Fine-tuning keras models

Puteaux, Fall/Winter 2020-2021

```
################################
##                            ##
##  Deep Learning in Python   ##
##                            ##
################################
```

```
§1 Introduction to Deep Learning in Python
```

```
§1.4 Fine-tuning keras models
```

1 Understanding model optimization

1.1 Why is optimization hard?

- Simultaneously optimize 1000's of parameters with complex relationships.
- Updates may not improve the model meaningfully.
- Updates will be too small (if the learning rate is low) or too large (if the learning rate is high).

1.2 Code of stochastic gradient descent:

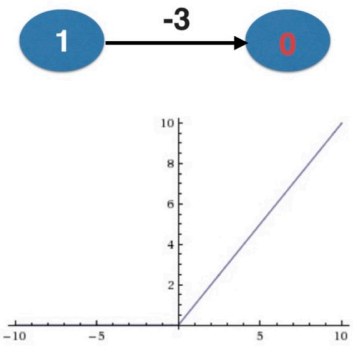
[1]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils.np_utils import to_categorical
from keras.optimizers import SGD


def data_preparation(df):
    df = df.reindex(columns=[
        'SHOT_CLOCK', 'DRIBBLES', 'TOUCH_TIME', 'SHOT_DIST', 'CLOSE_DEF_DIST',
        'SHOT_RESULT'
    ])
    df['SHOT_CLOCK'] = df['SHOT_CLOCK'].fillna(0)
    df['SHOT_RESULT'].replace('missed', 0, inplace=True)
    df['SHOT_RESULT'].replace('made', 1, inplace=True)
    df.columns = df.columns.str.lower()
    return df
```

```
data = pd.read_csv('ref1. Basketball shot log.csv')
data = data_preparation(data)

predictors = data.drop(['shot_result'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(data.shot_result)
input_shape = (n_cols, )
```

[2]:
```
def get_new_model(input_shape=input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape=input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return (model)


lr_to_test = [.000001, 0.01, 1]

# loop over learning rates
for lr in lr_to_test:
    model = get_new_model()
    my_optimizer = SGD(lr=lr)
    model.compile(optimizer=my_optimizer, loss='categorical_crossentropy')
    model.fit(predictors, target)
```

```
4003/4003 [==============================] - 4s 882us/step - loss: 0.8326
4003/4003 [==============================] - 3s 842us/step - loss: 0.6750
4003/4003 [==============================] - 4s 959us/step - loss: nan
```

1.3 What is the dying neuron problem?

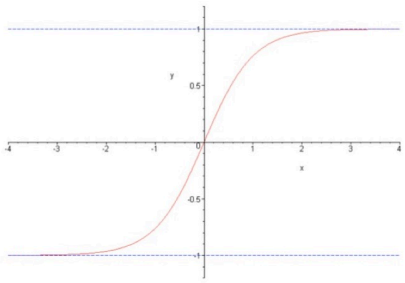


1.4 What is the vanishing gradient problem?

- This occurs when many layers have very small slopes:

  - e.g., *due to being on at part of the hyperbolic tangent (`tanh`) curve*

- In deep networks, updates to backpropagation were close to 0.



tanh function

1.5 Practice question for diagnosing optimization problems:

- Which of the following could prevent a model from showing an improved loss in its first few epochs?

☐ Learning rate too low.

☐ Learning rate too high.

☐ Poor choice of the activation function.

☒ All of the above.

1.6 Practice exercises for understanding model optimization:

▶ Package pre-loading:

[3]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
```

▶ Data pre-loading:

[4]:
```
df = pd.read_csv('ref4. Titanic.csv')

df.replace(False, 0, inplace=True)
df.replace(True, 1, inplace=True)

predictors = df.drop(['survived'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(df.survived)
input_shape = (n_cols, )
```

▶ Code pre-loading:

[5]:
```
def get_new_model(input_shape=input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape=input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return (model)
```

