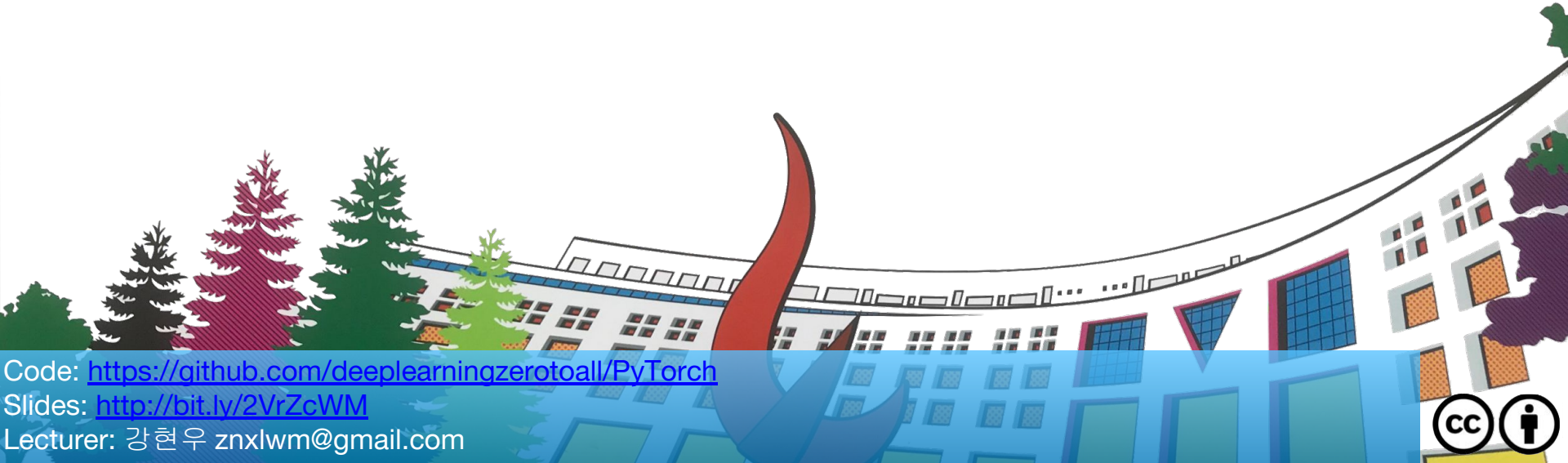
ML/DL for Everyone Season2

ReLU



Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Lecturer: 강현우 znxlwm@gmail.com

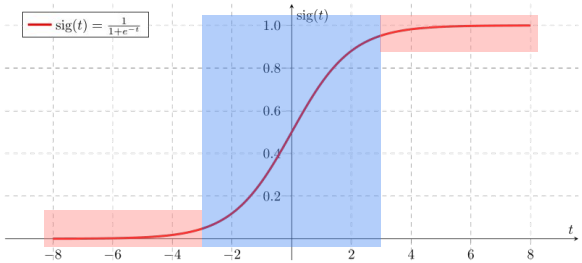
ReLU

- Problem of Sigmoid
- ReLU
- Optimizer in PyTorch
- Review: MNIST
- Code: mnist_softmax
- Code: mnist_nn

