(Deep) Neural Networks

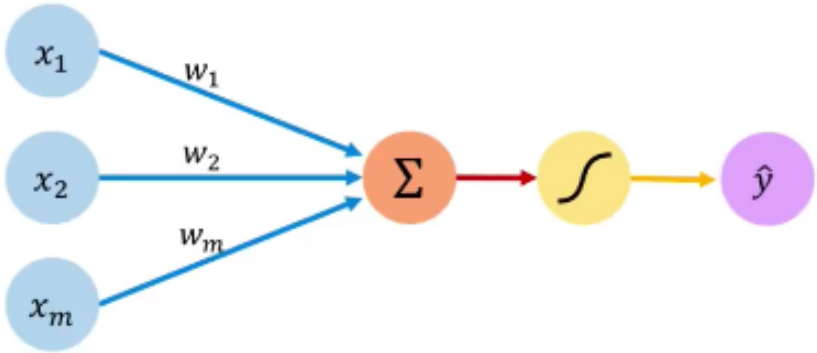
K Sri Rama Murty

IIT Hyderabad

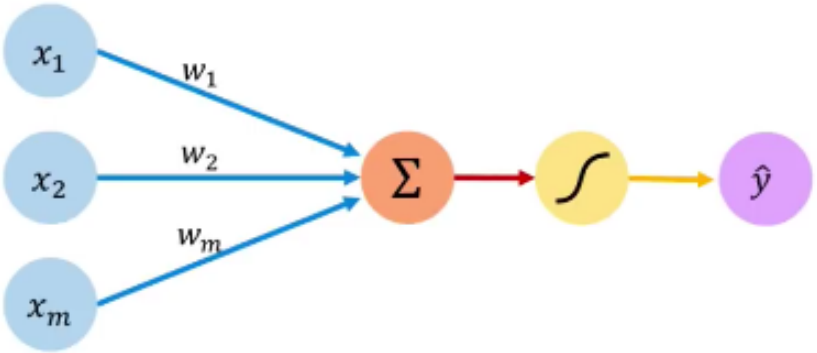
ksrm@ee.iith.ac.in

April 25, 2021

Summary of Linear/Logistic Regression

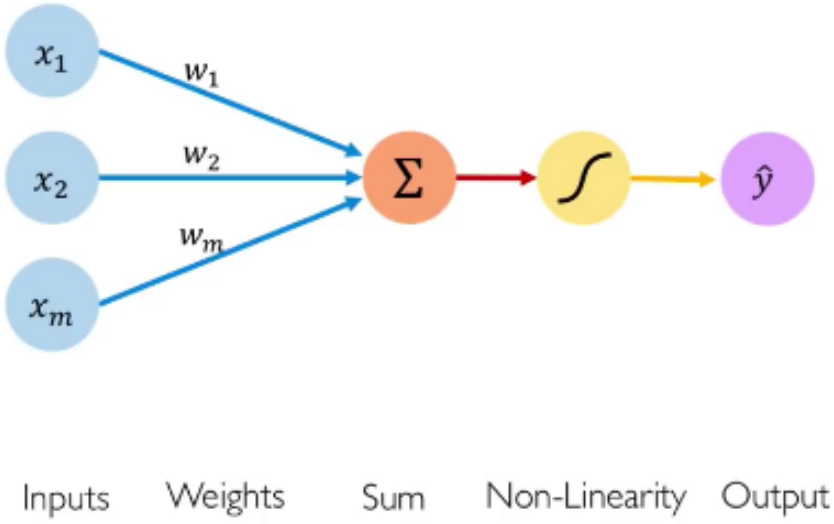


Summary of Linear/Logistic Regression



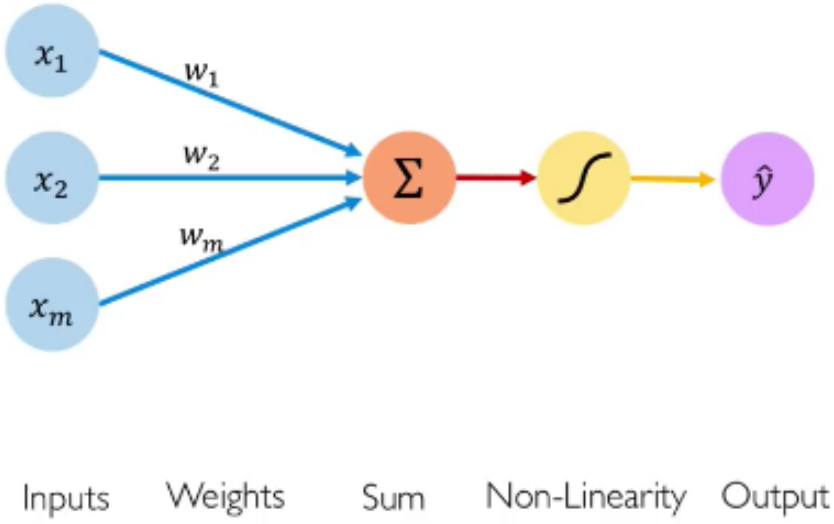
- Pass linear aggregated input though activation function

Summary of Linear/Logistic Regression



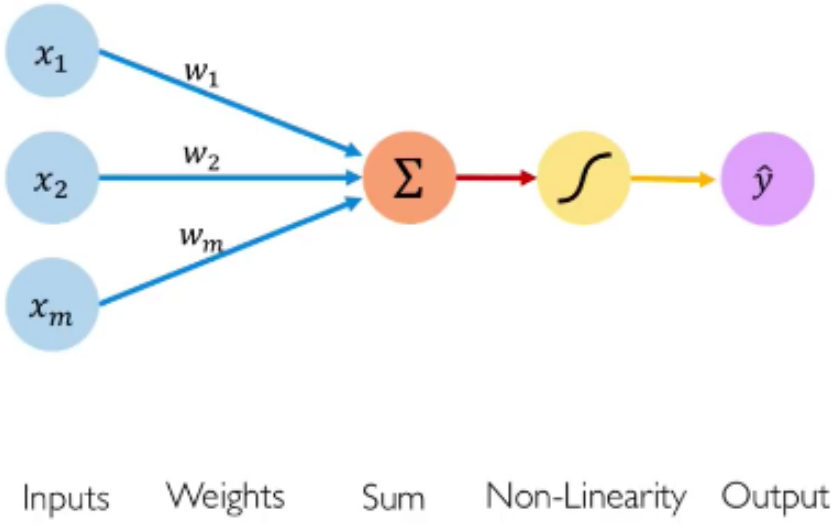
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$

Summary of Linear/Logistic Regression



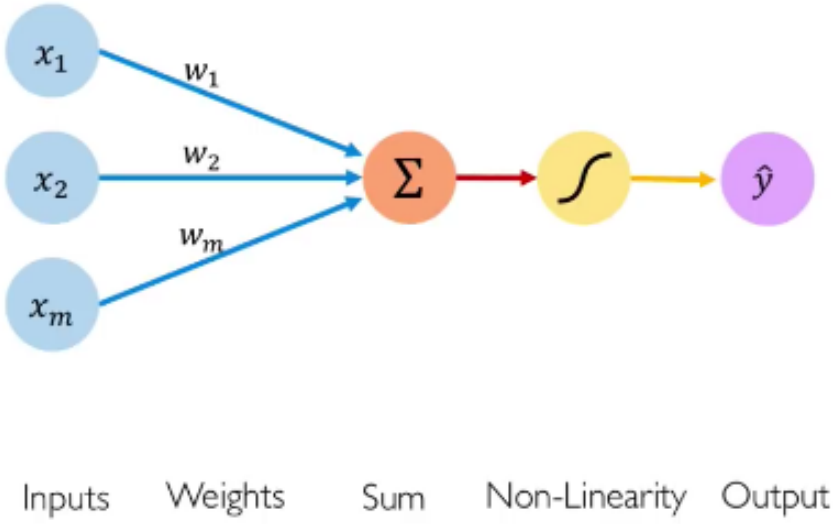
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression

Summary of Linear/Logistic Regression



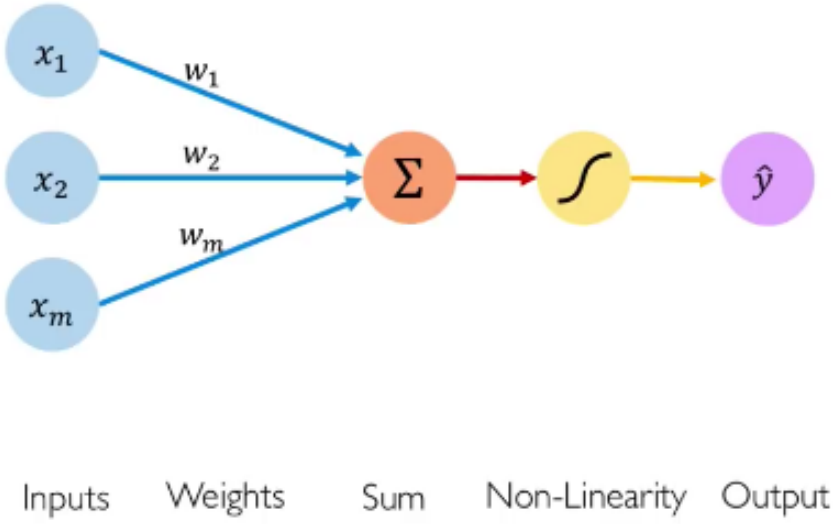
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression

Summary of Linear/Logistic Regression



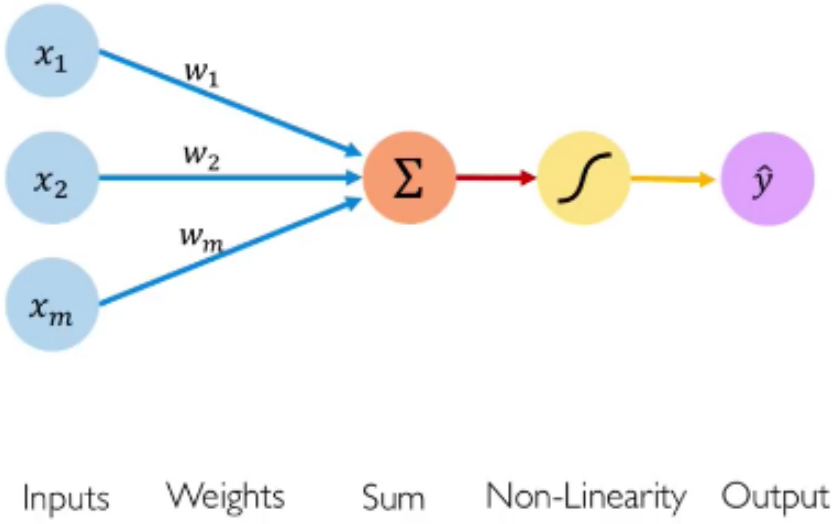
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression
  - f(.) is softmax: Multiclass logistic regression

Summary of Linear/Logistic Regression



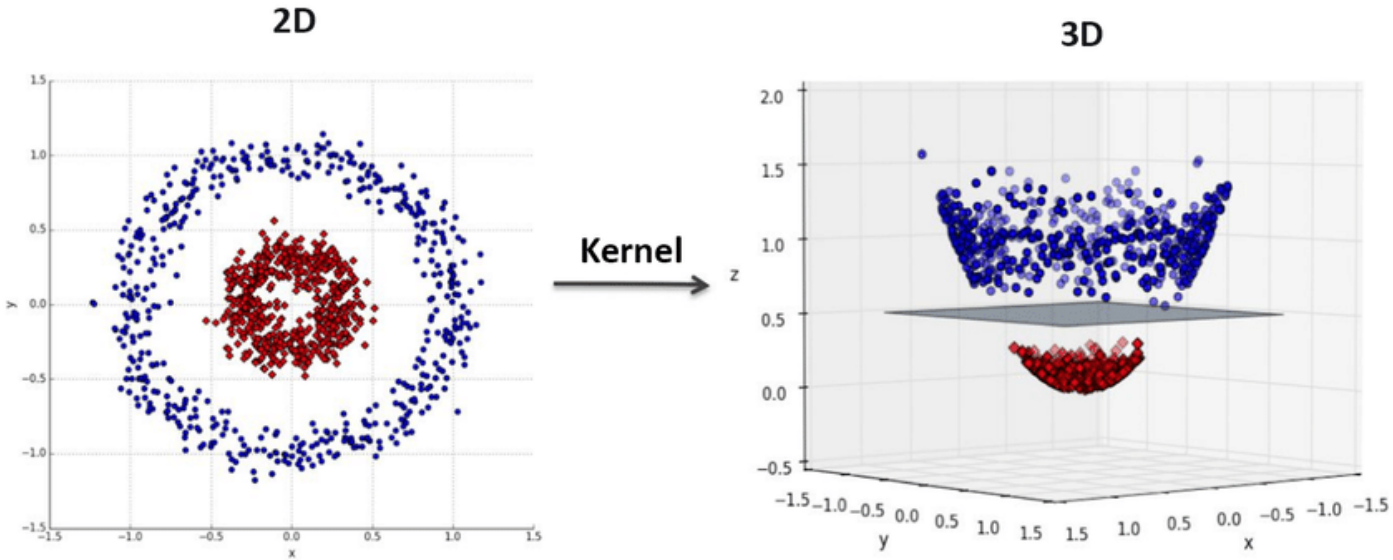
- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression
  - f(.) is softmax: Multiclass logistic regression
  - f(.) is hard-limiting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

Summary of Linear/Logistic Regression

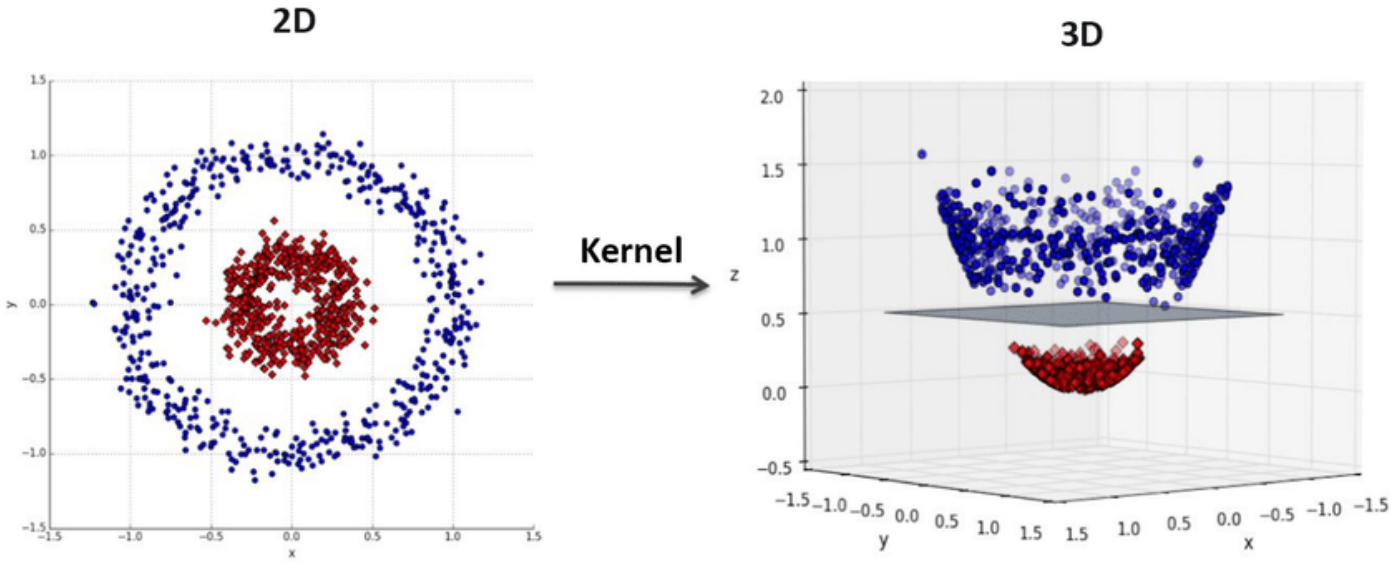


- Pass linear aggregated input though activation function
  - Output $\hat{t} = y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i=1}^{D} w_i x_i\right)$
  - f(.) is linear: Linear regression
  - f(.) is sigmoid: Logistic regression
  - f(.) is softmax: Multiclass logistic regression
  - f(.) is hard-limiting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

Cover's Theorem

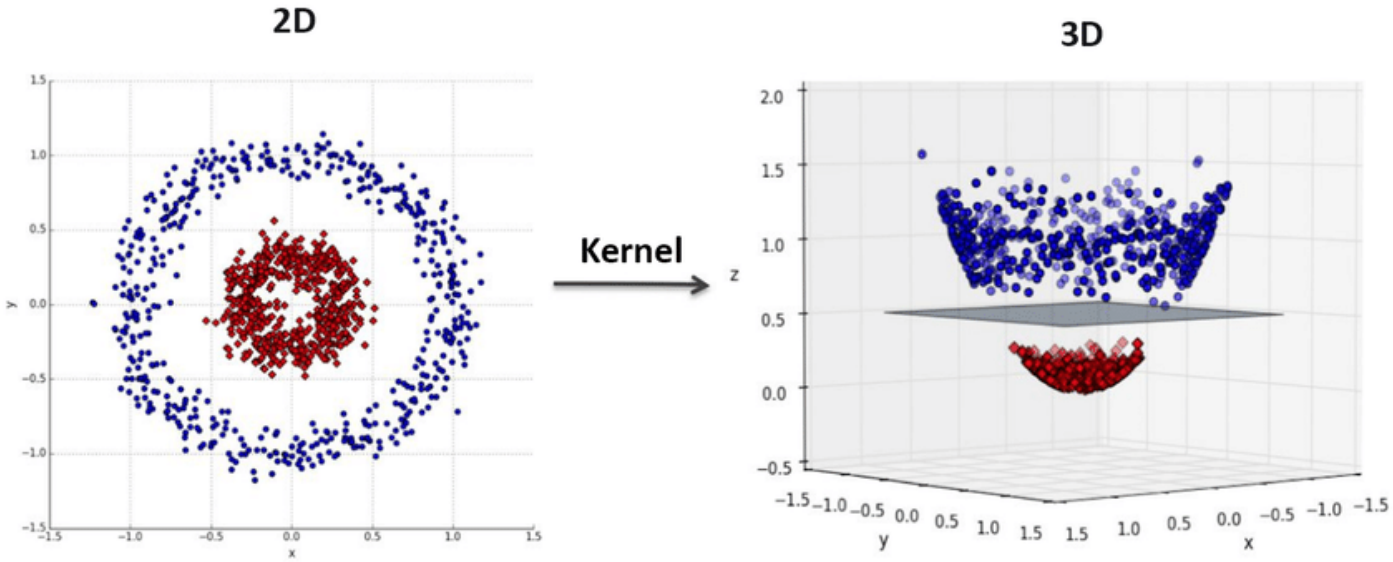


Cover's Theorem



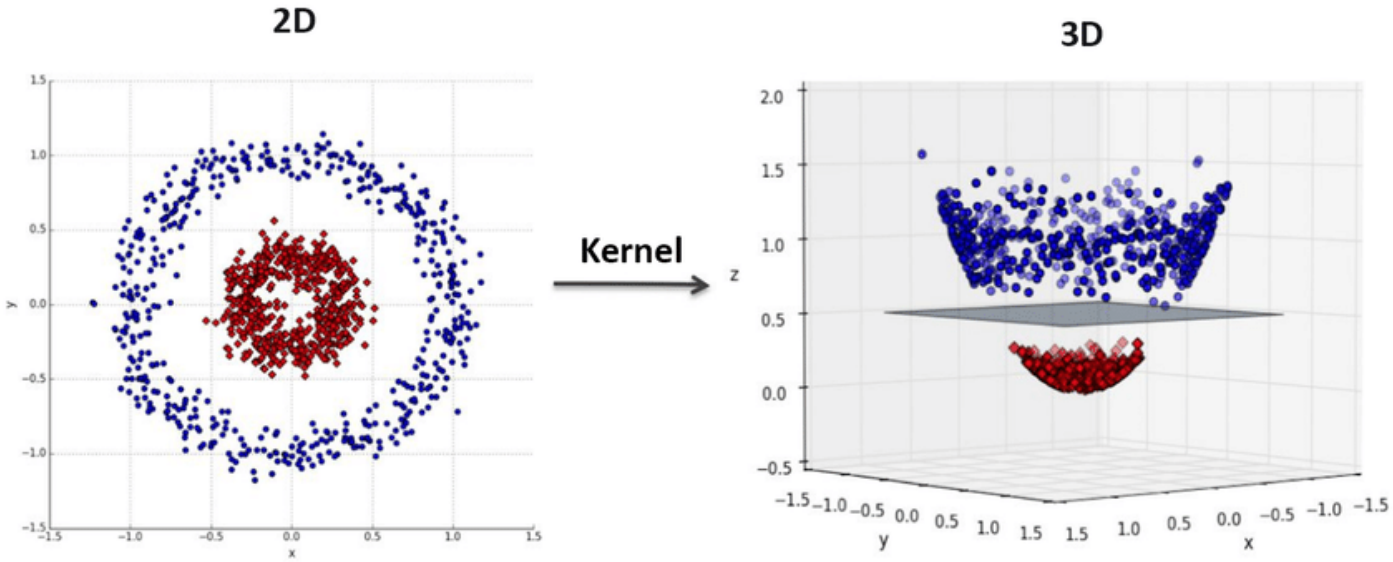
- A complex pattern classification problem can be transformed to a linearly seprabale one, provided

Cover's Theorem



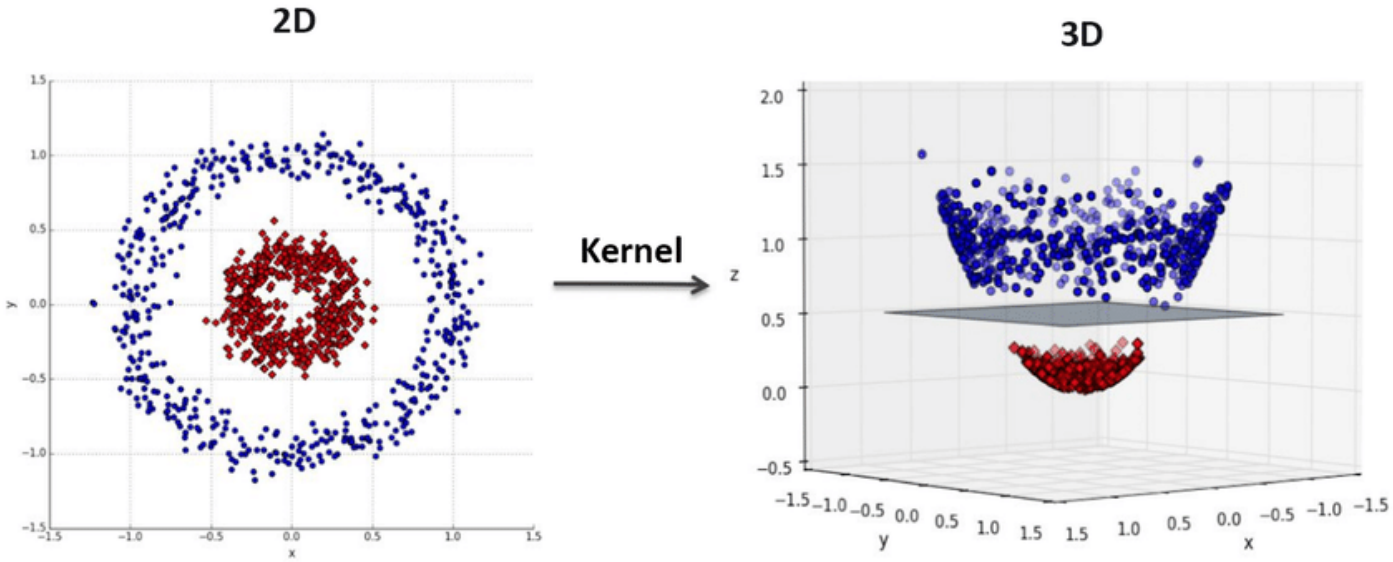
- A complex pattern classification problem can be transformed to a linearly seprabale one, provided
  - The transformation from input space to feature space is nonlinear

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 $c = 1$: $k(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + 1)^2$
  - $k(\mathbf{x}, \mathbf{y}) = \phi^{\mathsf{T}}(\mathbf{x})\phi(\mathbf{y})$ 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

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 $c = 1$: $k(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + 1)^2$
  - $k(\mathbf{x}, \mathbf{y}) = \phi^{\mathsf{T}}(\mathbf{x})\phi(\mathbf{y})$ 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
- 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 $\mathbf{x} \in \mathbb{R}^D \rightarrow \boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$

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

Data-Dependent Transformation

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

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

- 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 $\mathbf{x} \in \mathbb{R}^D \rightarrow \boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$

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

- 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 $\mathbf{x} \in \mathbb{R}^D \rightarrow \boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$

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

- 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 $\mathbf{x} \in \mathbb{R}^D \rightarrow \boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$

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

- 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 $\mathbf{x} \in \mathbb{R}^D \rightarrow \boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$

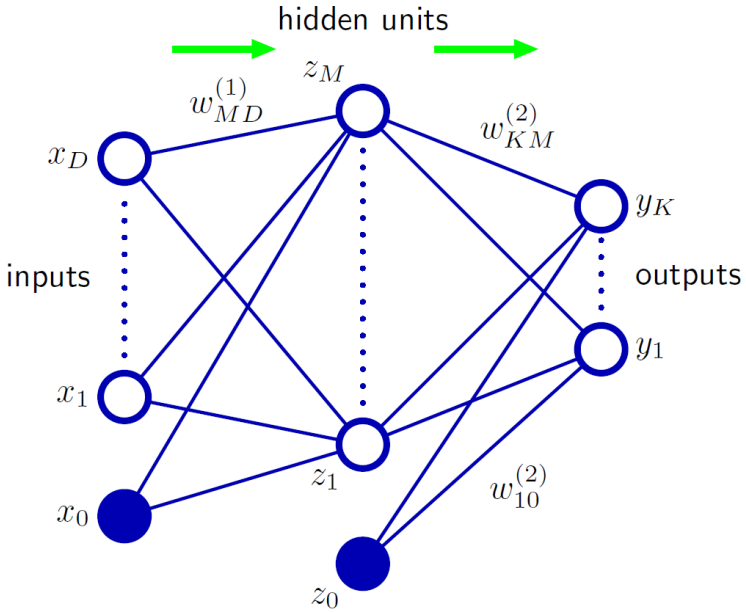
$$\hat{t} = f\left(\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}[n])\right) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}[n])\right)$$

