# Week1-MNIST_Basic_Example

## 1 MNIST DataSet

Let's use Tensorflow2.0 to apply some basic Machine Learning Algorithms to MNIST dataset. First, let's look at some of the native low level Tensorflow modules to load data. We will see later how this is can also be done directly with Keras.

```
In [7]: #import the library and dataset
        import matplotlib.pyplot as plt
        import numpy as np
        import tensorflow as tf
        import tensorflow_datasets as tfds
```

```
In [5]: #Load the train dataset.
        mnist_train = tfds.load(name="mnist", split="train")
        assert isinstance(mnist_train, tf.data.Dataset)
        print(mnist_train)
```

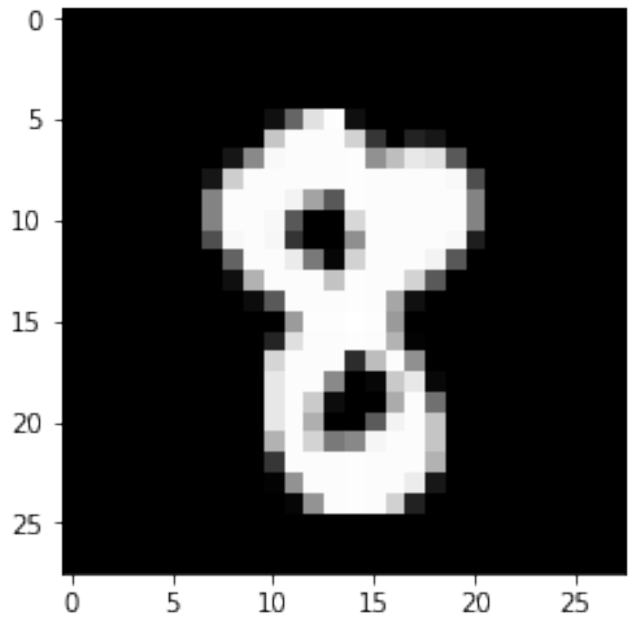
```
Downloading and preparing dataset mnist (11.06 MiB) to /Users/user1/tensorflow_datasets/mnist/


WARNING: Logging before flag parsing goes to stderr.
W0117 22:18:38.615947 4469585344 dataset_builder.py:316] Dataset mnist is hosted on GCS. It wi
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead set
data_dir=gs://tfds-data/datasets.



HBox(children=(IntProgress(value=0, description='Dl Completed...', max=19, style=ProgressStyle



Dataset mnist downloaded and prepared to /Users/user1/tensorflow_datasets/mnist/1.0.0. Subsequ
<_OptionsDataset shapes: {image: (28, 28, 1), label: ()}, types: {image: tf.uint8, label: tf.i
```

All tfds datasets contain feature dictionaries mapping feature names to Tensor values. A typical dataset, like MNIST, will have 2 keys: "image" and "label". Below we inspect a single example.

Note: In graph mode, see the tf.data guide to understand how to iterate on a tf.data.Dataset.

```
In [8]: # Checkout what anyone image looks like.
        for mnist_example in mnist_train.take(1):  # Only take a single example
          image, label = mnist_example["image"], mnist_example["label"]

          plt.imshow(image.numpy()[:, :, 0].astype(np.float32), cmap=plt.get_cmap("gray"))
          print("Label: %d" % label.numpy())
```

```
Label: 8
```



## 1.1 DatasetBuilder

tfds.load is really a thin conveninence wrapper around DatasetBuilder. We can accomplish the same as above directly with the MNIST DatasetBuilder.

```
In [11]: mnist_builder = tfds.builder("mnist")
         mnist_builder.download_and_prepare()
         mnist_train = mnist_builder.as_dataset(split="train")
         mnist_train
```

```
Out[11]: <_OptionsDataset shapes: {image: (28, 28, 1), label: ()}, types: {image: tf.uint8, lal
```

## 1.2 Input pipelines

Once you have a tf.data.Dataset object, it's simple to define the rest of an input pipeline suitable for model training by using the tf.data API. Repeat the dataset so that we have an infinite stream of examples, shuffle, and create batches

```
In [10]: mnist_train = mnist_train.repeat().shuffle(1024).batch(32)

         # prefetch will enable the input pipeline to asynchronously fetch batches while
         # your model is training.
         mnist_train = mnist_train.prefetch(tf.data.experimental.AUTOTUNE)

         # Now you could loop over batches of the dataset and train
         # for batch in mnist_train:
         #   ...
```

