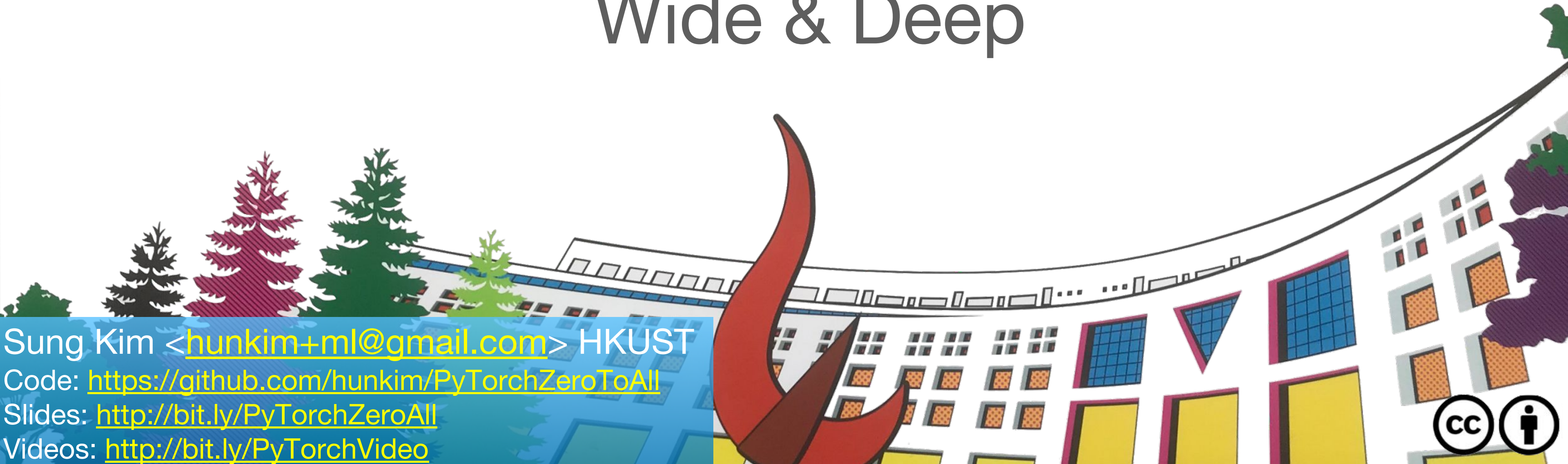
ML/DL for Everyone with PYTORCH

Lecture 7:
Wide & Deep



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

Code: https://github.com/hunkim/PyTorchZeroToAll

Slides: http://bit.ly/PyTorchZeroAll

Videos: http://bit.ly/PyTorchVideo

Call for Comments

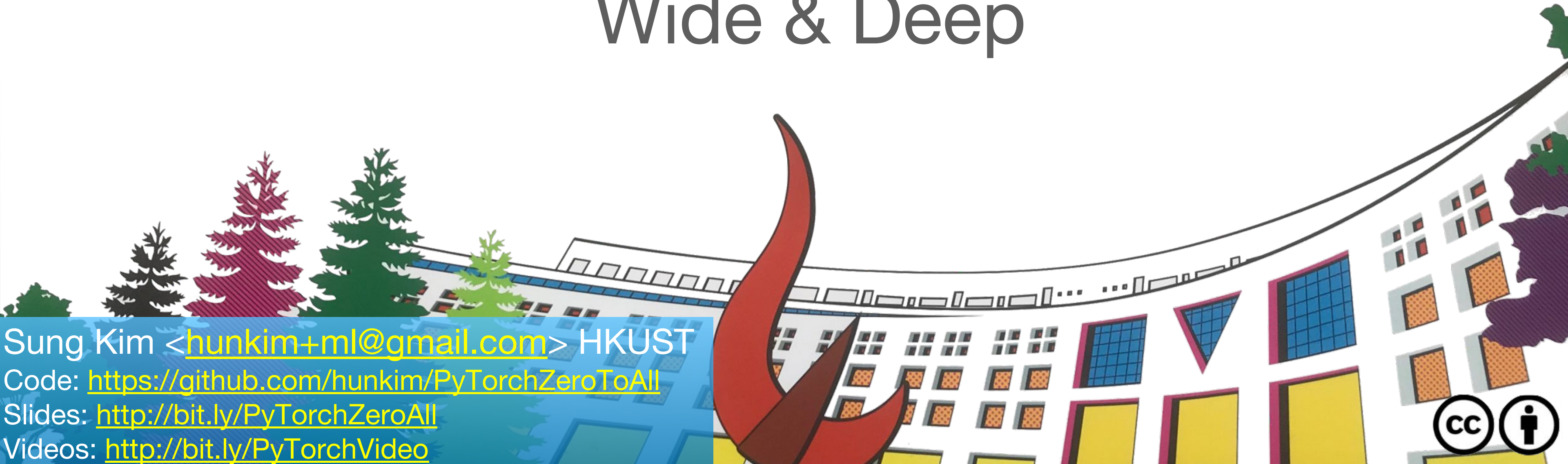
Please feel free to add comments directly on these slides.

