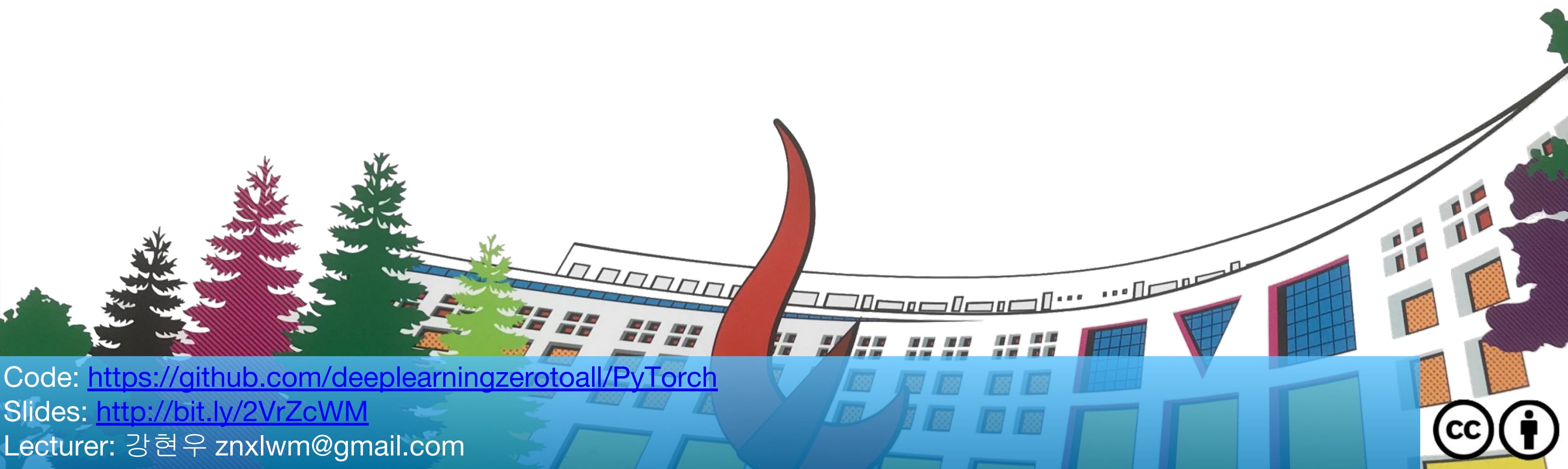
ML/DL for Everyone Season2

Weight initialization



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

Weight initialization

- Why good initialization?
- RBM / DBN
- Xavier / He initialization
- Code: mnist_nn_xavier
- Code: mnist_nn_deep

Geoffrey Hinton’s summary of findings up to today

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity.

https://medium.com/@andreykurenkov/a-brief-history-of-neural-nets-and-deep-learning-part-4-61be90639182

Why good initialization?