- 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

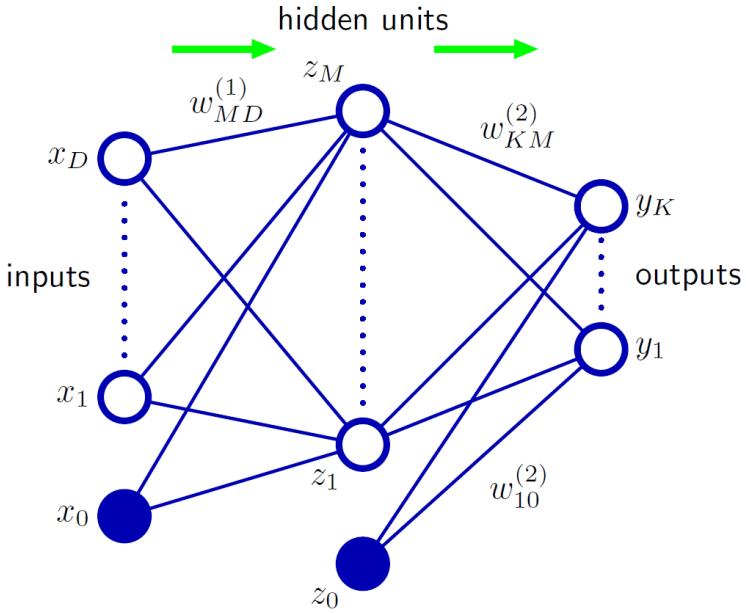
$$\phi_j(\mathbf{x}) = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i\right) \qquad j = 1, 2 \cdots, M$$

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

2-Layer Feed Forward Network

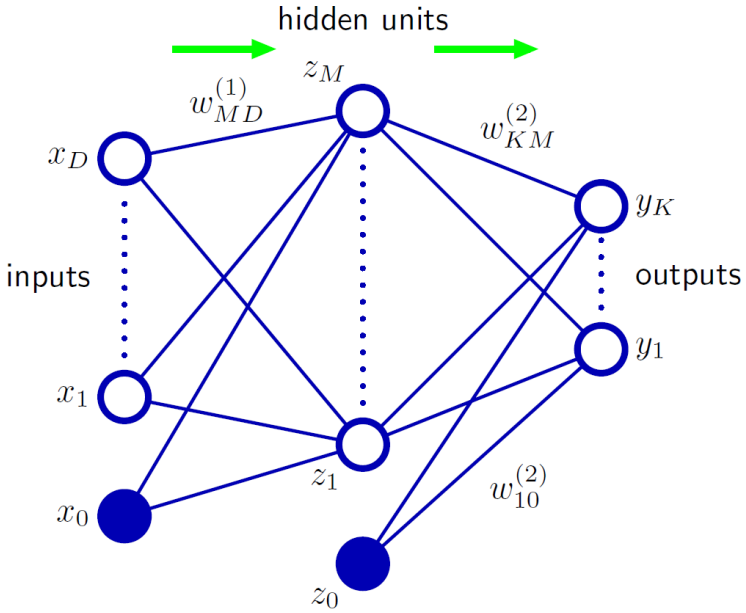


2-Layer Feed Forward Network



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

2-Layer Feed Forward Network

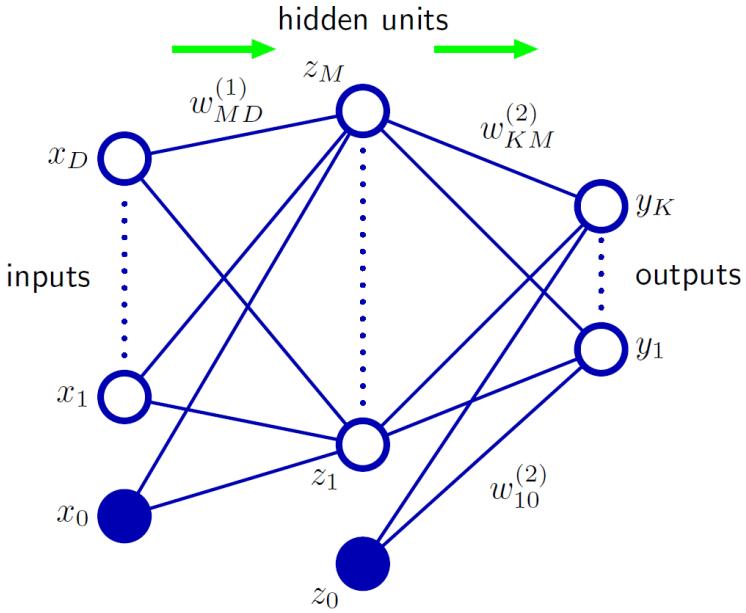


- At j^{th} hidden unit:

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

- 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



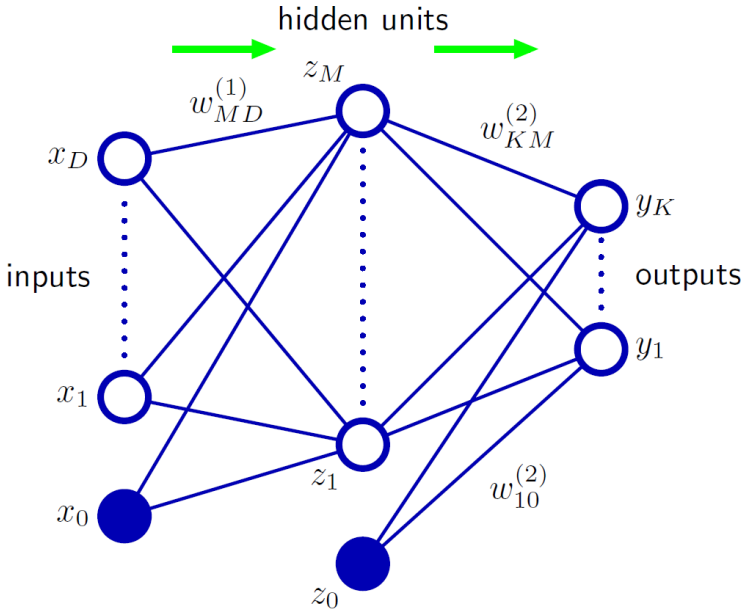
- At j^{th} hidden unit:

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

- $h(.)$ 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: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

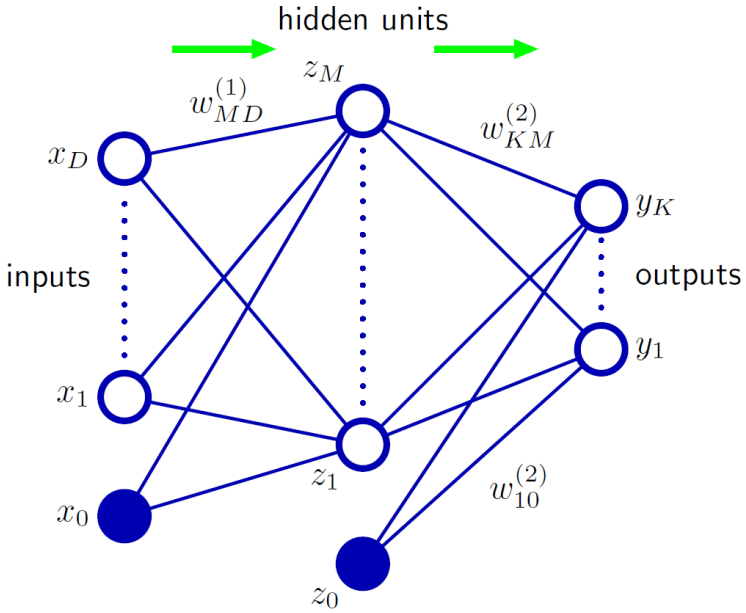
- At j^{th} hidden unit:

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

- $h(.)$ 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:

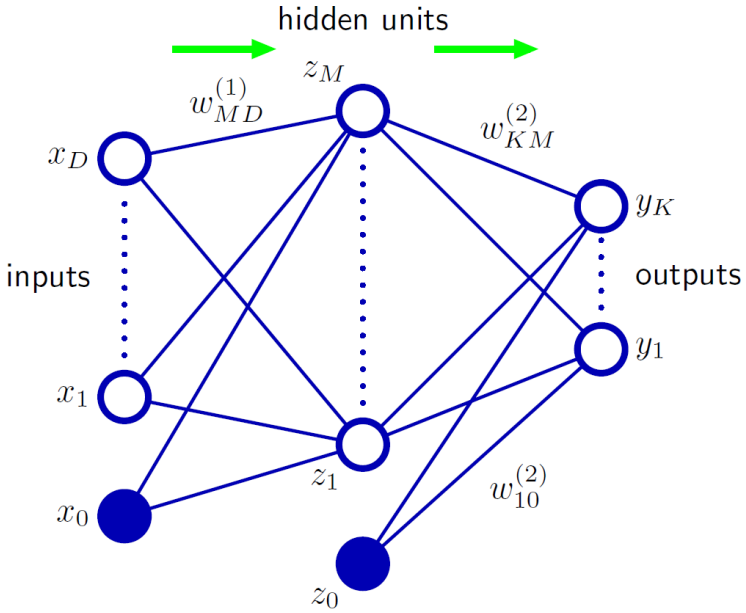
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

- $h(.)$ 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: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:

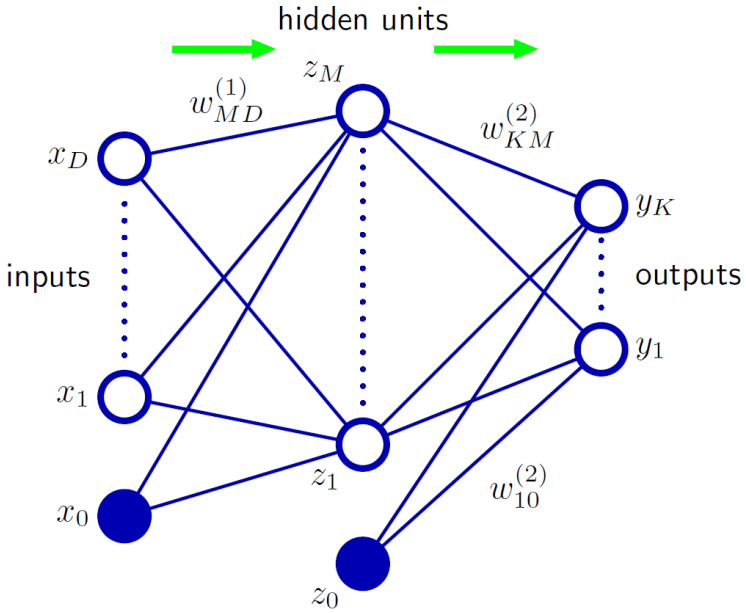
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

- $h(.)$ 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

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:

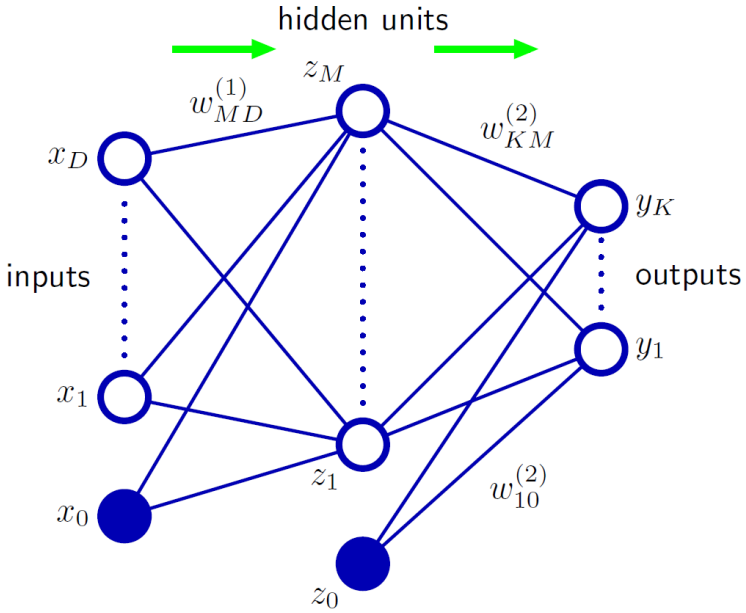
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

- $h(.)$ 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:

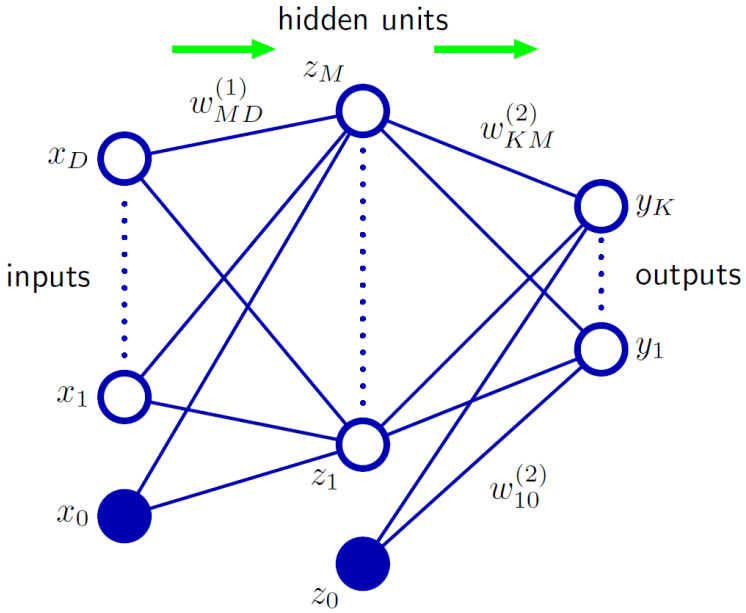
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

- $h(.)$ 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: $\mathbf{y} \in R^K$
- $w^{(1)}_{ji} : x_i \rightarrow z_j$
- $w^{(2)}_{kj} : z_j \rightarrow y_k$

- At j^{th} hidden unit:

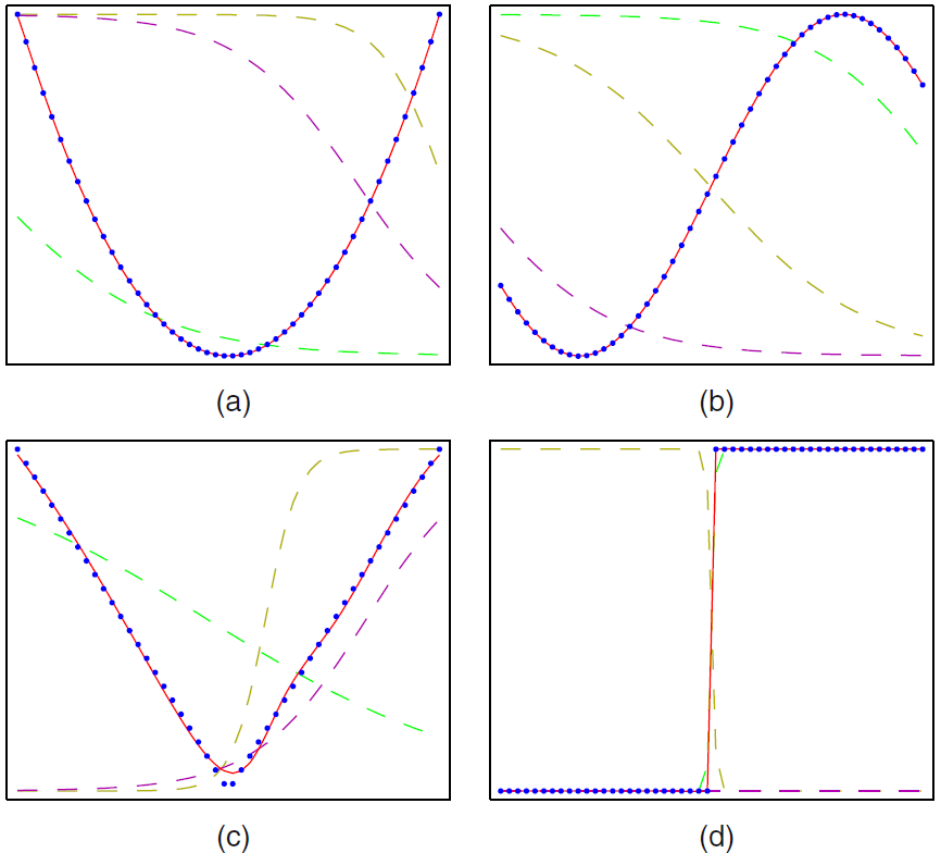
$$a_j = \sum_{i=0}^{D} w^{(1)}_{ji} x_i \qquad z_j = h(a_j)$$

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

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

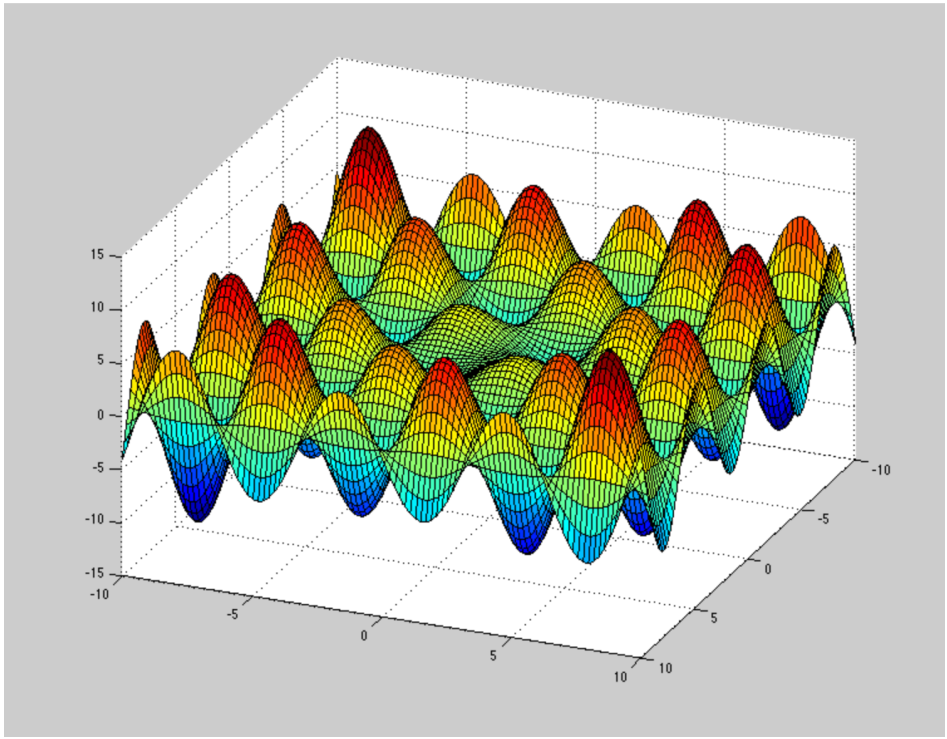
- Choice of $f(.)$ depends on task
  - Linear for Regression
  - Sigmoid for Binary Classification
  - Softmax for Multiclass Classification
- $y_k = f\left(\sum_{i=0}^{M} w^{(2)}_{kj} h\left(\sum_{i=0}^{D} w^{(1)}_{ji} x_i\right)\right)$

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 $\boldsymbol{\delta}$

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

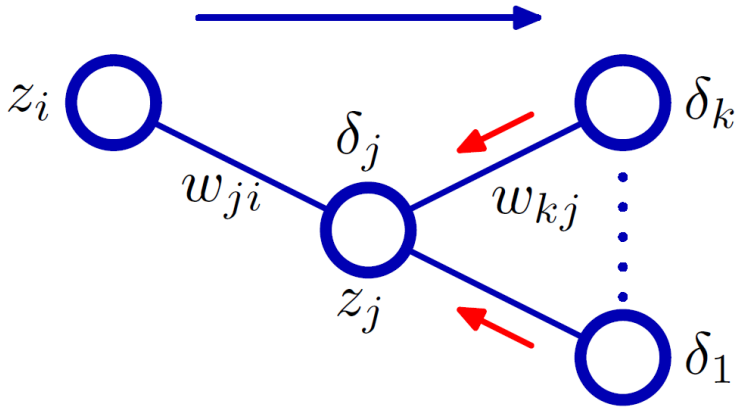
$$= (y_k - t_k)\, z_j = \delta_k z_j$$

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

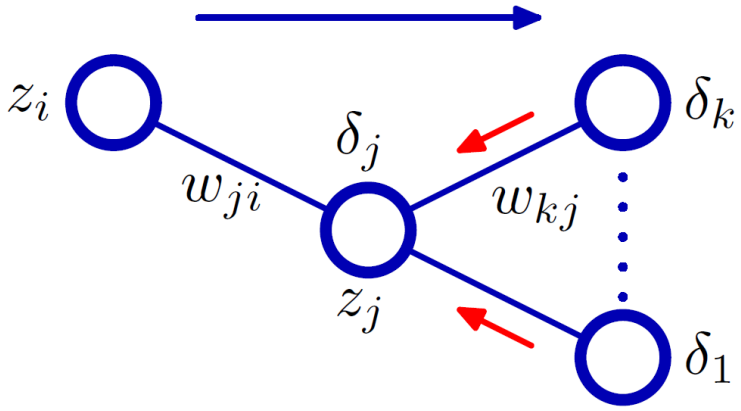
$$= \sum_{k=1}^{K} (y_k - t_k)\, w_{kj}^{(2)}\, h'(a_j)\, x_i$$

$$= \left(h'(a_j) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k \right) x_i$$

Propagation of Inputs and Errors

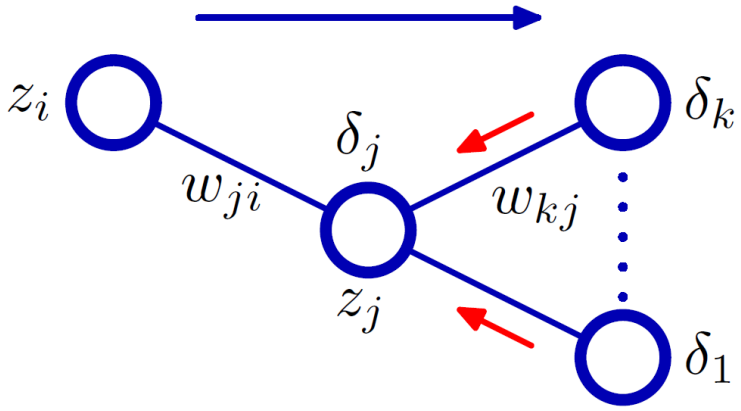


Propagation of Inputs and Errors



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

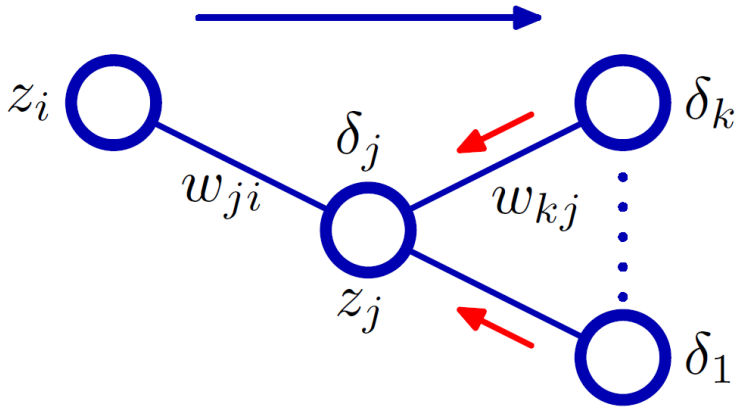
Propagation of Inputs and Errors



- 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: $w^{new} = w^{old} - \eta$ error $\times$ input

Weight Updates

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

Weight Updates

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

Weight Updates

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

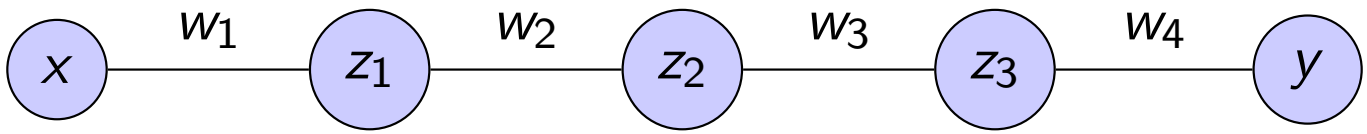
Virtues & Limitations of BP

- 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)

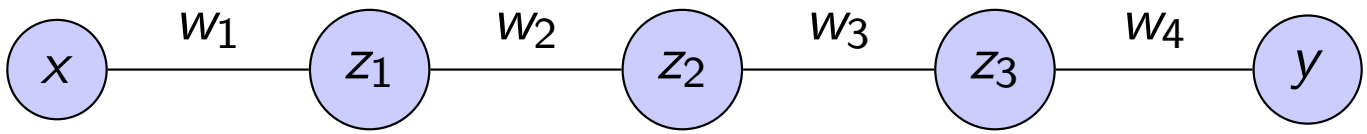
Virtues & Limitations of BP

- 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



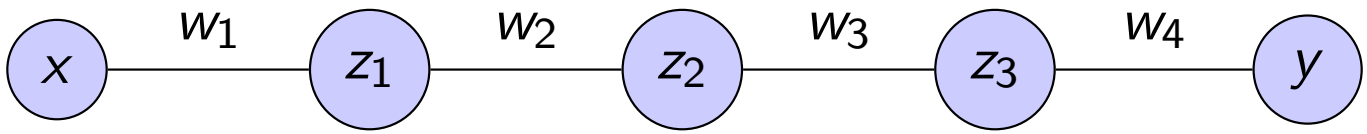
Vanishing Gradients



$$a_1 = w_1 x$$

$$z_1 = h(a_1)$$

Vanishing Gradients



$$a_1 = w_1 x$$

$$z_1 = h(a_1)$$

$$a_2 = w_2 z_1$$

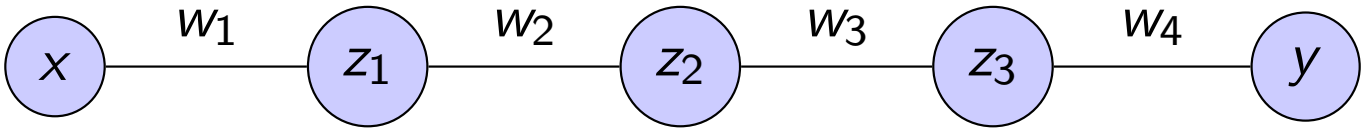
$$z_2 = h(a_2)$$

$$a_3 = w_3 z_2$$

$$z_3 = h(a_3)$$

$$y = w_4 z_3$$

Vanishing Gradients

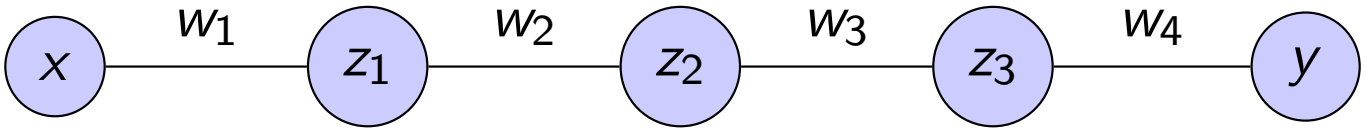


$$a_1 = w_1 x$$
$$z_1 = h(a_1)$$
$$a_2 = w_2 z_1$$
$$z_2 = h(a_2)$$
$$a_3 = w_3 z_2$$
$$z_3 = h(a_3)$$
$$y = w_4 z_3$$

