Introduction to DL Framework

정연석

0. Intro

1) 실습 강의 <<< Introduction

2) 강의 진행 방식

- Introduction to DL Framework → Pytorch Tutorial → (Tensorflow)

3) Case 별 시청 방식

- Framework 사용 경험이 적은 분
- Framework 사용에 숙련된 분

4) 질문과 답변 자유롭게

0. Intro 5) 오늘 강의의 핵심

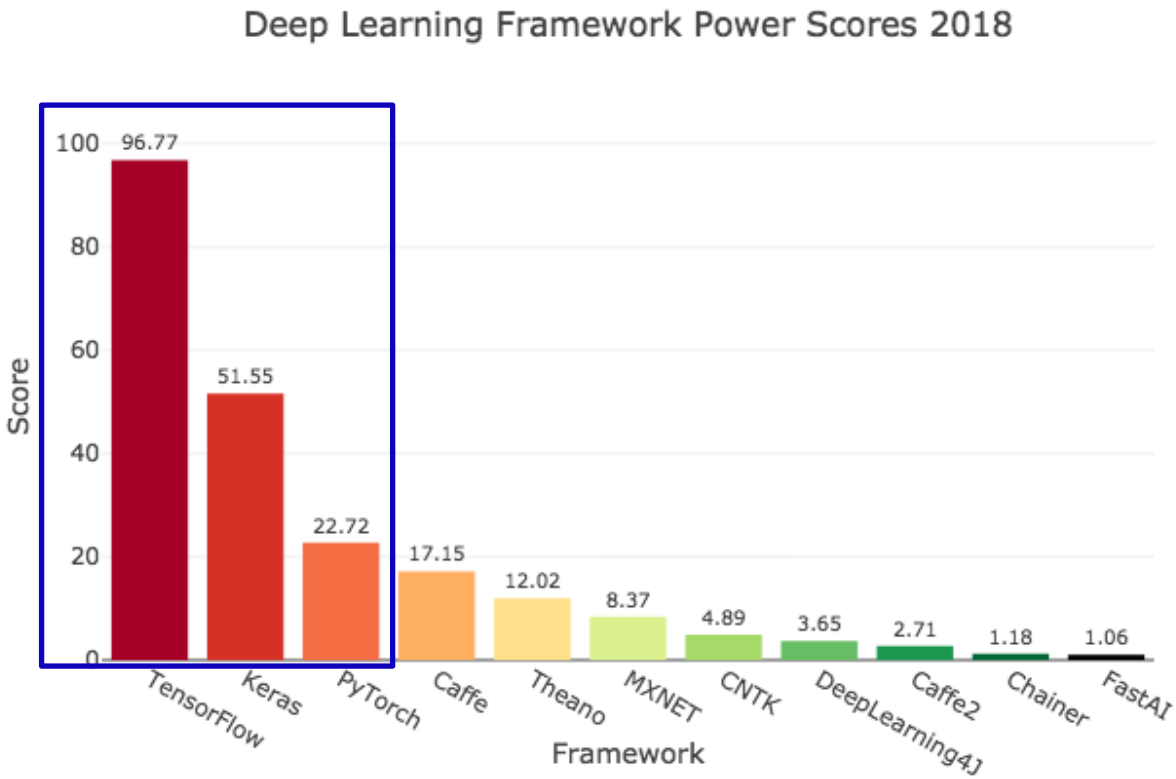
1> 각 Framework는 서로를 닮아가고 있습니다.