## 1.3 DatasetInfo

After generation, the builder contains useful information on the dataset:

```
In [13]: info = mnist_builder.info
         print(info)
```

```
tfds.core.DatasetInfo(
    name='mnist',
    version=1.0.0,
    description='The MNIST database of handwritten digits.',
    homepage='http://yann.lecun.com/exdb/mnist/',
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
    },
    supervised_keys=('image', 'label'),
    citation="""@article{lecun2010mnist,
      title={MNIST handwritten digit database},
      author={LeCun, Yann and Cortes, Corinna and Burges, CJ},
      journal={ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist},
      volume={2},
      year={2010}
    }""",
    redistribution_info=,
)
```

DatasetInfo also contains useful information about the features:

```
In [14]: print(info.features)
         print(info.features["label"].num_classes)
         print(info.features["label"].names)
```

```
FeaturesDict({
    'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
})
10
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

You can also load the DatasetInfo directly with tfds.load using with_info=True.

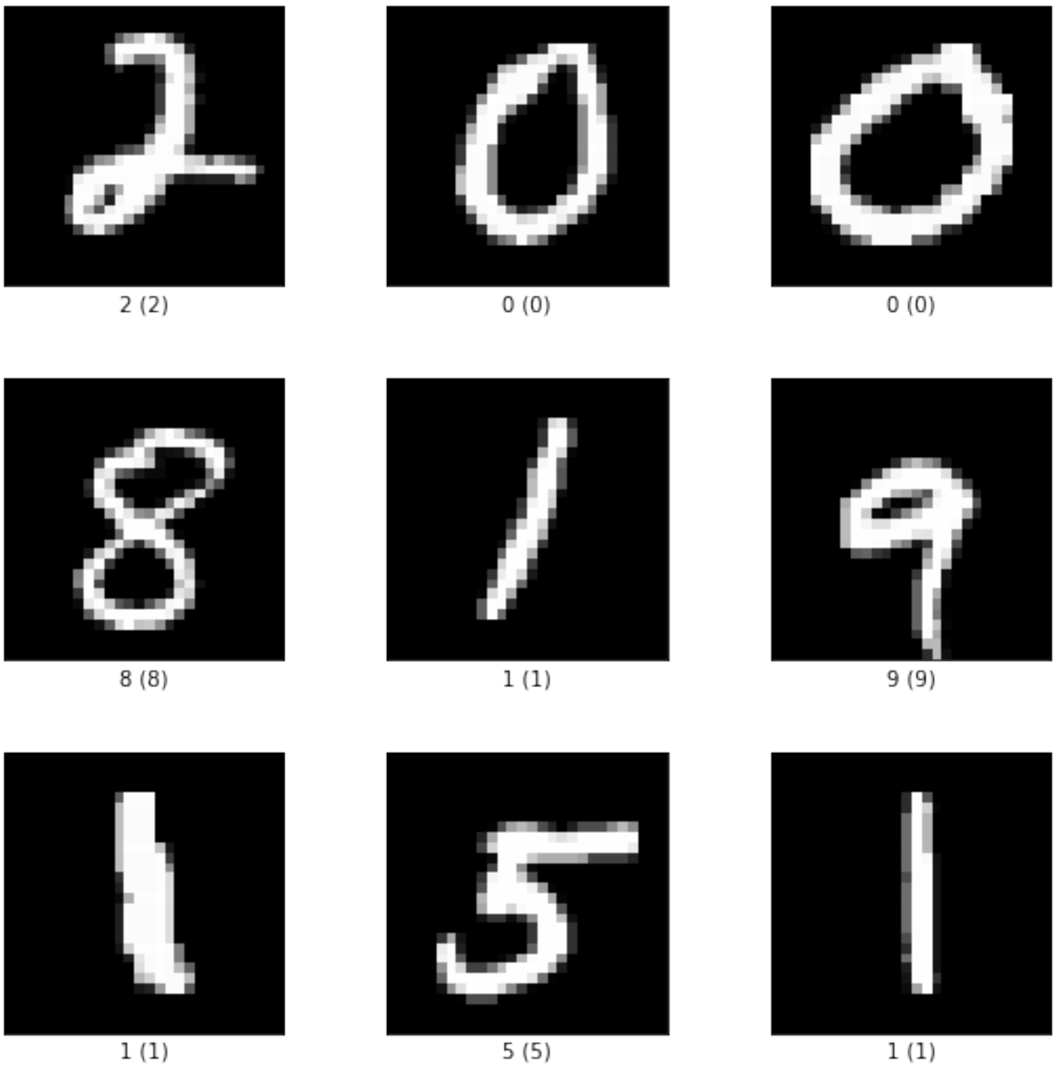
```
In [15]: mnist_test, info = tfds.load("mnist", split="test", with_info=True)
         print(info)
```

