# **IST 597 Assignment – 2**

# **Vaibhav Singh**

# **PSU ID – 997500644**

# **Access ID – vxs5308**

## **Dataset - MNIST**

***Design MLP with 2 hidden layers (Input Layer - 2 hidden Layer - Output layer) to classify objects (fashion MNIST) and digits (MNIST).***

Answer -

* I have designed a model which includes a default mode with no regularization, l2 mode and dropout.
* I have added knobs for each mode and hyperparameter setting in the class itself.
* For L2 regularization, I have updated loss to loss + (lambda/(2 \* num\_training\_samples)) \* sum of squares of each term in each weight matrix
* I have performed data preprocessing prior to using this model
  + Reshaped the data
  + Normalized the image data
  + One hot encoding

# Combined model with L2 and dropout enable knobs

import sys

from tensorflow.python.ops.gen\_math\_ops import square

import numpy as np

# Define class to build mlp model

class MLP\_combined(object):

def \_\_init\_\_(self, size\_input, size\_hidden\_1, size\_hidden\_2, size\_output, reg\_mode = "default", lambda\_l2 = 0.8, dropout = 0.3, lr = 0.05, device=None):

"""

size\_input: int, size of input layer

size\_hidden: int, size of hidden layer

size\_output: int, size of output layer

device: str or None, either 'cpu' or 'gpu' or None. If None, the device to be used will be decided automatically during Eager Execution

"""

import sys

self.size\_input, self.size\_hidden\_1, self.size\_hidden\_2, self.size\_output, self.reg\_mode, self.lambda\_l2, self.dropout, self.lr, self.device =\

size\_input, size\_hidden\_1, size\_hidden\_2, size\_output, reg\_mode, lambda\_l2, dropout, lr, device

# Initialize weights between input layer and hidden layer 1 - (128, 128)

self.W1 = tf.Variable(tf.random.normal([self.size\_input, self.size\_hidden\_1]))

# Initialize biases for hidden layer 1 - (1,128)

self.b1 = tf.Variable(tf.random.normal([1, self.size\_hidden\_1]))

# Initialize weights between hidden layer 1 and hidden layer 2 - (128, 128)

self.W2 = tf.Variable(tf.random.normal([self.size\_hidden\_1, self.size\_hidden\_2]))

# Initialize biases for hidden layer 2 - (1, 128)

self.b2 = tf.Variable(tf.random.normal([1, self.size\_hidden\_2]))

# Initialize weights between hidden layer 2 and output layer - (128, 10)

self.W3 = tf.Variable(tf.random.normal([self.size\_hidden\_2, self.size\_output]))

# Initialize biases for output layer - (1,10)

self.b3 = tf.Variable(tf.random.normal([1, self.size\_output]))

# Define variables to be updated during backpropagation

self.variables = [self.W1, self.W2, self.W3, self.b1, self.b2, self.b3]

def forward(self, X):

"""

forward pass

X: Tensor, inputs

"""

if self.device is not None:

with tf.device('gpu:0' if self.device=='gpu' else 'cpu'):

self.y = self.compute\_output(X)

else:

self.y = self.compute\_output(X)

return self.y

def loss(self, y\_pred, y\_true):

'''

y\_pred - Tensor of shape (batch\_size, size\_output)

y\_true - Tensor of shape (batch\_size, size\_output)

'''

#Use categorical cross entropy in place of MSE

cce = tf.keras.losses.CategoricalCrossentropy()

#loss = tf.nn.softmax\_cross\_entropy\_with\_logits(y\_true, y\_pred)

loss = cce(y\_true, y\_pred)

if self.reg\_mode == "L2":

#Applying L2 regularization

#print ("#Applying L2 regularization")

para\_list = [self.W1, self.W2, self.W3]

regularizer = sum([tf.reduce\_sum(tf.square(\_var)) for \_var in para\_list])

#loss = self.lambda\_l2 \* loss + ((1 - self.lambda\_l2)/ number\_of\_train\_examples) \* regularizer

loss = loss + ((self.lambda\_l2/(2 \* number\_of\_train\_examples)) \* regularizer)

return loss

def backward(self, X\_train, y\_train):

