(Deep) Neural Networks

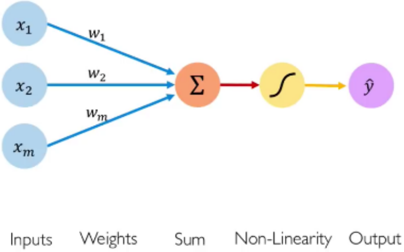
K Sri Rama Murty

IIT Hyderabad

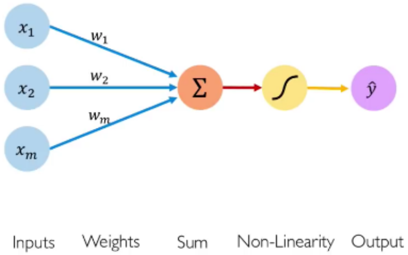
ksrm@ee.iith.ac.in

March 22, 2022

Summary of Linear/Logistic Regression

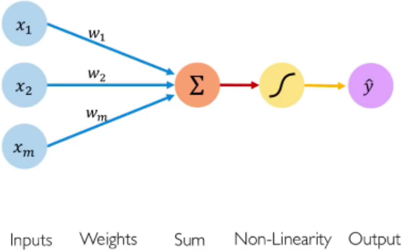


Summary of Linear/Logistic Regression



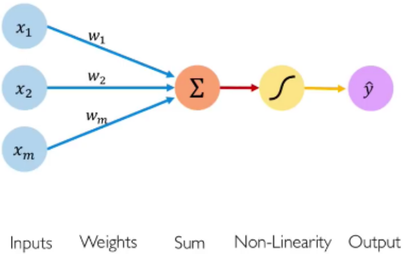
- Pass linear aggregated input though activation function

Summary of Linear/Logistic Regression



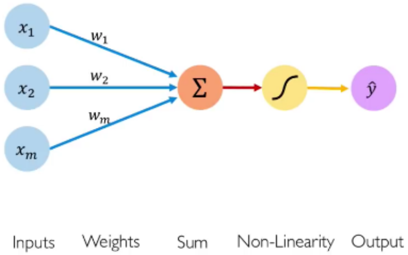
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$

Summary of Linear/Logistic Regression



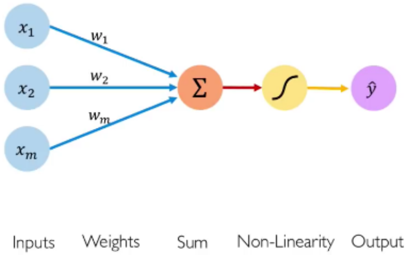
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression

Summary of Linear/Logistic Regression



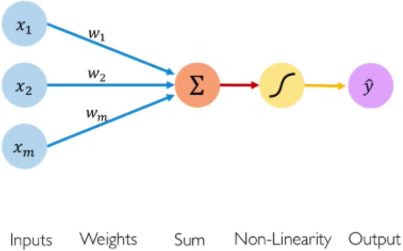
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression

Summary of Linear/Logistic Regression



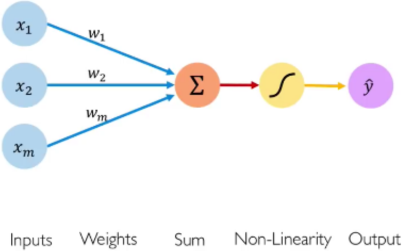
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression
  - f(.) is softmax: Multiclass logistic regression

Summary of Linear/Logistic Regression



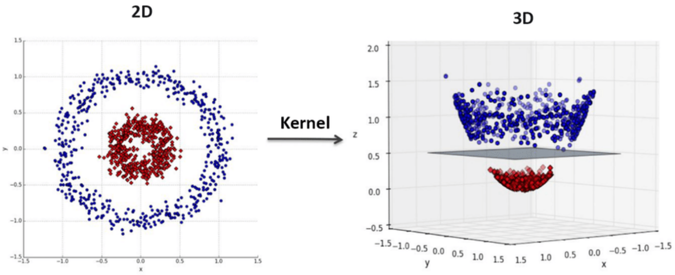
- Pass linear aggregated input though activation function
    - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
    - f(.) is linear: Linear regression
    - f(.) is sigmoid: Logistic regression
    - f(.) is softmax: Multiclass logistic regression
    - f(.) is hard-limiting function: Perceptron

Summary of Linear/Logistic Regression

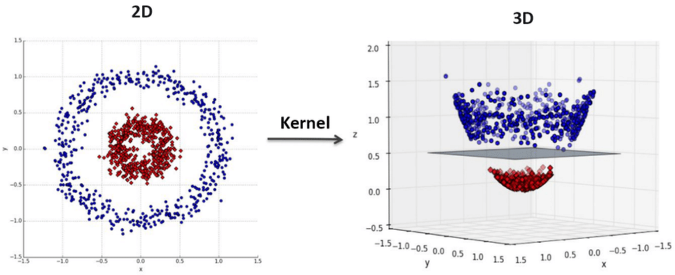


- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression
  - f(.) is softmax: Multiclass logistic regression
  - f(.) is hard-limiting function: Perceptron
- Models linear ip-op relation or linearly separable boundaries

Cover's Theorem

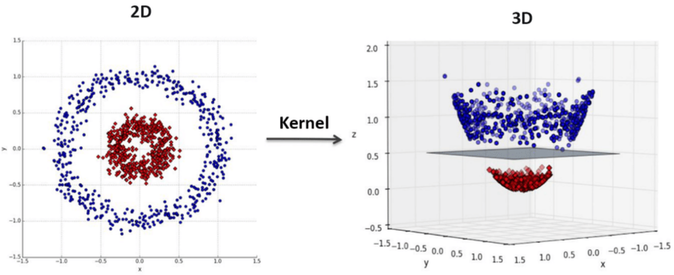


Cover's Theorem



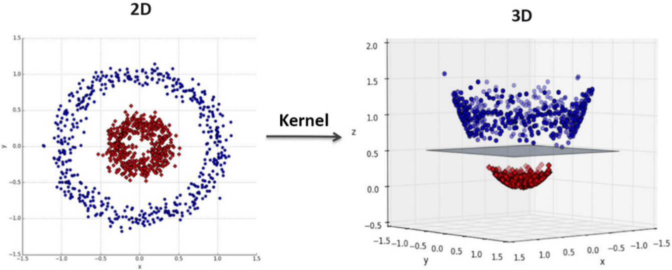
- A complex pattern classification problem can be transformed to a linearly seprabale one, provided

Cover's Theorem



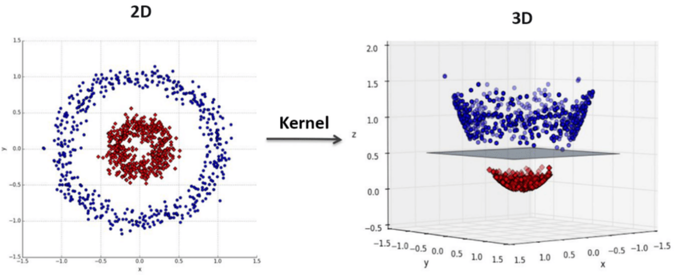
- A complex pattern classification problem can be transformed to a linearly seprabale one, provided
  - The transformation from input space to feature space is nonlinear