► **Changing optimization parameters practice:**

[6]:
```
# Import the SGD optimizer
from keras.optimizers import SGD

# Create list of learning rates: lr_to_test
lr_to_test = [.000001, 0.01, 1]

# Loop over learning rates
for lr in lr_to_test:
    print('\n\nTesting model with learning rate: %f\n' % lr)

    # Build new model to test, unaffected by previous models
    model = get_new_model()

    # Create SGD optimizer with specified learning rate: my_optimizer
    my_optimizer = SGD(lr=lr)

    # Compile the model
    model.compile(optimizer=my_optimizer, loss='categorical_crossentropy')

    # Fit the model
    model.fit(predictors, target)
```

```
Testing model with learning rate: 0.000001

28/28 [==============================] - 0s 11ms/step - loss: 5.9586


Testing model with learning rate: 0.010000

28/28 [==============================] - 0s 9ms/step - loss: 2.9420


Testing model with learning rate: 1.000000

28/28 [==============================] - 0s 10ms/step - loss: 2151.7005
```

2 Model validation

2.1 Why is it important to choose validation in deep learning?

- Repeated training from cross-validation would take a long time, so it is common to use validation split rather than cross-validation.
- Deep learning is widely used in large datasets because the single validation score is based on a large amount of data and is reliable.

2.2 Code of model validation:

[7]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils.np_utils import to_categorical


def data_preparation(df):
    df = df.reindex(columns=[
        'SHOT_CLOCK', 'DRIBBLES', 'TOUCH_TIME', 'SHOT_DIST', 'CLOSE_DEF_DIST',
        'SHOT_RESULT'
    ])
    df['SHOT_CLOCK'] = df['SHOT_CLOCK'].fillna(0)
    df['SHOT_RESULT'].replace('missed', 0, inplace=True)
    df['SHOT_RESULT'].replace('made', 1, inplace=True)
    df.columns = df.columns.str.lower()
    return df


data = pd.read_csv('ref1. Basketball shot log.csv')
data = data_preparation(data)

predictors = data.drop(['shot_result'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(data.shot_result)
input_shape = (n_cols, )


def get_new_model(input_shape=input_shape):
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape=input_shape))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    return (model)


model = get_new_model()
```

[8]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(predictors, target, validation_split=0.3)
```

```
2802/2802 [==============================] - 4s 2ms/step - loss: 0.6957 -
accuracy: 0.5992 - val_loss: 0.6519 - val_accuracy: 0.6150
```

[8]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8253d66dd0>
```

2.3 Code of early stopping:

[9]:
```
from keras.callbacks import EarlyStopping

early_stopping_monitor = EarlyStopping(patience=2)

model.fit(predictors,
          target,
          validation_split=0.3,
          epochs=20,
          callbacks=[early_stopping_monitor])
```

```
Epoch 1/20
2802/2802 [==============================] - 3s 1ms/step - loss: 0.6531 -
accuracy: 0.6168 - val_loss: 0.6548 - val_accuracy: 0.6145
Epoch 2/20
2802/2802 [==============================] - 3s 1ms/step - loss: 0.6519 -
accuracy: 0.6188 - val_loss: 0.6514 - val_accuracy: 0.6146
Epoch 3/20
2802/2802 [==============================] - 3s 1ms/step - loss: 0.6510 -
accuracy: 0.6192 - val_loss: 0.6522 - val_accuracy: 0.6186
Epoch 4/20
2802/2802 [==============================] - 3s 1ms/step - loss: 0.6503 -
accuracy: 0.6200 - val_loss: 0.6521 - val_accuracy: 0.6158
```

[9]:
```
<tensorflow.python.keras.callbacks.History at 0x7f824a6a1590>
```

2.4 What kind of experimentations could be included in deep learning?

- Experiment with different architectures.
- More layers.
- Fewer layers.
- Layers with more nodes.
- Layers with fewer nodes.
- Creating a great model requires experimentation.

2.5 Practice exercises for model validation:

▶ **Package pre-loading:**

[10]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
```