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

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

Vanishing Gradients



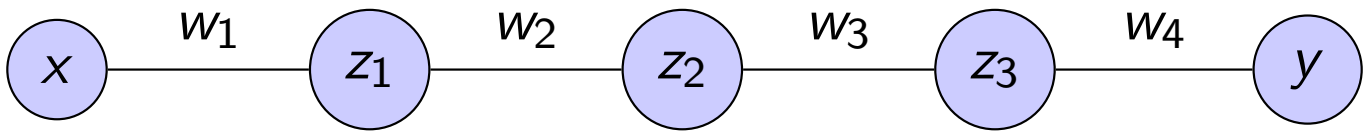
$$a_1 = w_1 x$$
$$z_1 = h(a_1)$$
$$a_2 = w_2 z_1$$
$$z_2 = h(a_2)$$
$$a_3 = w_3 z_2$$
$$z_3 = h(a_3)$$
$$y = w_4 z_3$$

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

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

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

Vanishing Gradients



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

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

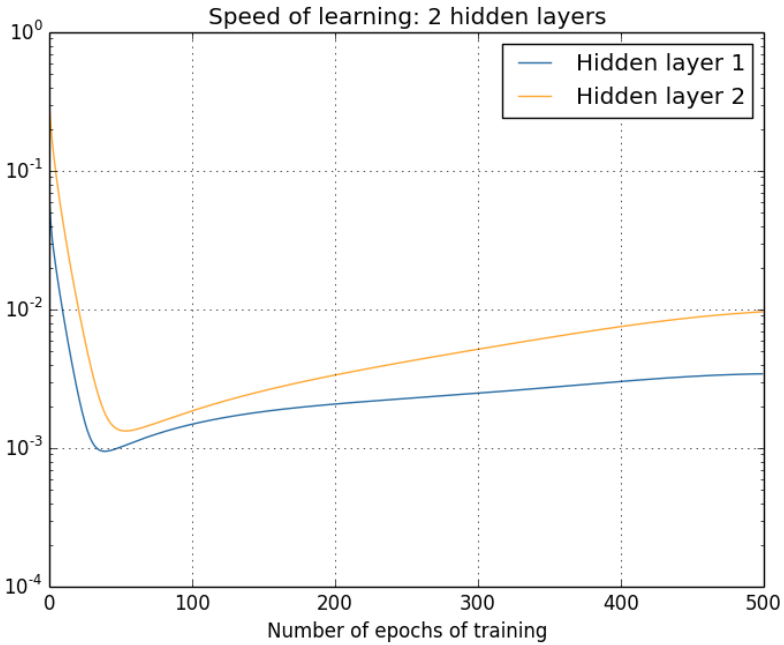
$$\frac{\partial J(\mathbf{w})}{\partial w_4} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_4} = (y-t)z_3$$

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

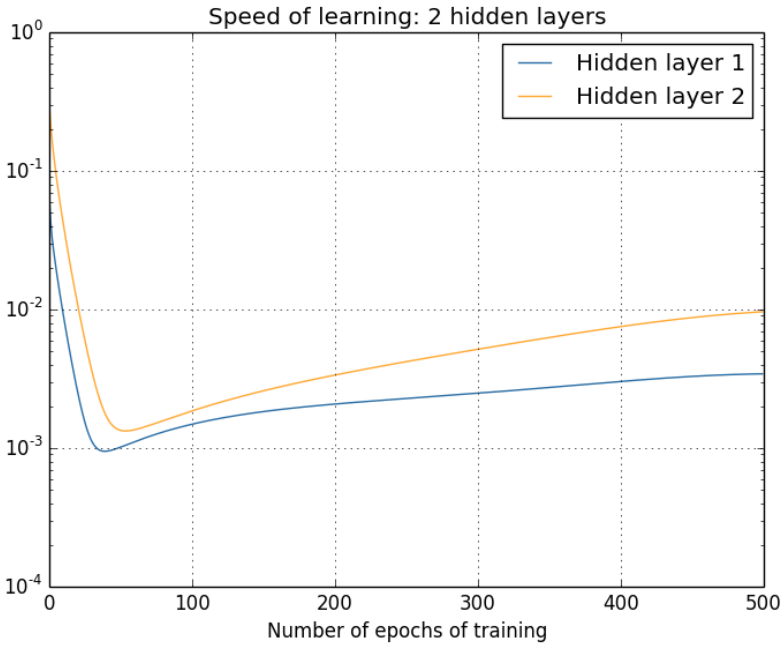
$$= (y-t)\, w_4\, h'(a_3)\, w_3\, h'(a_2)\, w_2\, h'(a_1)\, x$$

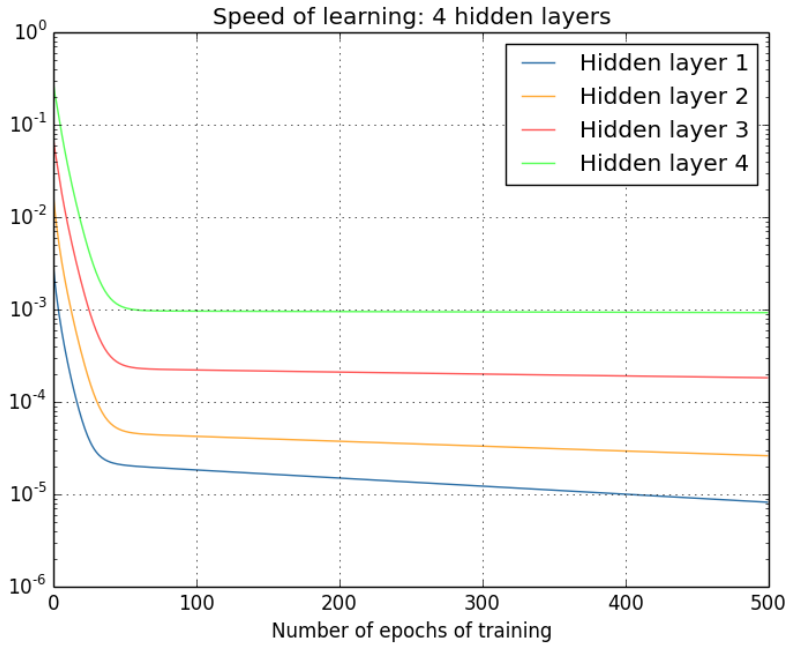
- Initial layer gradients are much smaller

Training Speed

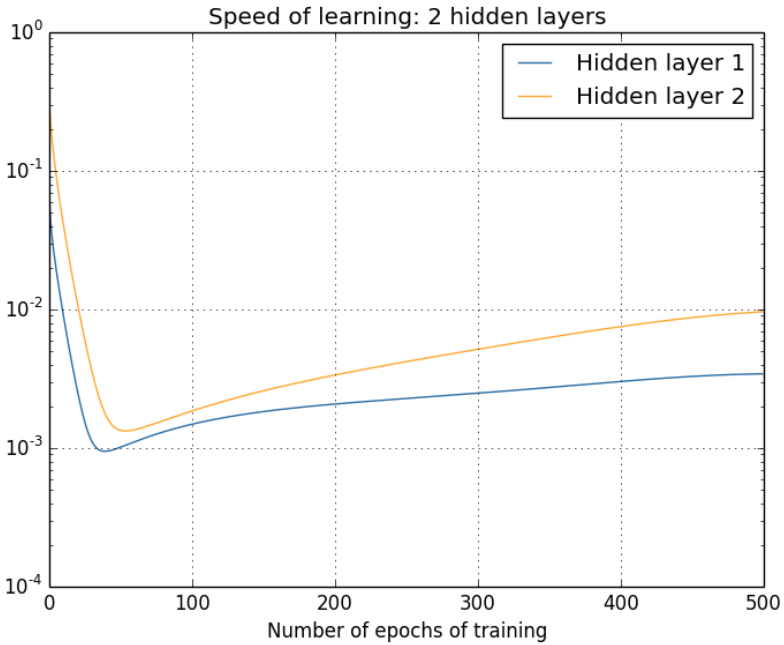


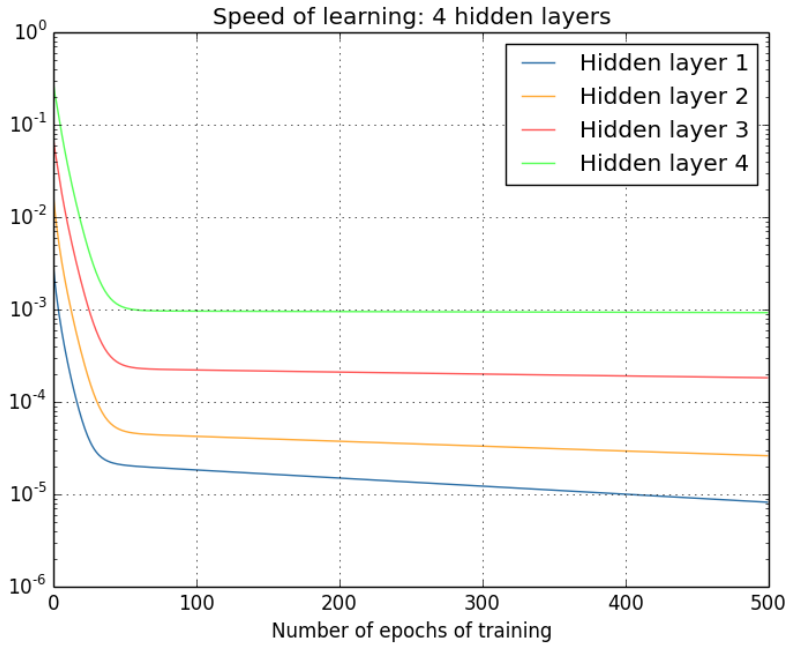
Training Speed





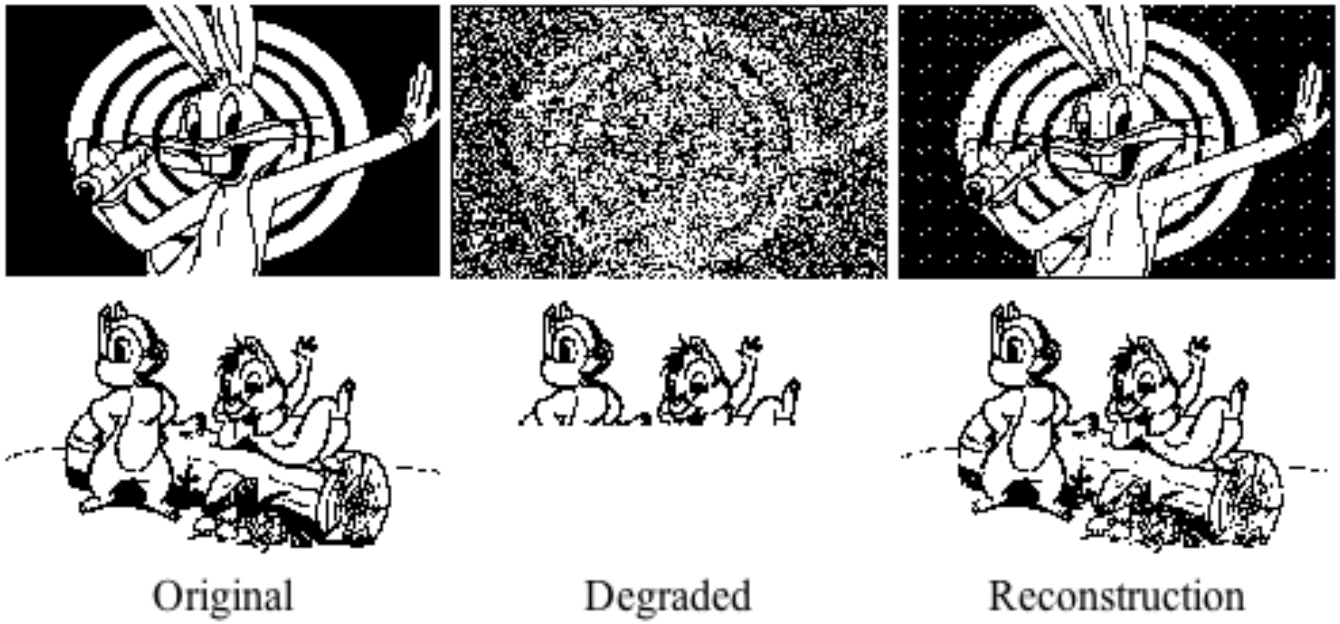
Training Speed





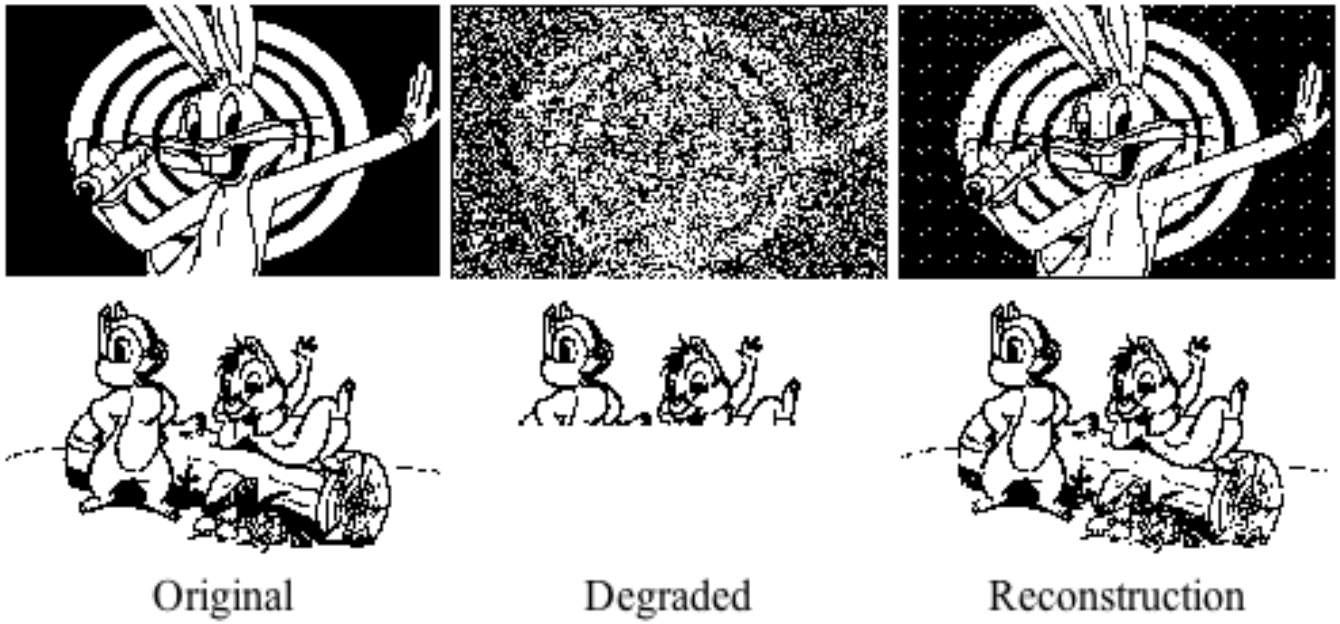
- Training speed is quantified using the norm of the weight update ΔW
- The updates are much smaller for initial layers - hence not trained
- The representations supplied to the deeper layers are not reliable

Pattern Recall - Energy-Based Models



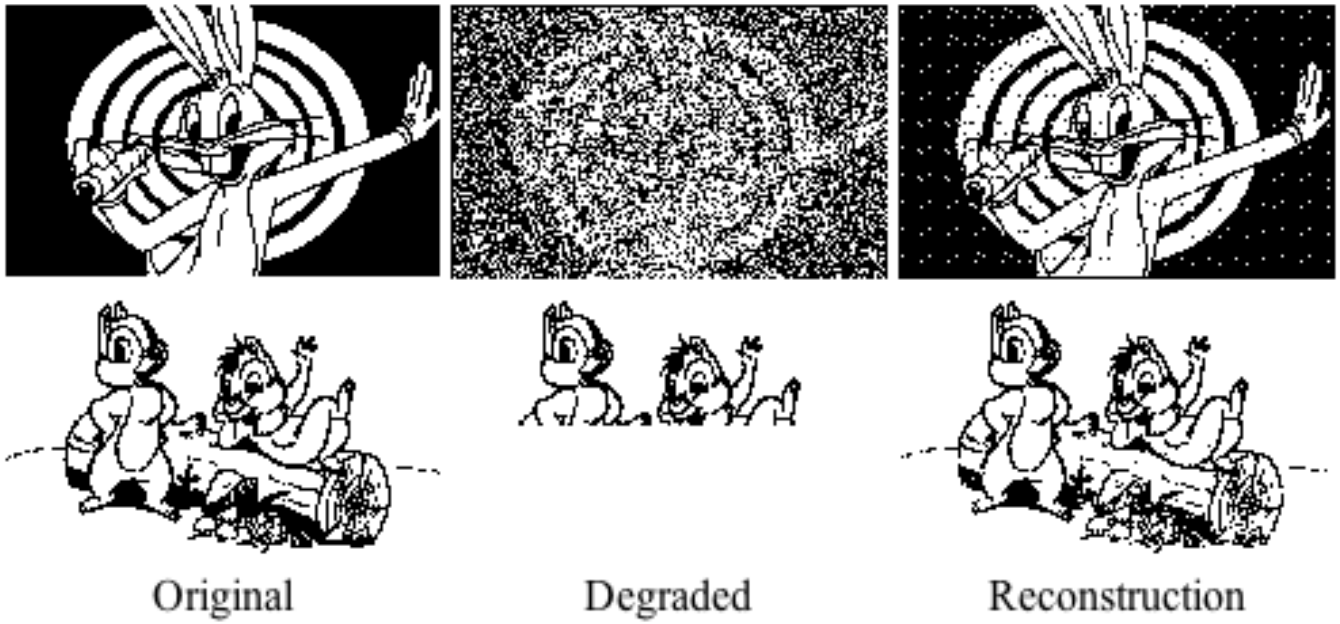
Original Degraded Reconstruction

Pattern Recall - Energy-Based Models



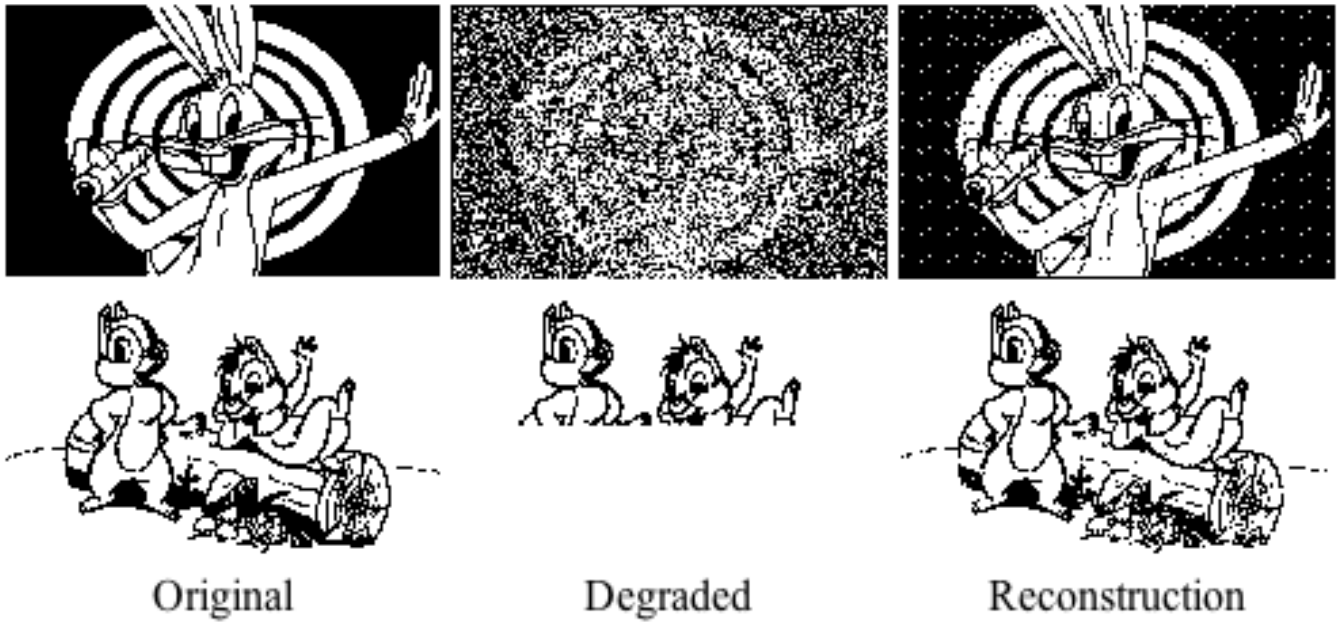
- A good representation should capture correlations in the input

Pattern Recall - Energy-Based Models



- A good representation should capture correlations in the input
  - Natural patterns admits much lower dimensional representations

Pattern Recall - Energy-Based Models



- A good representation should capture correlations in the input
  - Natural patterns admits much lower dimensional representations
  - Should be able to recall the pattern from partial/degraded input

Hopfiled Network

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

- Train the network to predict pattern from partial/noisy inputs

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

- Train the network to predict pattern from partial/noisy inputs
- The parameters of the network are adjusted to minimize energy

$$E = -\sum_{i,j} w_{ji} v_i v_j$$

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

- Train the network to predict pattern from partial/noisy inputs
- The parameters of the network are adjusted to minimize energy

$$E = -\sum_{i,j} w_{ji} v_i v_j$$

- Hard-ball dynamics ensure traversal to a minima on energy landscape

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

- Train the network to predict pattern from partial/noisy inputs
- The parameters of the network are adjusted to minimize energy

$$E = -\sum_{i,j} w_{ji} v_i v_j$$

- Hard-ball dynamics ensure traversal to a minima on energy landscape
- Capacity of a Hopfiled network is $\log_2 D$.

Hopfiled Network

- Hopfield networks are EB models with hard-limiting activation
- Correlation among the pixels is exploited for pattern storage
- Each pixel is predicted from other pixels as

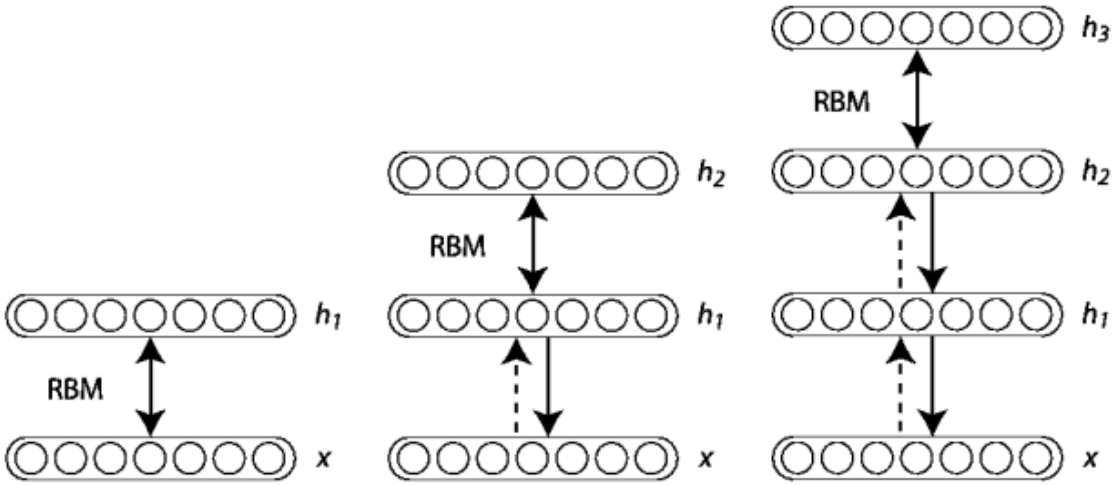
$$v_j = \text{sign}\left(\sum_{i \neq j} w_{ji} v_i\right)$$

- Train the network to predict pattern from partial/noisy inputs
- The parameters of the network are adjusted to minimize energy

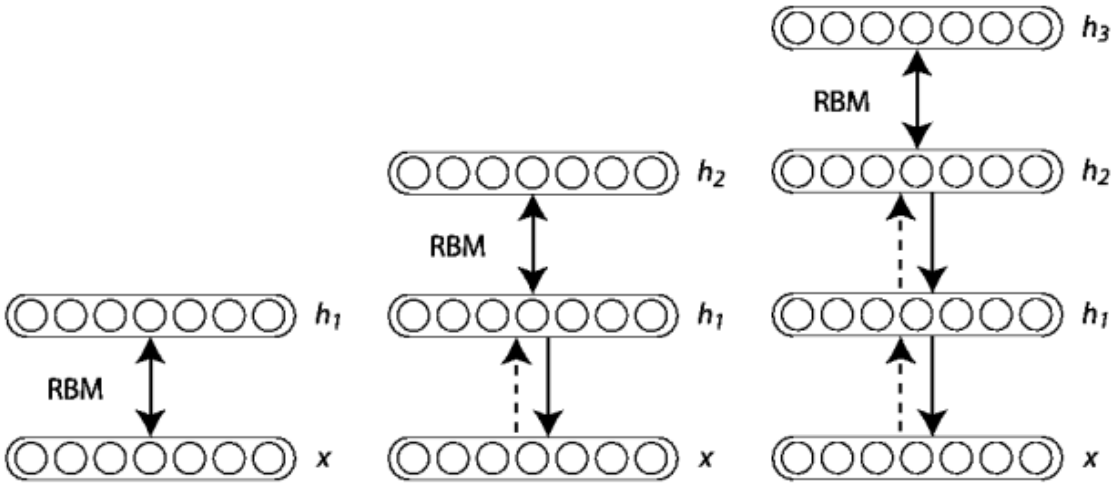
$$E = -\sum_{i,j} w_{ji} v_i v_j$$

- Hard-ball dynamics ensure traversal to a minima on energy landscape
- Capacity of a Hopfiled network is $\log_2 D$.
- Hardball can get stuck in the local minima of the energy landscape

