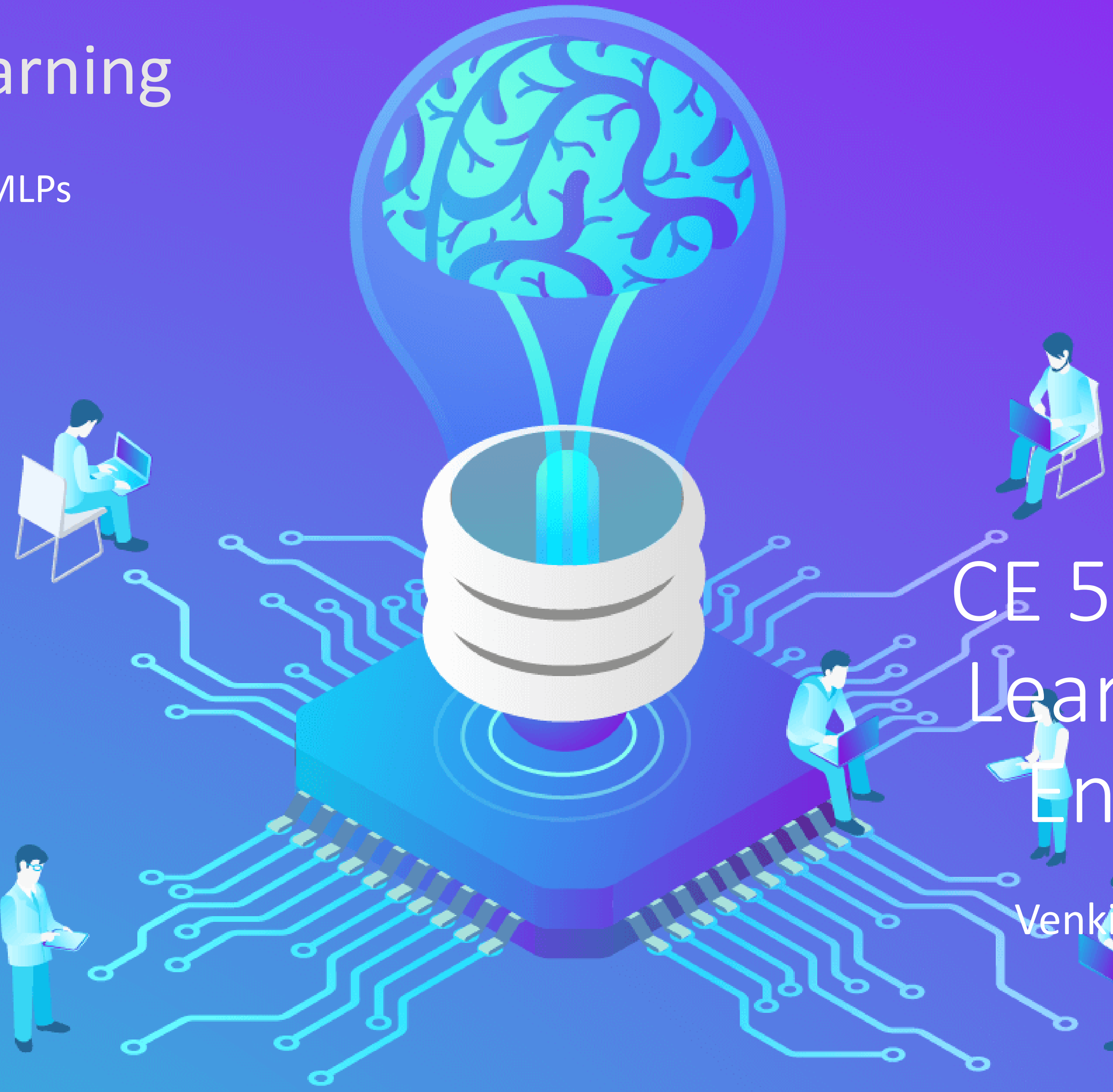
Machine Learning

Implementing MLPs



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap

- What is Machine Learning
- How is it useful for Civil Engineers
- Overview of Machine Learning Methods
- Linear Regression
  - Bivariate
  - Regression interpretation
  - Multivariate
- Logistic Regression
  - Maximum likelihood estimation
  - Regularization (introduction)
- Naïve Bayesian Classifier
  - What is it
  - What makes it naïve
  - Bayes theorem
  - Prior, likelihood and posterior
- K-Nearest Neighbor
  - How does the algorithm work
  - Why is it a lazy learner
  - How to do regression and classification
- Introduction to Decision Trees
  - Fundamentals
  - Information Gain, Entropy and Gini Index
  - ID3 algorithm
  - Classification and Regression Trees (CART)
  - Multi-Adaptive Regression Splines (MARS)