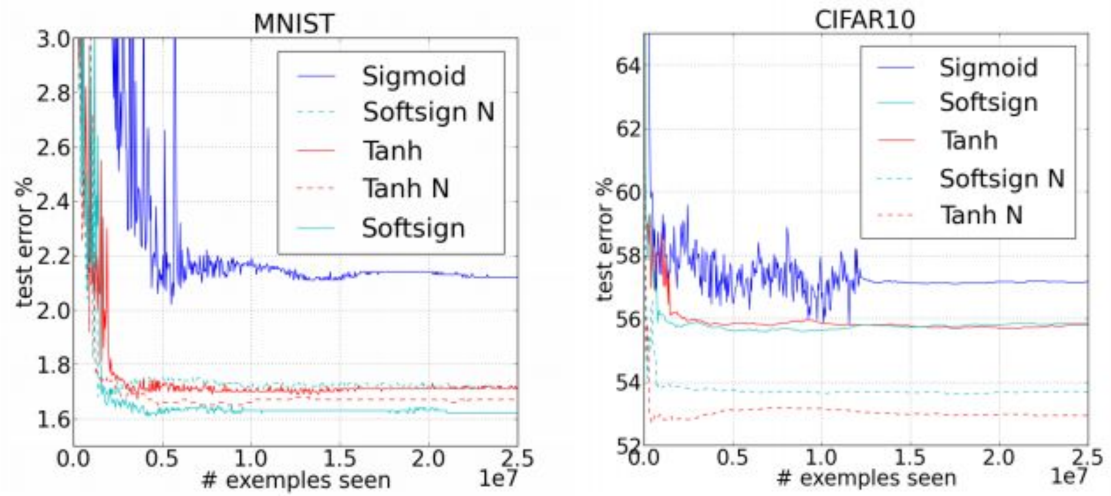


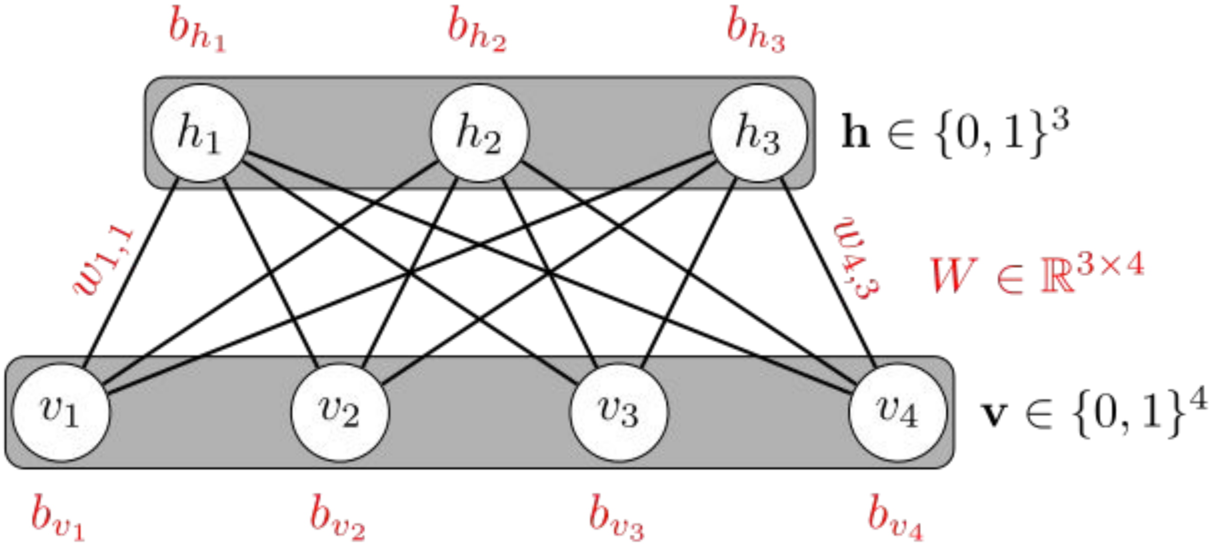
Figure 12: *Test error curves during training on MNIST and CIFAR10, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Need to set the initial weight values wisely

- Not all 0’s
- Challenging issue
- Hinton et al. (2006) “A Fast Learning Algorithm for Deep Belief Nets” - Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine



- Restricted: no connections within a layer
- KL divergence: compare actual to recreation