2> 본인의 나아가고자 하는 분야에서 많이 사용하는 Framework를 선택

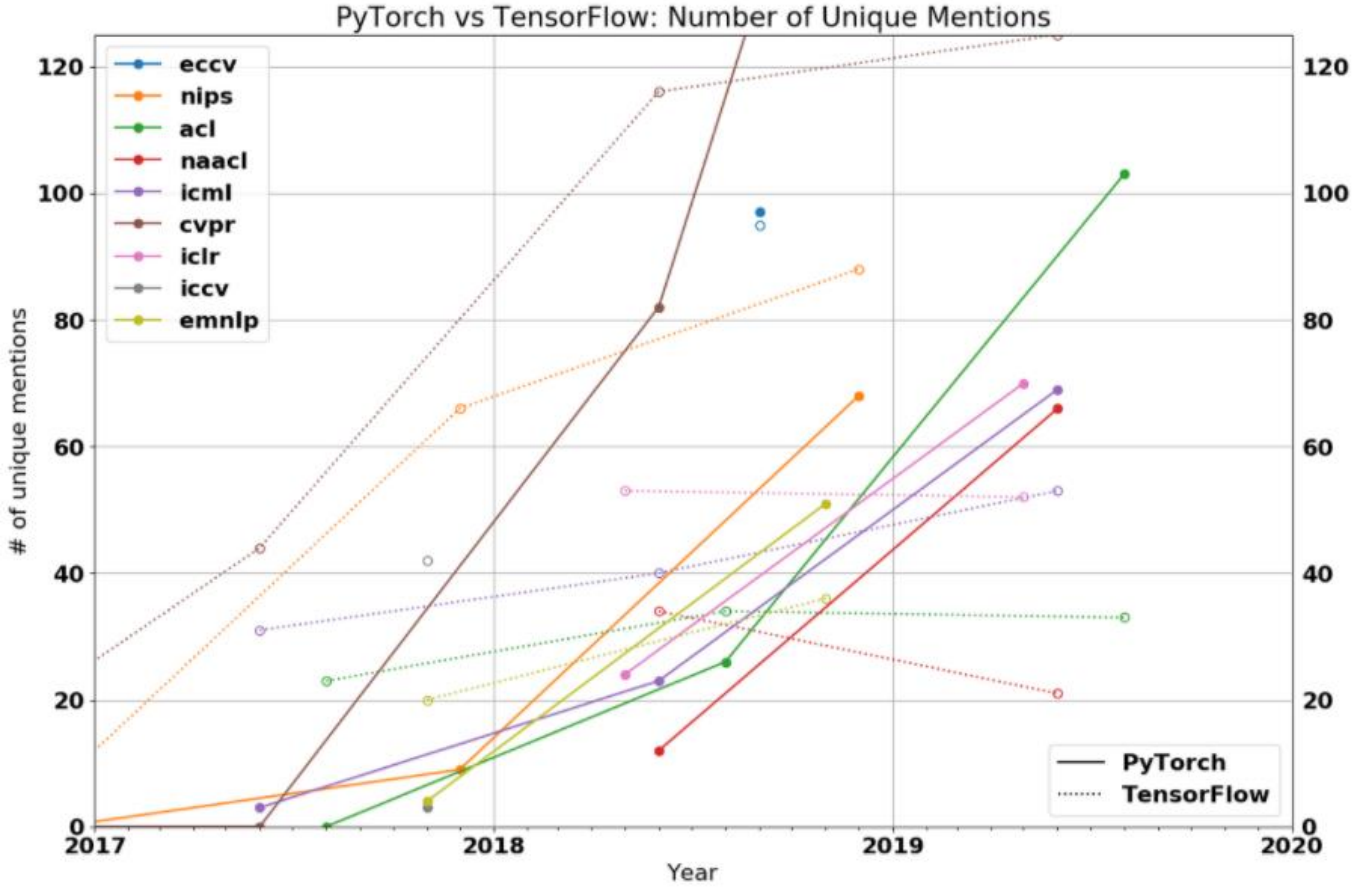
3> 주요 Framework는 다 배워보면 좋습니다.

4> (강의와 약간 별개로) 여러 분야에 도전해보는 것도 추천합니다.

0. Intro 6) Framework들에 대한 변천사



0. Intro 6) Framework들에 대한 변천사



1. Keras 0) 모델을 build하는 방법

3가지

Kaggle, Github 등에 다양한 코드가 있고

우리는 여러 코드를 다 이해할 수 있어야 하기 때문에

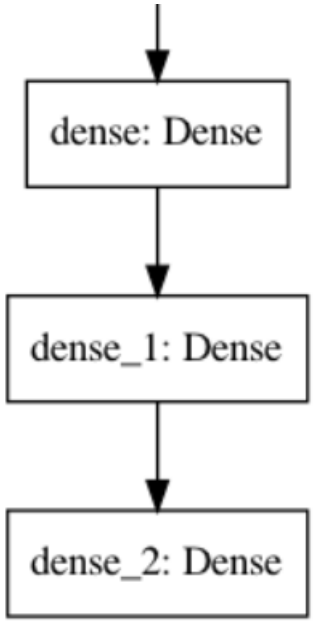
모든 방식에 대해 알아야 합니다.

