CSPB 4622 - Truong - Machine Learning

Dashboard / My courses / 2247:CSPB 4622 / 4 November - 10 November / Quiz 11. optimization

Started on	Monday, 11 November 2024, 4:07 PM
State	Finished
Completed on	Monday, 11 November 2024, 4:09 PM
Time taken	1 min 14 secs
Marks	6.00/6.00
Grade	**10.00** out of 10.00 (**100**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Which of the following statements is **NOT** a reason for adding a dropout layer?

- a. Adding a dropout layer trains an ensemble of models.
- b. Adding a dropout layer forces the network to have a redundant representation.
- c. Adding a dropout layer functions as regularization.
- [✓ d. Adding a dropout layer makes the network more expressive during training time.] ✓

Your answer is correct. In the article *Surprising properties of dropout in deep networks,* Helmbold and Long (2018, p. 15) write, “dropout training can hijack part of the expressiveness of the wide layer to control the artificial variance due to dropout rather than fitting the underlying patterns in the data.” Helmbold, D. P., & Long, P. M. (2018). Surprising properties of dropout in deep networks. *Journal of Machine Learning Research*, *18*, 1–28. https://doi.org/https://www.jmlr.org/papers/volume18/16-549/16-549.pdf

The correct answer is: Adding a dropout layer makes the network more expressive during training time.

Question **2**

Correct

Mark 1.00 out of 1.00

Your friend built a neural network model shown below but encountered an issue while training and is asking you for advice. **Model:**

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input_shape=(784,)))
model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(NUM_CAT, activation=tf.nn.softmax))


model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_27 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_29 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
```

Training log:

```
opt = tf.keras.optimizers.SGD(learning_rate=0.5)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
model.fit(train_images, train_labels, validation_split=0.2, epochs=100)
```

```
Epoch 21/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2619 - accuracy: 0.9065 - val_loss: 0.4275 - val_accuracy: 0.8648
Epoch 22/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2543 - accuracy: 0.9088 - val_loss: 0.4236 - val_accuracy: 0.8737
Epoch 23/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2528 - accuracy: 0.9103 - val_loss: 0.4233 - val_accuracy: 0.8768
Epoch 24/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2453 - accuracy: 0.9126 - val_loss: 0.4342 - val_accuracy: 0.8763
Epoch 25/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2414 - accuracy: 0.9128 - val_loss: 0.4497 - val_accuracy: 0.8722
Epoch 26/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2493 - accuracy: 0.9094 - val_loss: 0.4280 - val_accuracy: 0.8637
Epoch 27/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2518 - accuracy: 0.9112 - val_loss: 0.4415 - val_accuracy: 0.8743
Epoch 28/100
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2497 - accuracy: 0.9109 - val_loss: 0.4247 - val_accuracy: 0.8799
Epoch 29/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.8550 - val_loss: nan - val_accuracy: 0.1030
Epoch 30/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 31/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 32/100
1500/1500 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 33/100
1500/1500 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 34/100
```

What is the most important fix you recommend to your friend?

- a. Try a smaller architecture
- b. Early stopping
- [✓ c. Reduce learning rate] ✓
- d. You can add regularization terms such as dropout, batch normalization, ridge, and lasso etc.

Your answer is correct. The training log shows that it strongly overfits after certain epochs, and the loss even diverges due to the very high learning rate. The obvious observation is that the learning rate is very high, so I recommend reducing the learning rate first and seeing how it goes. The good values for the learning rate can be obtained by trying multiple learning rate values and monitoring the train/validation loss or accuracy, and choosing the one that leads the best validation accuracy.

The correct answer is: Reduce learning rate

Question **3**

Correct

Mark 1.00 out of 1.00

Your friend built a neural network model shown below but encountered an issue while training and is asking you for advice. **Model:**

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input_shape=(784,)))
model.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(NUM_CAT, activation=tf.nn.softmax))


model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_27 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_29 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
```

Training log:

```
opt = tf.keras.optimizers.SGD(learning_rate=0.5)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
model.fit(train_images, train_labels, validation_split=0.2, epochs=100)
```

```
Epoch 21/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2619 - accuracy: 0.9065 - val_loss: 0.4275 - val_accuracy: 0.8648
Epoch 22/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2543 - accuracy: 0.9088 - val_loss: 0.4236 - val_accuracy: 0.8737
Epoch 23/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2528 - accuracy: 0.9103 - val_loss: 0.4233 - val_accuracy: 0.8768
Epoch 24/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2453 - accuracy: 0.9126 - val_loss: 0.4342 - val_accuracy: 0.8763
Epoch 25/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2414 - accuracy: 0.9128 - val_loss: 0.4497 - val_accuracy: 0.8722
Epoch 26/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2493 - accuracy: 0.9094 - val_loss: 0.4280 - val_accuracy: 0.8637
Epoch 27/100
1500/1500 [==============================] - 5s 3ms/step - loss: 0.2518 - accuracy: 0.9112 - val_loss: 0.4415 - val_accuracy: 0.8743
Epoch 28/100
1500/1500 [==============================] - 4s 3ms/step - loss: 0.2497 - accuracy: 0.9109 - val_loss: 0.4247 - val_accuracy: 0.8799
Epoch 29/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.8550 - val_loss: nan - val_accuracy: 0.1030
Epoch 30/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 31/100
1500/1500 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 32/100
1500/1500 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 33/100
1500/1500 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.0993 - val_loss: nan - val_accuracy: 0.1030
Epoch 34/100
```

What are possible recommendations that can improve the result? (Select all that apply).

- ☐ a. Increase learning rate
- ☑ b. Early stopping ✓
- ☑ c. Add regularization terms such as dropout, batch normalization, ridge, and lasso etc. ✓
- ☐ d. More epochs
- ☐ e. Larger architecture

Your answer is correct. You could add regularization terms such as dropout, batch normalization, ridge and lasso, etc. to improve the result.

The correct answers are: Early stopping, Add regularization terms such as dropout, batch normalization, ridge, and lasso etc.

Question **4**

Correct

Mark 1.00 out of 1.00

The optimization goal of Gradient Descent is to minimize the loss function at the output.

Select one:

[✓ True] ✓

False

The correct answer is 'True'.

Question **5**

Correct

Mark 1.00 out of 1.00

Adding momentum to SGD (Stochastic Gradient Descent) can make the convergence faster.

Select one:

[✓ True] ✓

False

The correct answer is 'True'.

Question **6**

Correct

Mark 1.00 out of 1.00

Stochastic Gradient Descent uses all the available training data into consideration (the entire batch) to calculate loss and update the weights at a time (in a single step or one epoch).

Select one:

True

[✓ False] ✓

The correct answer is 'False'.