▶ **Data pre-loading:**

[11]:
```
df = pd.read_csv('ref4. Titanic.csv')

df.replace(False, 0, inplace=True)
df.replace(True, 1, inplace=True)

predictors = df.drop(['survived'], axis=1).to_numpy()
n_cols = predictors.shape[1]
target = to_categorical(df.survived)
input_shape = (n_cols, )
```

▶ **Evaluating model accuracy on validation dataset practice:**

[12]:
```
# Save the number of columns in predictors: n_cols
n_cols = predictors.shape[1]
input_shape = (n_cols, )

# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
hist = model.fit(predictors, target, validation_split=0.3)
```

```
20/20 [==============================] - 0s 20ms/step - loss: 1.2657 - accuracy:
0.5784 - val_loss: 1.1484 - val_accuracy: 0.6231
```

▶ **Early stopping optimization optimizing practice:**

[13]:
```
# Import EarlyStopping
from keras.callbacks import EarlyStopping
```

```
# Save the number of columns in predictors: n_cols
n_cols = predictors.shape[1]
input_shape = (n_cols, )

# Specify the model
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=input_shape))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Fit the model
model.fit(predictors,
          target,
          validation_split=0.3,
          epochs=30,
          callbacks=[early_stopping_monitor])
```

```
Epoch 1/30
20/20 [==============================] - 0s 19ms/step - loss: 1.6996 - accuracy:
0.5500 - val_loss: 0.6122 - val_accuracy: 0.6791
Epoch 2/30
20/20 [==============================] - 0s 3ms/step - loss: 0.6733 - accuracy:
0.6122 - val_loss: 0.5289 - val_accuracy: 0.7388
Epoch 3/30
20/20 [==============================] - 0s 3ms/step - loss: 0.6220 - accuracy:
0.6723 - val_loss: 0.5741 - val_accuracy: 0.7239
Epoch 4/30
20/20 [==============================] - 0s 3ms/step - loss: 0.6540 - accuracy:
0.6644 - val_loss: 0.6322 - val_accuracy: 0.6791
```

[13]: <tensorflow.python.keras.callbacks.History at 0x7f8235a8eb10>

▶ Package re-pre-loading:

[14]:
```
import matplotlib.pyplot as plt
```

▶ Code pre-loading:

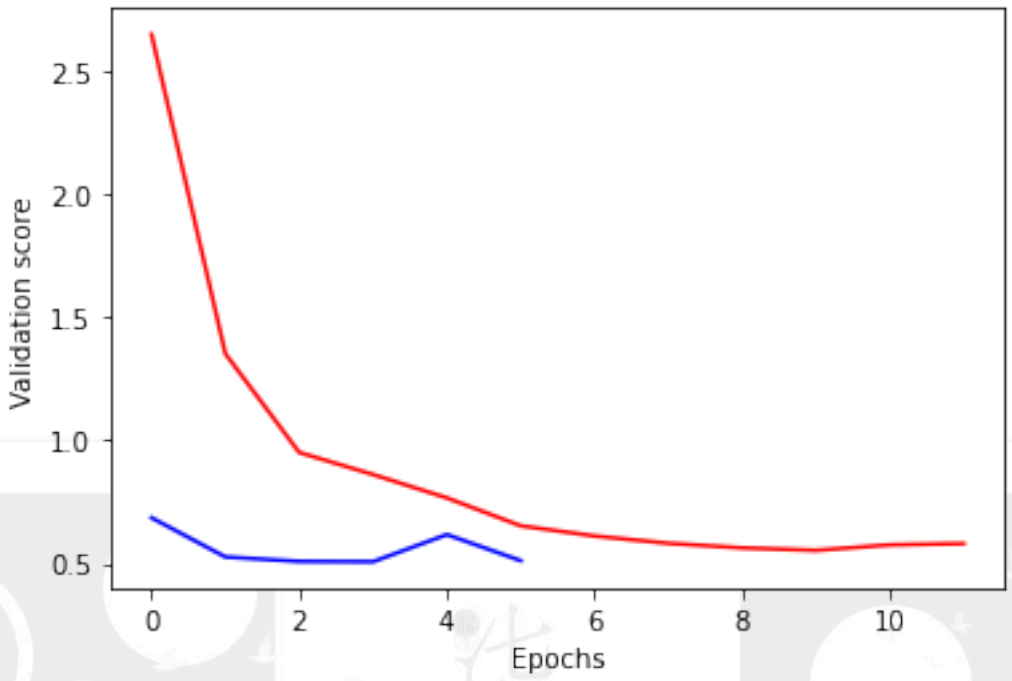
[15]:
```
model_1 = Sequential()
model_1.add(Dense(10, activation='relu', input_shape=input_shape))
```

```
model_1.add(Dense(10, activation='relu'))
model_1.add(Dense(2, activation='softmax'))
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

▶ **Experimenting with wider networks practice:**

[16]:
```
# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Create the new model: model_2
model_2 = Sequential()

# Add the first and second layers
model_2.add(Dense(100, activation='relu', input_shape=input_shape))
model_2.add(Dense(100, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Fit model_1
model_1_training = model_1.fit(predictors,
                               target,
                               epochs=15,
                               validation_split=0.2,
                               callbacks=[early_stopping_monitor],
                               verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors,
                               target,
                               epochs=15,
                               validation_split=0.2,
                               callbacks=[early_stopping_monitor],
                               verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r',
         model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
```

```
plt.show()
```



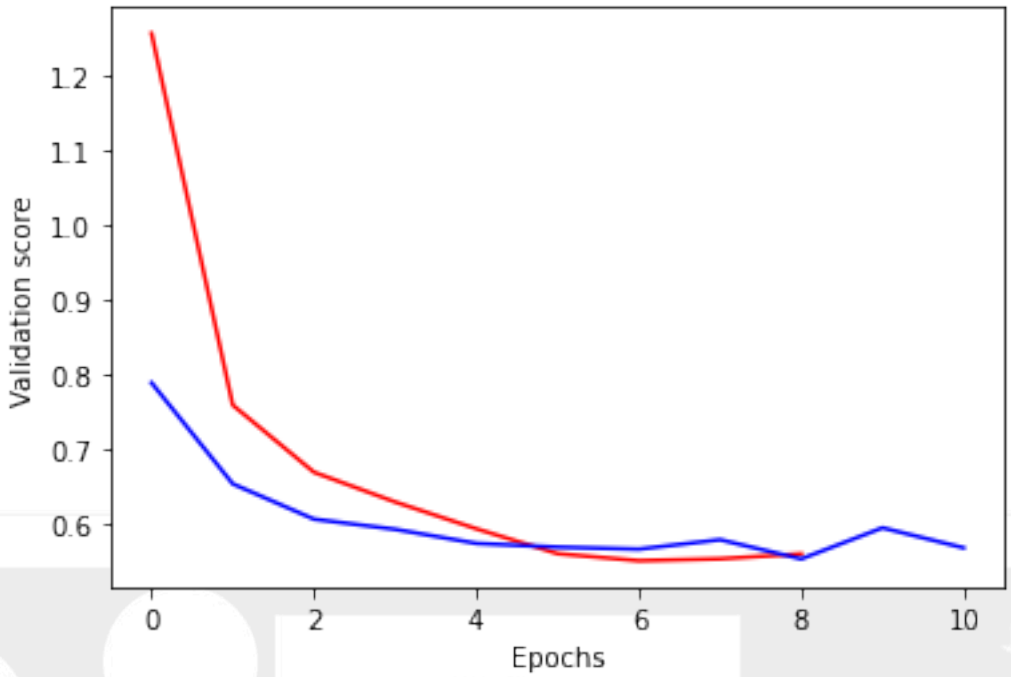
▶ **Code re-pre-loading:**

[17]:
```
model_1 = Sequential()
model_1.add(Dense(50, activation='relu', input_shape=input_shape))
model_1.add(Dense(2, activation='softmax'))
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

