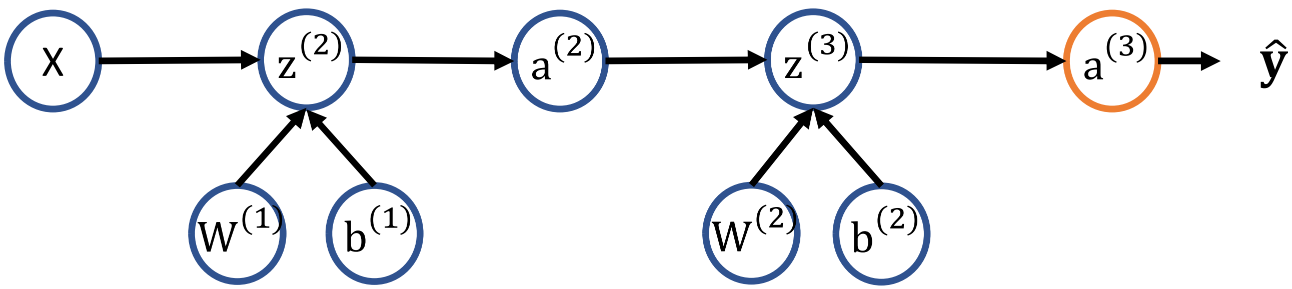
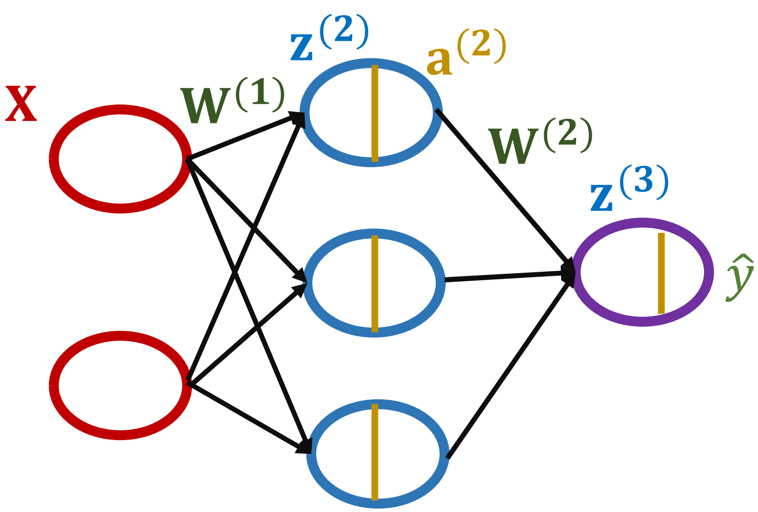
Training an MLP

Forward and Backward Propagation

Flow graph - Forward propagation



How do we evaluate our prediction?

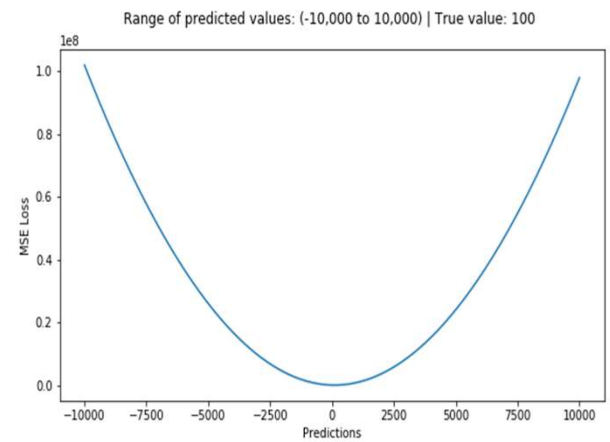


$$z^{(2)} = w^{(1)}x$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = w^{(2)}a^{(2)}$$
$$\hat{y} = a^{(3)} = f(z^{(3)})$$

Loss Function - Examples

❑ Regression:

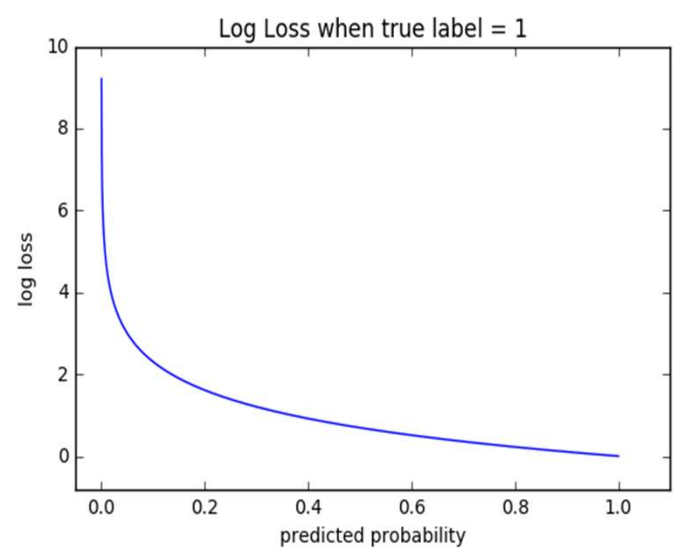
- Mean Squared Error: $\frac{1}{N}\sum_j (y_j^{\text{actual}} - y_j^{\text{predicted}})^2$



❑ Classification:

