Deep Neural Networks

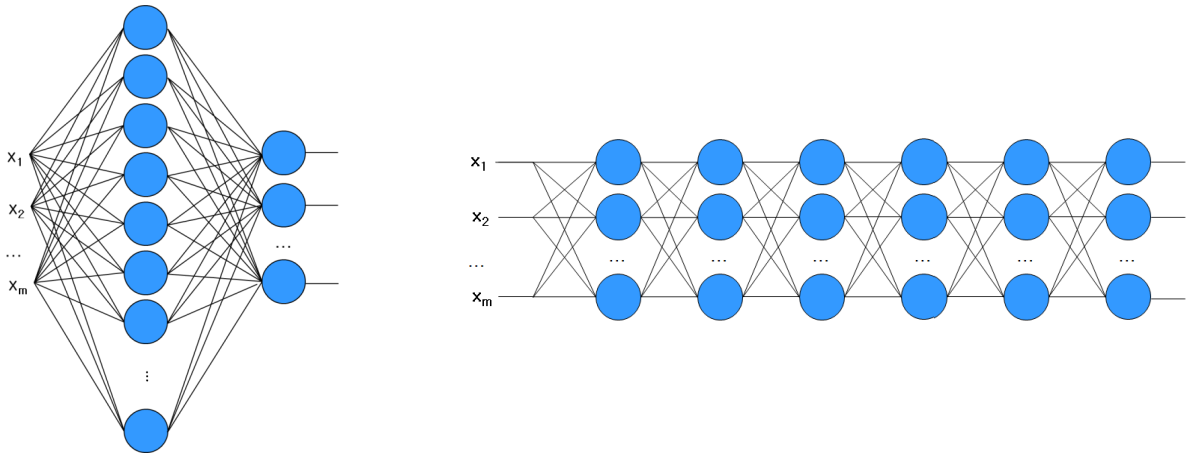
Content

- **Why Deep Neural Networks**
- **Vanishing Gradient & Activation Functions**
- **Drop-Out**
- **Batch Normalization**

Why Deep Neural Networks

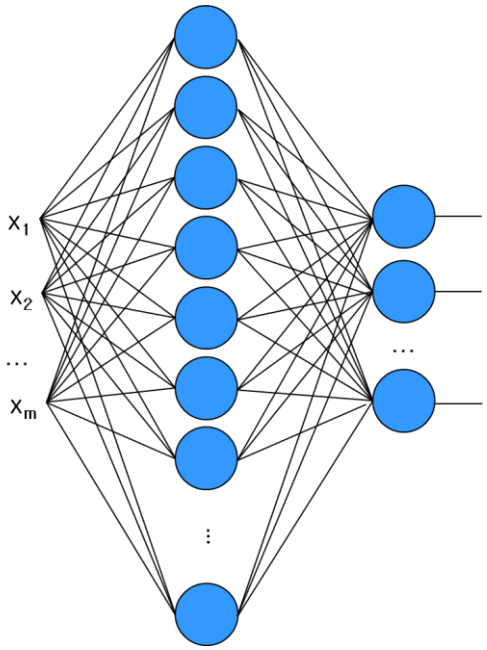
How Different

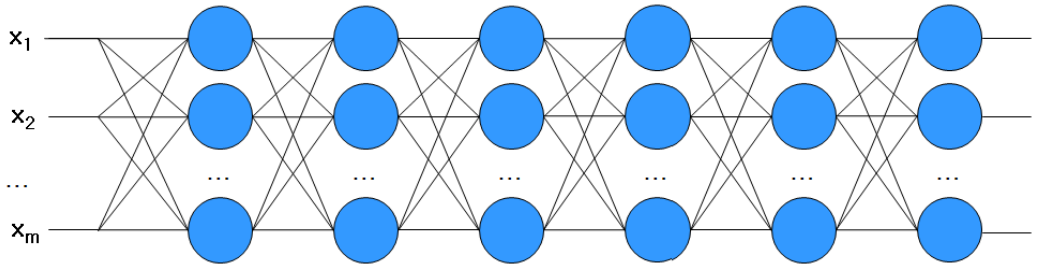
- **Shallow Network vs Deep Network**
  - Theoretically, it can mimic any functions, but the number of nodes can exponentially increase in a shallow network
  - In order to avoid the exponential explosion, we can adopt deep networks (many layers)
  - Deep networks have very strong power but cause many problems!!



Hard to Handle

- **Pros and Cons of Deep Networks**
  - Pros: Powers from deep structure
  - Cons: Hard to optimize, Overfitting, Internal covariate shift

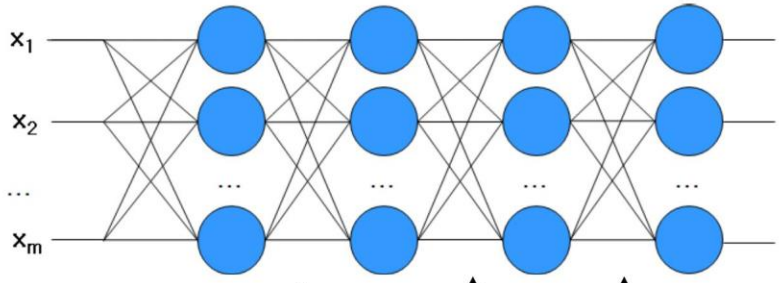




Gradient Vanishing & Activation Functions

Gradient Vanishing & Exploding

- **Gradient is easy to vanish or explode**
  - To many terms are multiplied.
  - If some are small numbers, gradient becomes very small.
  - If some are large numbers, gradient becomes very large.