- Ensemble learners
  - Introduction
- Their benefits and drawbacks
- Simple (voting) ensemble learners
- Bagging and Pasting
- Generic bagging classifiers
- Random Forest classifiers
- Boosting Classifier
  - Adaboost
- Unsupervised classification
  - K Means Learning
- Introduction to Artificial Neural Networks
- Perceptron
- Overview of MLPs

Python – Introduction
Python – Functions
Python - Pandas
Python – np, scipy, statsmodels
Python – Scikit learn – linear, metrics
Python – Matplotlib, seaborn
Python – Mixed_Naive_Bayes
Python – scikit learn neighbors module
Python – scikit learn ensemble voting
Python – scikit learn bagging classifier
Python – scikit learn RandomForestClassifer
Python – scikit learn AdaBoostClassifier
Python – scikit learn Kmeans
Python – scikit learn perceptron

R – Classification and Regression Trees using rpart
R – Drawing trees using rpart.plot
R - Multiadaptive Regression Splines (MARS) using Earth Algorithm

Artificial Neural Networks

Goals

- Implementing MLPs in Python
  - Keras
  - Tensorflow
- Illustrative case-study for a binary classifier

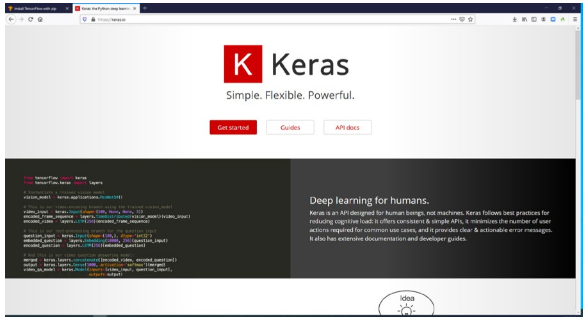
Introduction

- General purpose ANN libraries have made implementation of these models easy
  - Python has led the way in the development of these software
- There are many general purpose packages available
  - Tensor Flow (Google)
  - Microsoft Cognitive Toolkit (CNTK)
  - MXNET (Apache)
  - PyTorch (Facebook)
  - Theano (University of Montreal)
- There are multipurpose front-end packages that simply the use of the above packages
  - Keras
  - TFLearn
  - Clarifai

Theano was a pioneer but its development has effectively stopped

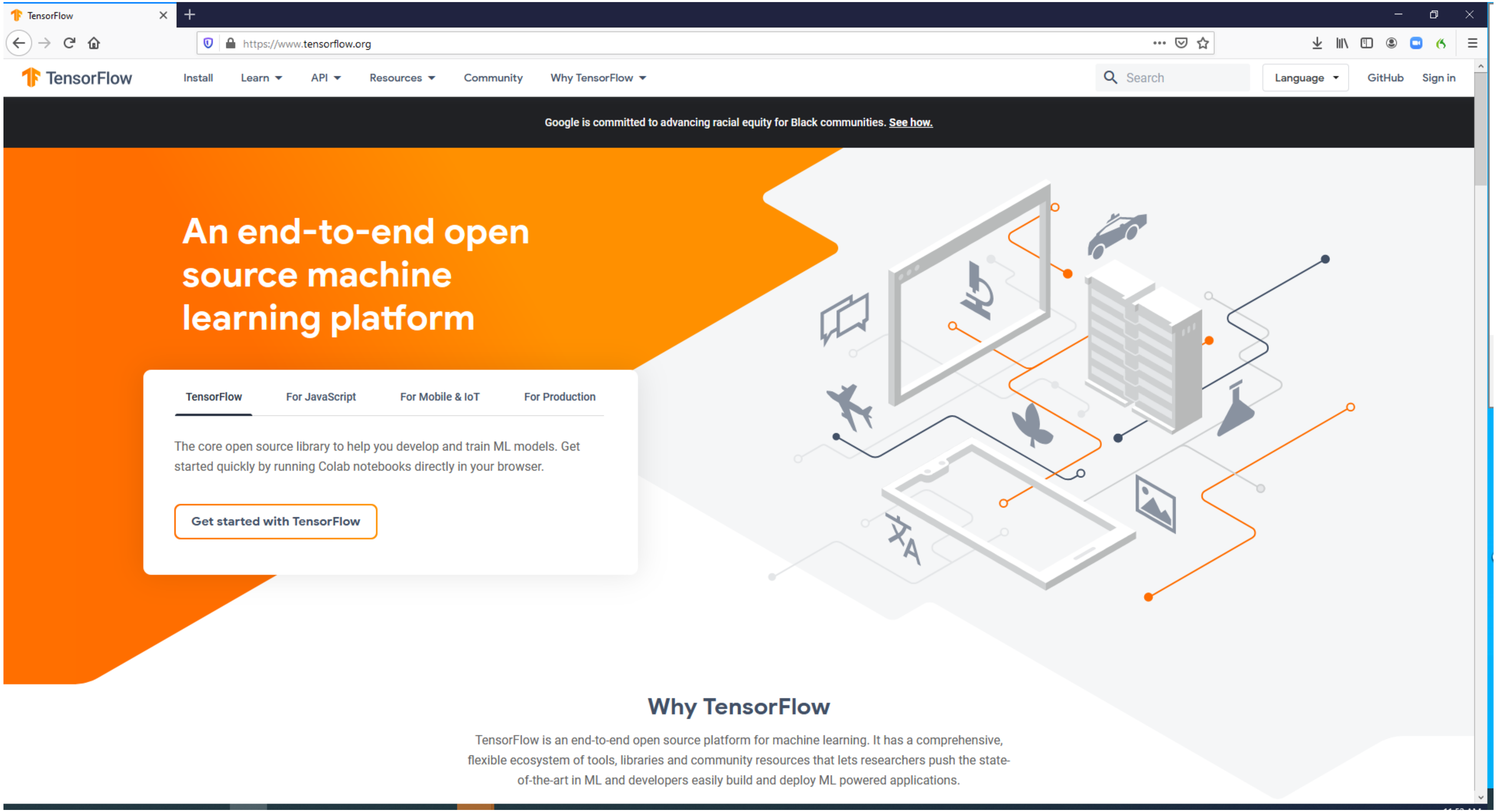
Keras and Tensor Flow have emerged as Industry Standard so we shall adopt here