"""

backward pass

"""

optimizer = tf.keras.optimizers.SGD(learning\_rate=self.lr)

#optimizer = tf.keras.optimizers.Adagrad(learning\_rate=0.05)

with tf.GradientTape() as tape:

predicted = self.forward(X\_train)

current\_loss = self.loss(predicted, y\_train)

grads = tape.gradient(current\_loss, self.variables)

optimizer.apply\_gradients(zip(grads, self.variables))

def compute\_output(self, X):

"""

Custom method to obtain output tensor during forward pass

"""

# Cast X to float32

X\_tf = tf.cast(X, dtype=tf.float32)

#Remember to normalize your dataset before moving forward

# Compute values in hidden layer 1

w1hat = tf.matmul(X\_tf, self.W1) + self.b1

h1hat = tf.nn.relu(w1hat)

# Compute values in hidden layer 2

w2hat = tf.matmul(h1hat, self.W2) + self.b2

h2hat = tf.nn.relu(w2hat)

if self.reg\_mode == "dropout":

# Adding dropout on layer 2

#print ("# Adding dropout on layer 2")

h2hat = tf.nn.dropout(h2hat, self.dropout)

# Compute output

output = tf.matmul(h2hat, self.W3) + self.b3

#Now consider two things , First look at inbuild loss functions if they work with softmax or not and then change this

#Second add tf.Softmax(output) and then return this variable

output = tf.nn.softmax(output)

#print (f"Output : {output}")

return output

***Design regularization approaches, and analyze drop/boost in performance of your model. Report results with atleast 2 regularization variants (droput, L1 penalty, L2, L1+L2, Normalization, Noise). Do you see any tradeoff with respect to bias-variance? Report your findings.***

Answer -

* I have implemented L2 regularization and dropout with the main model (MLP\_combined) itself as you can see above. Initially, I had 3 different models but this new approach made the code more modular and proved handy for inference.
* With different set of hyperparameters, we get different sets of results. As expected, we see substantial difference in execution times as well.
* For most of my optimizations, I am getting good performance with dropout and very minimal improvement with L2 regularization.
* When the number of neurons in hidden layer is less(say 100) then L2 tends to outperform dropout and I will discuss that in the next question. Dropout however improves performs with increase in epochs.
* The performance across test and training set tends to stay almost the same hence I would say that variance is low in almost all cases and regularization doesn't affect that much.
* Bias within the data tends to improve with dropout but with optimal settings I am getting an average bias for all models

***How did you perform hyper-parameter optimization? Report your approach, also show settings for best model.***

Answer -

* I have tried various optimizations but I couldn't document all sorts of results.
* Cases with low batch size tends to take a lot of execution time and hence I couldn't update all those results.
* With high batch size, on increasing epochs the performance also improves a lot.
* To ease the effort in execution, I have made the code modular.
* Ideally, higher batch sizes work well higher learning rates.
* Dropout of 0.2 to 0.3 tends to work fine.
* I also tried various optimizers such as Adam, Adagrad, RMSProp but SGD itself performs the best in my findings
* With higher number of hidden layers(such as 512,128), accuracy improved much faster but reached minima very soon after which improvement is very gradual.
* In some set of hyperparameters, the default mode with no regularization tends to give low accuracies and the result is substantially improved with dropout.
* Higher lambda for l2 reg values yielded bad performance in general.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Initial hyperparams | Hyperparams | Train Accuracy | Train Cross entropy Loss | Comments |
| default | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 512  shuffle - 1000  learning rate - 0.05 | 72.10% | 0.063 | Getting better Test Cross entropy loss but similar Accuracy in test |
| L2 | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 512  shuffle - 1000  learning rate - 0.05  lambda - 0.05 | 74.416% | 0.0081 | Getting better Test Cross entropy loss but similar Accuracy in test |
| dropout | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 512  shuffle - 1000  learning rate - 0.05  dropout - 0.3 | 68.72% | 0.0098 | Convergence is initially slower with this set of hidden layer combination. |
| dropout | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 512  shuffle - 1000  learning rate - 0.05  dropout - 0.4 | 66.92% | 0.0098 | Convergence is initially slower with this set of hidden layer combination.  However overall performance can improve a lot with more epochs since our batch size is high. |
| default | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 64  shuffle - 100  learning rate - 0.05 | 74.91% | 0.063 |  |
| dropout | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 64  shuffle - 100  learning rate -  dropout - 0.3 | 81.12% | 0.047 | **Accuracy with dropout is the best in this setting** |
| l2 | epochs = 20  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 64  shuffle - 100  learning rate - 0.05 | 72.2% | 0.07 |  |
| dropout | epochs = 250  side\_hidden\_1 = size\_hidden\_2 = 100 | batch size - 512  shuffle - 1000  learning rate - 0.05  dropout - 0.3 | 80.88% | 0.0059 | **Accuracy and loss with dropout are the best in this setting. The convergence of loss is better here** |