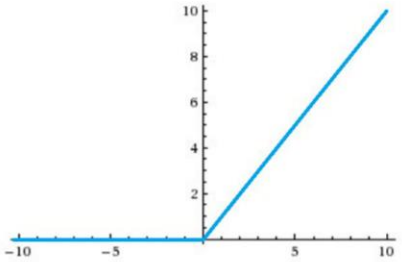
$$\frac{\partial E_n}{\partial w_{kj}} = -(t_{nk} - o_{nk}) o_{nk} (1 - o_{nk}) h_{nj}$$

$$\frac{\partial E_n}{\partial w_{ji}} = -h_{nj}(1 - h_{nj}) x_{ni} \sum_{k=1}^{m} w_{kj} (t_{nk} - o_{nk}) o_{nk} (1 - o_{nk})$$

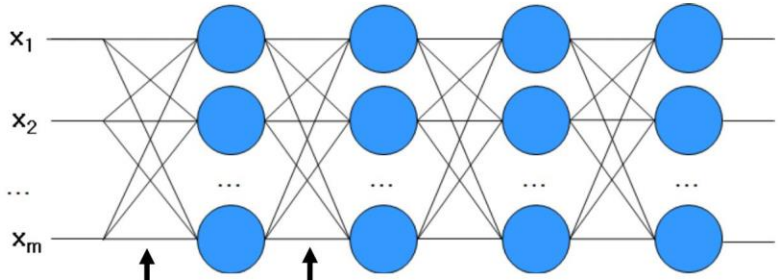
$$\frac{\partial E}{\partial w_{ip}} = \left(\sum_{j=1}^{J} \left(\sum_{k=1}^{K} -(t_n - o_{nk}) o_{nk} (1 - o_{nk}) w_{kj} \right) h_{nj} (1 - h_{nj}) w_{ji} \right) h_{ni} (1 - h_{ni}) h_{np}$$

Activation Function

- **Using another functions instead of sigmoid**
  - Rectified Linear Unit (ReLU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

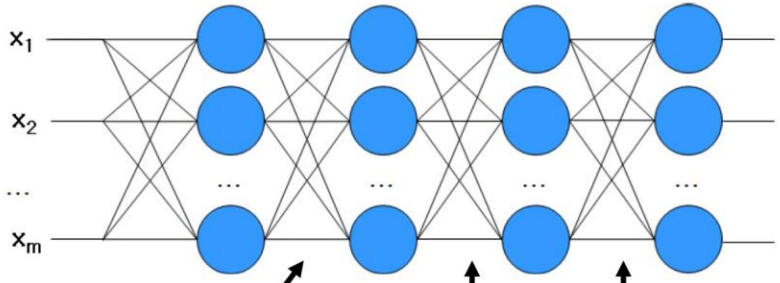


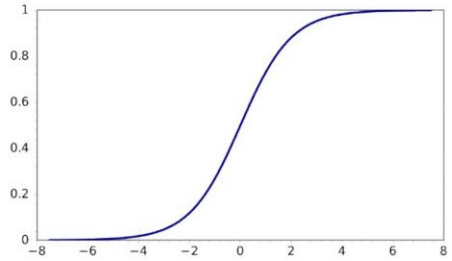
$$\frac{\partial E_n}{\partial w_{ji}} = \text{Some formular with 3 mulitiplications of } \frac{\partial f}{\partial w}$$

$$\frac{\partial E_n}{\partial w_{hg}} = \text{Some formular with 4 mulitiplications of } \frac{\partial f}{\partial w}$$

Activation Function

- **Vanishing Gradient**
  - The major terms are the derivatives of the activation function





$$\frac{\partial \text{Sigmoid}}{\partial w} \leq \frac{1}{4}$$

$$\frac{\partial E_n}{\partial w_{kj}} = -(t_{nk} - o_{nk}) o_{nk}(1 - o_{nk}) h_{nj}$$

$$\frac{\partial E_n}{\partial w_{ji}} = -h_{nj}(1 - h_{nj}) x_{ni} \sum_{k=1}^{m} w_{kj}(t_{nk} - o_{nk}) o_{nk}(1 - o_{nk})$$

$$\frac{\partial E}{\partial w_{ip}} = \left(\sum_{j=1}^{J} \left(\sum_{k=1}^{K} -(t_n - o_{nk}) o_{nk}(1 - o_{nk}) w_{kj} \right) h_{nj}(1 - h_{nj}) w_{ji} \right) h_{ni}(1 - h_{ni}) h_{np}$$

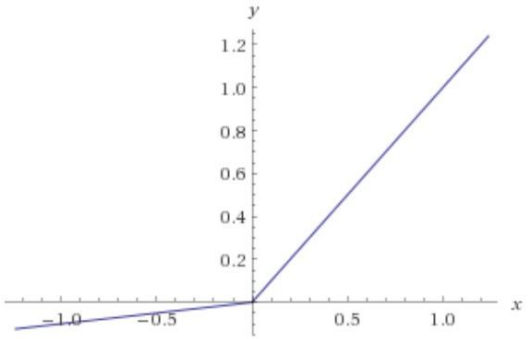
Activation Function

- **Advantage**
  - No vanishing gradient problems.
    - Reduce the number of terms to be muliplied
  - Sparse activation
    - In a randomly initialized network, only about 50% of hidden units are activated
  - Fast computation:
    - 6 times faster than sigmoid function
- **Disadvantage**
  - Knockout Problem