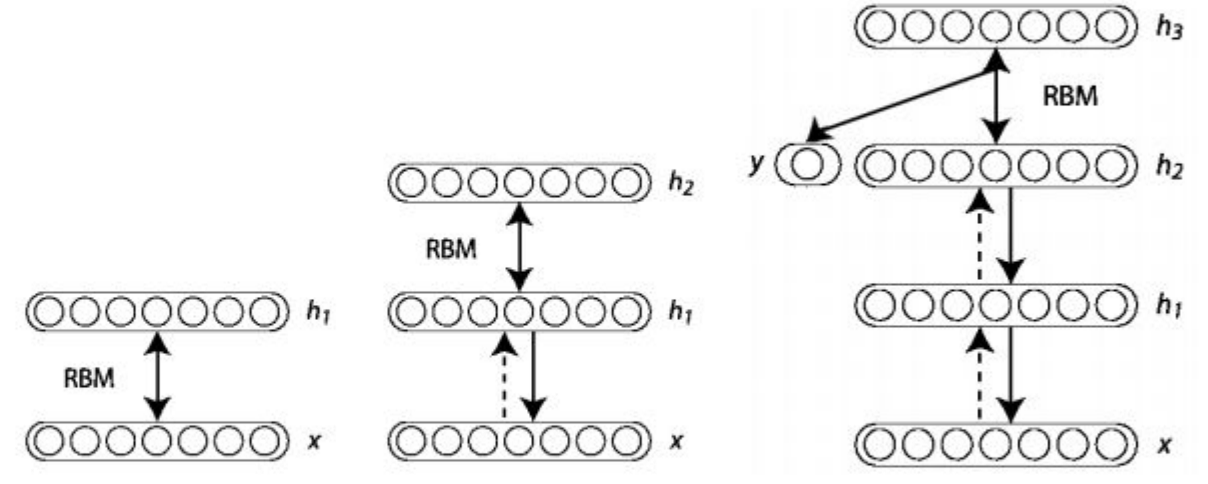
https://en.wikipedia.org/wiki/File:Restricted-boltzmann-machine.svg

How can we use RBM to initialize weights?

- Apply the RBM idea on adjacent two layers as a pre-training step
- Continue the first process to all layers
- This will set weights
- Example: Deep Belief Network
  - Weight initialized by RBM

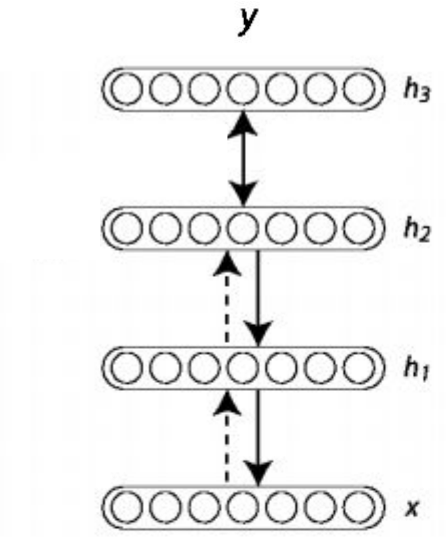
Deep Belief Network

Pre-training



(a) RBM for $\mathbf{x}$ (b) RBM for $\mathbf{h}^1$ (c) RBM for $\mathbf{h}^2$ and y

Fine-tuning



http://www.dmi.usherb.ca/~larocheh/publications/deep-nets-icml-07.pdf

Xavier / He initialization

- No need to use complicated RBM for weight initializations
- Simple methods are OK

  - **Xavier initialization**: X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010
  - **He initialization**: K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

Xavier / He initialization

- Xavier Normal initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

- He Normal initialization

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

- Xavier Uniform initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, +\sqrt{\frac{6}{n_{in} + n_{out}}})$$

- He Uniform initialization

$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}})$$

Code: mnist_nn_xavier

```
def xavier_uniform_(tensor, gain=1):
    .. math::
        a = \text{gain} \times \sqrt{\frac{6}{\text{fan_in} + \text{fan_out}}}

    Also known as Glorot initialization.

    Args:
        tensor: an n-dimensional `torch.Tensor`
        gain: an optional scaling factor

    Examples:
        >>> w = torch.empty(3, 5)
        >>> nn.init.xavier_uniform_(w, gain=nn.init.calculate_gain('relu'))
    """
    fan_in, fan_out = _calculate_fan_in_and_fan_out(tensor)
    std = gain * math.sqrt(2.0 / (fan_in + fan_out))
    a = math.sqrt(3.0) * std  # Calculate uniform bounds from standard deviation
    with torch.no_grad():
        return tensor.uniform_(-a, a)
```

