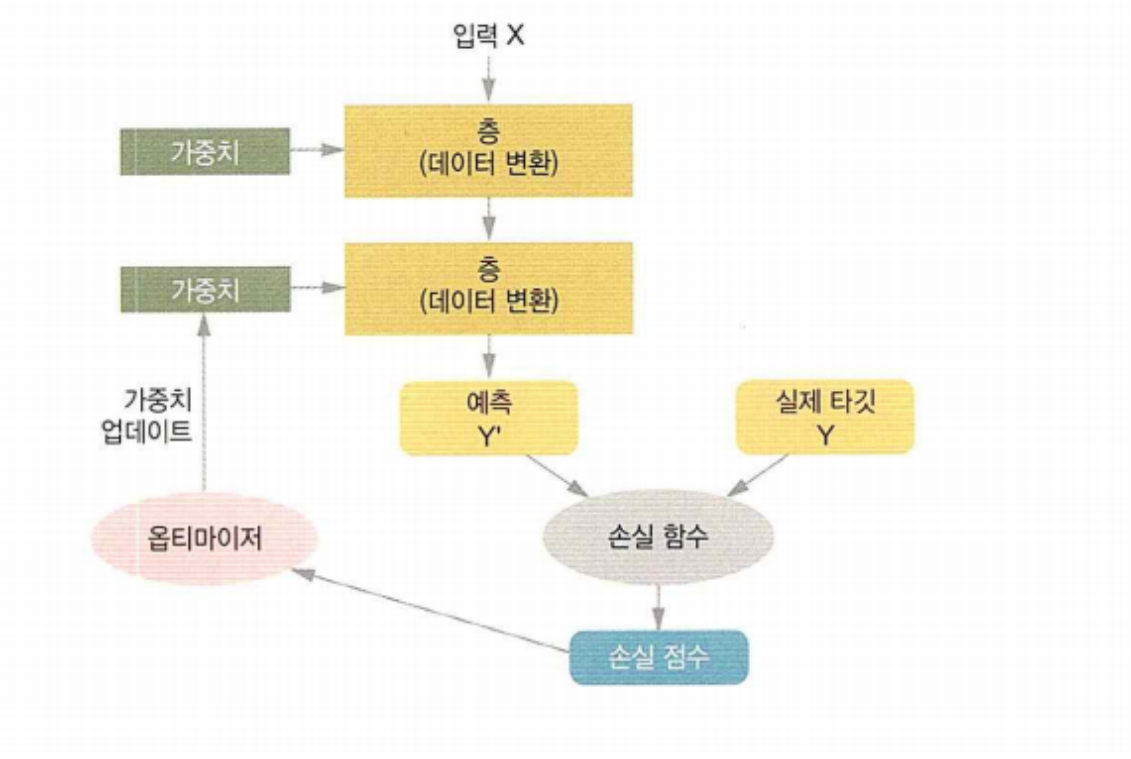
1. neural network

신경망의 구조

신경망 훈련에 관련된 요소들

- 네트워크를 구성하는 층
- 입력 데이터와 그에 상응하는 타겟데이터
- 학습에 사용할 피드백 신호를 정의하는 손실 함수
- 학습 진행 방식을 결정하는 옵티마이저



Layer

Layer 란 무엇인가?

- 하나 이상의 텐서를 입력으로 받아 하나 이상의 텐서를 출력하는 데이터 처리 모듈
- 대부분 **가중치**라는 상태를 가진다.
- 가중치는 확률적 경사 하강법에 의해 학습 되는 하나 이상의 텐서 이며 여기에 네트워크가 학습한 **정보**가 담겨 있다**.**

Layer 종류

- 완전 연결층 (Fully Connected Layer)
- 밀집층 (Dense Layer)
- 순환층 (Recurrent Layer)
- 합성곱층 (Convolution Layer)

