SDU

Deep Learning

Fall
2021

Introduction to KERAS 2

Three API Styles

- **The Sequential Model**
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases

- **The functional API**
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases

- **Model subclassing**
  - Maximum flexibility
  - Larger potential error surface

The Functional API

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input

inputs = Input(shape=(10,))
x = Dense(64, activation=‘relu’)(inputs)
x = Dense(64, activation=‘relu’)(x)
output = Dense(10, activation=‘softmax’)(x)

Model = Model(inputs, output)
Model.compile(optimizer=‘rmsprop’,
                                        loss=‘categorical_crossentropy’,
                                        metrics=[‘accuracy’])
model.fit(x, y, epochs=10, batch_size=32)
```

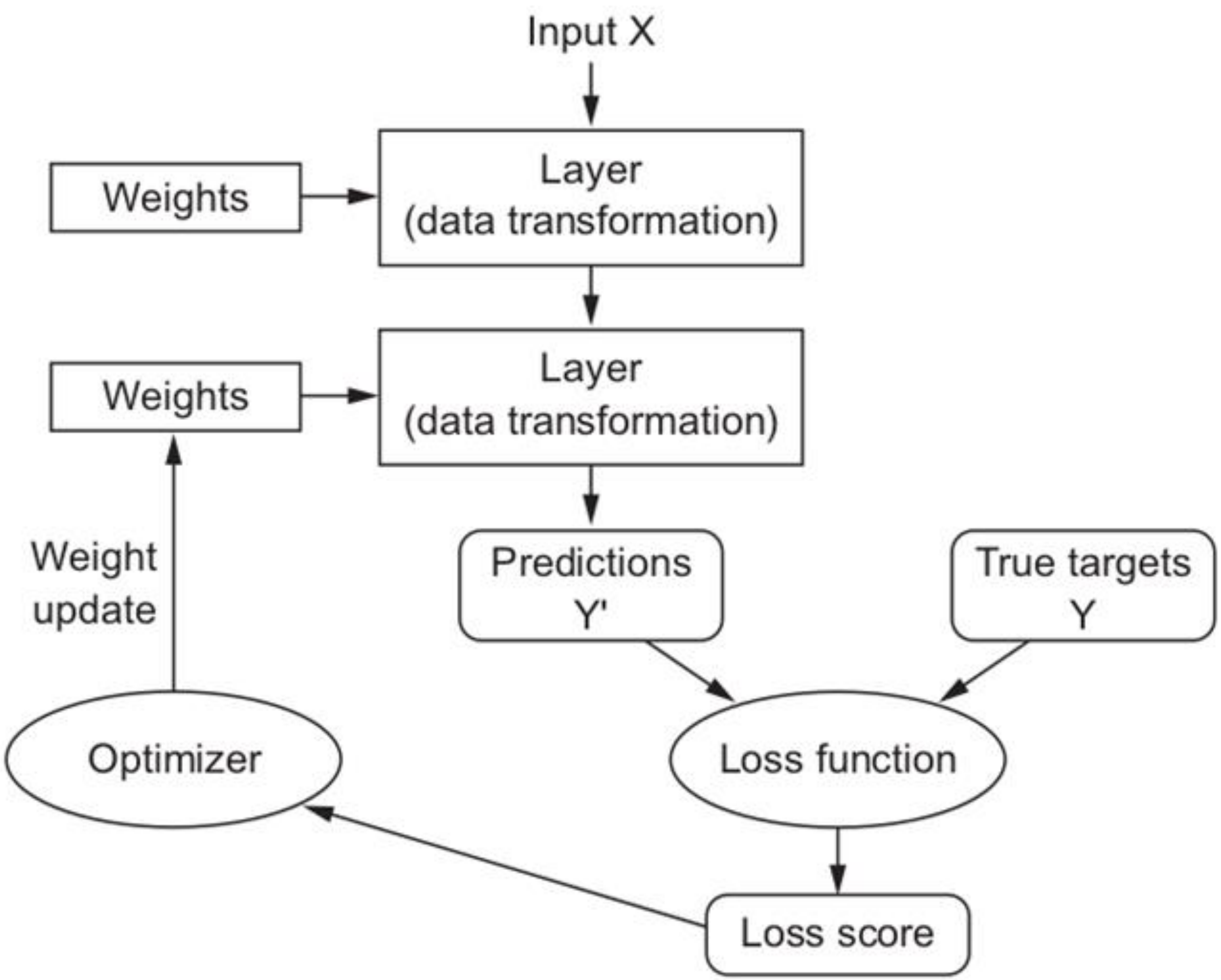
Defining the Model

