제8장: 딥러닝모형의 성능향상

Soyoung Park

Pusan National University
Department of Statistics

soyoung@pusan.ac.kr

딥러닝모형의 모수추정의 문제

- 딥러닝 모형은 수천만개의 모수를 추정해야함

1.

2.

모수초기치(kernel initializers)와 활성함수

- 비선형 함수의 도입은 활성함수의 선택과 딥러닝 모수초기치 선택이 vanishing gradient/ exploding gradient 문제를 초래

1. Glorot 초기치

- linear, sigmoid, softmax, tanh 등 활성함수에 잘 작동하여 모형의 성능과 모수추정에 상당한 기여

2. He 초기치

- ReLu 활성함수

2. He 초기치

3. LeCun 초기치

정규화(Normalization)

- 딥러닝 모형의 수렴을 위해 자주 사용하는 기법
-
-

Batch Normalization

Instance Normalization

Layer Normalization

Layer Normalization for Convolutional Neural Network

If layer normalization is working on the outputs from a convolution layer, the math has to be modified slightly since it does not make sense to group all the elements from distinct channels together and compute the mean and variance. Each channel is considered as an "independent" sample and all the normalization was done for that specific channel only within the sample.

Assume the input tensor has shape $[m, H, W, C]$, for each channel $c \in \{1, 2, \cdots, C\}$

$$\mu_{i,c} = \frac{1}{HW} \sum_{j=1}^{H} \sum_{k=1}^{W} x_{i,j,k,c}$$

$$\sigma_{i,c}^2 = \frac{1}{HW} \sum_{j=1}^{H} \sum_{k=1}^{W} (x_{i,j,k,c} - \mu_{i,c})^2$$

$$\hat{x}_{i,j,k,c} = \frac{x_{i,j,k,c} - \mu_{i,c}}{\sqrt{\sigma_{i,c}^2 + \epsilon}}$$

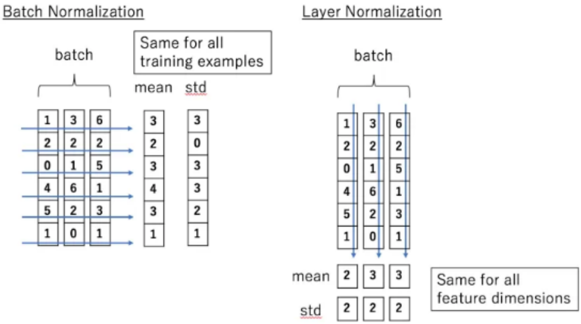
Specifically for each channel, we have learnable parameters γ_c and β_c, such that

$$y_{i,:,:,c} = \gamma_c \hat{x}_{i,:,:,c} + \beta_c \equiv \mathrm{LN}_{\gamma_c,\beta_c}(x_{i,:,:,c})$$

[1]

[1] https://leimao.github.io/blog/Layer-Normalization/

B-N vs. L-N



2

[2] https://blog.naver.com/baek2sm/222176799509

Normalization

- Normalization에서 정규화는 α와 β는 역전파에 의해 추정되고, 평균(μ)와 분산(σ^2)은 normalization 단위 별로 계산됨
    - B-N에서는 평균과 분산이 batch 단위로 계산됨
    - 즉,
- Validation 또는 test data에 적용시킬때, 어떤 평균과 분산을 사용해야 하는지에 대한 문제 발생
    -
    -

Normalization

- 정규화는 vanishing gradient문제를 방지하는 중요한 수단을 가짐
  -
  -

Fashion MNIST data

fashion_mnist.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
[ ] import tensorflow as tf
    import matplotlib.pyplot as plt
    fashion_mnist=tf.keras.datasets.fashion_mnist
```

```
[ ] (x_train, y_train_sparse),(x_test, y_test_sparse)=fashion_mnist.load_data()
    print(x_train.shape, x_test.shape)
    from tensorflow.keras.utils import to_categorical, plot_model
    y_train=to_categorical(y_train_sparse)
    y_test=to_categorical(y_test_sparse)
    print(y_test.shape)
```

