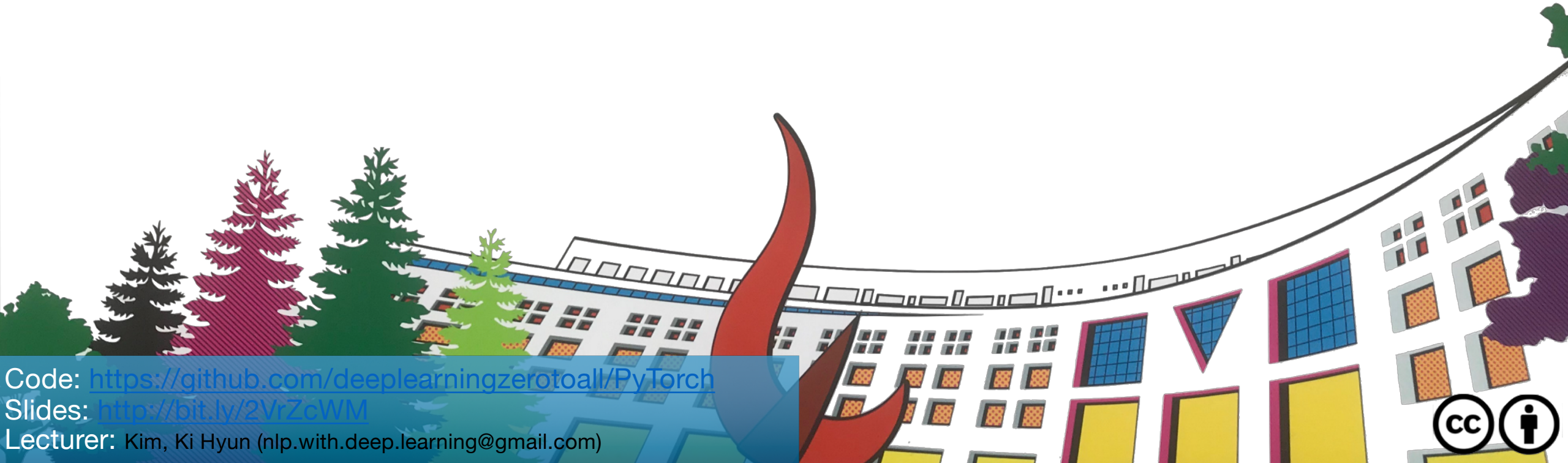
ML/DL for Everyone Season2

with PYTORCH

Softmax Classification



Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Lecturer: Kim, Ki Hyun (nlp.with.deep.learning@gmail.com)

Softmax Classification

- Softmax
- Cross Entropy
- Low-level Implementation
- High-level Implementation
- Training Example

Import

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
# For reproducibility
torch.manual_seed(1)
```

```
<torch._C.Generator at 0x7f8740076fb0>
```

Discrete Probability Distribution

Softmax

Convert numbers to probabilities with softmax.

$$P(class = i) = \frac{e^i}{\sum e^i}$$

```
z = torch.FloatTensor([1, 2, 3])
```

PyTorch has a `softmax` function.

```
hypothesis = F.softmax(z, dim=0)
print(hypothesis)
```

```
tensor([0.0900, 0.2447, 0.6652])
```

Since they are probabilities, they should add up to 1. Let's do a sanity check.

```
hypothesis.sum()
```

```
tensor(1.)
```

Cross Entropy

$$H(P, Q) = -\mathbb{E}_{\mathrm{x} \sim P(\mathrm{x})}[\log Q(\mathrm{x})] = -\sum_{x \in \mathcal{X}} P(x) \log Q(x)$$

Cross Entropy Loss (Low-level)

For multi-class classification, we use the cross entropy loss.

$$L = \frac{1}{N} \sum -y \log(\hat{y})$$

where $\hat{y}$ is the predicted probability and y is the correct probability (0 or 1).

```
z = torch.rand(3, 5, requires_grad=True)
hypothesis = F.softmax(z, dim=1)
print(hypothesis)
```

```
tensor([[0.2645, 0.1639, 0.1855, 0.2585, 0.1277],
        [0.2430, 0.1624, 0.2322, 0.1930, 0.1694],
        [0.2226, 0.1986, 0.2326, 0.1594, 0.1868]], grad_fn=<SoftmaxBackwa
rd>)
```

```
y = torch.randint(5, (3,)).long()
print(y)
```

```
tensor([0, 2, 1])
```

Cross Entropy Loss (Low-level)

```
y_one_hot = torch.zeros_like(hypothesis)
y_one_hot.scatter_(1, y.unsqueeze(1), 1)
```

```
tensor([[1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 1., 0., 0., 0.]])
```

```
cost = (y_one_hot * -torch.log(hypothesis)).sum(dim=1).mean()
print(cost)
```

```
tensor(1.4689, grad_fn=<MeanBackward1>)
```

Cross-entropy Loss with torch.nn.functional

