Keras Functional API

First Example

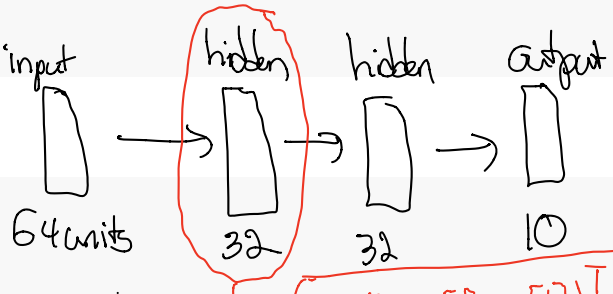
We will define the same model using the sequential API and the functional API (the code is from Chapter 7.1 of Chollet). The model has the following structure:

- Input layer has 64 features
- Two hidden layers, each with 32 units
- Output layer has 10 units with softmax activation

Sequential API

This is the interface to Keras we have used so far.

```
from keras import models
from keras import layers

seq_model = models.Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

seq_model.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
seq_model.fit(train_x, train_y,
  validation_data = (X_val, y_val),
  epochs = 1000, batch_size = 32)
```



Functional API

```
from keras import models
from keras import layers
from keras import Input

# input layer
input_tensor = Input(shape=(64,))

# first hidden layer
x = layers.Dense(32, activation='relu')(input_tensor)

# second hidden layer
x = layers.Dense(32, activation='relu')(x)

# output layer
output_tensor = layers.Dense(10, activation='softmax')(x)

# only now, put the pieces together into a model leading from input to output
model = models.Model(inputs = input_tensor, outputs = output_tensor)

# compile and fit as usual
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics = ['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=128)
```

$z^{[1]} = b^{[1]} + (w^{[1]})^T a^{[0]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$a^{[1]} = f(a^{[0]})$

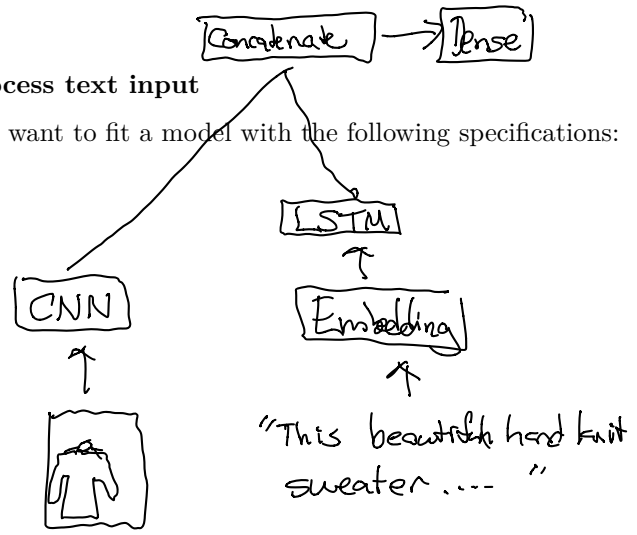
a function f(input_tensor)

could be written in 2 pieces:

f = layers.Dense(32, activation='relu')

x = f(input_tensor)

Multiple Inputs



Example 1: CNN to process image input, RNN to process text input

You work for an online store that sells clothing, like Etsy. You want to fit a model with the following specifications:

Inputs:

- a picture of a piece of clothing
- a text description of the clothing

Outputs:

- estimated sale price

We could use a network architecture like:

- a CNN to process the picture
- an RNN to process the text description
- combine activation output from the CNN and RNN
- dense output layer to predict price

Example 2: RNN to process two different text inputs

You work at a tech company, and are developing a user help page that will answer users' questions based on help documentation. You want to fit a model with the following specifications:

Inputs:

- a text question a user typed
- text of relevant help articles

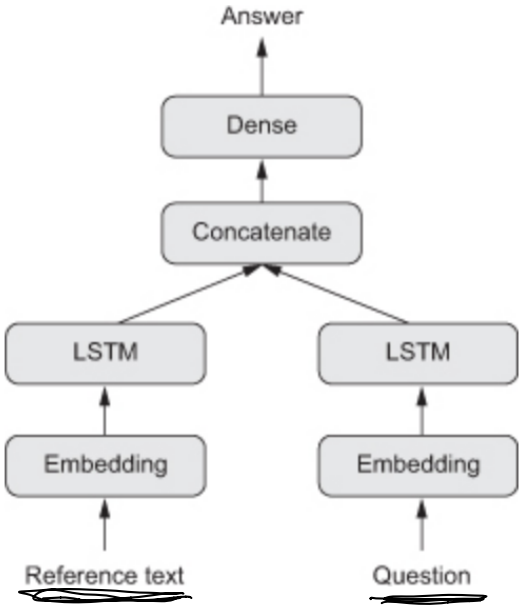
Outputs:

- answer to the question, which is one of K categories (for example, K-1 possible solutions, or else send them to a human being to answer the question.)

We could use a network architecture like:

- a first RNN to process the question
- a second RNN to process the text with the answers
- feed the output from the RNNs into a dense layer to generate a predicted class for the answer.