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

Picture from http://www.tssablog.org/archives/3280

ML/DL for Everyone with PYTORCH

Lecture 7:
Wide & Deep



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

Code: https://github.com/hunkim/PyTorchZeroToAll

Slides: http://bit.ly/PyTorchZeroAll

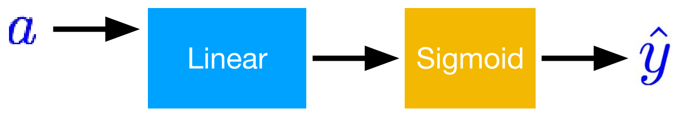
Videos: http://bit.ly/PyTorchVideo

HKUST PHD Program Application

GPA (a)	Admission?
2.1	0
4.2	1
3.1	0
3.3	1

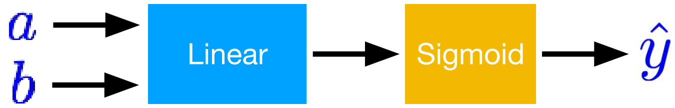
```
x_data =[[2.1],
         [4.2],
         [3.1],
         [3.3]]

y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```



GPA enough?
How about experience and others?

GPA (a)	Experience (b)	Admission?
2.1	0.1	0
4.2	0.8	1
3.1	0.9	0
3.3	0.2	1



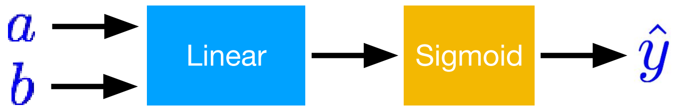

```
x_data =[[2.1, 0.1],
         [4.2, 0.8],
         [3.1, 0.9],
         [3.3, 0.2]]
```

```
y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```

Matrix Multiplication

```
x_data =[[2.1, 0.1],
         [4.2, 0.8],
         [3.1, 0.9],
         [3.3, 0.2]]
```

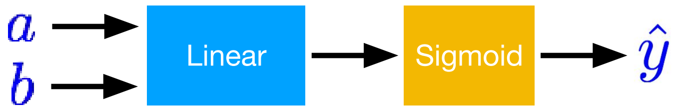
```
y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```



Matrix Multiplication

```
x_data =[[2.1, 0.1],
         [4.2, 0.8],
         [3.1, 0.9],
         [3.3, 0.2]]
```

```
y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```



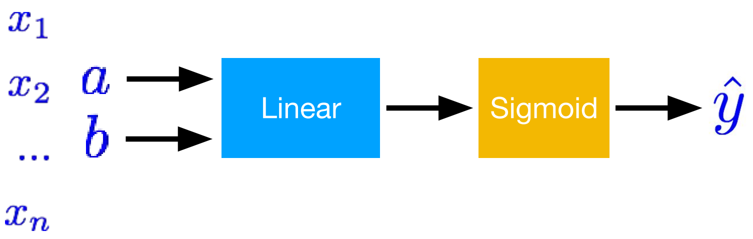
$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w \in \mathbb{R}^{2 \times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\cdot \\ y_n \end{bmatrix}}_{y \in \mathbb{R}^{N \times 1}}$$

