7.1 Keras Function API

17t4015s
内間けんじ

7.1.0 Keras Function APIの利用

7章以前でのNNはSepuentialモデル前提のものだったが、それでは**多入力モデル**や**多出力モデル**が扱えない。

→Kerasではそういったものを扱うための**Function API**が存在する。

7.1.1 Keras Function API概要

・SequentialモデルとFunctionAPIの比較

```
#Sequentialモデル
seq_model = 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'))
```

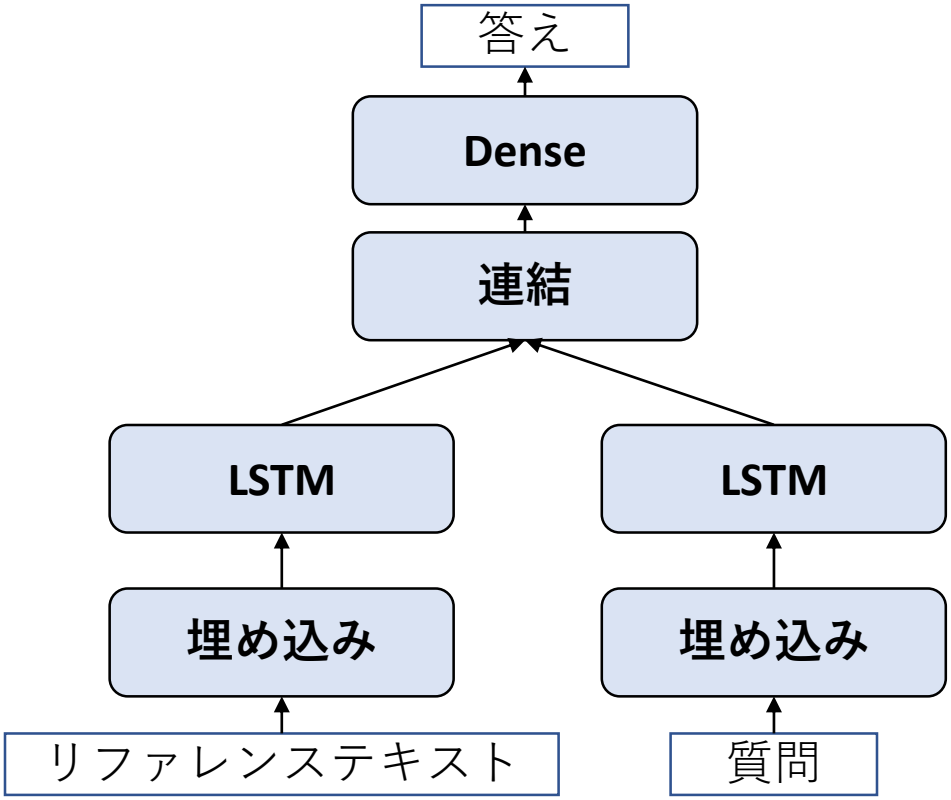
```
#Function API
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10,activation='softmax')(x)
```

7.1.1 Keras Function API

```
>>> func_model = Model(input_tensor, output_tensor)
>>> func_model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_11 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_12 (Dense)             (None, 10)                330
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
>>> _
```