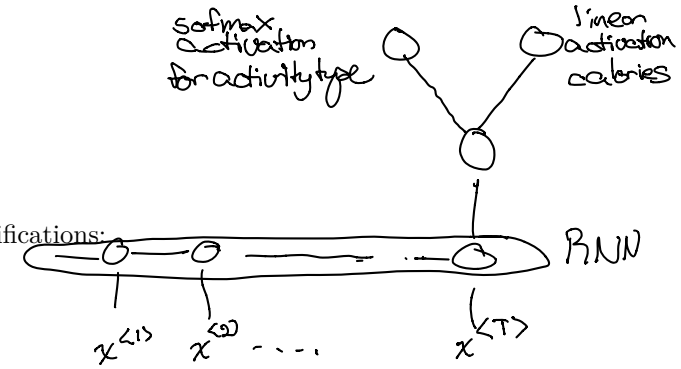
Figure from Secton 7.1 of Chollet



Example code for multiple inputs Example 2 from Chapter 7 of Chollet:

```
from keras.models import Model
from keras import layers
from keras import Input

text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

# first input layer is a text sequence.
# - We don't specify the length since it may differ across observations
# - data type is integer because we sparse-encoded the text
# - we can give the input layer a name if we want
text_input = Input(shape=(None,), dtype='int32', name='text')

# 64 dimensional word embedding of text input
embedded_text = layers.Embedding(64, text_vocabulary_size)(text_input)

# LSTM to process text
encoded_text = layers.LSTM(32)(embedded_text)

# second input layer is the question; same structure as for text input
question_input = Input(shape=(None,), dtype='int32', name='question')

# 32 dimensional word embedding of question input
embedded_question = layers.Embedding(32, question_vocabulary_size)(question_input)

# LSTM to process question
encoded_question = layers.LSTM(16)(embedded_question)

# concatenate the activation outputs from the text LSTM and the question LSTM
# axis = -1 means to concatenate along the last axis.
concatenated = layers.concatenate([encoded_text, encoded_question], axis=-1)

# the answer is one of a fixed number of categories.
# probabilities for these categories are calculated using a softmax activation
# acting on the concatenated output from the question and the reference text
answer = layers.Dense(answer_vocabulary_size,
    activation='softmax')(concatenated)

# The network now has two inputs!!
model = Model(inputs = [text_input, question_input], outputs = answer)

# Compile as usual
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])

# To fit, provide a dictionary with the values of the inputs
# names in the dictionary must match the names used when defining the layers
model.fit({'text': text, 'question': question},
  answers,
  epochs=10, batch_size=128)
```

Multiple Outputs



Example 1

You work for FitBit. You want to fit a model with the following specifications:

Input:

- measurements of acceleration over time recorded by a FitBit

Output:

- categorical activity type (lying down, sitting, standing, walking, running, bicycling, ...)
- numeric energy expenditure in calories

We could use a network architecture like:

- RNN to process the acceleration measurements from the FitBit
- A first dense output layer with softmax activation to classify activity type
- A second dense output layer with linear activation to estimate numeric energy expenditure

Example 2

You are kindof creepy (example from Chollet...). You want to take a series of social media posts from a single anonymous person and predict attributes of that person such as age, gender, and income level.

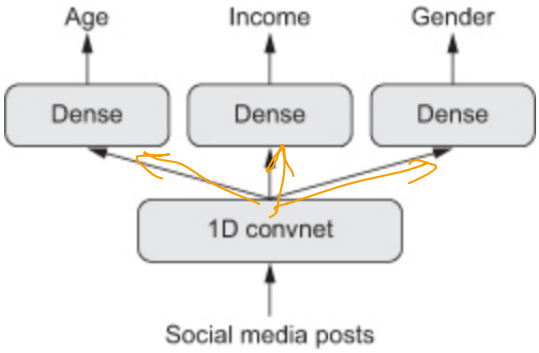
Input:

- social media posts

Output:

- age
- gender
- income level

Figure from Secton 7.1 of Chollet



Example code for multiple outputs Example 2 from Chapter 7 of Chollet:

```
from keras import layers
from keras import Input
from keras.models import Model
vocabulary_size = 50000
num_income_groups = 10

# input layer
posts_input = Input(shape=(None,), dtype='int32', name='posts')

# word embedding and hidden layers
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

# separate output layers for each target variable
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)

# define model with list of outputs
model = Model(inputs=posts_input,
              outputs=[age_prediction, income_prediction, gender_prediction])

# compile model, specifying dictionary of losses used for each output variable
# and their relative contributions to the final loss
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'},
              loss_weights={'age': 0.25,
                            'income': 1.,
                            'gender': 10.})

# fit model:
# x is posts
# y is dictionary of y's for each target variable
model.fit(posts, {'age': age_targets,
                  'income': income_targets,
                  'gender': gender_targets},
          epochs=10, batch_size=64)
```

$$J(b,w) = 0.25\, J^{age}(b,w) + 1 \cdot J^{income}(b,w) + 10 \cdot J^{gender}(b,w)$$

- Pick the loss weights so that the relative contributions to J from different tasks (regression for age, classification for income, classification for gender) are similar in magnitude, we want network to do well for all 3 tasks.