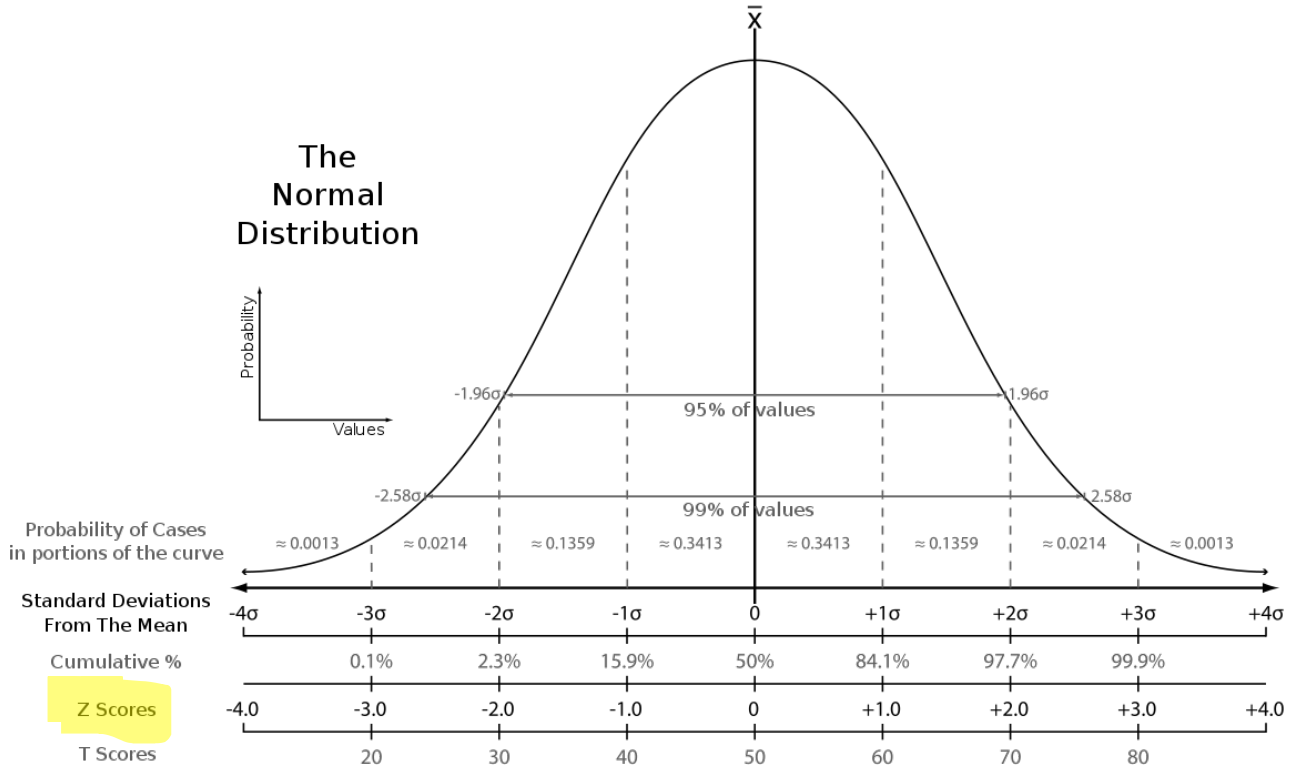
If the population mean and population standard deviation are known, the standard score of a raw score x[1] is calculated as

$$z = \frac{x - \mu}{\sigma}$$

where:

μ is the mean of the population.

σ is the standard deviation of the population.



Ex: Student A scored 1800 on SAT, and B scored 24 on ACT. Which student performed better relative to other test-takers?

Comparison of scores measured on different scales: ACT and SAT

When scores are measured on different scales, they may be converted to z-scores to aid comparison.[7] give the following example comparing student scores on the SAT and ACT high school tests. The table shows the mean and standard deviation for total score on the SAT and ACT. Suppose that student A scored 1800 on the SAT, and student B scored 24 on the ACT. Which student performed better relative to other test-takers?

	SAT	ACT
Mean	1500	21
Standard deviation	300	5

The z-score for student A is $z = \frac{x - \mu}{\sigma} = \frac{1800 - 1500}{300} = 1$

The z-score for student B is $z = \frac{x - \mu}{\sigma} = \frac{24 - 21}{5} = 0.6$

Because student A has a higher z-score than student B, student A performed better compared to other test-takers than did student B.