Problem of Sigmoid

Input → Network → Output

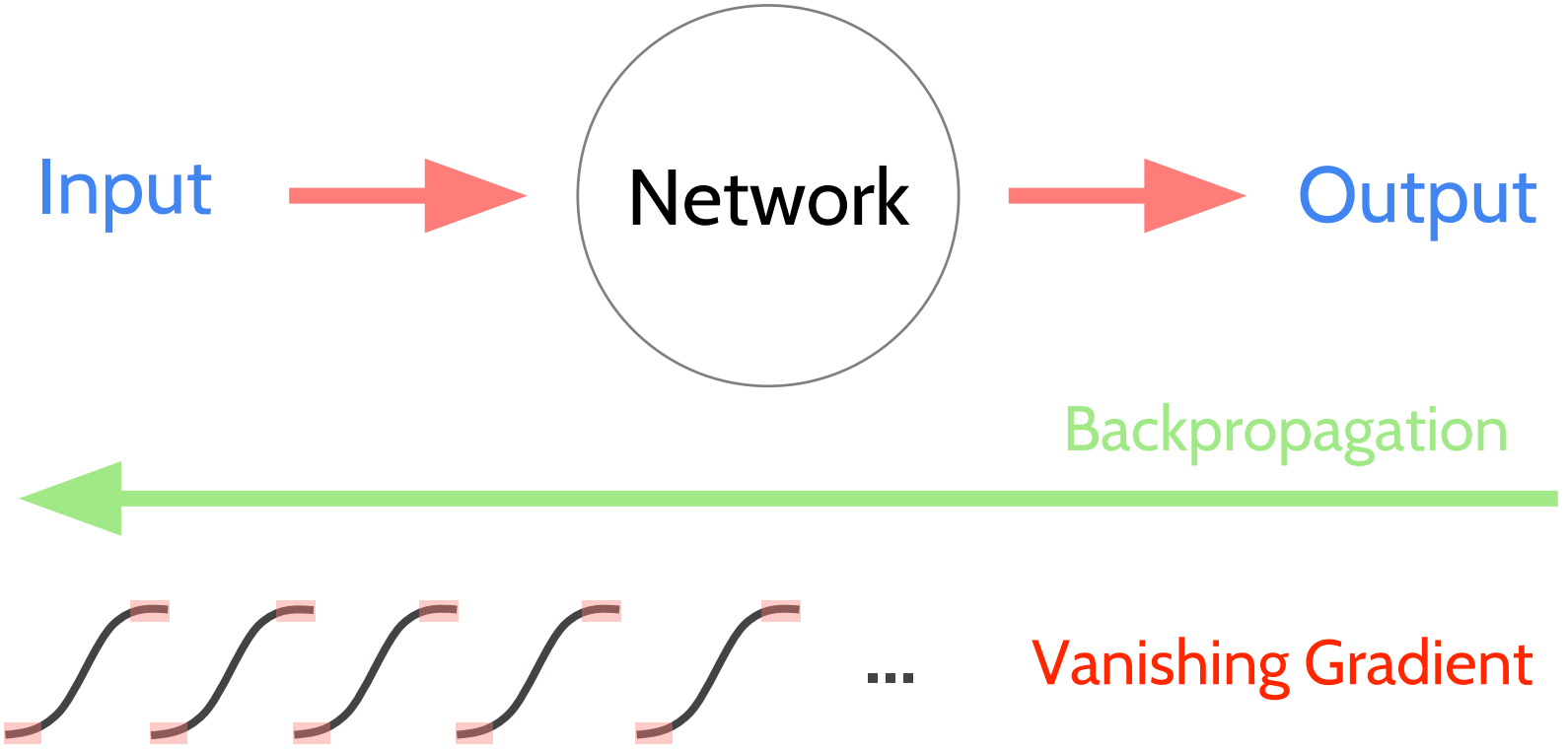
Backpropagation ←



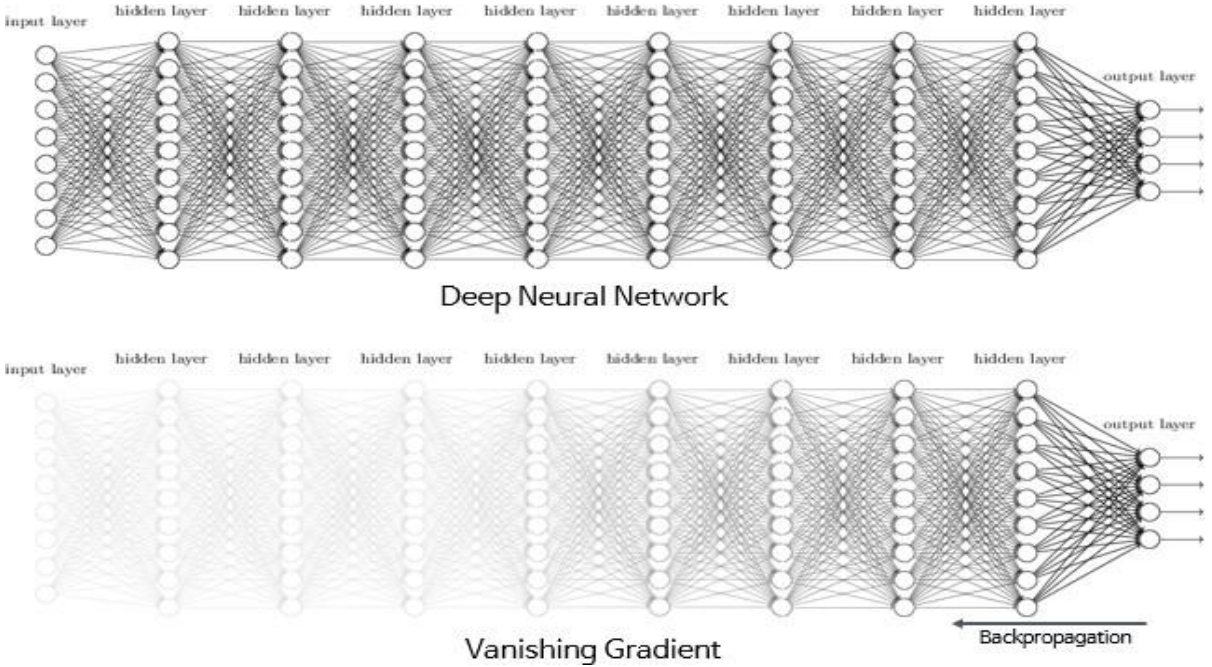
Vanishing Gradient

https://commons.wikimedia.org/wiki/File:Sigmoid-function-2.svg

Problem of Sigmoid

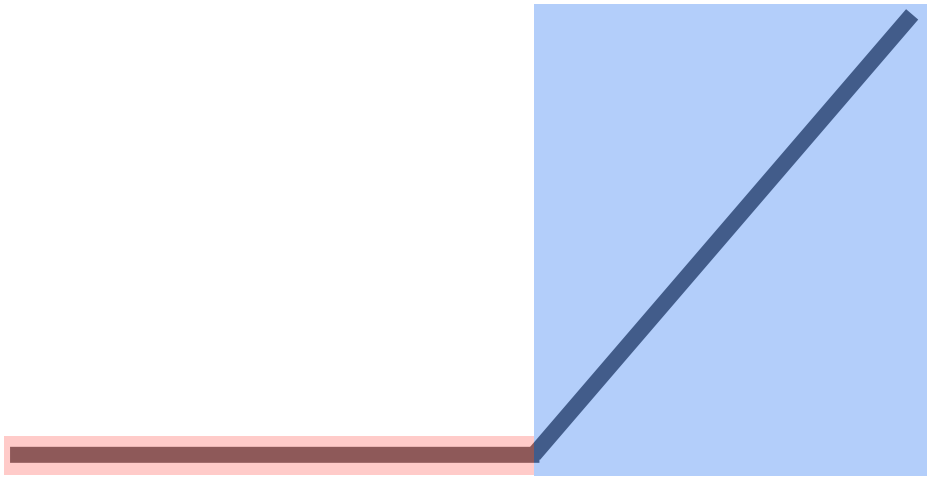