Restricted Boltzmann Machine

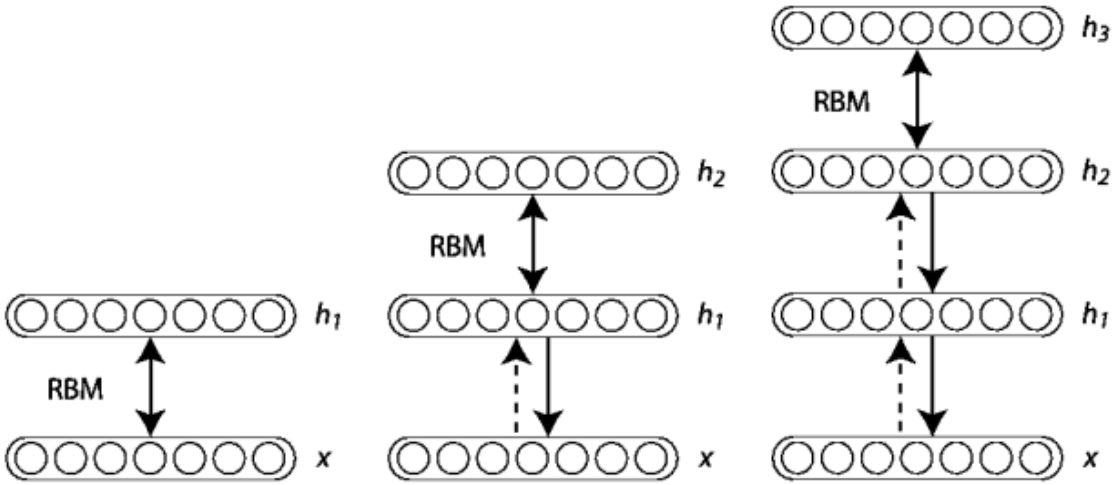


Restricted Boltzmann Machine



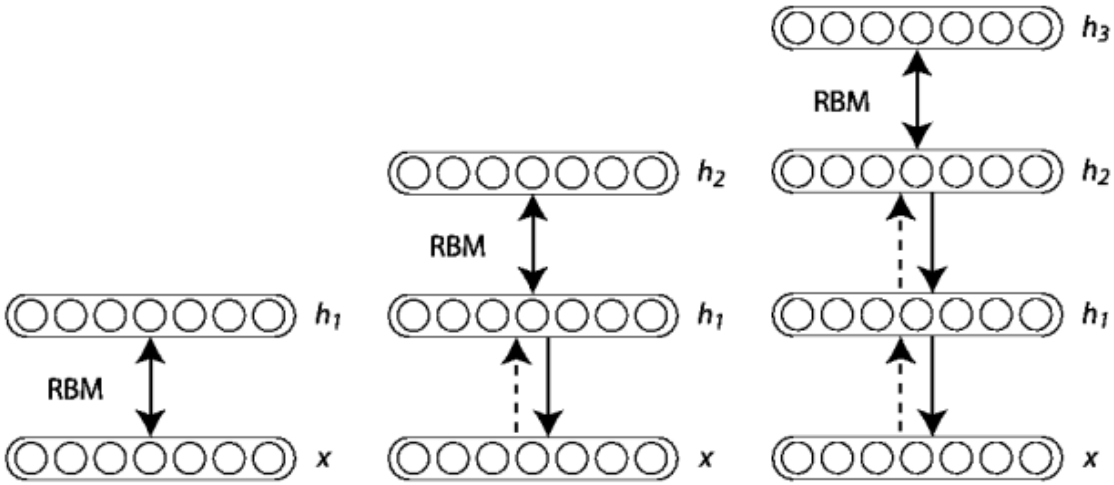
- Stochastic neuron to simulate rubber ball dynamics

Restricted Boltzmann Machine



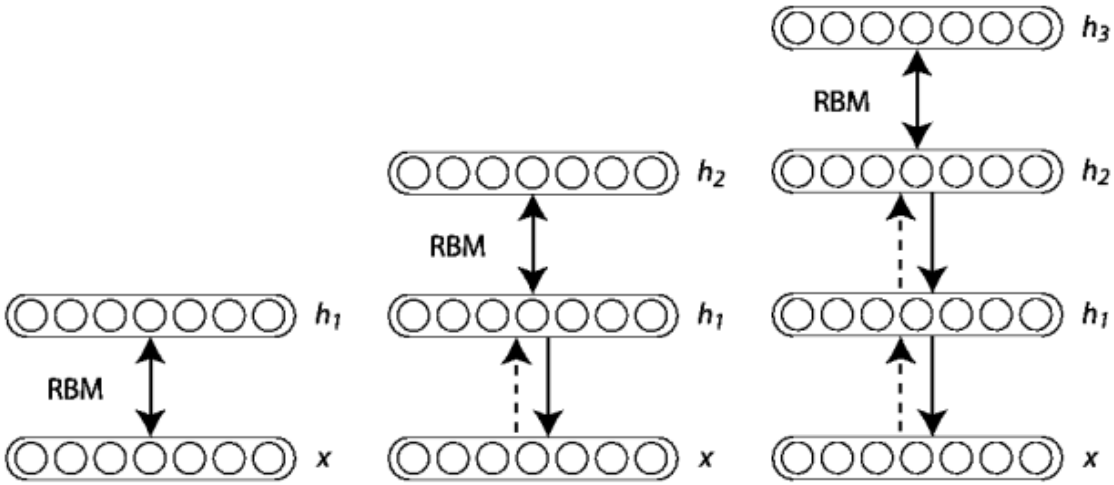
- Stochastic neuron to simulate rubber ball dynamics
- Activation probabilities: $p(h_j = 1/\mathbf{v}) = \sigma\left(\sum_{i=0}^{D} w_{ji} v_i\right)$

Restricted Boltzmann Machine



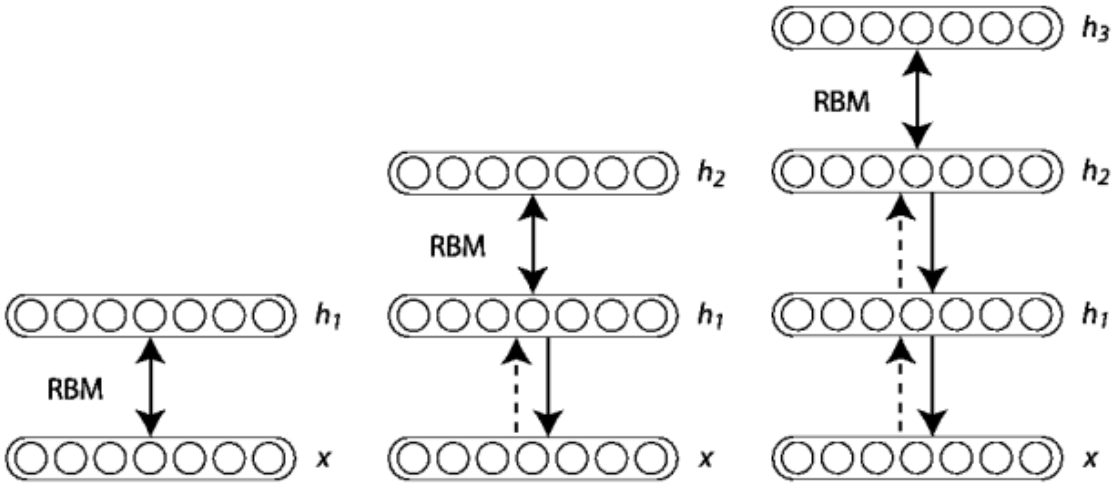
- Stochastic neuron to simulate rubber ball dynamics
- Activation probabilities: $p(h_j = 1/\mathbf{v}) = \sigma\left(\sum_{i=0}^{D} w_{ji} v_i\right)$
- The energy of an B-B RBM is given by $E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^{\mathsf{T}}\mathbf{W}\mathbf{h}$

Restricted Boltzmann Machine



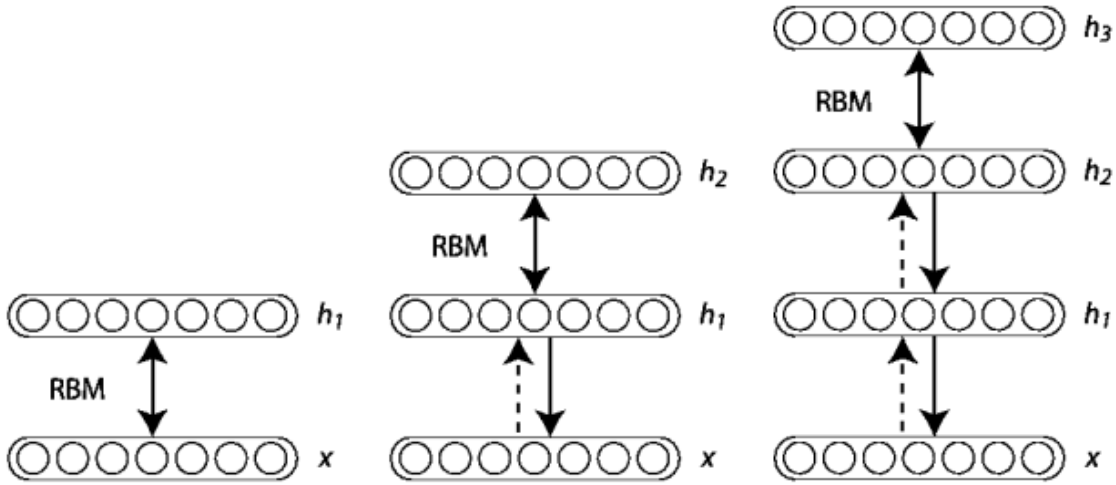
- Stochastic neuron to simulate rubber ball dynamics
- Activation probabilities: $p(h_j = 1/\mathbf{v}) = \sigma\left(\sum_{i=0}^{D} w_{ji} v_i\right)$
- The energy of an B-B RBM is given by $E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^{\mathsf{T}}\mathbf{W}\mathbf{h}$
- The joint density over visible and hidden units is $p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$

Restricted Boltzmann Machine



- Stochastic neuron to simulate rubber ball dynamics
- Activation probabilities: $p(h_j = 1/\mathbf{v}) = \sigma\left(\sum_{i=0}^{D} w_{ji} v_i\right)$
- The energy of an B-B RBM is given by $E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^{\mathsf{T}}\mathbf{W}\mathbf{h}$
- The joint density over visible and hidden units is $p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$
- Estimate $\mathbf{W}$ to maximize $p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$

Restricted Boltzmann Machine



- Stochastic neuron to simulate rubber ball dynamics
- Activation probabilities: $p(h_j = 1/\mathbf{v}) = \sigma\left(\sum_{i=0}^{D} w_{ji} v_i\right)$
- The energy of an B-B RBM is given by $E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^{\mathsf{T}}\mathbf{W}\mathbf{h}$
- The joint density over visible and hidden units is $p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$
- Estimate $\mathbf{W}$ to maximize $p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$
- Contrastive divergence is used to estimate the parameters

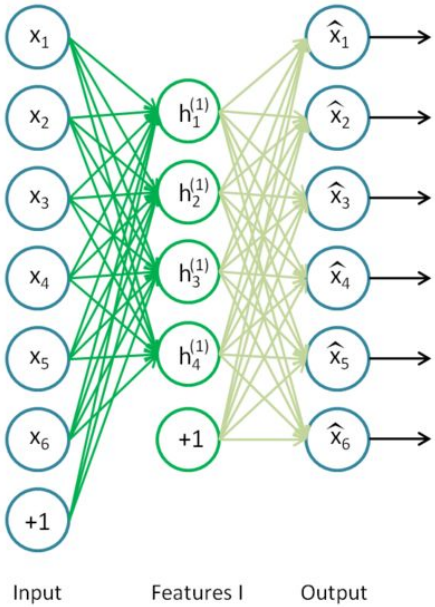
Unsupervised Pretraining

- Unsupervised pretraining forces model close to $p(\mathbf{x})$ (Hinton 2006)
- Representations good for $p(\mathbf{x})$ are good for $p(\mathbf{t}/\mathbf{x})$?
- Initializes near better local minima for $p(\mathbf{t}/\mathbf{x})$?

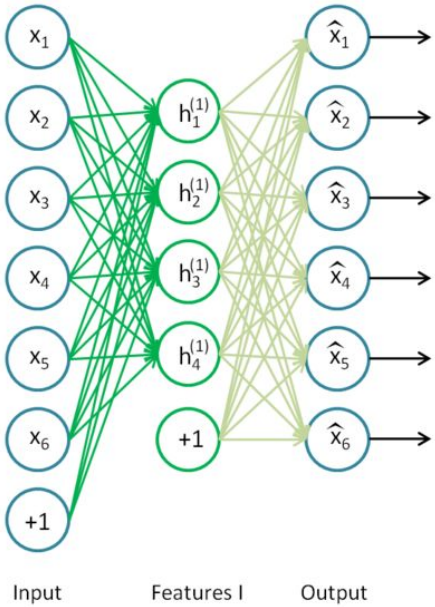
Unsupervised Pretraining

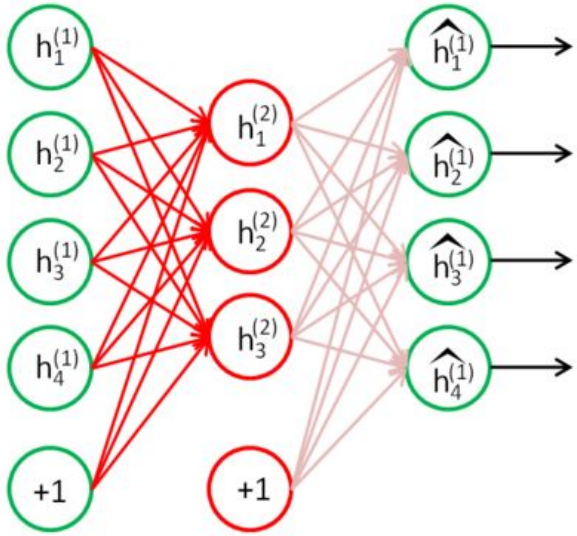
- Unsupervised pretraining forces model close to $p(\mathbf{x})$ (Hinton 2006)
- Representations good for $p(\mathbf{x})$ are good for $p(\mathbf{t}/\mathbf{x})$?
- Initializes near better local minima for $p(\mathbf{t}/\mathbf{x})$?
- Offers easy layer-wise training with local criterion
- Large amounts of unlabeled data can be used in pretraining
- Achieved significant improvements in speech recognition (Hinton 2008)

Pretraining using Autoencoders

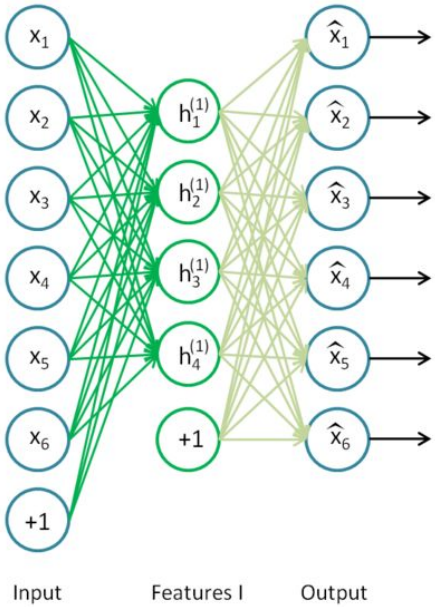


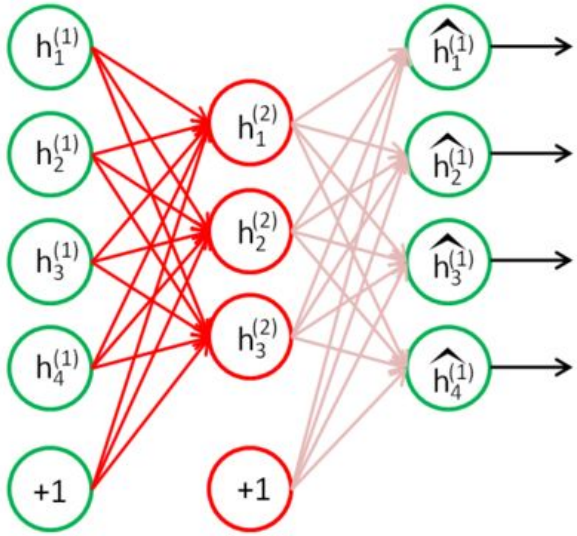
Pretraining using Autoencoders

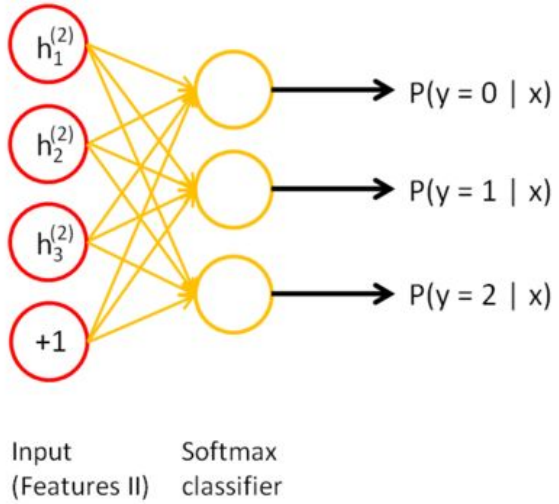




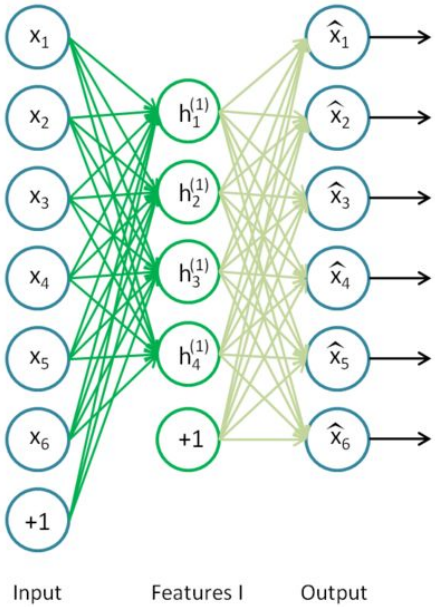
Pretraining using Autoencoders

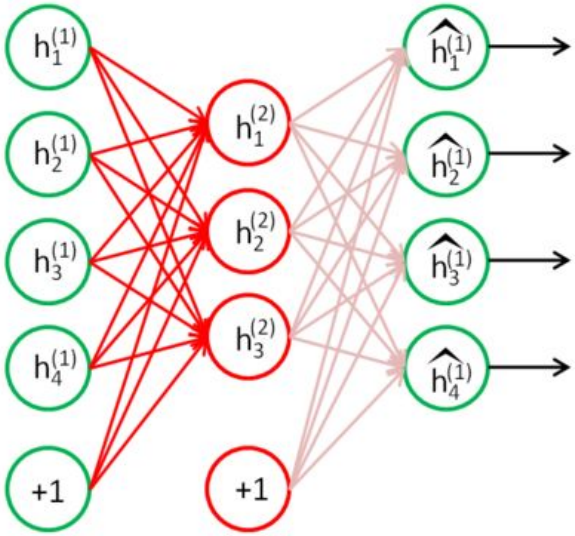


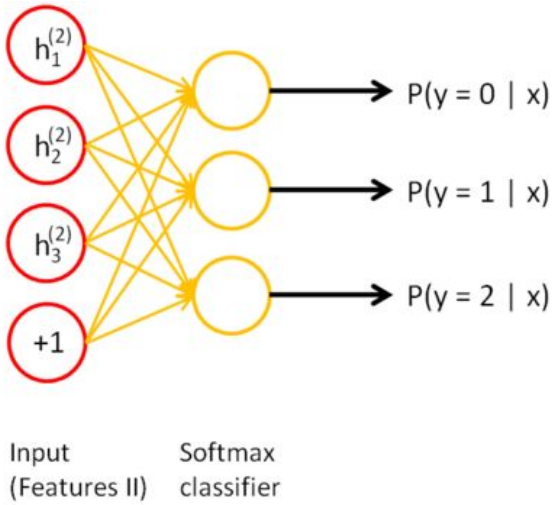


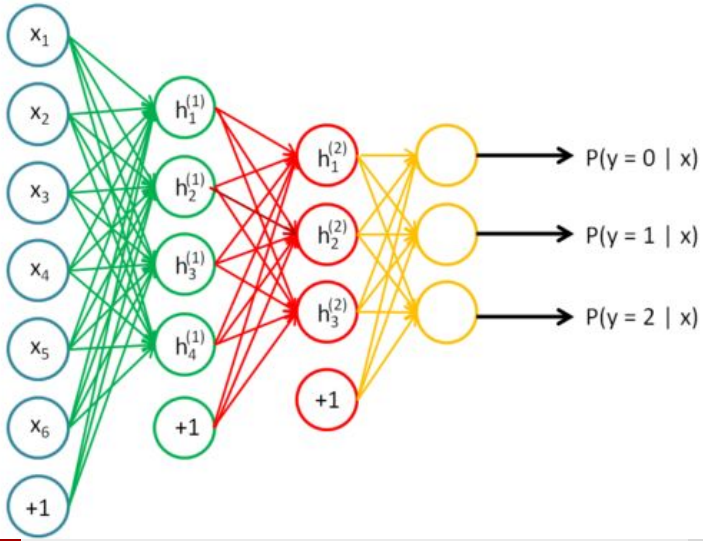


Pretraining using Autoencoders

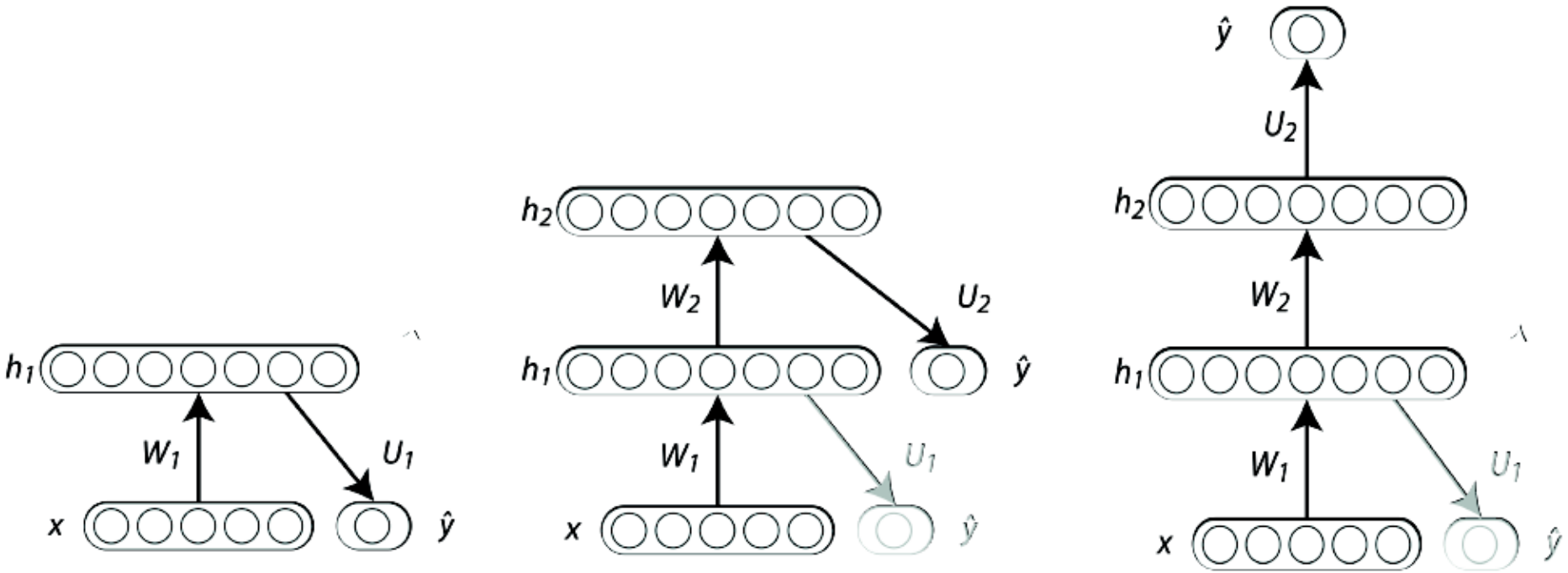








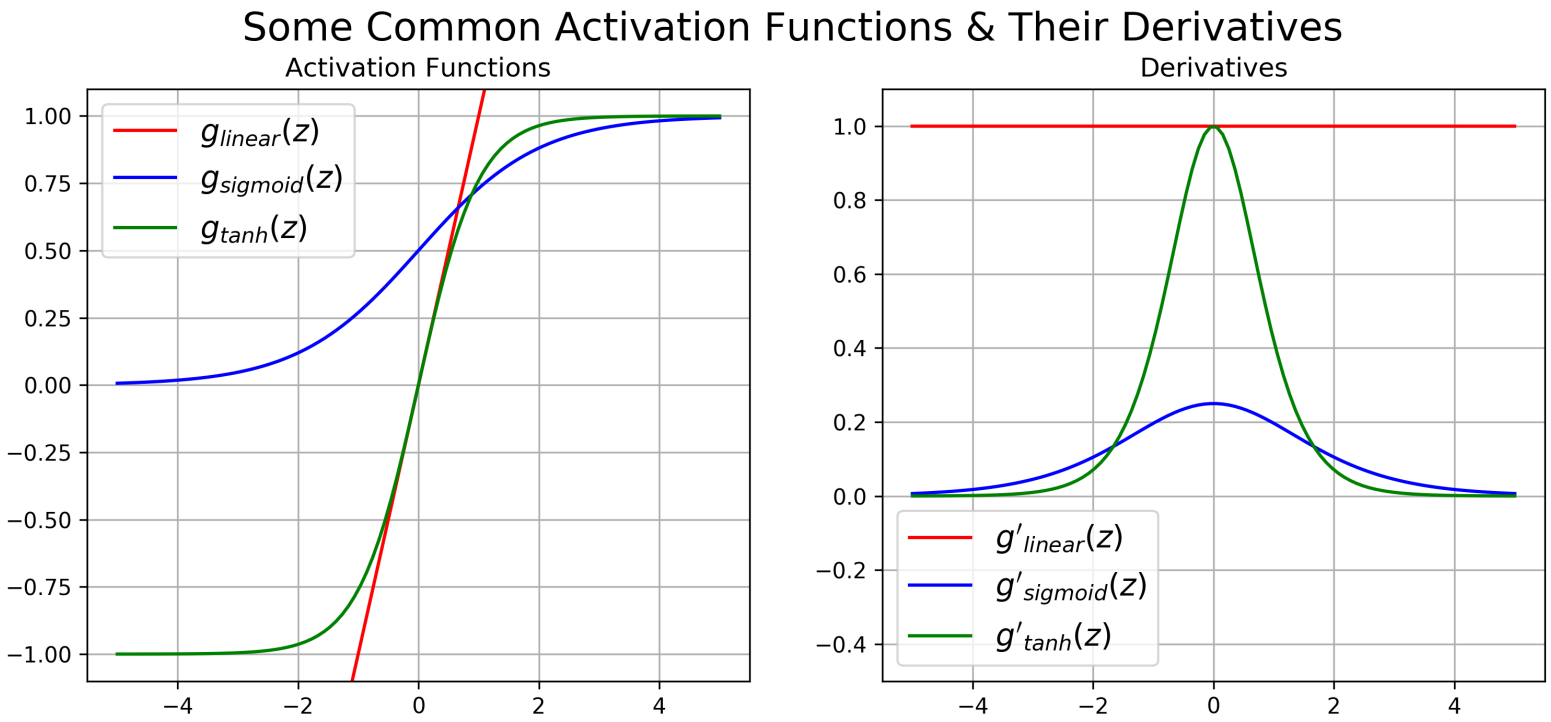
Supervised Pretraining



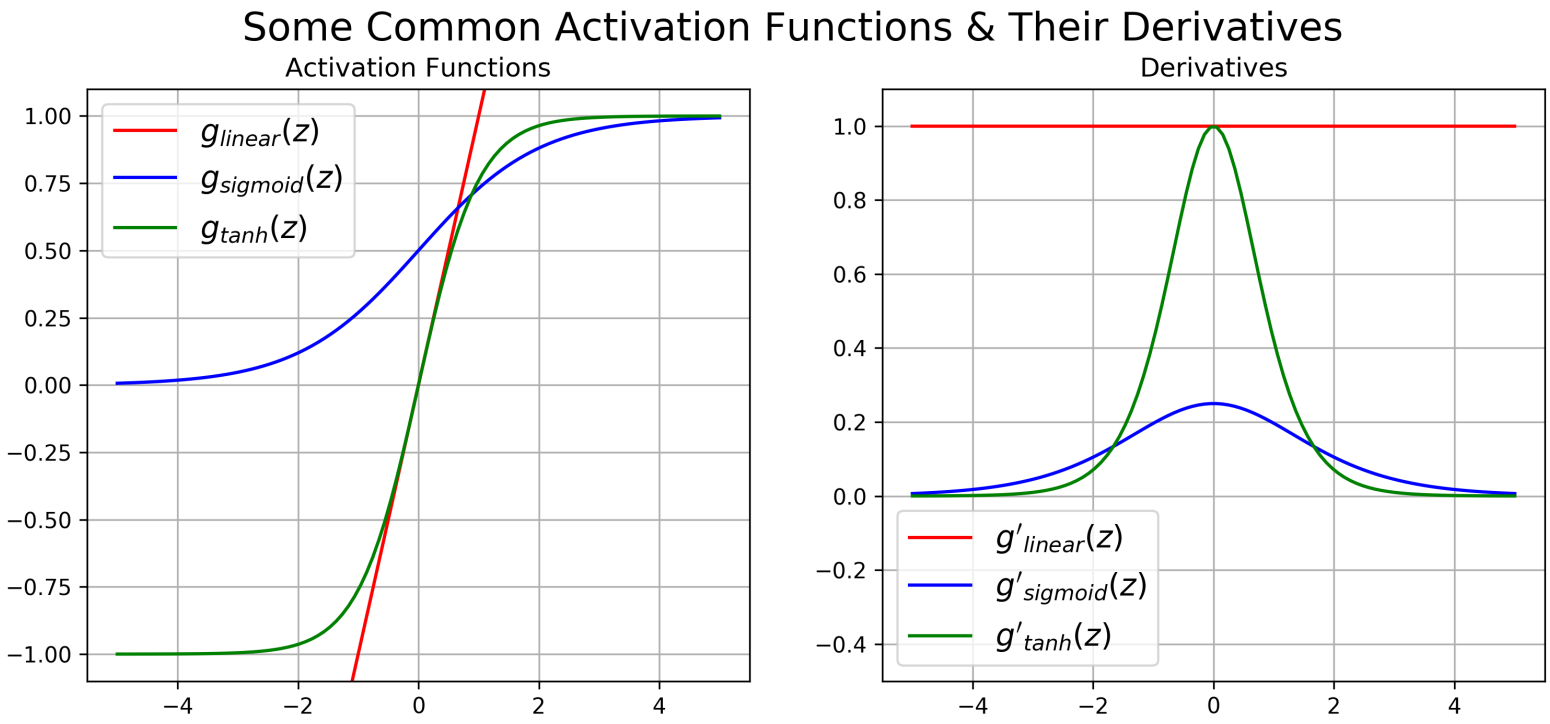
- Generally performs worse than unsupervised pretraining, but better than random initialization of DNN

