Machine Learning

Implementing MLPs II
Multinomial Classification

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
- MLP – Basic Implementation
  - Keras + Tensorflow
  - Binary Classifier

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

- Use other forms of classifiers
  - Multinomial classifiers
  - OneHot encoding
- Other ways to optimize ANN learning
  - Adam Method

Illustrative Examples

Multilayer Perceptron (Multinomial Classification)

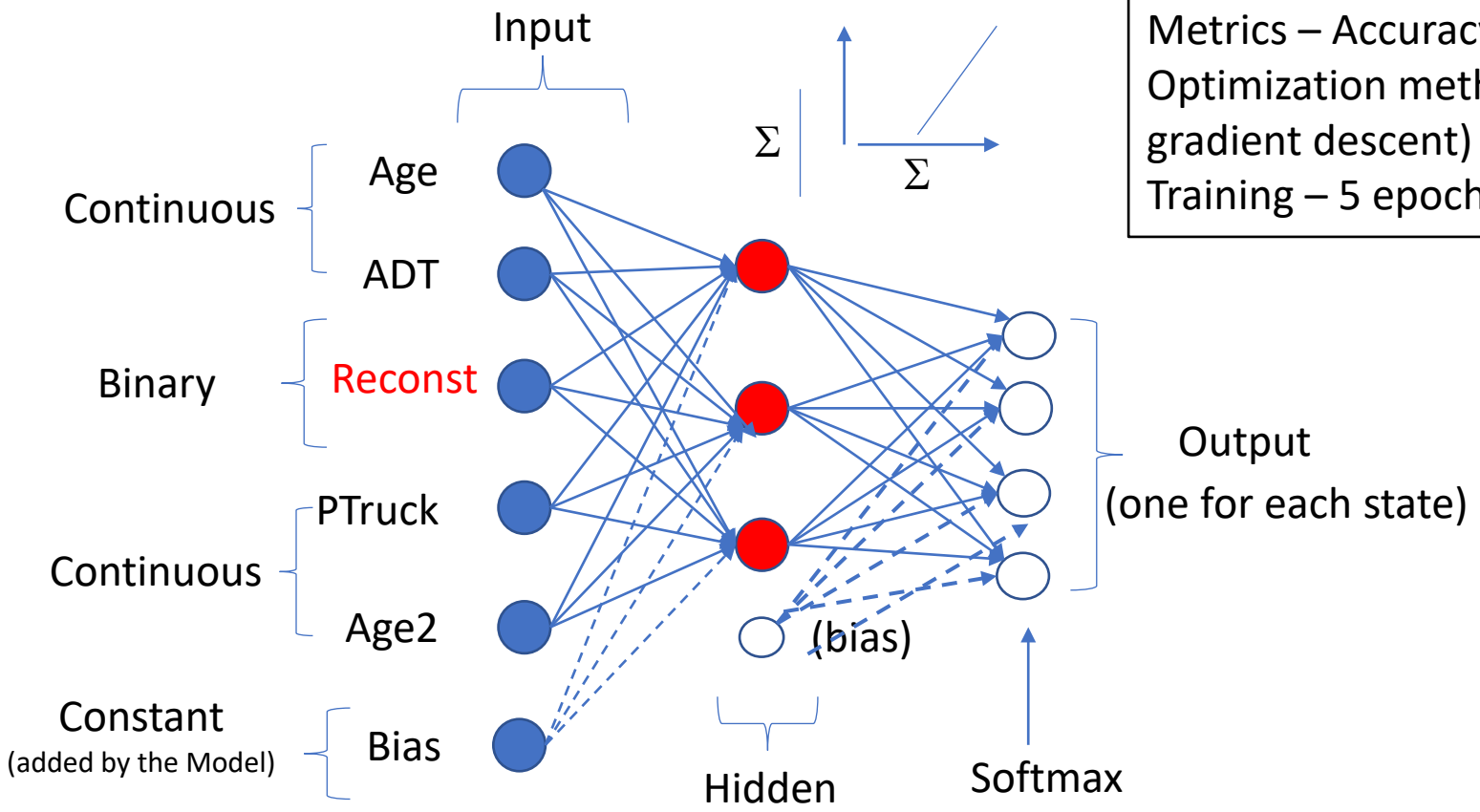
Illustrative Example

- Predicting damage to culverts in Texas
- Create a MLP model to classify into 4 states