Problem of Sigmoid



ReLU

$$f(x) = \max(0, x)$$



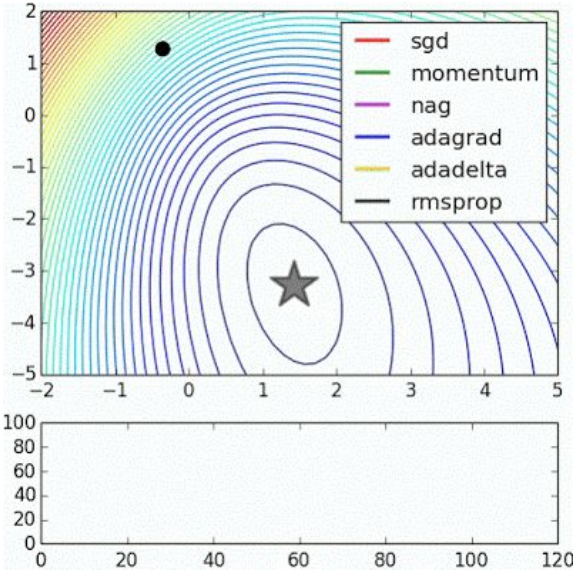
```
x = torch.nn.sigmoid(x)

x = torch.nn.relu(x)
```

```
torch.nn.sigmoid(x)
torch.nn.tanh(x)
torch.nn.relu(x)
torch.nn.leaky_relu(x, 0.01)
```

Optimizer in PyTorch