Derivatives of Activation Functions

Derivatives of Activation Functions

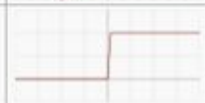
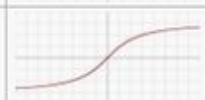
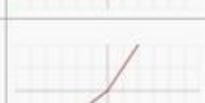


Derivatives of Activation Functions

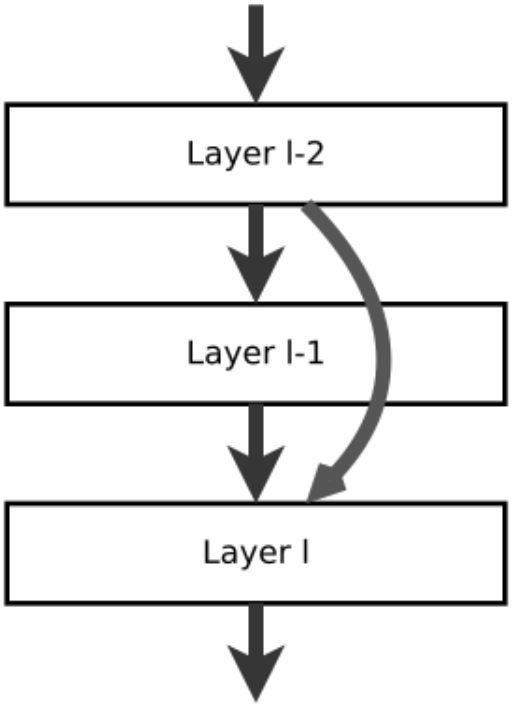


- The magnitude of derivatives of tanh(.) and $\sigma(.)$ are less than 1
- Deep cascade of such activation layers lead to vanishing gradients
- ReLU address this issue as it offers unity gradient for +ve values

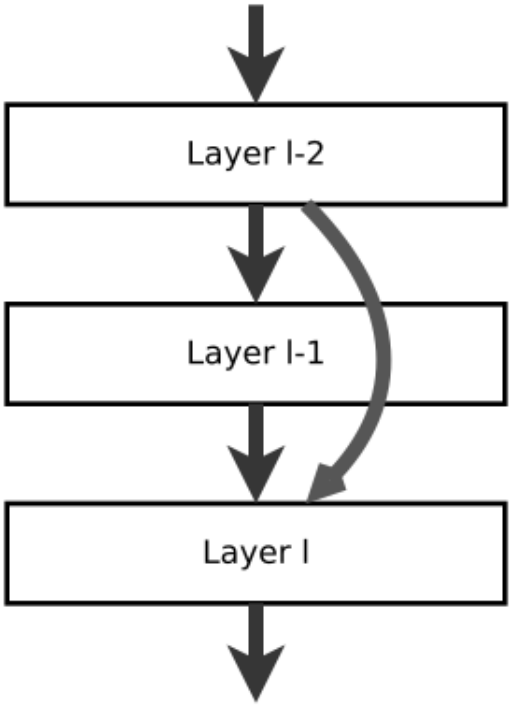
Choice of Nonlinearity

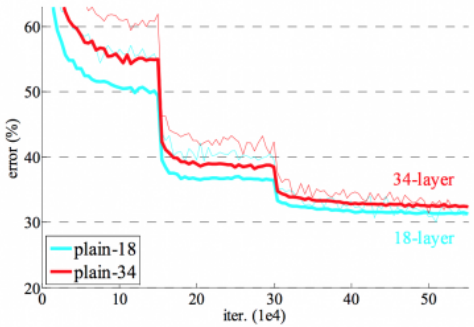
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$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

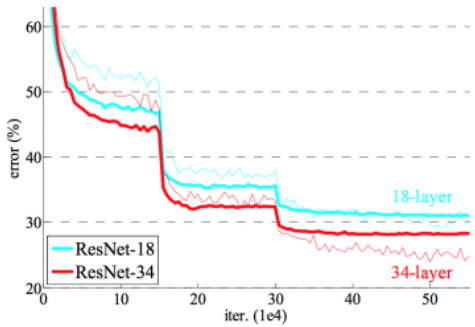
Residual Networks (ResNet)



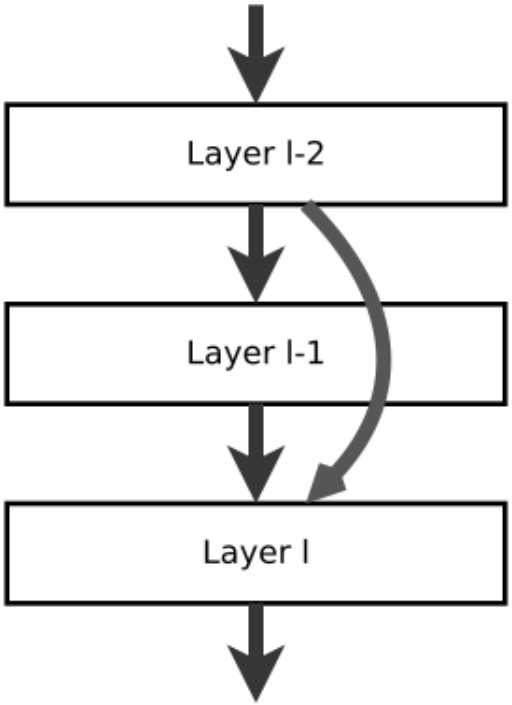
Residual Networks (ResNet)





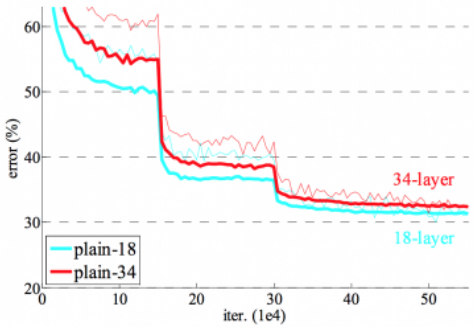


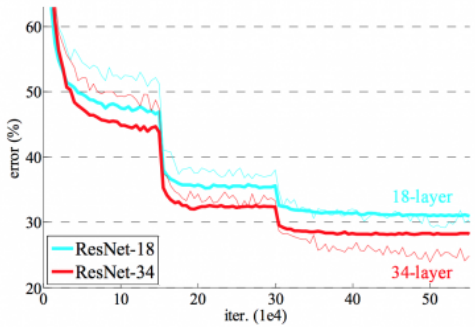
Residual Networks (ResNet)



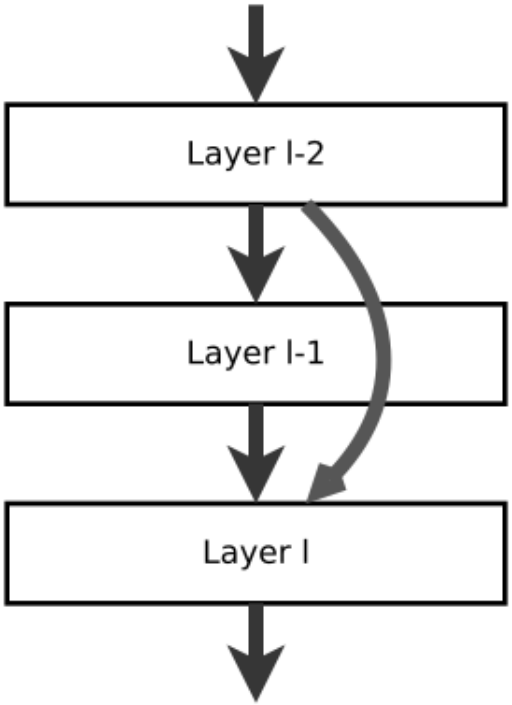
- 18 layer net has lower error than 34 layer net!
- Skip-connections to overcome vanishing grd.

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





Residual Networks (ResNet)

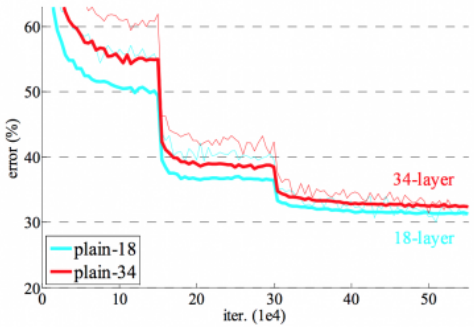


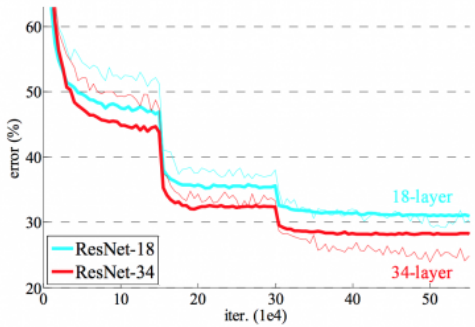
- 18 layer net has lower error than 34 layer net!
- Skip-connections to overcome vanishing grd.

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

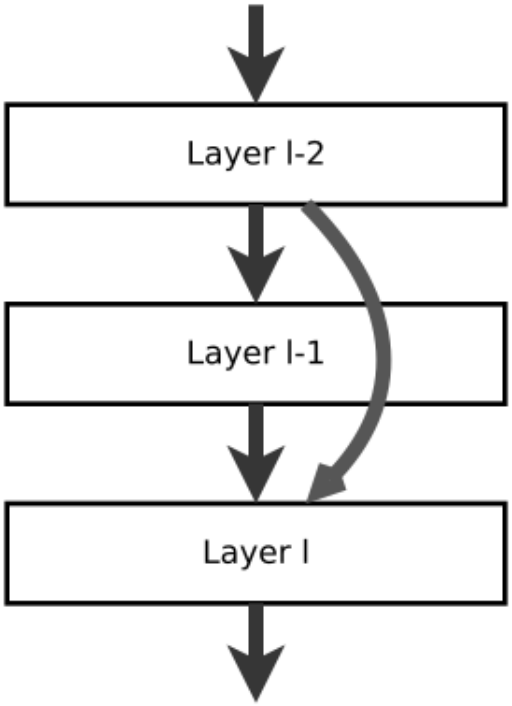
- If there are no weights on skip connections

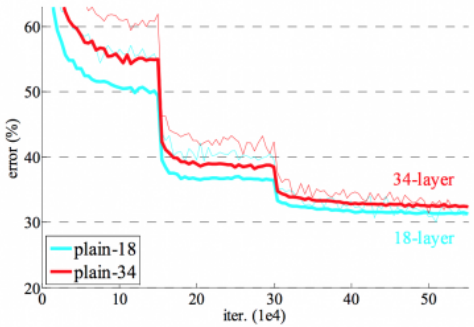
$$\mathbf{z}^{(l)} = h\left(g(\mathbf{z}^{(l-2)}) + \mathbf{z}^{(l-2)}\right)$$

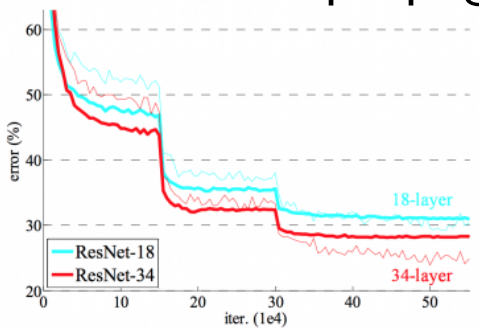




Residual Networks (ResNet)







- 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

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

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

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

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

$$y_k = f(a_k)$$

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

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

$$y_k = f(a_k)$$

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

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

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

$$y_k = f(a_k)$$

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

Error gradient w.r.t input $\mathbf{x}$

$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial x_i}$$

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

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

$$y_k = f(a_k)$$

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

Error gradient w.r.t input $\mathbf{x}$

$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial x_i}$$

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

Sensitivity of Error to Changes in Input

Forward pass input $\mathbf{x}$

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

$$z_j = h(a_j)$$

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

$$y_k = f(a_k)$$

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

Error gradient w.r.t input $\mathbf{x}$

$$\frac{\partial J}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial x_i}$$

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

$$= \left(h'(a_j) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k \right) w_{ji}^{(1)}$$

- Certain features may not influence the error
- Such features may be pruned to reduce input dimensions!

Feature Selection / Adversarial Examples

- Feature selection (pruning): Sensitivity of error to changes in inputs

$$\nabla_{\mathbf{x}} J(\mathbf{W}) = \nabla_{\mathbf{a}_1} J(\mathbf{w}) \nabla_{\mathbf{x}} (\mathbf{a}_1)$$

Feature Selection / Adversarial Examples

- Feature selection (pruning): Sensitivity of error to changes in inputs
$$\nabla_{\mathbf{x}} J(\mathbf{W}) = \nabla_{\mathbf{a}_1} J(\mathbf{w}) \nabla_{\mathbf{x}} (\mathbf{a}_1)$$
- Adversarial example creation (Fast gradient sign method)
$$\tilde{\mathbf{x}} = \mathbf{x} - \epsilon \, \nabla_{\mathbf{x}} J(\mathbf{W})$$

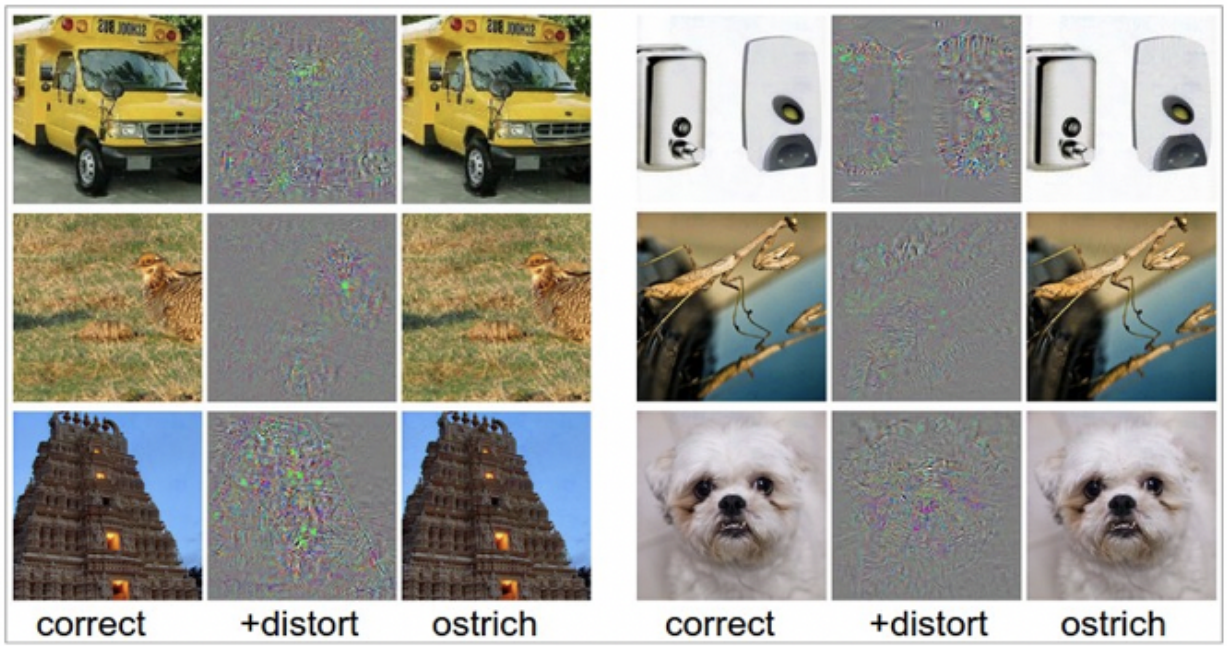
Feature Selection / Adversarial Examples

- Feature selection (pruning): Sensitivity of error to changes in inputs

$$\nabla_{\mathbf{x}} J(\mathbf{W}) = \nabla_{\mathbf{a}_1} J(\mathbf{w}) \nabla_{\mathbf{x}} (\mathbf{a}_1)$$

- Adversarial example creation (Fast gradient sign method)

$$\tilde{\mathbf{x}} = \mathbf{x} - \epsilon \, \nabla_{\mathbf{x}} J(\mathbf{W})$$



Deep Fool

Deep Fool

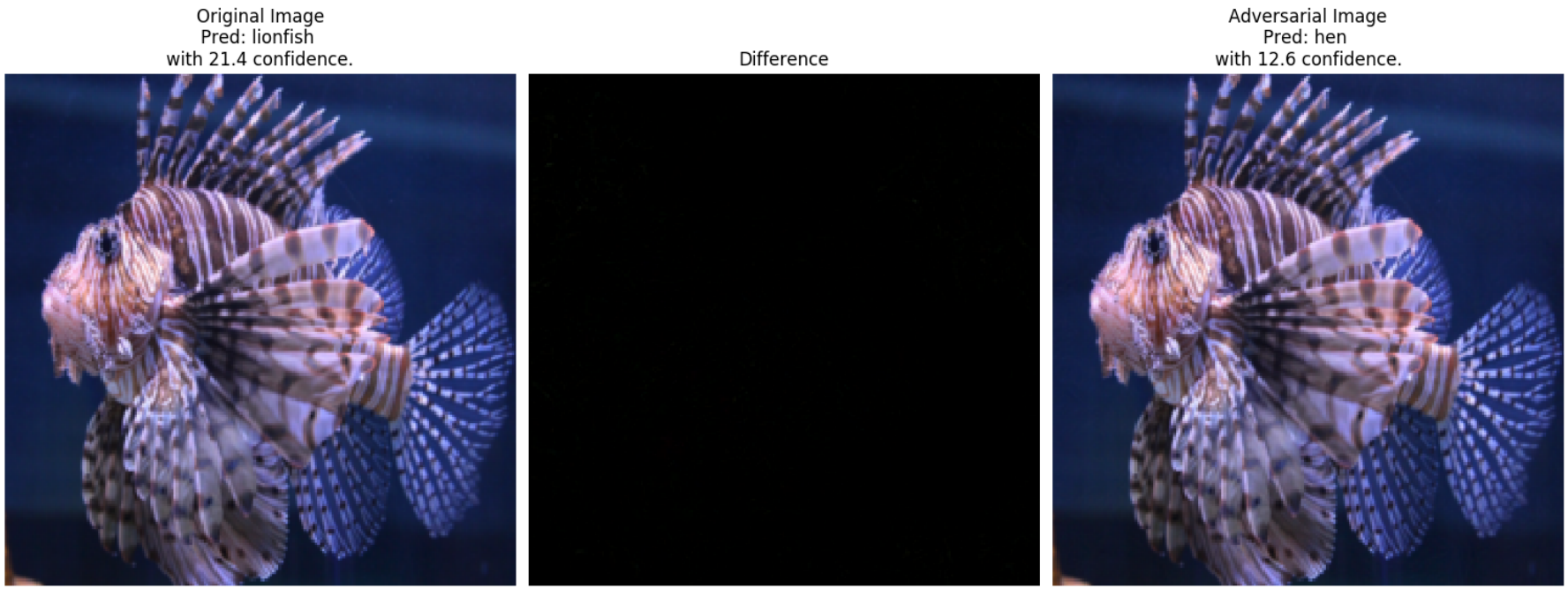
- Estimate the smallest possible perturbation needed to alter the inferred class

$$\Delta \mathbf{x} = \arg\min_{\epsilon} \|\boldsymbol{\epsilon}\|_2 \ni f(\mathbf{x} + \boldsymbol{\epsilon}) \neq f(\mathbf{x})$$

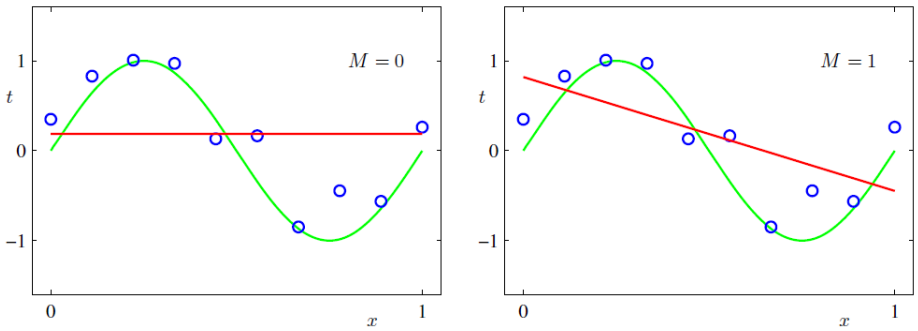
Deep Fool

- Estimate the smallest possible perturbation needed to alter the inferred class

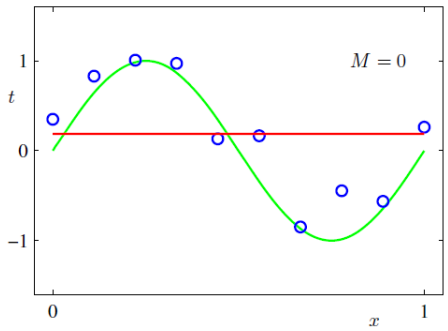
$$\Delta \mathbf{x} = \arg\min_{\epsilon} \|\epsilon\|_2 \ni f(\mathbf{x} + \epsilon) \neq f(\mathbf{x})$$