Cover's Theorem



- A complex pattern classification problem can be transformed to a linearly seprabale one, provided
    - The transformation from input space to feature space is nonlinear
    - The dimensionality of the feature space is high enough

Cover's Theorem



- A complex pattern classification problem can be transformed to a linearly seprabale one, provided
    - The transformation from input space to feature space is nonlinear
    - The dimensionality of the feature space is high enough
    - $\mathbf{x} = [x_1 \ x_2] \rightarrow \phi(\mathbf{x}) = [x_1^2 \ x_2^2 \ \sqrt{2}x_1x_2]$

Data Independent Transformation

Data Independent Transformation

- Apply linear models in feature space $\phi(\mathbf{x})$:

$$\hat{t} = y(\phi(\mathbf{x}), \mathbf{w}) = h\left(\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x})\right)$$

Data Independent Transformation

- Apply linear models in feature space $\phi(\mathbf{x})$:

$$\hat{t} = y(\phi(\mathbf{x}), \mathbf{w}) = h\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x})\right)$$

- Polynomial kernel of order n: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\mathsf{T}\mathbf{y} + c)^n$
  - For $\mathbf{x} \in \mathbb{R}^2$, $n = 2$ and $c = 1$: $k(\mathbf{x}, \mathbf{y}) = (x_1 y_1 + x_2 y_2 + 1)^2$
  - $k(\mathbf{x}, \mathbf{y}) = \phi^\mathsf{T}(\mathbf{x})\phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1 \; x_1^2 \; x_2^2 \; \sqrt{2}x_1 \; \sqrt{2}x_2 \; \sqrt{2}x_1x_2]^\mathsf{T}$
  - Polynomial kernels are explicit kernels

Data Independent Transformation

- Apply linear models in feature space $\phi(\mathbf{x})$:

$$\hat{t} = y(\phi(\mathbf{x}), \mathbf{w}) = h\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x})\right)$$

- Polynomial kernel of order n: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\mathsf{T}\mathbf{y} + c)^n$
  - For $\mathbf{x} \in \mathbb{R}^2$, $n = 2$ and $c = 1$: $k(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + 1)^2$
  - $k(\mathbf{x}, \mathbf{y}) = \phi^\mathsf{T}(\mathbf{x})\phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1\ x_1^2\ x_2^2\ \sqrt{2}x_1\ \sqrt{2}x_2\ \sqrt{2}x_1x_2]^\mathsf{T}$
  - Polynomial kernels are explicit kernels
- Gaussian kernel: $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{h}\right)$
  - Gaussian kernel is an infinite dimensional kernel
  - Feature representation $\phi(\mathbf{x})$ is not available: *Implicit kernel*
  - However, inner product can be evaluated in the transformed space

Data Independent Transformation

- Apply linear models in feature space $\phi(\mathbf{x})$:

$$\hat{t} = y(\phi(\mathbf{x}), \mathbf{w}) = h\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x})\right)$$

- Polynomial kernel of order n: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\mathsf{T}\mathbf{y} + c)^n$
  - For $\mathbf{x} \in \mathbb{R}^2$, $n = 2$ and $c = 1$: $k(\mathbf{x}, \mathbf{y}) = (x_1 y_1 + x_2 y_2 + 1)^2$
  - $k(\mathbf{x}, \mathbf{y}) = \phi^\mathsf{T}(\mathbf{x})\phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1\ x_1^2\ x_2^2\ \sqrt{2}x_1\ \sqrt{2}x_2\ \sqrt{2}x_1x_2]^\mathsf{T}$
  - Polynomial kernels are explicit kernels
- Gaussian kernel: $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{h}\right)$
  - Gaussian kernel is an infinite dimensional kernel
  - Feature representation $\phi(\mathbf{x})$ is not available: *Implicit kernel*
  - However, inner product can be evaluated in the transformed space
- There is no reason to believe that the same transformation suits well for all domains - image, speech, medical, forensic, financial etc.,

Data-Dependent Transformation

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^M$

$$\hat{t} = f\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j\phi_j(\mathbf{x}[n])\right)$$

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^M$

$$\hat{t} = f\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j\phi_j(\mathbf{x}[n])\right)$$

- Goal is to design the basis functions $\phi_j(\mathbf{x})$ from the data $(\mathcal{X}, \mathcal{T})$

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^M$

$$\hat{t} = f\left(\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}[n])\right)$$

- Goal is to design the basis functions $\phi_j(\mathbf{x})$ from the data $(\mathcal{X}, \mathcal{T})$
  - Make $\phi_j(\mathbf{x}[n])$ depend on some parameters $\mathbf{W}^{(1)}$

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^M$

$$\hat{t} = f\left(\mathbf{w}^{\mathsf{T}} \phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}[n])\right)$$

- Goal is to design the basis functions $\phi_j(\mathbf{x})$ from the data $(\mathcal{X}, \mathcal{T})$
  - Make $\phi_j(\mathbf{x}[n])$ depend on some parameters $\mathbf{W}^{(1)}$
  - Adjust the parameters $\mathbf{W}^{(1)}$ along with $\mathbf{w}$ during training

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^M$

$$\hat{t} = f\left(\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}[n])\right)$$

- Goal is to design the basis functions $\phi_j(\mathbf{x})$ from the data $(\mathcal{X}, \mathcal{T})$
  - Make $\phi_j(\mathbf{x}[n])$ depend on some parameters $\mathbf{W}^{(1)}$
  - Adjust the parameters $\mathbf{W}^{(1)}$ along with $\mathbf{w}$ during training
- Neural networks use nonlinear function of linear combination of inputs for basis functions

$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right) \qquad j = 1, 2 \cdots, M$$

Data-Dependent Transformation

- Desired output is estimated as linear combination of nonlinear basis functions of the inputs $\mathbf{x} \in \mathbb{R}^D \to \phi(\mathbf{x}) \in \mathbb{R}^M$

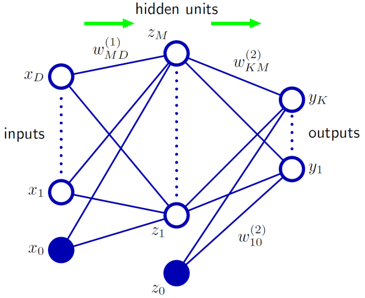
$$\hat{t} = f\left(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}[n])\right)$$

- Goal is to design the basis functions $\phi_j(\mathbf{x})$ from the data $(\mathcal{X}, \mathcal{T})$
  - Make $\phi_j(\mathbf{x}[n])$ depend on some parameters $\mathbf{W}^{(1)}$
  - Adjust the parameters $\mathbf{W}^{(1)}$ along with $\mathbf{w}$ during training
- Neural networks use nonlinear function of linear combination of inputs for basis functions