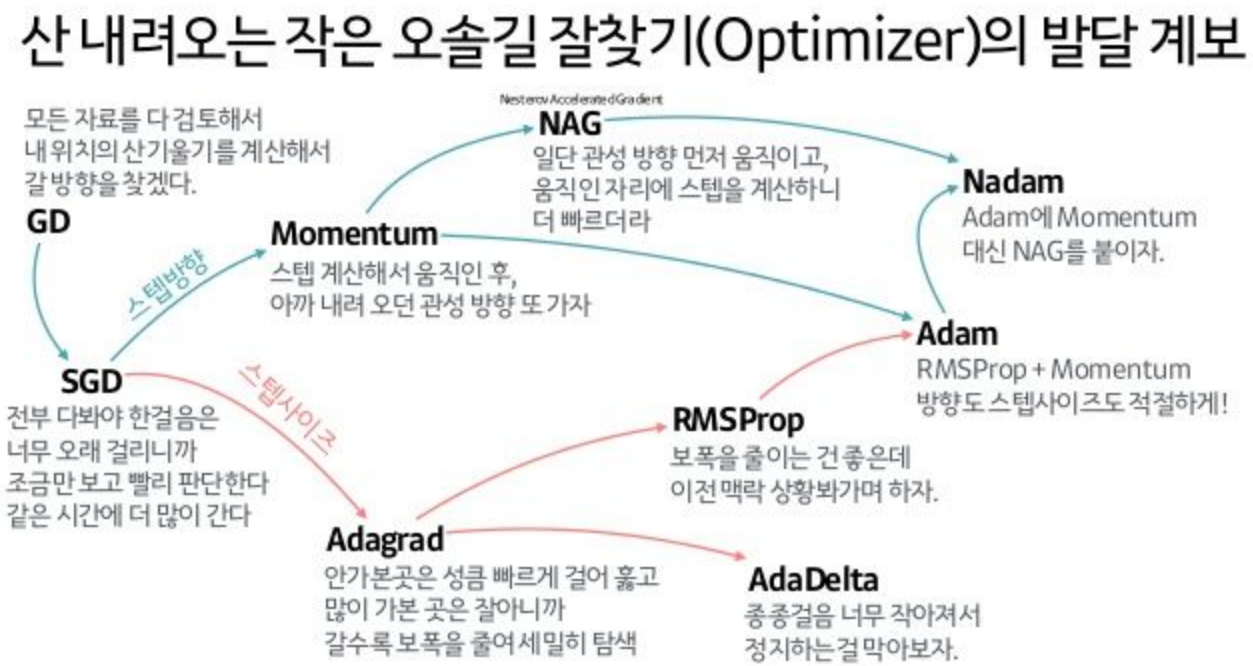
`torch.optim` is a package implementing various optimization algorithms. Most commonly used methods are already supported, and the interface is general enough, so that more sophisticated ones can be also easily integrated in the future.