Matrix Multiplication

```
x_data =[[2.1, 0.1],
         [4.2, 0.8],
         [3.1, 0.9],
         [3.3, 0.2]]
```

```
y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```

$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x\in\mathbb{R}^{N\times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w\in\mathbb{R}^{2\times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\,\cdot \\ y_n \end{bmatrix}}_{y\in\mathbb{R}^{N\times 1}}$$

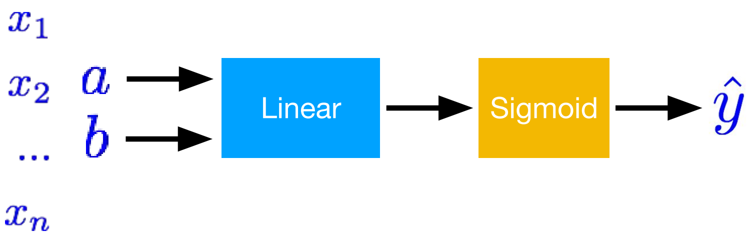


$$XW = \hat{Y}$$

Matrix Multiplication

```
x_data =[[2.1, 0.1],
         [4.2, 0.8],
         [3.1, 0.9],
         [3.3, 0.2]]
```

```
y_data =[[0.0],
         [1.0],
         [0.0],
         [1.0]]
```

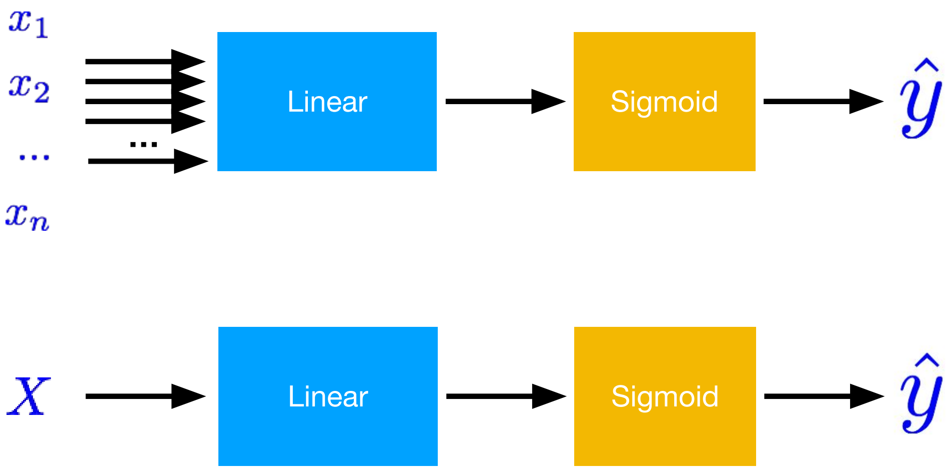


$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w \in \mathbb{R}^{2 \times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\cdot \\ y_n \end{bmatrix}}_{y \in \mathbb{R}^{N \times 1}}$$

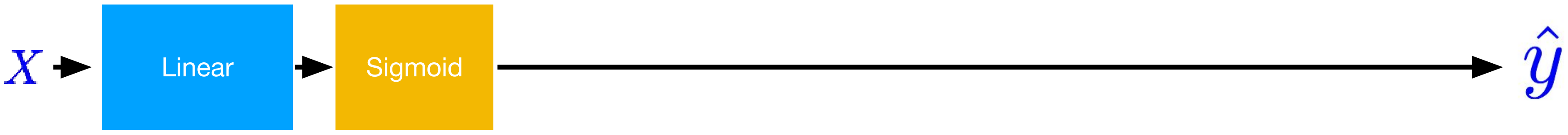
$$XW = \hat{Y}$$