Keras



- Keras is an open-source deep learning library written in Python.
- The project was started in 2015 by Francois Chollet.
  - It quickly became a popular deep learning library.
- Developing ANN models in TensorFlow, Theano, Pytorch was generally cumbersome
  - Several hundred or thousand lines of code for even simple tasks
- Keras provided workflows for many ANNs which made their development easy
- Keras also supported multiple back-end models
  - TensorFlow, Theano, CNTK
- Keras is still available and popular
  - By default it uses TensorFlow backend
- In 2019 Google integrated a version of Keras into TensorFlow 2.0

For Keras Documentation Go to: https://keras.io/

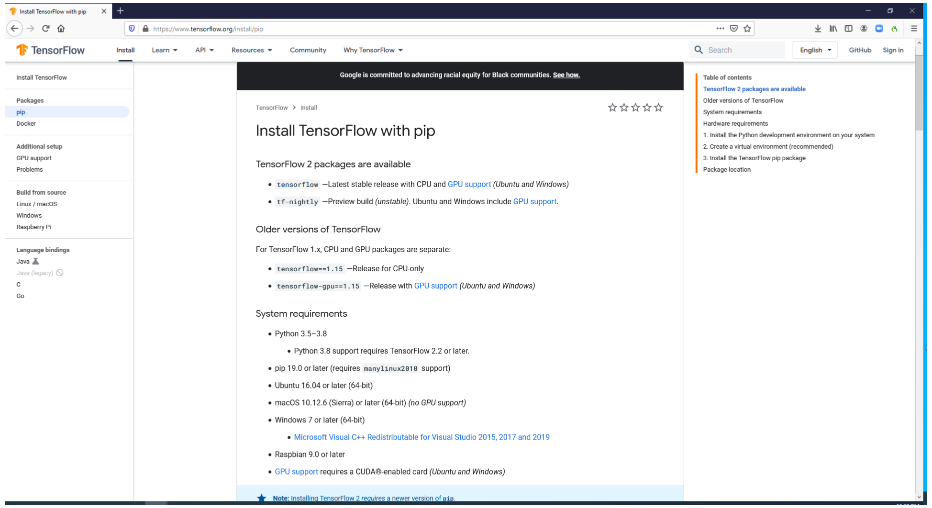

TensorFlow
https://www.tensorflow.org
TensorFlow
Install
Learn
API
Resources
Community
Why TensorFlow
Search
Language
GitHub
Sign in
Google is committed to advancing racial equity for Black communities. See how.
An end-to-end open source machine learning platform
TensorFlow
For JavaScript
For Mobile & IoT
For Production
The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.
Get started with TensorFlow
Why TensorFlow
TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow

- TensorFlow is a development ecosystem for building machine learning models
  - Developed by Google (initially for internal use)
    - Version 1.0 of TensorFlow was released to public in 2017
    - Version 2.0 released in 2019
      - Current release
  - It is a symbolic calculation library
    - Has many routines such as automatic differentiation, optimization that are useful for building machine learning models
- TensorFlow provides various levels of abstraction and ways to access its functionality
  - Beginners and Researchers most likely will use the Keras interface
- Starting TensorFlow Version 2.0 a version of Keras is incorporated within TensorFlow
  - Need not download and open two libraries
  - Everything can be performed from within TensorFlow
    - If you are using Version 1.x of TensorFlow you still need two packages