7.1.2 多入力モデル

Function API では多入力モデルの構築が可能。
〇質問応答モデル



7.1.2 多入力モデル

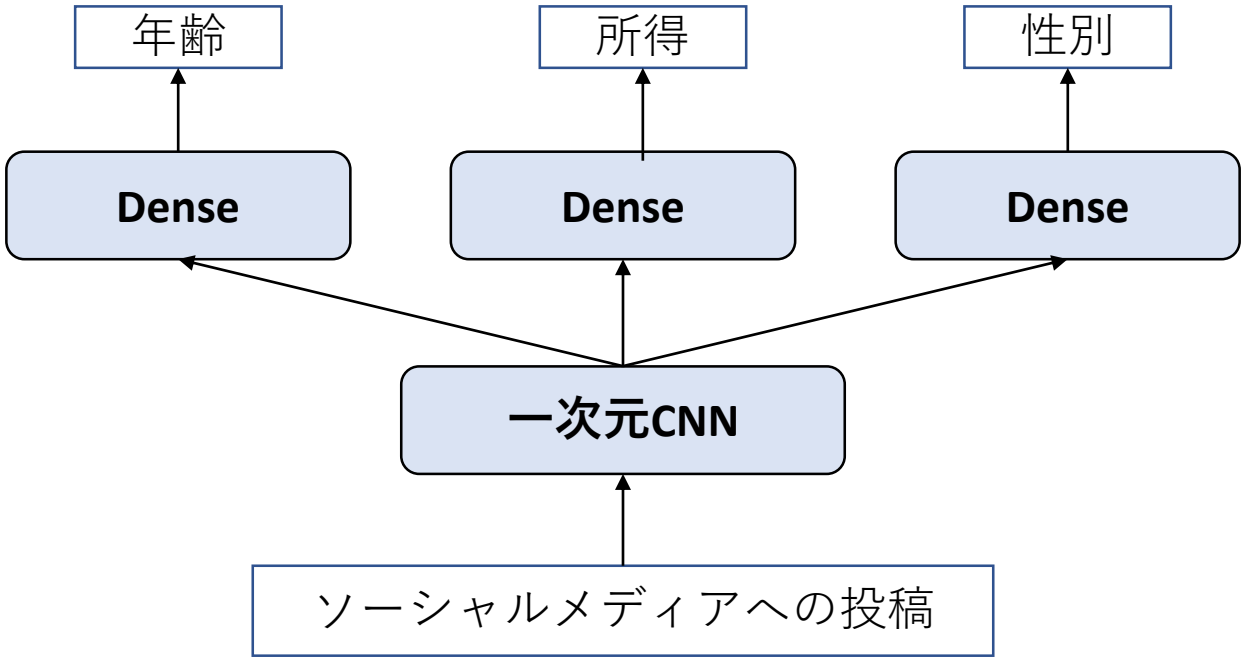
```
text_vocabulary_size = 10000
question_vocablary_saize = 10000
answer_vocabulary_size = 500

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

question_input = Input(shape=(None,),dtype='int32',name='question’)
embedded_question = layers.Embedding(question_vocablary_saize,32)(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)
```

7.1.2 多入力モデル

```
>>> model.summary()
_____________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=====================================================================================
text (InputLayer)            (None, None)              0
_____________________________________________________________________________________
question (InputLayer)        (None, None)              0
_____________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 64)          640000      text[0][0]
_____________________________________________________________________________________
embedding_2 (Embedding)      (None, None, 32)          320000      question[0][0]
_____________________________________________________________________________________
lstm_2 (LSTM)                (None, 32)                12416       embedding_1[0][0]
_____________________________________________________________________________________
lstm_3 (LSTM)                (None, 16)                3136        embedding_2[0][0]
_____________________________________________________________________________________
concatenate_2 (Concatenate)  (None, 48)                0           lstm_2[0][0]
                                                                   lstm_3[0][0]
_____________________________________________________________________________________
dense_1 (Dense)              (None, 500)               24500       concatenate_2[0][0]
=====================================================================================
Total params: 1,000,052
Trainable params: 1,000,052
Non-trainable params: 0
_____________________________________________________________________________________
>>>
```

7.1.3 多出力モデル

○ソーシャルメディアモデル



7.1.3 多出力モデル

```
vocabulary_size = 50000
num_income_groups = 10

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)
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)

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])
```