```
tfds.core.DatasetInfo(
    name='mnist',
    version=1.0.0,
    description='The MNIST database of handwritten digits.',
    homepage='http://yann.lecun.com/exdb/mnist/',
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
    },
    supervised_keys=('image', 'label'),
    citation="""@article{lecun2010mnist,
      title={MNIST handwritten digit database},
      author={LeCun, Yann and Cortes, Corinna and Burges, CJ},
      journal={ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist},
      volume={2},
      year={2010}
    }""",
    redistribution_info=,
)
```

## 1.4 Visualization

For image classification datasets, you can use tfds.show_examples to display some examples.

```
In [16]: fig = tfds.show_examples(info, mnist_test)
```



```
In [17]: mnist_train.

Out[17]: <function tensorflow.python.data.ops.dataset_ops.DatasetV2.list_files(file_pattern, sl
```

## 2 Logistic Regression Example

Logistic regression implementation with TensorFlow v2 library.

## 2.1 MNIST Dataset Overview

This example is using MNIST handwritten digits. The dataset contains 60,000 examples for training and 10,000 examples for testing. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 255.

In this example, each image will be converted to float32, normalized to [0, 1] and flattened to a 1-D array of 784 features (28*28).

More info: http://yann.lecun.com/exdb/mnist/

```
In [18]: import tensorflow as tf
         import numpy as np
```

```
In [55]: # MNIST dataset parameters.
         num_classes = 10 # 0 to 9 digits
         num_features = 784 # 28*28

         # Training parameters.
         learning_rate = 0.001
         training_steps = 500
         batch_size = 256
         display_step = 100
```

Keras is a wrapper to run the highlevel tensorflow operations easily. Tensorflow2.0 has Keras library integrated as a module and this makes all the operations easier to execute including loading datasets through Keras ..

```
In [76]: # Prepare MNIST data.
         from tensorflow.keras.datasets import mnist
         (x_train, y_train), (x_test, y_test) = mnist.load_data()

         # Convert to float32.
         x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)

         # Flatten images to 1-D vector of 784 features (28*28).
         x_train, x_test = x_train.reshape([-1, num_features]), \
                                x_test.reshape([-1, num_features])

         # Normalize images value from [0, 255] to [0, 1].
         x_train, x_test = x_train / 255., x_test / 255.
```

```
In [21]: # Use tf.data API to shuffle and batch data.
         train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
         train_data = train_data.repeat().shuffle(5000).batch(batch_size).prefetch(1)
```

```
In [22]: # Weight of shape [784, 10], the 28*28 image features, and total number of classes.
         W = tf.Variable(tf.ones([num_features, num_classes]), name="weight")
         # Bias of shape [10], the total number of classes.
         b = tf.Variable(tf.zeros([num_classes]), name="bias")
```

```
        # Logistic regression (Wx + b).
        def logistic_regression(x):
            # Apply softmax to normalize the logits to a probability distribution.
            return tf.nn.softmax(tf.matmul(x, W) + b)

        # Cross-Entropy loss function.
        def cross_entropy(y_pred, y_true):
            # Encode label to a one hot vector.
            y_true = tf.one_hot(y_true, depth=num_classes)
            # Clip prediction values to avoid log(0) error.
            y_pred = tf.clip_by_value(y_pred, 1e-9, 1.)
            # Compute cross-entropy.
            return tf.reduce_mean(-tf.reduce_sum(y_true * tf.math.log(y_pred)))

        # Accuracy metric.
        def accuracy(y_pred, y_true):
            # Predicted class is the index of highest
            # score in prediction vector (i.e. argmax).
            correct_prediction = tf.equal(tf.argmax(y_pred, 1),
                                          tf.cast(y_true, tf.int64))
            return tf.reduce_mean(tf.cast(
                correct_prediction, tf.float32))