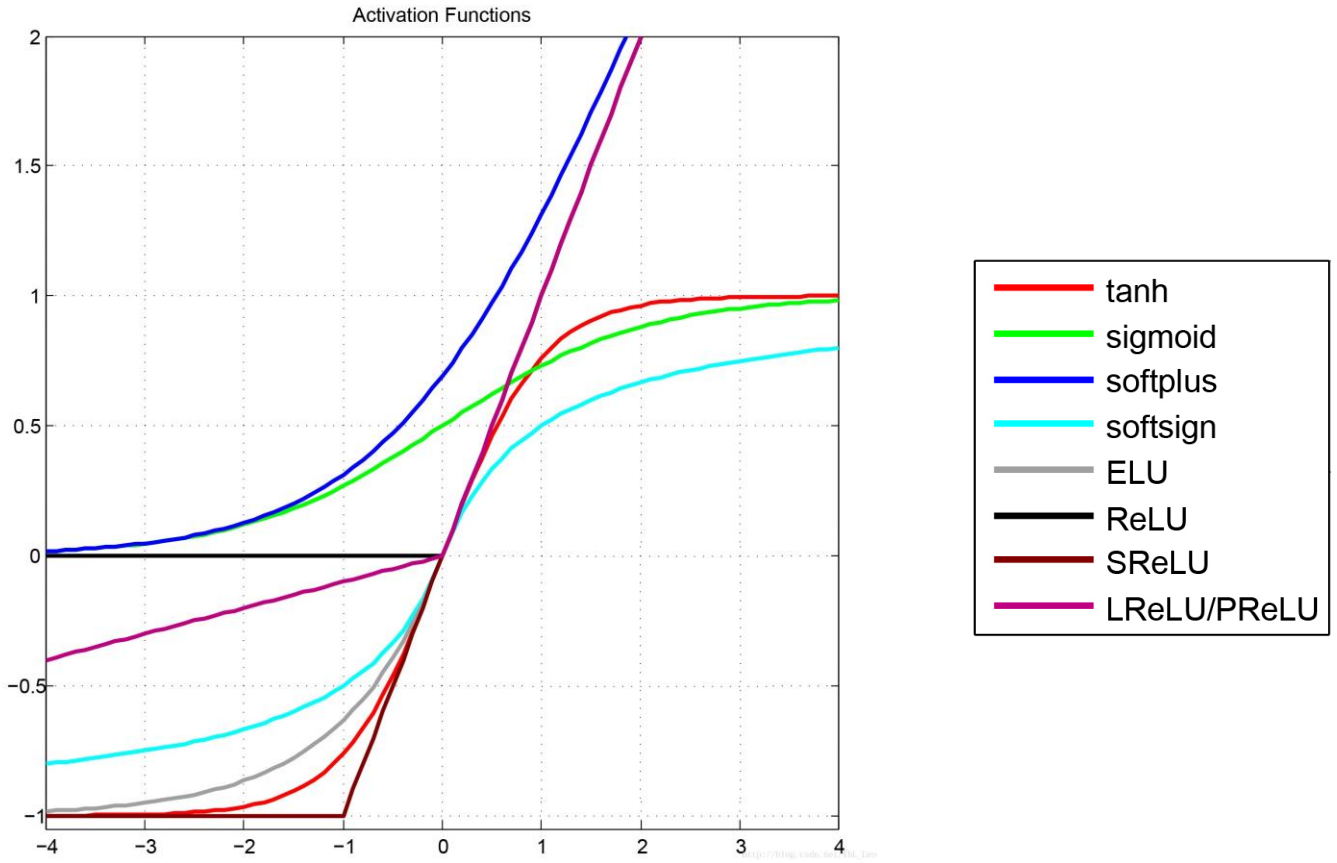
Activation Function

- **You may use another**
  - Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$