```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

- The same model defined using the functional API:

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Alrighty, let's put it together



Regularization

- Adding weight regularization

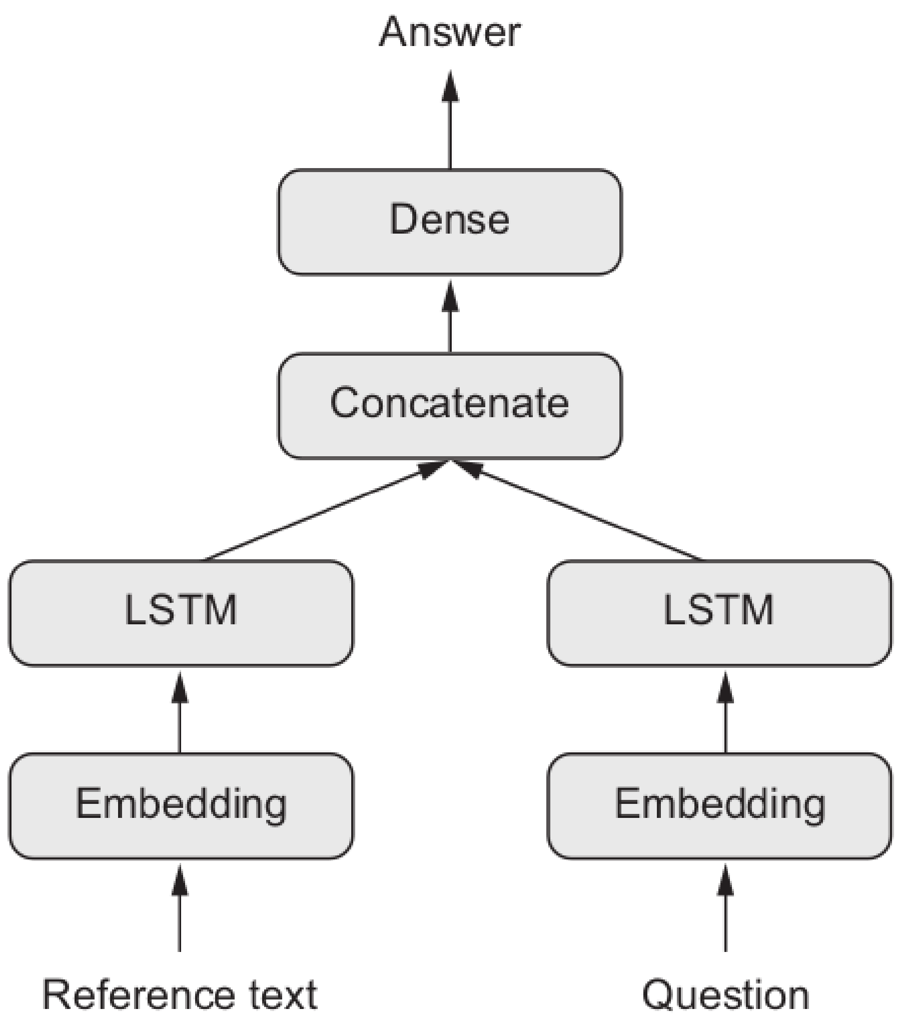
```
from tensorflow.keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16,        kernel_regularizer=regularizers.l2(0.001),
          activation='relu', input_shape=(10000,)))

model.add(layers.Dense(1, activation='sigmoid'))
```

  - l2(0.001) means every coefficient in the weight matrix of the layer will add 0.001 * weight_coefficient_value to the total loss of the network.
  - Note that because this penalty is only added at training time, the loss for this network will be much higher at training than at test time.

