Tensorflow More for GAN

MNIST 분류 모델 리뷰 I - DNN

```python
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)
```

```
(60000, 28, 28, 1) (60000,)
```

```python
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)
```

sparse_categorical_crossentropy
종속변수의 원핫인코딩을
생략할 수 있는 Loss 함수

MNIST 분류 모델 리뷰 II - CNN

```python
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)
```

```
(60000, 28, 28, 1) (60000,)
```

```python
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)
```

sparse_categorical_crossentropy

종속변수의 원핫인코딩을
생략할 수 있는 Loss 함수

TF Dataset 사용

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

batch를 분리해서 직접 학습하기

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

train_on_batch 대신 GradientTape 사용하기

```python
model = tf.keras.Model(x, y)

# model.compile을 하지 않음.
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()
accuracy = tf.keras.metrics.SparseCategoricalAccuracy()

for e in range(10):
    print(f"{e} epochs")
    accuracy.reset_states()
    for i, (x_batch, y_batch) in enumerate(datasets):
        with tf.GradientTape() as tape:
            y_pred = model(x_batch, training=True)
            loss = loss_fn(y_batch, y_pred)
            grads = tape.gradient(loss, model.trainable_weights)
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
        accuracy.update_state(y_batch, y_pred)

        if i % 50 == 0:
            print(f"{i} batch, {loss}, {accuracy.result()}")

    print()
```

GradientTape를 이용할 때에는 학습에 사용하는 model을 이용할 때에 "training=True"를 반드시 넣어주어야 한다.

@tf.function

```python
model = tf.keras.Model(x, y)

# model.compile을 하지 않음.
loss_fn = tf.keras.losses.SparseCateg
optimizer = tf.keras.optimizers.Adam(
accuracy = tf.keras.metrics.SparseCat

@tf.function
def train_step(batch):
    x_batch, y_batch = batch
    with tf.GradientTape() as tape:
        y_pred = model(x_batch, training=True)
        loss = loss_fn(y_batch, y_pred)
        grads = tape.gradient(loss, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
    accuracy.update_state(y_batch, y_pred)

    return loss, accuracy.result()
```

```python
for e in range(10):
    print(f"{e} epochs")
    accuracy.reset_states()
    for i, batch in enumerate(datasets):
        loss, acc = train_step(batch)

        if i % 50 == 0:
            print(f"{i} batch, {loss}, {acc}")

    print()
```

Custom Model Class의 이용

```python
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)
```

```python
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.

```python
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.cnn(x_batch, training=True)
            loss = self.custom_loss(y_batch, y_pred)
            grads = tape.gradient(loss, self.cnn.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.cnn.trainable_weights))

        return {'loss': loss}
```

accuracy metric 추가

```python
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.

```python
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

```python
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.

```python
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

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

```python
import tensorflow as tf

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

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

```python
import tensorflow as tf

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

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

```python
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

```python
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.cnn.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()
```

```python
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

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

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