1. Keras 1) Sequential

1> Sequential : 각 layer를 순차적으로 담을 수 있는 Class

('Stack' in official guide)

```
model = keras.Sequential(
    [
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(10)
    ]    )
```



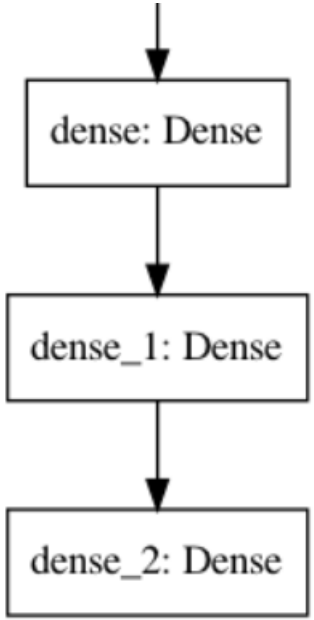
https://keras.io/guides/sequential_model/

1. Keras 1) Sequential

2> add라는 메서드를 이용하면 더 깔끔하고 직관적으로 모델을 build할 수 있다.

```
model = keras.Sequential()

model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(10, actication="softmax"))
```



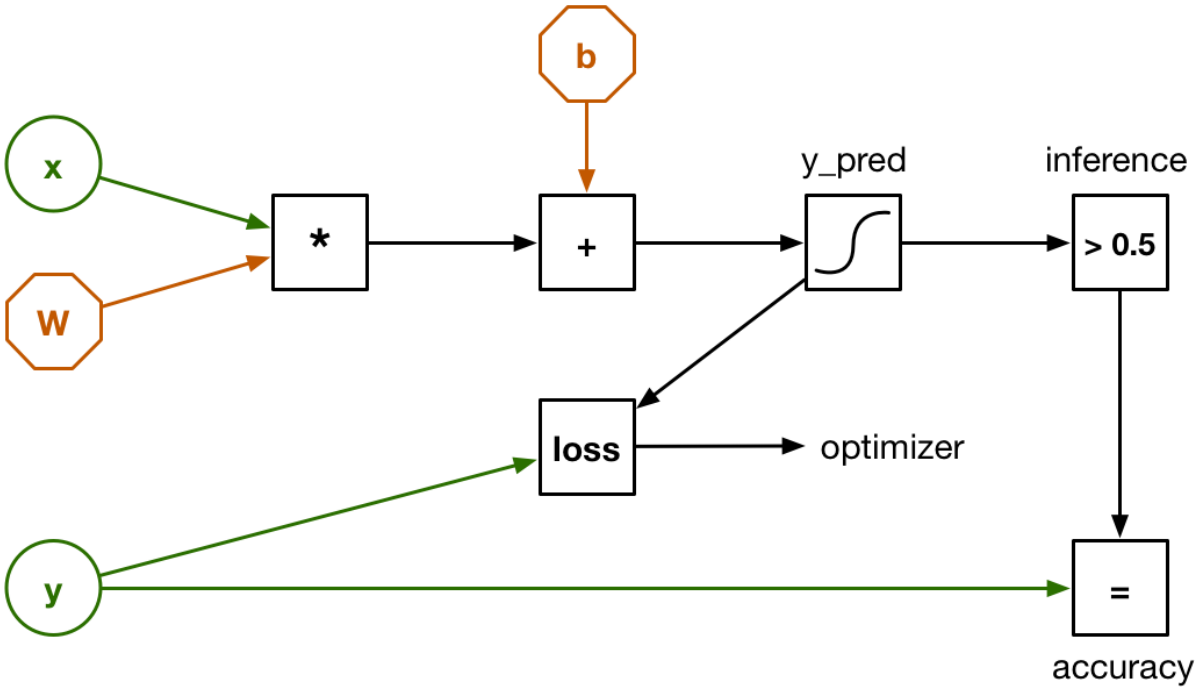
https://keras.io/guides/sequential_model/

순차적으로 담는 것이 쉽고 직관적이어서 좋은데

어떤 단점이 있을까요?