	Code	Meaning	Description
Good	9	Excellent	As new
	8	Very Good	No problems noted.
	7	Good	Some minor problems.
Fair	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.
Serious	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.
Critical	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 – 4 states (4 nodes)
  - Needs a softmax activation layer

Performance Measure (loss) – 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 4 output
+ 4 bias term (output) = 16

Total # Weights = 34

OneHot Encoding

- For multi-scale classification you need to split your data such that there are 0 and 1 for each node
  - Number of nodes equal to Number of states
  - Each state is encoded with 0 or 1
    - Dummy Variables

State

State
0
1
2
3

Integer Encoding

State

0	1	2	3
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

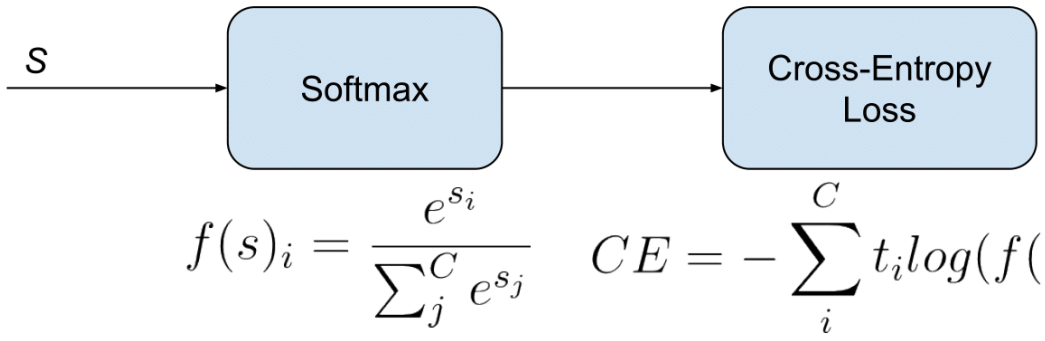
One Hot Encoding

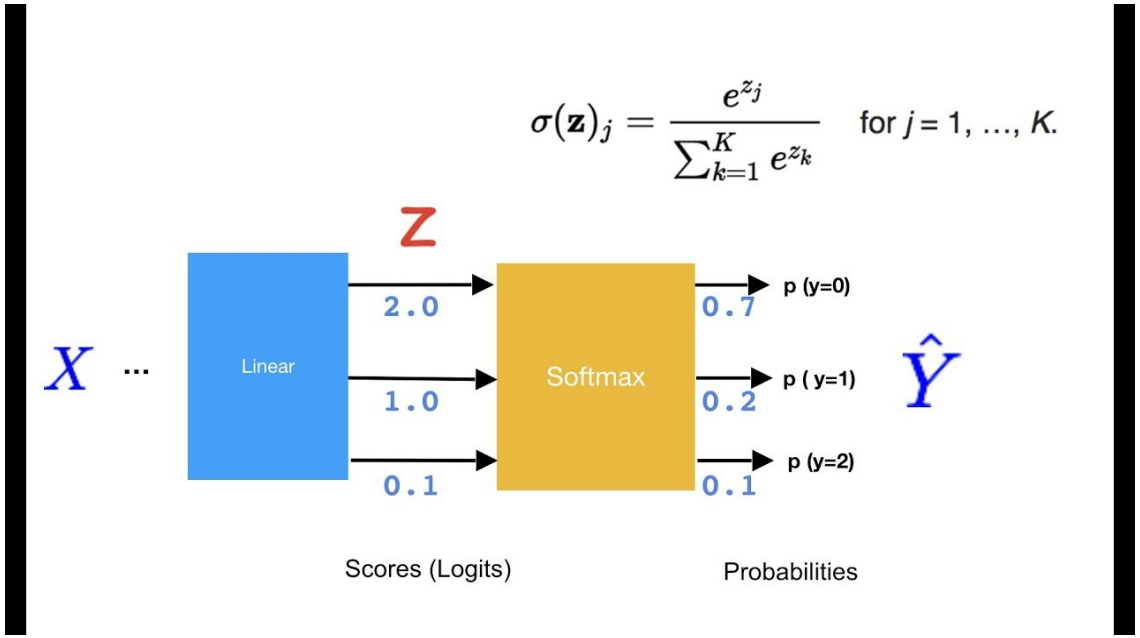
Keras has a utility that Allows Change from integer to One Hot Encoding

```
from keras.utils import np_utils
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
```