Other Activation Functions



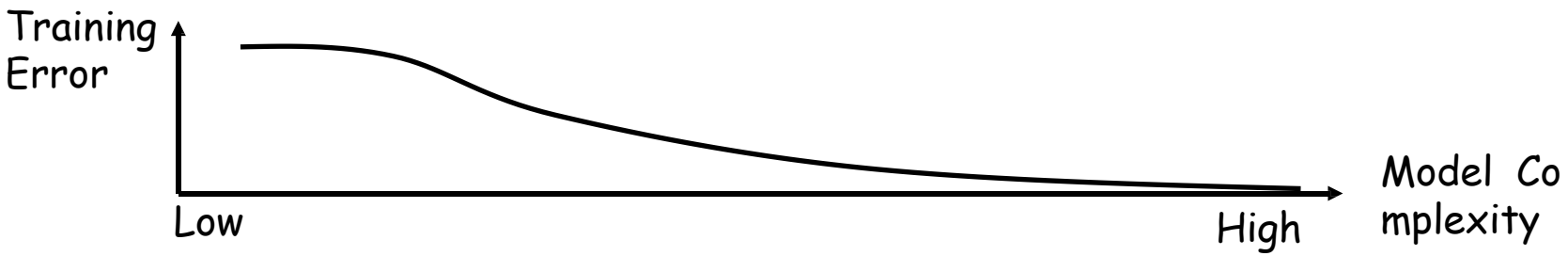
Activation Function

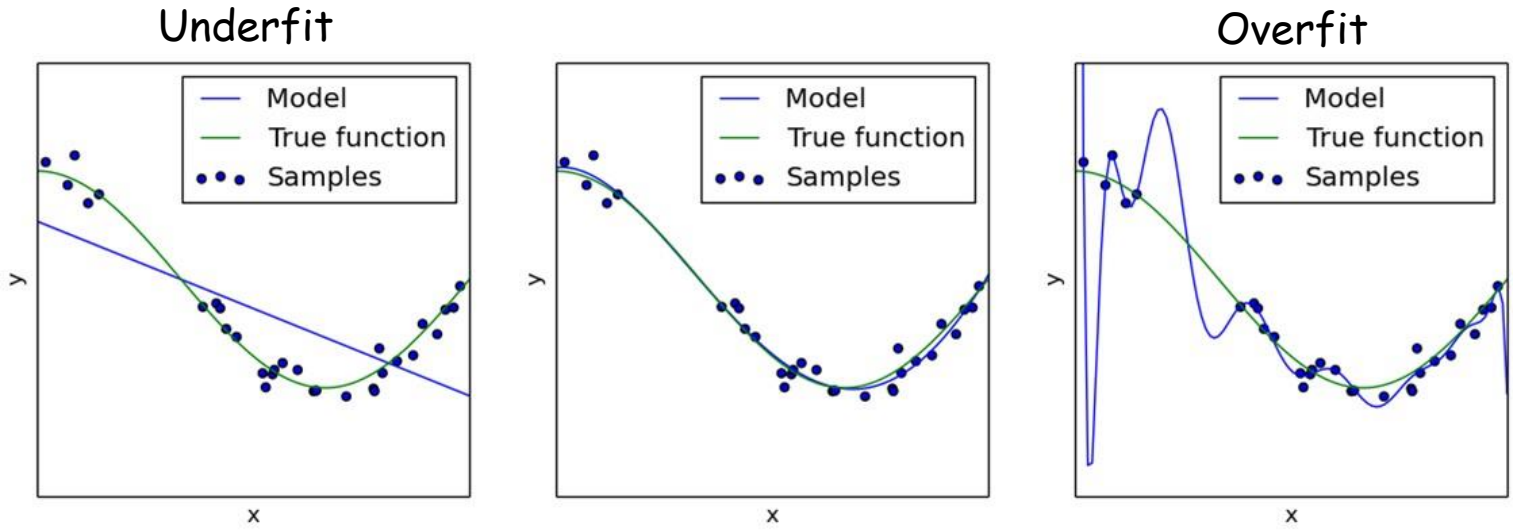
- **Summary**
  - Sigmoid functions and their combinations generally work better but are sometimes avoided due to the vanishing gradient problem
  - ReLU function is a general activation function and is used in most cases these days
  - If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
  - ReLU function is usually used in the hidden layers
  - As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case R eLU doesn't provide with optimum results

Overfitting & Dropout

Overfitting

- **Overfitting**





Regularization

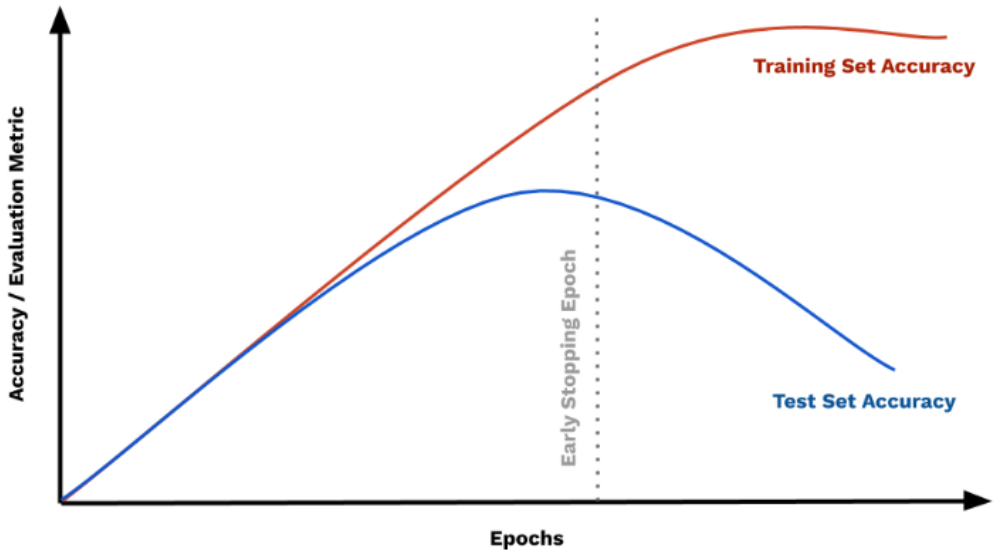
- **What is Regularization**
  - Introducing additional information to prevent over-fitting

- **Approaches**
  - Early stopping, Max norm constraints, Weight decay
    Dropout, DropConnect, Stochastic pooling

Regularization

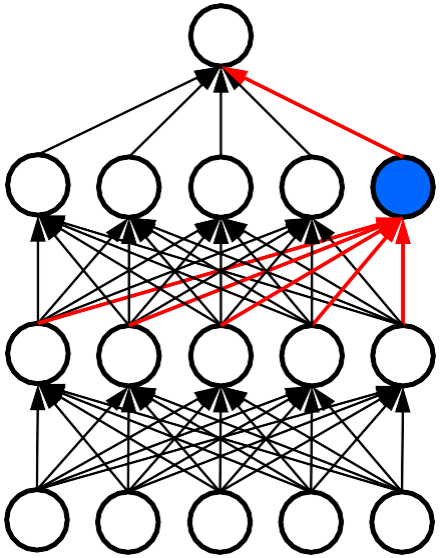
Early Stopping

Stop learning before converging to much
Hard to determine the time to stop
Usually determined by validation dataset