```
linear = torch.nn.Linear(2, 1)
y_prd = linear(x_data)
```

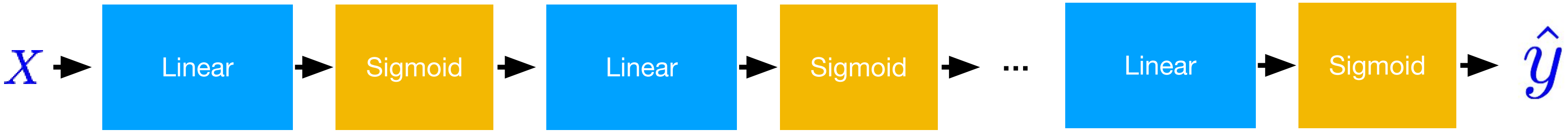
Go Wide!



Go Deep!



Go Deep!



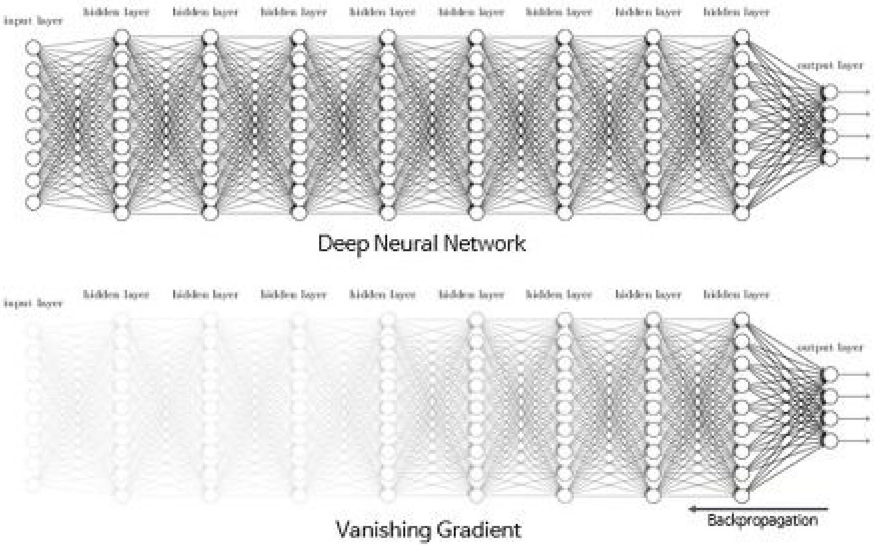

```
sigmoid = torch.nn.Sigmoid()

l1 = torch.nn.Linear(2, 4)
l2 = torch.nn.Linear(4, 3)
l3 = torch.nn.Linear(3, 1)