Conceptual Model

- Categorical Cross-Entropy
  - Also called ‘softmax’ loss
- Softmax activation function
  - Computes logit for each state
  - Takes a weight average
    - So the probabilities normalize to unity





The Output layer must have the softmax activation function

The hidden layers can have other types of activation functions

How many Hidden Nodes - Regularization

- Identifying hidden nodes is both science and art
  - Will require some trial-and-error experimentation
- We can add regularization to decay some weights
  - Small weights (decayed out to zero do not add much to the network)
- Regularization essentially adds an additional variable to the loss function (objective function being minimized)
- Keras allows regularization to be added for each layer
- Regularization can be applied on
  - Kernel Regularizer
    - Reduce all weights including bias terms
  - Activity Regularizer – Regularizes layer output
    - used for optimizing hidden nodes
  - Bias terms

Original Loss Function

$$L(x,y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

where $h_\theta x_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

L2 Regularizer

Here λ is a weighting factor (typically a small number 0.001)

- `kernel_regularizer` : instance of `keras.regularizers.Regularizer`
- `bias_regularizer` : instance of `keras.regularizers.Regularizer`
- `activity_regularizer` : instance of `keras.regularizers.Regularizer`

Optimization Methods

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

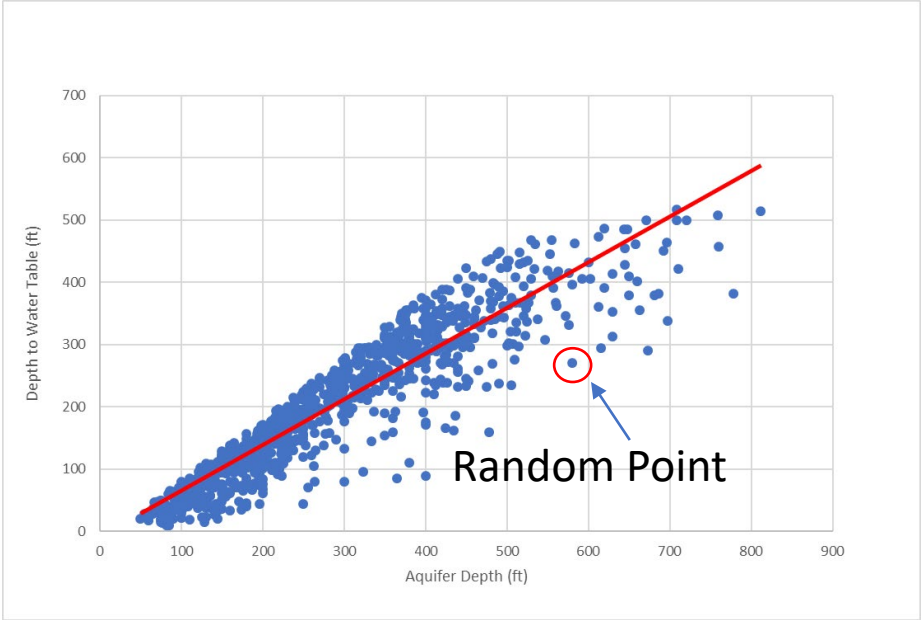
$$L = \sum (y_o - (mx + b))^2$$

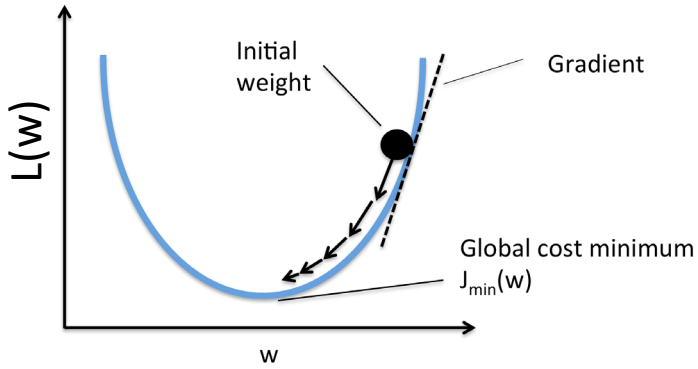
$$w_m^{t+1} = w_m^t + \eta \frac{\partial L}{\partial m}$$