```
# Low level
torch.log(F.softmax(z, dim=1))
```

```
tensor([[-1.3301, -1.8084, -1.6846, -1.3530, -2.0584],
        [-1.4147, -1.8174, -1.4602, -1.6450, -1.7758],
        [-1.5025, -1.6165, -1.4586, -1.8360, -1.6776]], grad_fn=<LogBackw
ard>)
```

```
# High level
F.log_softmax(z, dim=1)
```

```
tensor([[-1.3301, -1.8084, -1.6846, -1.3530, -2.0584],
        [-1.4147, -1.8174, -1.4602, -1.6450, -1.7758],
        [-1.5025, -1.6165, -1.4586, -1.8360, -1.6776]],
       grad_fn=<LogSoftmaxBackward>)
```

Cross-entropy Loss with torch.nn.functional

```
# Low level
(y_one_hot * -torch.log(F.softmax(z, dim=1))).sum(dim=1).mean()
```

```
tensor(1.4689, grad_fn=<MeanBackward1>)
```

```
# High level
F.nll_loss(F.log_softmax(z, dim=1), y)
```

```
tensor(1.4689, grad_fn=<NllLossBackward>)
```

PyTorch also has `F.cross_entropy` that combines `F.log_softmax()` and `F.nll_loss()`.

```
F.cross_entropy(z, y)
```

```
tensor(1.4689, grad_fn=<NllLossBackward>)
```

Cross-entropy Loss with torch.nn.functional

```
F.cross_entropy(z, y)
```

```
tensor(1.4689, grad_fn=<NllLossBackward>)
```

Training with Low-level Cross Entropy Loss

```
x_train = [[1, 2, 1, 1],
           [2, 1, 3, 2],
           [3, 1, 3, 4],
           [4, 1, 5, 5],
           [1, 7, 5, 5],
           [1, 2, 5, 6],
           [1, 6, 6, 6],
           [1, 7, 7, 7]]
y_train = [2, 2, 2, 1, 1, 1, 0, 0]
x_train = torch.FloatTensor(x_train)
y_train = torch.LongTensor(y_train)
```

Training with Low-level Cross Entropy Loss

```
# 모델 초기화
W = torch.zeros((4, 3), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
# optimizer 설정
optimizer = optim.SGD([W, b], lr=0.1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # Cost 계산 (1)
    hypothesis = F.softmax(x_train.matmul(W) + b, dim=1) # or .mm or @
    y_one_hot = torch.zeros_like(hypothesis)
    y_one_hot.scatter_(1, y_train.unsqueeze(1), 1)
    cost = (y_one_hot * -torch.log(F.softmax(hypothesis, dim=1))).sum(dim=1

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

```
Epoch    0/1000 Cost: 1.098612
Epoch  100/1000 Cost: 0.901535
Epoch  200/1000 Cost: 0.839114
Epoch  300/1000 Cost: 0.807826
Epoch  400/1000 Cost: 0.788472
Epoch  500/1000 Cost: 0.774822
Epoch  600/1000 Cost: 0.764449
Epoch  700/1000 Cost: 0.756191
Epoch  800/1000 Cost: 0.749398
Epoch  900/1000 Cost: 0.743671
Epoch 1000/1000 Cost: 0.738749
```

Training with F.cross_entropy

```
# 모델 초기화
W = torch.zeros((4, 3), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
# optimizer 설정
optimizer = optim.SGD([W, b], lr=0.1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # Cost 계산 (2)
    z = x_train.matmul(W) + b # or .mm or @
    cost = F.cross_entropy(z, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

```
Epoch    0/1000 Cost: 1.098612
Epoch  100/1000 Cost: 0.761050
Epoch  200/1000 Cost: 0.689991
Epoch  300/1000 Cost: 0.643229
Epoch  400/1000 Cost: 0.604117
Epoch  500/1000 Cost: 0.568255
Epoch  600/1000 Cost: 0.533922
Epoch  700/1000 Cost: 0.500291
Epoch  800/1000 Cost: 0.466908
Epoch  900/1000 Cost: 0.433507
Epoch 1000/1000 Cost: 0.399962
```

High-level Implementation with nn.Module

```
class SoftmaxClassifierModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(4, 3) # Output0| 3!

    def forward(self, x):
        return self.linear(x)
```

```
model = SoftmaxClassifierModel()
```

High-level Implementation with nn.Module

```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=0.1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    prediction = model(x_train)

    # cost 계산
    cost = F.cross_entropy(prediction, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

```
Epoch    0/1000 Cost: 1.849513
Epoch  100/1000 Cost: 0.689894
Epoch  200/1000 Cost: 0.609259
Epoch  300/1000 Cost: 0.551218
Epoch  400/1000 Cost: 0.500141
Epoch  500/1000 Cost: 0.451947
Epoch  600/1000 Cost: 0.405051
Epoch  700/1000 Cost: 0.358733
Epoch  800/1000 Cost: 0.312912
Epoch  900/1000 Cost: 0.269521
Epoch 1000/1000 Cost: 0.241922
```