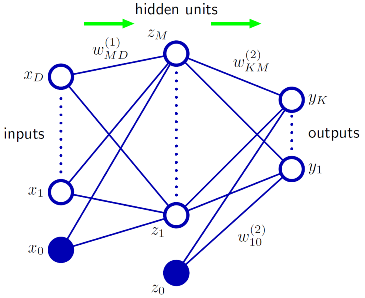
$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right) \qquad j = 1, 2 \cdots, M$$

- $h(.)$ can be sigmoid, tanh, relu, softplus etc.,

2-Layer Feed Forward Network

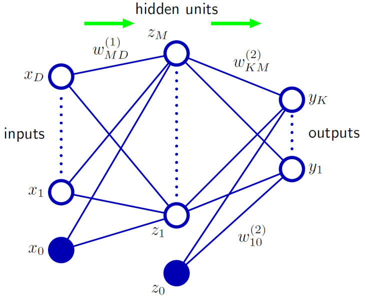


2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w_{ji}^{(1)} : x_i \rightarrow z_j$
- $w_{kj}^{(2)} : z_j \rightarrow y_k$

2-Layer Feed Forward Network

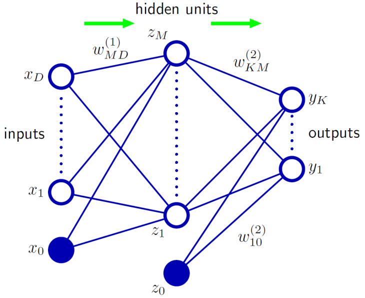


- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \to z_j$
- $w^{(2)}_{kj} : z_j \to y_k$

- At j^{th} hidden unit:

$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

2-Layer Feed Forward Network



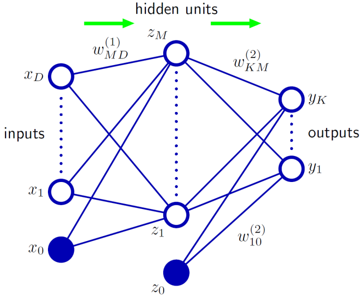
- At j^{th} hidden unit:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)$$

- $h(.)$ is a nonlinear function

- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w_{ji}^{(1)} : x_i \rightarrow z_j$
- $w_{kj}^{(2)} : z_j \rightarrow y_k$

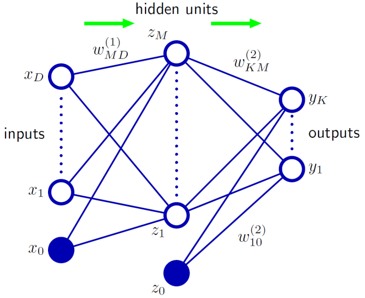
2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$
- $h(.)$ is a nonlinear function
- At k^{th} output unit:
$$a_k = \sum_{i=0}^{M} w^{(2)}_{kj} z_j \qquad y_k = f(a_k)$$

2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:

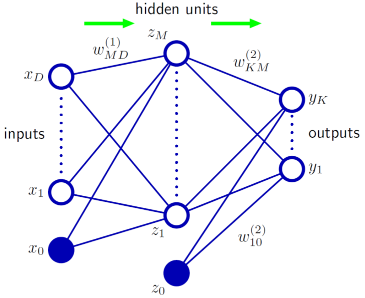
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

- $h(.)$ is a nonlinear function
- At k^{th} output unit:

$$a_k = \sum_{i=0}^{M} w^{(2)}_{kj} z_j \qquad y_k = f(a_k)$$

- Choice of $f(.)$ depends on task

2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w_{ji}^{(1)} : x_i \to z_j$
- $w_{kj}^{(2)} : z_j \to y_k$

- At j^{th} hidden unit:

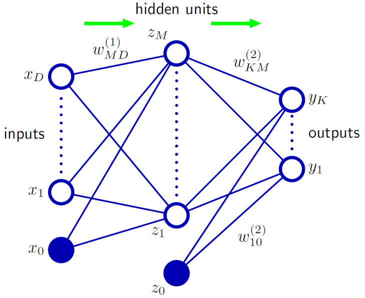
$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)$$

- $h(.)$ is a nonlinear function
- At k^{th} output unit:

$$a_k = \sum_{i=0}^{M} w_{kj}^{(2)} z_j \qquad y_k = f(a_k)$$

- Choice of $f(.)$ depends on task
  - Linear for Regression

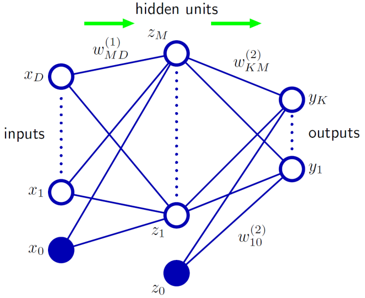
2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$
- $h(.)$ is a nonlinear function
- At k^{th} output unit:
$$a_k = \sum_{i=0}^{M} w^{(2)}_{kj} z_j \qquad y_k = f(a_k)$$
- Choice of $f(.)$ depends on task
  - Linear for Regression
  - Sigmoid for Binary Classification

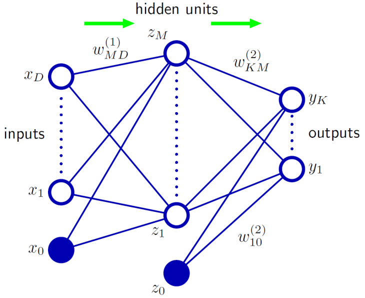
2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$
- $h(.)$ is a nonlinear function
- At k^{th} output unit:
$$a_k = \sum_{i=0}^{M} w^{(2)}_{kj} z_j \qquad y_k = f(a_k)$$
- Choice of $f(.)$ depends on task
  - Linear for Regression
  - Sigmoid for Binary Classification
  - Softmax for Multiclass Classification

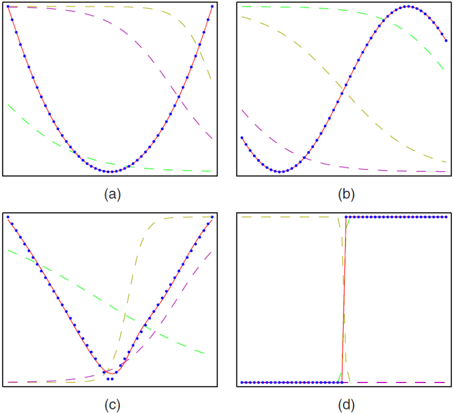
2-Layer Feed Forward Network



- Input: $\mathbf{x} \in \mathbf{R}^D$
- Hidden: $\mathbf{z} = \phi(\mathbf{x}) \in \mathbf{R}^M$
- Output: $\mathbf{y} \in R^K$
- $w_{ji}^{(1)} : x_i \rightarrow z_j$
- $w_{kj}^{(2)} : z_j \rightarrow y_k$

