ML/DL for Everyone with PYTORCH

Lecture 6:
Logistic Regression

Sung Kim <hunkim+ml@gmail.com> HKUST

Call for Comments

Please feel free to add comments directly on these slides.

Other slides: http://bit.ly/PyTorchZeroAll

ML/DL for Everyone with PYTORCH

Lecture 6:
Logistic Regression

Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

Linear model
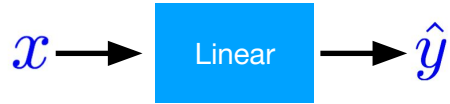
$x \longrightarrow$ Linear $\longrightarrow \hat{y}$

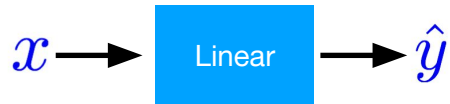| Hours (x)	Points
1	2
2	4
3	6
4	?

Binary prediction (0 or 1) is very useful!

- Spent *N* hours for study, pass or fail?

- GPA and GRE scores for the HKUST PHD program, admit or not?

- Soccer game against Japan, win or lose?

- She/he looks good, propose or not?
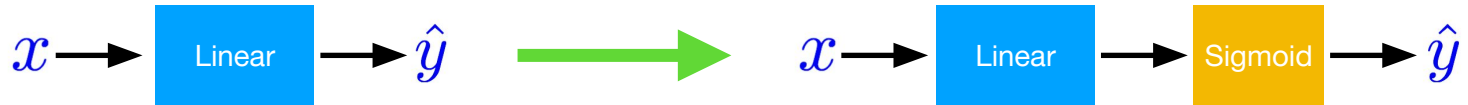
- …

Linear to binary (pass/fail, 0/1)

$x \longrightarrow$ Linear $\longrightarrow \hat{y}$

Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?

Logistic regression: pass/fail (0 or 1)

Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?

Meet Sigmoid

$x \rightarrow$ | Linear | $\rightarrow \hat{y}$ → $x \rightarrow$ | Linear | $\rightarrow$ | Sigmoid | $\rightarrow \hat{y}$

Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?
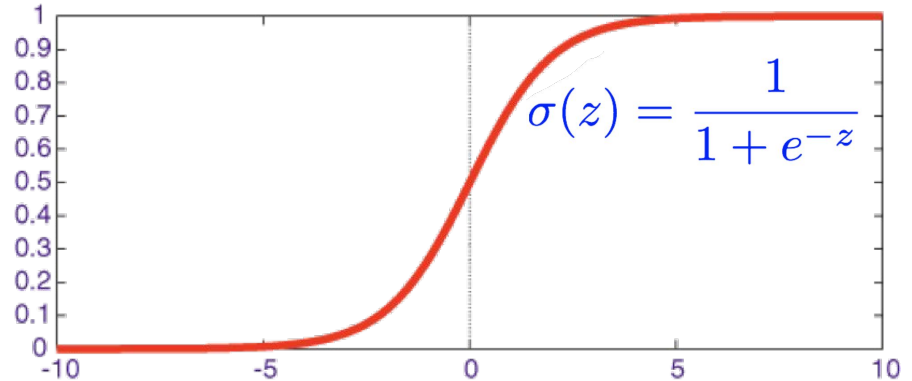
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Meet Sigmoid

$$1 : \hat{y} > 0.5$$

Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Meet sigmoid

$x \longrightarrow$ [Linear] $\longrightarrow \hat{y}$

$x \longrightarrow$ [Linear] $\longrightarrow$ [Sigmoid] $\longrightarrow \hat{y}$
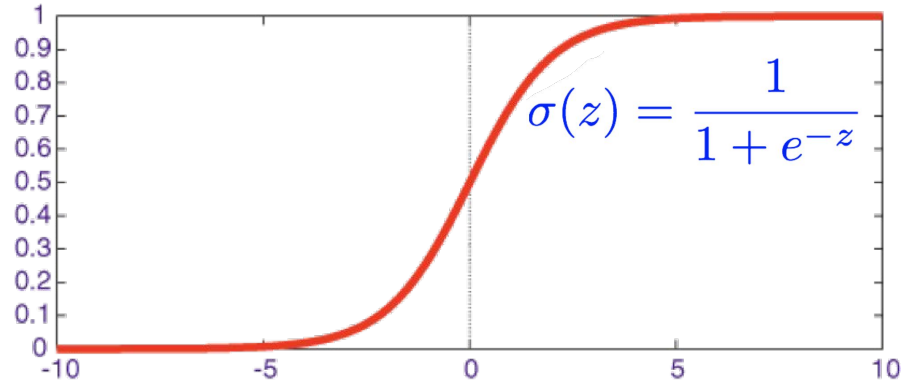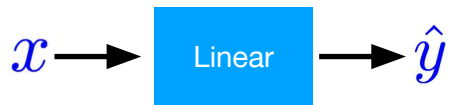
$$\hat{y} = x * w + b$$