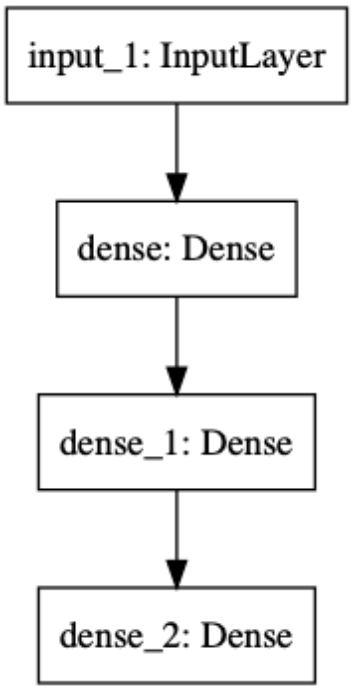
```
from tensorflow.keras import layers

layer = layers.Dense(32, input_shape=(784,)
```

- 첫번째 차원이 784인 2D 텐서만 입력으로 받는 층
- 이 층은 첫 번째 차원 크기가 32로 변환된 텐서를 출력할 것이다.
- 따라서 32차원의 벡터를 입력으로 받는 하위 층이 연결 되어야 한다.
- 하지만 케라스에서는 모델에 추가된 층을 자동으로 상위 층의 크기에 맞추어 주기 때문에 호환성을 걱정하지 않아도 된다.

예제

```
model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,))
model.add(layer.Dense(10))
model.add(layer.Dense(20))
```

Sequential Layer



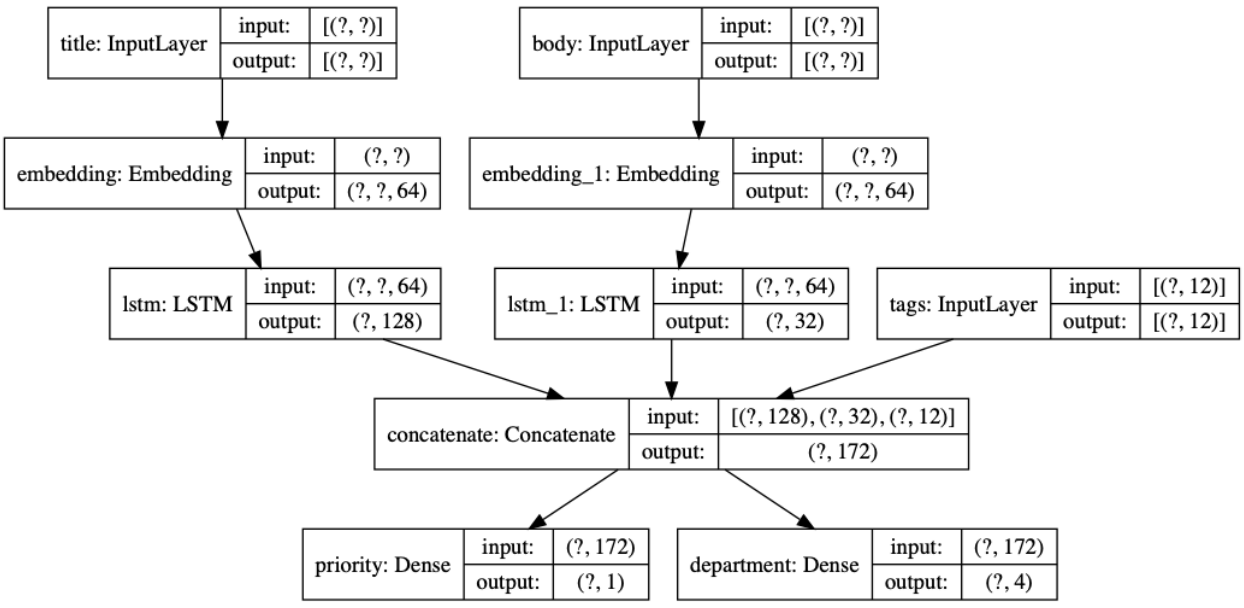
위와 같은 방식으로 Layer를 순차적으로 쌓는 방식을 의미 한다.

```
model = models.Sequential()
model.add(tf.keras.layers.Dense(32, input_shape=(784,)))
model.add(tf.keras.layers.Dense(10))
model.add(tf.keras.layers.Dense(20))
model.summary()
tf.keras.utils.plot_model(model, 'sequential.png')
# 잘안될수도 있음
```