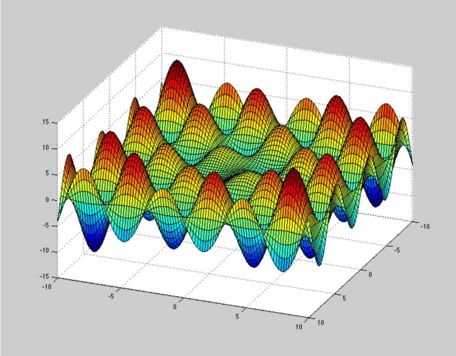
- At j^{th} hidden unit:
$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)$$
- $h(.)$ is a nonlinear function
- At k^{th} output unit:
$$a_k = \sum_{i=0}^{M} w_{kj}^{(2)} z_j \qquad y_k = f(a_k)$$
- Choice of $f(.)$ depends on task
  - Linear for Regression
  - Sigmoid for Binary Classification
  - Softmax for Multiclass Classification
- $y_k = f\left(\sum_{i=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$

Universal Approximator



- (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$ (c) $f(x) = |x|$, (d) $f(x) = sign(x)$
- One hidden layer with 3 tanh(.) units

Weight Space Symmetry



- Error $J(\mathbf{W}) = \mathbb{E}[\|\mathbf{t} - \mathbf{y}\|]$ is nonconvex in $\mathbf{W} = [\mathbf{W}^{(1)}\ \mathbf{W}^{(2)}]$.
- There are $2^M\ M!$ symmetric points with the same error
  - Order of neuronal units in hidden layer does not matter: $M!$
  - Weights leading to and going out of a hidden unit can be negated: 2^M
- Architecture, nonlinearity, loss function and dataset

Parameter Estimation

Parameter Estimation

- Network weights $\mathbf{W} = [\mathbf{W}^{(1)}\ \mathbf{W}^{(2)}]$ have to be adjusted to minimize

$$J(\mathbf{W}) = \sum_{n=1}^{N} J_n(\mathbf{W})$$

$$J_n(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

Parameter Estimation

- Network weights $\mathbf{W} = [\mathbf{W}^{(1)}\ \mathbf{W}^{(2)}]$ have to be adjusted to minimize

$$J(\mathbf{W}) = \sum_{n=1}^{N} J_n(\mathbf{W})$$

$$J_n(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{K} (y_{nk} - t_{nk})^2$$

- Network parameters can be updated using gradient descent

$$\mathbf{W}^{new} = \mathbf{W}^{old} - \eta \nabla J(\mathbf{W})$$

Parameter Estimation

- Network weights $\mathbf{W} = [\mathbf{W}^{(1)}\ \mathbf{W}^{(2)}]$ have to be adjusted to minimize

$$J(\mathbf{W}) = \sum_{n=1}^{N} J_n(\mathbf{W})$$

$$J_n(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

- Network parameters can be updated using gradient descent

$$\mathbf{W}^{new} = \mathbf{W}^{old} - \eta \nabla J(\mathbf{W})$$

- Gradients $\frac{\partial J(\mathbf{W})}{\partial w_{ji}^{(1)}}$ and $\frac{\partial J(\mathbf{W})}{\partial w_{kj}^{(2)}}$ are computed using error backpropagation

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagate Error $\boldsymbol{\delta}$

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}}$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagate Error $\boldsymbol{\delta}$

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}}$$

$$= (y_k - t_k)\, z_j = \delta_k z_j$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagate Error $\boldsymbol{\delta}$

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}}$$

$$= (y_k - t_k)\, z_j = \delta_k z_j$$

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}}$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagate Error $\boldsymbol{\delta}$

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}}$$

$$= (y_k - t_k)\, z_j = \delta_k z_j$$

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}}$$

$$= \sum_{k=1}^{K} (y_k - t_k)\, w_{kj}^{(2)}\, h'(a_j)\, x_i$$

Backpropagation

Forward pass input $\mathbf{x}$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$y_k = f(a_k)$$

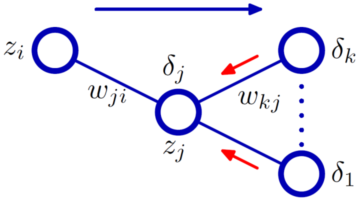
$$J(\mathbf{W}) = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

Backpropagate Error $\boldsymbol{\delta}$

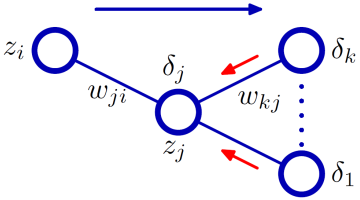
$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}^{(2)}} = (y_k - t_k)\, z_j = \delta_k z_j$$

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}} = \sum_{k=1}^{K} (y_k - t_k)\, w_{kj}^{(2)}\, h'(a_j)\, x_i = \left(h'(a_j) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k \right) x_i$$

Propagation of Inputs and Errors

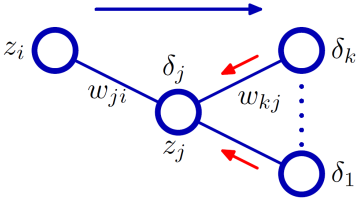


Propagation of Inputs and Errors



- Evaluate the input at the i^{th} node by forward passing the input

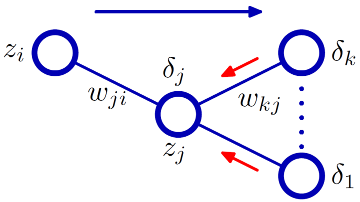
Propagation of Inputs and Errors



- Evaluate the input at the i^{th} node by forward passing the input
- Evaluate the error at the j^{th} node by backpropagating error

$$\delta_j = h'(a_j) \sum_{k=1}^{K} w_{kj} \delta_k$$

Propagation of Inputs and Errors



- Evaluate the input at the i^{th} node by forward passing the input
- Evaluate the error at the j^{th} node by backpropagating error

$$\delta_j = h'(a_j) \sum_{k=1}^{K} w_{kj}\delta_k$$

- Update the weight w_{ji}: $w_{ji}(\tau) = w_{ji}(\tau - 1) - \eta\delta_j z_i$

Weight Updates

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta \, \text{error} \times \text{input}$

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta\, \text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta\, \text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass
  - Evaluate *error* at each node though backpropagation

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta \, \text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass
  - Evaluate *error* at each node though backpropagation
  - Update the weight connecting two nodes using respective input & error

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta\,\text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass
  - Evaluate *error* at each node though backpropagation
  - Update the weight connecting two nodes using respective input & error
- Weight updates for a 2-layer feed FF network

$$w_{kj}^{(2)}(\tau) = w_{kj}^{(2)}(\tau - 1) - \eta_2 \sum_{n \in \mathcal{B}} \delta_{nk} z_{nj}$$

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta \, \text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass
  - Evaluate *error* at each node though backpropagation
  - Update the weight connecting two nodes using respective input & error
- Weight updates for a 2-layer feed FF network

$$w_{kj}^{(2)}(\tau) = w_{kj}^{(2)}(\tau - 1) - \eta_2 \sum_{n \in \mathcal{B}} \delta_{nk} z_{nj}$$

$$w_{ji}^{(1)}(\tau) = w_{ji}^{(1)}(\tau - 1) - \eta_1 \sum_{n \in \mathcal{B}} \left(h'(a_{nj}) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_{nk} \right) x_{ni}$$

Weight Updates

- Typical weight update in GD: $w^{new} = w^{old} - \eta \, \text{error} \times \text{input}$
  - Evaluate *input* at each node though forward pass
  - Evaluate *error* at each node though backpropagation
  - Update the weight connecting two nodes using respective input & error
- Weight updates for a 2-layer feed FF network

$$w_{kj}^{(2)}(\tau) = w_{kj}^{(2)}(\tau - 1) - \eta_2 \sum_{n \in \mathcal{B}} \delta_{nk} z_{nj}$$

$$w_{ji}^{(1)}(\tau) = w_{ji}^{(1)}(\tau - 1) - \eta_1 \sum_{n \in \mathcal{B}} \left(h'(a_{nj}) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_{nk} \right) x_{ni}$$

- $\mathcal{B}$ denotes a random mini-batch of samples drawn from dataset.