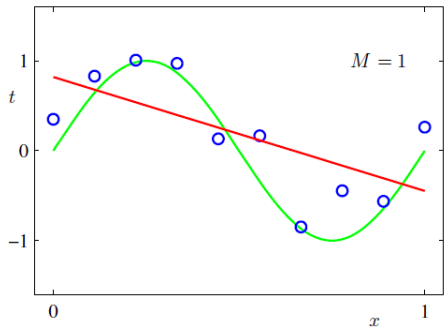


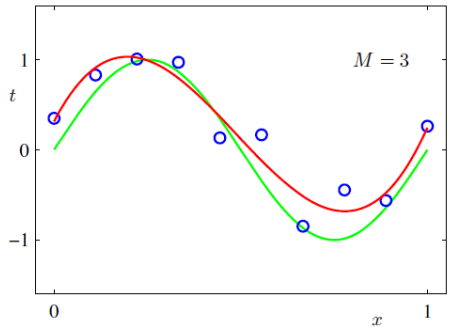
Overfitting

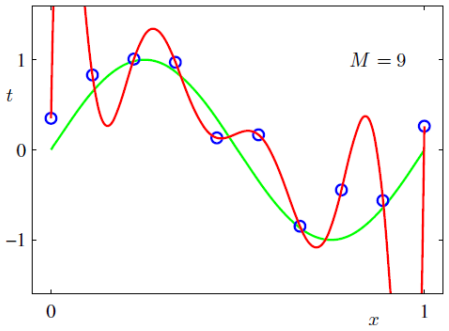


Overfitting

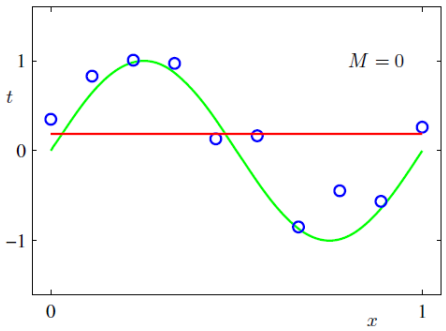


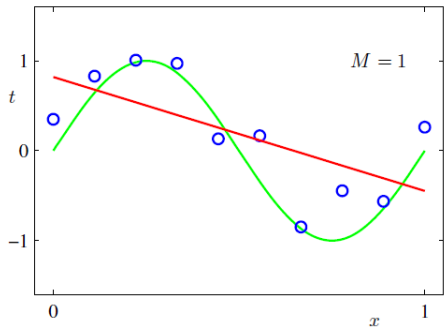


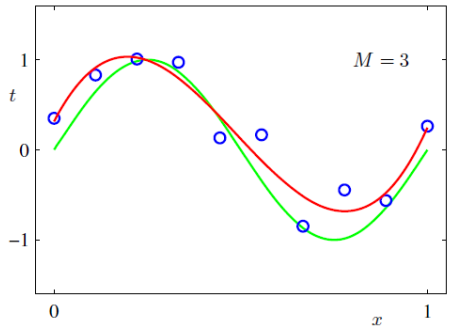


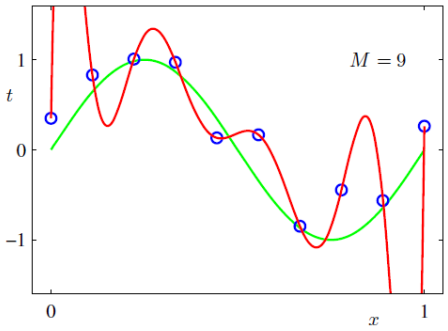


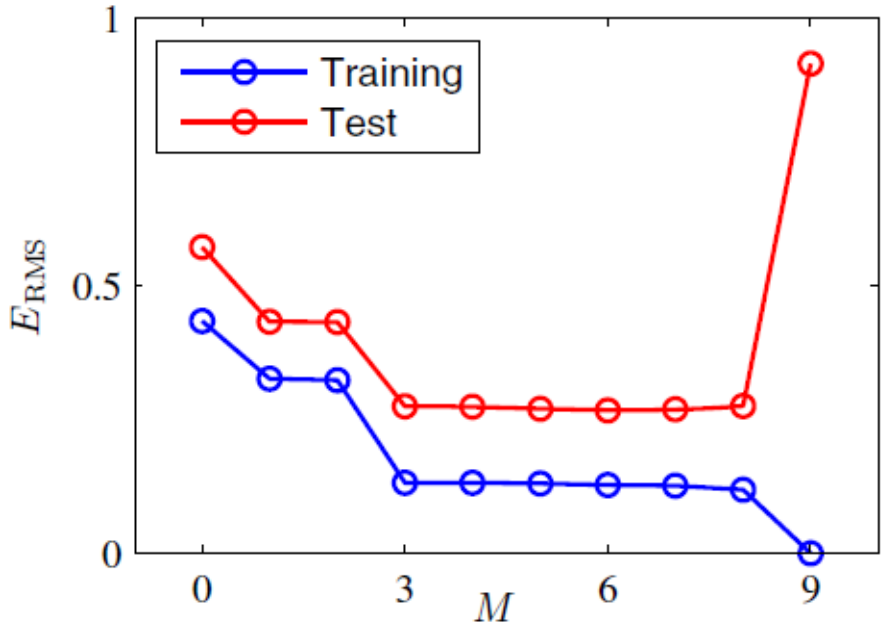
Overfitting



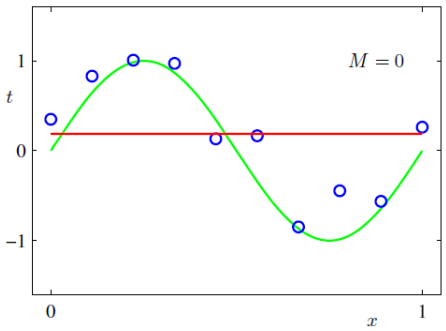


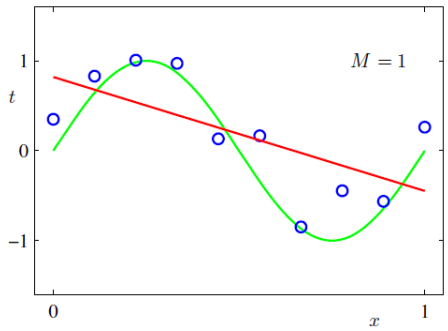


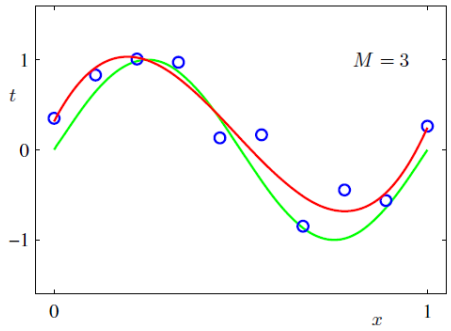


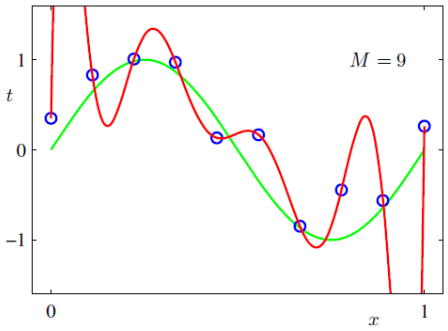


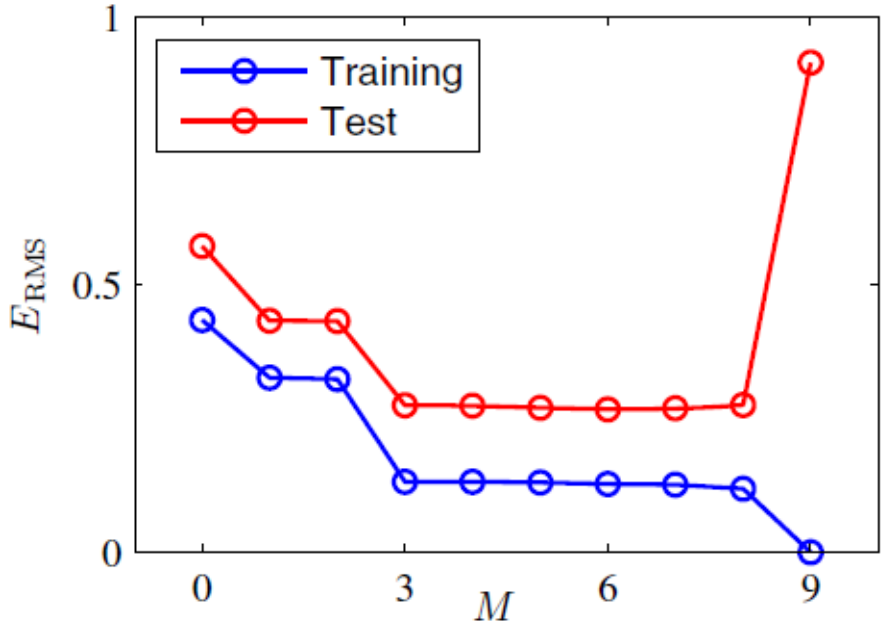
Overfitting





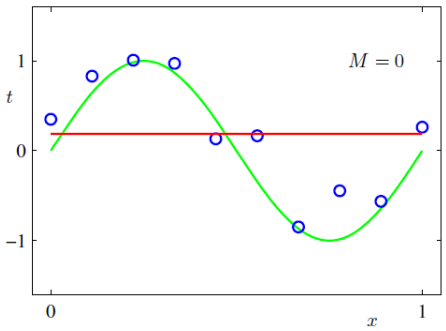


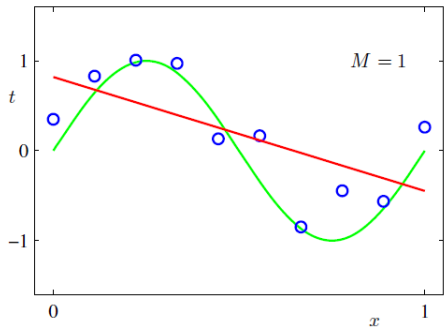


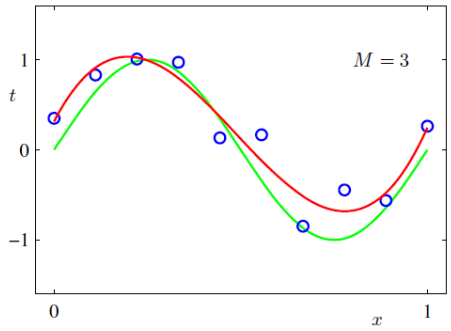


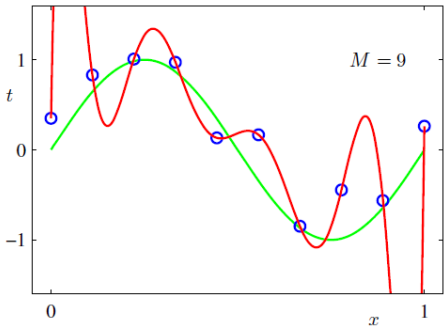
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

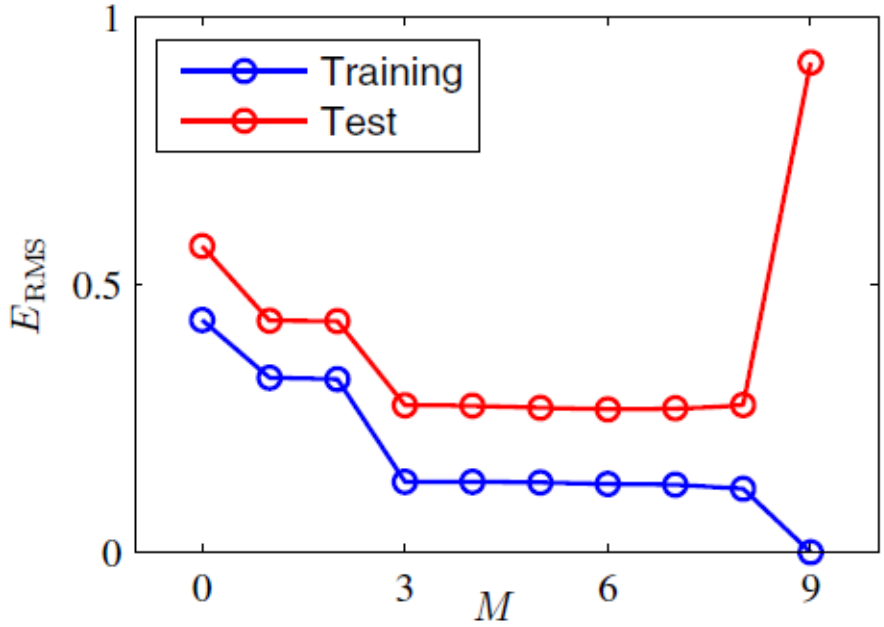
Overfitting











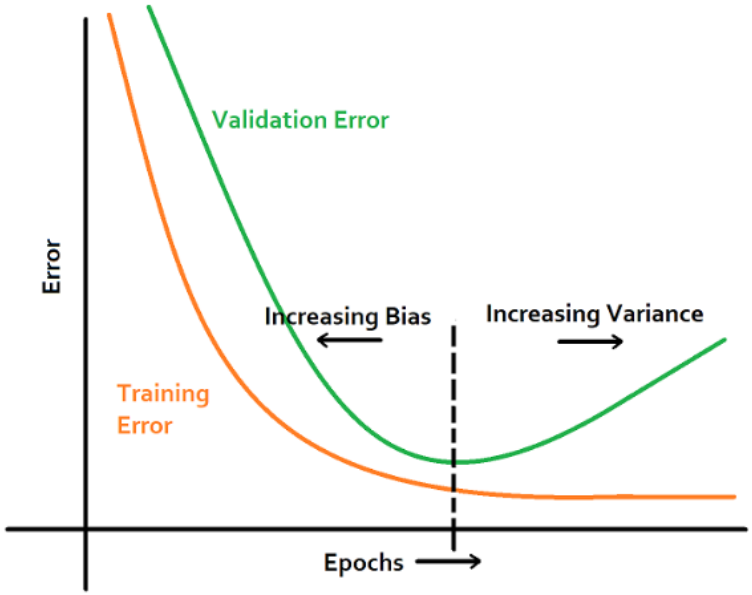
	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

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

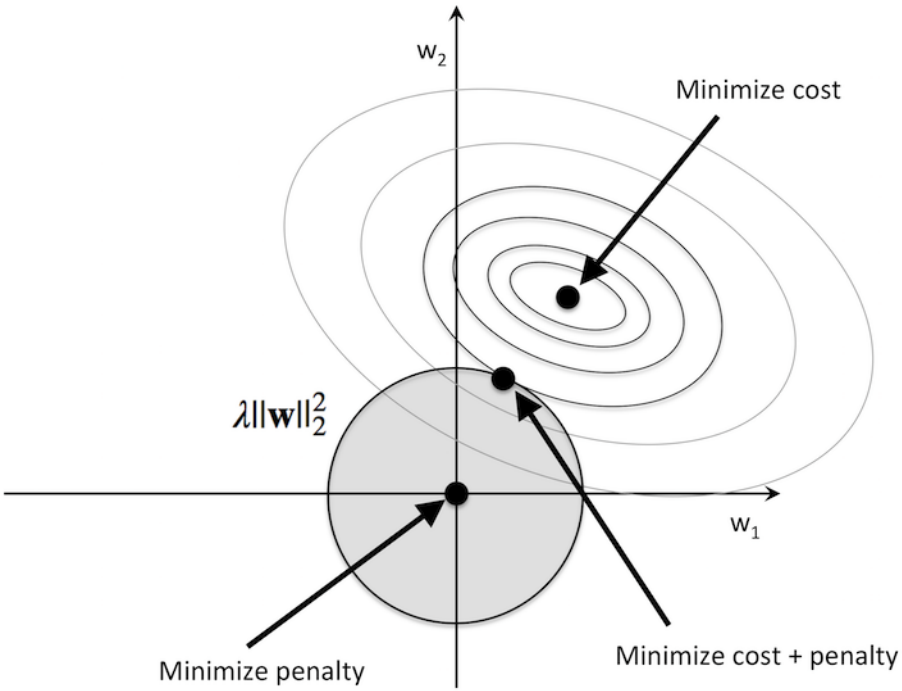


Weight Regularization

- 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

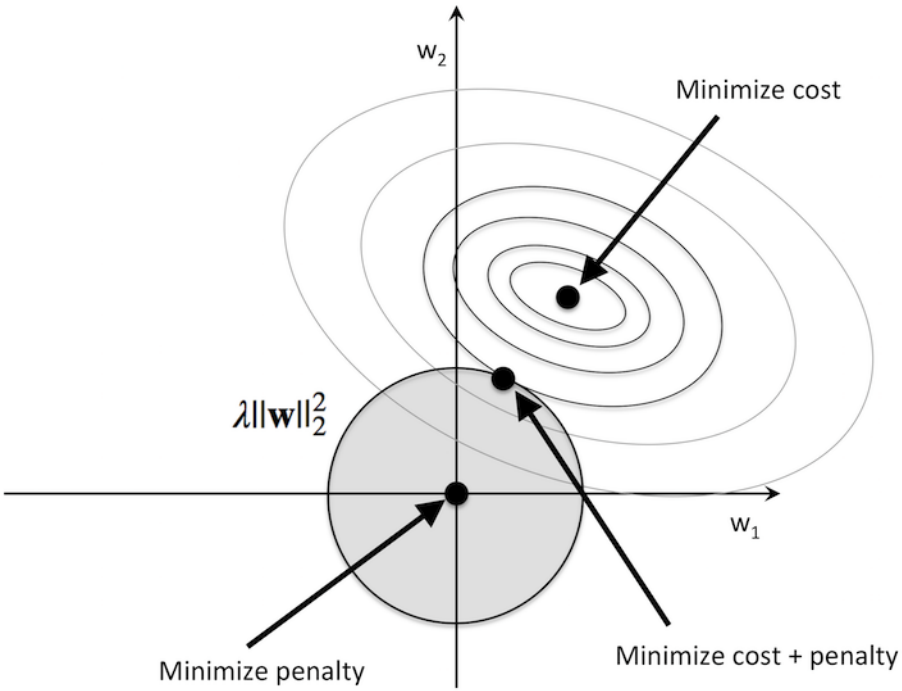


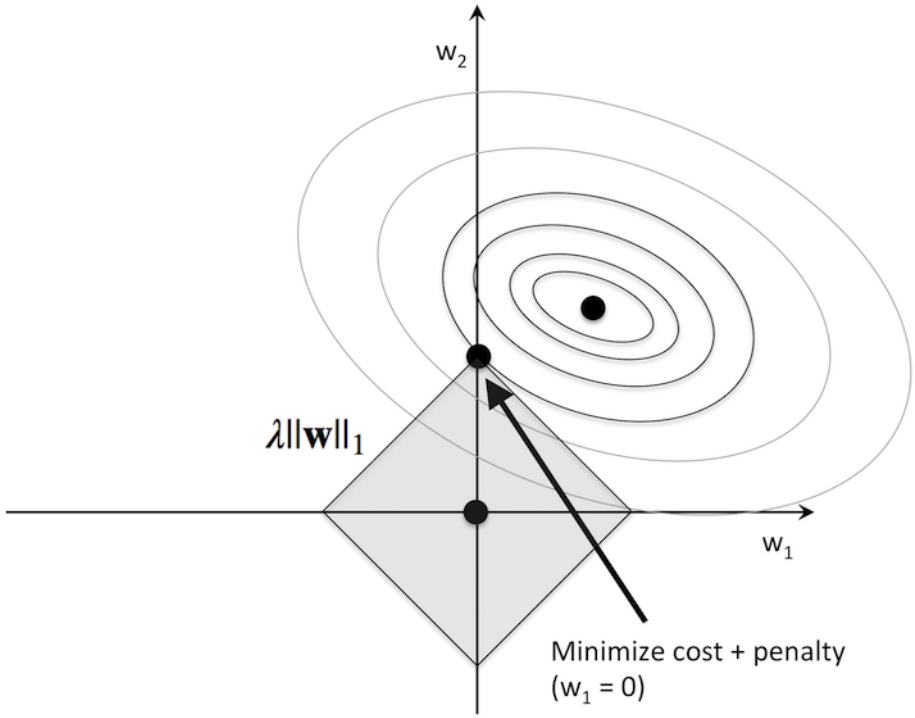
Weight Regularization

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

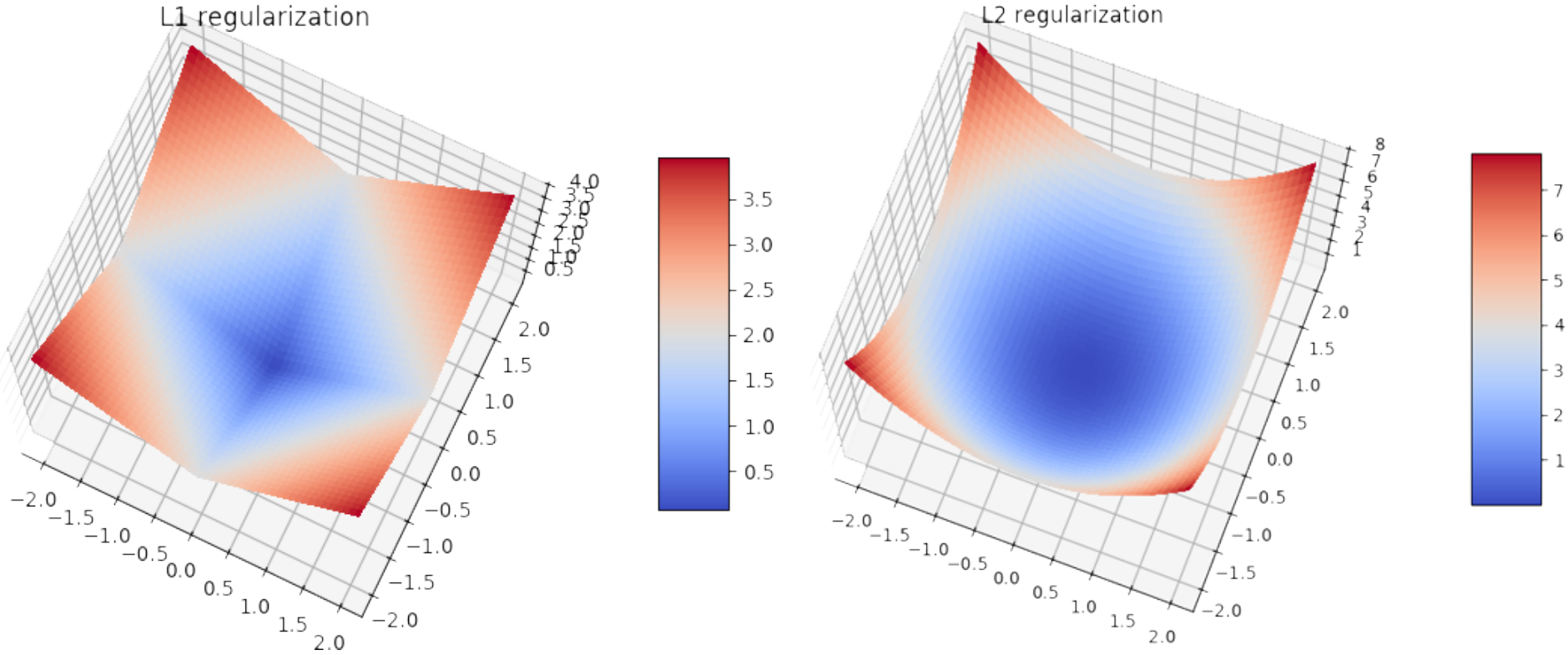
$$J(\mathbf{W}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}(y_{nk} - d_{nk})^2 + \lambda\|\mathbf{W}\|_p$$

- λ controls the trade-off between bias and variance





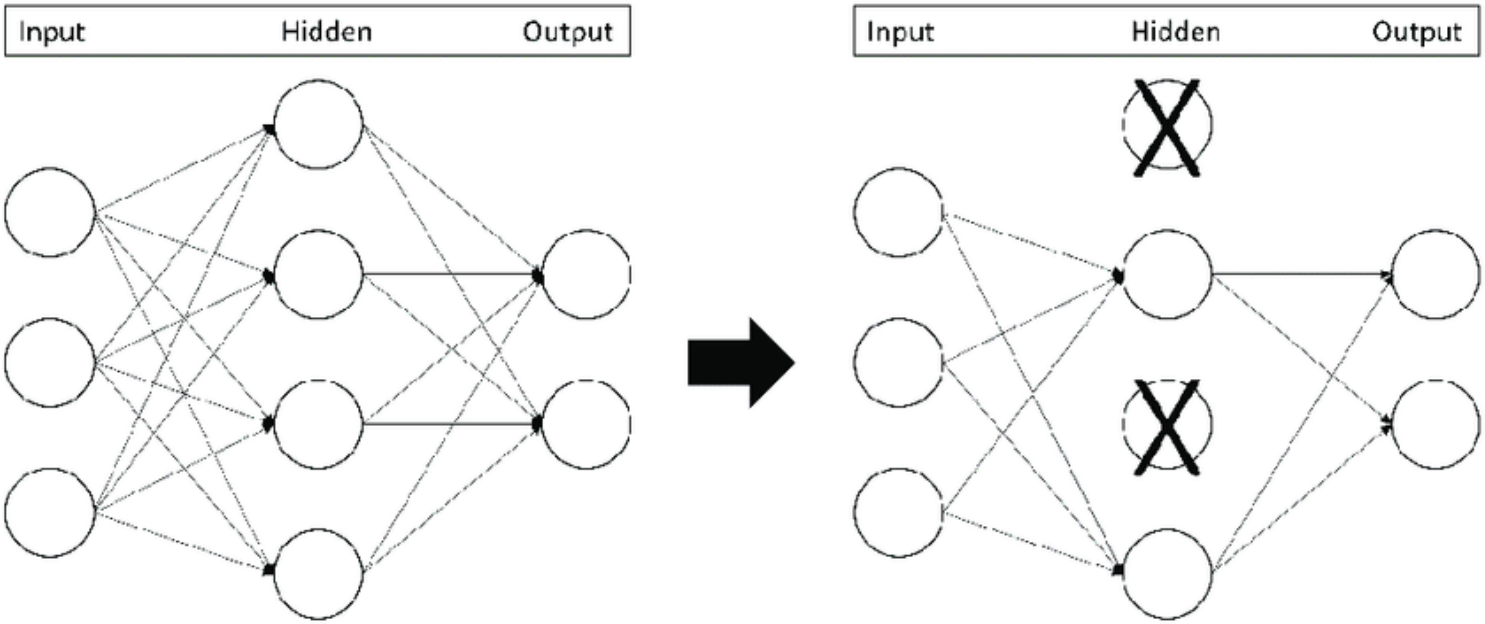
L_1 vs L_2



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

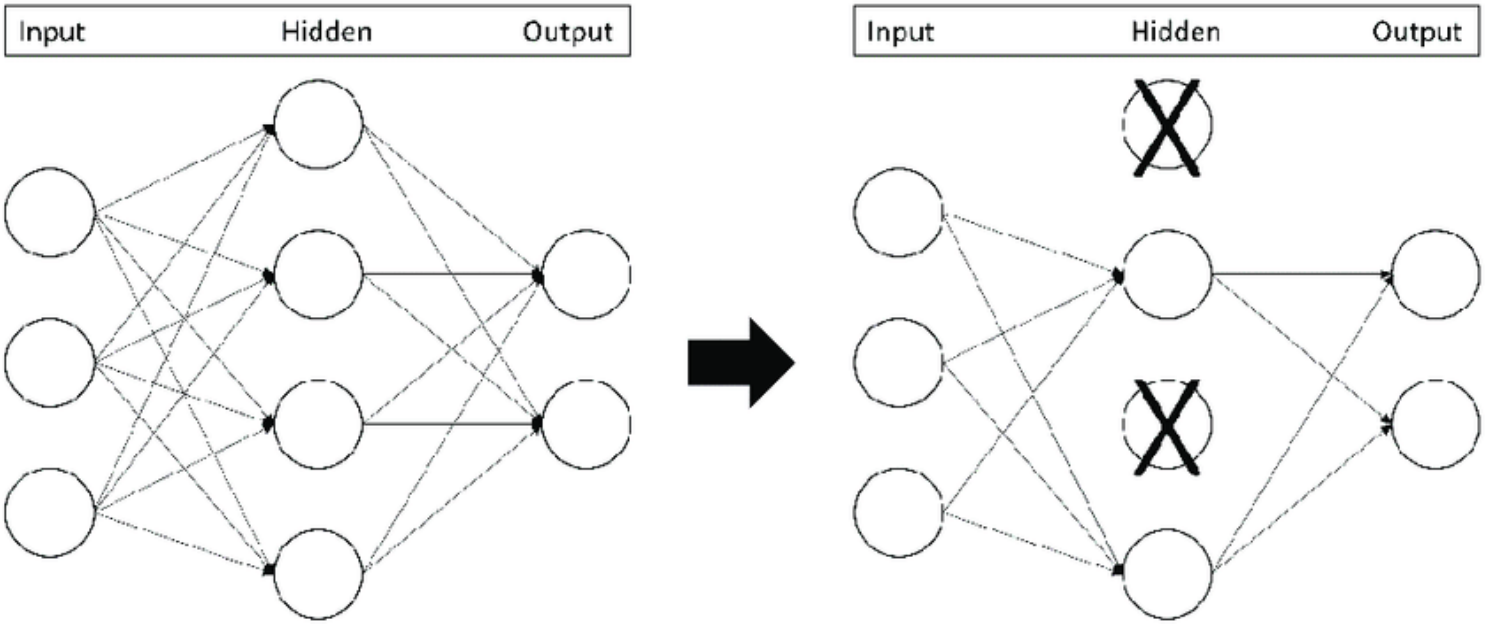
Dropout

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

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

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

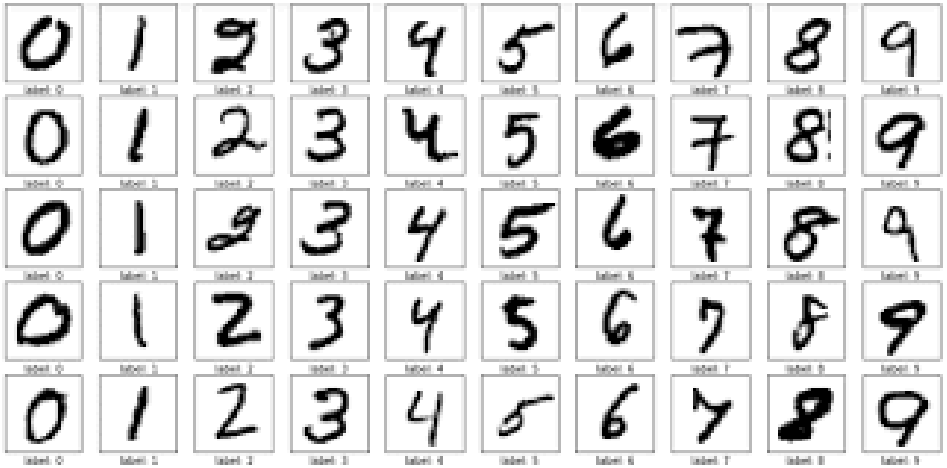
Limitations for Fully Connected (Dense) Networks