Dropout

```
model = models.Sequential()

model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(1, activation='sigmoid'))
```

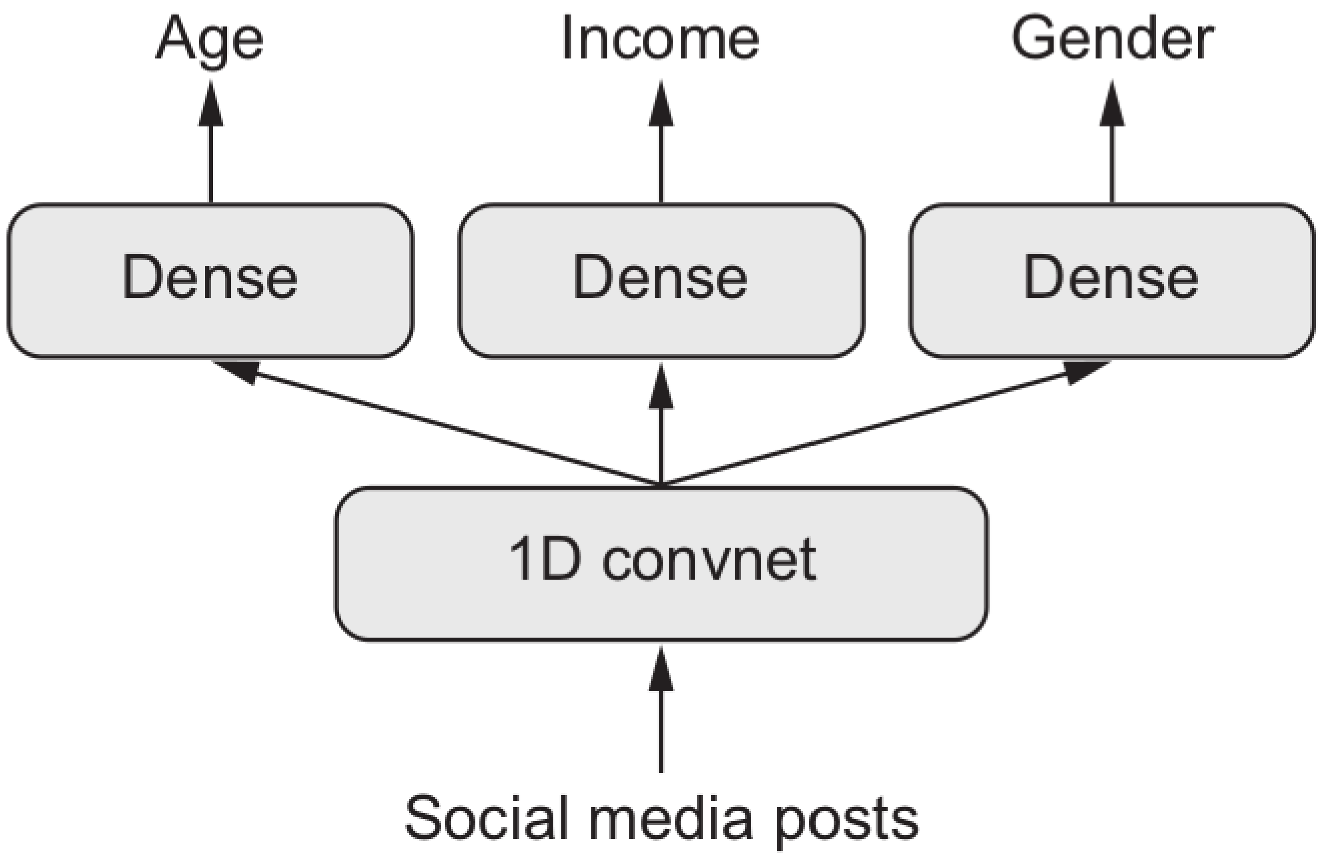
Multi Input Models



Multi Input Model using the functional API

```
text_input = Input(shape=(None,), dtype='int32', name='text')
embedded_text = layers.Embedding(64, text_vocabulary_size)(text_input)
encoded_text = layers.LSTM(32)(embedded_text)

question_input = Input(shape=(None,), dtype='int32', name='question')
embedded_question = layers.Embedding(32, question_vocabulary_size)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

concatenated = layers.concatenate([encoded_text, encoded_question], axis=-1)
answer = layers.Dense(answer_vocabulary_size, activation='softmax')(concatenated)


model = Model([text_input, question_input], answer)

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
```

Multi-Output Models



Multi-Output Models

```
posts_input = Input(shape=(None,), dtype='int32', name='posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
... # Construct the network how you like it
x = layers.Dense(128, activation='relu')(x)

age_prediction = layers.Dense(1, name='age')(x)
income_prediction = layers.Dense(num_income_groups, activation='softmax’,
                                                    name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)

model = Model(posts_input,[age_prediction, income_prediction,    gender_prediction])
```

Multi-Output Models

```
model.compile(optimizer='rmsprop’,
          loss=['mse', 'categorical_crossentropy', 'binary_crossentropy’])


model.compile(optimizer='rmsprop’,
          loss={'age': 'mse’,
                'income': 'categorical_crossentropy’,
                'gender': 'binary_crossentropy'})
```

- Fitting, etc, of the model remains the same as with a normal network.

Callbacks

- When training a model, there are many things you cannot predict
- Sometimes it would be helpful to intervene when something goes wrong
- Keras provides callbacks:
  - Model checkpointing: Saving the current weights of the model at different points during training.