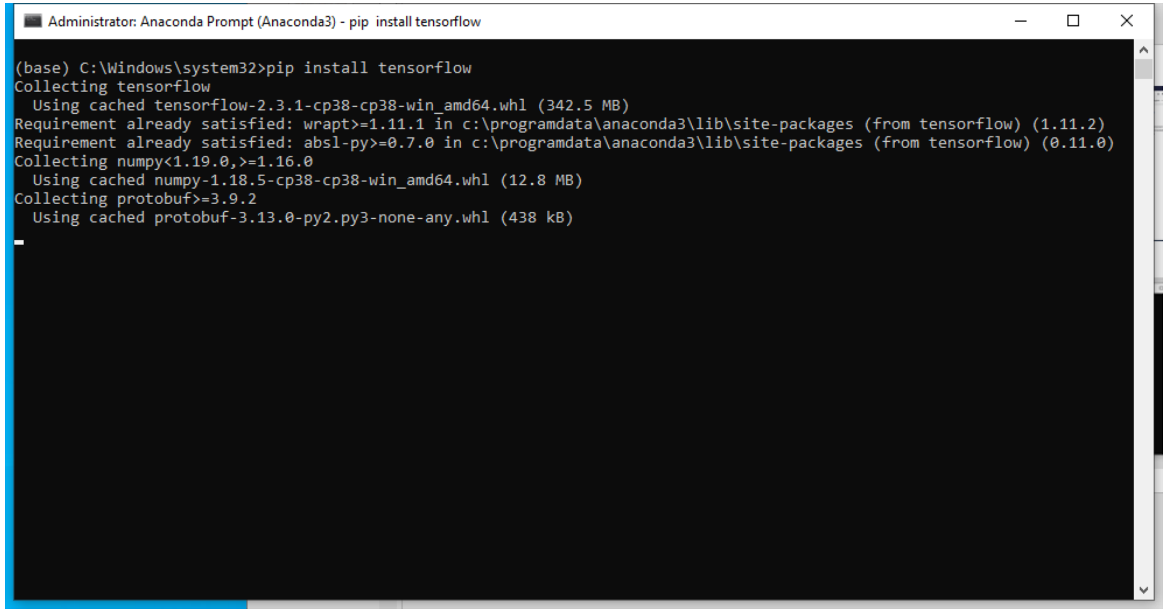
Installing Tensorflow

- You may not have tensorflow installed
- You may have an older version installed



You need to have administrator privileges

```
import tensorflow as tf
tf.__version__
```



TensorFlow > 2.0

- There are several improvements made in TensorFlow 2.0
  - Better execution
  - More consistent syntax
- A version of Keras is included in TensorFlow > 2.0
  - No need to import two separate packages
    - TF is imported in the background for Keras
  - Keras embedded in TensorFlow and can be used

Multi-backend Keras and tf.keras:

At this time, we recommend that Keras users who use multi-backend Keras with the TensorFlow backend switch to `tf.keras` in TensorFlow 2.0. `tf.keras` is better maintained and has better integration with TensorFlow features (eager execution, distribution support and other).

Keras 2.2.5 was the last release of Keras implementing the 2.2.* API. It was the last release to only support TensorFlow 1 (as well as Theano and CNTK).

The current release is Keras 2.3.0, which makes significant API changes and add support for TensorFlow 2.0. The 2.3.0 release will be the last major release of multi-backend Keras. Multi-backend Keras is superseded by `tf.keras`.

Bugs present in multi-backend Keras will only be fixed until April 2020 (as part of minor releases).

For more information about the future of Keras, see the Keras meeting notes.

TensorFlow Ecosystem is vast - https://www.tensorflow.org/resources/libraries-extensions

Illustrative Examples

Multilayer Perceptron (Binary Classification)

Illustrative Example

- Predicting damage to culverts in Texas
- Create a MLP model to classify satisfactory and unsatisfactory states