- Cross Entropy Loss: $-(y \log y' + (1-y)\ \log(1-y'))$

Actual Value (y)	Predicted Value (y')	Loss
0	0	0
0	1	∞
1	0	∞
1	1	0



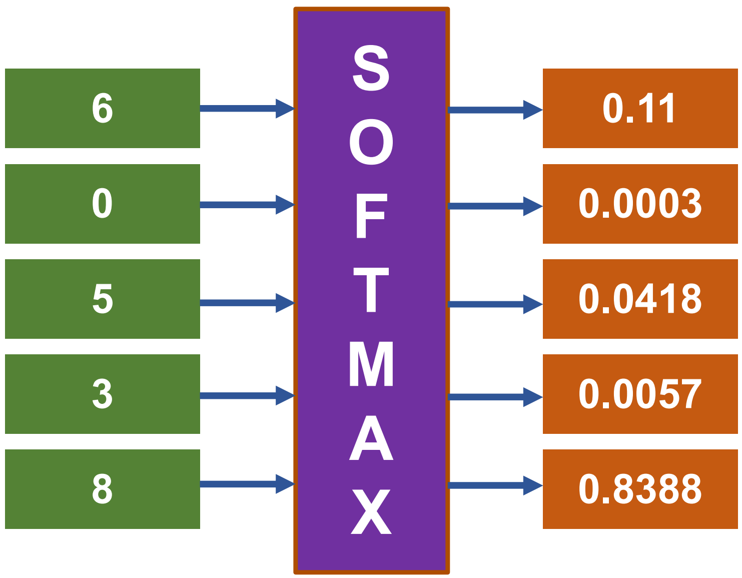
Softmax Activation

- For multi-class classification:
  - We need multiple outputs (1 output per class)
  - We would like to estimate the conditional probability $p(y = c \mid x)$

- We use the softmax activation function at the output:

$$\mathbf{f(x) = softmax(x)} = \left[\frac{\mathbf{e}^{\mathbf{x}_1}}{\sum_c \mathbf{e}^{\mathbf{x}_c}} \;\cdots\; \frac{\mathbf{e}^{\mathbf{x}_c}}{\sum_c \mathbf{e}^{\mathbf{x}_c}}\right]$$

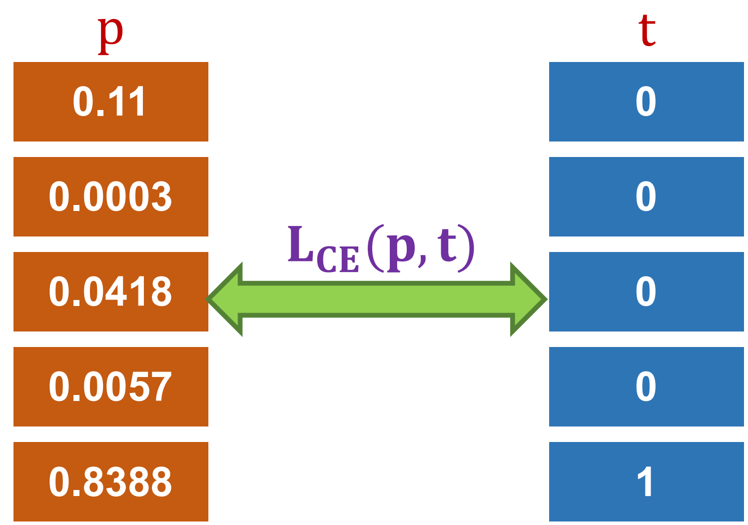
- Normalizes the output



Cross Entropy

$$\text{Loss} = -\sum_{i=1}^{n} t_i \log p_i, \qquad \text{for n classes}$$

where $\mathbf{t_i}$ is the truth label and $\mathbf{p_i}$ is the Softmax probability for the $\mathbf{i^{th}}$ class.



$$L_{CE} = -\sum_{i=1} t_i \log p_i$$

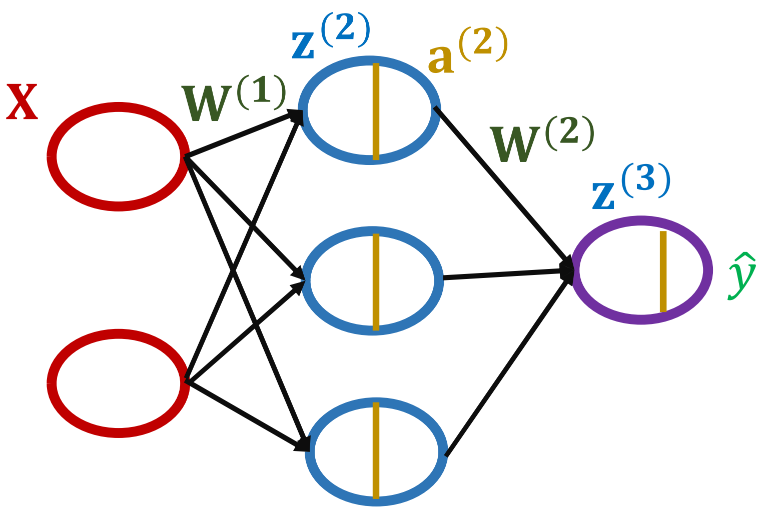
$$= -[0 \times \log_2 0.11 + 0 \times \log_2 0.0003 + 0 \times \log_2 0.0418 + 0 \times \log_2 0.057 + 1 \times \log_2 0.8388]$$

$$= -\log_2 0.8388 = 0.2536$$

Training - MLP

- **Step 1** – We initialized the network with random weights.
- **Step 2** – Perform forward propagation and determine $\hat{y}$.
- **Step 3** – Determine the Loss Function, $|y - \hat{y}|$.
- **Step 4** – Do backward propagation and determine change in weights.
- **Step 5** – Update all weights in all layers.
- **Step 6** – Repeat Steps 2 -5 until convergence.

Forward Propagation



$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f\left(z^{(2)}\right) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f\left(z^{(3)}\right) \quad (4)$$

$$J = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 \quad (5)$$

$$J = \frac{1}{2}\sum_i \left(y_i - f\left(z^{(3)}\right)\right)^2 = \frac{1}{2}\sum_i \left(y_i - f\left(a^{(2)}W^{(2)}\right)\right)^2 = \frac{1}{2}\sum_i \left(y_i - f\left(f\left(z^{(2)}\right)W^{(2)}\right)\right)^2$$

$$= \frac{1}{2}\sum_i \left(y_i - f\left(f\left(XW^{(1)}\right)W^{(2)}\right)\right)^2$$

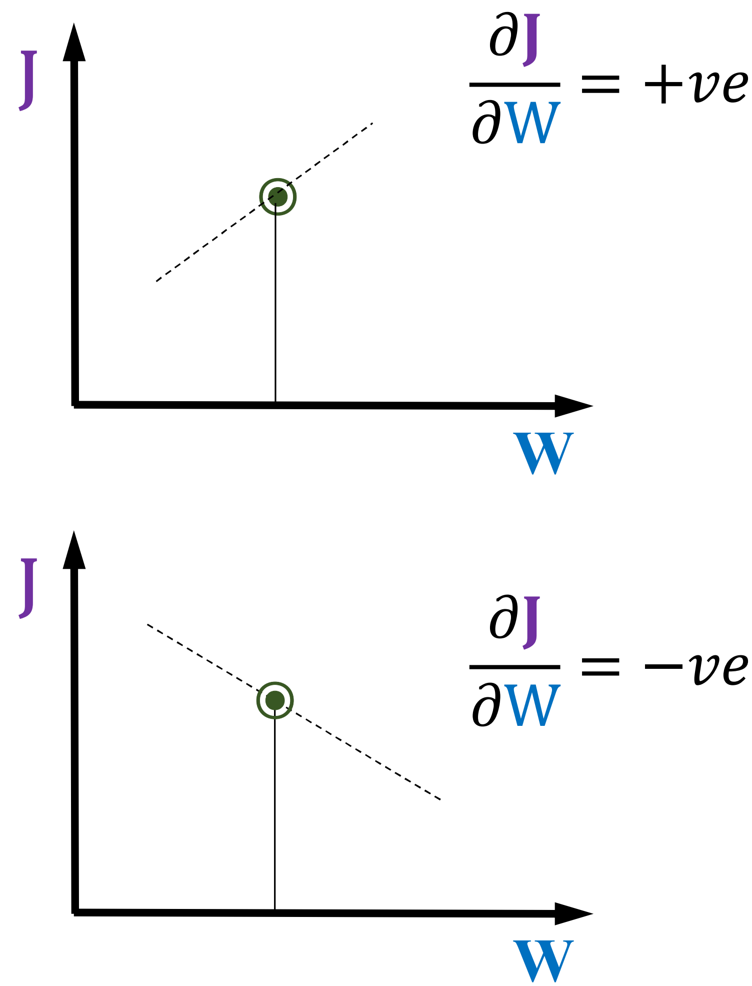
MLP - Training

$$J = \frac{1}{2}\sum_i \left(y_i - f\left(f\left(X W^{(1)}\right) W^{(2)}\right)\right)^2$$

- How does J change, when we change $W^{(1)}$, $W^{(2)}$?

$$\boxed{\frac{\partial \mathbf{J}}{\partial \mathbf{W}^{(1)}}, \frac{\partial \mathbf{J}}{\partial \mathbf{W}^{(2)}}} \Rightarrow \frac{\partial \mathbf{J}}{\partial \mathbf{W}}$$

- Perform Gradient Descent, where the weights are updated, by computing the gradient of the error function **J** at **W.**



MLP:: Gradient Descent

- Stochastic Gradient Descent

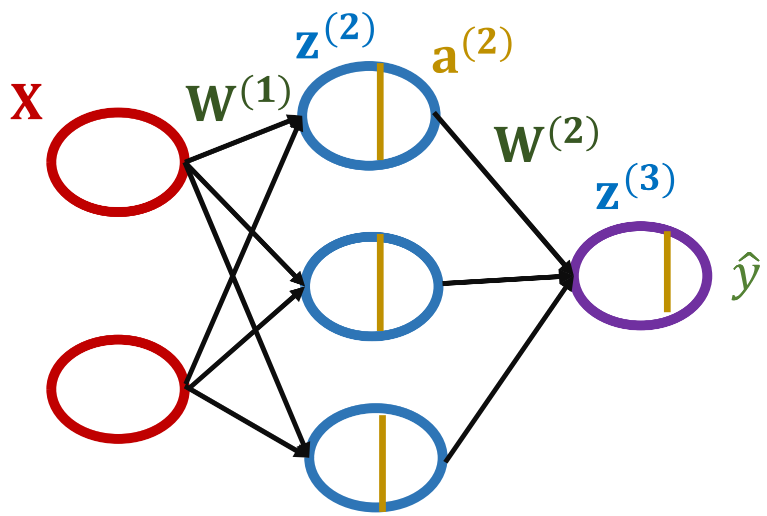
$$\begin{bmatrix} 8 & 3 \\ 2 & 11 \\ 9 & 5 \end{bmatrix} \begin{array}{l} \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \longrightarrow \text{update } W \\ \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \longrightarrow \text{update } W \\ \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \longrightarrow \text{update } W \end{array}$$

- Batch Gradient Descent

$$\begin{bmatrix} 8 & 3 \\ 2 & 11 \\ 9 & 5 \end{bmatrix} \begin{array}{l} \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \\ \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \\ \longrightarrow \frac{\partial \mathbf{J}}{\partial W} \quad + \end{array}$$

$$\sum \frac{\partial \mathbf{J}}{\partial W} \longrightarrow \text{update } W$$

MLP : Backward Propagation



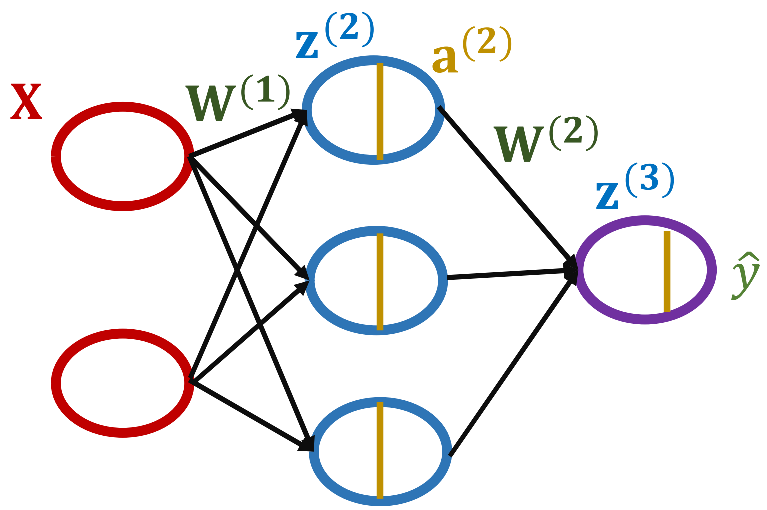
$$\mathbf{J} = \frac{1}{2}\sum_{i}\left(y_i - f\left(f\left(X\,W^{(1)}\right)W^{(2)}\right)\right)^2$$

- How do we compute $\frac{\partial \mathbf{J}}{\partial \mathbf{W}^{(1)}}, \frac{\partial \mathbf{J}}{\partial \mathbf{W}^{(2)}}$?

- $\mathbf{J}$ is a function of $\mathbf{W}^{(2)}$
- $\mathbf{J}$ is a function of a function of $\mathbf{W}^{(1)}$

We use Chain Rule to compute these partial derivatives

Backward Propagation



$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f\left(z^{(2)}\right) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f\left(z^{(3)}\right) \quad (4)$$

$$J = \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 \quad (5)$$

$$J = \frac{1}{2}\sum_i \left(y_i - f\left(f\left(XW^{(1)}\right)W^{(2)}\right)\right)^2$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial \hat{y}} \quad \frac{\partial \hat{y}}{\partial z^{(3)}} \quad \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial \hat{y}} \quad \frac{\partial \hat{y}}{\partial z^{(3)}} \quad \frac{\partial z^{(3)}}{\partial a^{(2)}} \quad \frac{\partial a^{(2)}}{\partial z^{(2)}} \quad \frac{\partial z^{(2)}}{\partial W^{(1)}}$$

Backward Propagation

$$\frac{\partial \mathbf{J}}{\partial W^{(2)}} = \frac{\partial \mathbf{J}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}}$$

$$= \sum_i (y - \hat{y})\; f'\left(z^{(3)}\right)\; a^{(2)}$$

$$\frac{\partial \mathbf{J}}{\partial W^{(1)}} = \frac{\partial \mathbf{J}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial W^{(1)}}$$

$$= \sum_i (y - \hat{y}) f'\left(z^{(3)}\right) W^{(2)} f'\left(z^{(2)}\right) X$$

$$z^{(2)} = X\, W^{(1)} \quad (1)$$

$$a^{(2)} = f\left(z^{(2)}\right) \quad (2)$$

$$z^{(3)} = a^{(2)} W^{(2)} \quad (3)$$

$$\hat{y} = f\left(z^{(3)}\right) \quad (4)$$

$$J = \frac{1}{2} \sum_i (y_i - \hat{y}_i)^2 \quad (5)$$

Training - MLP

- **Step 1** – We initialized the network with random weights.

- **Step 2** – Forward Propagation

$$z^{(2)} = X\,W^{(1)} \quad (1)$$

$$a^{(2)} = f\left(z^{(2)}\right) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f\left(z^{(3)}\right) \quad (4)$$

$$J = \frac{1}{2}\sum_{i}(y_i - \hat{y}_i)^2 \quad (5)$$

- **Step 3** – Backward Propagation

$$\frac{\partial \mathbf{J}}{\partial W^{(2)}} = \sum_{i}(y - \hat{y})f'\left(z^{(3)}\right)a^{(2)}$$

$$\frac{\partial \mathbf{J}}{\partial W^{(1)}} = \sum_{i}(y - \hat{y})f'\left(z^{(3)}\right)W^{(2)}f'\left(z^{(2)}\right)X$$

- **Step 4** – Update all weights simultaneously

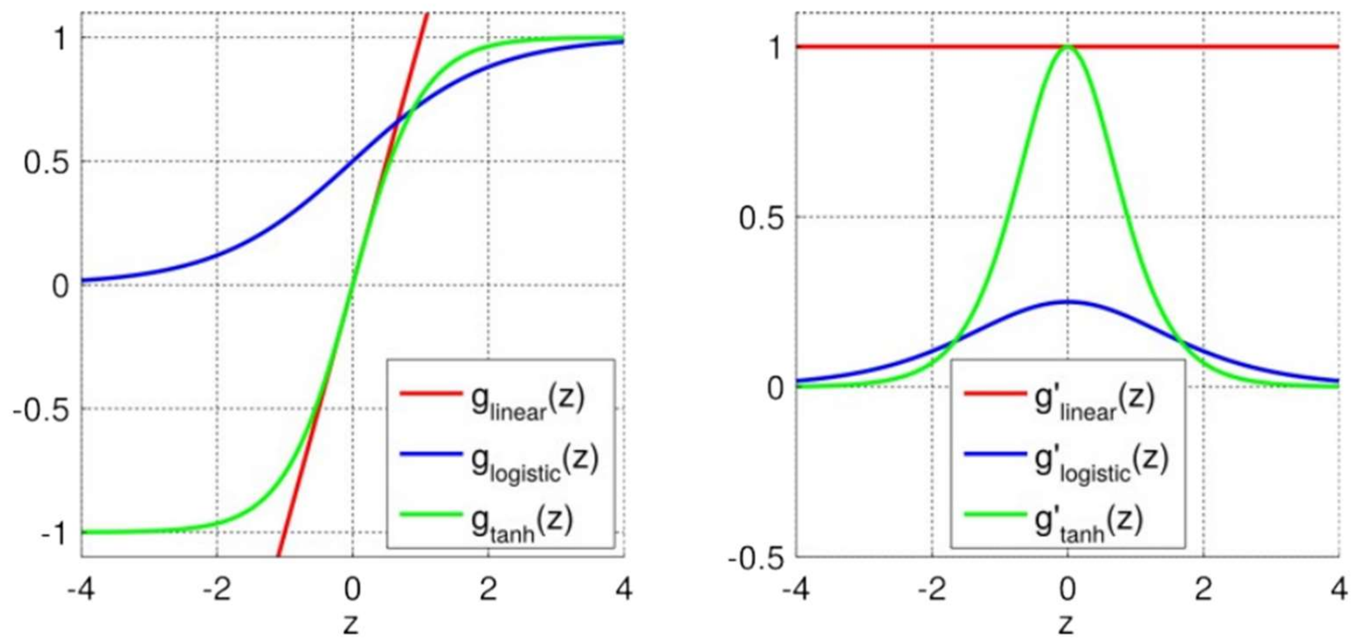
$$w^{(t+1)} = w^{(t)} - \eta^{(t)}\nabla_w J(w)$$

- Repeat Step – 2, 3 and 4 until convergence

Vanishing/Exploding Gradient Problem

- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large).

- Makes it very difficult to train deep networks, or simple recurrent networks over many time steps.



Overfitting in Neural Networks

- Deep Neural Networks are prone to overfitting because of the large number of parameters it encloses.

- The network can become less prone to overfitting by removing certain layers of the network or decreasing the number of neurons.

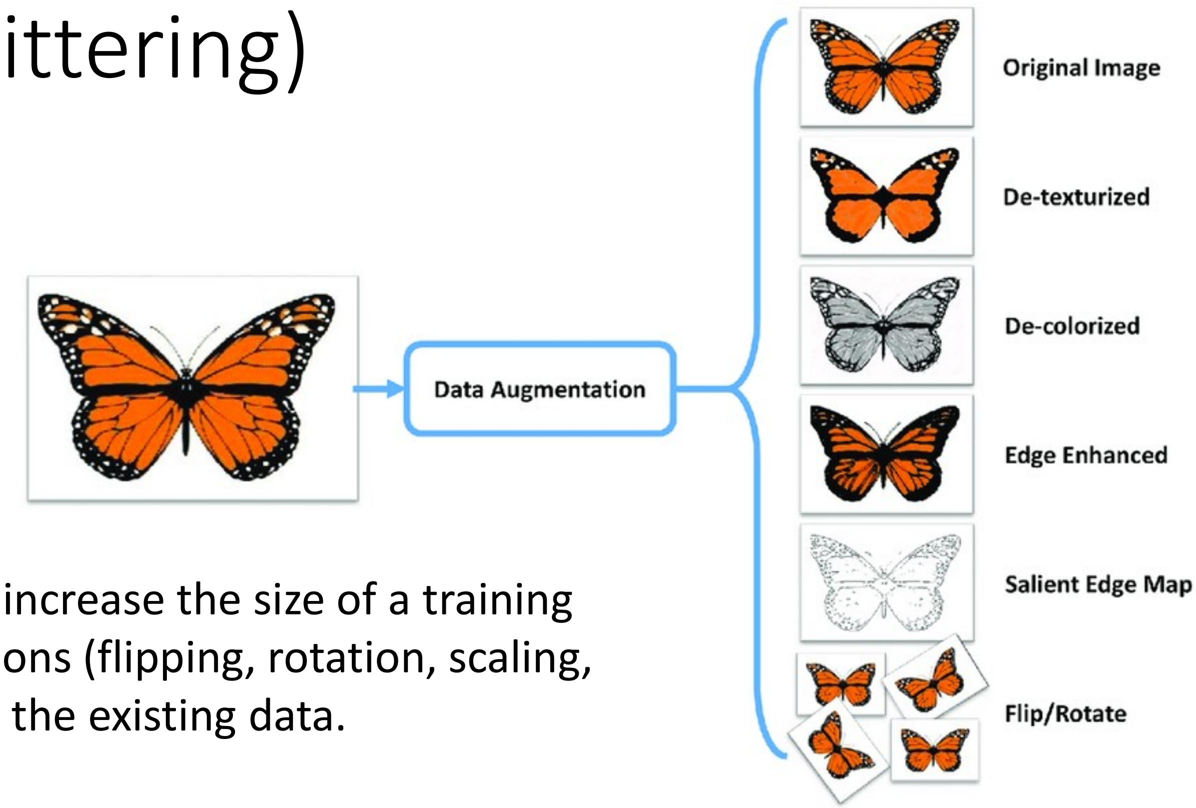
- Different strategies have been proposed to take care of the overfitting.

    - Reduce overfitting by training the network on more examples.

    - Reduce overfitting by changing the complexity of the network.
        - Changing the network structure (number of weights)
        - Changing the network parameters (values of weights)

Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Deep Image [Wu et al. 2015]

- Data augmentation is used to artificially increase the size of a training dataset by applying various transformations (flipping, rotation, scaling, cropping, shearing, and adding noise) to the existing data.

- It helps the model to learn invariant features and improves its ability to handle different variations and deformations

http://arxiv.org/pdf/1501.02876v2.pdf

Weight Regularization

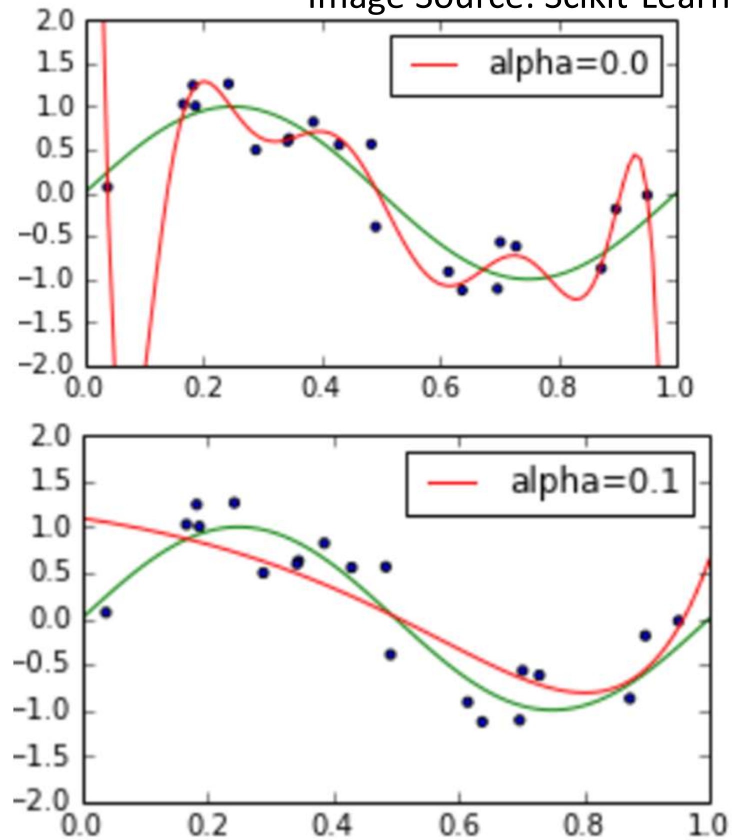
- Penalize the model during training based on the magnitude of the weights.
- Ridge Regression tries to reduce the length $||w||$ of the parameter vector, promoting lesser dependency of $\hat{y}$ on predictors (lower model complexity).

$$L_{Ridge} = J + \alpha \frac{1}{2} \sum_{i=1}^{P} w_i^2$$

- Lasso Regression tries to reduce the city block distance $|w|$ of the parameter vector, causing automatic feature selection.

$$L_{Lasso} = J + \alpha \sum_{i=1}^{P} |w_i|$$

Image Source: Scikit-Learn

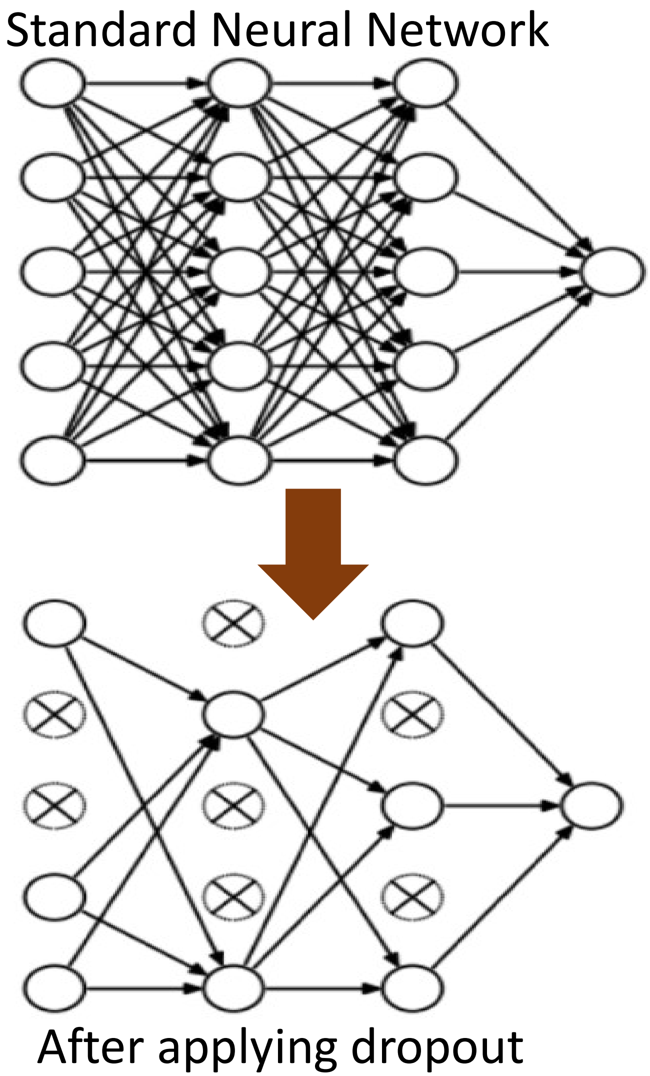


Dropout

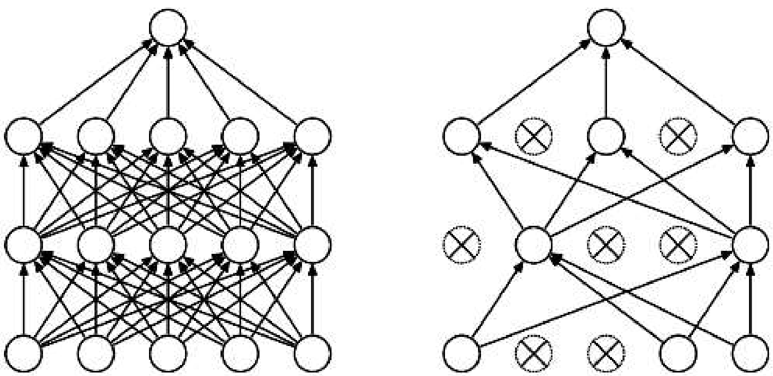
- Used for preventing overfitting in deep neural network which contains a large number of parameters
- Ridge and Lasso reduces overfitting by modifying the loss function, whereas in dropout a certain number of neurons at a layer is deactivated from firing during training
- The dropout rate is a hyperparameter that determines the probability of dropping out each neuron, usually ranging from 0.2 to 0.5.
- Higher dropout rates provide more regularization but may also slow down the convergence of the network.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

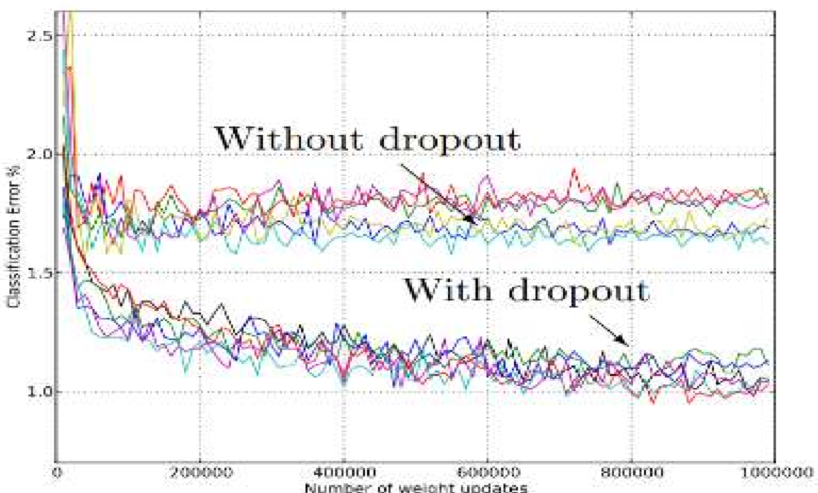
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf



Dropout

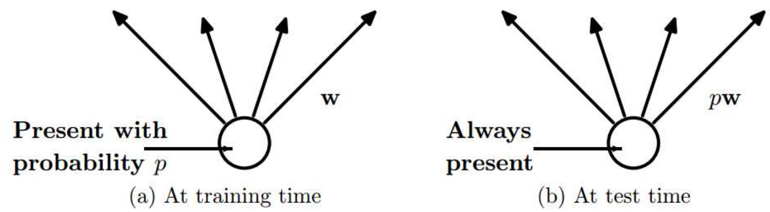


(a) Standard Neural Net (b) After applying dropout.



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Main Idea: approximately combining exponentially many different neural network architectures efficiently



Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Early Stopping

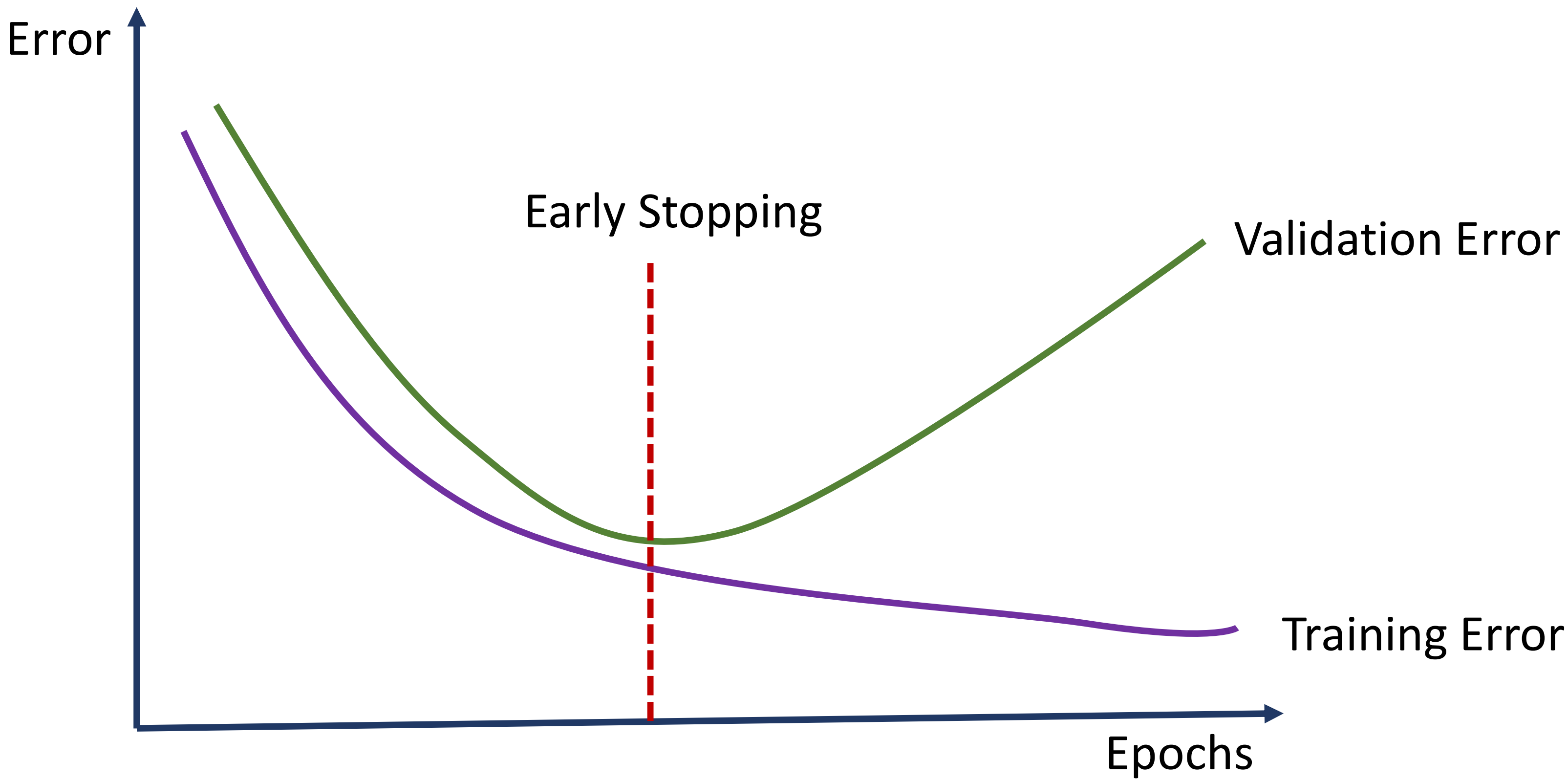
- While training a large network, at some point the model stops generalizing and starts learning the statistical noise in the training dataset.

Early Stopping

- During training, evaluate the model on a validation dataset after each epoch.
- Provided the performance of the model on the validation dataset starts to degrade stop the training process instantly.
- The model at the time the training stopped is used for the test analysis

- We train the network on a larger number of training epochs than may normally be required, to give the network plenty of opportunity to fit, then begin to overfit the training dataset.
- A trigger for stopping the training process is chosen.
    - No change in metric over a given number of epochs.
    - An absolute change in metric or a decrease in performance observed over a certain number of epochs.

Early Stopping



Batch Normalization

- Batch Normalization (BN) is a normalization method/layer for neural networks.
- Usually inputs to neural networks are normalized
- A new layer is added so the gradient can "see" the normalization and made adjustments if needed.
  - The new layer has the power to learn the identity function to de-normalize the features if necessary!

https://arxiv.org/abs/1502.03167

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

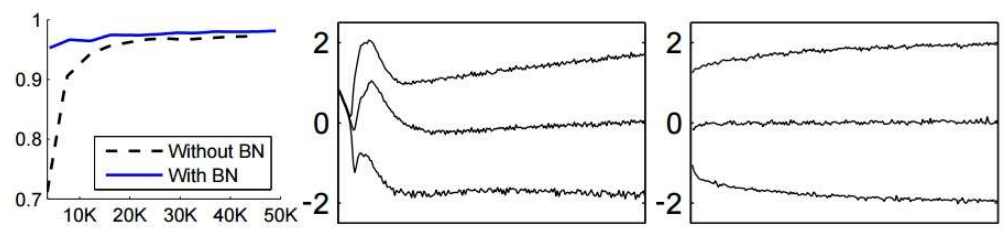
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



(a) (b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]