# Assignment1.1 Sutow Brett

June 8, 2021

[1]:
```
#Example One#
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
```

[2]:
```
batch_size = 128
num_classes = 10
epochs = 20
```

[3]:
```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
60000 train samples
10000 test samples
```

[4]:
```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
```

```
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 5s 11ms/step - loss: 0.4359 -
accuracy: 0.8619 - val_loss: 0.1130 - val_accuracy: 0.9647
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1104 -
accuracy: 0.9670 - val_loss: 0.0888 - val_accuracy: 0.9726
Epoch 3/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0727 -
accuracy: 0.9775 - val_loss: 0.0709 - val_accuracy: 0.9792
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0574 -
accuracy: 0.9826 - val_loss: 0.0847 - val_accuracy: 0.9772
```

```
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0464 -
accuracy: 0.9855 - val_loss: 0.0785 - val_accuracy: 0.9814
Epoch 6/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0409 -
accuracy: 0.9880 - val_loss: 0.0795 - val_accuracy: 0.9796
Epoch 7/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0364 -
accuracy: 0.9899 - val_loss: 0.0822 - val_accuracy: 0.9816
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0327 -
accuracy: 0.9905 - val_loss: 0.0865 - val_accuracy: 0.9830
Epoch 9/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0281 -
accuracy: 0.9913 - val_loss: 0.0799 - val_accuracy: 0.9826
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0248 -
accuracy: 0.9929 - val_loss: 0.0844 - val_accuracy: 0.9836
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0237 -
accuracy: 0.9930 - val_loss: 0.0879 - val_accuracy: 0.9842
Epoch 12/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0232 -
accuracy: 0.9932 - val_loss: 0.0978 - val_accuracy: 0.9802
Epoch 13/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0238 -
accuracy: 0.9934 - val_loss: 0.0949 - val_accuracy: 0.9833
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0191 -
accuracy: 0.9948 - val_loss: 0.0899 - val_accuracy: 0.9829
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0185 -
accuracy: 0.9946 - val_loss: 0.1074 - val_accuracy: 0.9821
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0179 -
accuracy: 0.9950 - val_loss: 0.1126 - val_accuracy: 0.9824
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0168 -
accuracy: 0.9956 - val_loss: 0.1350 - val_accuracy: 0.9826
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0186 -
accuracy: 0.9950 - val_loss: 0.1266 - val_accuracy: 0.9823
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0173 -
accuracy: 0.9953 - val_loss: 0.1194 - val_accuracy: 0.9822
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0144 -
accuracy: 0.9957 - val_loss: 0.1305 - val_accuracy: 0.9832
```

```
Test loss: 0.1305491179227829
Test accuracy: 0.9832000136375427
```

[6]:
```
#Example Two#
import sys
from random import random
from operator import add

from pyspark.sql import SparkSession
```