After running inference, I noticed that seeds tend to have a substantial impact on the results.

***Report your results across multiple trials (minimum 10). Beside accuracy you will also report standard error and plot graph showing variance.***

Answer –

############## Setting values as per dataset ##############

size\_input = 784

size\_hidden\_1 = 100

size\_hidden\_2 = 100

size\_output = 10

number\_of\_train\_examples = X\_train.shape[0]

number\_of\_test\_examples = X\_test.shape[0]

print (f"{number\_of\_test\_examples},{number\_of\_train\_examples}")

########################################################

'''

Additional steps done above

1. Reshape to flatten

2. Normalization

3. One hot encoding

'''

#######################################################

reg\_type\_list = ["default", "L2", "dropout"]

inference\_stats\_dict = {"default":{}, "L2": {}, "dropout": {}}

lambda\_val = 0.05

dropout\_val = 0.3

NUM\_EPOCHS = 20

random\_seeds = list(np.random.randint(low=2000, high=4000, size=10))

inf\_iters = 10

device\_type = 'gpu'

shuffle\_size = 1000

batch\_size = 512

learning\_rate = 0.05

test\_ds = tf.data.Dataset.from\_tensor\_slices((X\_test, y\_test)).batch(100)

A picture containing text

Description automatically generated

Capturing a screenshot from the results obtained after running trials for 10 different seeds.

In this process, I ran overall 30 iterations of training and testing and documented every result above.

Updating mean and variance for losses and accuracies as was suggested in the announcement –

Text

Description automatically generated

From the results, we can see that we get the most consistent best results for dropout for the following 10 seeds

## **Dataset - Fashion MNIST**

The model stays the same for Fashion MNIST but the hyperparameters change.

Unfortunately, I couldn’t document all possible values and hence updating the final 10 trials I performed with my set of settings

A screenshot of a computer

Description automatically generated with medium confidence

## Model: default ###

Cross entropy loss across trials : [0.11522347412109375, 0.08467548828125, 0.08576741333007812, 0.1145487060546875, 0.08642153930664062, 0.09960307006835938, 0.07544818725585938, 0.104943408203125, 0.1073142578125, 0.11834266357421876]

Accuracy across trials : [0.2849, 0.4742, 0.4646, 0.2888, 0.4625, 0.3818, 0.5296, 0.3482, 0.333, 0.2655]

## Model: L2 ###

Cross entropy loss across trials : [0.11575765380859375, 0.08069005737304688, 0.09145341186523437, 0.11504547119140625, 0.07218370971679687, 0.10011902465820313, 0.08523603515625, 0.11567591552734376, 0.10225344848632813, 0.1164573974609375]

Accuracy across trials : [0.2839, 0.5004, 0.4335, 0.2883, 0.5532, 0.3809, 0.4725, 0.2836, 0.3672, 0.2795]

## Model: dropout ###

Cross entropy loss across trials : [0.09012791137695313, 0.05209363403320313, 0.07554984130859375, 0.059787115478515625, 0.06057958984375, 0.07330875244140625, 0.07488440551757812, 0.06791614379882813, 0.06906673583984375, 0.0644167724609375]

Accuracy across trials : [0.4389, 0.6743, 0.529, 0.6259, 0.623, 0.5426, 0.5337, 0.5756, 0.5689, 0.5966]

Text

Description automatically generated