        # Stochastic gradient descent optimizer.
        optimizer = tf.optimizers.SGD(learning_rate)
```

```
In [23]: # Optimization process.
        def run_optimization(x, y):
            # Wrap computation inside a GradientTape for automatic differentiation.
            with tf.GradientTape() as g:
                pred = logistic_regression(x)
                loss = cross_entropy(pred, y)

            # Compute gradients.
            gradients = g.gradient(loss, [W, b])

            # Update W and b following gradients.
            optimizer.apply_gradients(zip(gradients, [W, b]))
```

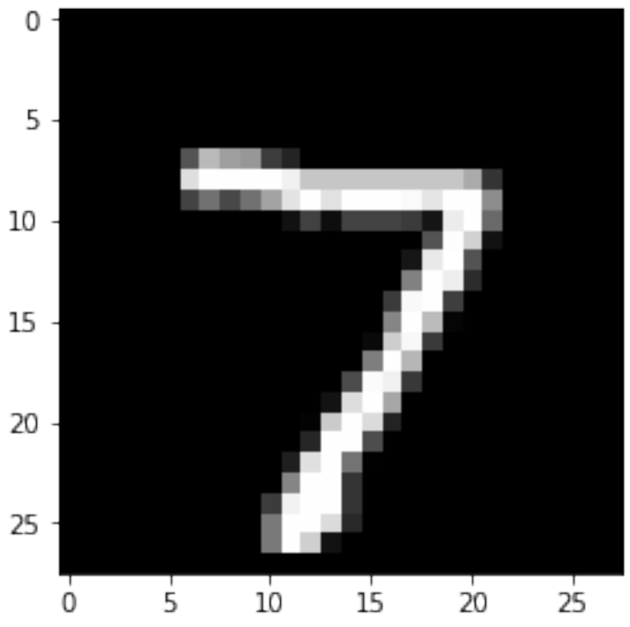
```
In [71]: # Run training for the given number of steps.
        for step, (batch_x, batch_y) in enumerate(train_data.take(training_steps), 1):
            # Run the optimization to update W and b values.
            run_optimization(batch_x, batch_y)

            if step % display_step == 0:
                pred = logistic_regression(batch_x)
                loss = cross_entropy(pred, batch_y)
                acc = accuracy(pred, batch_y)
                print("step: %i, loss: %f, accuracy: %f" % (step, loss, acc))
```

```
step: 100, loss: 392.648926, accuracy: 0.882812
step: 200, loss: 465.223419, accuracy: 0.878906
step: 300, loss: 500.419891, accuracy: 0.871094
step: 400, loss: 699.341919, accuracy: 0.750000
step: 500, loss: 469.810974, accuracy: 0.882812
```

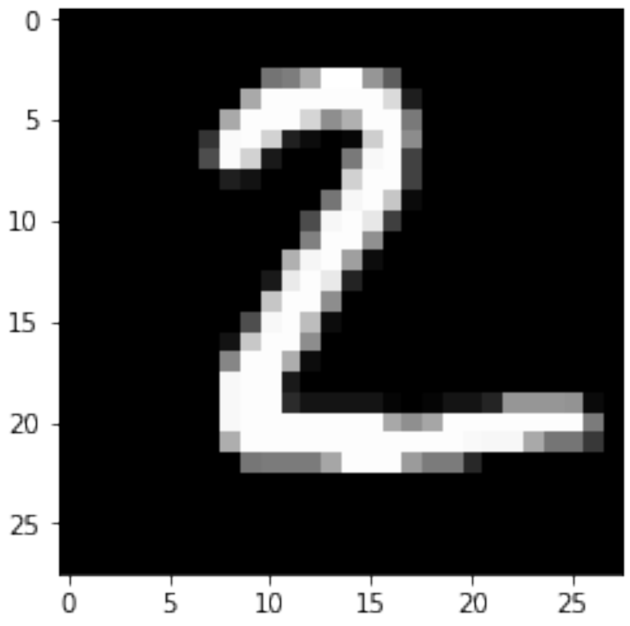
```
In [72]: # Test model on validation set.
         pred = logistic_regression(x_test)
         print("Test Accuracy: %f" % accuracy(pred, y_test))
```