$\longrightarrow$
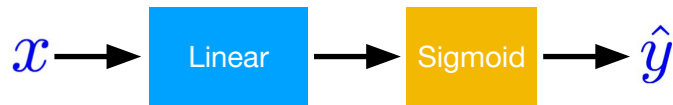
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Meet Cross Entropy Loss

$x \longrightarrow$ [Linear] $\longrightarrow \hat{y}$

$x \longrightarrow$ [Linear] $\longrightarrow$ [Sigmoid] $\longrightarrow \hat{y}$

$$\hat{y} = x * w + b$$

$\longrightarrow$
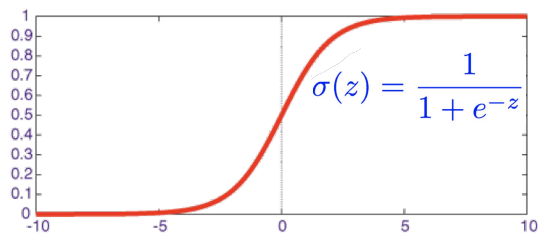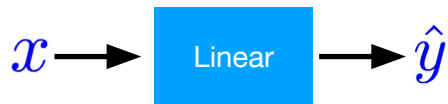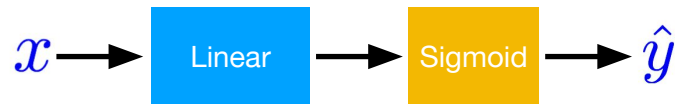
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

$$loss = \frac{1}{N} \sum_{n=1}^{N} (\hat{y_n} - y_n)^2$$

$$loss = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

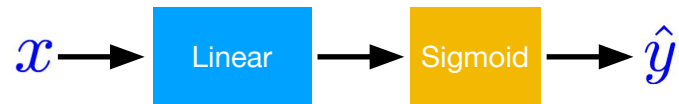| Hours (x)	Points	fail/pass
1	2	0
2	4	0
3	6	1
4	?	?
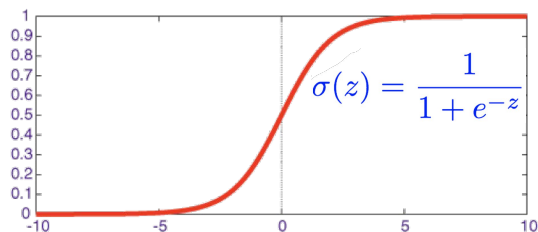
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(Binary) Cross Entropy Loss

$$loss = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

y	y_pred	loss
1	0.2	
1	0.8	
0	0.1	
0	0.9	

z

log(z)

Logistic regression

$$x \rightarrow \boxed{\text{Linear}} \rightarrow \boxed{\text{Sigmoid}} \rightarrow \hat{y}$$
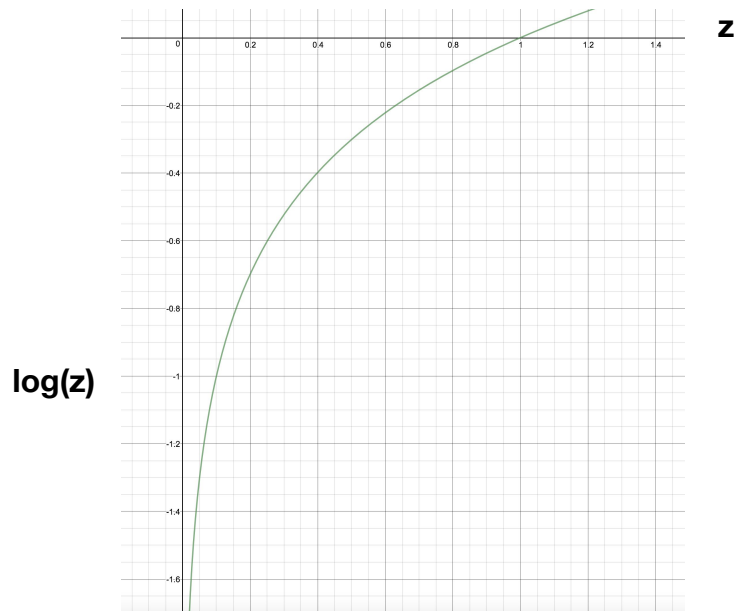
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

`torch.nn.functional.sigmoid(input)`

Applies the element-wise function $f(x) = 1/(1 + exp(-x))$

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

class Model(torch.nn.Module):

    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred
```

Logistic regression

$x \rightarrow$ [Linear] $\rightarrow$ [Sigmoid] $\rightarrow \hat{y}$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = \sigma(x * w + b)$$

Applies the element-wise function $f(x) = 1/(1 + exp(-x))$
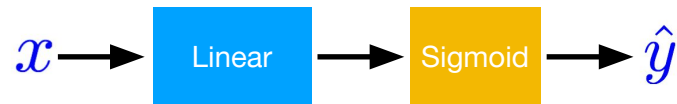
```
import torch.nn.functional as F

class Model(torch.nn.Module):