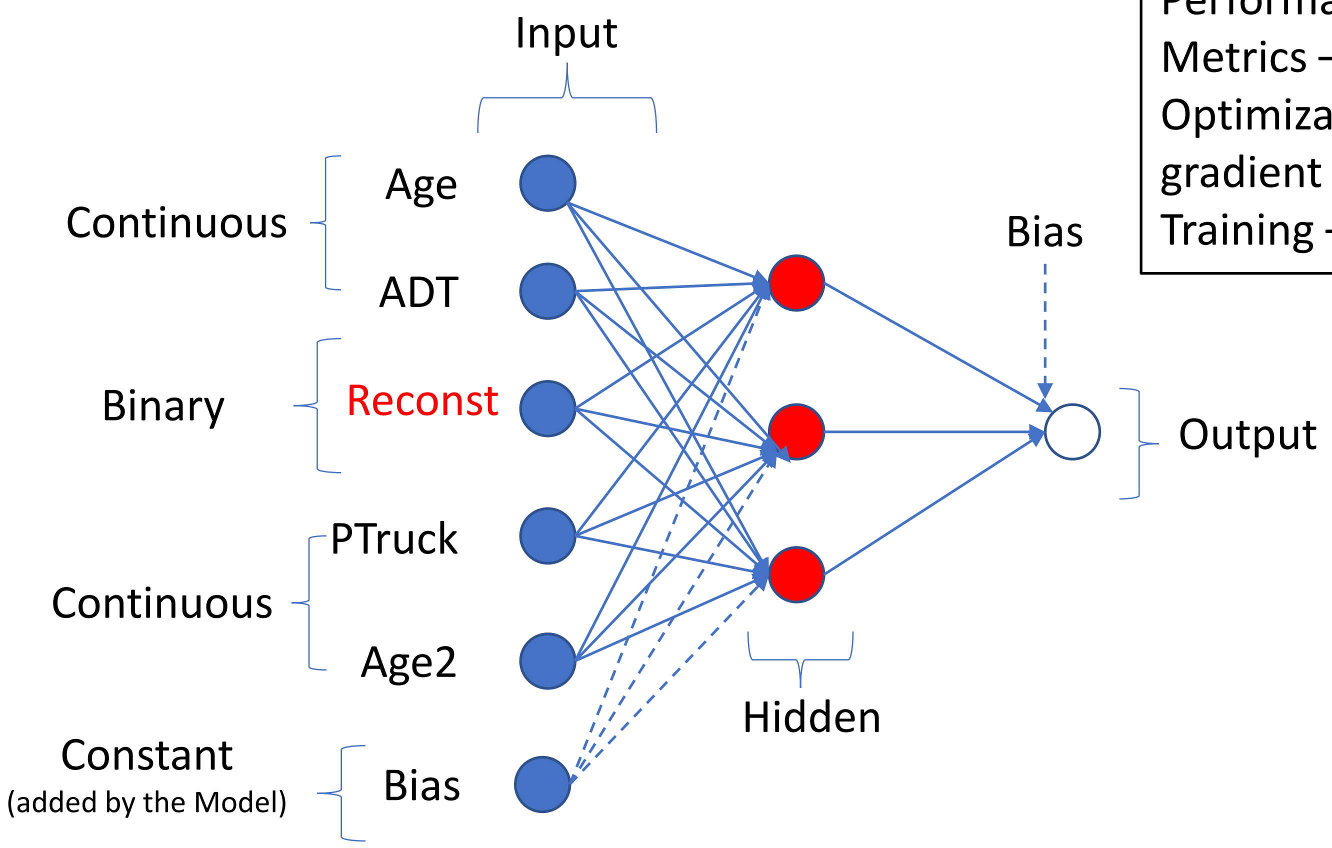
Satisfactory

Unsatisfactory

Code	Meaning	Description
9	Excellent	As new
8	Very Good	No problems noted.
7	Good	Some minor problems.
6	Satisfactory	Structural elements show some minor deterioration.
5	Fair	All primary structural elements are sound but may have minor section loss, cracking, spalling or scour.
4	Poor	Advanced section loss, deterioration, spalling or scour.
3	Serious	Loss of section, deterioration, spalling or scour has seriously affected primary structural components. Local failures are possible. Fatigue cracks in steel or shear cracks in concrete may be present.
2	Critical	Advanced deterioration of primary structural elements. Fatigue cracks in steel or shear cracks in concrete may be present or scour may have removed substructure support. Unless closely monitored it may be necessary to close the bridge until corrective action is taken.
1	Imminent Failure	Major deterioration or section loss present in critical structural components or obvious vertical or horizontal movement affecting structure stability. Bridge is closed to traffic but with corrective action may put back in light service.
0	Failed	Out of service, beyond corrective action.

Source: United States Department of Transportation. Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges. Washington, D.C., 1995, page 38.

Conceptual Model



Inputs – Same as Before
Hidden Nodes – Requires some trial and error
Activation Functions:
- Hidden layer – RELU
- Output layer – Sigmoid
  - Needs a binary output

Performance Measure (loss) – Binary Cross Entropy
Metrics – Accuracy
Optimization method – Adam (version of stochastic gradient descent)
Training – 5 epochs; batch size 32; validation split 0.2

