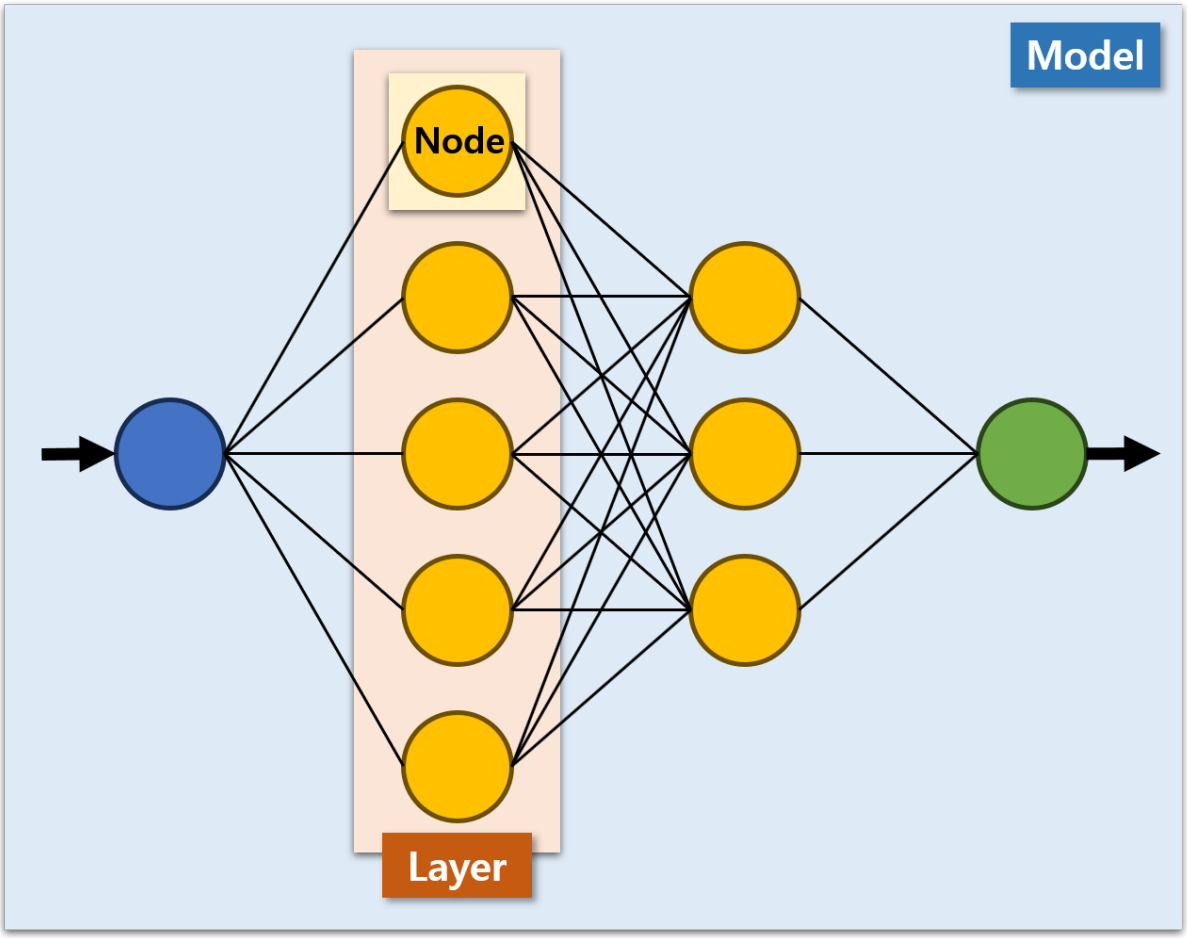
[Keras]

1. 케라스(Keras) 기초부터 모델 생성까지

- 파이썬으로 작성된 오픈 소스 신경망 라이브러리
- CUP와 GPU에서 실행 가능
- 사용자 친화성, 모듈성, 확장성을 통해 빠르고 간편한 프로토타이핑 가능
- 컨볼루션 신경망, 순환 신경망, 그리고 둘의 조합까지 모두 지원
- Tensorflow, CNTK, THEANO에서 사용하는 API



1) 노드

- 생물학의 신경세포(neuron)과 같은 개념
- 신경망을 구성하나 하나의 단위
- 입력데이터를 받아 이를 처리하고 출력 데이터 생성

2) 레이어

- 신경망(Newural Network)와 같은 개념

- 노드의 집합

3) 모델

- 레이어들을 구성하고 데이터가 신경망을 통과하는 흐름을 지정하는 방법

2. 딥러닝 모델

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import plot_model

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

1단계) 데이터 준비

- 학습데이터(Train Data) 준비
- 테스트데이터(Test Data) 준비

In [ ]:
```
'''
# 학습 데이터
train_x = np.arange(1, 51)
train_y = train_x  * 2

# 테스트 데이터
test_x = np.arange(51, 101)
test_y = test_x * 2
'''
```