    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred
```

Creates a criterion that measures the Binary Cross Entropy between the target and the output:

$$loss(o, t) = -1/n \sum_{i} (t[i] * log(o[i]) + (1 - t[i]) * log(1 - o[i]))$$

$$loss = -\frac{1}{N} \sum_{n=1}^{N} y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)$$

```
criterion = torch.nn.BCELoss(size_average=True)
```

Logistic regression

```python
x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()

criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training Loop
for epoch in range(1000):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```

Logistic regression

```python
x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()

criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(1000):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```
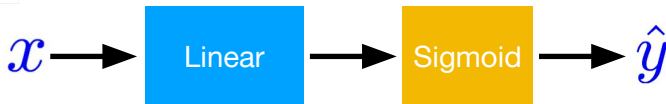
(1) Design your model using class

$$x \rightarrow \boxed{\text{Linear}} \rightarrow \boxed{\text{Sigmoid}} \rightarrow \hat{y}$$

Logistic regression

```python
x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()

criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(1000):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```
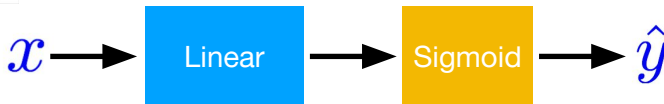
1 **Design your model using class**

$$x \longrightarrow \boxed{\text{Linear}} \longrightarrow \boxed{\text{Sigmoid}} \longrightarrow \hat{y}$$

2 **Construct loss and optimizer (select from PyTorch API)**

3 **Training cycle (forward, backward, update)**

```python
x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()

criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training Loop
for epoch in range(1000):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```

Logistic regression

```
0 1.6369143724441528
1 1.6119738817214966
2 1.5872894525527954
3 1.5628681182861328
4 1.5387169122695923
5 1.514843225479126
6 1.4912540912628174
7 1.467956781387329
8 1.4449583292007446
9 1.4222657680511475
10 1.3998862504959106
...
…
988 0.39138174057006836
989 0.39128318428993225
990 0.39118456840515137
991 0.3910861015319824
992 0.39098766446113586
993 0.3908892273902893
994 0.39079099893569946
995 0.39069271087646484
996 0.3905944228172302
997 0.39049631357192993
998 0.39039820432662964
999 0.3903001546859741
predict 1 hour  1.0 False
predict 7 hours 7.0 True
```

Exercise 6-1: Try other activation functions
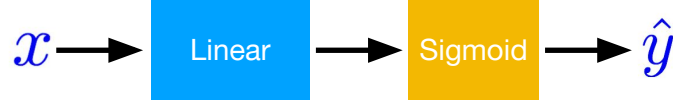
Non-linear Activations
- ReLU
- ReLU6
- ELU
- SELU
- PReLU
- LeakyReLU
- Threshold
- Hardtanh
- Sigmoid
- Tanh

Lecture 7:
Wide and Deep

Backup slides

Building fun models

- Neural Net components
  - CNN
  - RNN
  - Activations

- Losses

- Optimizers

torch.nn

⊟ Convolution Layers

    Conv1d

    Conv2d

    Conv3d

    ConvTranspose1d

    ConvTranspose2d

    ConvTranspose3d

⊟ Recurrent layers

    RNN

    LSTM

    GRU

    RNNCell

    LSTMCell

    GRUCell

⊕ Containers

⊕ Convolution Layers

⊕ Pooling Layers

⊕ Padding Layers

⊕ Non-linear Activations

⊕ Normalization layers

⊕ Recurrent layers

⊕ Linear layers

⊕ Dropout layers

⊕ Sparse layers

⊕ Distance functions

⊕ Loss functions

⊕ Vision layers

⊟ Non-linear Activations

    ReLU

    ReLU6

    ELU

    SELU

    PReLU

    LeakyReLU

    Threshold

    Hardtanh

    Sigmoid

    Tanh

    LogSigmoid

    Softplus

    Softshrink

    Softsign

    Tanhshrink

    Softmin

    Softmax

    Softmax2d

    LogSoftmax

http://pytorch.org/docs/master/nn.html

Loss functions

Table 1: List of losses analysed in this paper. $\mathbf{y}$ is true label as one-hot encoding, $\hat{\mathbf{y}}$ is true label as $+1/-1$ encoding, $\mathbf{o}$ is the output of the last layer of the network, $\cdot^{(j)}$ denotes jth dimension of a given vector, and $\sigma(\cdot)$ denotes probability estimate.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y} - \sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y} - \sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\dfrac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2 + \|\mathbf{y}\|_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \dfrac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

https://arxiv.org/pdf/1702.05659.pdf

torch.optim

- *class*torch.optim.Adadelta
- *class*torch.optim.Adagrad
- *class*torch.optim.Adam
- *class*torch.optim.Adamax
- *class*torch.optim.ASGD
- *class*torch.optim.RMSprop
- *class*torch.optim.Rprop
- *class*torch.optim.SGD

Three simple steps

1 **Design your model using class**

2 **Construct loss and optimizer
(select from PyTorch API)**

3 **Training cycle
(forward, backward, update)**

Exercise 6-1

- Try different optimizers

**Lecture 7:
Wide and Deep**