Dropout

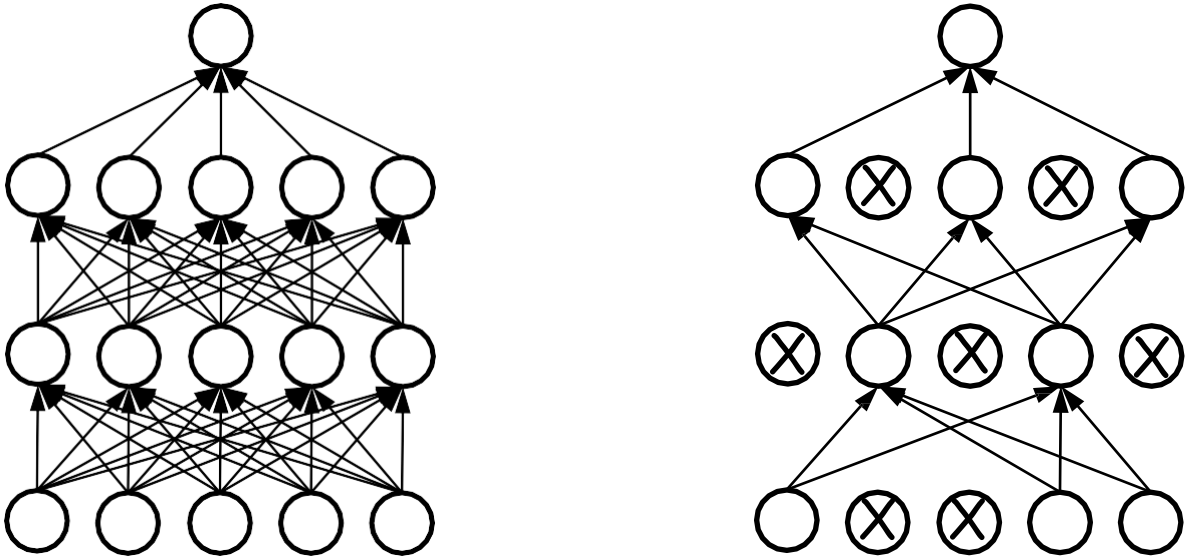
- **In a complex Neural Network**
  - All nodes do not take the same amount of responsibility
    - While training, some nodes are correlated
  - All nodes are not equally trained
    - Some nodes are trained much, but some are not



- If the output of the node is bad, t he connection weight will decrease.
- If connection weight is close to 0, precedent connection weights are hardly trained.

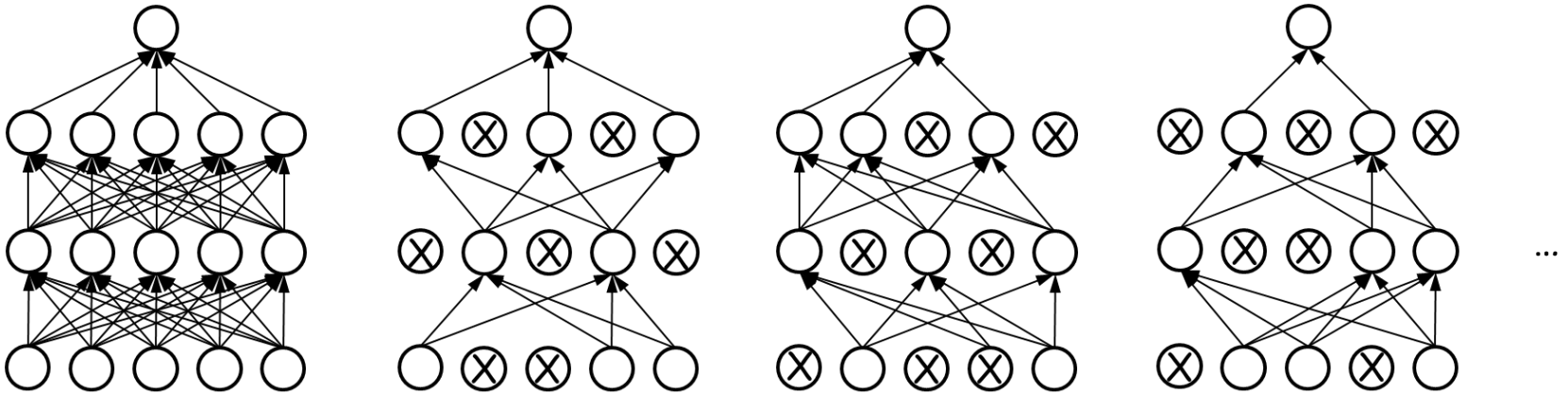
Dropout

- **How can we reduce the structural complexity?**
  - Let's simply remove some nodes, and
  - Train the simplified neural network
  - Hmm??



