# Machine Learning: Introduction to Linear Regression, Logistic Regression, and Neural Networks

# 6.1 Tensorflow for MNIST Digits Classification

# Tensorflow for MNIST Digits Classification

Goal of this Section:

- Show how Tensorflow can be used for the MNIST Digits Classification problem

# What is Tensorflow?

From the Tensorflow website:

- 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.
- See website for details, examples, tutorials, etc https://www.tensorflow.org/
- Some key features:
  - Can easily set up neural networks, train, and predict
  - Version of Tensorflow optimized to make use of GPU chips to speed up training
- Many online resources available:
  - Courses on Udemy and Coursera
  - Tutorials on Youtube and various blog sites

# Tensorflow for MNIST Digits Classification

- This section shows how Tensorflow can be used for the MNIST Digits Classification problem discussed in the last chapter
- See:
  - Course Version: IntroML/Code/Version5.1/driver_neuralnetwork_mnist.py
  - Tensorflow Version: IntroML/Code/Version5.1/driver_tensorflow_mnist.py
- Recall components of driver:

1. Data loading/preparation
2. Neural Network Definition
3. Compilation
4. Training
5. Prediction

- Following slides show side-by-side comparison of course code and tensorflow versions for each component

# Code Comparison: Data Loading/Preparation

- Import
  - Course code: import NeuralNetwork
  - Tensorflow: import tensorflow as tf
- Data loading/preparation:
  - Course code: load_mnist outputs feature matrices (# features x # samples) and label vectors (1 x # samples)
  - Tensorflow: requires samples axis to be along rows so take transpose using .T functionality for numpy arrays

Course Code Driver

```
import load_mnist
import NeuralNetwork
import matplotlib.pyplot as plt
import numpy as np
import metrics
import plot_results
import time

# (1) Set up data
ntrain = 6000
nvalid = 1000
nclass = 10
Xtrain,Ytrain,Xvalid,Yvalid = load_mnist.load_mnist(ntrain,nvalid)
```

Tensorflow Driver

```
import load_mnist
import matplotlib.pyplot as plt
import metrics
import numpy as np
import tensorflow as tf
import time

# (1) Set up data
ntrain = 6000
nvalid = 1000
nclass = 10
Xtrain,Ytrain,Xvalid,Yvalid = load_mnist.load_mnist(ntrain,nvalid)
# take transpose of inputs for tensorflow  - sample axis along rows
XtrainT = Xtrain.T
YtrainT = Ytrain.T
XvalidT = Xvalid.T
YvalidT = Yvalid.T
```

# Code Comparison: Neural Network Definition

- Neural Network Definition Tensorflow:
  - tf.keras.models.Sequential method is used to build neural network as a sequence of layers
  - tf.keras.layers.Dense is equivalent to the add_layer method from this course
  - input_shape(784, ) in first layer defines number of features
  - 128 & nclass are number of units
  - kernel_regulizer defines regularization (can use l1 or l2) and lamb is coefficient
  - activation specifies activation function

Course Code Driver

```
# (2) Define model
np.random.seed(10)
lamb = 0.0
model = NeuralNetwork.NeuralNetwork(784)
model.add_layer(128,"tanh",lamb)
model.add_layer(nclass,"softmax",lamb)
```

Tensorflow Driver

```
# (2) Define model
lamb = 0.0
model = tf.keras.models.Sequential([
tf.keras.layers.Dense(128,input_shape=(784,), activation="tanh", kernel_regularizer=tf.keras.regularizers.l2(lamb)),
tf.keras.layers.Dense(nclass,activation="softmax", kernel_regularizer=tf.keras.regularizers.l2(lamb))])
# (3) Compile model
```

# Code Comparison: Compilation

- Neural Network Definition Tensorflow:
  - Adam optimizer specified by tf.keras.optimizers.Adam function
  - Loss function for multi-class classification  specified as “sparse_categorical_crossentropy
  - Must specify “accuracy” in metrics input to ensure accuracy is computed for each epoch

Course Code Driver

```
# (3) Compile model
optimizer = {"method": "Adam", "learning_rate": 0.02, "beta1": 0.9, "beta2": 0.999, "epsilon": 1e-7}
model.compile("crossentropy",optimizer)
model.summary()
```

Tensorflow Driver

```
# (3) Compile model
optimizer = tf.keras.optimizers.Adam(lr=0.02, beta_1=0.9, beta_2=0.999, epsilon=1e-7)
model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

# Code Comparison: Compilation

- Training in Tensorflow:
  - Use fit method to perform training
  - Same inputs as in course code (training data, epochs, batch_size, validation_data)
  - Use transposed feature matrix and label vector for training dataset
  - Use transposed feature matrix and label vector for validation dataset

Course Code Driver

```
# (4) Train model
epochs = 40
time_start = time.time()
history = model.fit(Xtrain,Ytrain,epochs,batch_size=ntrain,validation_data=(Xvalid,Yvalid))
time_end = time.time()
print("Train time: {}".format(time_end - time_start))
```

Tensorflow Driver

```
# (4) Train model
epochs = 40
time_start = time.time()
history = model.fit(trainT,YtrainT,epochs=epochs,batch_size=ntrain,validation_data=(XvalidT,YvalidT))
time_end = time.time()
print("Train time: {}".format(time_end - time_start))
```

# Code Comparison: Prediction

- Prediction in Tensorflow:
  - Use predict method to predict results for validation data set
  - predict method outputs result of activation $A^{[2]}$ (#samples x 10) at final layer
    - Use argmax to find predicted class
    - Yvalid_pred: Use expand_dims to convert to (1 x #samples) matrix
  - history.history is dictionary of results from training
    - "loss"/"acc"  has history of training dataset loss/accuracy
    - "val_loss"/"val_acc" has history of validation dataset loss/accuracy

Course Code Driver

```
# (5) Predictions and plotting
# confusion matrix
Yvalid_pred = model.predict(Xvalid)
metrics.confusion_matrix(Yvalid,Yvalid_pred,nclass)
# plot loss, accuracy, and animation of results
plot_results.plot_results_history(history,["loss","valid_loss"])
plot_results.plot_results_history(history,["accuracy","valid_accuracy"])
plot_results.plot_results_mnist_animation(Xvalid,Yvalid,Yvalid_pred,100)
plt.show()
```

Tensorflow Driver

```
# (5) Predictions and plotting
Yvalid_pred = np.expand_dims(np.argmax(model.predict(XvalidT),axis=1),axis=0)
metrics.confusion_matrix(Yvalid,Yvalid_pred,nclass)
# plot loss, accuracy, and animation of results
plot_results.plot_results_history(history.history,["loss","val_loss"])
plot_results.plot_results_history(history.history,["acc","val_acc"])
plot_results.plot_results_mnist_animation(Xvalid,Yvalid,Yvalid_pred,100)
plt.show()
```

# 6.2 Demo of Tensorflow on a GPU