Will Spaeth

2-7-20

Homework 1

**-----------------------**

**./hw1\_script.py**

**-----------------------**

#!/usr/bin/env python3

import pickle

import os

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense

import numpy as np

def main():

# Load data

# Ins: (256, 8), float64

# Outs: (256, 1), float64

with open("hw1\_dataset.pkl", "rb") as fp:

hw1\_dataset = pickle.load(fp)

# Create & train 10 different models

prediction\_errors = []

for i in range(10):

# Build & compile model

model = dnn(hidden\_sizes=[8, 4, 4], hidden\_act="elu", output\_act="linear")

model.compile(optimizer="adam", loss="mse")

model.summary()

# Train model

model.fit(

x=hw1\_dataset["ins"],

y=hw1\_dataset["outs"],

epochs=3000,

batch\_size=32

)

# Predict and calculate absolute error

predictions = model.predict(hw1\_dataset["ins"])

prediction\_errors.append( np.squeeze(np.abs(hw1\_dataset["outs"] - predictions)) )

# Plot learning curve for this model

plot\_learning\_curve(i, model.history)

# Compute the absolute prediction errors for all runs, combine the data and generate a histogram of the absolute errors

prediction\_errors = np.concatenate(prediction\_errors, axis=0)

create\_histogram(prediction\_errors)

def dnn(hidden\_sizes, hidden\_act="sigmoid", output\_act="tanh"):

"""Construct a simple deep neural network"""

inputs = Input(shape=(8,))

hidden\_stack\_out = hidden\_stack(hidden\_sizes, hidden\_act)(inputs)

outputs = Dense(1, activation=output\_act)(hidden\_stack\_out)

return Model(inputs=inputs, outputs=outputs)

def hidden\_stack(hidden\_sizes, hidden\_act="sigmoid"):

"""Represents a stack of neural layers"""

layers = []

for size in hidden\_sizes:

layers.append(Dense(size, activation=hidden\_act))

def hidden\_stack\_layer(inputs):

"""Layer hook for stack"""

for i in range(len(layers)):

if i == 0:

carry\_out = layers[i](inputs)

else:

carry\_out = layers[i](carry\_out)

return carry\_out

return hidden\_stack\_layer

def create\_histogram(prediction\_errors):

fig = plt.figure()

ax = fig.add\_subplot(1, 1, 1)

ax.hist(prediction\_errors, 50)

plt.ylabel("Count")

plt.xlabel("Error")

fig.savefig("error\_histogram.png", dpi=fig.dpi)

def plot\_learning\_curve(experiment\_num, history):

save\_path = "learning\_curves/"

if not os.path.exists(save\_path):

os.mkdir(save\_path)

fig = plt.figure()

ax = fig.add\_subplot(1, 1, 1)

ax.plot(history.history["loss"])

plt.ylabel("MSE")

plt.xlabel("Epochs")

fig.savefig(save\_path + f"experiment\_{experiment\_num}.png", dpi=fig.dpi)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**------------------------------------------**

**Resulting 10 Learning Curves**

**-----------------------------------**

A close up of a device

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

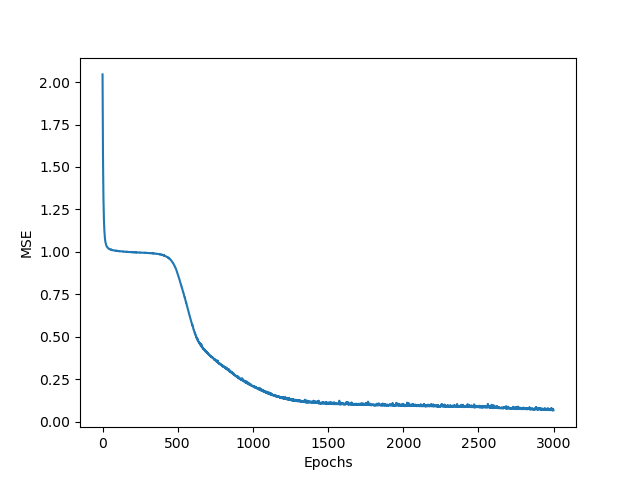
Description automatically generatedA screenshot of a cell phone

Description automatically generated

A close up of a device

Description automatically generatedA screenshot of a cell phone

Description automatically generated

**A screenshot of a cell phone

Description automatically generated**

A close up of a device

Description automatically generated**A screenshot of a cell phone

Description automatically generated**

**------------------------------------------**

**Resulting Error Histogram**

**-----------------------------------**

A screenshot of a cell phone

Description automatically generated