From Wikipedia

Standardization in Python

$$z = \frac{x - \mu}{\sigma}$$

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Note that mean and std to calculate z-score for test data are from "training population", not test population.

```
array([[1.23247e+00, 0.00000e+00, 8.14000e+00, ..., 2.10000e+01,
        3.96900e+02, 1.87200e+01],
       [2.17700e-02, 8.25000e+01, 2.03000e+00, ..., 1.47000e+01,
        3.95380e+02, 3.11000e+00],
       [4.89822e+00, 0.00000e+00, 1.81000e+01, ..., 2.02000e+01,
        3.75520e+02, 3.26000e+00],
       ...,
```

```
array([[-0.27224633, -0.48361547, -0.43576161, ...,  1.14850044,
         0.44807713,  0.8252202 ],
       [-0.40342651,  2.99178419, -1.33391162, ..., -1.71818909,
         0.43190599, -1.32920239],
       [ 0.1249402 , -0.48361547,  1.0283258 , ...,  0.78447637,
         0.22061726, -1.30850006],
       ...,
```

Standard Scaler from Scikit Learn.
What's wrong?

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Difference between "logistic regression" and "binary classification"

- Logistic regression → output a value between 0 and 1
- Binary classification -> class A or B
- Logistic regression is **a simple yet very effective classification algorithm** so it is commonly used for
  - many binary classification tasks and
  - regression problems with output value between 0 and 1 (or 0 and 100%)

How about bad pallet or good pallet term project?

Pros and cons of logistic regression (using one output neuron), instead of binary classification mechanism (using two output neurons)

- https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression
- https://stats.stackexchange.com/questions/207049/neural-network-for-binary-classification-use-1-or-2-output-neurons

In general, it is recommend to use one output neuron, since it is simpler and faster.

Logistic Regression Example `Diabetes_NN.ipynb` on Canvas

- How to access Google Drive files in Colab and
- How to handle **csv** files using **pandas** to predict the onset of diabetes in Pima Indians using a Keras NN model.
- References
  - https://www.kaggle.com/uciml/pima-indians-diabetes-database
  - https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
  - https://datanonymous.wordpress.com/using-a-neural-network-to-predict-diabetes-in-pima-indians/
  - https://towardsdatascience.com/pima-indian-diabetes-prediction-7573698bd5fe
- Test and study this code thoroughly!!

Intro Pandas: Making Data Frame (2D Table) from csv file (1/3)

Data Frame

```
diabetes_data = pd.read_csv('/content/drive/MyDrive/DL_data/simple_csv/diabetes.csv')
print(diabetes_data.shape)
diabetes_data.head()
```

```
(768, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Intro Pandas (2/3)

```
# selecting a single column
diabetes_data['Glucose']
```

```
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
```

```
# selecting multiple columns and saving them to another DataFrame
twoCol = diabetes_data[['Glucose','BloodPressure']]
twoCol.info()
twoCol.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Glucose        768 non-null    int64
 1   BloodPressure  768 non-null    int64
dtypes: int64(2)
memory usage: 12.1 KB
```

	Glucose	BloodPressure
0	148	72
1	85	66
2	183	64
3	89	66
4	137	40

Intro Pandas (2/3)

Converting "dictionary" to DF and adding a column to DF

```
import pandas as pd

# Define a dictionary containing student data
data = {'Name': ['AJ', 'BJ', 'CJ', 'DJ'],
        'Age':  [30,    40,   50,   60] } # must have only 4

print(data['Name']) # to access the dic using key

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Declare a list to be converted into a column
major = ['EE', 'ECE', 'CS', 'Math']

# Using 'Major' as the column name, and assign it to the col
df['Major'] = major

# Accessing cols
df['PicFile'] = df['Name']+df['Age'].astype(str)+".jpg"

print(df)
```

```
['AJ', 'BJ', 'CJ', 'DJ']
  Name  Age Major   PicFile
0   AJ   30    EE  AJ30.jpg
1   BJ   40   ECE  BJ40.jpg
2   CJ   50    CS  CJ50.jpg
3   DJ   60  Math  DJ60.jpg
```

df.to_csv('my.csv')

Pandas: to append a row