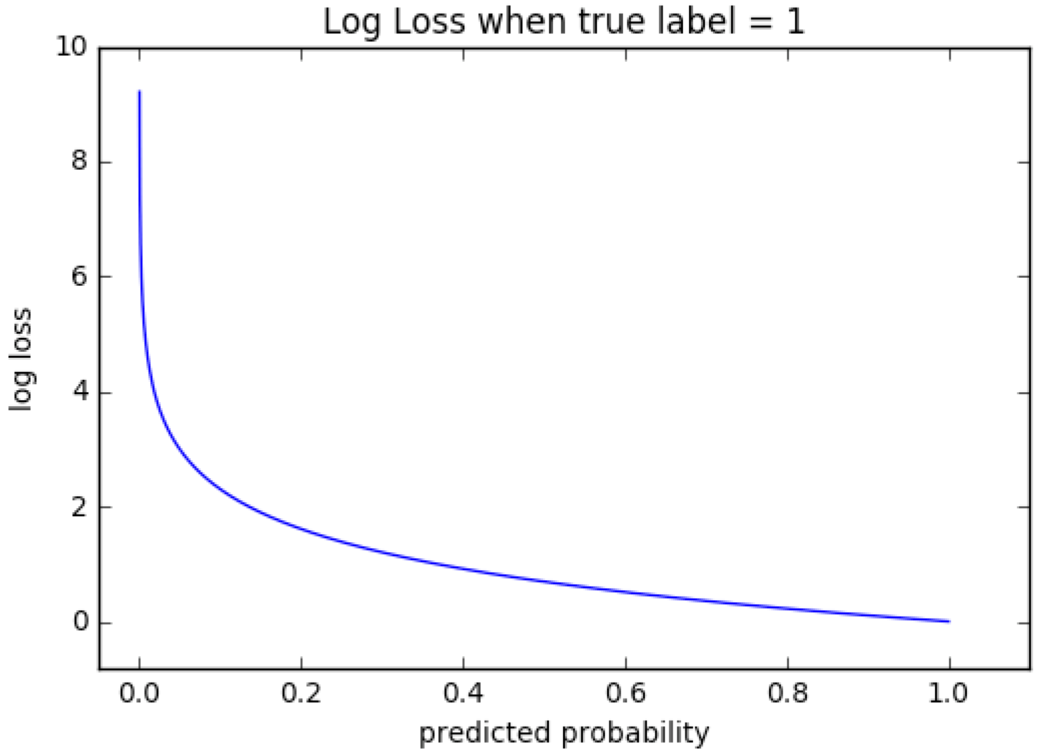
Input - Hidden
(5 inputs x 3 hidden nodes)
+ 3 bias terms = 18 weights

Hidden – Output
3 (hidden nodes) x 1 output
+ 1 bias term (output)

Total # Weights = 22

Conceptual Model

- What is (binary) Cross-Entropy:
  - A measure of difference between two probability distributions for a given random variable or a state
  - Often used inter-changeably with log loss function
- Cross-entropy loss increases when the predicted class deviates from the actual class
- For a perfect classifier log loss or cross-entropy loss is zero



$$\boxed{L = -\sum_{i=1}^{N} (y_i log(p_i) + (1 - y_i)(1 - log(p_i)))}$$

How many Hidden Nodes

- Identifying hidden nodes is both science and art
  - Will require some trial-and-error experimentation
- There is some subjectivity to this decision
  - But it is an important decision that can affect the result
- Several methods proposed in the literature
  - Heuristics
  - Regularization
    - Dropout

A reasonable number of hidden nodes is problem specific and a function of number of inputs; number of outputs; number of datapoints as well as the objective function being used train the model

One Heuristic (of the many in the literature)

$$N_h = \frac{I + \sqrt{N}}{L}$$

N # training data
I # Inputs

Recommended Upper Bounds

$$N_h = (2I + 1)$$

$$N_h = \frac{N}{I + 1}$$

Typically far less number of nodes may be sufficient

Lower Bound

Theoretically 1 but generally 2

Too few nodes → Insufficient generalization; Too many nodes → Overfitting

Optimization Methods

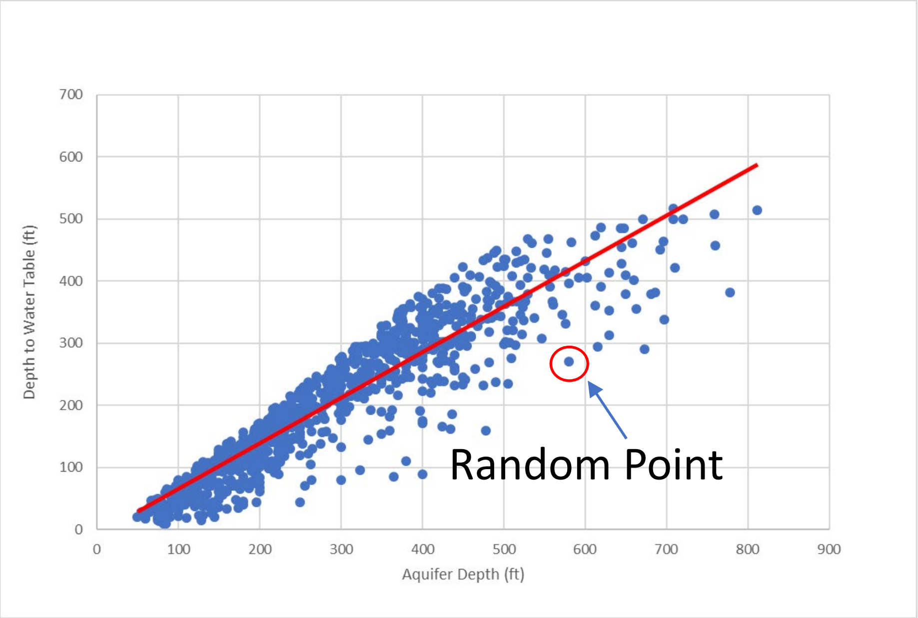
- There are several methods
  - Stochastic Gradient Descent (SGD) is typically used
  - Others are variants of SGD