In [ ]:
```
# 학습 데이터
train_x = np.random.uniform(1, 5, size = 100)
train_y = 10 * train_x + 3

# 테스트 데이터
test_x = np.random.uniform(5, 10, size = 100)
```

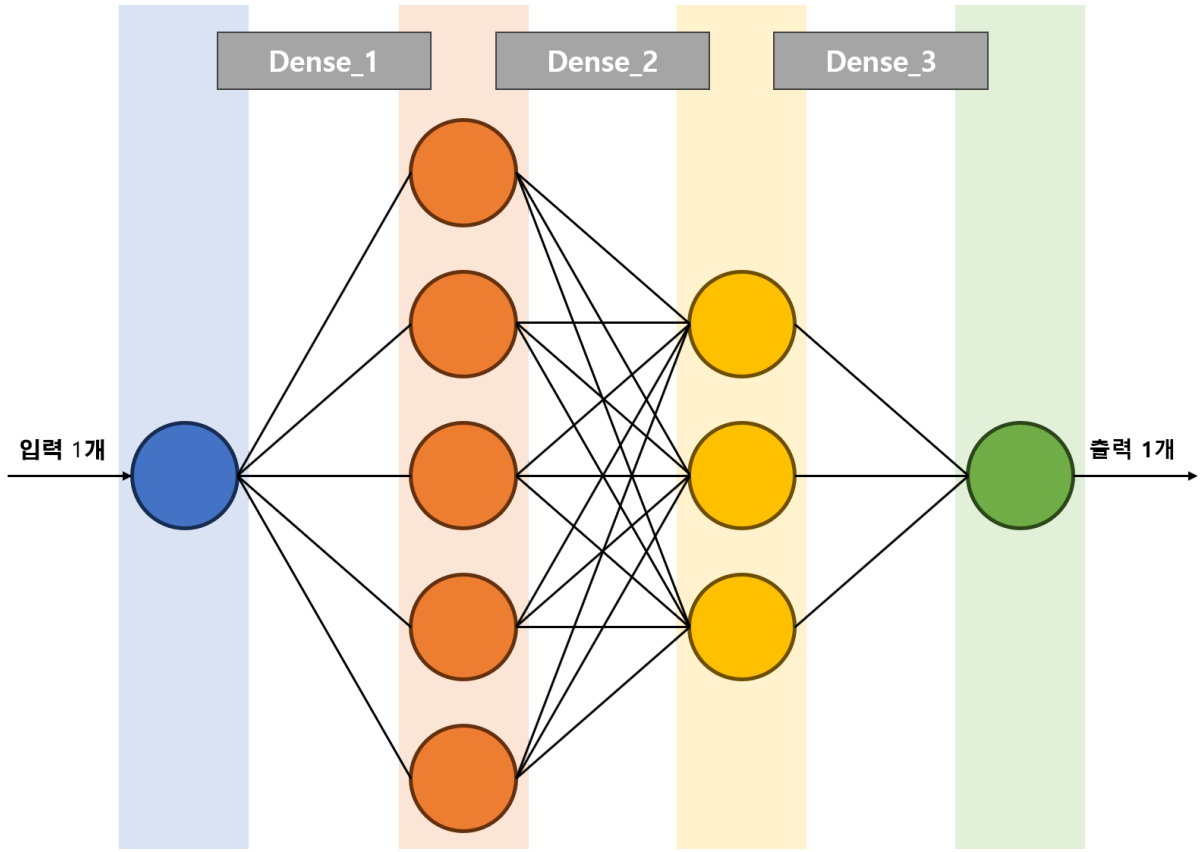
In [ ]:
```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter Plot')

plt.scatter(train_x, train_y, color = 'blue', s = 1)
```

2단계) 모델 구성

Dense

- 첫번째 인자(units) : 출력 뉴런의 수를 설정
- input_dim : 입력 뉴련의 수를 설정
- kernel_initializer : 가중치를 초기화하는 방법을 설정

- uniform : 균일 분포
- normal : 가우시안 분포
- activation : 활성화함수를 설정
- linear : 디폴트 값으로 입력값과 가중치로 계산된 결과 값이 그대로 출력
- sigmoid : 시그모이드 함수로 이진분류에서 출력층에 사용
- softmax : 소프드맥스 함수로 다중클래스 분류문제에서 출력층에 사용
- relu: Rectified Linear Unit 함수로 은닉층에서 사용



In [ ]:
```
# 모델 구성 시작
model = Sequential()

# add()을 이용한 첫 번째 레이어 추가
# 1개의 입력 노드, 5개의 출력 노드, 활성화함수 relu
# relu함수 : 음수일 때 그레이디언트는 0, 양수일때 1
model.add(Dense(5, input_dim = 1, activation = 'relu', name = "Dense_1"))

# add()을 이용한 두 번째 레이어 추가
# 3개의 출력 노드
model.add(Dense(3, name = "Dense_2"))

# add()을 이용한 세 번째 레이어 추가
model.add(Dense(1, name = "Dense_3"))
```