Virtues & Limitations of BP

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
    - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
    - Does the gradient always point in the right direction?

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
  - Does the gradient always point in the right direction?
- Major drawback of BP is *slow rate of convergence*

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
  - Does the gradient always point in the right direction?
- Major drawback of BP is *slow rate of convergence*
  - The algorithm operates entirely on the bassis of 1^{st} order statistics

Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
  - Does the gradient always point in the right direction?
- Major drawback of BP is *slow rate of convergence*
  - The algorithm operates entirely on the bassis of 1^{st} order statistics
  - Smaller learning rates are preferred for stable learning

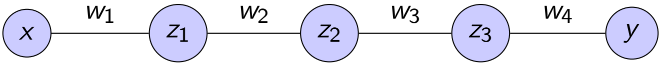
Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
  - Does the gradient always point in the right direction?
- Major drawback of BP is *slow rate of convergence*
  - The algorithm operates entirely on the bassis of 1^{st} order statistics
  - Smaller learning rates are preferred for stable learning
  - Initial layers experience smaller updates (vanishing gradients)

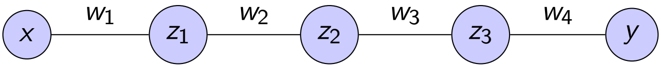
Virtues & Limitations of BP

- BP is the *workhorse* behind the deep learning algorithms
- Elegant in assigning the hidden units their share/responsibility of error
- Linear computational complexity $\mathcal{O}(\mathbf{W})$
  - Weight perturbation requires $\mathcal{O}(\mathbf{W}^2)$ operations
- Gradient descent may get trapped in local minima or saddle-points
  - Does the gradient always point in the right direction?
- Major drawback of BP is *slow rate of convergence*
  - The algorithm operates entirely on the bassis of 1^{st} order statistics
  - Smaller learning rates are preferred for stable learning
  - Initial layers experience smaller updates (vanishing gradients)
  - Weight space dynamics is influenced by weight initialization, batch size, order of presentation, learning rate schedule.

Vanishing Gradients



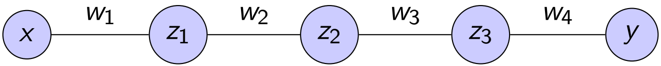
Vanishing Gradients



$$a_1 = w_1 x$$

$$z_1 = h(a_1)$$

Vanishing Gradients



$$a_1 = w_1 x$$

$$z_1 = h(a_1)$$

$$a_2 = w_2 z_1$$

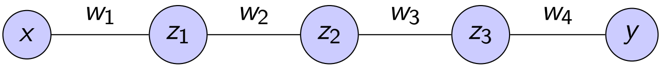
$$z_2 = h(a_2)$$

$$a_3 = w_3 z_2$$

$$z_3 = h(a_3)$$

$$y = w_4 z_3$$

Vanishing Gradients

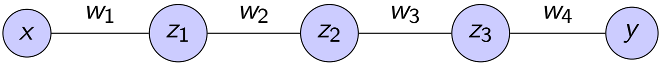


$$
\begin{aligned}
a_1 &= w_1 x \\
z_1 &= h(a_1) \\
a_2 &= w_2 z_1 \\
z_2 &= h(a_2) \\
a_3 &= w_3 z_2 \\
z_3 &= h(a_3) \\
y &= w_4 z_3
\end{aligned}
$$

$$J(\mathbf{w}) = \frac{1}{2}(y - t)^2$$

$$\frac{\partial J(\mathbf{w})}{\partial w_4} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_4} = (y - t)z_3$$

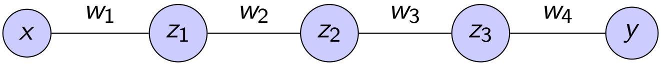
Vanishing Gradients



$$a_1 = w_1 x$$
$$z_1 = h(a_1)$$
$$a_2 = w_2 z_1$$
$$z_2 = h(a_2)$$
$$a_3 = w_3 z_2$$
$$z_3 = h(a_3)$$
$$y = w_4 z_3$$

$$J(\mathbf{w}) = \frac{1}{2}(y - t)^2$$
$$\frac{\partial J(\mathbf{w})}{\partial w_4} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_4} = (y - t)z_3$$
$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

Vanishing Gradients



$$a_1 = w_1 x$$

$$z_1 = h(a_1)$$

$$a_2 = w_2 z_1$$

$$z_2 = h(a_2)$$

$$a_3 = w_3 z_2$$

$$z_3 = h(a_3)$$

$$y = w_4 z_3$$

$$J(\mathbf{w}) = \frac{1}{2}(y - t)^2$$

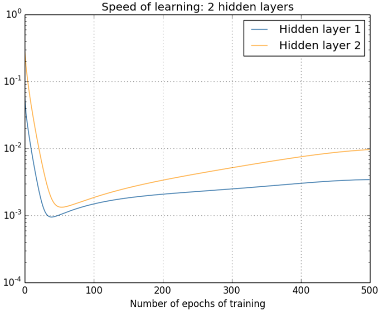
$$\frac{\partial J(\mathbf{w})}{\partial w_4} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_4} = (y - t) z_3$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

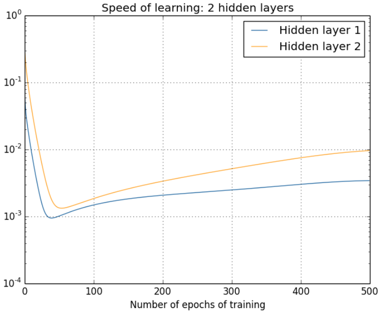
$$= (y - t)\, w_4\, h'(a_3)\, w_3\, h'(a_2)\, w_2\, h'(a_1)\, x$$

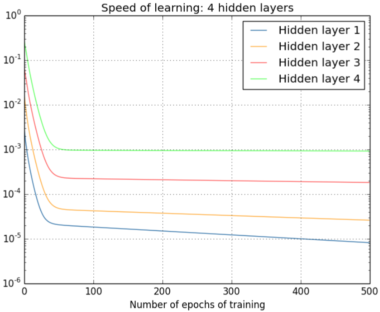
- Initial layer gradients are much smaller

Training Speed

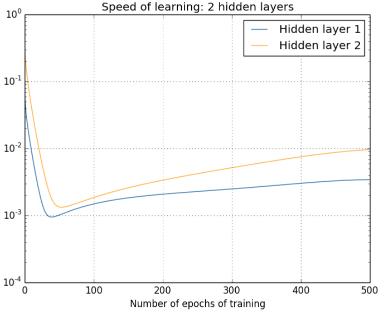


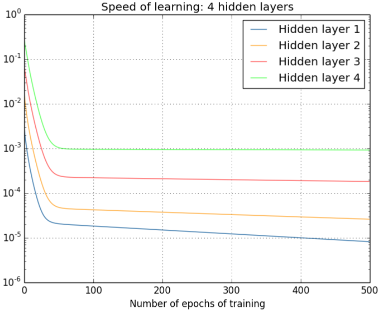
Training Speed





Training Speed

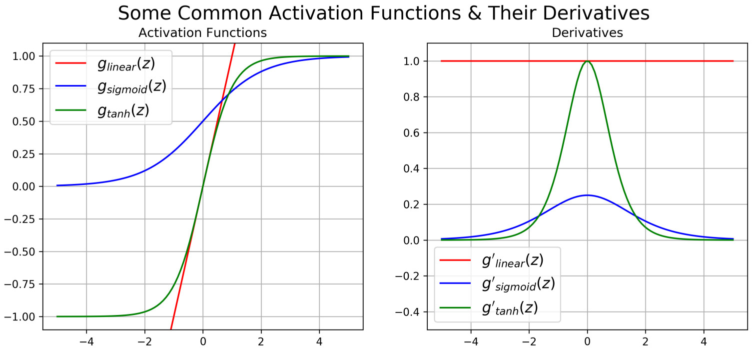




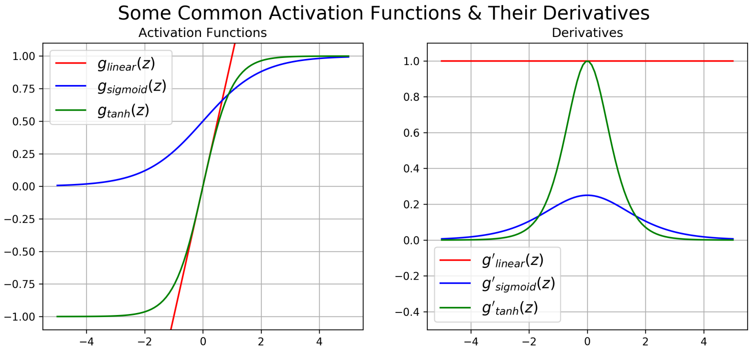
- Training speed is quantified using the norm of the weight update ΔW
- The updates are much smaller for initial layers - hence not trained
- The representations supplied to the deeper layers are not reliable

Derivatives of Activation Functions

Derivatives of Activation Functions

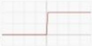


Derivatives of Activation Functions

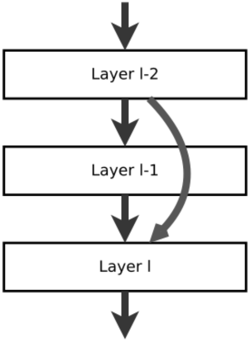


- The magnitude of derivatives of tanh(.) and $\sigma(.)$ are less than 1
- Deep cascade of such activation layers lead to vanishing gradients
- ReLU address this issue as it offers unity gradient for $+$ve values

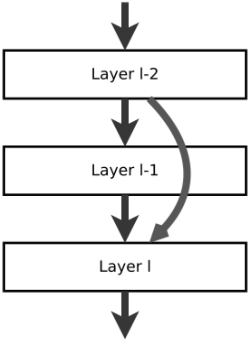
Choice of Nonlinearity

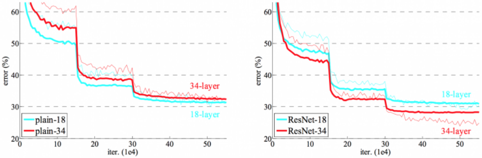
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Residual Networks (ResNet)

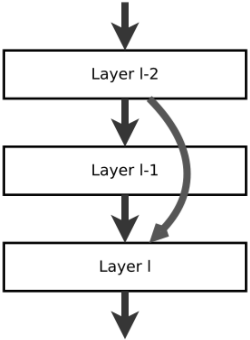


Residual Networks (ResNet)



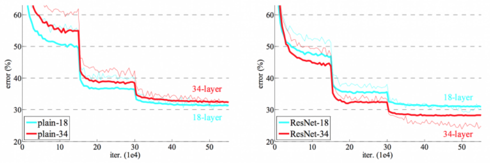


Residual Networks (ResNet)

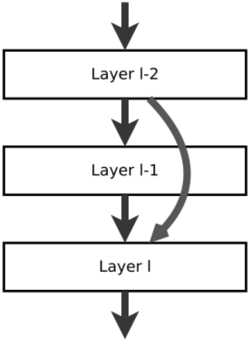


- 18 layer net has lower error than 34 layer net!
- Skip-connections to overcome vanishing grd.

$$\mathbf{z}^{(l)} = h\left(\mathbf{W}^{(l-1,l)}\mathbf{z}^{(l-1)} + \mathbf{W}^{(l-2,l)}\mathbf{z}^{(l-2)}\right)$$



Residual Networks (ResNet)

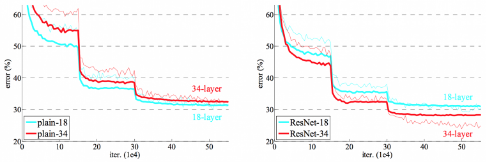


- 18 layer net has lower error than 34 layer net!
- Skip-connections to overcome vanishing grd.

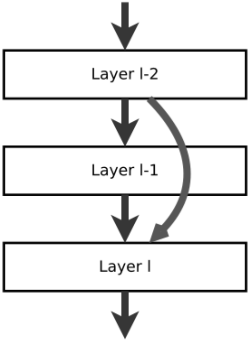
$$\mathbf{z}^{(l)} = h\left(\mathbf{W}^{(l-1,l)}\mathbf{z}^{(l-1)} + \mathbf{W}^{(l-2,l)}\mathbf{z}^{(l-2)}\right)$$

- If there are no weights on skip connections

$$\mathbf{z}^{(l)} = h\left(g(\mathbf{z}^{(l-2)}) + \mathbf{z}^{(l-2)}\right)$$



Residual Networks (ResNet)



- 18 layer net has lower error than 34 layer net!
- Skip-connections to overcome vanishing grd.

$$\mathbf{z}^{(l)} = h\left(\mathbf{W}^{(l-1,l)}\mathbf{z}^{(l-1)} + \mathbf{W}^{(l-2,l)}\mathbf{z}^{(l-2)}\right)$$

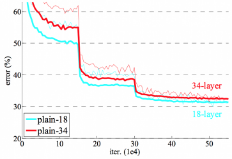
- If there are no weights on skip connections

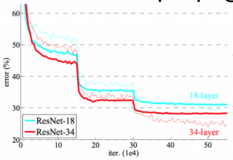
$$\mathbf{z}^{(l)} = h\left(g(\mathbf{z}^{(l-2)}) + \mathbf{z}^{(l-2)}\right)$$

- Backpropagation

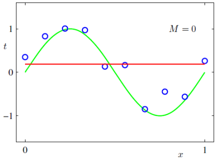
$$\Delta\mathbf{W}^{(l-1,l)} = -\eta\, \mathbf{z}^{(l-1)}\, \boldsymbol{\delta}_l$$

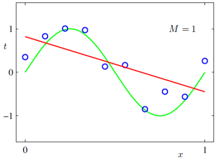
$$\Delta\mathbf{W}^{(l-2,l)} = -\eta\, \mathbf{z}^{(l-2)}\, \boldsymbol{\delta}_l$$



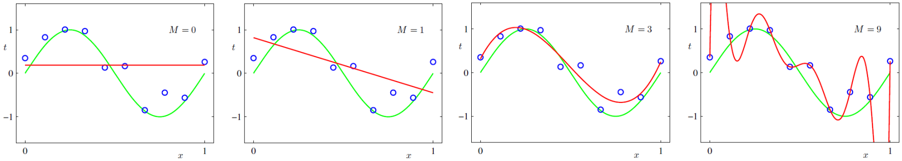


Overfitting

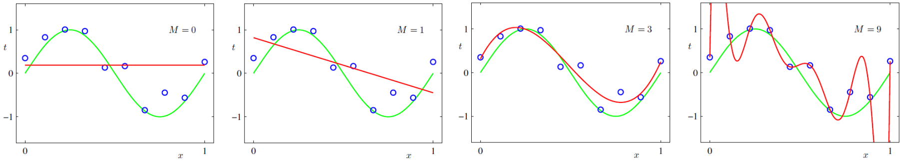


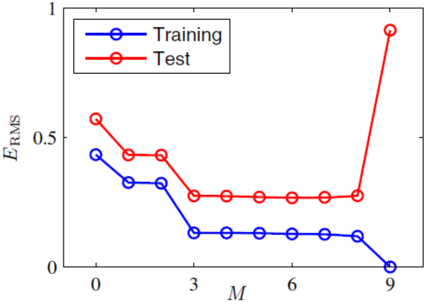


Overfitting

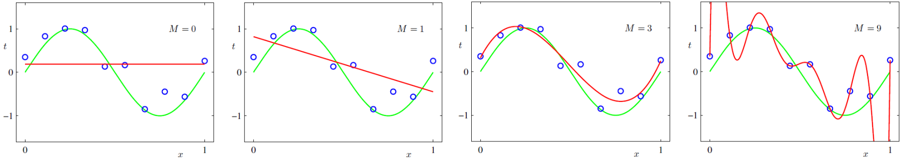


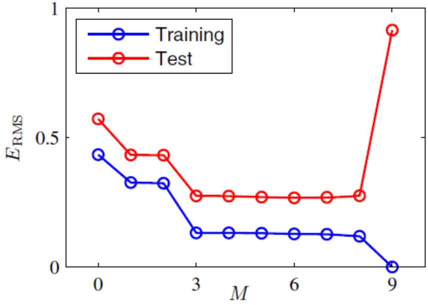
Overfitting





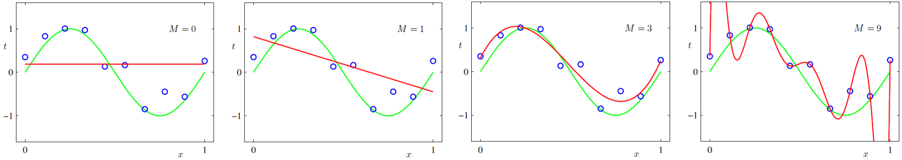
Overfitting

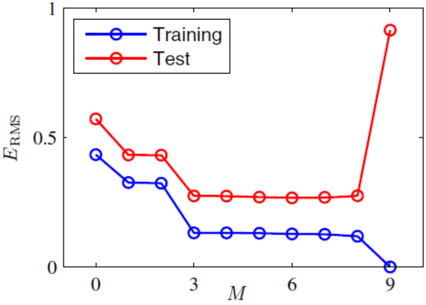




	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Overfitting





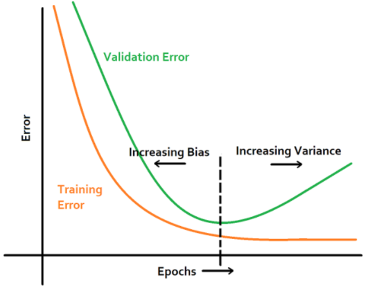
	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Regularization Techniques

- DNNs with large number of parameters tend to overfit
  - Error on training data reduces, but not on validation/test data
  - Performance on test data could be worse than smaller models (paradox)
  - With larger parameters, the model gets tuned to noise in training data
  - Weights blow up to capture the noisy fluctuations
- Arrest the growth of the weights to avoid overfitting to training data
- Finding the right balance between bias and variance trade-off.
- Common regularization techniques for DNNs include:
  - Early Stopping, explicit weight regularization, activity regularization/constraints, dropout, input corruption with noise

Early Stopping

- Initialize the weights with small random values
  - Glorot Normal - $w_{ji} \in \mathcal{N}\left(0, \frac{\sqrt{2}}{f_{in}+f_{out}}\right)$
  - Update the weights using backpropagation algorithm
- Monitor training and validation errors after every epoch.
- Stop the training when validation error starts to increase

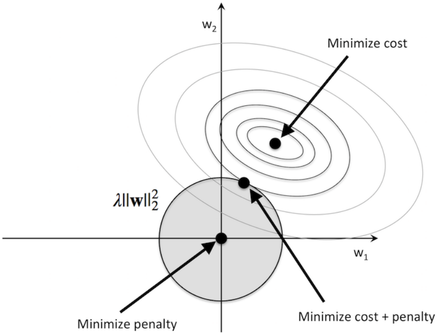


Weight Regularization

- Add a penalty term to the error function to discourage weight growth

$$J(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_{nk} - d_{nk})^2 + \lambda\|\mathbf{W}\|_p$$

- λ controls the trade-off between bias and variance

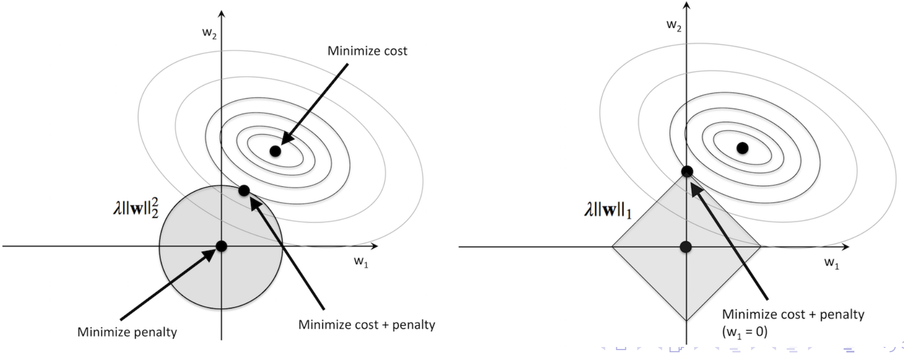


Weight Regularization

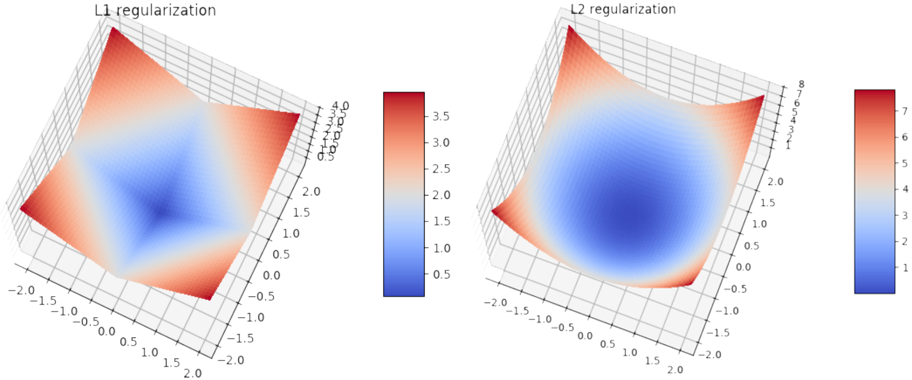
- Add a penalty term to the error function to discourage weight growth

$$J(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_{nk} - d_{nk})^2 + \lambda\|\mathbf{W}\|_p$$

- λ controls the trade-off between bias and variance



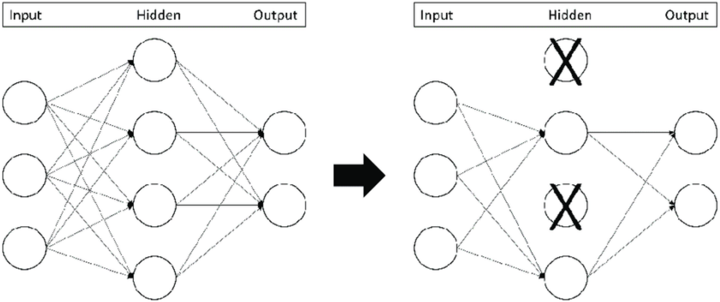
L_1 vs L_2



- L_1 regularization promotes sparser solutions
- L_1 regularization $\implies$ Laplacian priors
- L_2 regularization $\implies$ Gaussian priors

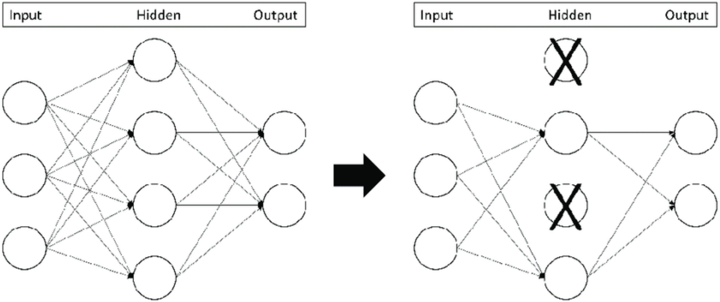
Dropout

Dropout



- Some nodes may tend to be too critical in network operation
- Avoid such situation by sharing the responsibility across the nodes

Dropout



- Some nodes may tend to be too critical in network operation
- Avoid such situation by sharing the responsibility across the nodes
- Drop nodes in the hidden layers with a probability $p{=}(0.5)$
- With M hidden units, it can create 2^M different network architectures

Dropout Implementation

- Network training
  - Forwardpass the input to evaluate neuronal outputs in the hidden layer
    $$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)\, m_j$$
  - m_j is the mask drawn from a Bernoulli distribution with parameter p
  - Backpropagate the error to evaluate the share of the hidden neuron
    $$\delta_j = m_j\, h'(a_j) \textstyle\sum_{k=1}^{K} w_{kj} \delta_k$$

Dropout Implementation

- Network training
  - Forwardpass the input to evaluate neuronal outputs in the hidden layer
  $$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)\, m_j$$
  - m_j is the mask drawn from a Bernoulli distribution with parameter p
  - Backpropagate the error to evaluate the share of the hidden neuron
  $$\delta_j = m_j\, h'(a_j) \sum_{k=1}^{K} w_{kj} \delta_k$$
- Inference from the network
  - Evaluate the network output for different dropouts & combine them
  - Use expected neuronal activation $z_j\, p_j$ to infer the network output
  - Expected neuronal activation $\implies$ GM over 2^M configurations

Dropout Implementation

- Network training
  - Forwardpass the input to evaluate neuronal outputs in the hidden layer

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad z_j = h(a_j)\, m_j$$

  - m_j is the mask drawn from a Bernoulli distribution with parameter p
  - Backpropagate the error to evaluate the share of the hidden neuron

$$\delta_j = m_j\, h'(a_j) \sum_{k=1}^{K} w_{kj}\delta_k$$

- Inference from the network
  - Evaluate the network output for different dropouts & combine them
  - Use expected neuronal activation $z_j\, p_j$ to infer the network output
  - Expected neuronal activation $\implies$ GM over 2^M configurations
- Interpretation of Dropout
  - Dropout can be interpreted as mixture of experts (novel combination)
  - A node is expected to perform in different configurations: Regularization

Limitations for Fully Connected (Dense) Networks

- FFNN offers data-dependent nonlinear transformation

$$\mathbf{z}[n] = \phi(\mathbf{x}[\boldsymbol{n}]) = g(\mathbf{W}^{(1)}\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}^{(2)}\mathbf{z}[n])$$

Limitations for Fully Connected (Dense) Networks

- FFNN offers data-dependent nonlinear transformation

$$\mathbf{z}[n] = \phi(\mathbf{x}[\boldsymbol{n}]) = g(\mathbf{W}^{(1)}\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}^{(2)}\mathbf{z}[n])$$

- FFNNs are memoryless models
  - Transformed representation depends only on current input
  - Impossible to choose a fixed-length window - varying context

Limitations for Fully Connected (Dense) Networks

- FFNN offers data-dependent nonlinear transformation

$$\mathbf{z}[n] = \phi(\mathbf{x}[\boldsymbol{n}]) = g(\mathbf{W}^{(1)}\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}^{(2)}\mathbf{z}[n])$$

- FFNNs are memoryless models
  - Transformed representation depends only on current input
  - Impossible to choose a fixed-length window - varying context
  - FFNNs are not capable of capturing the sequential information
  - Need to explore nonlinear sequential models for signal processing

Limitations for Fully Connected (Dense) Networks

- FFNN offers data-dependent nonlinear transformation

$$\mathbf{z}[n] = \phi(\mathbf{x}[n]) = g(\mathbf{W}^{(1)}\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}^{(2)}\mathbf{z}[n])$$

- FFNNs are memoryless models
    - Transformed representation depends only on current input
    - Impossible to choose a fixed-length window - varying context
    - FFNNs are not capable of capturing the sequential information
    - Need to explore nonlinear sequential models for signal processing
- Incorporate sequential information in the transformed representation
    - Finite memory: $\mathbf{z}[n] = \phi(\mathbf{x}[n-k] : \mathbf{x}[n+k])$ CNNs
    - Infinite memory: $\mathbf{z}[n] = \phi(\mathbf{x}[-\infty] : \mathbf{x}[\infty])$ RNNs