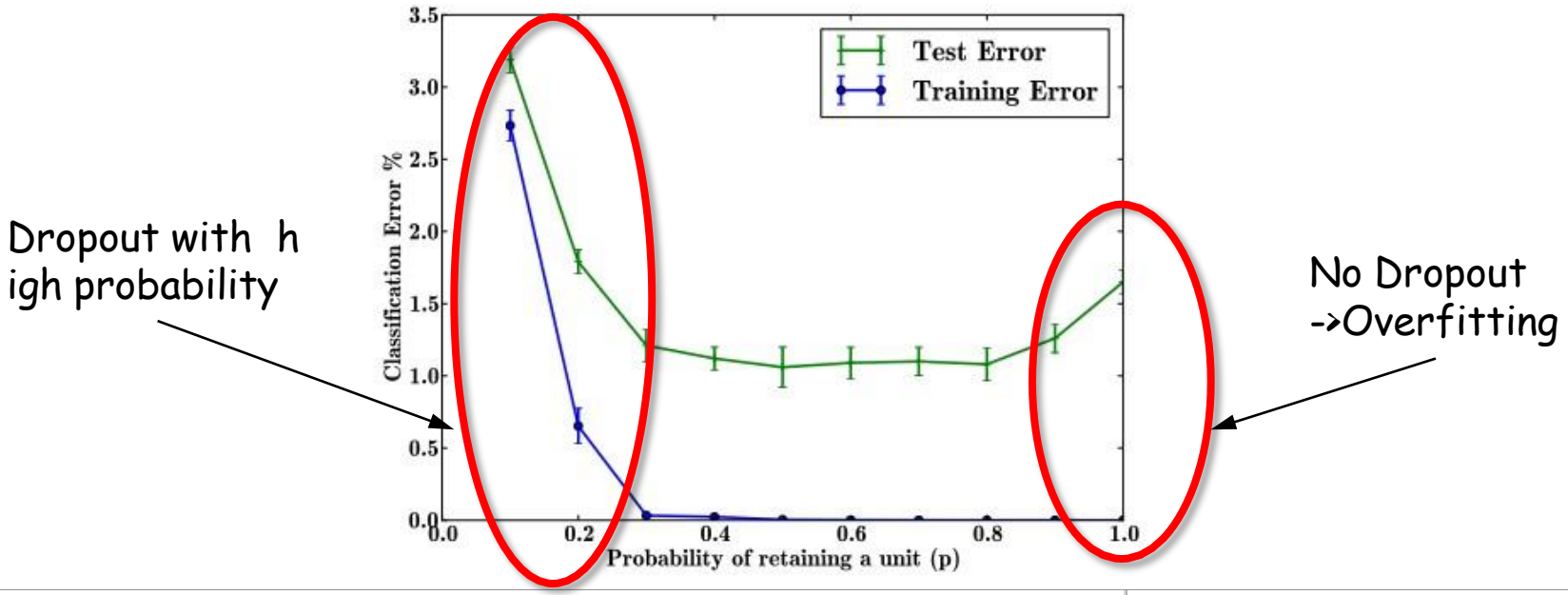
Dropout

- **Do this at every epoch**
  - Randomly choose nodes with a probability of *p*
    - Usually p = 0.5
  - Train the simplified neural network
    - At every epoch, we train different neural network which share connection weight each other



Dropout

- **The effect of the dropout rate p:**
  - An architecture of 784-2048-2048-2048-10 is used on the MNIST dataset.
  - The dropout rate p is changed from small numbers (most units are dropped out) to 1.0 (no dropout).



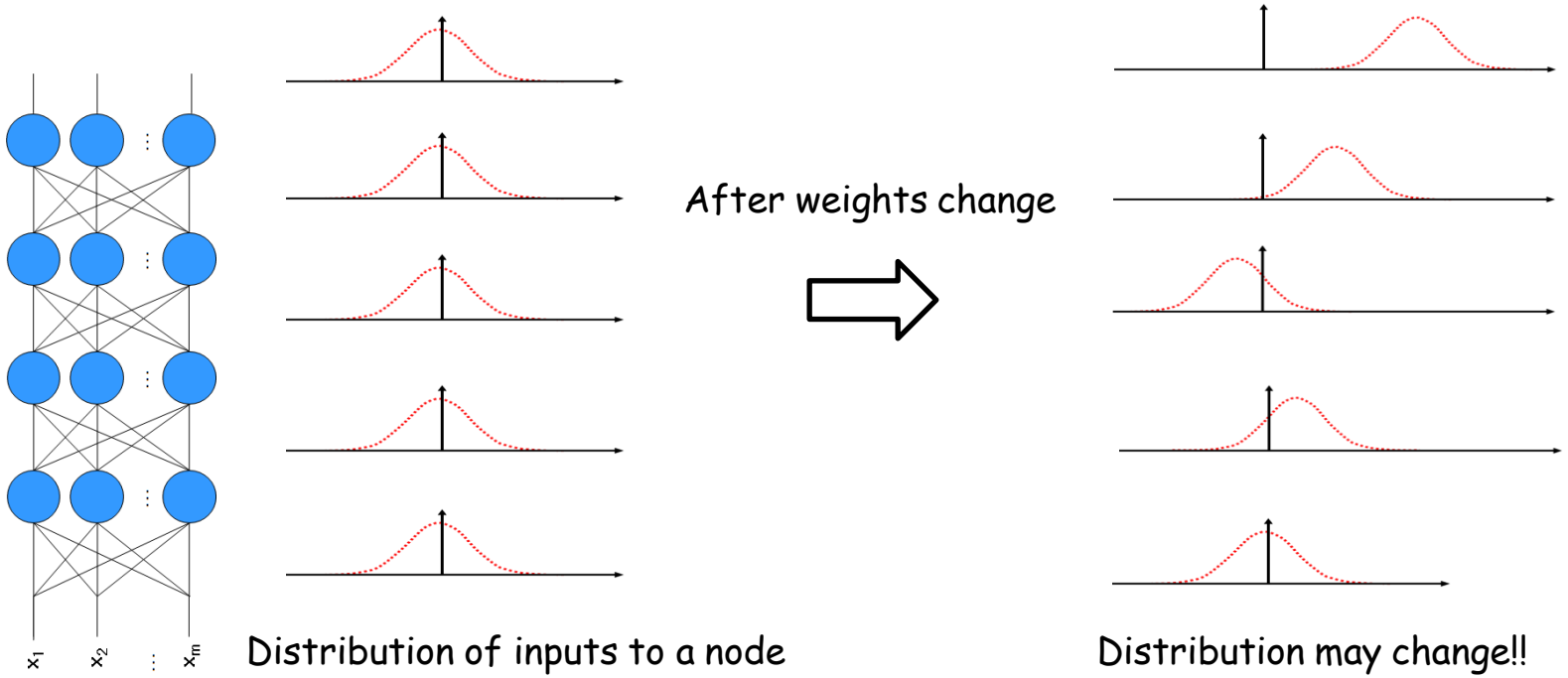
Dropout

- **Summary**
  - Dropout is a very good and fast regularization method.
  - Dropout is a bit slow to train (2-3 times slower than without dropout).
  - If the amount of data is average-large – dropout excels. When data is big enough, dropout does not help much.
  - Dropout achieves better results than former used regularization methods (Weight Decay).