Sequential Layer 가 아닌 모델

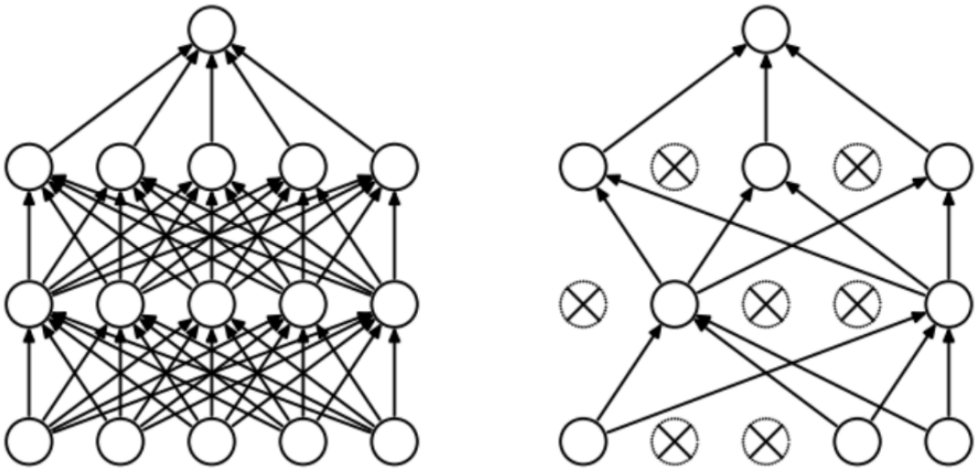
```
num_tags = 12  # Number of unique issue tags
num_words = 10000  # Size of vocabulary obtained when preprocessing text data
num_departments = 4  # Number of departments for predictions

title_input = keras.Input(
```

```
    shape=(None,), name="title"
)  # Variable-length sequence of ints
body_input = keras.Input(shape=(None,), name="body")  # Variable-length sequence of ints
tags_input = keras.Input(
    shape=(num_tags,), name="tags"
)  # Binary vectors of size `num_tags`

# Embed each word in the title into a 64-dimensional vector
title_features = layers.Embedding(num_words, 64)(title_input)
# Embed each word in the text into a 64-dimensional vector
body_features = layers.Embedding(num_words, 64)(body_input)

# Reduce sequence of embedded words in the title into a single 128-dimensional vector
title_features = layers.LSTM(128)(title_features)
# Reduce sequence of embedded words in the body into a single 32-dimensional vector
body_features = layers.LSTM(32)(body_features)

# Merge all available features into a single large vector via concatenation
x = layers.concatenate([title_features, body_features, tags_input])

# Stick a logistic regression for priority prediction on top of the features
priority_pred = layers.Dense(1, name="priority")(x)
# Stick a department classifier on top of the features
department_pred = layers.Dense(num_departments, name="department")(x)

# Instantiate an end-to-end model predicting both priority and department
model = keras.Model(
    inputs=[title_input, body_input, tags_input],
    outputs=[priority_pred, department_pred],
)
```



Dense Layer

- Dense Layer 는 다음과 같이 입력과 출력이 모두 연결된 Layer이다.
-



(a) Standard Neural Net (b) After applying dropout.

손실 함수와 옵티마이저

- loss function (objecttive function) : 훈련하는 동안 최소화 될 값
  - 회귀 타입에 쓰이는 손실함수
    1. MSE : Regression 문제
    2. MAE
    3. MSLE
    4. MAPE
    5. KLD
    6. Poisson
    7. Logcosh
    8. Cosine Similarity
    9. Huber

10. CTC(Connection Temporal Classification) : 시퀀스 학습

- 분류에 쓰이는 손실함수
  1. Binary cross-entropy : 2개의 calss
  2. Categorical cross-entropy : n개의 class
  3. Sparse categorical cross-entropy
  4. Hinge
  5. Squared Hinge
  6. Categorical Hinge

- Optimizer : loss function 을 기반으로 네트워크가 어떻게 업데이트 될지 결정한다.
  - 경사하강법
  - 확률적 경사하강법
  - 뉴턴 방법
  - 등등 경우에 따라 다양한 옵티마이저가 존재한다.