  - Early stopping: Interrupting training when the validation loss is no longer improving (and of course, saving the best model obtained during training).
  - Dynamically adjusting the value of certain parameters during training: Such as the learning rate of the optimizer.
  - Logging training and validation metrics during training, or visualizing the representations learned by the model as they're updated: The Keras progress bar is a callback!

Implementing your own Callback

- Callbacks are implemented by sub-classing the class `keras.callbacks.Callback`.

- You can implement the following functions:
  - on_epoch_begin
  - on_epoch_end
  - on_batch_begin
  - on_batch_end
  - on_train_begin
  - on_train_end

TensorBoard / Weights and Biases

- Callbacks allow you to send run information to services that help you track and visualize runs.
- TensorBoard runs locally on a logs folder
  - Highly modular, but base features are lacking
- Weights and Biases runs in the cloud by sending the information to their servers. Only need to log onto their servers once on your local machine
  - No/Low modularity, but base feautures are rich and easy to setup

TensorBoard

```
tensorboard_callback = TensorBoard(log_dir=/path/to/logs/’, update_freq='batch')

model.fit(x=train_data, y=val_data, epochs=10, callbacks=[tensorboard_callback])
```

Terminal or CMD:

```
C:\Users\Name\Desktop>tensorboard --logdir /path/to/logs/
>> ‘TensorBoard 2.4.0 at http://localhost:6006/ (Press CTRL+C to quit)’
```

Weights and Biases

```
import wandb
wandb.init(project='project name)

from wandb.keras import
model.fit(x=train_data, y=val_data, epochs=10, callbacks=[WandbCallback()])
```

Early Stopping

```
import tensorflow.keras as keras

callbacks_list = [
        keras.callbacks.EarlyStopping(
                monitor='acc',
                patience=1),
        keras.callbacks.ModelCheckpoint(
                filepath='my_model.h5',
                monitor='val_loss',
                save_best_only=True)]
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

model.fit(x, y, epochs=10, batch_size=32, callbacks=callbacks_list,
        validation_data=(x_val, y_val))
```