out1     = sigmoid(l1(x_data))
out2     = sigmoid(l2(out1))
y_pred = sigmoid(l3(out2))
```

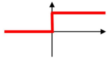
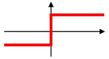
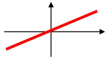
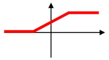
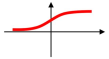
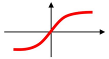
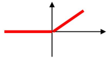
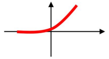
Sigmoid Activation Functions

Sigmoid: Vanishing Gradient Problem



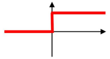
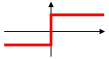
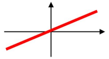
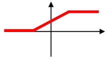
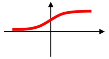
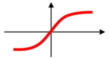
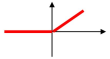
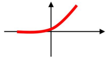
http://excelsior-cjh.tistory.com/entry/RNN-LSTMLong-Short-Term-Memory-networks

Activation Functions

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/

Activation Functions

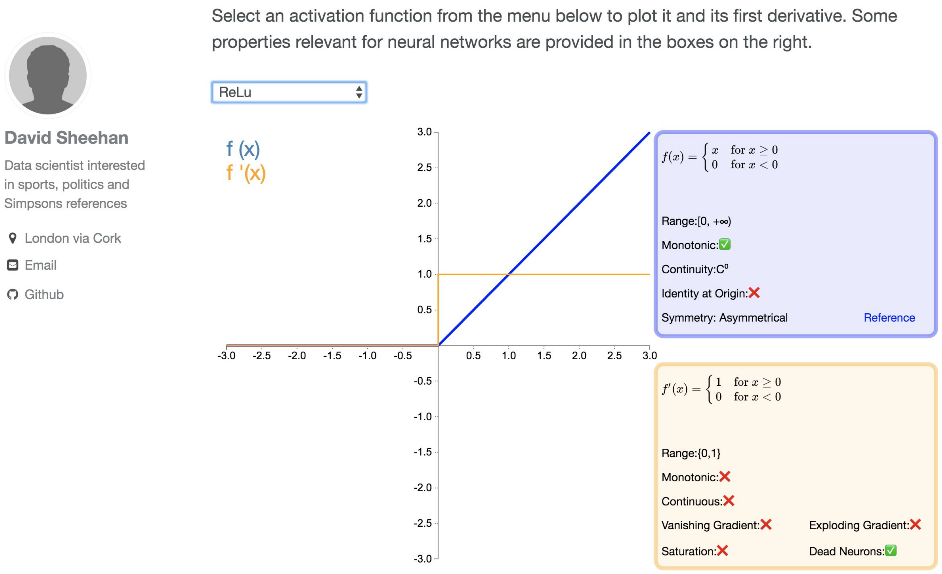
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/

Non-linear Activations

- ReLU
- ReLU6
- ELU
- SELU
- PReLU
- LeakyReLU
- Threshold
- Hardtanh
- Sigmoid
- Tanh
- LogSigmoid
- Softplus
- Softshrink
- Softsign
- Tanhshrink
- Softmin
- Softmax
- Softmax2d
- LogSoftmax

Many Activation Functions



https://dashee87.github.io/data%20science/deep%20learning/visualising-activation-functions-in-neural-networks/

Classifying Diabetes

-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

```
xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)

x_data = Variable(torch.from_numpy(xy[:, 0:-1]))
y_data = Variable(torch.from_numpy(xy[:, [-1]]))

print(x_data.data.shape) # torch.Size([759, 8])
print(y_data.data.shape) # torch.Size([759, 1])
```

Wide & Deep

```python
class Model(torch.nn.Module):

    def __init__(self):
        """
        In the constructor we instantiate two nn.Linear module
        """
        super(Model, self).__init__()
        self.l1 = torch.nn.Linear(8, 6)
        self.l2 = torch.nn.Linear(6, 4)
        self.l3 = torch.nn.Linear(4, 1)

        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        out1 = self.sigmoid(self.l1(x))
        out2 = self.sigmoid(self.l2(out1))
        y_pred = self.sigmoid(self.l3(out2))
        return y_pred
```

Classifying Diabetes

```python
xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = Variable(torch.from_numpy(xy[:, 0:-1]))
y_data = Variable(torch.from_numpy(xy[:, [-1]]))

class Model(torch.nn.Module):
    def __init__(self):
        """
        In the constructor we instantiate two nn.Linear module
        """
        super(Model, self).__init__()
        self.l1 = torch.nn.Linear(8, 6)
        self.l2 = torch.nn.Linear(6, 4)
        self.l3 = torch.nn.Linear(4, 1)

        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Variables.
        """
        out1 = self.sigmoid(self.l1(x))
        out2 = self.sigmoid(self.l2(out1))
        y_pred = self.sigmoid(self.l3(out2))
        return y_pred

# our model
model = Model()

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

# Training loop
for epoch in range(100):
        # 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()
```

1 Design your model using class

2 Construct loss and optimizer (select from PyTorch API)

3 Training cycle (forward, backward, update)

Exercise 7-1

- Classifying Diabetes with deep nets
  - More than 10 layers
- Find other classification datasets
  - Try with deep network
- Try different activation functions
  Sigmoid to something else

WHAT
NEXT

Lecture 8: DataLoader