Batch Normalization

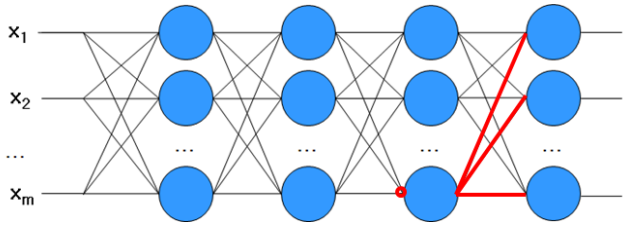
Batch Normalization

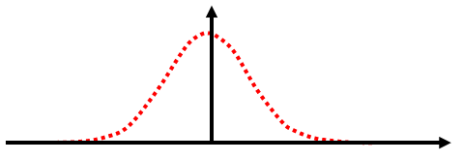
- **Internal Covariate Shift**
  - The distribution of the input to a layer changes if you change the connection weights
  - This change propagates to the upper layers



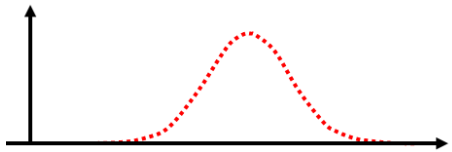
Batch Normalization

- **Internal Covariate Shift**
  - Input distribution of the red node





  - While learning, red connection weights will change based on the input distribution
  - After learning, the whole connection weights changes, which cause the change of the input distribution

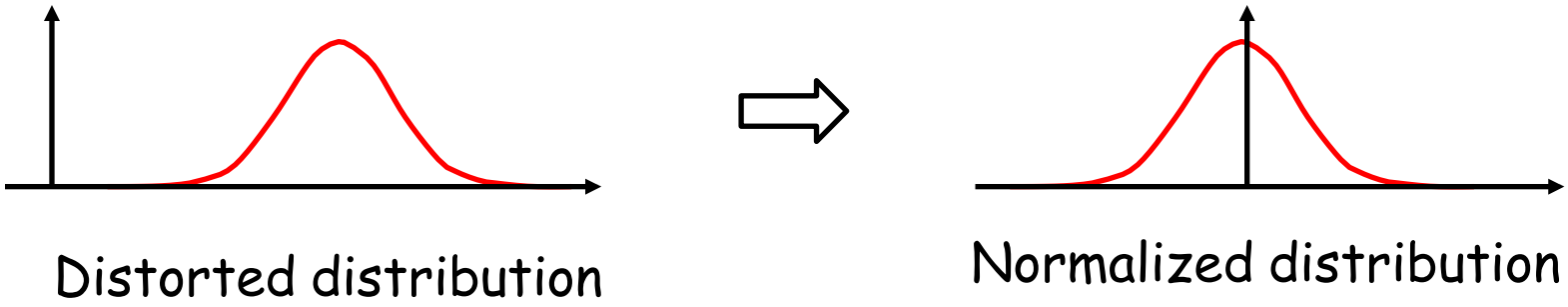


  - The assumption of the learning is broken

Batch Normalization

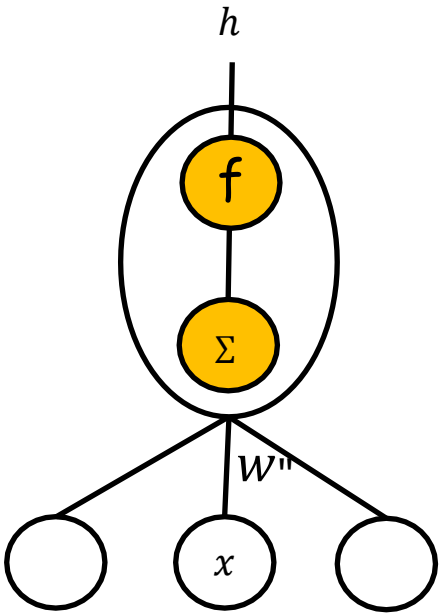
- **Internal Covariate Shift**
  - It disturbs the learning process,
  - Learning is getting slow down

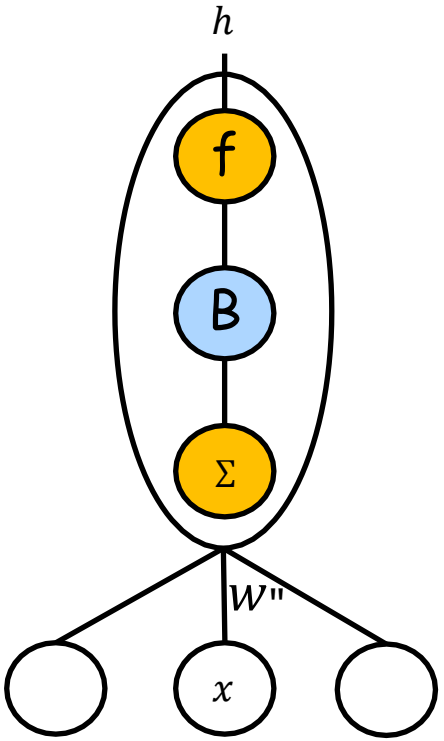
- **What shall we do?**
  - Why don't we normalize the distribution of inputs