- torch.optim.SGD
- torch.optim.Adadelta
- torch.optim.Adagrad
- torch.optim.Adam
- torch.optim.SparseAdam
- torch.optim.Adamax
- torch.optim.ASGD
- torch.optim.LBFGS
- torch.optim.RMSprop
- torch.optim.Rprop



https://pytorch.org/docs/master/optim.html#torch-optim
http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Optimizers



https://www.slideshare.net/yongho/ss-79607172

Review: reading data

```
import torchvision.datasets as dsets
...
mnist_train = dsets.MNIST(root="MNIST_data/", train=True, transform=transforms.ToTensor(),
download=True)

mnist_test = dsets.MNIST(root="MNIST_data/", train=False, transform=transforms.ToTensor(),
download=True)

data_loader = torch.utils.DataLoader(DataLoader=mnist_train, batch_size=batch_size,
shuffle=True, drop_last=True)
...
for epoch in range(training_epochs):
...
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
```

Code: mnist_softmax

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
...

# MNIST data image of shape 28 * 28 = 784
linear = torch.nn.Linear(784, 10, bias=True).to(device)

# Initialization
torch.nn.init.normal_(linear.weight)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(linear.parameters(), lr=learning_rate)
```

Train

```
total_batch = len(data_loader)
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)

    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        optimier.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        avg_cost += cost / total_batch

    print("Epoch: ", "%04d" % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

```
Epoch: 0001 cost = 4.848181725
Epoch: 0002 cost = 1.464641452
Epoch: 0003 cost = 0.977406442
Epoch: 0004 cost = 0.790303528
Epoch: 0005 cost = 0.686833322
Epoch: 0006 cost = 0.618483305
Epoch: 0007 cost = 0.568978727
Epoch: 0008 cost = 0.531290889
Epoch: 0009 cost = 0.501056492
Epoch: 0010 cost = 0.476258367
Epoch: 0011 cost = 0.455025375
Epoch: 0012 cost = 0.437031567
Epoch: 0013 cost = 0.421489984
Epoch: 0014 cost = 0.408599854
Epoch: 0015 cost = 0.396514893
Learning finished
Accuracy: 0.8944000005722046
```

Code: mnist_nn

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
...

# MNIST data image of shape 28 * 28 = 784
linear1 = torch.nn.Linear(784, 256, bias=True).to(device)
linear2 = torch.nn.Linear(256, 256, bias=True).to(device)
linear3 = torch.nn.Linear(256, 10, bias=True).to(device)
relu = torch.nn.ReLU()

# Initialization
torch.nn.init.normal_(linear1.weight)
torch.nn.init.normal_(linear2.weight)
torch.nn.init.normal_(linear3.weight)

# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3).to(device)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
Epoch: 0001 cost = 129.332550049
Epoch: 0002 cost = 36.226955414
Epoch: 0003 cost = 22.935277939
Epoch: 0004 cost = 16.051362991
Epoch: 0005 cost = 11.619537354
Epoch: 0006 cost = 8.622934341
Epoch: 0007 cost = 6.308451176
Epoch: 0008 cost = 4.843585014
Epoch: 0009 cost = 3.543032646
Epoch: 0010 cost = 2.632565498
Epoch: 0011 cost = 2.056926012
Epoch: 0012 cost = 1.580777287
Epoch: 0013 cost = 1.261717200
Epoch: 0014 cost = 1.084918380
Epoch: 0015 cost = 0.804111660
Learning finished
Accuracy: 0.9465000033378601
```

What’s Next?

- Weight initialization