In [ ]:
```
# 모델 확인하기
model.summary()
```

Type *Markdown* and LaTeX: α^2

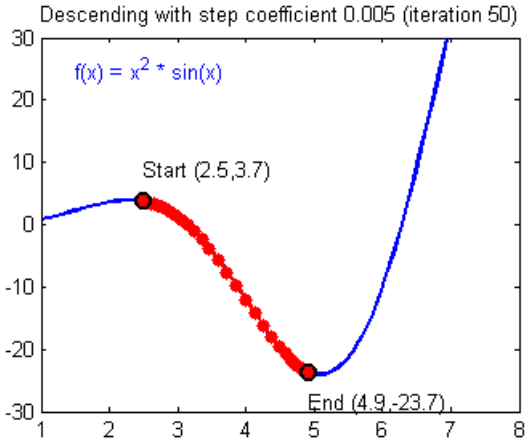
In [ ]:
```
plot_model(model, show_shapes=True, dpi=80)
```

3단계) 모델 준비

- loss : 모델이 예측한 값과 실제 타겟 값 사이의 차이를 측정하는 함수
  - MSE(Means Squared Error, 평균제곱오차)

$$MSE = (1/n) * \Sigma(y_i - \hat{y}_i)^2$$

- optimizer : 손실 함수를 최소화하기 위해 모델의 파라미터를 업데이트 하는 알고리즘
  - 경사 하강법(Gradient Descent)



(출처 : 공돌이의 수학정리노트)

  - 아담(Adam)
  - RMSprop
- Metrics : 모델의 성능을 측정하기 위해 사용되는 지표

In [ ]:
```
model.compile(loss='mse', optimizer='adam', metrics = ['acc'])
```

4단계) 모델 학습

- fit(x, y)함수
  - epochs : 전체 데이터셋을 한 번 훈련하는 단위
  - batch_size : 한 번에 모델에 입력되는 데이터의 개수, 모델이 한번에 처리하는 데이터의 양

In [ ]:
```
model.fit(train_x, train_y, epochs = 100, batch_size = 1)
```

5단계) 모델 평가

- evaluate() : 모델 성능 평가를 위해 테스트 데이터 셋을 사용하여 손실값(loss)과 지정된 메트릭(metric) 값을 계산
- loss : 손실값, 모델의 예측 값과 실제 타겟 값과 차이
- acc : epoch에서 계산된 정확도 값, 0에 가까울 수록 정확도가 높음

In [ ]:
```
model.evaluate(train_x, train_y, batch_size = 1)
```

6단계) 예측

- test_x의 값을 넣었을 때, 예측값 확인

In [ ]:
```
pred_y = model.predict(test_x)
print(pred_y)
```

In [ ]:
```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter Plot')

plt.scatter(train_x, train_y, color = 'blue', s = 1)
plt.scatter(test_x, pred_y, color = 'red', s = 1)
plt.show()
```

4. 딥러닝 모델 예제 - iris

참고 : https://pinkwink.kr/1128 (https://pinkwink.kr/1128)

In [ ]:
```
import seaborn as sns
import pandas as pd
import numpy as np

sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species", palette="husl")
```

1단계) 데이터 *준비*

In [ ]:
```
# 데이터 전처리

from sklearn.preprocessing import LabelEncoder

X = iris.iloc[:,0:4].values
y = iris.iloc[:,4].values

X, y
```

In [ ]:
```
encoder =  LabelEncoder()
# setosa : 0, versicolor : 1, virginica :2
y1 = encoder.fit_transform(y)
# 원핫인코딩 방식 변환
Y = pd.get_dummies(y1).values
Y
```

In [ ]:
```
# 학습데이터 (80%), 훈련데이터 (20%) 분류

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state =1
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

2단계) 모델 구성

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

model = Sequential()

# softmax : 소프드맥스 함수로 다중클래스 분류문제에서 출력층에 사용
# relu: Rectified Linear Unit 함수로 은닉층에서 사용

model.add(Dense(64,input_shape=(4,),activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(3,activation='softmax'))
```

3단계) 모델 준비

In [ ]:
```
model.compile(loss='categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])

model.summary()
```

4단계) 모델 학습

In [ ]:
```
# validation_data : 검증데이터

model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100)
```

5단계) 모델 평가

- loss (손실) : Loss는 모델의 예측과 실제 값 사이의 차이를 측정하는 지표
- accuracy (정확도) : 0과 1 사이의 값으로 표현되며, 1에 가까울수록 더 좋은 성능
  - Accuracy = (올바르게 분류된 샘플 수) / (전체 샘플 수)

In [ ]:
```
# 손실, 정확도 계산

loss, accuracy = model.evaluate(X_test, y_test)

loss, accuracy
```

6단계) 예측

In [ ]:
```
y_pred = model.predict(X_test)
y_test_class = np.argmax(y_test,axis=1)
y_pred_class = np.argmax(y_pred,axis=1)

result = pd.DataFrame(X_test, columns = iris.columns[:-1])
result['species'] = y_test_class
result['pred species'] = y_pred_class

result
```