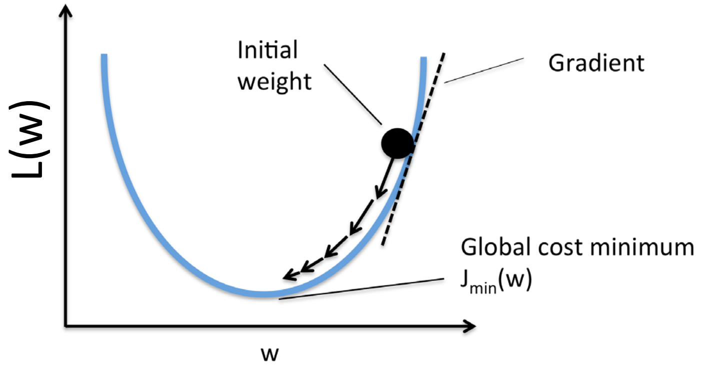
$$L = \sum (y_o - (mx + b))^2$$

$$\left.\begin{aligned} w_m^{t+1} &= w_m^t + \eta \frac{\partial L}{\partial m} \\ w_b^{t+1} &= w_b^t + \eta \frac{\partial L}{\partial b} \end{aligned}\right\}$$

Weight Update Equations

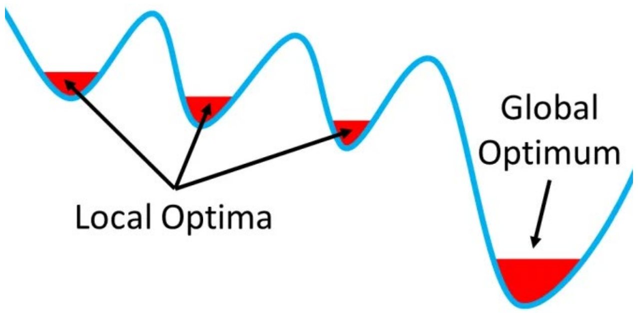
η Is the learning rate (user-specified)





Initialize guesses for b and m
Pick a point randomly compute the derivatives at that point
Use the derivatives to update the weights
Use all points in the batch to compute the loss function
Repeat till Loss function converges to an optimum

Optimization Method – Derivative Computations



- Weight updating requires computation of derivatives
  - This is the biggest source of uncertainty
  - An inaccurate derivative calculation can make converge to a local optima or miss a global optima.

- Derivatives can be computed in three ways
  - Analytical differentiation
    - Not possible or cumbersome for complex models
  - Numerical differentiation
    - Finite-difference methods
    - First-order truncations are error prone
    - Computationally expensive
  - Symbolic integration
    - Not possible for all functions
    - Can have redundant elements
  - Automatic differentiation
    - Convert the function to a sequence of primitive operations
    - Primitive operations have known derivatives
    - Use Chain-Rule to solve the original derivative

TensorFlow uses Reverse-Mode AutoDifferentiation to solve gradients

Reverse Model Automatic Differentiation (Example)

$$z = x_1 x_2 + sin(x_1)$$

Find: $\frac{dz}{dx_1}$ and $\frac{dz}{dx_2}$

Forward Mode

Decompose the function into primitive equations (whose derivatives are known)

$$\left\{\begin{aligned} w_1 &= x_1 \\ w_2 &= x_2 \\ w_3 &= w_1 w_2 \\ w_4 &= \sin(w_1) \\ w_5 &= w_3 + w_4 \\ z &= w_5 \end{aligned}\right.$$

Reverse Mode

Use chain-rule to solve the derivative

$$\frac{dz}{dw_1} = \frac{dz}{dw_3}\frac{dw_3}{dw_1} + \frac{dz}{dw_4}\frac{dw_4}{dw_1} = w_2 + \cos(w_1)$$

Randomly pick a point (x_1, x_2,z) and substitute to get the numeric value of the derivative

Python Code

```
# Step 1: Import Libraries
import os
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import
train_test_split
import tensorflow as tf
from sklearn import metrics
import seaborn as sns
from matplotlib import pyplot as plt
```

```
# Step 2: Change working directory
dir = ‘D:\\Code'
os.chdir(dir)
```

```
# Step 3: Read the dataset
a = pd.read_csv('TXculvertdata.csv') # read our dataset
features = ['SVCYR','ADT','Reconst','PTRUCK'] # INPUT DATA FEATURES
X = a[features] # DATAFRAME OF INPUT FEATURES
SVCYR2 = a['SVCYR']**2  # Add SVCYR square to the dataset
X['SVCYR2'] = SVCYR2  # CALCULATE THE SQUARE OF AGE
Y = a['Culvert_Damage'] # ADD IT TO THE INPUT FEATURE DATAFRAME
```

```
# Step 4: Apply Data scaling
scaler = StandardScaler() # Initialize standardscaler
X_scl = scaler.fit_transform(X)
```

```
# Step 5: Split into training and testing data
X_train,X_test, y_train,y_test = train_test_split(X_scl,Y, test_size=0.30,
                                        random_state=10)
```

```
# Step6: Setup Keras Model
model = tf.keras.Sequential([
   tf.keras.layers.Dense(12, input_dim=5, activation='relu'),
   tf.keras.layers.Dense(12,activation='relu'),
   tf.keras.layers.Dense(1,activation='sigmoid'),
])
```

```
# Step 7: Compile Model
model.compile(loss='binary_crossentropy',
          metrics=['accuracy'],
          optimizer='sgd')

# Step 8: Fit the Model
model.fit(X_train, y_train, epochs=150, batch_size=500)
```