7.1.3 多出力モデル

```
>>> model.summary()
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=============================================================================================
posts (InputLayer)           (None, None)              0
_____________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 50000)       12800000    posts[0][0]
_____________________________________________________________________________________________
conv1d_1 (Conv1D)            (None, None, 128)         32000128    embedding_1[0][0]
_____________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D) (None, None, 128)       0           conv1d_1[0][0]
_____________________________________________________________________________________________
conv1d_2 (Conv1D)            (None, None, 256)         164096      max_pooling1d_1[0][0]
_____________________________________________________________________________________________
conv1d_3 (Conv1D)            (None, None, 256)         327936      conv1d_2[0][0]
_____________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D) (None, None, 256)       0           conv1d_3[0][0]
_____________________________________________________________________________________________
conv1d_4 (Conv1D)            (None, None, 256)         327936      max_pooling1d_2[0][0]
_____________________________________________________________________________________________
conv1d_5 (Conv1D)            (None, None, 256)         327936      conv1d_4[0][0]
_____________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 256)            0           conv1d_5[0][0]
_____________________________________________________________________________________________
age (Dense)                  (None, 1)                 257         global_max_pooling1d_1[0][0]
_____________________________________________________________________________________________
income (Dense)               (None, 10)                2570        global_max_pooling1d_1[0][0]
_____________________________________________________________________________________________
gender (Dense)               (None, 1)                 257         global_max_pooling1d_1[0][0]
=============================================================================================
Total params: 45,951,116
Trainable params: 45,951,116
Non-trainable params: 0
_____________________________________________________________________________________________
>>>
```