```
df0 = pd.DataFrame({'fn':[], 'label':[]}) # init a dataframe with col names
print(df0.index)
df0.loc[len(df0.index)] = ['3.10.png', 3.10]
df0.loc[len(df0.index)] = ['-4.15.png', -4.15]
print(df0)
```

```
RangeIndex(start=0, stop=0, step=1)
           fn  label
0   3.10.png   3.10
1  -4.15.png  -4.15
```

Prepare Training (80%) & Test (20%) Datasets

```
NumTrain = int(0.8*diabetes_data.shape[0]) # shape[0] has # of rows (samples)
print(NumTrain)
X = diabetes_data.to_numpy()[:NumTrain, 0:8]
y = diabetes_data.to_numpy()[:NumTrain, 8]
X_test = diabetes_data.to_numpy()[NumTrain:, 0:8]
y_test = diabetes_data.to_numpy()[NumTrain:, 8]
print(X.shape)
print(y.shape)
print(X_test.shape)
print(y_test.shape)
```

```
614
(614, 8)
(614,)
(154, 8)
(154,)
```

If you use this way, must check if balanced.

Diabetes_NN.ipynb

Using Sci Kit Learn

```
from sklearn.model_selection import train_test_split
#X = diabetes_data.iloc[:,0:8] #0-8 columns are dependent variables
X = diabetes_data.to_numpy()[:,0:8]
#y = diabetes_data.iloc[:,8]   #8 column is independent variable (label)
y = diabetes_data.to_numpy()[:,8]
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=17)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_test)
```

```
(576, 8)
(576,)
(192, 8)
(192,)
[0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1.
 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
```

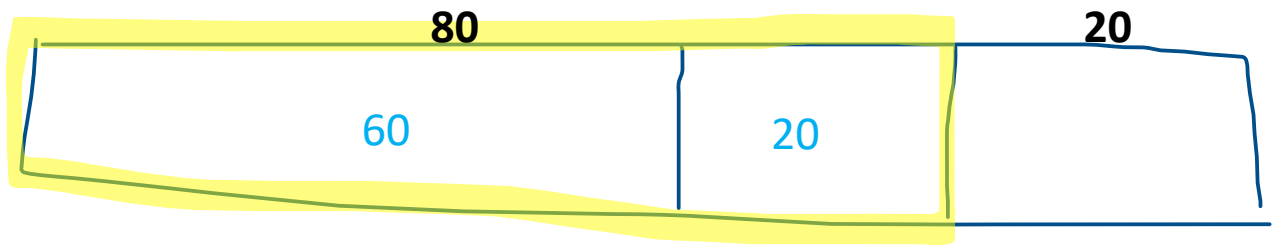
How to split a dataset into train, validation, and test sets using scikit-learn (sklearn) (1 /2)

Use train_test_split twice — first to create train+validation and test sets, then to further split the train+validation set.

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# 1. Load a sample dataset
data = load_iris()
X = data.data      # features
y = data.target    # labels

# 2. Split into train+validation and test sets (80% / 20%)
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

How to split a dataset into train, validation, and test sets using scikit-learn (sklearn) (2 /2)

```
# 3. Split the train+validation set into train and validation sets (80% / 20%)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.25, random_state=42, stratify=y_train_val
)
# Note: 0.25 × 0.8 = 0.2 → overall 60% train, 20% val, 20% test

# 4. Check the shapes
print("Train:", X_train.shape, y_train.shape)
print("Validation:", X_val.shape, y_val.shape)
print("Test:", X_test.shape, y_test.shape)
```



Diabetes_NN.ipynb

```
from keras.models import Sequential
from keras.layers import Input, Dense
from keras import callbacks
model = Sequential()
model.add(Input(shape=(8,)))  # 8 features in the input
for layer in range(2): # experiment for variable layers
  model.add(Dense(100*(3-layer), activation='relu'))
#model.add(Dense(128, activation='tanh'))
#model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

For variable number of layers

```
Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_80 (Dense)            (None, 300)               2700

 dense_81 (Dense)            (None, 200)               60200
```

