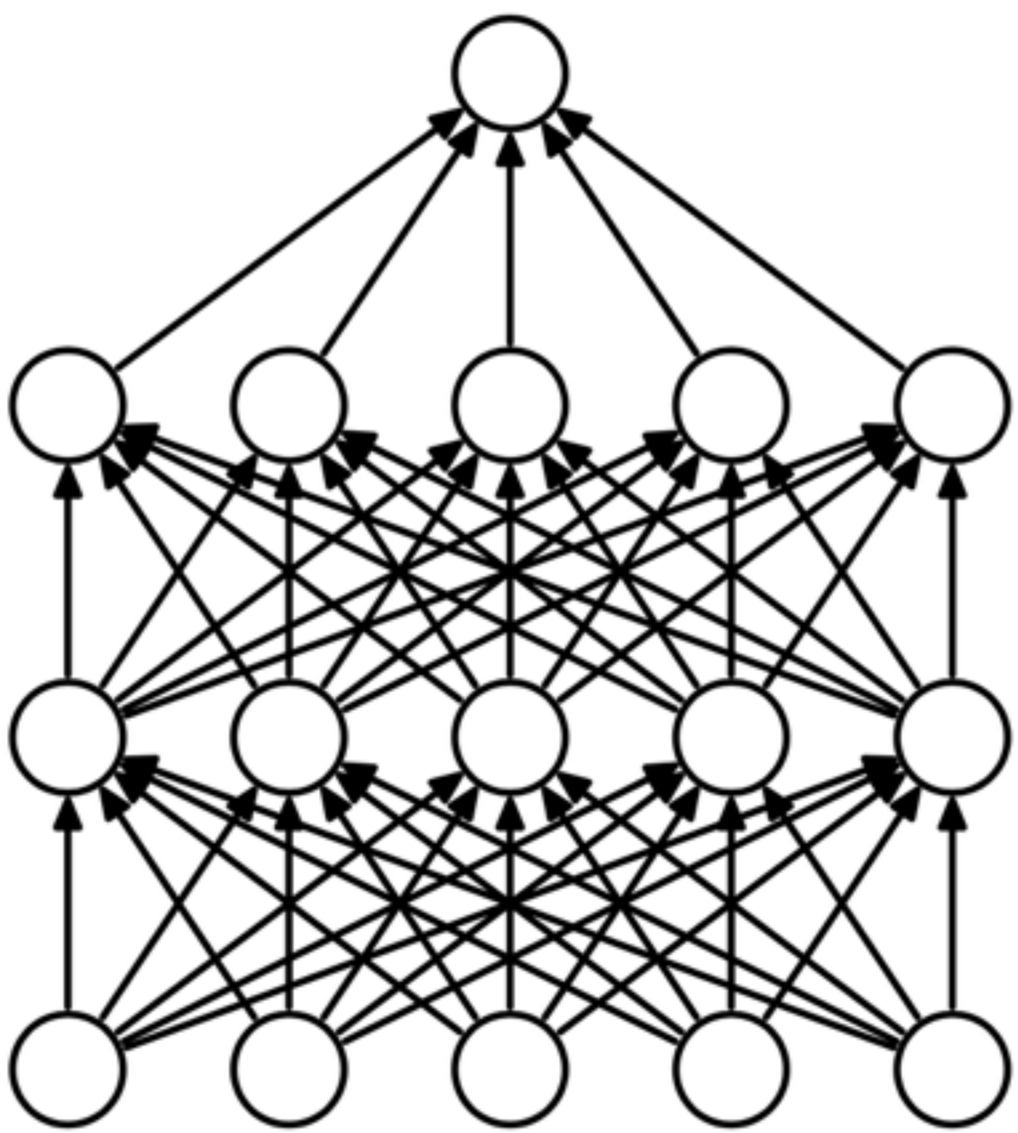
# Training Neural Networks

## Some considerations

Gaurav Kumar
Center for Language and Speech Processing
gkumar@cs.jhu.edu

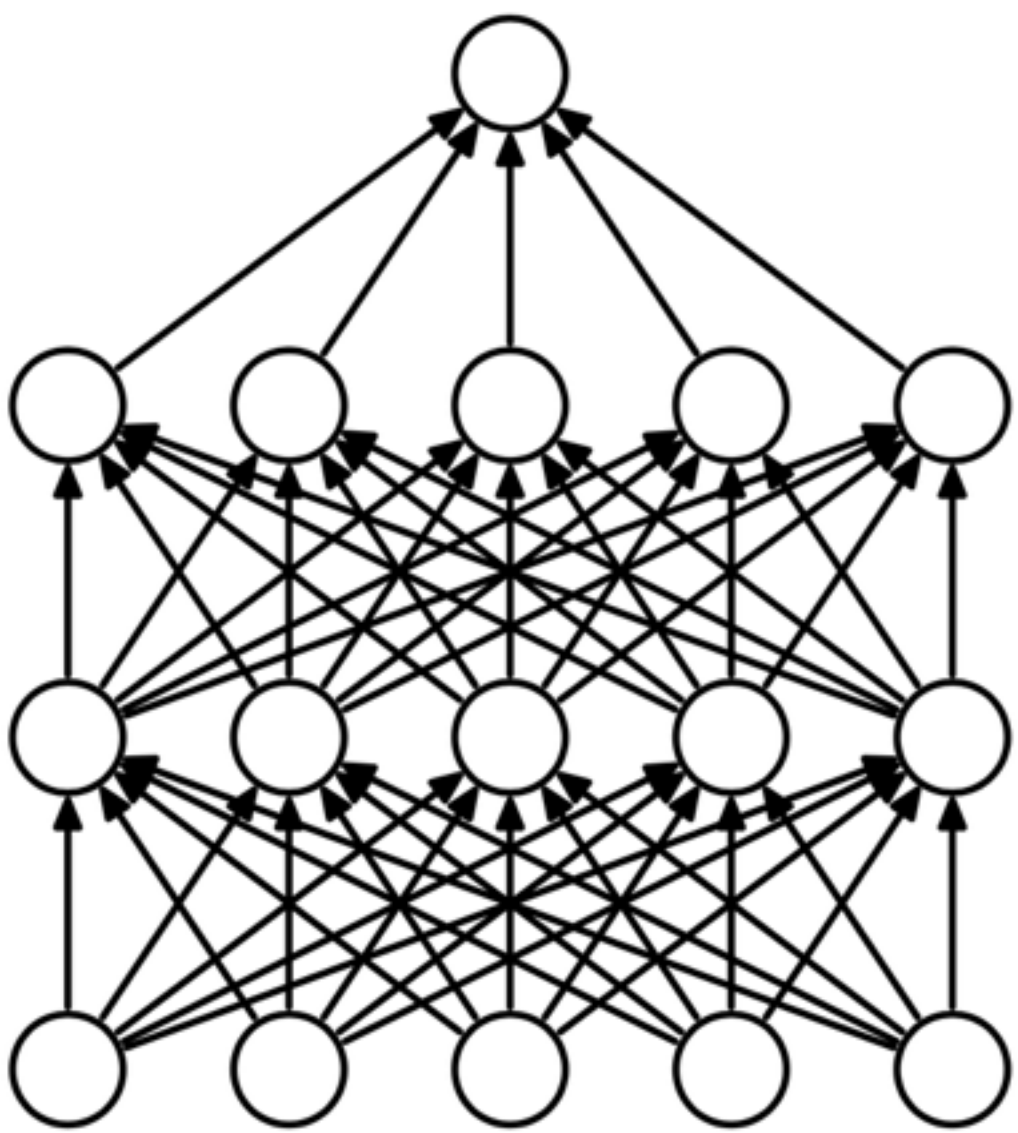
# Universal Approximators



- Neural networks can approximate any[1] function.
  - Capacity
    - Layers
    - hidden layer size
    - A*bsence* of regularization
    - Optimal activation functions and hyper-parameters.
  - Training data

[1] K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Netw.* 2, 5 (July 1989) : proved this for a specific class of functions.

