Tensorflow More for GAN

MNIST 분류 모델 리뷰 I - DNN

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1) / 255
x_test = x_test.reshape(-1, 28, 28, 1) / 255
print(x_train.shape, y_train.shape)
```

```
tf.keras.backend.clear_session()

x = tf.keras.Input(shape=(28, 28, 1))
h = tf.keras.layers.Flatten()(x)
h = tf.keras.layers.Dense(32, activation='swish')(h)
h = tf.keras.layers.Dense(32, activation='swish')(h)
y = tf.keras.layers.Dense(10, activation="softmax")(h)

model = tf.keras.Model(x, y)
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')
model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.1)
```

MNIST 분류 모델 리뷰 II - CNN

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1) / 255
x_test = x_test.reshape(-1, 28, 28, 1) / 255
print(x_train.shape, y_train.shape)
```

```
tf.keras.backend.clear_session()

x = tf.keras.Input(shape=(28, 28, 1))
h = tf.keras.layers.Conv2D(32, 3, 2, activation='swish')(x)
h = tf.keras.layers.Conv2D(64, 3, 2, activation='swish')(h)
h = tf.keras.layers.Conv2D(128, 6, activation='swish')(h)
h = tf.keras.layers.Flatten()(h)
y = tf.keras.layers.Dense(10, activation="softmax")(h)

model = tf.keras.Model(x, y)
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')
model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.1)
```

TF Dataset 사용

```
datasets = tf.data.Dataset.from_tensor_slices((x_train, y_train))
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')
model.fit(datasets.shuffle(1000).batch(128), epochs=10, batch_size=128)
```

batch를 분리해서 직접 학습하기

```
datasets = tf.data.Dataset.from_tensor_slices((x_train, y_train))
for e in range(10):
    print(f"{e} epochs")
    for i, (x_batch, y_batch) in enumerate(datasets.shuffle(1000).batch(128)):
        result = model.train_on_batch(x_batch, y_batch)
        if i % 50 == 0:
            print(f"{i} batch, {result}")

    model.evaluate(x_test, y_test)
    print()
```

MNIST 분류 모델 리뷰 II - CNN

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_train = x_train.reshape(-1, 28, 28, 1) / 255
x_test = x_test.reshape(-1, 28, 28, 1) / 255
print(x_train.shape, y_train.shape)
```

```
tf.keras.backend.clear_session()

x = tf.keras.Input(shape=(28, 28, 1))
h = tf.keras.layers.Conv2D(32, 3, 2, activation='swish')(x)
h = tf.keras.layers.Conv2D(64, 3, 2, activation='swish')(h)
h = tf.keras.layers.Conv2D(128, 6, activation='swish')(h)
h = tf.keras.layers.Flatten()(h)
y = tf.keras.layers.Dense(10, activation="softmax")(h)

model = tf.keras.Model(x, y)
model.compile(loss='sparse_categorical_crossentropy', metrics='accuracy')
model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.1)
```

Custom Model Class의 이용

```
def make_cnn():
    x = tf.keras.Input(shape=[28, 28, 1])
    h = tf.keras.layers.Conv2D(32, 3, 2, activation='swish')(x)
    h = tf.keras.layers.Conv2D(64, 3, 2, activation='swish')(h)
    h = tf.keras.layers.Flatten()(h)
    y = tf.keras.layers.Dense(10, activation="softmax")(h)
    return tf.keras.Model(x, y)
```

```
tf.keras.backend.clear_session()
cnn = make_cnn()
model = Custom(cnn)
model.fit(x_train, y_train, epochs=10, batch_size=128)
```

1. model.fit 함수를 이용할 수 있는 방향으로 class를 구성을 한 실습 예제.
2. class 구성에 따라 model 이용 방법은 달라질 수 있다.
3. Model Class를 자유롭게 사용하기 위해서는 다음의 공부가 더 필요합니다.
    a. python class
    b. Tensorflow Model

Class Custom

학습하는 부분을 직접 컨트롤 하기 위하여 custom model class를 직접 작성한다.