1. Keras 2) Functional API

1> Deep Neural Network는 거대한 합성 함수이다.



https://keras.io/guides/functional_api/

1. Keras 2) Functional API

1> Deep Neural Network는 거대한 합성 함수이다.

```
model = keras.Sequential(
    [
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(10)
    ]    )
```

```
layer1 = layers.Dense(64, activation="relu")
layer2 = layers.Dense(64, activation="relu")
layer3 = layers.Dense(10, actication="softmax")

# Call layers on a test input
x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

그래서 위 두 코드는 같은 모델입니다.

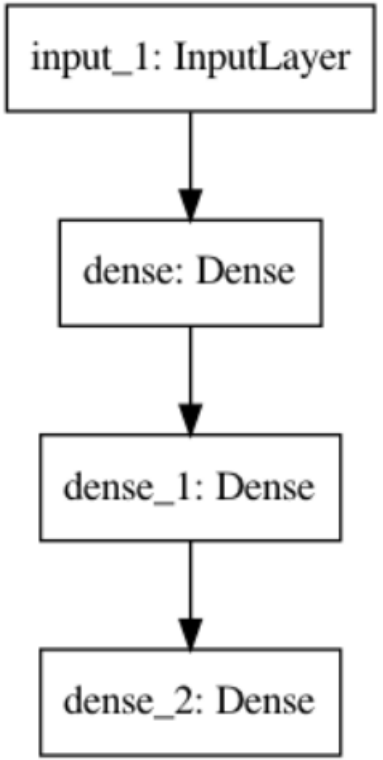
https://keras.io/guides/functional_api/

1. Keras 2) Functional API

2> Callable한 class instance를 사용

```
inputs = keras.Input(shape=(784,))
A1 = layers.Dense(64, activation="relu")(inputs)
A2 = layers.Dense(64, activation="relu")(A1)
outputs = layers.Dense(10, activation="softmax")(A2)

model = keras.Model(inputs=inputs, outputs=outputs)
```



Callable을 모르면 보세요 :
https://frhyme.github.io/others/callable_object/

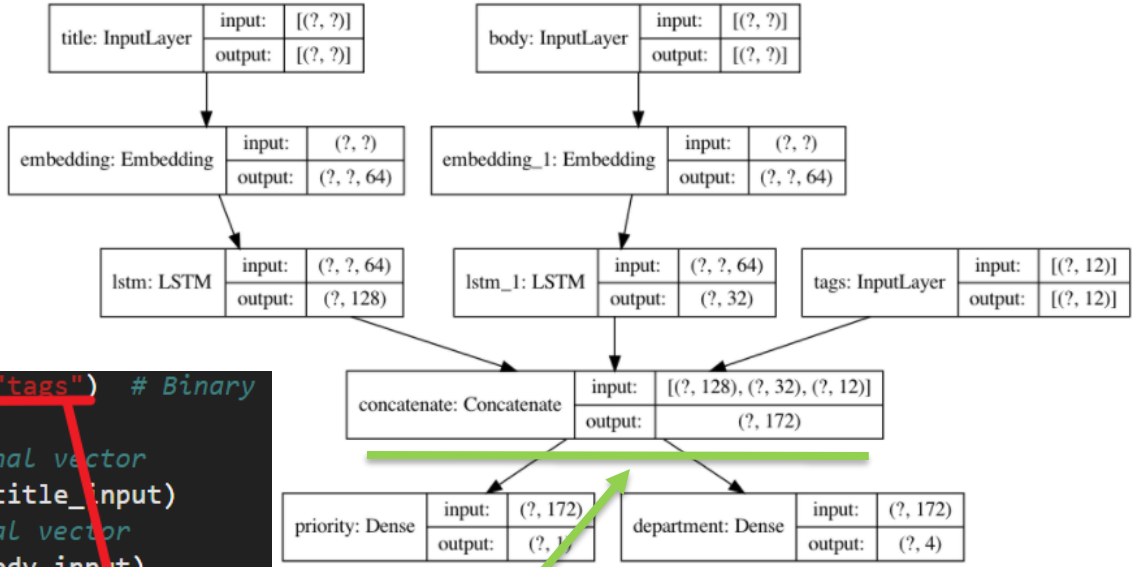
https://keras.io/guides/functional_api/

1. Keras 2) Functional API

3> 여러 input과 output을 처리할 수 있다.

(Sequential의 한계 극복)

https://keras.io/guides/functional_api/