Diabetes_NN.ipynb

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['mae'])
cb_list = [
    callbacks.EarlyStopping(monitor='val_loss', patience=5,
                            restore_best_weights=True),
  ]
model.fit(x_train, y_train, epochs=100, batch_size=10, verbose=1,
          validation_data=(x_test, y_test),
          callbacks=cb_list)
```

Instead of 'accuracy'

HPO for Heart Disease Prediction Logistic Regression Models

Assignment in F2024

https://www.kaggle.com/datasets/cherngs/heart-disease-cleveland-uci

HPO for Construction Task Regression Models to Predict Duration

Possible Future Project

Training Data in CSV

Task Type	Location	Weather	resources	Budget	Size	Scalability	Materials	**Duration**
1007	2	0	…	100,000	…	0	…	**50**
1008	3	1	…	200,000	…	1	…	**20**
…	…	…	…	…	…	…	…	**…**
2208	4	2	…	…	…	0	…	**43**

Test Data in CSV

Task Type	Location	Weather	resources	Budget	Size	Scalability	Materials	Duration
1007	4	0	…	134,000	…	0	…	??
3021	5	1	…	150,000	…	1	…	??

HPO for Construction Task Regression Models to Predict Duration

Possible Future Project

Training Data in CSV

Task Type	Location	Weather	resources	Budget	Size	Scalability	Materials	**Duration**
1007	2	0	…	100,000	…	0	…	**50**
1008	3	1	…	200,000	…	1	…	**20**
…	…	…	…	…	…	…	…	…
2208	4	2	…	…	…	0	…	**43**

Can we use CNN models for this problem?

Test Data in CSV

Task Type	Location	Weather	resources	Budget	Size	Scalability	Materials	Duration
1007	4	0	…	134,000	…	0	…	??
3021	5	1	…	150,000	…	1	…	??

Q. numpy.mean(a, axis=None, …)

Compute the arithmetic mean along the specified axis.

```
X = np.array([[1,2],[3,4]])
np.mean(X) # default is to compute the mean of flattened array
```

```
2.5
```

```
np.mean(X, axis=0) # axis=0 means row. row is collapsed
```

```
array([      ])
```

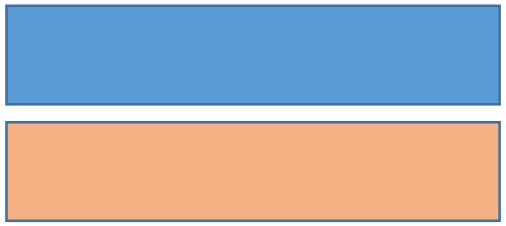
```
np.mean(X, axis=1) # axis=1 means column. column is collapsed
```

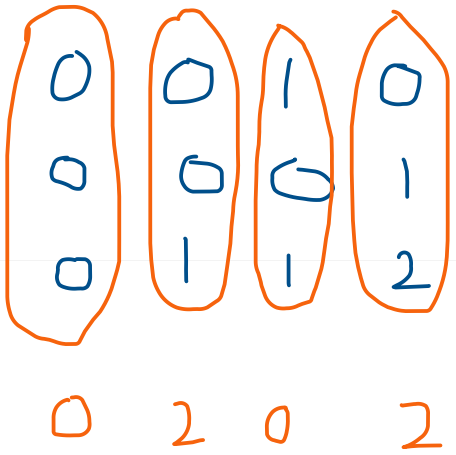
```
array([      ])
```

https://www.sharpsightlabs.com/blog/numpy-axes-explained/

Q. axis=0, axis=1

```
threeBy4 = np.array([[0,0,1,0], [0,0,0,1], [0,1,1,2]])

print(np.argmax(threeBy4, axis=0))
print(np.argmax(threeBy4, axis=1))
```





Q. Numpy Concatenate

```
t = np.array([1,2,3,4,5,6])
np.concatenate([t[:2], t[4:]])
```

```
array([                ])
```

https://www.tutorialspoint.com/numpy/numpy_concatenate.htm

Q. Differences btw Normalization and Standardization?

- Normalization → scales to a range, usually [0, 1].
- Standardization → Center values around mean 0 with standard deviation 1

Questions

- How do we standardize (normalize) **test** data?

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
print(train_data)

# note that the mean and std are from training population.
# Read textbook page 115
test_data -= mean
test_data /= std
```

- Do we standardize (normalize) target labels?