Introduction to Machine Learning: Linear and Logistic Regression and Neural Networks Using Python

Chapter 8 Introduction to Tensorflow

Machine Learning Frameworks

- A machine learning framework consists of building blocks for designing, training and validating deep neural networks, through a high level programming interface.
- There are many machine learning frameworks, including Tensorflow, Pytorch, Matlab, sklearn, etc. See following link for a listing:

https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software

- Why go through all the trouble of learning the math and algorithms and then building a basic framework from scratch in this course, when one can use one of these frameworks “off the shelf”?
- As mentioned at the beginning of the course, it is difficult to truly understand what a framework is doing without going through the math and algorithms and doing the coding.

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
  - There is a version of Tensorflow optimized to make use of GPU chips, that are faster than CPU for training
- Many online resources available:
  - Courses on Udemy and Coursera
  - Tutorials on Youtube and various blog sites

Introduction to Tensorflow

Section	Title	Description
8.1	Tensorflow for MNIST Digits Classification	This section shows how to use Tensorflow for the MNIST digits classification problem. We will create a driver analogous to driver_casestudy_mnist.
8.2	Demo of Tensorflow on a GPU	This section present a demo of running Tensorflow on a GPU for the MNIST digits classification problem

8.1 Tensorflow for MNIST Digits Classification

Tensorflow for MNIST Digits Classification

Goal of this Section:

- Show how Tensorflow can be used for MNIST Digits Classification

Tensorflow for MNIST Digits Classification

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 drivers
  - Course Driver: driver_casestudy_mnist.py
  - Tensorflow Driver: driver_casestudy_mnist_tensorflow.py

Import Packages

Course Code Driver

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

Tensorflow Driver

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

Code Comparison: Data Loading/Preparation

- 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 (feature matrix should be #samples x # features) so take transpose using .T functionality for numpy arrays

Course Code Driver

```
# (1) Set up data
ntrain = 60000
nvalid = 10000
nclass = 10
Xtrain,Ytrain,Xvalid,Yvalid = load_mnist.load_mnist(ntrain,nvalid)
```

Tensorflow Driver

```
# (1) Set up data
ntrain = 60000
nvalid = 10000
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(nfeature,) in first layer defines number of features
  - 128 & nclass are number of units
  - kernel_regularizer defines regularization (can use l1 or l2) and lamb is coefficient
  - activation specifies activation function

Course Code Driver

```
# (2) Define model
nfeature = Xtrain.shape[0]
np.random.seed(10)
lamb = 0.0
model = NeuralNetwork.NeuralNetwork(nfeature)
model.add_layer(128,"relu",lamb)
model.add_layer(nclass,"softmax",lamb)
```

Tensorflow Driver

```
# (2) Define model
nfeature = Xtrain.shape[0]
lamb = 0.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, input_shape=(nfeature,), activation="relu",kernel_regularizer=tf.keras.regularizers.l2(lamb)),
    tf.keras.layers.Dense(nclass,activation="softmax",kernel_regularizer=tf.keras.regularizers.l2(lamb))])
```

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
  - summary() produces summary of layers and number of parameters

Course Code Driver

```
# (3) Compile model
optimizer = Optimizer.Adam(0.02,0.9,0.999,1e-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: Training

- 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
    - Take transpose and use onehot_inverse to compute class for each sample
  - history.history is dictionary of results from training
    - “loss”/”accuracy” has history of training dataset loss/accuracy
    - “val_loss”/”val_accuracy” has history of validation dataset loss/accuracy

Course Code Driver

```
# (5) Predictions and plotting
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,
    model.get_Afinal(),100)
plt.show()
```

Tensorflow Driver

```
Afinal = model.predict(XvalidT).T
Yvalid_pred = onehot.onehot_inverse(Afinal)
plot_results.plot_results_history(history.history,["loss","val_loss"])
plot_results.plot_results_history(history.history,["accuracy","val_accuracy"])
plot_results.plot_results_mnist_animation(Xvalid,Yvalid,Yvalid_pred,Afinal,100)
plt.show()
```

Timings for MNIST Digits Classification

Run	Machine	Chip	Code	Training Time 60K Train samples 10K Valid samples	Accuracy Train	Accuracy Valid
1	My Windows	CPU: Intel i5-6200 2.30GHz	Course Framework	65.2 seconds	91.5%	91.6%
2	My Windows	CPU: Intel i5-6200 2.30GHz	Tensorflow2.1	24.3 seconds	94.9%	94.6%

New Code for MNIST Using Tensorflow

Function/component	Input	Description
driver_casestudy_ mnist_tensorflow		Driver

8.1 Tensorflow for MNIST Classification DEMO

- Code located at: IntroML/Code/Version5.1
- Course Driver: driver_casestudy_mnist.py
- Tensorflow Driver: driver_casestudy_mnist_tensorflow.py

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

8.2 Demo of Tensorflow on a GPU

Demo of Tensorflow on a GPU

Goal of this Section:

- Perform a Demo of Tensorflow on a GPU

What is a GPU and Why Use for Machine Learning?

- A GPU (Graphical Processing Unit) is a specialized chip designed for image processing (used for gaming)
- Parallel structure of GPU make them more efficient than CPU for algorithms that process data in parallel
- Source:

https://en.wikipedia.org/wiki/Graphics_processing_unit

- Machine learning frameworks have been developed to take advantage of the features of GPU and run much faster than on a CPU

NVIDIA GPU and Machine Learning Frameworks

- NVIDIA (GPU chip company) provides GPU support for number of machine learning frameworks including: Tensorflow, Pytorch, Matlab, etc. See

https://developer.nvidia.com/deep-learning-frameworks

- See following for instructions on installing GPU version of Tensorflow

https://www.tensorflow.org/install/gpu

Tensorflow on a GPU

- In previous section, we showed a Tensorflow code for MNIST classification
- No change to code is required to run on GPU
- In Demo will run Tensorflow on Azure VM with/without GPU
  - Tensorflow will automatically run with GPU turned on
  - Add following before import of tensorflow to turn off GPU and use CPU

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

Results MNIST Digits Classification

Run	Machine	Chip	Code	Training Time 40 Epochs 60K Train samples 10K Valid samples	Accuracy Train	Accuracy Valid
1	My Windows	CPU: Intel i5-6200 2.30GHz	Course Framework	65.2 seconds	91.2%	90.6%
2	My Windows	CPU: Intel i5-6200 2.30GHz	Tensorflow2.1	24.3 seconds	93.2%	92.9%
3	Azure VM	CPU: Intel Xeon E5-2690 v3 2.60GHz	Course Framework	45.0 seconds	91.2%	90.6%
4	Azure VM	CPU: Intel Xeon E5-2690 v3 2.60GHz	Tensorflow2.1	13.5 seconds	93.2%	92.7%
5	Azure VM	GPU: NVIDIA Tesla M60	Tensorflow2.1	7.9 seconds	93.5%	93.3%