```
tags_input = keras.Input(shape=(num_tags,), name="tags")  # Binary

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

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

# Stick a logistic regression for priority prediction on top of the
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)
```

1. Keras 3) Model Class

1> 모델을 OOP 방식으로 관리할 수 있다.

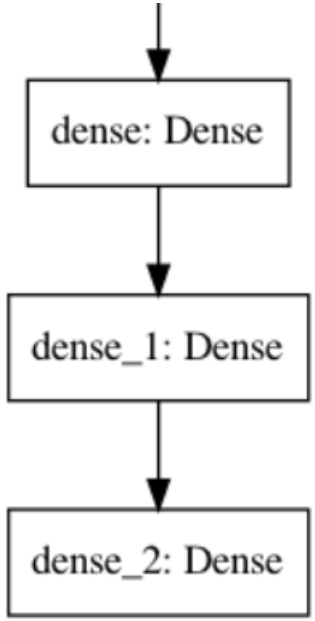
2> tf.keras.Model을 기본적으로 상속받아서 구현

```
class MyModel(tf.keras.Model):

  def __init__(self):
    super(MyModel, self).__init__()
    self.dense1 = layers.Dense(64, activation=tf.nn.relu)
    self.dense2 = layers.Dense(64, activation=tf.nn.relu)
    self.dense2 = layers.Dense(10, activation=tf.nn.softmax)

  def call(self, inputs):
    A1 = self.dense1(inputs)
    A2 = self.dense2(A1)
    return self.dense3(A2)

model = MyModel()
```



https://keras.io/api/models/model/

초보자를 위한 빠른 시작

https://www.tensorflow.org/tutorials/quickstart/beginner

전문가를 위한 빠른 시작

https://www.tensorflow.org/tutorials/quickstart/advanced

1. Keras

- Keras는 쉽습니다.
- Keras도 거의 모든 model을 구현할 수 있습니다.
- 이제는 Tensorflow에 흡수되어서 Keras와 구분 지을 필요가 없습니다.

2. Tensorflow

1) 2.0 변화의 방향성

1> 쉬워지려고 합니다. – keras를 내부 패키지로

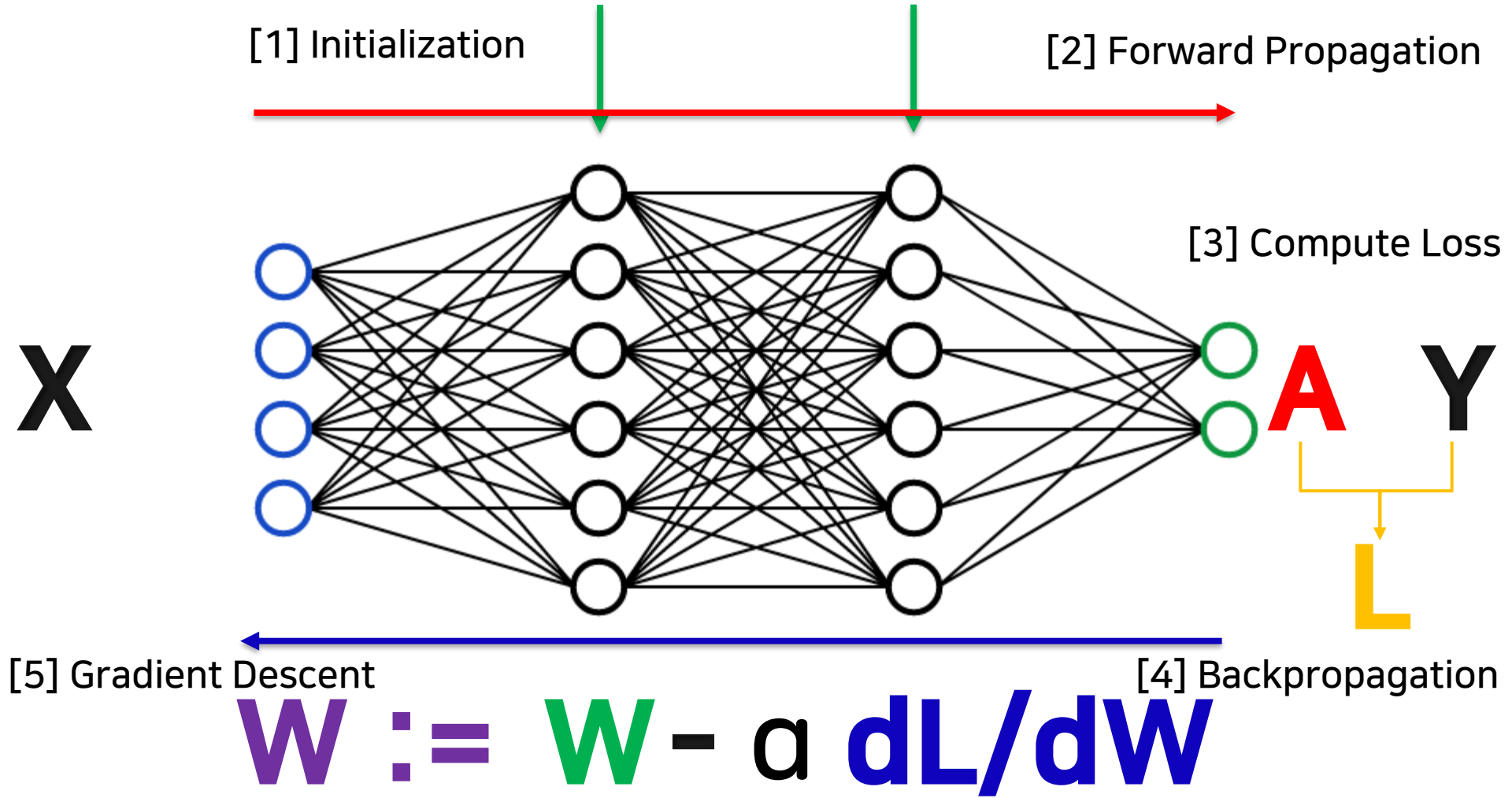
2> pytorch를 닮아갑니다.

2) Deploy

1> Tensorflow Lite, Tensorflow.js 등

2. Tensorflow 3) Custom Train

간단히 배우기 전에 Neural Network 복습


[1] Initialization
[2] Forward Propagation
[3] Compute Loss
X
A
Y
L
[5] Gradient Descent
[4] Backpropagation
W := W - α dL/dW

2. Tensorflow 3) Custom Train