Batch Normalization

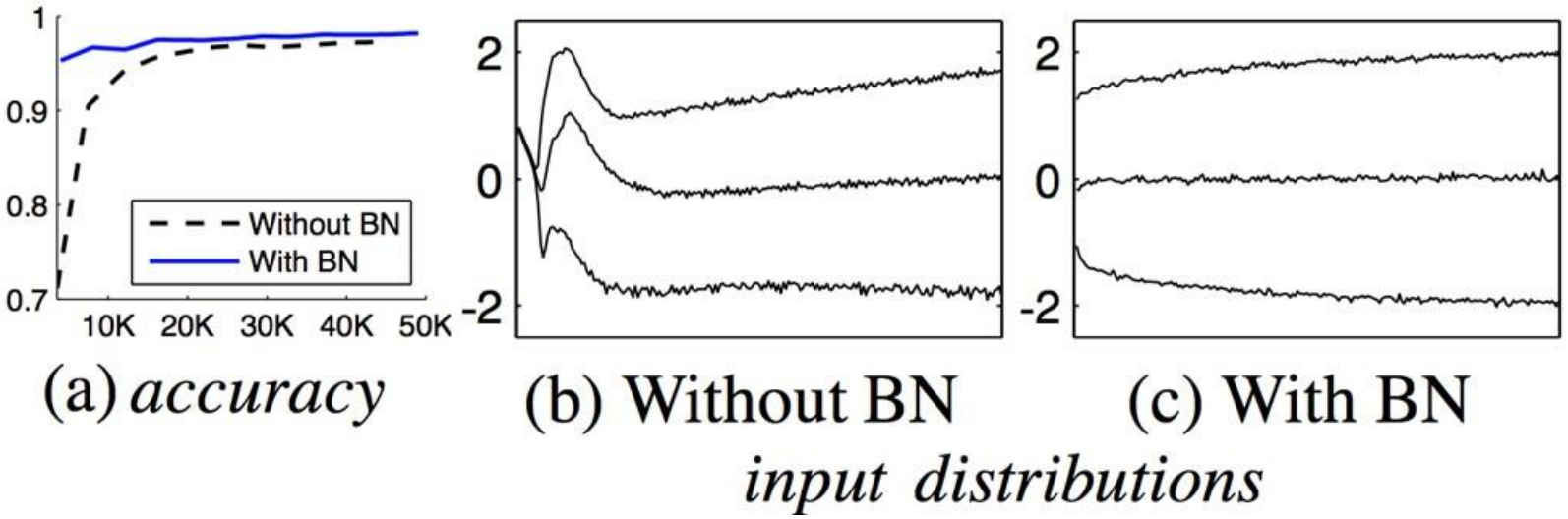
- **For a Single Node**





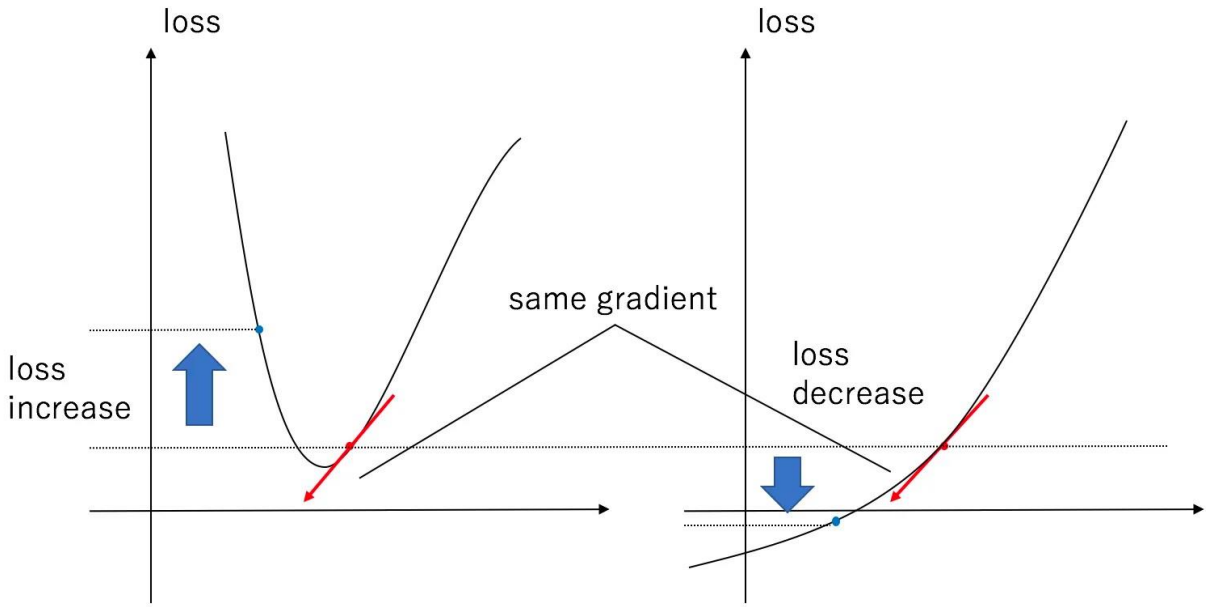
Batch Normalization

- **Performance with BN**



Batch Normalization

- **Advantage**
  - We may have a more smoother loss function. Why?
  - We may use a larger learning rate. Why?



https://arxiv.org/abs/1502.03167
http://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/

Batch Normalization

Advantage

Learning faster: Learning rate can be increased compare to non-batch-normalized version.

Not-Stuck in the saturation mode: Even if ReLU is not used.

Higher accuracy: Flexible on data distribution in every hidden layer

Batch Normalization

- **Disadvantage**
  - Expensive: Memory and time
    - Must keep interim results of all instances in a batch
    - Especially in CNN, usually an image is large
  - Hard to apply when the batch size is small
    - If batches are small, the means and variances cannot approximate the global ones.
  - Hard to apply to recurrent networks
    - It doesn't match to structure of recurrent networks
    - Hard to implement with recurrent networks