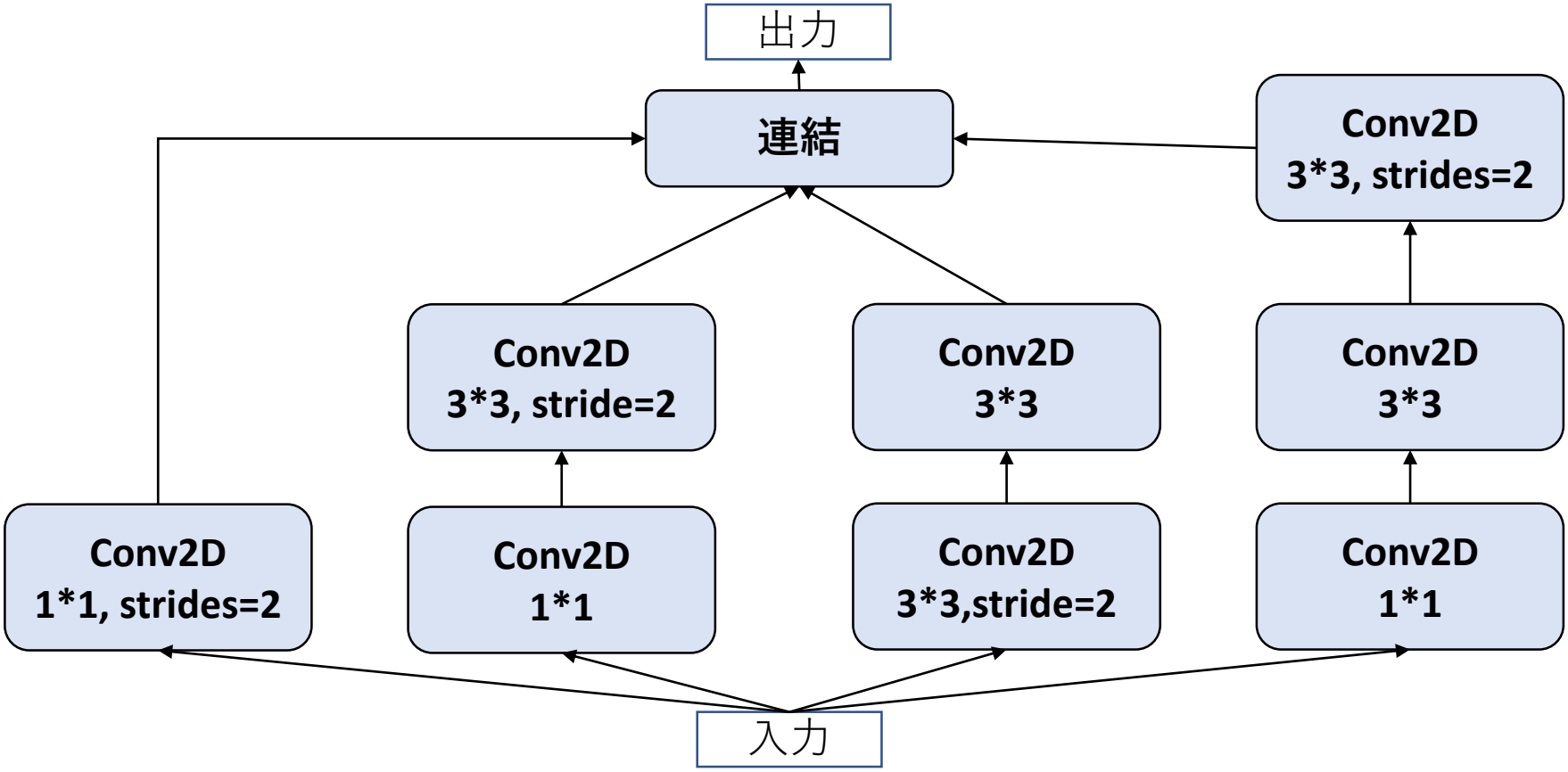
7.1.4 層の有向巡回グラフ

〇Inceptionモジュール

InceptionはCNN用のよく用いられるネットワークアーキテクチャ。

Christian Szegedyをはじめとすつ Googleの開発者によって2014年頃に開発された

7.1.4 層の有向巡回グラフ



(1*1)の畳み込み、 (3*3)の畳み込み、特徴量の連結といった3~4個の分岐で構成

7.1.4 層の有向巡回グラフ

4次元の入力テンソル(x)があるものとする

```
branch_a = layers.Conv2D(128,1,activation='relu',strides=2)(x)

branch_b = layers.Conv2D(128,1,activation='relu')(x)
branch_b = layers.Conv2D(128,3,activation=‘relu’,strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3,stride=2)(x)
branch_c = layers.Conv2D(128,3,activation=‘relu’)(branch_c)

branch_d = layers.Conv2D(128,1,activation=‘relu’)(x)
branch_d = layers.Conv2D(128,3,activation='relu')(branch_d)
branch_d = layers.Conv2D(128,1,activation='relu',strides=2)(branch_d)

output = layers.concatenate([branch_a,branch_b,branch_c,branch_d],axis=-1)
```

7.1.5 層の重みの共有

Function APIでは層のインスタンスを繰り返し再利用することができ、複数の分岐を共有するモデルを構築できる。

例えば、「文Aの文Bに対する類似度」を求めることは「文Bの文Aに対する類似度」を求めることであり対象である。このとき2つの入力文を別々のモデルで学習させるよりも単一の層で両方の入力を処理した方がよい

7.1.5 層の重みの共有

lstm層が同時に学習されるモデル**(共有LSTMモデル)**

```
lstm = layers.LSTM(32)

left_input = Input (shape=(None,128))
left_output = lstm(left_input)

right_input = Input(shape=(None,128))
right_output = lstm(right_input)

merged = layers.concatenate([left_output,right_output],axis=-1)
predictions = layers.Dense(1,activation='sigmoid')(merged)
```

7.1.5 層の重みの共有

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_7 (InputLayer)         (None, None, 128)         0         
_________________________________________________________________
input_8 (InputLayer)         (None, None, 128)         0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 32)                20608     
_________________________________________________________________
concatenate_3 (Concatenate)  (None, 64)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65        
=================================================================
Total params: 20,673
Trainable params: 20,673
Non-trainable params: 0
_________________________________________________________________
>>>
```

7.1.6 層としてのモデル

Function API では、モデルを「より大きな層」として扱うことができる。

```
xception_base = applications.Xception(weights = None,
include_top=False)

left_input = Input(shape=(250,250,3))
right_input = Input(shape=(250,250,3))

left_features = xception_base(left_input)
right_features = xception_base(right_input)

merged_features =
layers.concatenate([left_features,right_features],axis=-1)
```

7.1.6 層としてのモデル

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
input_5 (InputLayer)         (None, 250, 250, 3)       0
_________________________________________________________________
input_6 (InputLayer)         (None, 250, 250, 3)       0
_________________________________________________________________
xception (Model)             multiple                  20861480  input_5[0][0]
                                                                 input_6[0][0]
_________________________________________________________________
concatenate_3 (Concatenate)  (None, 8, 8, 4096)        0         xception[3][0]
                                                                 xception[4][0]
=================================================================
Total params: 20,861,480
Trainable params: 20,806,952
Non-trainable params: 54,528
_________________________________________________________________
>>>
```