```
x_train=x_train/255.0
x_test=x_test/255.0
class_names=['T-shirt/top', 'Trouser','Pullover', 'Dress','Coat','Sandal','Shirt','Sneaker','Bag','AnkleBoot']
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(x_train[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[y_train_sparse[i]])
plt.show()
```

Fashion MNIST data

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Dense,Activation,LeakyReLU,PReLU,BatchNormalization
model=Sequential()
model.add(Flatten(input_shape=[28,28]))
model.add(Dense(256, activation='elu', kernel_initializer='he_normal'))
model.add(BatchNormalization(momentum=0.9))
model.add(Dense(128, activation='selu', kernel_initializer='lecun_normal'))
model.add(BatchNormalization(center=False, scale=False))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_1 (Dense)             (None, 256)               200960

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_2 (Dense)             (None, 128)               32896

 batch_normalization_2 (Batc  (None, 128)              256
 hNormalization)

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 236,426
Trainable params: 235,658
Non-trainable params: 768
_________________________________________________________________
```

Fashion MNIST data

```
[ ] model.compile(loss='categorical_crossentropy',optimizer='nadam',metrics=['accuracy'])
    results=model.fit(x_train, y_train, batch_size=32, epochs=10, validation_split=0.1)
```

```
Epoch 1/10
1688/1688 [==============================] - 17s 8ms/step - loss: 0.4747 - accuracy: 0.8305 - val_loss: 0.3974 - val_accuracy: 0.8565
Epoch 2/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.3655 - accuracy: 0.8673 - val_loss: 0.3781 - val_accuracy: 0.8663
Epoch 3/10
1688/1688 [==============================] - 14s 8ms/step - loss: 0.3282 - accuracy: 0.8803 - val_loss: 0.3223 - val_accuracy: 0.8837
Epoch 4/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.3010 - accuracy: 0.8888 - val_loss: 0.3273 - val_accuracy: 0.8803
Epoch 5/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2800 - accuracy: 0.8964 - val_loss: 0.3134 - val_accuracy: 0.8882
Epoch 6/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2635 - accuracy: 0.9024 - val_loss: 0.3180 - val_accuracy: 0.8853
Epoch 7/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2509 - accuracy: 0.9064 - val_loss: 0.3171 - val_accuracy: 0.8892
Epoch 8/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2387 - accuracy: 0.9104 - val_loss: 0.3086 - val_accuracy: 0.8863
Epoch 9/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2248 - accuracy: 0.9158 - val_loss: 0.3208 - val_accuracy: 0.8843
Epoch 10/10
1688/1688 [==============================] - 13s 8ms/step - loss: 0.2163 - accuracy: 0.9188 - val_loss: 0.3413 - val_accuracy: 0.8848
```

Fashion MNIST data

```
[ ] import pandas as pd
```

```
[12] pred = model.predict(x_train[0:5,])
     pd.DataFrame(pred).round(2)
```

```
[27] categorical_test_labels = pd.DataFrame(y_train[0:5,]).idxmax(axis=1)
     categorical_pred = pd.DataFrame(pred).idxmax(axis=1)
```

```
[34] from sklearn.metrics import confusion_matrix, plot_confusion_matrix
```

```
categorical_pred.tolist()
```

```
[30] categorical_test_labels.tolist()
```

```
[31] conf_matrix = confusion_matrix(categorical_test_labels.tolist(), categorical_pred.tolist())
```

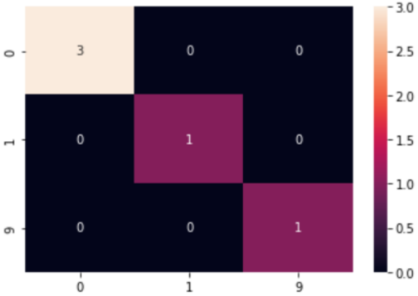
```
[32] conf = pd.DataFrame(conf_matrix, index = [i for i in (0,1,9)], columns=[i for i in (0,1,9)])
```

```
conf
```

Fashion MNIST data

```python
import seaborn as sn
sn.heatmap(conf, annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f919eeead90>



Fashion MNIST data

```
pred2 = model.predict(x_train)
pred2
```

```
[36] pred2.shape, y_train.shape
```

```
((60000, 10), (60000, 10))
```

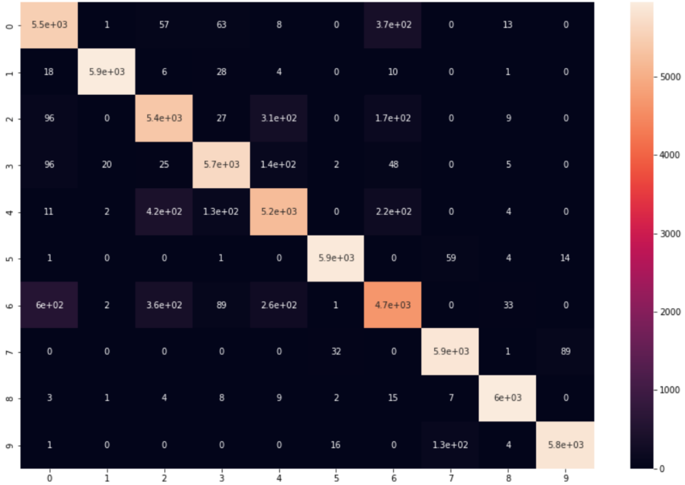
```
[37] cate_test_labels = pd.DataFrame(y_train).idxmax(axis=1)
     cate_pred = pd.DataFrame(pred2).idxmax(axis=1)
```

```
[38] conf_matrix_2 = confusion_matrix(cate_test_labels.astype(str), cate_pred.astype(str))
     conf_matrix_2
```

```
[39] plt.figure(figsize=(15,10))
     sn.heatmap(conf_matrix_2, annot=True)
```

Fashion MNIST data

<matplotlib.axes._subplots.AxesSubplot at 0x7f921e0d63d0>



Dropout

- Dropout은 과대적합이 발생했을 때, 규제화하는 방법
- 각 층의 활성함수 이전 또는 이후에 일정비율의 노드를 임의로 제거하고, 학습을 시키는 방법
-
- 일반적으로 10 - 50%를 dropout $\rightarrow$

Fashion MNIST data

Dropout

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, BatchNormalization,Dropout
model=Sequential()
model.add(Flatten(input_shape=[28,28]))
model.add(Dropout(0.1))
model.add(Dense(256, activation='elu', kernel_initializer='he_normal'))
model.add(Dropout(0.1))
model.add(Dense(128, activation='selu', kernel_initializer='lecun_normal'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 784)               0

 dropout_1 (Dropout)         (None, 784)               0

 dense_4 (Dense)             (None, 256)               200960

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 10)                1290

=================================================================
Total params: 235,146
```

Fashion MNIST data

```
model.compile(loss='categorical_crossentropy',optimizer='nadam',metrics=['accuracy'])
results=model.fit(x_train, y_train, batch_size=32, epochs=10, validation_split=0.1)
```

```
Epoch 1/10
1688/1688 [==============================] - 11s 6ms/step - loss: 0.5153 - accuracy: 0.8131 - val_loss: 0.4003 - val_accuracy: 0.8448
Epoch 2/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.4023 - accuracy: 0.8533 - val_loss: 0.4400 - val_accuracy: 0.8315
Epoch 3/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.3670 - accuracy: 0.8640 - val_loss: 0.3657 - val_accuracy: 0.8655
Epoch 4/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.3447 - accuracy: 0.8712 - val_loss: 0.3335 - val_accuracy: 0.8780
Epoch 5/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.3291 - accuracy: 0.8778 - val_loss: 0.3381 - val_accuracy: 0.8783
Epoch 6/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.3149 - accuracy: 0.8820 - val_loss: 0.3312 - val_accuracy: 0.8798
Epoch 7/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.3044 - accuracy: 0.8863 - val_loss: 0.3166 - val_accuracy: 0.8830
Epoch 8/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2953 - accuracy: 0.8879 - val_loss: 0.3218 - val_accuracy: 0.8770
Epoch 9/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2841 - accuracy: 0.8928 - val_loss: 0.3113 - val_accuracy: 0.8902
Epoch 10/10
1688/1688 [==============================] - 10s 6ms/step - loss: 0.2802 - accuracy: 0.8936 - val_loss: 0.3319 - val_accuracy: 0.8803
```

Regularization

- 과대적합이 발생했을 때, 가장 확실한 방법 중 하나는 규제화 (regularization)를 통한 은닉층의 모수를 줄이는 것
    -
    -

L_1, L_2 규제화