▶ **Network layers adding practice:**

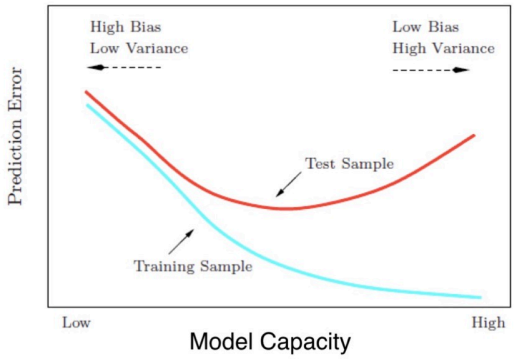
[18]:
```
# The input shape to use in the first hidden layer
input_shape = (n_cols, )

# Create the new model: model_2
model_2 = Sequential()

# Add the first, second, and third hidden layers
model_2.add(Dense(50, activation='relu', input_shape=input_shape))
model_2.add(Dense(50, activation='relu'))
model_2.add(Dense(50, activation='relu'))

# Add the output layer
```

```
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Fit model 1
model_1_training = model_1.fit(predictors,
                               target,
                               epochs=20,
                               validation_split=0.4,
                               callbacks=[early_stopping_monitor],
                               verbose=False)

# Fit model 2
model_2_training = model_2.fit(predictors,
                               target,
                               epochs=20,
                               validation_split=0.4,
                               callbacks=[early_stopping_monitor],
                               verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r',
         model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```



3 Thinking about model capacity

3.1 What is the connection between overfitting and model capacity?



3.2 What is a good workflow for optimizing model capacity?

- Start with a small network.
- Gradually increase capacity.
- Keep increasing capacity until the validation score is no longer improving.

3.3 How do sequential experiments function?

Hidden Layers	Nodes Per Layer	Mean Squared Error	Next Step
1	100	5.4	Increase Capacity
1	250	4.8	Increase Capacity
2	250	4.4	Increase Capacity
3	250	4.5	Decrease Capacity
3	200	4.3	Done

3.4 Practice question for experimenting with model structures:

- Run an experiment to compare two identical networks except that the 2nd network had an extra hidden layer. It could be seen that this 2nd network (the deeper network) had better performance. Given that, which of the following would be a good experiment to run next for even better performance?

  ☐ Try a new network with fewer layers than anything you have tried yet.

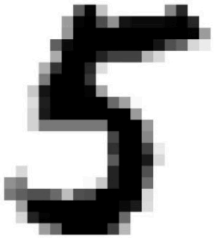
  ☒ Use more units in each hidden layer.

  ☐ Use fewer units in each hidden layer.

4 Stepping up to images

4.1 How to recognize handwritten digits?

- The MNIST dataset.
- 28 x 28 grid flattened to 784 values for each image.
- Value in each part of the array denotes the darkness of that pixel.



4.2 Practice exercises for model validation:

► **Package pre-loading:**

[19]:
```
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
```

▶ **Data pre-loading:**

[20]:
```
df = pd.read_csv('ref8. MNIST.csv', header=None)
X = df.iloc[:, 1:].to_numpy()
y = to_categorical(df.iloc[:, 0])
```

▶ **Evaluating model accuracy on validation dataset practice:**

[21]:
```
# Create the model: model
model = Sequential()

# Add the first hidden layer
model.add(Dense(50, activation='relu', input_shape=(784, )))

# Add the second hidden layer
model.add(Dense(50, activation='relu'))

# Add the output layer
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fit the model
model.fit(X, y, validation_split=0.3)
```

```
44/44 [==============================] - 0s 10ms/step - loss: 42.7795 -
accuracy: 0.2837 - val_loss: 7.9051 - val_accuracy: 0.5840
```

[21]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8251848a50>
```

5 Final thoughts

5.1 What are the next steps?

- Start with standard prediction problems on tables of numbers.
- Images (with convolutional neural networks) are common next steps.
- keras.io for excellent documentation.
- The graphical processing unit (GPU) provides dramatic speedups in model training times.

- Need a CUDA compatible GPU.
- Here is a blog post that explains how to training on using GPUs in the cloud.