$$w_b^{t+1} = w_b^t + \eta \frac{\partial L}{\partial b}$$

Weight Update Equations

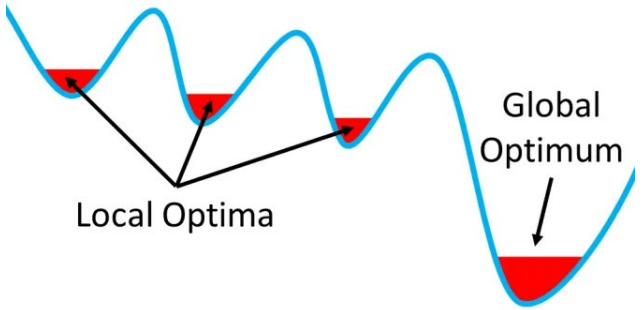
η Is the learning rate (user-specified)





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)

$$w_1 = x_1$$
$$w_2 = x_2$$
$$w_3 = w_1 w_2$$
$$w_4 = \sin(w_1)$$
$$w_5 = w_3 + w_4$$
$$z = w_5$$

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

Optimization Method - Adam

- Adam is a variant of stochastic gradient descent
  - Closely related to other variants
    - RMSProp
    - AdaMax
- Has three (four) hyperparameters
  - Initial step size (α)
  - Exponential Decay rates ($\beta1$,$\beta2$) for momentum
  - Error tolerance (ε)

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

ADAM Method

- Each parameter can have its own learning rate
- Gradients from previous steps are used to update weights (momentum)
  - In SGD only weights from previous steps are used
  - Use of gradients from previous steps tends to dampen oscillations
  - Older gradients are weighted less than recent gradients
- Squares of gradients are also used
  - Adds second moment (variance)
- The weights are second-order unbiased
  - The expected mean and (uncentered) variance of the weight must equal the unbiased mean and variance
- The initialization of weights (as zeros) adds bias
  - This needs to be corrected (which the algorithm does)
- Exponential smoothing is used to average previous and current gradient terms
- Initial learning rate is adapted over iterations
- Hyper-parameters for exponential decay are seldom changed from the recommendations
  - $\beta 1 = 0.9$ and $\beta 2 = 0.999$

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$$\eta : Initial\ Learning\ rate$$
$$g_t : Gradient\ at\ time\ t\ along\ \omega^j$$
$$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$$
$$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$$
$$\beta_1, \beta_2 : Hyperparameters$$

ADAM is called Adaptive Moment method

Adaptive because the learning rate (step size) of weights change

Moments because first and second moments are used

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 = 'C:\\Dropbox\\000CE5333Machine Learning\\Week9ANNS\\Code'
os.chdir(dir)
```

```
# Step 3: Read the dataset
a = pd.read_csv('TXculvertdata1.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['Class'] # ADD IT TO THE INPUT FEATURE DATAFRAME
```

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

```
# Step 5: Apply One Hot Encoding to the data
# Apply One Hot Encoding to the data
Y = a['Class'] # ADD IT TO THE INPUT FEATURE DATAFRAME
# Perform One Hot encoding
Y_he = tf.keras.utils.to_categorical(Y)
```

```
# Step 6: 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)
```

Python Code (Cont.)