```
Test Accuracy: 0.840600
```

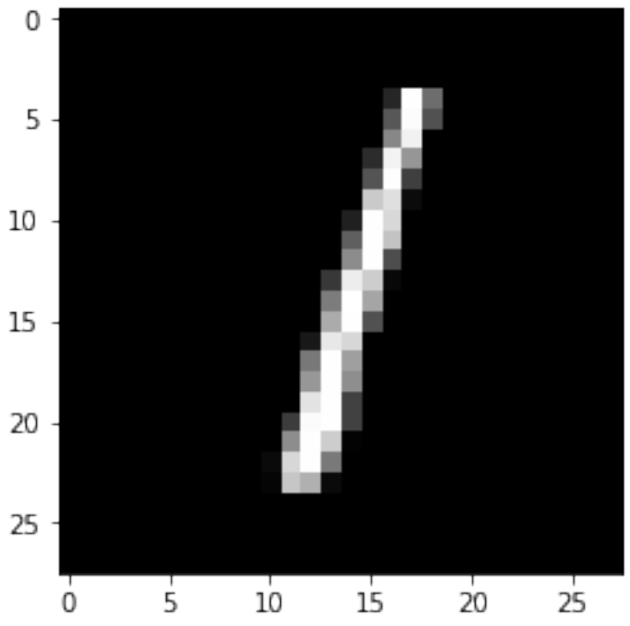
```
In [73]: # Predict 5 images from validation set.
         n_images = 5
         test_images = x_test[:n_images]
         predictions = logistic_regression(test_images)

         # Display image and model prediction.
         for i in range(n_images):
             plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray')
             plt.show()
             print("Model prediction: %i" % np.argmax(predictions.numpy()[i]))
```



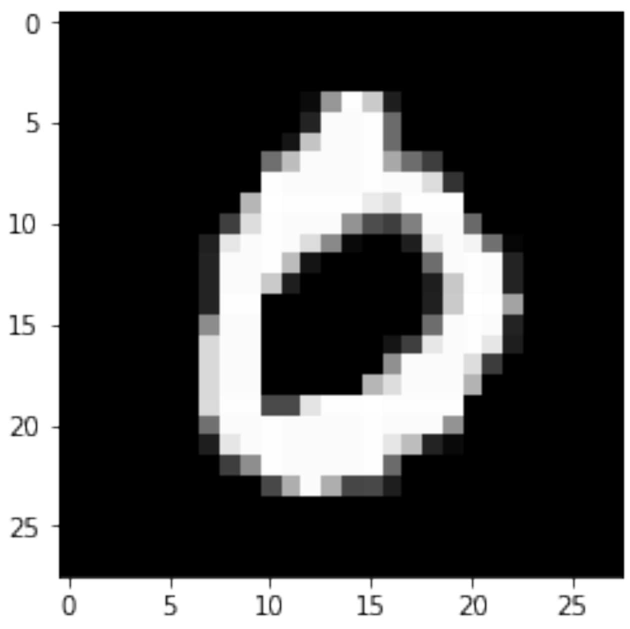
```
Model prediction: 7
```



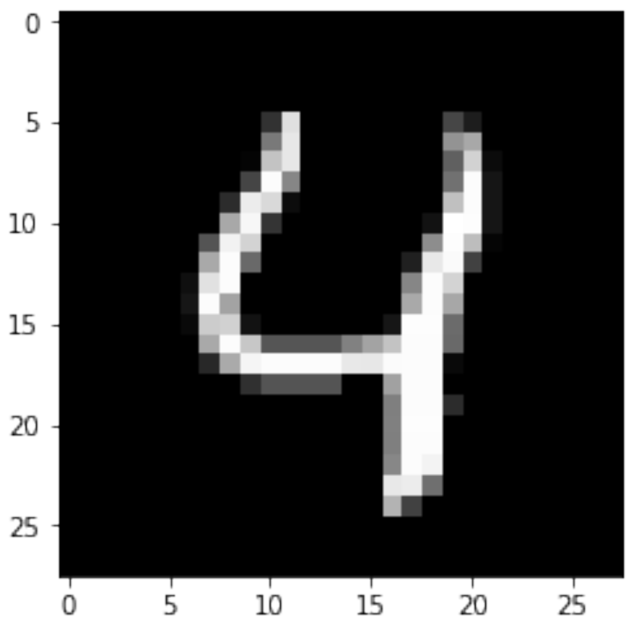
```
Model prediction: 2
```



```
Model prediction: 1
```



```
Model prediction: 0
```



```
Model prediction: 4
```