```python
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()
```
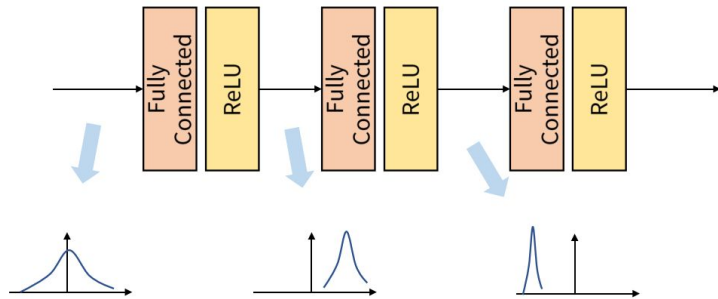
BatchNormalization

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, μ 는 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: $\mathcal{B} = \{x_{1...m}\}$;
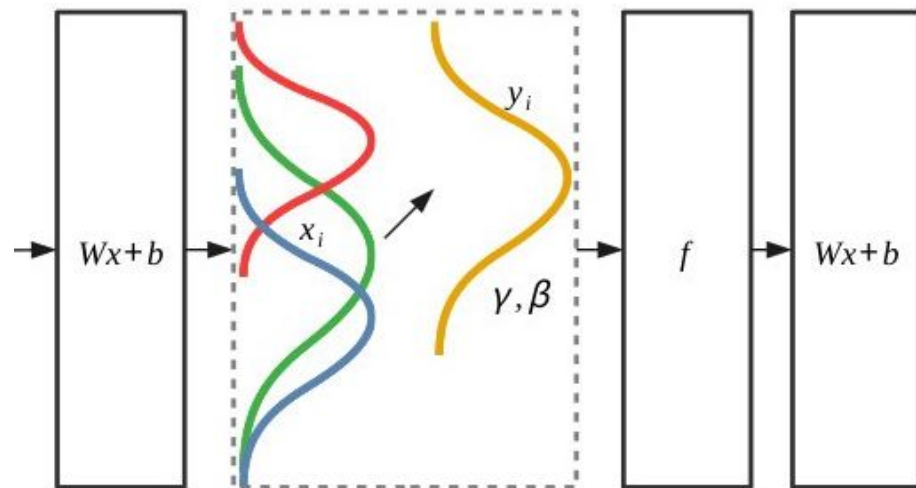Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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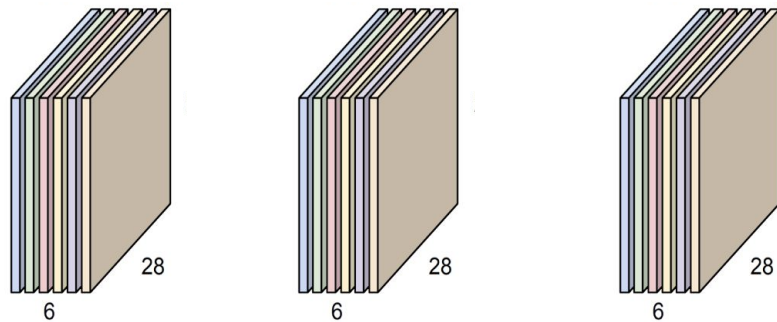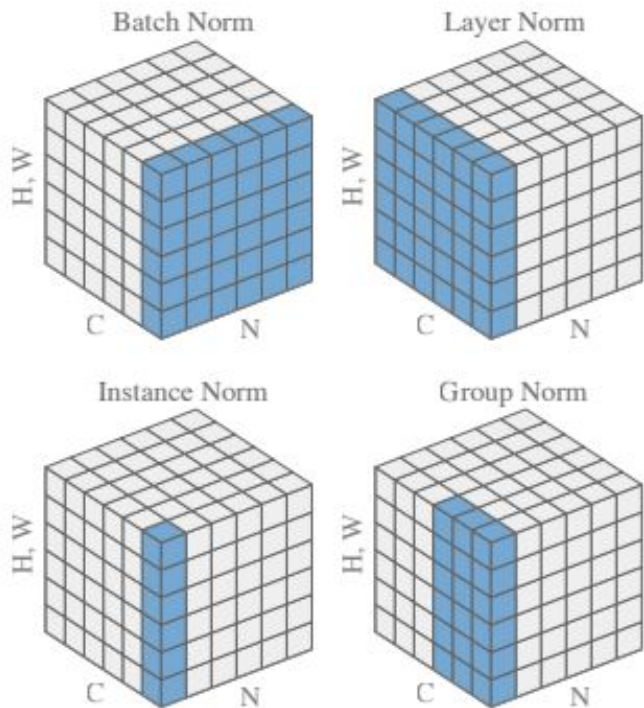
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

그 외의 Normalization들

Batch Norm   Layer Norm

Instance Norm   Group Norm

(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

Embedding

One-hot encoding VS Embedding

One-hot encoding

	cat	mat	on	sat	the
the =>	0	0	0	0	1
cat =>	1	0	0	0	0
sat =>	0	0	0	1	0

...   ...

A 4-dimensional embedding

cat =>	1.2	-0.1	4.3	3.2
mat =>	0.4	2.5	-0.9	0.5
on =>	2.1	0.3	0.1	0.4

...   ...

Embedding layer
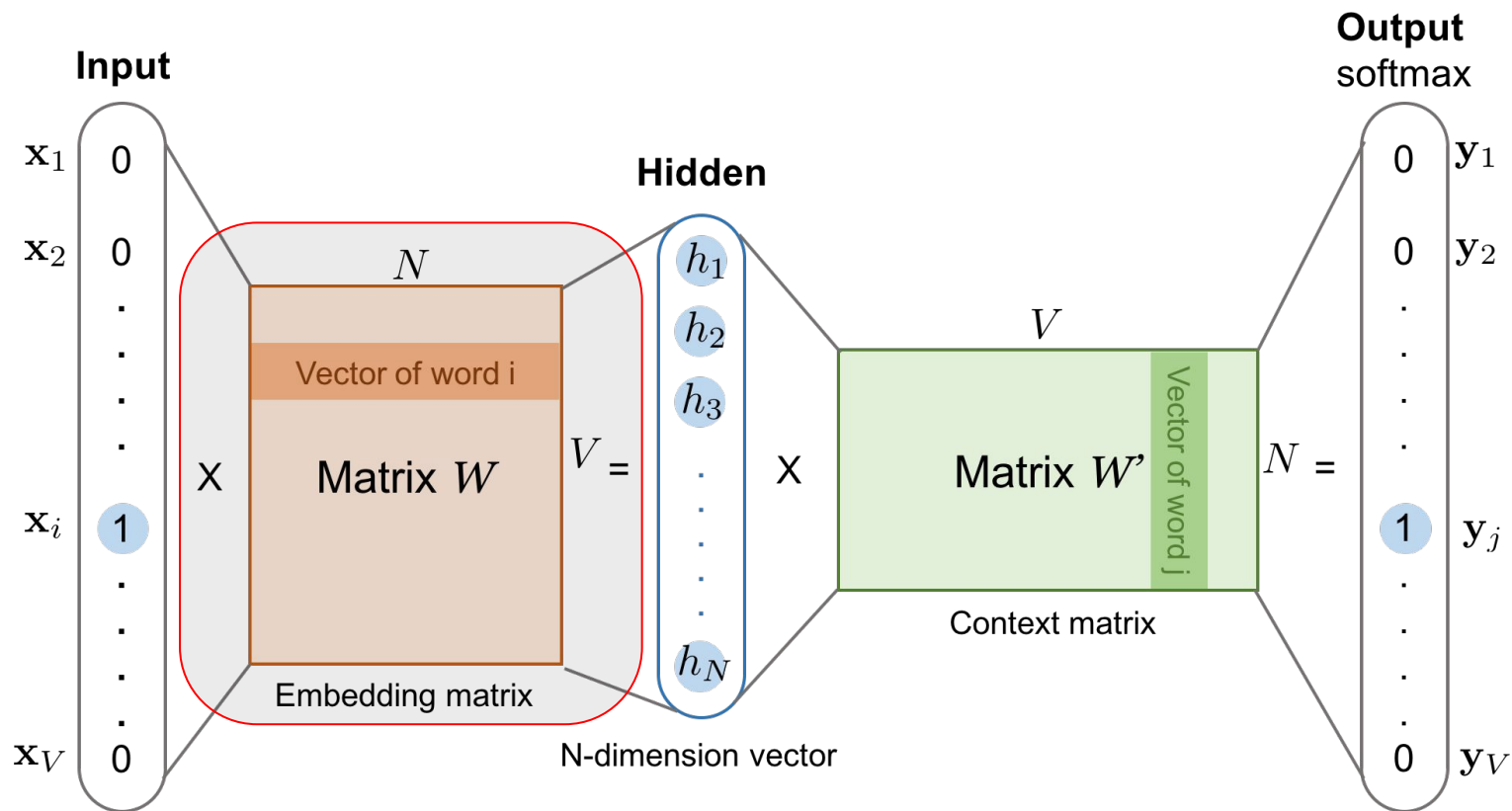
Embedding layer

[837, 789, 24, 12000, 584]

Height
범주의 개수

Embedding Layer!

4.2	1.8	2.3	0.8	3.2
1.2	0.9	9.9	2.5	1.1
...	...	...	...	...
...	...	...	...	...
2.4	8.4	1.1	0.7	2.2

Width:
임베딩시킬 차원수

	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Embedding Example – IMDB