Python Code (Cont.)

```
# Step 8: Fit the Model
#model.fit(X_train, y_train, epochs=100, batch_size=500)

#Step 8a: If you want to do cross-validation
model.fit(X_train, y_train, epochs=150,batch_size=100,validation_split=0.25)
```

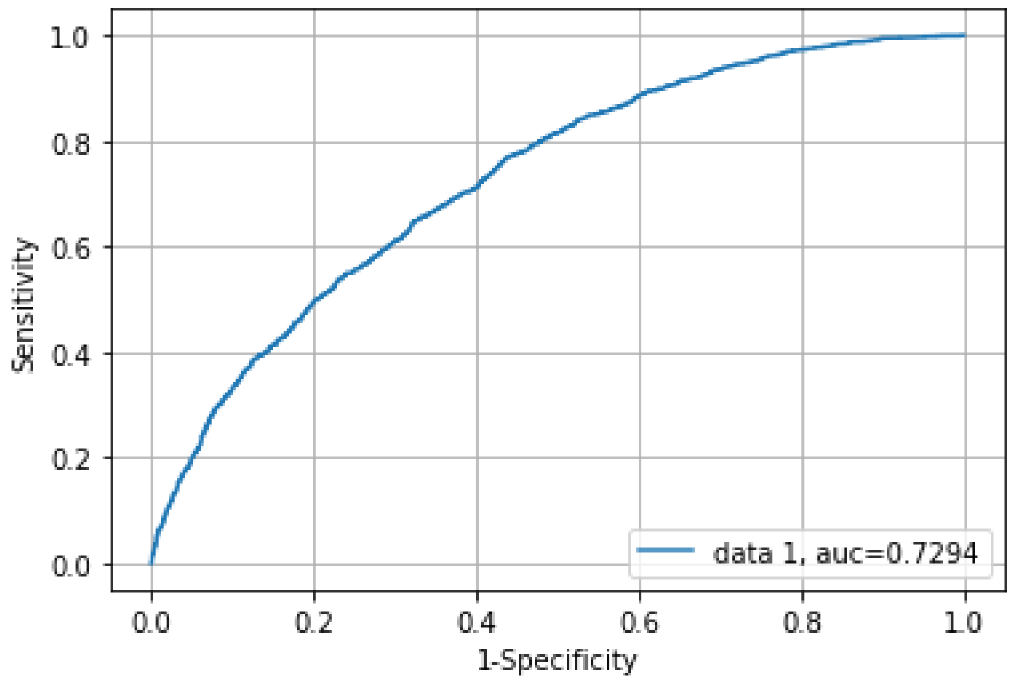
```
# Step 9: Make Predictions
y_pred = model.predict_classes(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix # y_test is going be rows (obs), y_pred (predicted) are cols
```

```
# Step 10: Evaluate usng accuracy, precision, recall
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)) # overall accuracy
print("Precision:",metrics.precision_score(y_test, y_pred)) # predicting 0 (Sat)
print("Recall:",metrics.recall_score(y_test, y_pred)) # predicting 1 (unsat)
```

```
# Step 11: Plot ROC Curve
y_pred_proba = model.predict_proba(X_test)
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(round(auc,4)))
plt.legend(loc=4)
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.grid()
plt.show()
```

```
# Step 12: Get model summary
model.summary() # get a model summary
wts = model.get_weights() # Get weights
np.savetxt('weight.csv' , wts , fmt='%s', delimiter=',')
model.save_weights('weights.hd5') #HDF format
```

Results



MLP – Confusion Matrix

	Predicted	
Obs	0	1
0	2755	750
1	1217	1264

5-12-1 ANN

Accuracy: 0.671
Precision: 0.628
Recall: 0.509

Comparison to Other Models

Model	Accuracy
Logistic	0.6657
Naïve Bayes	0.6557
KNN	0.6236
Ensemble	0.6676
Random Forest	0.676
AdaBoost	0.673

You Should Know

- What are MLPs
- What is TensorFlow
- What is Keras
- Elements of MLP
- Stochastic Gradient Descent
- Automatic Differentiation
- Implementing a Simple MLP using Keras with TensorFlow Backend