# Universal Approximators

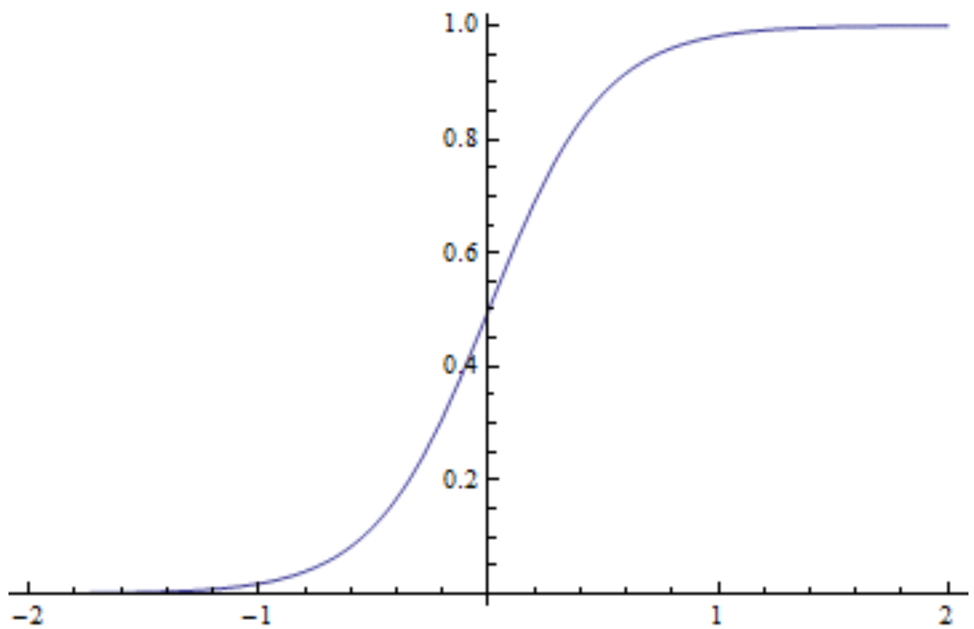


- We will focus on two important aspects of training:
  - Ideal properties of parameters during training
  - Generalization error
- Other things to consider:
  - Hyper-parameter optimization
  - Choice of model, loss functions
  - Learning rates (Use Adadelta or Adam)
  - …

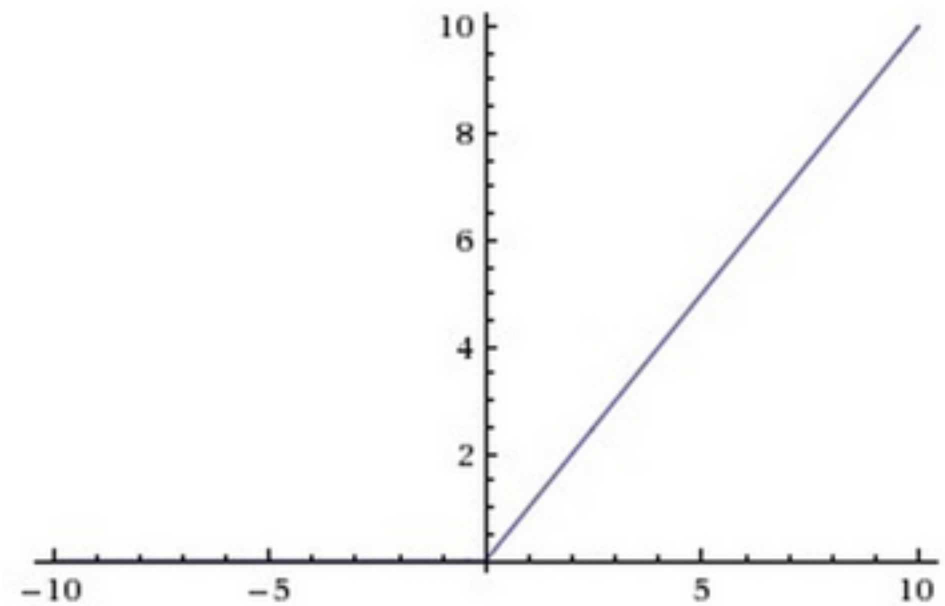
# Properties of Parameters

- Responsive to activation functions
- Numerically stable

# Activation Saturation




Sigmoid




Relu

# Initialization of weight matrices

- Are you using a non-recurrent NN ?
    - Use the Xavier initialization
    - (use small values to initialize bias vectors)

Glorot & Bengio (2010), He et.al (2015)

# Initialization of weight matrices (Xavier, He)

- Tanh $$\left[-\sqrt{\frac{6}{fan_{in} + fanout}}, \sqrt{\frac{6}{fan_{in} + fanout}}\right]$$

- Sigmoid $$\left[-4\sqrt{\frac{6}{fan_{in} + fanout}}, 4\sqrt{\frac{6}{fan_{in} + fanout}}\right]$$

- Relu $$\left[-\sqrt{\frac{2}{fan_{in} + fanout}}, \sqrt{\frac{2}{fan_{in} + fanout}}\right]$$

# Initialization of weight matrices

- Are you using a recurrent NN ?
  - With LSTMs : Use the Saxe initialization
    - All weight matrices initialized to be orthonormal (Gaussian noise -> SVD)
  - Without LSTMS
    - All weight matrices initialized to identity

Saxe et al, 2014,

# Watch your input

- A high variance in input features may cause saturation very early
  - Mean subtraction : Same mean across all features
  - Normalization : Same scale across all features

# Numerical stability

- Floating point precision causes values to overflow or underflow

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}.$$

- Instead, compute

$$\text{softmax}(\boldsymbol{z}) \text{ where } \boldsymbol{z} = \boldsymbol{x} - \max_i x_i$$

# Numerical stability

$$L = -t\log(p) - (1-t)\log(1-p)$$

- Cross Entropy Loss
  - Probabilities close to 0 for the correct label will cause underflow
  - Use range clipping. All values between 0.000001 and 0.999999.

# Generalization

## Preventing Overfitting

# Regularization

- L2 regularization $\frac{\lambda}{2}\|\vec{w}\|^2$

- L1 regularization $\lambda \| w \|_1$

- Gradient clipping (max norm constraints)

# Regularization

- Perform layer-wise regularization
  - After computing the activated value of each layer, normalize with the L2 norm.
  - No regularization hyper-parameters
  - No waiting till back-propagation for weight penalties to flow in

# Dropout

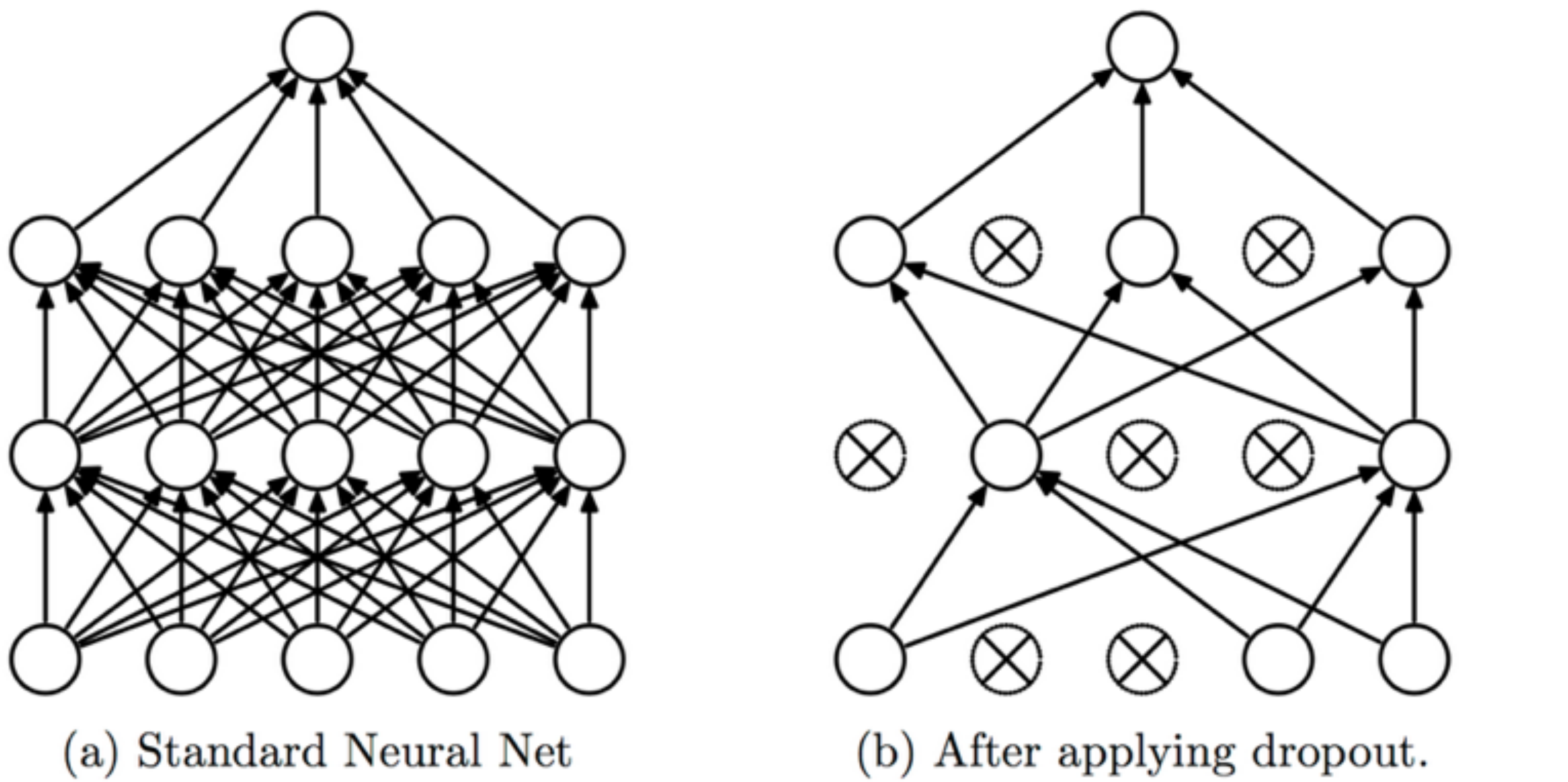
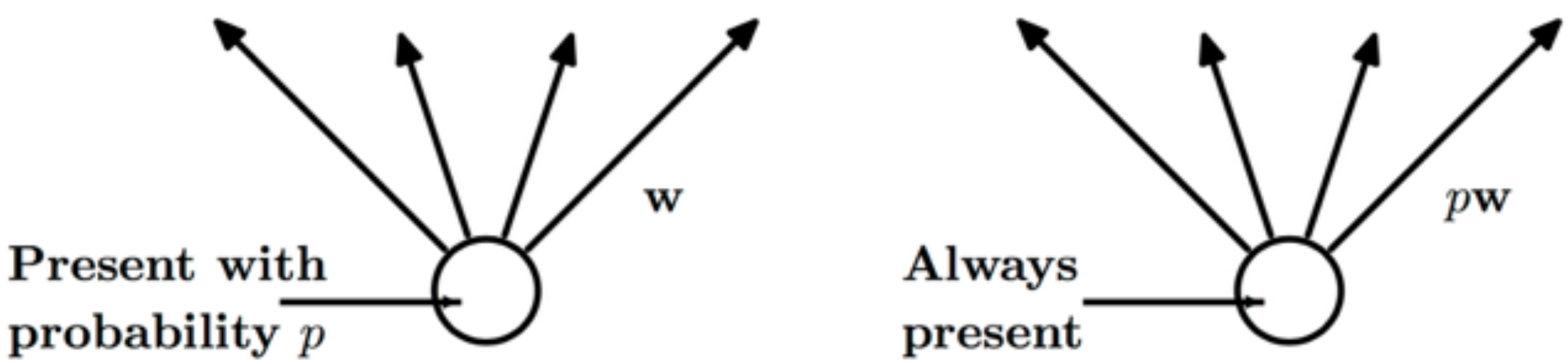



Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15

# Dropout




(a) At training time

(b) At test time

Figure 2: **Left**: A unit at training time that is present with probability $p$ and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by $p$. The output at test time is same as the expected output at training time.

# Dropout

- Interpret as regularization
- Interpret as training an ensemble of thinned networks