```
class Custom(tf.keras.Model):
    def __init__(self, cnn):
        super(Custom, self).__init__()
        self.cnn = cnn

        self.optimizer = tf.keras.optimizers.Adam()
        self.custom_loss = tf.keras.losses.SparseCategoricalCrossentropy()

        self.compile()

    def train_step(self, batch):
        x_batch, y_batch = batch

        with tf.GradientTape() as tape:
            y_pred = self.model(x_batch)
            loss = self.custom_loss(y_batch, y_pred)

        grads = tape.gradient(loss, self.model.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))

        return {'loss': loss}
```

accuracy metric 추가

```
class Custom(tf.keras.Model):
    def __init__(self, cnn):
        super(Custom, self).__init__()
        self.cnn = cnn

        self.optimizer = tf.keras.optimizers.Adam()
        self.custom_loss = tf.keras.losses.SparseCategoricalCrossentropy()

        self.loss_metric = tf.keras.metrics.Mean()
        self.acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()

        self.compile()

    def update_metrics(self, loss, y_batch, y_pred):
        self.loss_metric.update_state(loss)
        self.acc_metric.update_state(y_batch, y_pred)

    def train_step(self, batch):
        x_batch, y_batch = batch

        with tf.GradientTape() as tape:
            y_pred = self.model(x_batch)
            loss = self.custom_loss(y_batch, y_pred)

        grads = tape.gradient(loss, self.model.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))

        self.update_metrics(loss, y_batch, y_pred)

        return {'loss': self.loss_metric.result(), 'acc': self.acc_metric.result()}
```

train step

1. batch 단위 학습 로직
2. grads – batch 입력에 대해서 계산된 미분값
3. apply_gradients – grads 값을 이용하여 Weight를 조정한다.
4. 출력할 log 내용을 dictionary 형태로 return 한다.

```
def train_step(self, batch):
    x_batch, y_batch = batch

    with tf.GradientTape() as tape:
        y_pred = self.model(x_batch)
        loss = self.custom_loss(y_batch, y_pred)

    grads = tape.gradient(loss, self.model.trainable_weights)
    self.optimizer.apply_gradients(zip(grads, self.model.trainable_weights))

    self.update_metrics(loss, y_batch, y_pred)
    return {'loss': self.loss_metric.result(), 'acc': self.acc_metric.result()}
```

$$grads = dloss/dW$$

validation/evaluation 을 위한 코드 추가

```
def test_step(self, batch):
    x_batch, y_batch = batch
    y_pred = self.model(x_batch)
    loss = self.custom_loss(y_batch, y_pred)
    self.update_metrics(loss, y_batch, y_pred)
    return {'loss': self.loss_metric.result(), 'acc': self.acc_metric.result()}
```

test_step 함수가 작성되면 다음의 코드를 이용할 수 있다.

```
tf.keras.backend.clear_session()

cnn = make_cnn()
model = Custom(cnn)
model.fit(x_train, y_train, epochs=10, batch_size=128, validation_split=0.1)
model.evaluate(x_test, y_test)
```

GradientTape

https://www.tensorflow.org/api_docs/python/tf/GradientTape

GradientTape - 미분값을 구하는 도구

$$y = x^2$$

$$dy/dx = 2x$$

```
x = tf.constant(3.0)
with tf.GradientTape() as g:
    g.watch(x)
    y = x * x

dy = g.gradient(y, x)  # dy = 2 * x at x = 3
print(dy)
```

이차 도함수 구하기

$$y = x^2$$
$$dy/dx = 2x$$
$$d^2y/dx^2 = 2$$