```python
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.imdb.load_data(num_words=10000)
print(x_train.shape, x_test.shape)

x_train = tf.keras.preprocessing.sequence.pad_sequences(x_train, maxlen=32)
x_test = tf.keras.preprocessing.sequence.pad_sequences(x_test, maxlen=32)

print(x_train.shape, x_test.shape)
```

- x_train: imdb 영화 댓글
- y_train: 호감, 비호감 평가
- 각 문장의 길이가 다르기 때문에 최대 길이 32 단어로 자름

Embedding Example – IMDB

```
word_index = tf.keras.datasets.imdb.get_word_index()
inverted_word_index = dict((i, word) for (word, i) in word_index.items())
list(inverted_word_index.items())[:10]

[(34701, 'fawn'),
 (52006, 'tsukino'),
 (52007, 'nunnery'),
 (16816, 'sonja'),
 (63951, 'vani'),
 (1408, 'woods'),
 (16115, 'spiders'),
 (2345, 'hanging'),
 (2289, 'woody'),
 (52008, 'trawling')]


print(" ".join([inverted_word_index[w] for w in x_train[0]]))
```

- x_train는 단어를 word index로 변환해 놓은 데이터
- 다시 영어 문장으로 바꾸어서 출력하는 코드

Embedding Example – IMDB

```python
X = tf.keras.Input(shape=[32])
H = tf.keras.layers.Embedding(10000, 20)(X)
H = tf.keras.layers.SimpleRNN(32)(H)
Y = tf.keras.layers.Dense(1, activation="sigmoid")(H)

model = tf.keras.Model(X, Y)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()


history = model.fit(x_train, y_train, epochs=10, batch_size=32)
```

- Embedding layer를 이용하여 20개의 feature로 embedding
- RNN을 사용

Embedding Example – IMDB

```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['accuracy', 'val_accuracy'])
plt.show()
```