Code: mnist_nn_xavier

```
...

# nn layers
linear1 = torch.nn.Linear(784, 256, bias=True)
linear2 = torch.nn.Linear(256, 256, bias=True)
linear3 = torch.nn.Linear(256, 10, bias=True)
relu = torch.nn.ReLU()

# xavier initialization
torch.nn.init.xavier_uniform_(linear1.weight)
torch.nn.init.xavier_uniform_(linear2.weight)
torch.nn.init.xavier_uniform_(linear3.weight)
```

```
Parameter containing:
tensor([[-0.0215, -0.0894,  0.0598,  ...,  0.0200,  0.0203,  0.1212],
        [ 0.0078,  0.1378,  0.0920,  ...,  0.0975,  0.1458, -0.0302],
        [ 0.1270, -0.1296,  0.1049,  ...,  0.0124,  0.1173, -0.0901],
        ...,
        [ 0.0661, -0.1025,  0.1437,  ...,  0.0784,  0.0977, -0.0396],
        [ 0.0430, -0.1274, -0.0134,  ..., -0.0582,  0.1201,  0.1479],
        [-0.1433,  0.0200, -0.0568,  ...,  0.0787,  0.0428, -0.0036]],
       requires_grad=True)
```

```
Epoch: 0001 cost = 0.249897048
Epoch: 0002 cost = 0.094330102
Epoch: 0003 cost = 0.061055195
Epoch: 0004 cost = 0.042816643
Epoch: 0005 cost = 0.032796543
Epoch: 0006 cost = 0.024419624
Epoch: 0007 cost = 0.020511184
Epoch: 0008 cost = 0.018132176
Epoch: 0009 cost = 0.015536907
Epoch: 0010 cost = 0.016846467
Epoch: 0011 cost = 0.012203062
Epoch: 0012 cost = 0.012871196
Epoch: 0013 cost = 0.011348661
Epoch: 0014 cost = 0.010990168
Epoch: 0015 cost = 0.006201488
Learning finished
Accuracy: 0.9804999828338623
```

Code: mnist_nn_deep

```
...

# nn layers
linear1 = torch.nn.Linear(784, 512, bias=True)
linear2 = torch.nn.Linear(512, 512, bias=True)
linear3 = torch.nn.Linear(512, 512, bias=True)
linear4 = torch.nn.Linear(512, 512, bias=True)
linear5 = torch.nn.Linear(512, 10, bias=True)
relu = torch.nn.ReLU()

# xavier initialization
torch.nn.init.xavier_uniform_(linear1.weight)
torch.nn.init.xavier_uniform_(linear2.weight)
torch.nn.init.xavier_uniform_(linear3.weight)
torch.nn.init.xavier_uniform_(linear4.weight)
torch.nn.init.xavier_uniform_(linear5.weight)

...
```

```
Epoch: 0001 cost = 0.283860594
Epoch: 0002 cost = 0.089265838
Epoch: 0003 cost = 0.056718789
Epoch: 0004 cost = 0.041850876
Epoch: 0005 cost = 0.030926639
Epoch: 0006 cost = 0.024389934
Epoch: 0007 cost = 0.021937676
Epoch: 0008 cost = 0.019161038
Epoch: 0009 cost = 0.016852187
Epoch: 0010 cost = 0.014415207
Epoch: 0011 cost = 0.013022121
Epoch: 0012 cost = 0.010289547
Epoch: 0013 cost = 0.015175694
Epoch: 0014 cost = 0.008412631
Epoch: 0015 cost = 0.008151450
Learning finished
Accuracy: 0.9818999767303467
```

What's Next?

- Dropout