- FFNN offers data-dependent nonlinear transformation

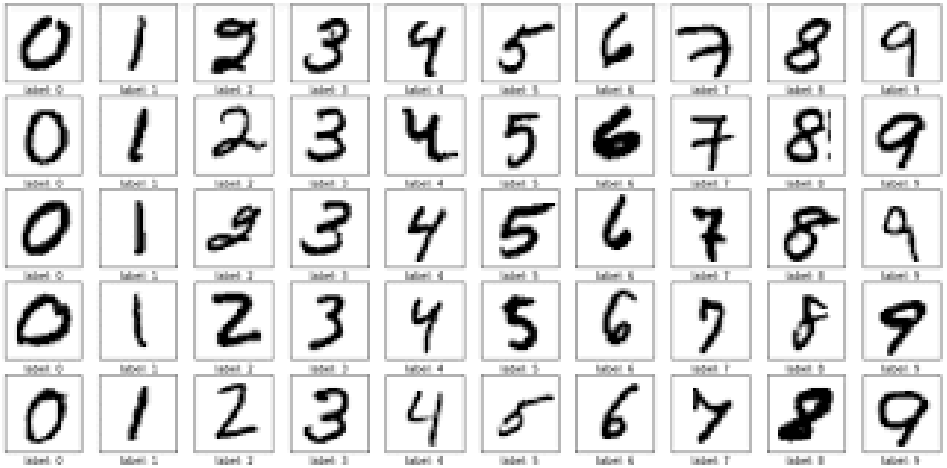
$$\mathbf{z}[n] = \phi(\mathbf{x}[n]) = g(\mathbf{W}^{(1)}\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}^{(2)}\mathbf{z}[n])$$

- 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

Motivation for CNNs

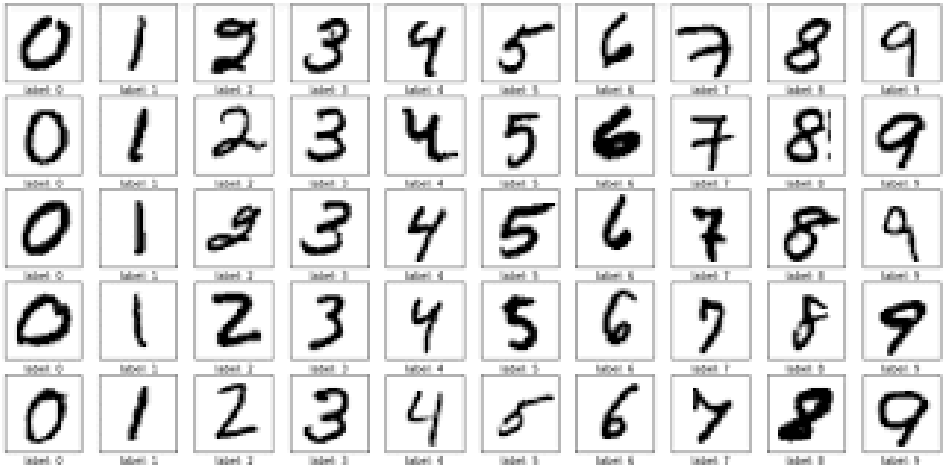


Motivation for CNNs



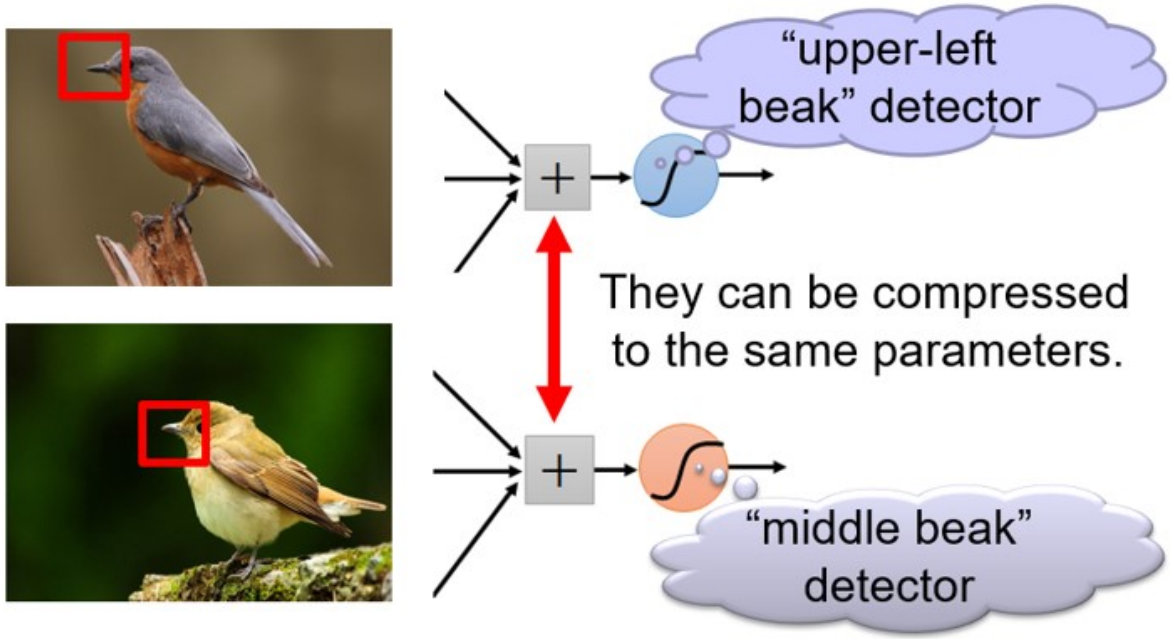
- There could be natural variations among handwritten digits
  - Identity of a digit should be invariant to translation, scaling and (small) rotation
  - Further, the network should be robust to elastic deformations

Motivation for CNNs

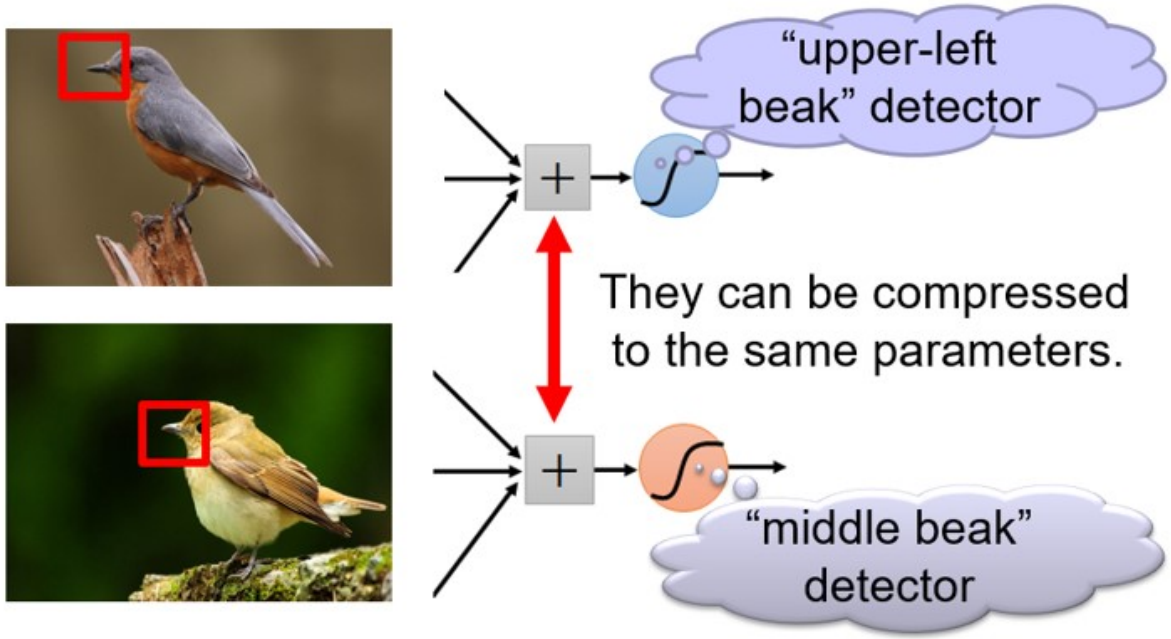


- There could be natural variations among handwritten digits
  - Identity of a digit should be invariant to translation, scaling and (small) rotation
  - Further, the network should be robust to elastic deformations
- One approach is vectorize the input, and train a dense (FC) network
  - Given a large dataset, the network in principle yields good solution
  - However, the key property of the images is ignored - nearby pixels are strongly correlated.

Local Feature Detectors



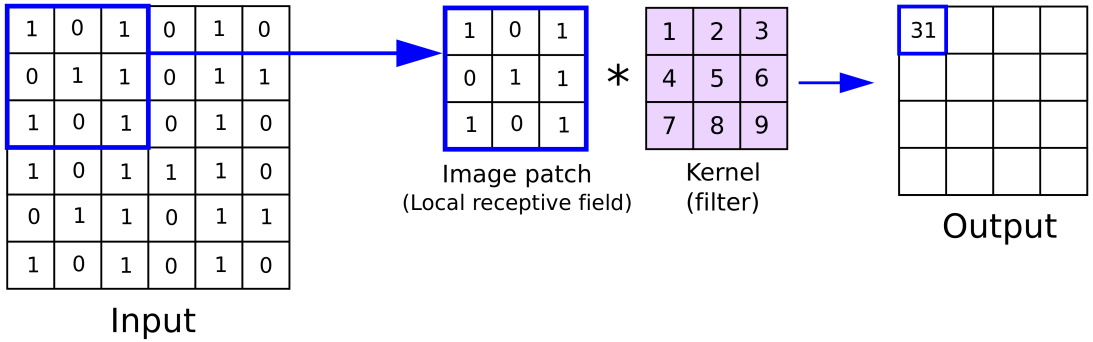
Local Feature Detectors



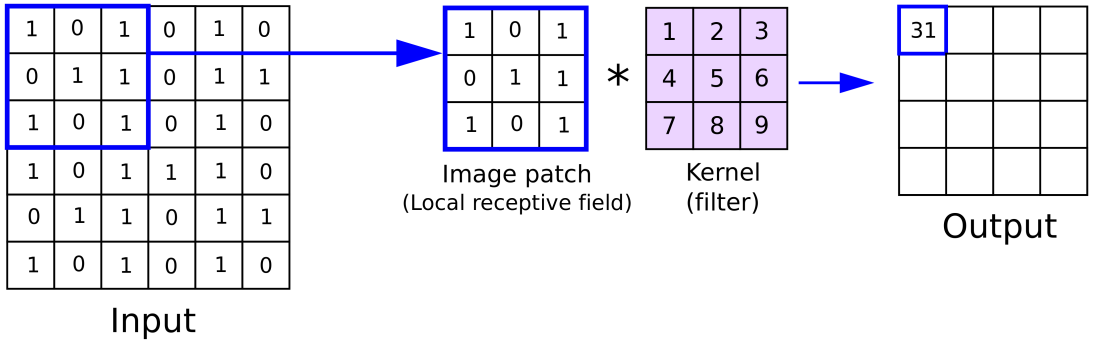
- Some patterns are localized to small regions in the image
- Some patterns can appear in different regions of the image
  - They can be detected using the same kernel (shared parameters)
- Consider a large number of 'small' detectors moving around the image

$$\mathbf{Z}_k[i,j] = \phi\left(\mathbf{X}[i,j] * \mathbf{W}_k[i,j]\right), \qquad k = 1, 2, 3, \ldots K$$

CNN

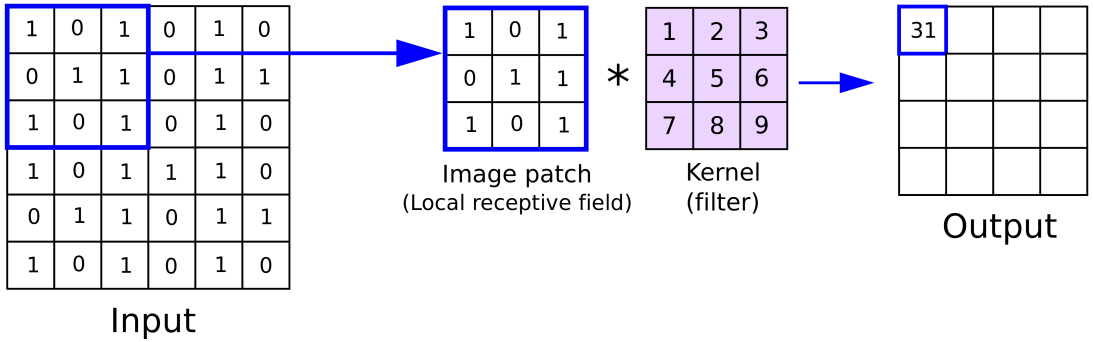

1 0 1 0 1 0
0 1 1 0 1 1
1 0 1 0 1 0
1 0 1 1 1 0
0 1 1 0 1 1
1 0 1 0 1 0
Input
1 0 1
0 1 1
1 0 1
Image patch
(Local receptive field)
*
1 2 3
4 5 6
7 8 9
Kernel
(filter)
31
Output

CNN



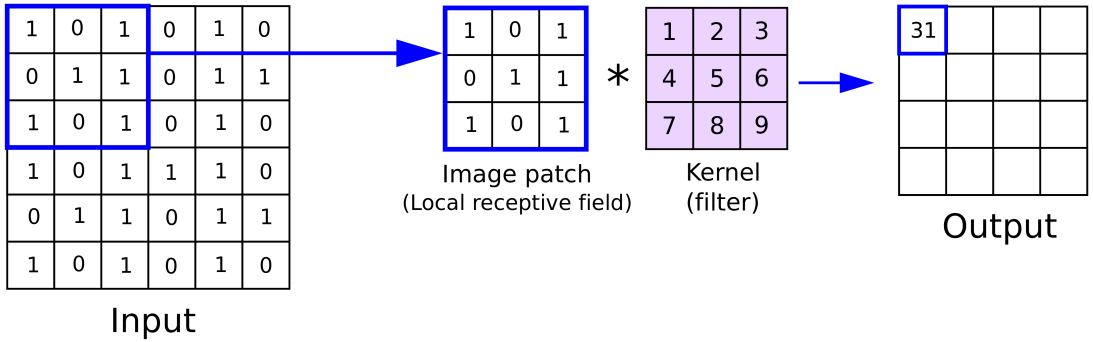
- Extract local features that depend on small subregions of the image

CNN



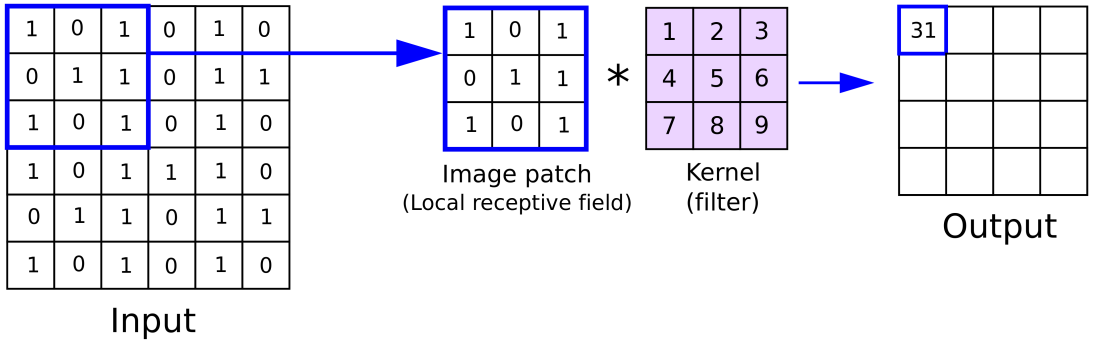
- Extract local features that depend on small subregions of the image
- Local feature detectors can be shared across the image - translations

CNN



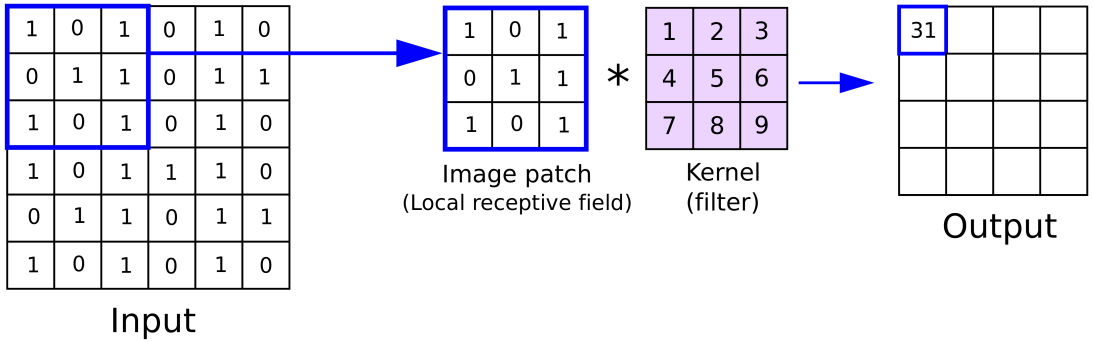
- Extract local features that depend on small subregions of the image
- Local feature detectors can be shared across the image - translations
- Information from local features is merged to detect global patterns

CNN



- Extract local features that depend on small subregions of the image
- Local feature detectors can be shared across the image - translations
- Information from local features is merged to detect global patterns
- These notions are incorporated in CNN though following mechanisms

CNN



- Extract local features that depend on small subregions of the image
- Local feature detectors can be shared across the image - translations
- Information from local features is merged to detect global patterns
- These notions are incorporated in CNN though following mechanisms
  - Local receptive field (kernel-size), weight sharing (stride), subsampling (pooling)

Why Pooling?

Why Pooling?

- The CNN kernels (FIR filters) can be interpreted as bandpass filters

Why Pooling?

- The CNN kernels (FIR filters) can be interpreted as bandpass filters
- A band-limited signal can be downsampled (Nyquist)

Why Pooling?

- The CNN kernels (FIR filters) can be interpreted as bandpass filters
- A band-limited signal can be downsampled (Nyquist)
- Feature maps can be subsampled to achieve scale invariance

Why Pooling?

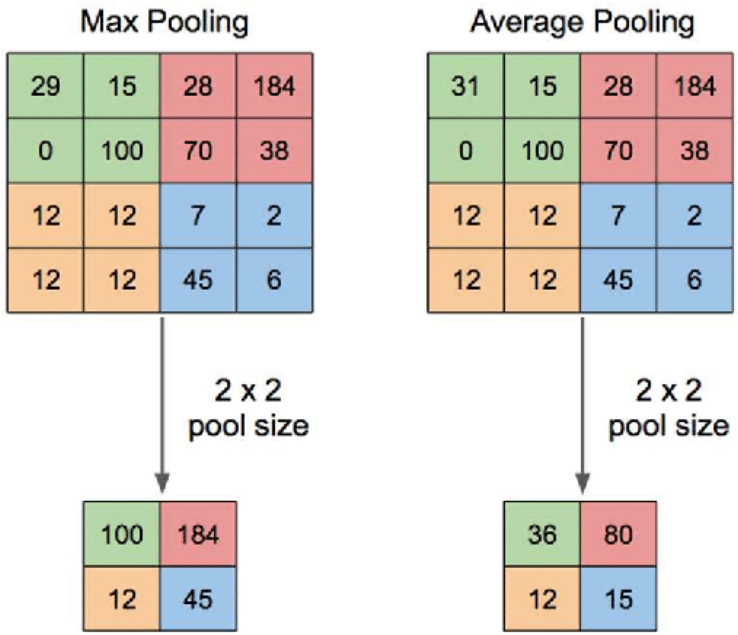
- The CNN kernels (FIR filters) can be interpreted as bandpass filters
- A band-limited signal can be downsampled (Nyquist)
- Feature maps can be subsampled to achieve scale invariance
  - Average pooling considers average of a subsampling window

Why Pooling?

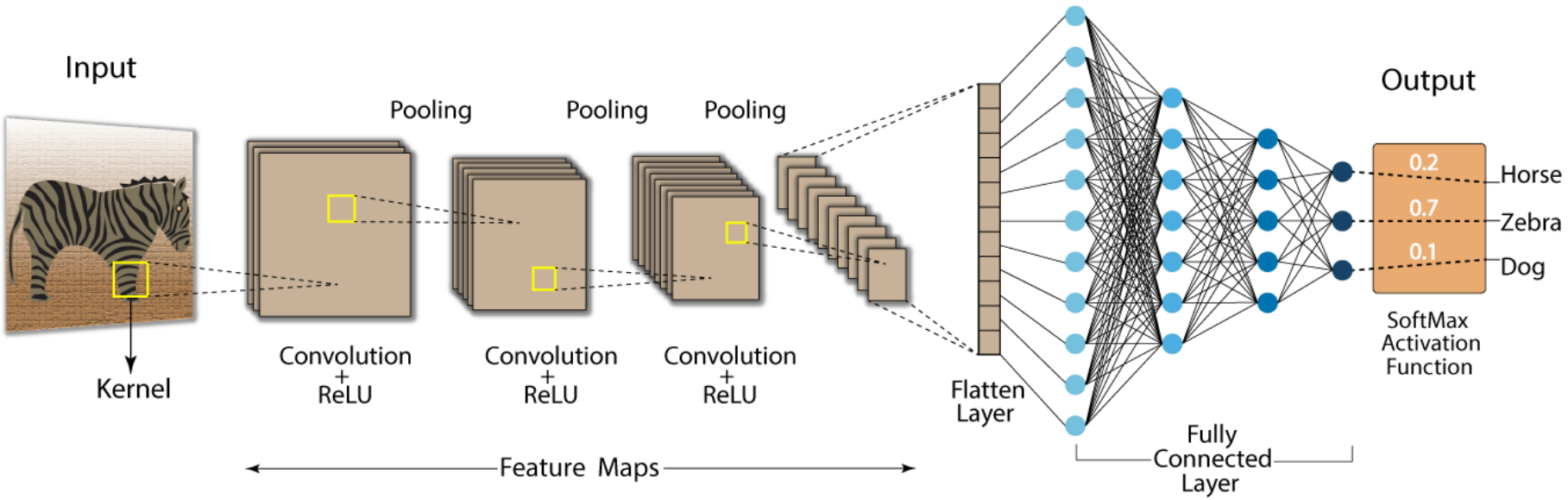
- The CNN kernels (FIR filters) can be interpreted as bandpass filters
- A band-limited signal can be downsampled (Nyquist)
- Feature maps can be subsampled to achieve scale invariance
  - Average pooling considers average of a subsampling window
  - Maxpooling considers maximum of a subsampling window

Why Pooling?