[14]:
```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "communist-label",
   "metadata": {},
   "outputs": [],
   "source": [
    "#Example One#\n",
    "from tensorflow import keras\n",
    "from tensorflow.keras.datasets import mnist\n",
    "from tensorflow.keras.models import Sequential\n",
    "from tensorflow.keras.layers import Dense, Dropout\n",
    "from tensorflow.keras.optimizers import RMSprop\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "aboriginal-reproduction",
   "metadata": {},
   "outputs": [],
   "source": [
    "batch_size = 128\n",
    "num_classes = 10\n",
    "epochs = 20\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "mediterranean-wagner",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
```

```
    "output_type": "stream",
    "text": [
     "Downloading data from https://storage.googleapis.com/tensorflow/
↪tf-keras-datasets/mnist.npz\n",
     "11493376/11490434 [==============================] - 1s 0us/step\n",
     "60000 train samples\n",
     "10000 test samples\n"
    ]
   }
  ],
  "source": [
   "# the data, split between train and test sets\n",
   "(x_train, y_train), (x_test, y_test) = mnist.load_data()\n",
   "\n",
   "x_train = x_train.reshape(60000, 784)\n",
   "x_test = x_test.reshape(10000, 784)\n",
   "x_train = x_train.astype('float32')\n",
   "x_test = x_test.astype('float32')\n",
   "x_train /= 255\n",
   "x_test /= 255\n",
   "print(x_train.shape[0], 'train samples')\n",
   "print(x_test.shape[0], 'test samples')\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 4,
  "id": "second-jason",
  "metadata": {},
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Model: \"sequential\"\n",
     "_________________________________________________________________\n",
     "Layer (type)                 Output Shape              Param #   \n",
     "=================================================================\n",
     "dense (Dense)                (None, 512)               401920    \n",
     "_________________________________________________________________\n",
     "dropout (Dropout)            (None, 512)               0         \n",
     "_________________________________________________________________\n",
     "dense_1 (Dense)              (None, 512)               262656    \n",
     "_________________________________________________________________\n",
     "dropout_1 (Dropout)          (None, 512)               0         \n",
     "_________________________________________________________________\n",
     "dense_2 (Dense)              (None, 10)                5130      \n",
```

```
     "=================================================================\n",
     "Total params: 669,706\n",
     "Trainable params: 669,706\n",
     "Non-trainable params: 0\n",
     "_________________________________________________________________\n",
     "Epoch 1/20\n",
     "469/469 [==============================] - 5s 11ms/step - loss: 0.4359 -␣
↪accuracy: 0.8619 - val_loss: 0.1130 - val_accuracy: 0.9647\n",
     "Epoch 2/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.1104 -␣
↪accuracy: 0.9670 - val_loss: 0.0888 - val_accuracy: 0.9726\n",
     "Epoch 3/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0727 -␣
↪accuracy: 0.9775 - val_loss: 0.0709 - val_accuracy: 0.9792\n",
     "Epoch 4/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0574 -␣
↪accuracy: 0.9826 - val_loss: 0.0847 - val_accuracy: 0.9772\n",
     "Epoch 5/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0464 -␣
↪accuracy: 0.9855 - val_loss: 0.0785 - val_accuracy: 0.9814\n",
     "Epoch 6/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0409 -␣
↪accuracy: 0.9880 - val_loss: 0.0795 - val_accuracy: 0.9796\n",
     "Epoch 7/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0364 -␣
↪accuracy: 0.9899 - val_loss: 0.0822 - val_accuracy: 0.9816\n",
     "Epoch 8/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0327 -␣
↪accuracy: 0.9905 - val_loss: 0.0865 - val_accuracy: 0.9830\n",
     "Epoch 9/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0281 -␣
↪accuracy: 0.9913 - val_loss: 0.0799 - val_accuracy: 0.9826\n",
     "Epoch 10/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0248 -␣
↪accuracy: 0.9929 - val_loss: 0.0844 - val_accuracy: 0.9836\n",
     "Epoch 11/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0237 -␣
↪accuracy: 0.9930 - val_loss: 0.0879 - val_accuracy: 0.9842\n",
     "Epoch 12/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0232 -␣
↪accuracy: 0.9932 - val_loss: 0.0978 - val_accuracy: 0.9802\n",
     "Epoch 13/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0238 -␣
↪accuracy: 0.9934 - val_loss: 0.0949 - val_accuracy: 0.9833\n",
     "Epoch 14/20\n",
```

```
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0191 -␣
↪accuracy: 0.9948 - val_loss: 0.0899 - val_accuracy: 0.9829\n",
     "Epoch 15/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0185 -␣
↪accuracy: 0.9946 - val_loss: 0.1074 - val_accuracy: 0.9821\n",
     "Epoch 16/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0179 -␣
↪accuracy: 0.9950 - val_loss: 0.1126 - val_accuracy: 0.9824\n",
     "Epoch 17/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0168 -␣
↪accuracy: 0.9956 - val_loss: 0.1350 - val_accuracy: 0.9826\n",
     "Epoch 18/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0186 -␣
↪accuracy: 0.9950 - val_loss: 0.1266 - val_accuracy: 0.9823\n",
     "Epoch 19/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0173 -␣
↪accuracy: 0.9953 - val_loss: 0.1194 - val_accuracy: 0.9822\n",
     "Epoch 20/20\n",
     "469/469 [==============================] - 4s 9ms/step - loss: 0.0144 -␣
↪accuracy: 0.9957 - val_loss: 0.1305 - val_accuracy: 0.9832\n",
     "Test loss: 0.1305491179227829\n",
     "Test accuracy: 0.9832000136375427\n"
    ]
   }
  ],
  "source": [
   "# convert class vectors to binary class matrices\n",
   "y_train = keras.utils.to_categorical(y_train, num_classes)\n",
   "y_test = keras.utils.to_categorical(y_test, num_classes)\n",
   "\n",
   "model = Sequential()\n",
   "model.add(Dense(512, activation='relu', input_shape=(784,)))\n",
   "model.add(Dropout(0.2))\n",
   "model.add(Dense(512, activation='relu'))\n",
   "model.add(Dropout(0.2))\n",
   "model.add(Dense(num_classes, activation='softmax'))\n",
   "\n",
   "model.summary()\n",
   "\n",
   "model.compile(loss='categorical_crossentropy',\n",
   "              optimizer=RMSprop(),\n",
   "              metrics=['accuracy'])\n",
   "\n",
   "history = model.fit(x_train, y_train,\n",
   "                    batch_size=batch_size,\n",
   "                    epochs=epochs,\n",
```

```
   "                    verbose=1,\n",
   "                    validation_data=(x_test, y_test))\n",
   "score = model.evaluate(x_test, y_test, verbose=0)\n",
   "print('Test loss:', score[0])\n",
   "print('Test accuracy:', score[1])"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 6,
  "id": "confidential-playback",
  "metadata": {},
  "outputs": [],
  "source": [
   "#Example Two#\n",
   "import sys\n",
   "from random import random\n",
   "from operator import add\n",
   "\n",
   "from pyspark.sql import SparkSession\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 14,
  "id": "academic-leeds",
  "metadata": {},
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Pi is roughly 3.142760\n"
    ]
   }
  ],
  "source": [
   "\n",
   "if __name__ == \"__main__\":\n",
   "    \"\"\"\n",
   "        Usage: pi [partitions]\n",
   "    \"\"\"\n",
   "    spark = SparkSession\\\n",
   "        .builder\\\n",
   "        .appName(\"PythonPi\")\\\n",
   "        .getOrCreate()\n",
   "\n",
```

```
    "    partitions = 2 #int(sys.argv[1]) if len(sys.argv) > 1 else 2#\n",
    "    n = 100000 * partitions\n",
    "\n",
    "    def f(_):\n",
    "        x = random() * 2 - 1\n",
    "        y = random() * 2 - 1\n",
    "        return 1 if x ** 2 + y ** 2 <= 1 else 0\n",
    "\n",
    "    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).
↪map(f).reduce(add)\n",
    "    print(\"Pi is roughly %f\" % (4.0 * count / n))\n",
    "\n",
    "    spark.stop()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "desperate-resident",
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.8.8"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

```
if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = 2 #int(sys.argv[1]) if len(sys.argv) > 1 else 2#
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 <= 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).
↪reduce(add)
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

```
Pi is roughly 3.142760
```

[ ]: