ML·DL Intermediate Course-120

PyTorch
Fully Connected Networks
(Multi Layer Perceptron)

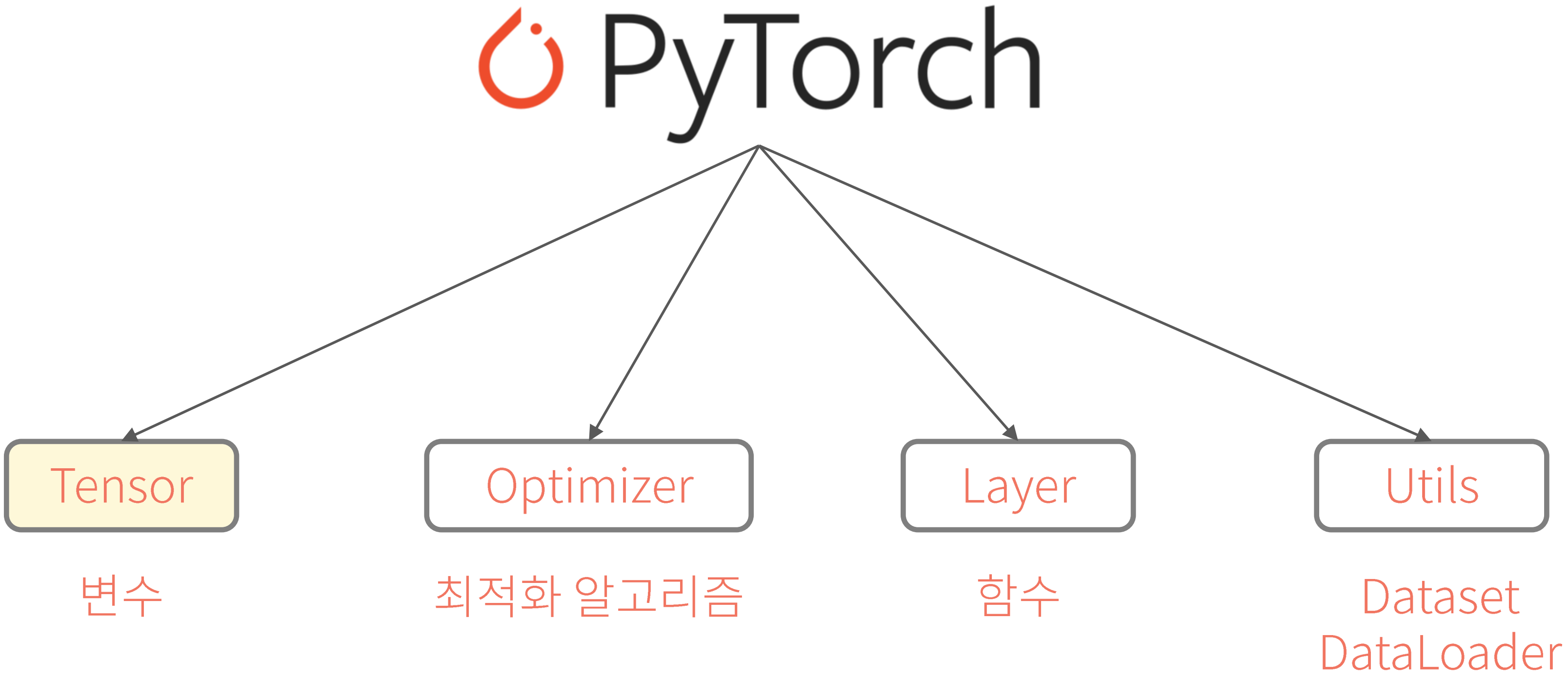
조준우
metamath@gmail.com

파이토치|PyTorch

PyTorch

- PyTorch is a Python-based scientific computing package serving two broad purposes:
  - A replacement for NumPy to use the power of GPUs and other accelerators.
  - An automatic differentiation library that is useful to implement neural networks

파이토치|PyTorch



PyTorch tensor

- Tensor
  - Multidimensional array
  - 물리학에서 물리량을 표현하기 위한 수학 도구
  - 인공지능 분야에서는 다차원 배열
  - 넘파이 어레이와 유사 ↔ 파이토치 텐서

자동 미분 가능한 tensor

```
import numpy as np
import torch

np.random.seed(0)                  #❶

x = np.random.rand(6).reshape(2,3) #❷

x_tensor = torch.tensor(x)         #❸
x_from_numpy = torch.from_numpy(x)
x_Tensor = torch.Tensor(x)
x_as_tensor = torch.as_tensor(x)

print(x, x.dtype)                  #❹
#>>> [[0.5488 0.7152 0.6028]
      [0.5449 0.4237 0.6459]] float64

print(x_tensor, x_tensor.dtype, x_tensor.requires_grad)
#>>> tensor([[0.5488, 0.7152, 0.6028],
             [0.5449, 0.4237, 0.6459]], dtype=torch.float64) torch.float64 False

print(x_from_numpy, x_from_numpy.dtype, x_from_numpy.requires_grad)
#>>> tensor([[0.5488, 0.7152, 0.6028],
             [0.5449, 0.4237, 0.6459]], dtype=torch.float64) torch.float64 False

print(x_Tensor, x_Tensor.dtype, x_Tensor.requires_grad)
#>>> tensor([[0.5488, 0.7152, 0.6028],
             [0.5449, 0.4237, 0.6459]]) torch.float32 False

print(x_as_tensor, x_as_tensor.dtype, x_as_tensor.requires_grad)
#>>> tensor([[0.5488, 0.7152, 0.6028],
             [0.5449, 0.4237, 0.6459]], dtype=torch.float64) torch.float64 False
```

```
x_tensor_grad = torch.tensor(x, requires_grad=True)

print(x_tensor_grad, x_tensor_grad.dtype, x_tensor_grad.requires_grad)

#>>> tensor([[100.0000,   0.7152,   0.6028],
             [  0.5449,   0.4237,   0.6459]], dtype=torch.float64,
        requires_grad=True) torch.float64 True
```

미분계수를 가질 수 있는 변수

tensor API

- detach()
- view() and reshape()
- item()
- squeeze()
- cat()
- stack()

tensor API

- detach()
  - Returns a new Tensor, detached from the current graph.
- view() and reshape()
- item()
- squeeze()
- cat()
- stack()

tensor API

- detach()
- view() and reshape()
  - 대부분 동일하나 약간 차이 있음
- item()
- squeeze()
- cat()
- stack()

tensor API

- detach()
- view() and reshape()
- item()
  - Returns the value of this tensor as a standard Python number. This only works for tensors with one element.
- squeeze()
- cat()
- stack()

tensor API

- detach()
- view() and reshape()
- item()
- squeeze()
  - Returns a tensor with all the dimensions of input of size 1 removed.
- cat()
- stack()

tensor API

- detach()
- view() and reshape()
- item()
- squeeze()
- cat()
  - Concatenates the given sequence of 'seq' tensors in the given dimension. All tensors must either have the same shape (except in the concatenating dimension) or be empty.
- stack()

tensor API

- detach()
- view() and reshape()
- item()
- squeeze()
- cat()
- stack()
  - Concatenates a sequence of tensors along a new dimension. All tensors need to be of the same size.

tensor 저장하기

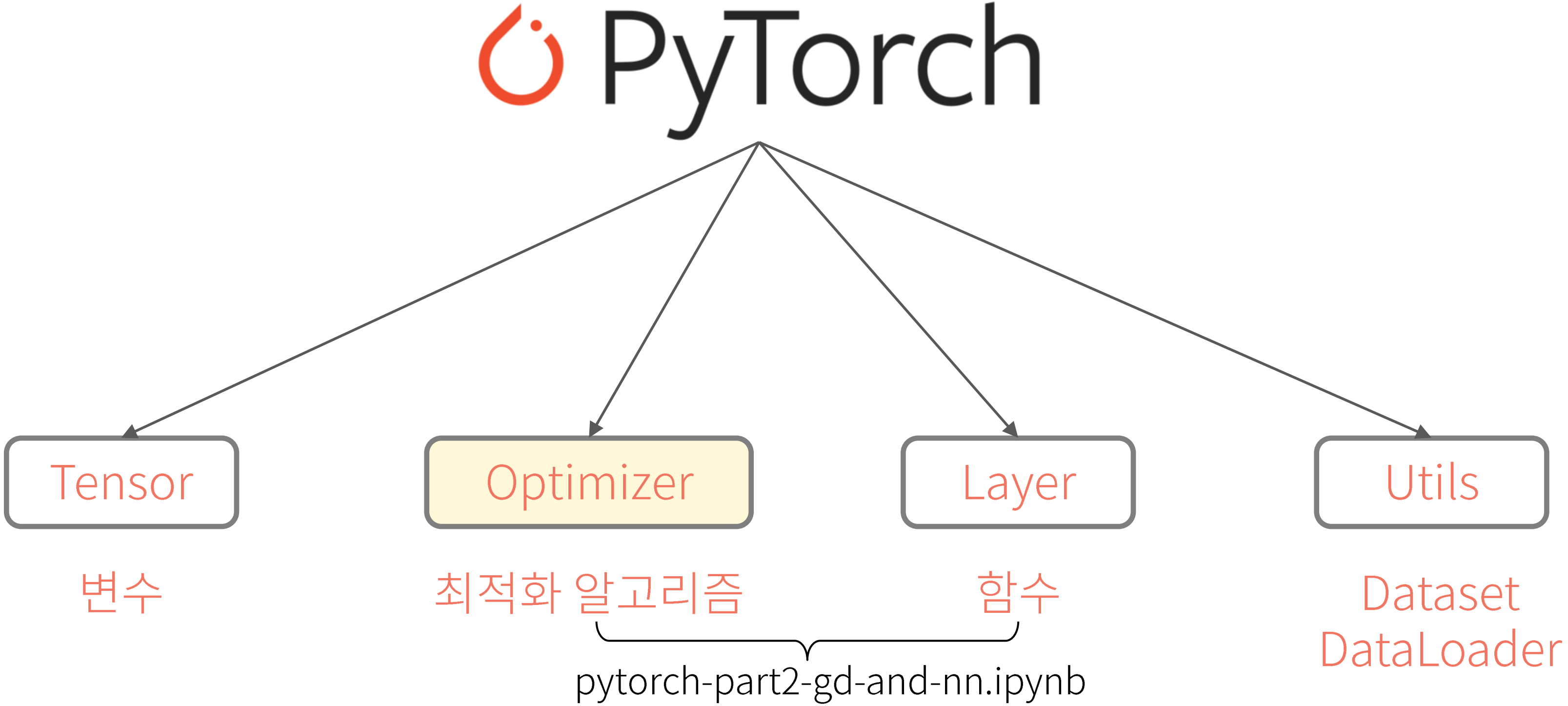
- save

```python
# 파일로 저장하기
foo = torch.arange(24).reshape(2,3,4)

with open('foo.t', 'wb') as f:
    torch.save(foo, f)
```

- load

```python
# 저장한 파일에서 읽어오기
bar = torch.load('foo.t')

foo-bar
```

파이토치|PyTorch



Gradient Descent with PyTorch

- Steepest Decent from Scratch

```
def SDM(f, df, x, eps=1.0e-6, \
        max_iter=1000, callback=None):

    for k in range(max_iter):
        c = df(x)


        if np.linalg.norm(c) < eps :
            print("Stop criterion break Iter.:\
               {:5d}, x: {}".format(k, x))
            break

        d = -c

        alpha = gss(f, df, x, d, delta=1.0e-3)[0]

        x = x + alpha * d
    else:
        print("Stop max iter:{:5d} x:{}".format(k, x))
```

이 코드를 pytorch를 이용하는 스타일로 바꾸기

Gradient Descent with PyTorch

- Steepest Decent torch version

```python
def SDM(f, df, x, eps=1.0e-6, \
        max_iter=1000, callback=None):

    for k in range(max_iter):
        c = df(x)


        if np.linalg.norm(c) < eps :
            print("Stop criterion break Iter.:\
               {:5d}, x: {}".format(k, x))
            break

        d = -c

        alpha = gss(f, df, x, d, delta=1.0e-3)[0]

        x = x + alpha * d
    else:
        print("Stop max iter:{:5d} x:{}".format(k, x))
```

```python
def SDM_torch(f, x, eps=1.0e-6, \
        max_iter=1000, callback=None):

    for k in range(max_iter):
        pass # forward!!!
        pass # backward!!!

        if np.linalg.norm(c) < eps :
            print("Stop criterion break Iter.:\
               {:5d}, x: {}".format(k, x))
            break

        d = -c

        alpha = gss(f, None, x, d, delta=1.0e-3)[0]

        pass # in-place update!!!
    else:
        print("Stop max iter:{:5d} x:{}".format(k, x))
```

Optimizer: torch.optim

- torch.optim is a package implementing various optimization algorithms.
- Most commonly used methods are already supported, and the interface is general enough, so that more sophisticated ones can be also easily integrated in the future.
- Algorithms
  - SGD: Implements stochastic gradient descent (optionally with **momentum**).
  - RMSprop: Implements RMSprop algorithm.
  - Adam: Implements Adam algorithm.
  - Adadelta, Adagrad, AdamW, SparseAdam, Adamax, LBFGS, $\cdots$

Optimizer: torch.optim

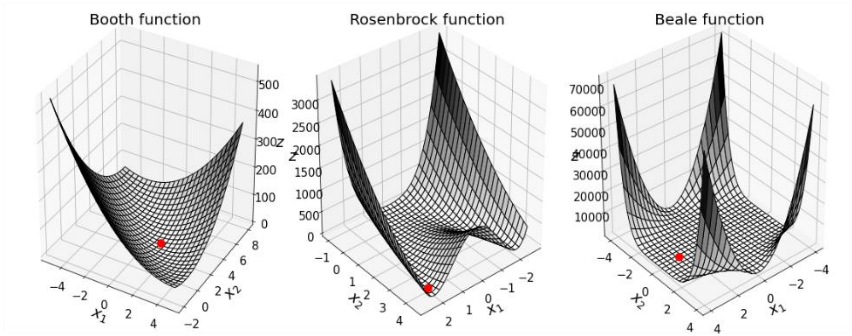
- 동작 방식

```
# var1 and var2으로 어떤 함수를 정의 한다.
# F = …

# 특정 알고리즘에 최적화 시킬 변수를 알려준다.
optimizer = optim.Adam([var1, var2], lr=0.0001)

# optimizer가 가지고 있는 최적화 시킬 변수의 grad를 0으로 초기화 시킨다.
optimizer.zero_grad()

# 변수 var1, var2로 함수값을 계산한다.(forward pass)
loss = F(var1, var2)

loss.backward() # 계산된 함숫값을 역전파한다.(backward pass)

# backward pass로 얻은 grad를 이용해서 변수를 업데이트 한다.
# 이 때 업데이트 방법은 optimizer에 따라 내부적으로 달라진다.
optimizer.step()
```

Optimizer: torch.optim

- 동작 방식

독립변수 **x**



함수값 계산 loss = F(var1, var2)
그래디언트 계산 loss.backward()

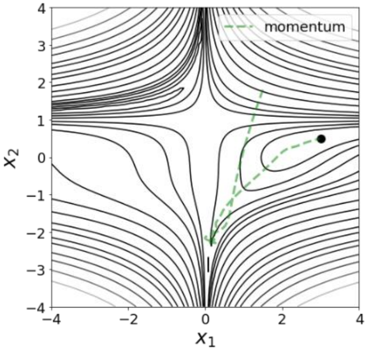
Optimizer
- SGD
- Momentum
- RMSProp
- Adam
- …

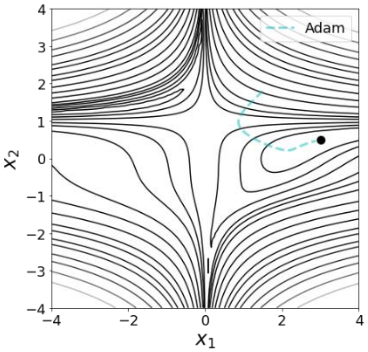
업데이트
optimizer.step()

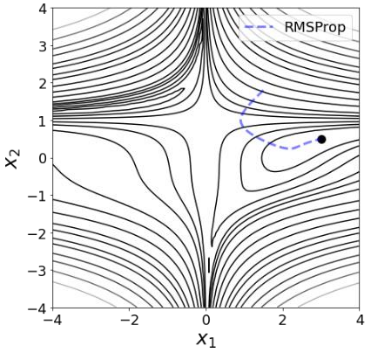
Optimizer: torch.optim

- 실습: Momentum, RMSProp, Adam등 다양한 옵티마이저 사용 가능

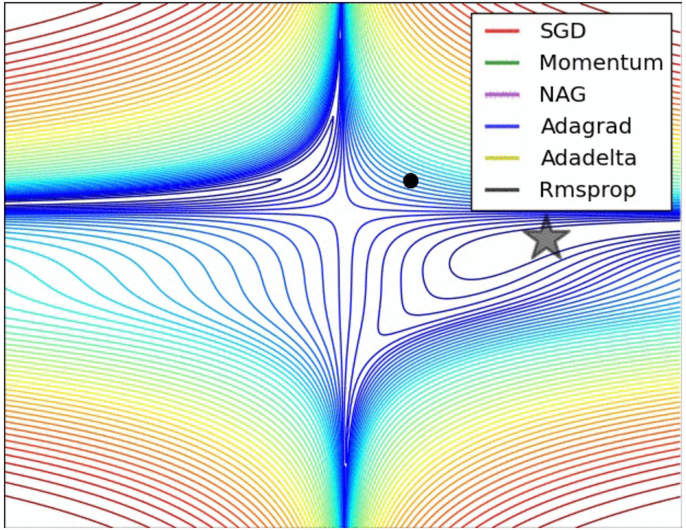
pytorch



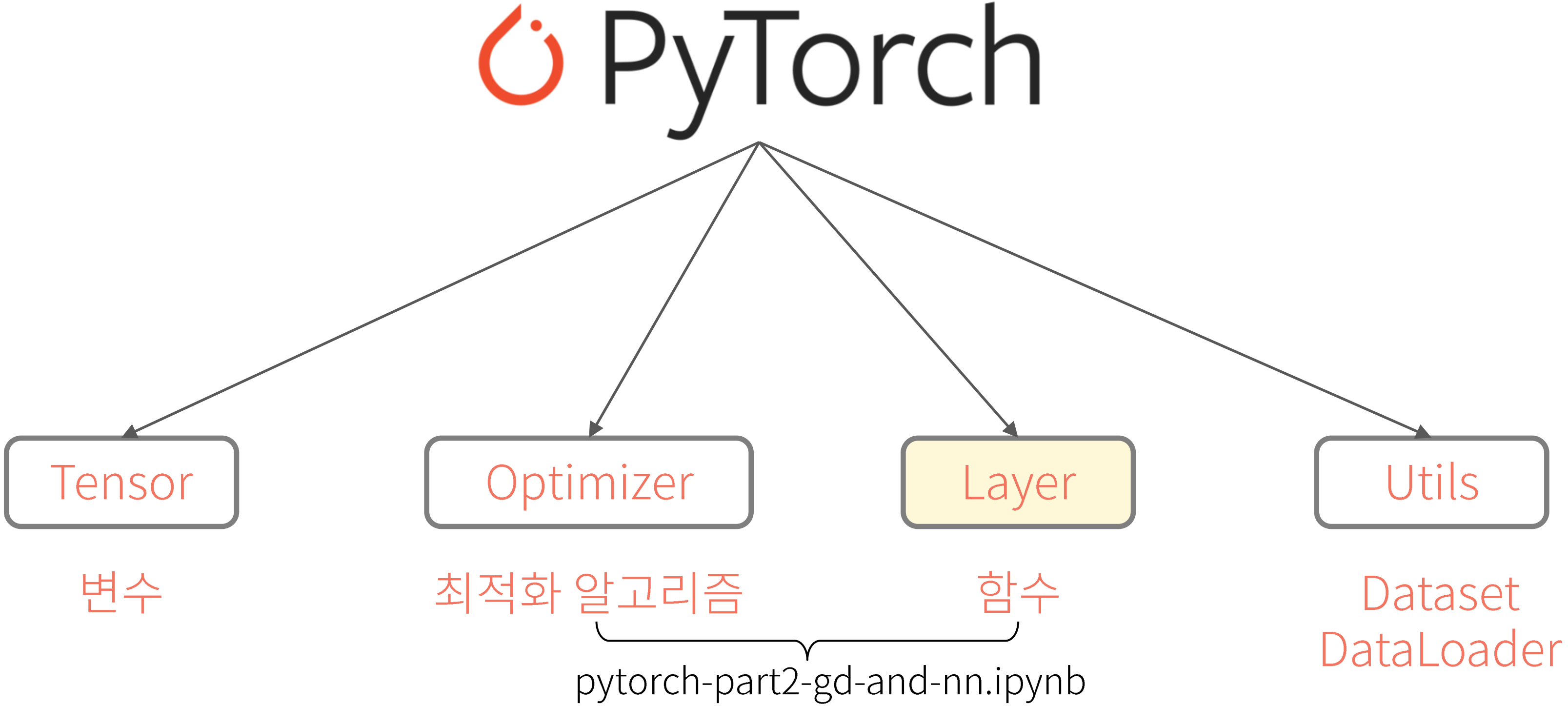




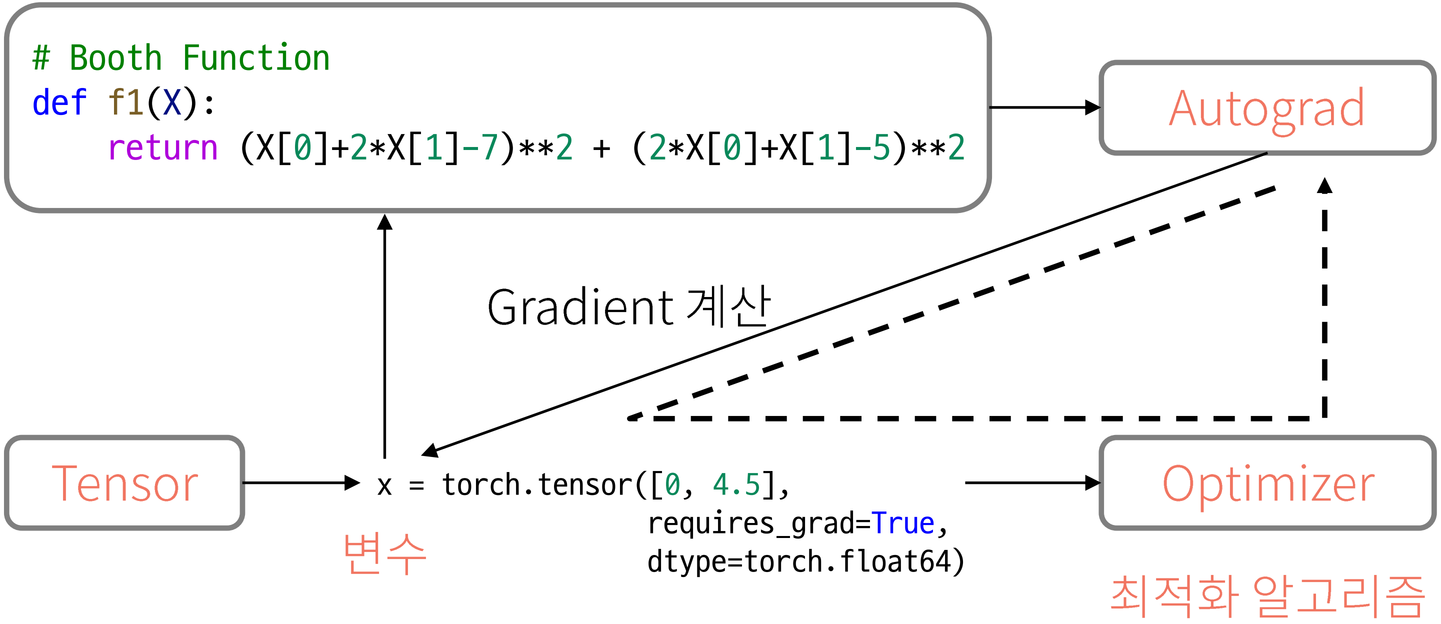
https://ruder.io/optimizing-gradient-descent/, Sebastian Ruder



파이토치|PyTorch



Previously on PyTorch



Basic building blocks for graph: torch.nn

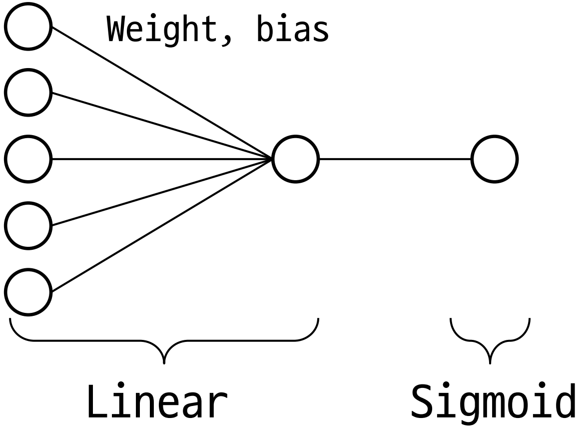
TORCH.NN

These are the basic building blocks for graphs:

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- Shuffle Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities
- Quantized Functions
- Lazy Modules Initialization

Container: torch.nn.sequence

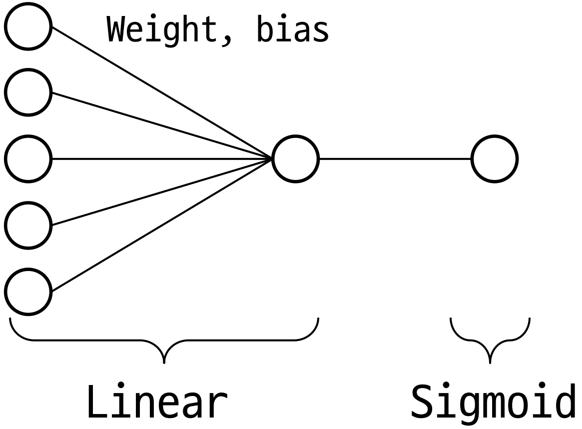
A sequential container. Modules will be added to it in the order they are passed in the constructor.




```
torch.nn.Sequential(
    torch.nn.Linear(5, 1),
    torch.nn.Sigmoid()
)
```

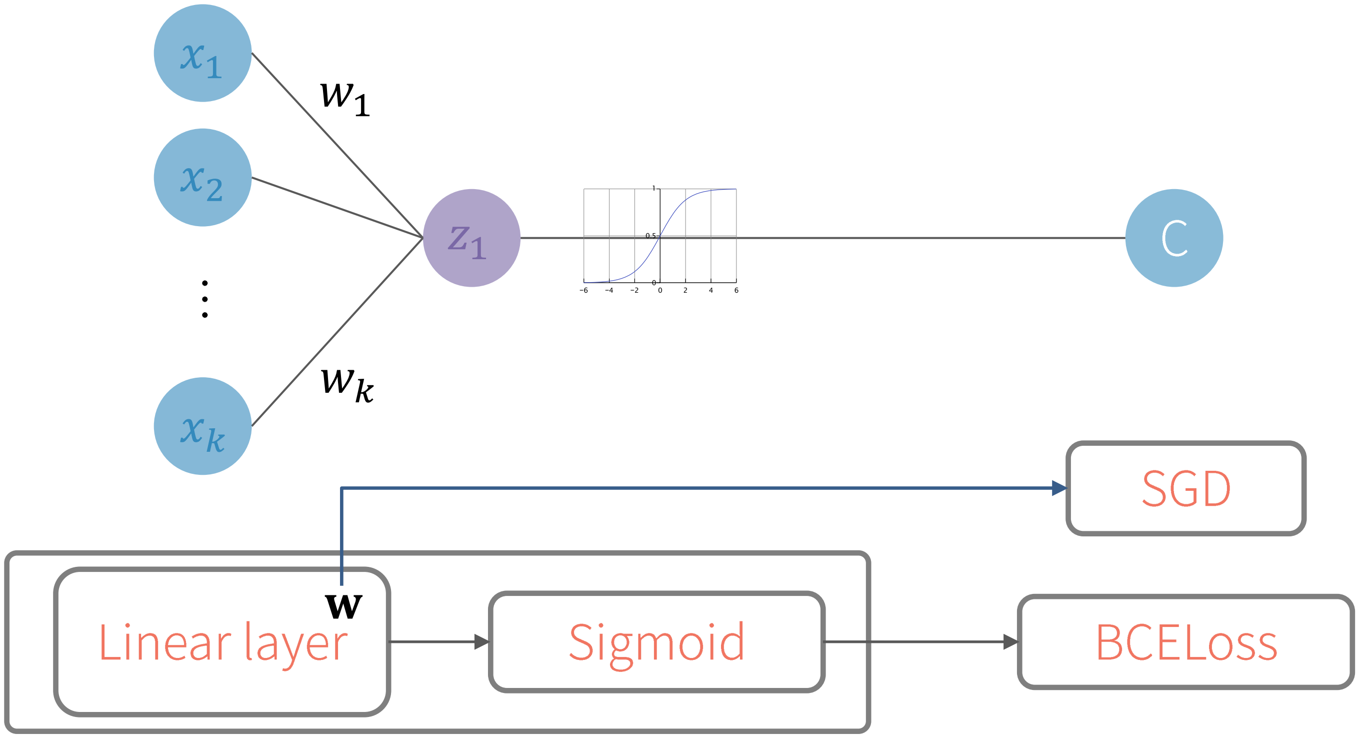
Container: torch.nn.Module

Base class for all neural network modules



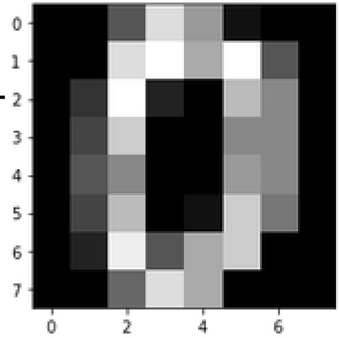

```
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(5,1)
        self.activation = torch.nn.Sigmoid()

    def forward(self, x):
        x = self.linear(x)
        o = self.activation(x)

        return o
```

Logistic Regression by PyTorch



Logistic Regression by PyTorch


```
# model define
models_sigmoid = [ torch.nn.Sequential(
                        torch.nn.Linear(64, 1),
                        torch.nn.Sigmoid()
                ) for i in range(10) ]
```

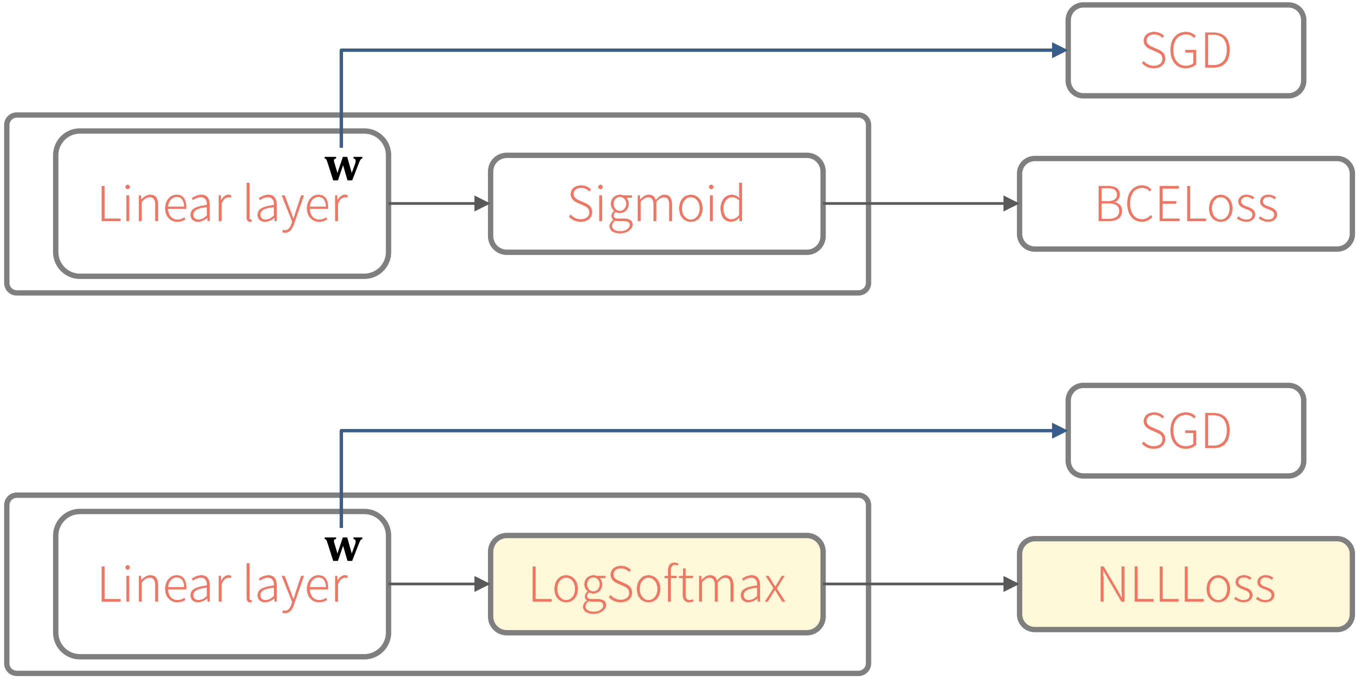


Logistic Regression by PyTorch


```python
# loss and optimizer
loss_fn = torch.nn.BCELoss(reduction='sum')
sgds = [ torch.optim.SGD(
          models_sigmoid[i].parameters(), lr=0.001
         ) for i in range(10)]
```

Softmax Regression by PyTorch

- 실습: Softmax regression으로 바꾸기



파이토치|PyTorch

PyTorch

pytorch-part3-fashin-mnist.ipynb

Tensor	Optimizer	Layer	Utils
변수	최적화 알고리즘	함수	Dataset DataLoader

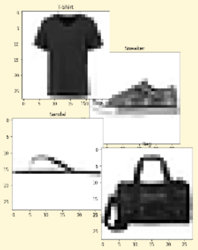
Utilities: Dataset

- torch.utils.data.Dataset
- 데이터를 가공하여 준비하고 인덱스로 접근하는 기능 제공

```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
```

Dataset

```
__init__()
__len__()
__getitem()__
```



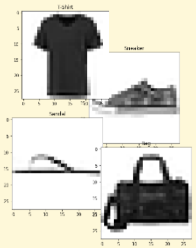
https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

Utilities: Custom Dataset

- torch.utils.data.Dataset을 상속받아
- __init__, __len__, __getitem__만 구현

Dataset

```
__init__()
__len__()
__getitem()__
```



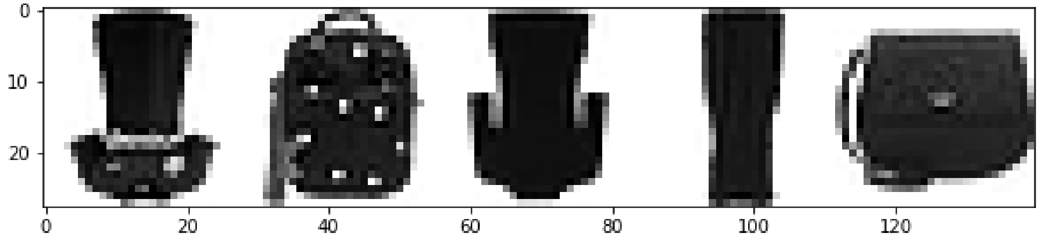
```python
class CustomImageDataset(Dataset):
    def __init__(self, csv, transform=None):
        self.data = pd.read_csv(csv).to_numpy()
        self.transform = transform

    def __len__(self):
        return self.data.shape[0]

    def __getitem__(self, idx):
        label = self.data[idx, 0]
        image = self.data[idx, 1:]

        if self.transform:
            image = self.transform(image)
        return image, label
```

https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

Utilities: transforms

- torchvision.transforms
- 이미지 데이터인 경우 데이터에 변형을 주는 함수

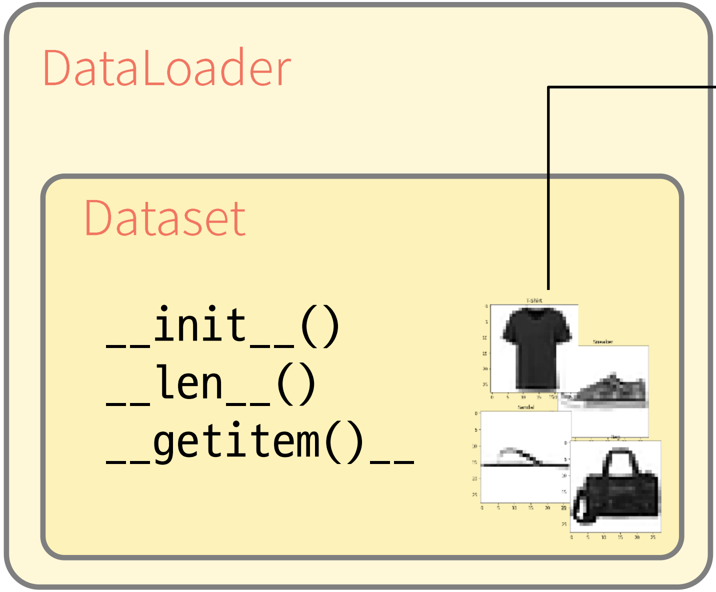
```
T = transforms.Compose([
    # ndarray->pillow image 아래쪽 flip을 수행하기 위해
    transforms.ToPILImage(),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.ToTensor(),
])

D_train = CustomImageDataset('fashion-mnist_train.csv', transform=T)
```



https://pytorch.org/vision/stable/auto_examples/plot_transforms.html

Utilities: DataLoader

- torch.utils.data.DataLoader
- Combines a dataset and a sampler, and provides an iterable over the given dataset.

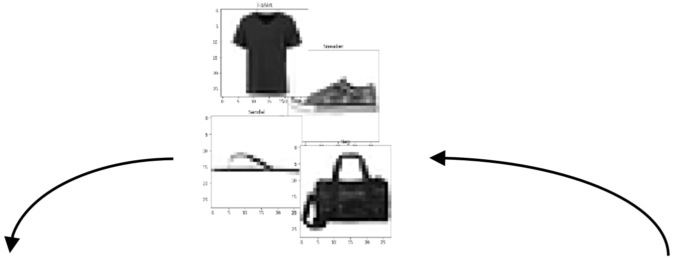


samples

- Automatic batching
- Single- and Multi-process Data Loading
- …

https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

Utilities: DataLoader



```python
for i, (X_batched, y_batched) in enumerate(train_loader):
    # zerograd, inference, loss, backward, step
    optimizer.zero_grad()
    score = model(X_batched)
    loss = criterion(score, y_batched)
    loss.backward()
    optimizer.step()
```

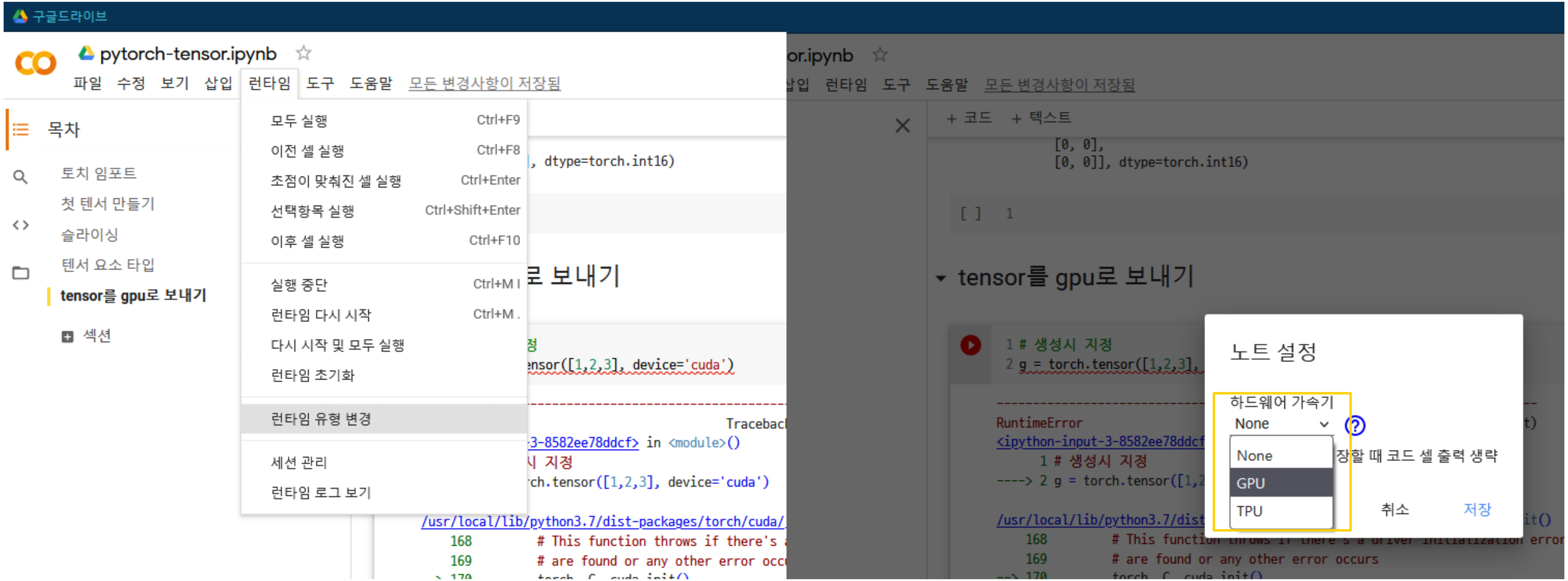
tensor를 gpu로 보내기

- Gpu로 연산을 처리해기 위해 tensor를 gpu 메모리로 올려야 함
- device
  - 생성시 지정
  - .to()
  - .cuda(), .cpu()
  - torch.cuda.FloatTensor

- 각 device에 있는 tensor끼리만 계산

tensor를 gpu로 보내기

- Colab 런타임 유형을 gpu로 변경



FashionMNIST

- An MNIST-like dataset of 70,000, 28x28 labeled fashion images

https://www.kaggle.com/zalando-research/fashionmnist

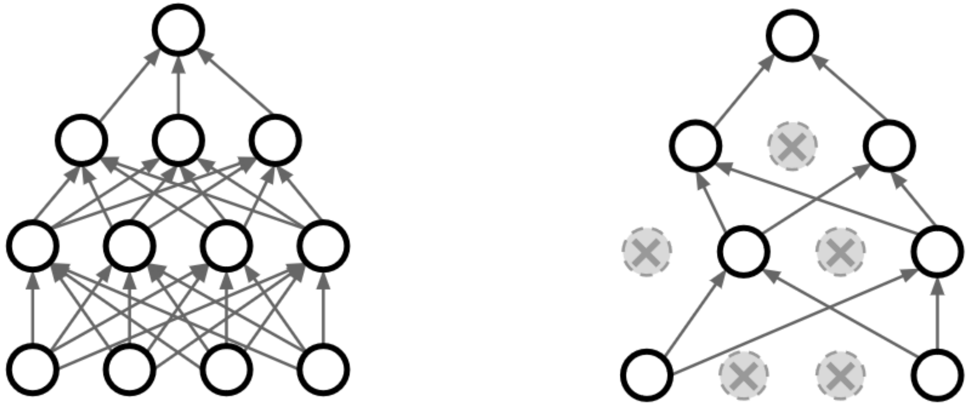
FashionMNIST:실습

- FashionMNIST 데이터 셋 만들기
  - `torchvision.datasets.FashionMNIST`
  - 사용자 정의 Dataset
  - Image Augmentation
- 정의된 Dataset, DataLoader로
  - Softmax regression
  - MLP
    - NLLLoss
    - BCELoss
  - CNN

Dropout

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



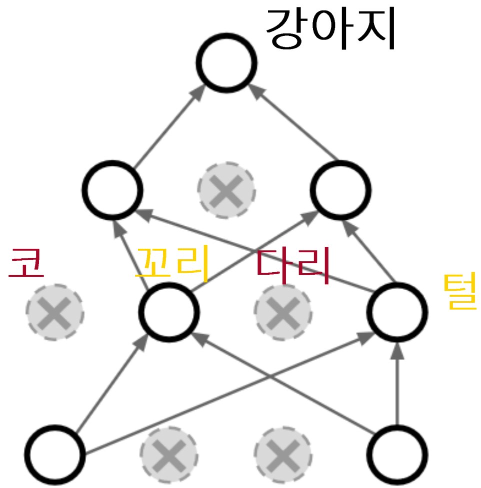
Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Slide credit: cs231n 2021

Dropout

Regularization: Dropout

How can this possibly be a good idea?



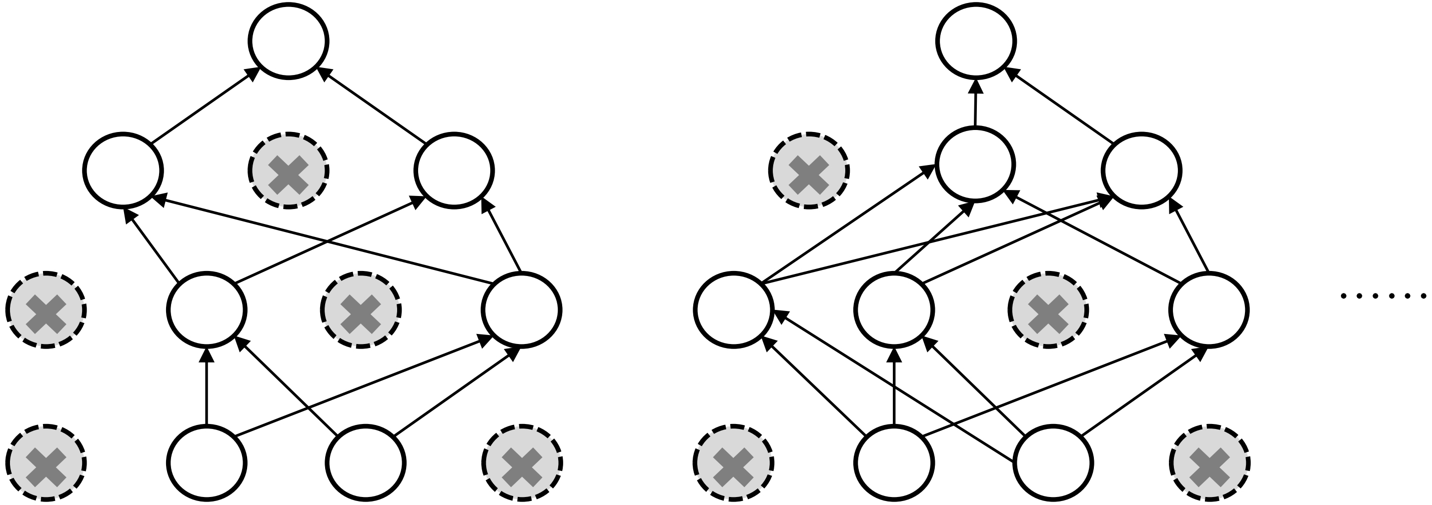
Prevents co-adaptation of features

Slide credit: cs231n 2021

Dropout

Regularization: Dropout

How can this possibly be a good idea?



Dropout is training a large ensemble of models
(that share parameters).

Dropout

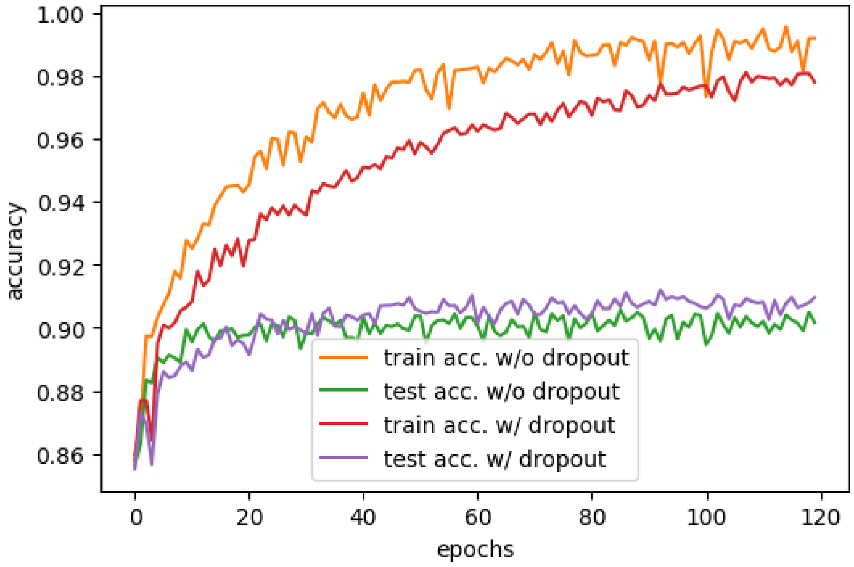
Linear(784,784)

ReLu

Linear(784,256)

ReLu

Linear(256,10)



Linear(784,784)

ReLu

Dropout(0.3)

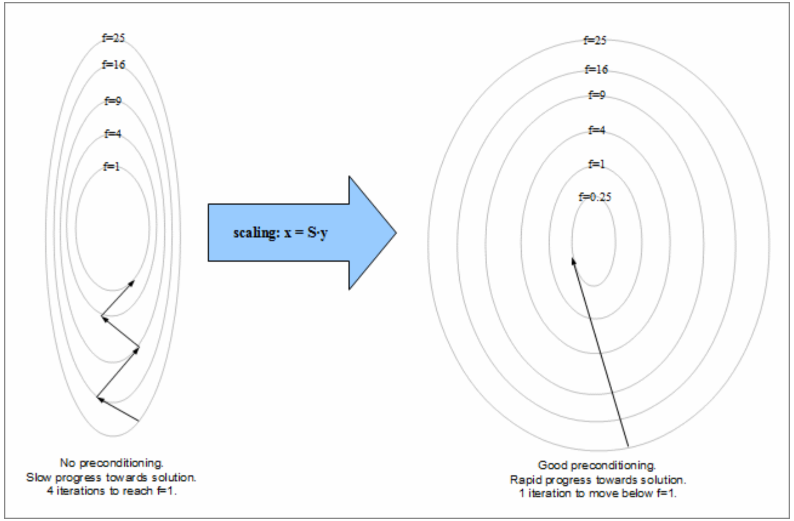
Linear(784,256)

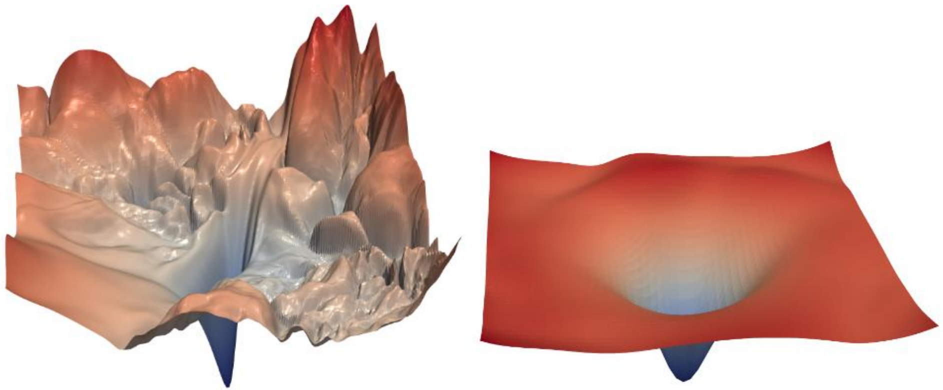
ReLu

Linear(256,10)

Preconditioning

- 최적화 잘되도록 목적함수를 조정





(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Image credit: https://www.alglib.net/optimization/preconditioning.php
Visualizing the Loss Landscape of Neural Nets, https://arxiv.org/pdf/1712.09913.pdf

Batch Normalization

Batch Normalization

[Ioffe and Szegedy, 2015]

“you want zero-mean unit-variance activations? just make them so.”

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

this is a vanilla differentiable function...

Slide credit: cs231n 2021

Batch Normalization Forward Pass

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

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

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

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

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

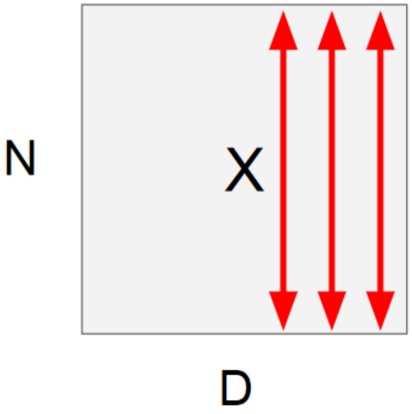
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization Forward Pass

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

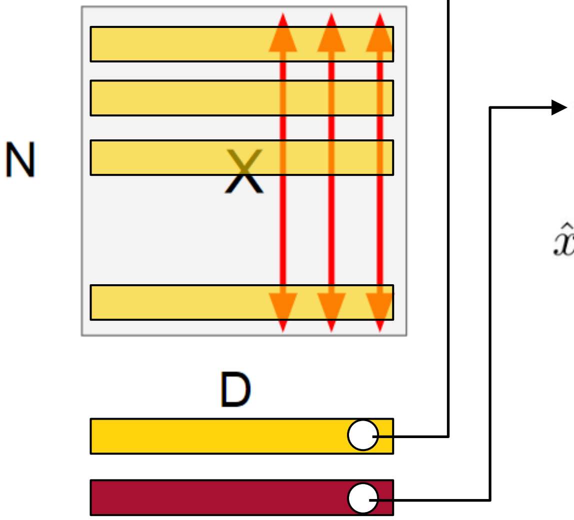
Normalized x, Shape is N x D

Batch Normalization Forward Pass

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



샘플 인덱스

차원 인덱스

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

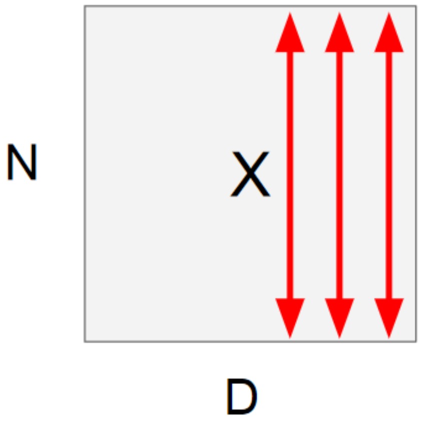
Normalized x, Shape is N x D

Batch Normalization Forward Pass

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Problem: What if zero-mean, unit variance is too hard of a constraint?

Batch Normalization Forward Pass

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

$$\boxed{y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j}$$ Output, Shape is N x D

Batch Norm.: Test Time

Batch Normalization: Test-Time

Estimates depend on minibatch; can't do this at test-time!

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\boxed{\begin{aligned} \mu_j &= \frac{1}{N}\sum_{i=1}^{N} x_{i,j} && \text{Per-channel mean, shape is D} \\ \sigma_j^2 &= \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2 && \text{Per-channel var, shape is D} \end{aligned}}$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Slide credit: cs231n 2021

Batch Norm.: Test Time

Batch Normalization: Test-Time

Input: $x : N \times D$

Learnable scale and shift parameters:

$\gamma, \beta : D$

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

Slide credit: cs231n 2021

BatchNorm

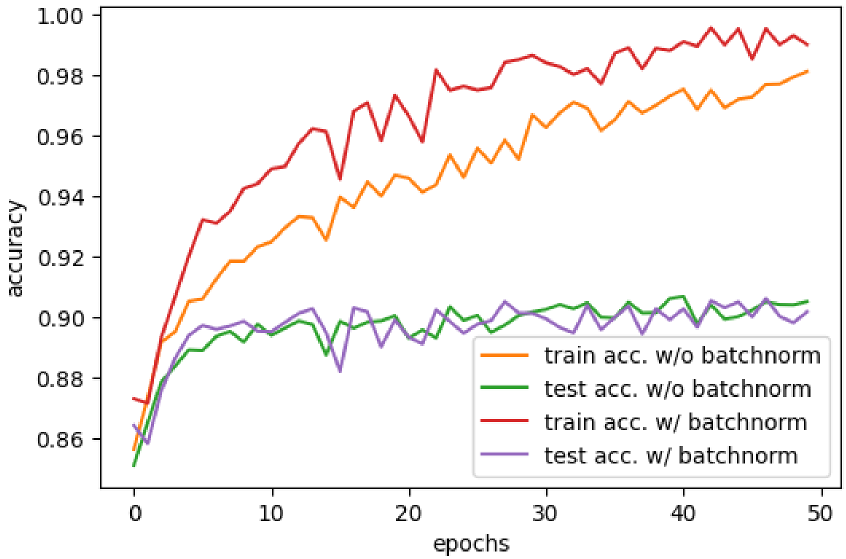
Linear(784,784)

ReLu

Linear(784,256)

ReLu

Linear(256,10)



Linear(784,784)

BatchNorm1d(784)

ReLu

Linear(784,256)

BatchNorm1d(256)

ReLu

Linear(256,10)