```
def train(model, inputs, outputs, learning_rate):
  with tf.GradientTape() as t:
    current_loss = loss(model(inputs), outputs)
  dW, db = t.gradient(current_loss, [model.W, model.b])
  model.W.assign_sub(learning_rate * dW)
  model.b.assign_sub(learning_rate * db)
```

t에 gradient를 기록

[2] Forward Propagation

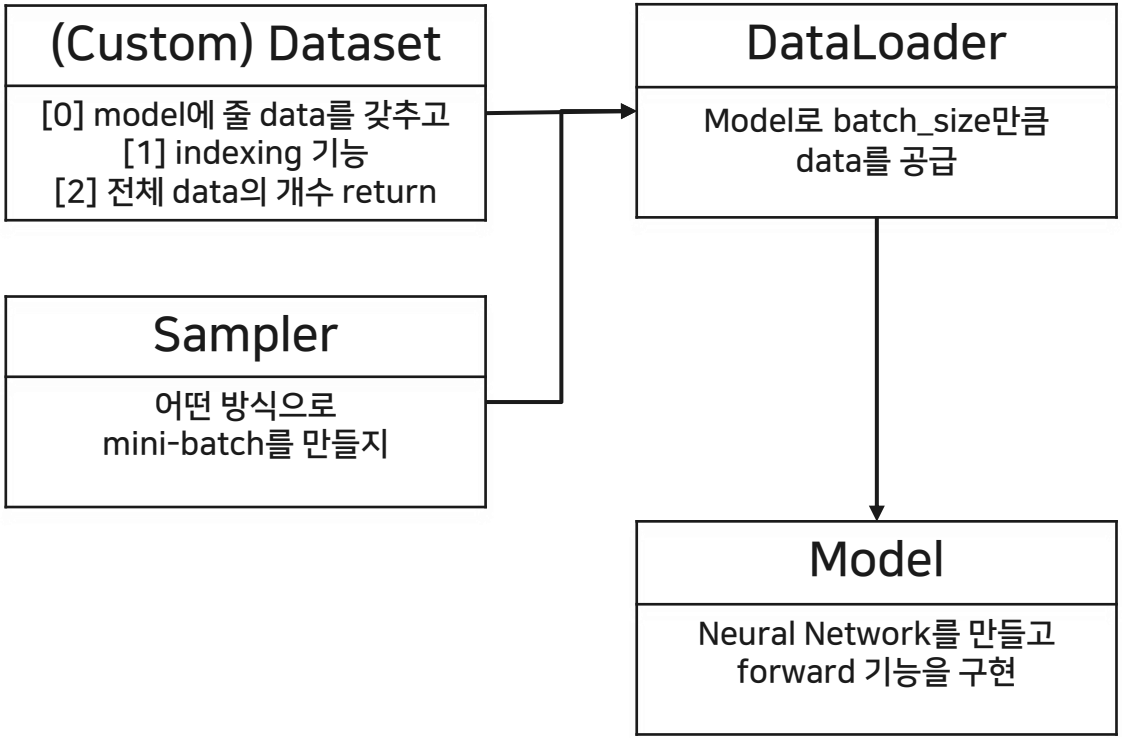
[3] Compute Loss

[4] BackPropagation

[5] Gradient Descent

https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough

3. Pytorch


(Custom) Dataset
[0] model에 줄 data를 갖추고
[1] indexing 기능
[2] 전체 data의 개수 return
Sampler
어떤 방식으로
mini-batch를 만들지
DataLoader
Model로 batch_size만큼
data를 공급
Model
Neural Network를 만들고
forward 기능을 구현

3. Pytorch

Pytorch Tutorial.ipynb 파일에서

발표 진행하겠습니다.

PyTorch

```
# model
class Net(nn.Module):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
   model.train()
   for batch_idx, (data, target) in enumerate(train_loader):
      data, target = data.to(device), target.to(device)
      optimizer.zero_grad()
      output = model(data)
      loss = F.nll_loss(output, target)

      loss.backward()
      optimizer.step()
      if batch_idx % args.log_interval == 0:
          print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

https://pytorch-lightning.readthedocs.io/en/latest/

https://github.com/PyTorchLightning/pytorch-lightning

Setup your trainer

```python
# Setup your trainer
import torch

import ignite.distributed as idist
from ignite.engine import Engine, Events
from ignite.contrib.handlers import ProgressBar


def create_trainer(model, optimizer, criterion, lr_scheduler, config):

    # Define any training logic for iteration update
    def train_step(engine, batch):
        x, y = batch[0].to(idist.device()), batch[1].to(idist.device())

        model.train()
        y_pred = model(x)
        loss = criterion(y_pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()

        return loss.item()

    # Define trainer engine
    trainer = Engine(train_step)

    if idist.get_rank() == 0:
        # Add any custom handlers
        @trainer.on(Events.ITERATION_COMPLETED(every=200))
        def save_checkpoint():
            fp = Path(config.get("output_path", "output")) / "checkpoint.pt"
            torch.save(model.state_dict(), fp)

        # Add progress bar showing batch loss value
        ProgressBar().attach(trainer, output_transform=lambda x: {"batch loss": x})

    return trainer
```

https://pytorch.org/ignite/