1. Which of the following statements is NOT a reason for adding a dropout layer? 3 / 3 points

- Adding a dropout layer forces the network to have a redundant representation.
- Adding a dropout layer functions as regularization.
- (selected) Adding a dropout layer makes the network more expressive during training time.
- Adding a dropout layer trains an ensemble of models.

Correct
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

2. Your friend built a neural network model shown below but encountered an issue while training and is asking you for advice. 3 / 3 points

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: 0.2510 - 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).

- [ ] Increase learning rate

- [x] Add regularization terms such as dropout, batch normalization, ridge, and lasso etc.

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

- [ ] More epochs
- [ ] Larger architecture
- [x] Early stopping

  Correct
  Your answer is correct. Early stopping could improve the results.

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

3 / 3 points

```
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?

- ○ You can add regularization terms such as dropout, batch normalization, ridge, and lasso etc.
- ○ Try a smaller architecture
- ○ Early stopping
- ◉ Reduce learning rate

  Correct

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.