```
x = tf.constant(5.0)
with tf.GradientTape() as g:
    g.watch(x)
    with tf.GradientTape() as gg:
        gg.watch(x)
        y = x * x

    dy = gg.gradient(y, x)  # dy = 2 * x at x = 5
    print(dy)

ddy = g.gradient(dy, x)  # ddy = 2 at x = 5
print(ddy)
```

이차 도함수 구하기

$$y = x^2$$
$$z = y^2$$
$$dy/dx = 2x$$
$$dz/dy = 2y$$
$$dz/dx = 4x + 3$$

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as g:
    g.watch(x)
    y = x * x
    z = y * y

dy_dx = g.gradient(y, x) # dy_dx = 2 * x at x = 3
print(dy_dx)

dz_dy = g.gradient(z, y) # dy_dy = 2 * y at y = 9
print(dz_dy)

dz_dx = g.gradient(z, x)  # dz_dx = 4 * x ^ 3 at x = 3
print(dz_dx)
```

Callback.on_epoch_end

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback#on_epoch_end

on_epoch_end

epochs 단위 학습이 끝날 때 실행

```
import matplotlib.pyplot as plt
import numpy as np

class Monitor(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        sample = np.random.randint(1, 10000, 5)
        y_pred = self.model.model.predict(x_test[sample])

        for i, rnd in enumerate(sample):
            plt.subplot(1, 5, 1 + i)
            plt.axis('off')
            plt.title(f"{rnd}:{np.argmax(y_pred[i])}")
            plt.imshow(x_test[rnd].reshape(28, 28), cmap='gray_r')
        plt.show()
```

```
tf.keras.backend.clear_session()

cnn = make_cnn()
model = Custom(cnn)
model.fit(x_train, y_train, epochs=10, batch_size=128, callbacks=[Monitor()])
```

numpy.random.randint [link]

```
random.randint(low, high=None, size=None, dtype=int)
```

size 개수 만큼의 low(포함)에서 high(제외) 까지 임의의 정수를 반환합니다.

```
def on_epoch_end(self, epoch, logs=None):
    sample = np.random.randint(1, 10000, 5)
    y_pred = self.model.model.predict(x_test[sample])

    for i, rnd in enumerate(sample):
        plt.subplot(1, 5, 1 + i)
        plt.axis('off')
        plt.title(f"{rnd}:{np.argmax(y_pred[i])}")
        plt.imshow(x_test[rnd].reshape(28, 28), cmap='gray_r')
    plt.show()
```

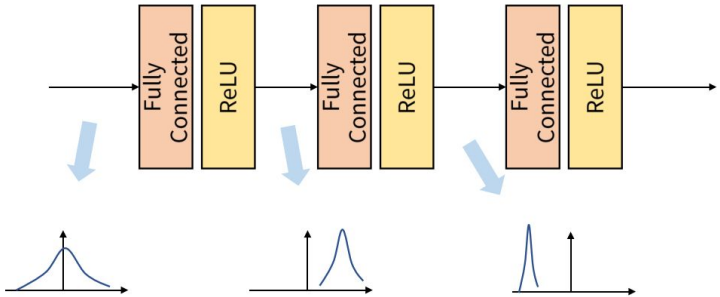
BatchNormalization

https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization

Batch Normalization

- Layer 사이에 중간 결과 데이터들을 batch 단위로 표준정규화하는 계층
  - 학습 단계에서는 batch 단위로 정규화하고
  - 모델 적용시의 σ, μ는 사용하던 값들의 이동평균값을 이용한다.
- 모델의 학습을 속도를 획기적으로 빠르게 한다.
- parameter에 대한 민감도가 낮아진다.
- 일반적으로 activation 전에 normalization을 한다.

$$BN(\mathbf{x}_i) = \gamma \odot \left(\frac{\mathbf{x}_i - \mu_B}{\sigma_B}\right) + \beta,$$



https://arxiv.org/pdf/1805.11604.pdf

Batch Normalization

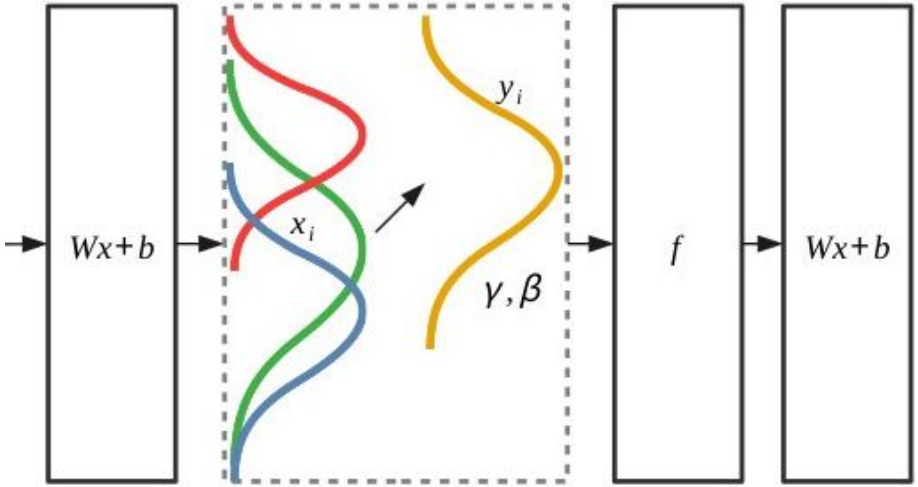
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

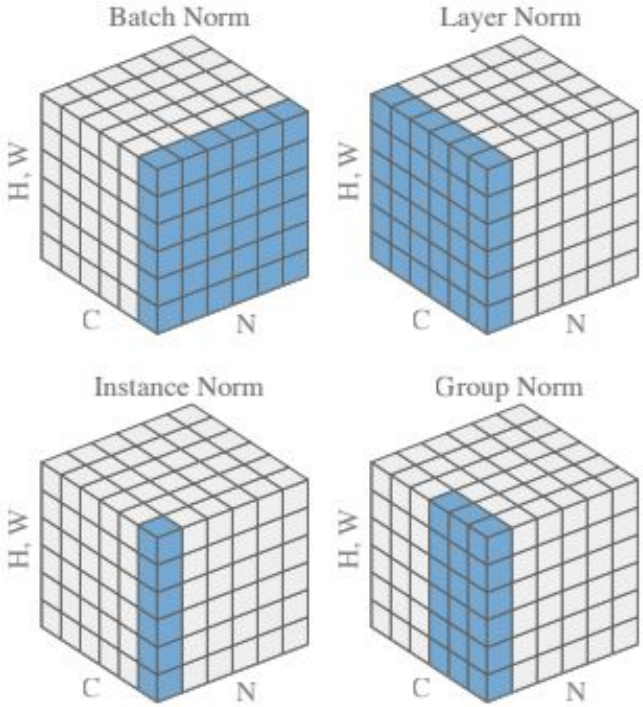
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

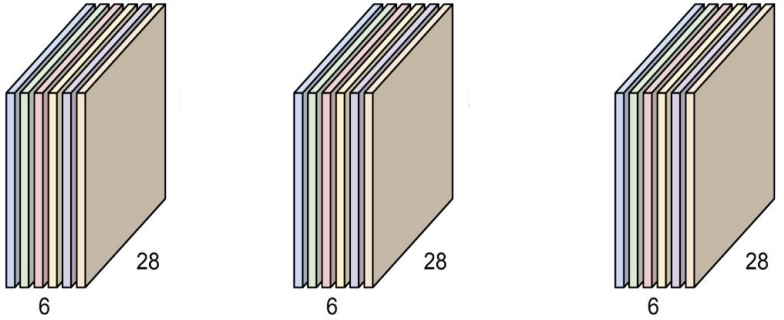
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



그 외의 Normalization들





(batch: 3, channel: 6, width, height: 28, 28)

- batchNorm: batch 내의 같은 색 channel을 모아서 Norm
- LayerNorm: 각 batch 별로 channel을 모아서 Norm
- InstanceNorm: batch 내의 channel 별로 Norm
- groupNorm: 각 batch 별로 channel을 그룹지어서 Norm