Fine-Tuning Keras Models

LaTeX, Puteaux, 2020, 2021

Table of Contents

1 Understanding model optimization

1.1 [note-1] Why optimization is hard

- Simultaneously optimizing 1000s of parameters with complex relationships.

- Updates may not improve the model meaningfully.

- Updates could be too small (if the learning rate is low) or too large (if the learning rate is high).

1.2 [code-1] Stochastic gradient descent

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


     def data_preparation(data):
         df = data.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('../Datasets/1. Basketball shot log.csv')
     df = data_preparation(data)

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

```python
    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 858us/step - loss: 1.2447
4003/4003 [==============================] - 4s 921us/step - loss: 0.6708
4003/4003 [==============================] - 4s 923us/step - loss: nan
```

1.3 [note-2] The dying neuron problem
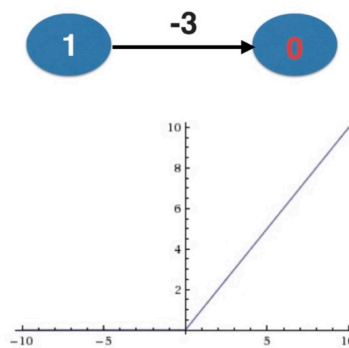
1.4 [note-3] Vanishing gradients

- It occurs when many layers have very small slopes (*e.g., due to being on at part of tanh curve*).

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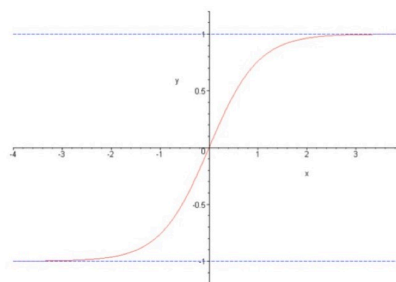
tanh function

1.5 [quiz-1] 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 [task-1] Changing optimization parameters

▶ **Package pre-loading**

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

▶ **Data pre-loading**

```python
[3]: df = pd.read_csv('../Datasets/2. 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**

```python
[4]: 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)
```

▶ **Task practice**

```python
[5]: # Import the SGD optimizer
     from tensorflow.keras.optimizers import SGD

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

```python
# 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 1ms/step - loss: 2.2489


Testing model with learning rate: 0.010000

28/28 [==============================] - 0s 992us/step - loss: 2.1813


Testing model with learning rate: 1.000000

28/28 [==============================] - 0s 932us/step - loss:
5823429736475.8662
```

2 Model validation

2.1 [note-1] Validation in deep learning

- Commonly use validation split rather than cross-validation.

- Deep learning is widely used on large datasets.

- A single validation score is based on a large amount of data and is reliable.

- Repeated training from cross-validation would take a long time.

2.2 [code-1] Model validation

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


     def data_preparation(data):
         df = data.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('../Datasets/1. Basketball shot log.csv')
     df = data_preparation(data)
     predictors = df.drop(['shot_result'], axis=1).to_numpy()
     n_cols = predictors.shape[1]
     input_shape = (n_cols, )
     target = to_categorical(df.shot_result)


     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()

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

```
2802/2802 [==============================] - 6s 2ms/step - loss: 0.6734 -
accuracy: 0.5988 - val_loss: 0.6577 - val_accuracy: 0.6079
```

```
[6]: <tensorflow.python.keras.callbacks.History at 0x7fd7f0f3e990>
```

2.3 [code-2] Early stopping

```
[7]: from tensorflow.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 [==============================] - 5s 2ms/step - loss: 0.6539 -
accuracy: 0.6179 - val_loss: 0.6534 - val_accuracy: 0.6154
Epoch 2/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6521 -
accuracy: 0.6172 - val_loss: 0.6583 - val_accuracy: 0.6166
Epoch 3/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6511 -
accuracy: 0.6191 - val_loss: 0.6516 - val_accuracy: 0.6152
Epoch 4/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6507 -
accuracy: 0.6190 - val_loss: 0.6503 - val_accuracy: 0.6190
Epoch 5/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6499 -
accuracy: 0.6198 - val_loss: 0.6494 - val_accuracy: 0.6198
Epoch 6/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6494 -
accuracy: 0.6206 - val_loss: 0.6514 - val_accuracy: 0.6183
Epoch 7/20
2802/2802 [==============================] - 5s 2ms/step - loss: 0.6493 -
accuracy: 0.6210 - val_loss: 0.6513 - val_accuracy: 0.6155
```

```
[7]: <tensorflow.python.keras.callbacks.History at 0x7fd7f17e1990>
```

2.4 [note-2] Experimentation

- Experiment with different architectures.

- More layers.

- Fewer layers.

- Layers with more nodes.

- Layers with fewer nodes.

- Creating a great model requires experimentation.

2.5 [task-1] Evaluating model accuracy on a validation dataset

▶ **Package pre-loading**

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

▶ **Data pre-loading**

```
[9]: df = pd.read_csv('../Datasets/2. 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]
     input_shape = (n_cols, )
     target = to_categorical(df.survived)
```

▶ **Task practice**

```
[10]: # 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 10ms/step - loss: 2.4766 - accuracy:
0.5772 - val_loss: 0.9333 - val_accuracy: 0.6418
```

2.6 [task-2] Early stopping: optimizing the optimization

▶ **Task practice**

[11]:
```python
# Import EarlyStopping
from tensorflow.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 10ms/step - loss: 1.1551 - accuracy:
0.5592 - val_loss: 0.6302 - val_accuracy: 0.7052
Epoch 2/30
20/20 [==============================] - 0s 3ms/step - loss: 0.6728 - accuracy:
0.6736 - val_loss: 0.7890 - val_accuracy: 0.6418
Epoch 3/30
20/20 [==============================] - 0s 3ms/step - loss: 0.7163 - accuracy:
0.6343 - val_loss: 0.5353 - val_accuracy: 0.7201
Epoch 4/30
20/20 [==============================] - 0s 4ms/step - loss: 0.6724 - accuracy:
0.6570 - val_loss: 0.5235 - val_accuracy: 0.7873
Epoch 5/30
20/20 [==============================] - 0s 3ms/step - loss: 0.5874 - accuracy:
0.6929 - val_loss: 0.5240 - val_accuracy: 0.7351
Epoch 6/30
20/20 [==============================] - 0s 3ms/step - loss: 0.5482 - accuracy:
0.7615 - val_loss: 0.6157 - val_accuracy: 0.7351
```

```
[11]: <tensorflow.python.keras.callbacks.History at 0x7fd7efe845d0>
```

2.7 [task-3] Experimenting with wider networks

▶ **Package pre-loading**

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

▶ **Code pre-loading**

```
[13]: def get_new_model(input_shape=input_shape):
          model = Sequential()
          model.add(Dense(10, activation='relu', input_shape=input_shape))
          model.add(Dense(10, activation='relu'))
          model.add(Dense(2, activation='softmax'))
          model.compile(optimizer='adam',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
          return model


      model_1 = get_new_model()
```

▶ **Task practice**

```
[14]: # 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()
```
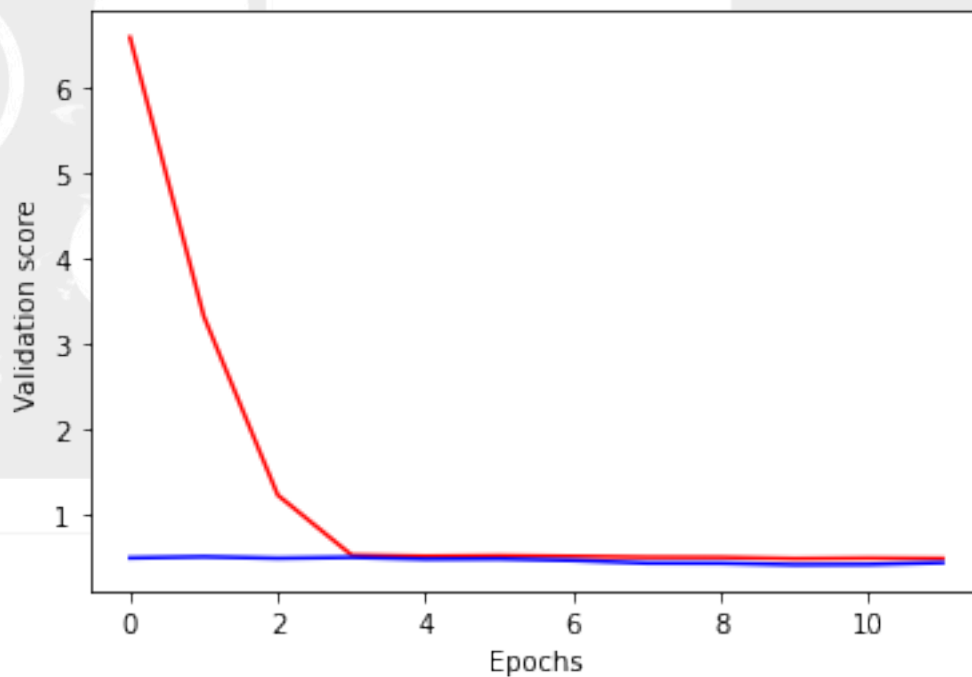

2.8 [task-4] Adding layers to a network

▶ **Code pre-loading**

```
[15]:  def get_new_model(input_shape=input_shape):
           model = Sequential()
           model.add(Dense(50, activation='relu', input_shape=input_shape))
```

```python
    model.add(Dense(2, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model



model_1 = get_new_model()
```

▶ **Network layers adding practice**

```
[16]:
# 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
```
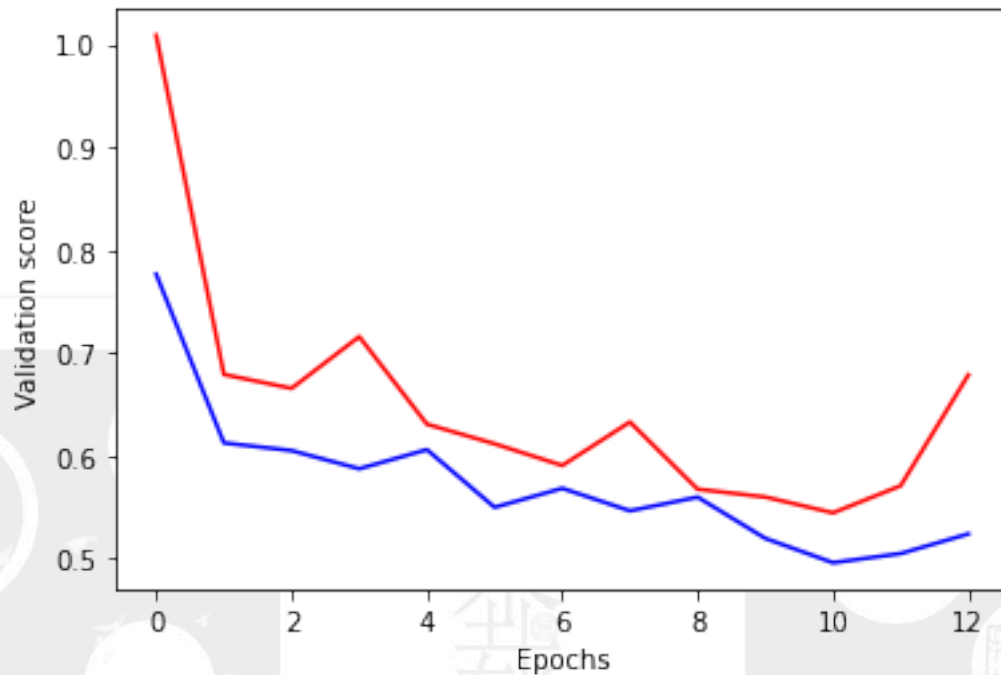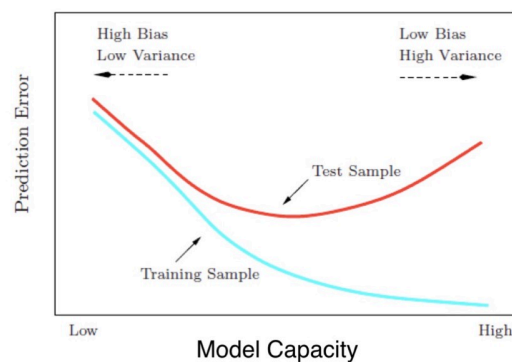
```python
plt.plot(model_1_training.history['val_loss'], 'r',
         model_2_training.history['val_loss'], 'b')
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```


3   Thinking about model capacity

3.1   [note-1] Overfitting

3.2   [note-2] 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 [note-3] Sequential experiments

| 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 [quiz-1] Practice question for experimenting with model structures:

- An experiment has run where has compared two networks that were identical 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 [note-1] Recognizing handwritten digits

- 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.
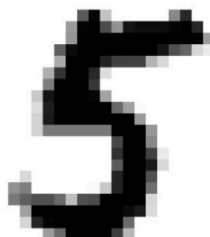
4.2 [task-1] Building an own digit recognition model

▶ **Package pre-loading**

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

▶ **Data pre-loading**

```python
[18]: df = pd.read_csv('../Datasets/3. MNIST.csv', header=None)
      X = df.iloc[:, 1:].to_numpy()
      y = to_categorical(df.iloc[:, 0])
```

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

```python
[19]: # 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 [==============================] - 1s 6ms/step - loss: 41.9168 - accuracy:
0.2303 - val_loss: 7.0109 - val_accuracy: 0.5740
```

```
[19]: <tensorflow.python.keras.callbacks.History at 0x7fd7b0148890>
```

5 Final thoughts

5.1 [note-1] 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.

6 Requirements

```python
from platform import python_version
import tensorflow as tf
import matplotlib

python_version = ('python=={}'.format(python_version()))
pandas_version = ('pandas=={}'.format(pd.__version__))
tensorflow_version = ('tensorflow=={}'.format(tf.__version__))
matplotlib_version = ('matplotlib=={}'.format(matplotlib.__version__))

writepath = '../../requirements.txt'
requirements = []
packages = [pandas_version, tensorflow_version, matplotlib_version]

with open(writepath, 'r') as file:
    for line in file:
        requirements.append(line.strip('\n'))

with open(writepath, 'a') as file:
    for package in packages:
        if package not in requirements:
            file.write(package + '\n')

max_characters = len(python_version)
for package in packages:
    if max(max_characters, len(package)) > max_characters:
        max_characters = max(max_characters, len(package))

print('#' * (max_characters + 8))
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * 2 + ' ' * 2 + python_version + ' ' *
      (max_characters - len(python_version) + 2) + '#' * 2)
for package in packages:
    print('#' * 2 + ' ' * 2 + package + ' ' *
          (max_characters - len(package) + 2) + '#' * 2)
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * (max_characters + 8))
```

```
#########################
##                     ##
##   python==3.7.9      ##
```

```
##   pandas==1.2.1       ##
##   tensorflow==2.4.1   ##
##   matplotlib==3.3.4   ##
##                       ##
#########################
```