```
# Step7: Setup Keras Model
model = tf.keras.Sequential([
tf.keras.layers.Dense(30, input_dim=5, activation='relu',  # Hidden layer
            kernel_regularizer=tf.keras.regularizers.l2(0.01)), # Add regularizer
tf.keras.layers.Dense(4, activation='softmax')]) # Output Layer
```

```
# Step 8: Compile Model
model.compile(loss='categorical_crossentropy',
                    metrics=['accuracy'],
                   optimizer='adam')
```

```
# Step 9: Fit the Model
history = model.fit(X_train, y_train, epochs=250,verbose=1, batch_size=128)
```

```
# Step 10: Plot History
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
```

```
# Step 11: Make Predictions
y_pred = model.predict(X_test)
y_classes = model.predict_proba(X_test)
y_testa = np.argmax(y_test,axis=-1)
y_preda = np.argmax(y_classes,axis=-1)
cm = confusion_matrix(y_target=y_testa,
                y_predicted=y_preda,
                binary=False)
cm
```

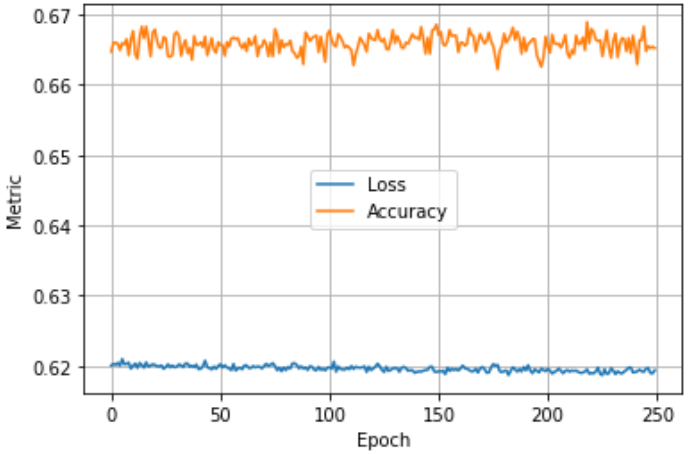
```
# Step 10: Evaluate accuracy
print("Accuracy:",metrics.accuracy_score(y_testa, y_preda)) # overall accuracy
```

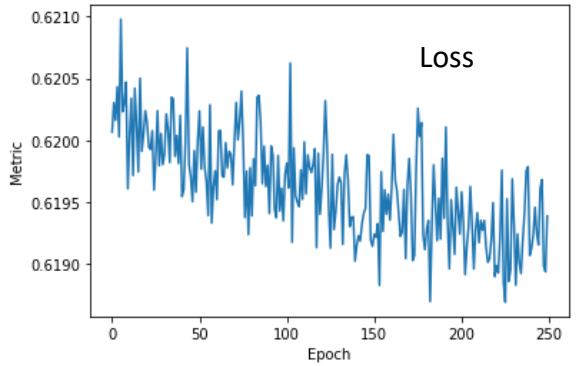
```
# Step 11: Get other metrics
model.summary() # get a model summary
wts = model.get_weights() # Get weights
np.savetxt('weight.csv' , wts , fmt='%s', delimiter=',')
model.save_weights('weights.hd5') #HDF format
```

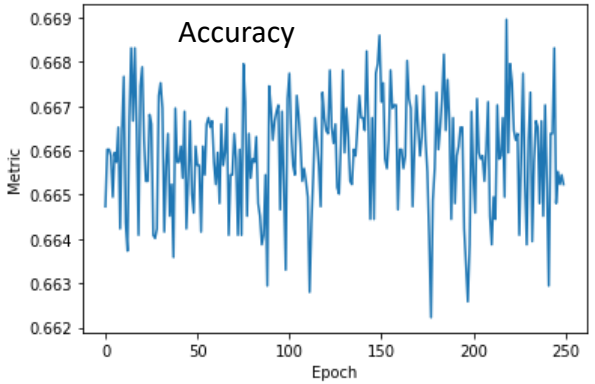
Results

- Model unable to predict serious and critical states
  - Too few exemplars (21 + 2) in testing data
  - Only 70 exemplars of ‘Serious’ state in entire data
    - 21 in testing
  - There are only 2 critical exemplars in the entire data so there were none in training

The ‘Good’ and ‘Fair’ states are oversampled in the dataset







		Predicted			
		Good	Fair	Serious	Critical
Observed	Good	2748	757	0	0
	Fair	1230	1228	0	0
	Serious	9	12	0	0
	Critical	0	2	0	0

You Should Know

- What are MLPs
- What is TensorFlow
- What is Keras
- Elements of MLP for a multiclass problem
- ADAM Method
- Automatic Differentiation
- Regularization
  - Weight regularization
- Implementing a Multiclass MLP using Keras with TensforFLow Backend