Predictions

```
model = load_model('model.h5')
predictions = model.predict(test_data)

# Sigmoid (Boolean)
predictions = (predictions > 0.5).astype(int).reshape(test_labels.shape)

# Softmax (Multi class)
predictions = predictions.argmax(axis=-1)

wrong_data = test_data[test_labels != predictions]
wrong_preds = predictions[test_labels != predictions]
```

Data Generators

- Data generators allows you to import, train, and discard batches of data from memory.
- Provides the option for ‘on the fly’ data augmentation
- Useful for large datasets, which cannot be stored in memory

Data Generators

```
tf.keras.preprocessing.image.ImageDataGenerator(
  featurewise_center=False,
  samplewise_center=False,
  featurewise_std_normalization=False,
  samplewise_std_normalization=False,
  zca_whitening=False,
  zca_epsilon=1e-06,
  rotation_range=0,
  width_shift_range=0.0,
  height_shift_range=0.0,
  brightness_range=None,
  shear_range=0.0,
  zoom_range=0.0,
  channel_shift_range=0.0,
  fill_mode="nearest",
  cval=0.0,
  horizontal_flip=False,
  vertical_flip=False,
  rescale=None,
  preprocessing_function=None,
  data_format=None,
  validation_split=0.0,
  dtype=None)
```

- https://keras.io/api/preprocessing/image/

Data Generators

```
train_datagen = ImageDataGenerator(
        rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        'data/train', target_size=(150, 150), batch_size=32, class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        'data/validation', target_size=(150, 150), batch_size=32, class_mode='binary')

model.fit(train_generator, steps_per_epoch=2000, epochs=50, v
        alidation_data=validation_generator, validation_steps=800)
```

- https://keras.io/api/preprocessing/image/

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. Preparing your data
5. Define a model better than base-line
6. Scaling up: Make the model overfit
7. Regularizing your model and tuning your hyperparameters
8. Finalize your final model

Deep Learning Workflow

1. **Defining the problem and assembling a dataset**
   - What will your input data be?
   - What type of problem are you facing? Classification? Regression? …

1. Choosing a measure of success
2. Decide on the evaluation protocol
3. Preparing your data
4. Define a model better than base-line
5. Scaling up: Make the model overfit
6. Regularizing your model and tuning your hyperparameters
7. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. **Choosing a measure of success**
   - How do you measure if the model is successful
   - Not to be confused with the loss function which is often only a surrogate for what you actually want achieve

1. Decide on the evaluation protocol
2. Preparing your data
3. Define a model better than base-line
4. Scaling up: Make the model overfit
5. Regularizing your model and tuning your hyperparameters
6. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. **Decide on the evaluation protocol**
   - hold-out validation
   - Cross-validation

1. Preparing your data
2. Define a model better than base-line
3. Scaling up: Make the model overfit
4. Regularizing your model and tuning your hyperparameters
5. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. **Preparing your data**
   - Clean data
   - Normalize data

5. Define a model better than base-line
6. Scaling up: Make the model overfit
7. Regularizing your model and tuning your hyperparameters
8. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. Preparing your data
5. **Define a model better than base-line**
   - Last-layer activation
   - Loss function
   - Optimization Algorithm

1. Scaling up: Make the model overfit
2. Regularizing your model and tuning your hyperparameters
3. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. Preparing your data
5. Define a model better than base-line
6. **Scaling up: Make the model overfit**
   - Add layers.
   - Make the layers bigger.
   - Train for more epochs.

1. Regularizing your model and tuning your hyperparameters
2. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. Preparing your data
5. Define a model better than base-line
6. Scaling up: Make the model overfit
7. **Regularizing your model and tuning your hyperparameters**
   - This will take the most time
   - Add dropout.
   - Try different architectures: add or remove layers.
   - Add L1 and/or L2 regularization
8. Finalize your final model

Deep Learning Workflow

1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Decide on the evaluation protocol
4. Preparing your data
5. Define a model better than base-line
6. Scaling up: Make the model overfit
7. Regularizing your model and tuning your hyperparameters
8. **Finalize your final model**
    - Save and distribute the model