케라스 소개

Keras 의 특징

- 동일한 코드로 CPU와 GPU에서 실행할 수 있다.
- 사용하기 쉬운 API를 가지고 있어 딥러닝 모델의 프로토타입을 빠르게 만들 수 있다.
- 다중입력, 다중 출력 모델, 모델 공유 등 어떤 네트워크 구조도 만들수 있다.

케라스를 사용한 개발 : 빠르게 둘러보기

Keras 작업의 흐름은 다음과 같다

1. 입력 텐서와 타깃 텐서로 이루어진 훈련 데이터를 정의한다
2. 입렵과 타깃을 매핑하는 층으로 이루어진 네트워크(또는 모델)를 정의한다.

3. 손실함수, 옵티마이저, 모니터링 하기 위한 측정 지표를 선택하여 학습 과정을 설정한다.
4. 훈련 데이터에 대한 모델의 fit() 메서드를 반복적으로 호출한다.

Model 정의

- 방법1 : Sequential 클래스를 사용하여 정의

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layer.Dense(32, actuvation = 'relu', input_shape=(784.)))
model.add(layer.Dense(10. activation = 'softmax'))
```

- 방법2 : 함수형 API를 사용하여 정의

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation = 'relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Model compile

모델 구조가 정의된 후에는 Sequential 모델을 사용했는지 함수형 API를 사용했는지에 상관없이 학습의 과정을 설정 해준다.

- 옵티마이저
- 손실함수
- 훈련시 모니터링 지표

```
model.compile(optimizer = optimizers.RMSprop(lr=0.001)
              loss= "mse"
              metrics = ['mae', 'acc'])
```

Model fit

입력데이터와 이에 대응하는 Target Data를 입력하여 모델을 학습시킨다.

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

Sequential/함수형 API 모델 예제

tf.keras 모델의 저장과 복원

가중치 저장/복원

- save_weights() 메서드로 가중치 저장
  - 현재 폴더에 simple_weights.h5 파일을 생성하고 모든 층의 가중치를 저장한다. (HDF5 포맷으로 저장한다.)
  - save_weights 메서드는 기본적으로 텐서플로의 페크포인트 포맷으로 가중치를 저장한다.

```
model.save_weights('simple_weights.h5')
```

- load_weights() 메서드로 가중치 로드
- 새로운 모델을 만들고 load_weight() 메서드를 사용하여 가중치를 로드한다.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=1, input_dim=1))
```

```
model.compile(optimizer='sgd', loss='mse')

model.load_weights('simple_weights.h5')
```

모델 전체 저장

- model.save() 메서드로 모델 전체 저장

```
model.save('simple_model.h5')
```

- tf.keras.models.load_model() 함수를 이용한 모델 로드
- lab2 를 통해 새로운 가중치를 로드

```
model = tf.keras.models.load_model('simple_model.h5')
model.evaluate(x_test, y_test)
```

TensorBoard 사용

- 모델 컴파일시 다음과 같은 내용을 추가한다.

```
model.compile(optimizer='sgd', loss='mse')
history = model.fit(x_train, y_train, epochs=300,
                    validation_split=0.3)
```

```
callback_list = [tf.keras.callbacks.TensorBoard(log_dir='logs')]
model.compile(optimizer='sgd', loss='mse')
history = model.fit(x_train, y_train, epochs=300,
                    callbacks=callback_list, validation_split=0.3)
```

- 학습후 TensorBoard 실행

- googleColab

```
!pip install tensorboardcolab
logdir = '/tmp/log'
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%load_ext tensorboard
%tensorboard --logdir logs
```

- local 실행

```
logdir = './logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
%load_ext tensorboard
%tensorboard --logdir=./logs
```