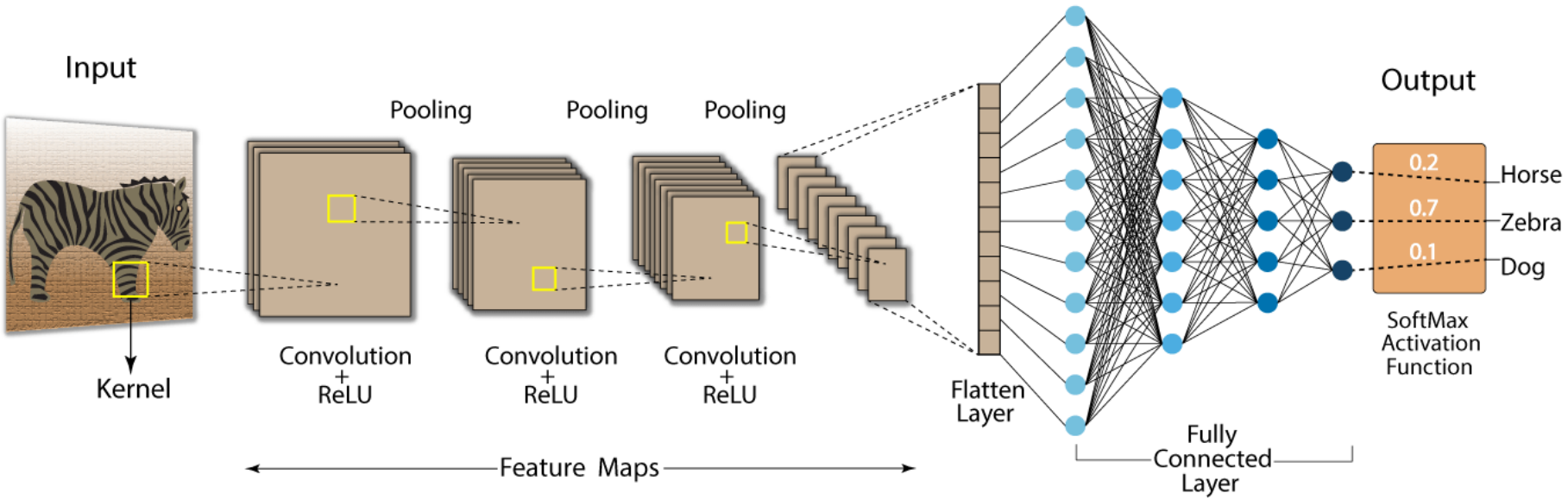
- The CNN kernels (FIR filters) can be interpreted as bandpass filters
- A band-limited signal can be downsampled (Nyquist)
- Feature maps can be subsampled to achieve scale invariance
  - Average pooling considers average of a subsampling window
  - Maxpooling considers maximum of a subsampling window



CNN Classifier Architecture

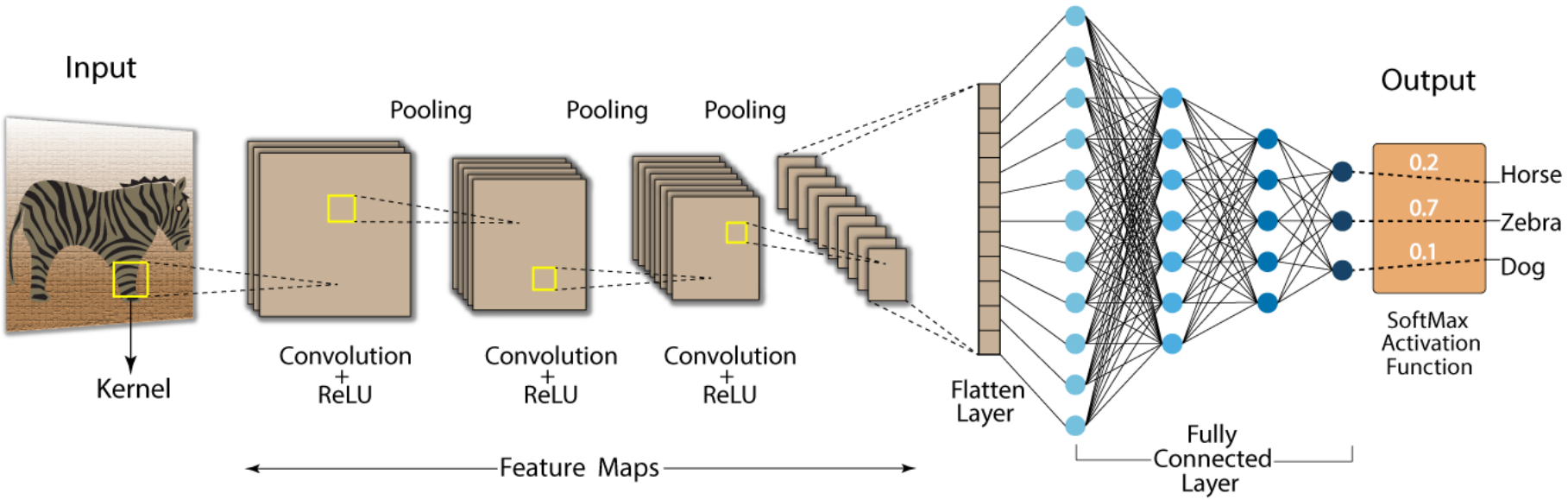


CNN Classifier Architecture



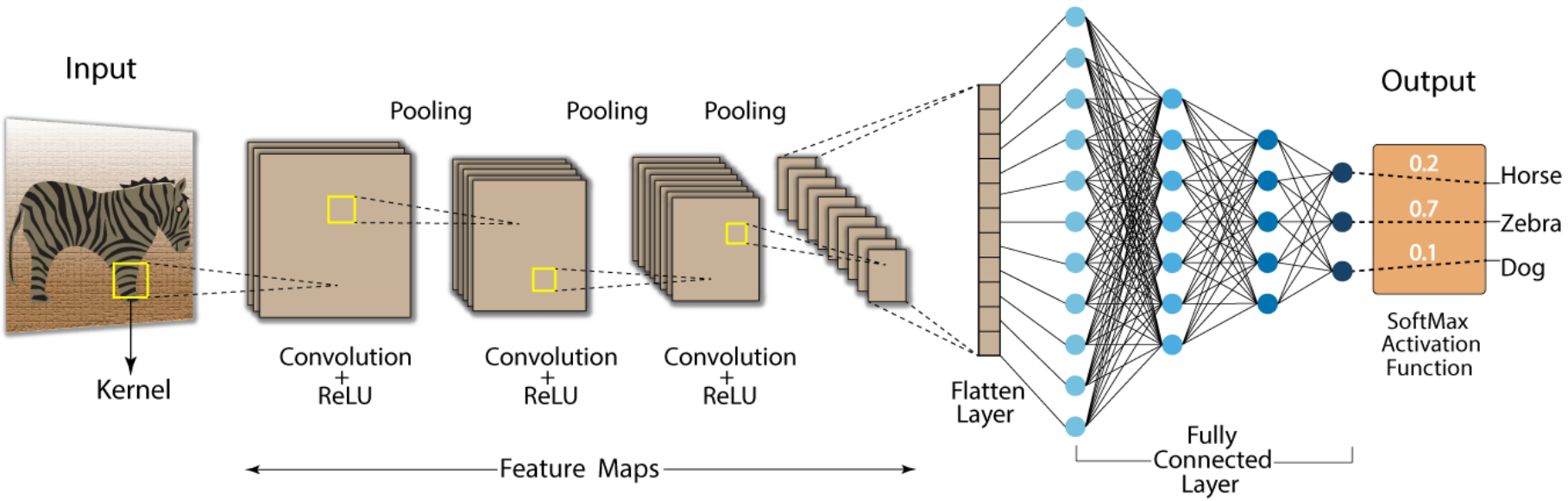
- Convolution layers extract local features from the input image

CNN Classifier Architecture



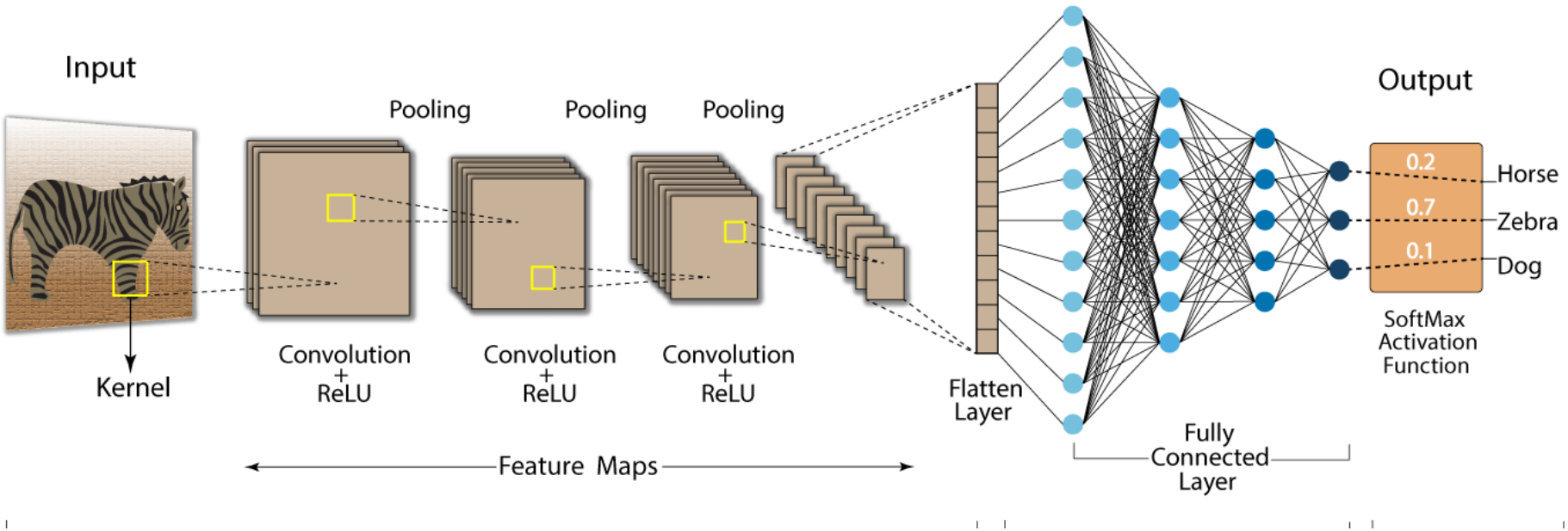
- Convolution layers extract local features from the input image
- The outputs of local detectors are flattened & given to dense network

CNN Classifier Architecture



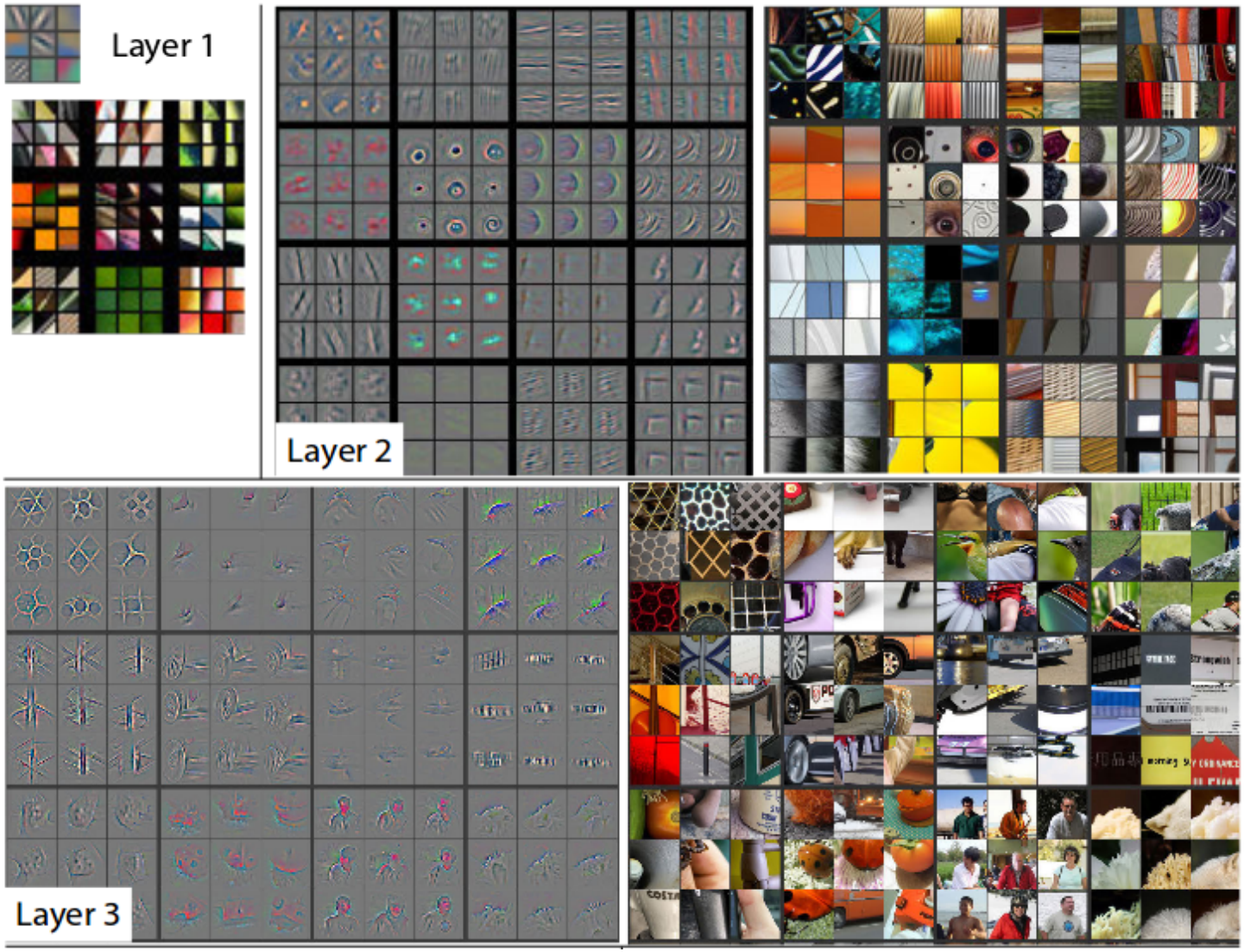
- Convolution layers extract local features from the input image
- The outputs of local detectors are flattened & given to dense network
- Dense layers performs classification on the detected features

CNN Classifier Architecture



- Convolution layers extract local features from the input image
- The outputs of local detectors are flattened & given to dense network
- Dense layers performs classification on the detected features
- CNNs replaced the hand-crafted features with self-learned features

CNN Feature Map Visualization



Homework

Homework

- The output of a convolution layer is given by

$$
\begin{aligned}
z[n] &= \phi\left(x[n] * w_c[n]\right) \\
&= \phi\left(\sum_k w_c[k]x[n-k]\right) \qquad n = 1, 2, \cdots N
\end{aligned}
$$

Homework

- The output of a convolution layer is given by

$$\begin{aligned} z[n] &= \phi\left(x[n] * w_c[n]\right) \\ &= \phi\left(\sum_k w_c[k]x[n-k]\right) \qquad n = 1, 2, \cdots N \end{aligned}$$

- Let the backpropagated gradients $\frac{\partial J()}{\partial z[n]}$ are known

Homework

- The output of a convolution layer is given by

$$z[n] = \phi\left(x[n] * w_c[n]\right)$$
$$= \phi\left(\sum_k w_c[k]x[n-k]\right) \qquad n = 1, 2, \cdots N$$

- Let the backpropagated gradients $\frac{\partial J()}{\partial z[n]}$ are known
- Derive update equations for the weights in the convolution layer.

$$\frac{\partial J()}{\partial w_c[k]} = ?$$

Homework

- The output of a convolution layer is given by

$$\begin{aligned} z[n] &= \phi\left(x[n] * w_c[n]\right) \\ &= \phi\left(\sum_k w_c[k]x[n-k]\right) \qquad n = 1, 2, \cdots N \end{aligned}$$

- Let the backpropagated gradients $\frac{\partial J()}{\partial z[n]}$ are known
- Derive update equations for the weights in the convolution layer.

$$\frac{\partial J()}{\partial w_c[k]} = ?$$

- Write down the batch update equations for $w_c[k]$.

Homework

- The output of a convolution layer is given by

$$\begin{aligned} z[n] &= \phi\left(x[n] * w_c[n]\right) \\ &= \phi\left(\sum_k w_c[k]x[n-k]\right) \qquad n = 1, 2, \cdots N \end{aligned}$$

- Let the backpropagated gradients $\frac{\partial J()}{\partial z[n]}$ are known
- Derive update equations for the weights in the convolution layer.

$$\frac{\partial J()}{\partial w_c[k]} = ?$$

- Write down the batch update equations for $w_c[k]$.
- How do you modify it to obtain gradients w.r.t the input $x[n]$?

Sequential Models

- Recurrent Neural Networks (RNNs) - Implicit incorporation of context

Sequential Models

- Recurrent Neural Networks (RNNs) - Implicit incorporation of context
  - Encapsulate the past history in the hidden state-vector

$$\mathbf{h}[n-1] = \phi(\cdots x[n-3],\, x[n-2],\, x[n-1])$$

Sequential Models

- Recurrent Neural Networks (RNNs) - Implicit incorporation of context
  - Encapsulate the past history in the hidden state-vector

$$\mathbf{h}[n-1] = \phi(\cdots x[n-3],\, x[n-2],\, x[n-1])$$

  - The state-vector can be recursively updated as

$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}_{hy}\,\mathbf{h}[n])$$

Sequential Models

- Recurrent Neural Networks (RNNs) - Implicit incorporation of context
  - Encapsulate the past history in the hidden state-vector

$$\mathbf{h}[n-1] = \phi(\cdots x[n-3],\, x[n-2],\, x[n-1])$$

  - The state-vector can be recursively updated as

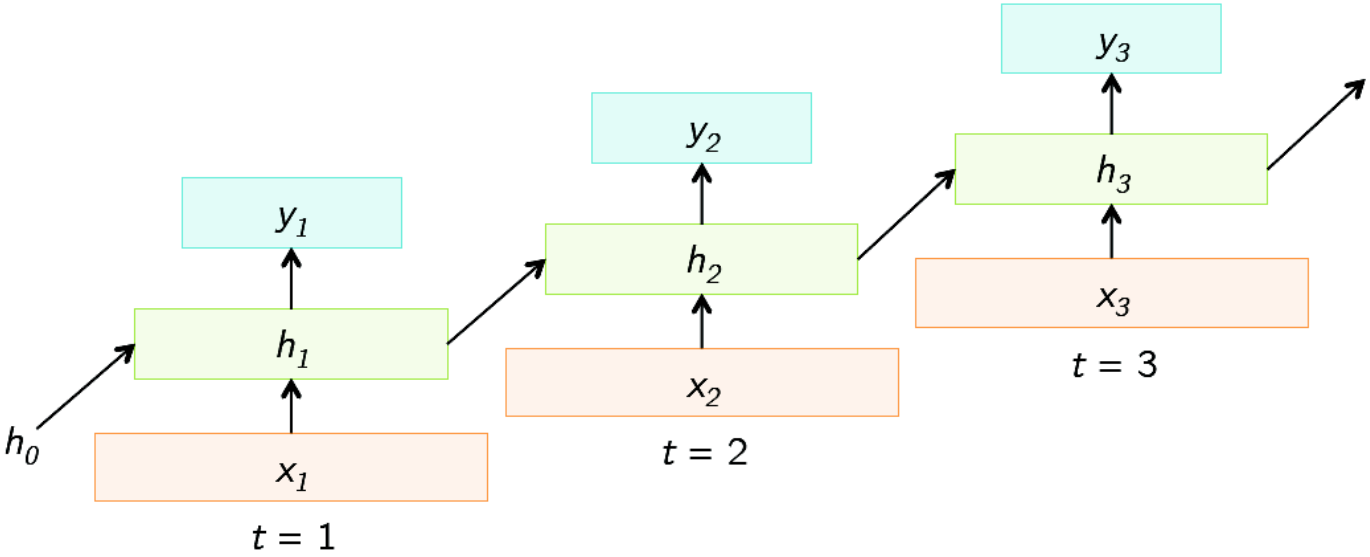
$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n]) \qquad \mathbf{y}[n] = f(\mathbf{W}_{hy}\,\mathbf{h}[n])$$

  - Variants of RNN: LSTMs and GRU
- Transformers - Explicit incorporation of context

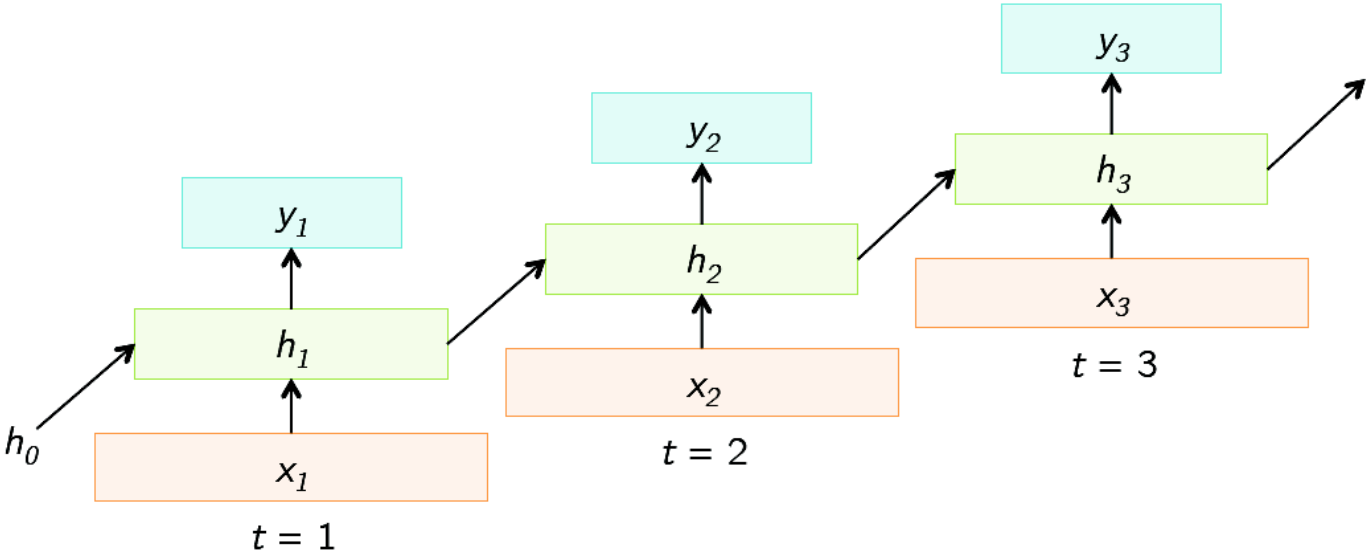
$$\mathbf{z}[n] = g\left(\sum_{k=n-N}^{n} a_{nk}\mathbf{W}^{(1)}\mathbf{x}[k]\right)$$

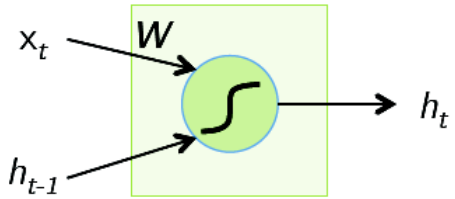
a_{nk} is the attention of the k^{th} frame at the n^{th} instant (similarity)

Recurrent Neural Networks

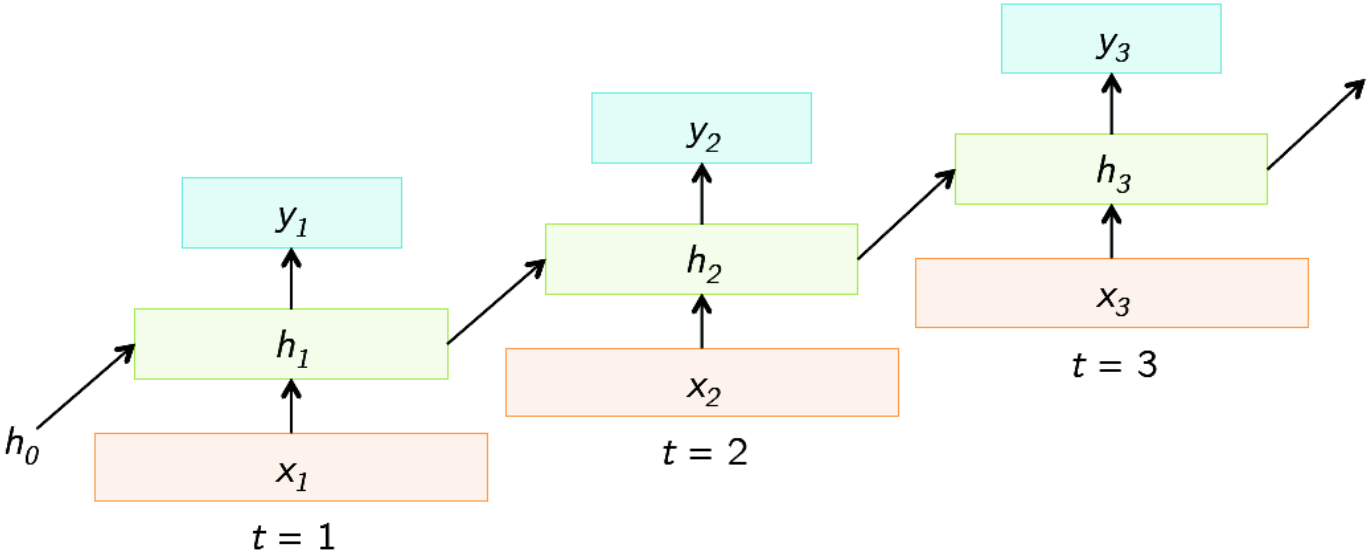


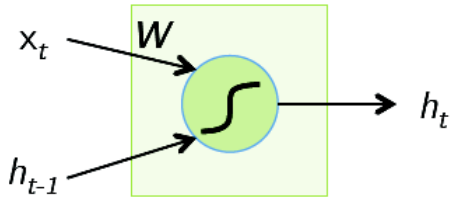
Recurrent Neural Networks



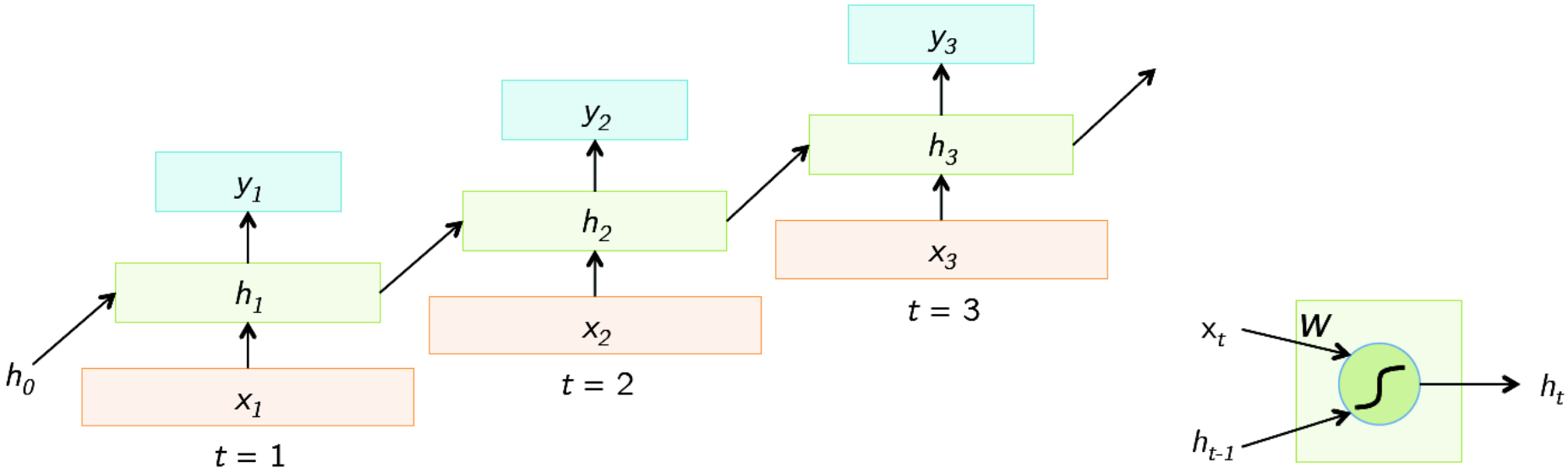


Recurrent Neural Networks





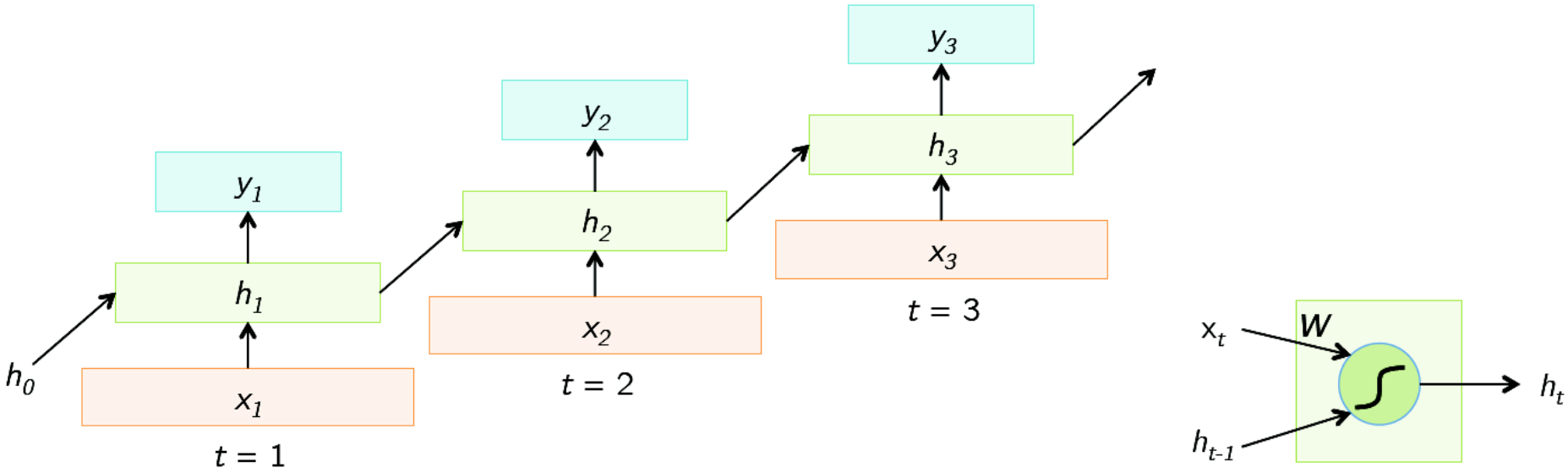
Recurrent Neural Networks



- RNN cell takes current input $\mathbf{x}[n]$ and previous state $\mathbf{h}[n-1]$ as inputs to produce the current state

$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n]) = g\left([\mathbf{W}_{hh}\,\mathbf{W}_{xh}]\begin{bmatrix}\mathbf{h}[n-1]\\ \mathbf{x}[n]\end{bmatrix}\right)$$

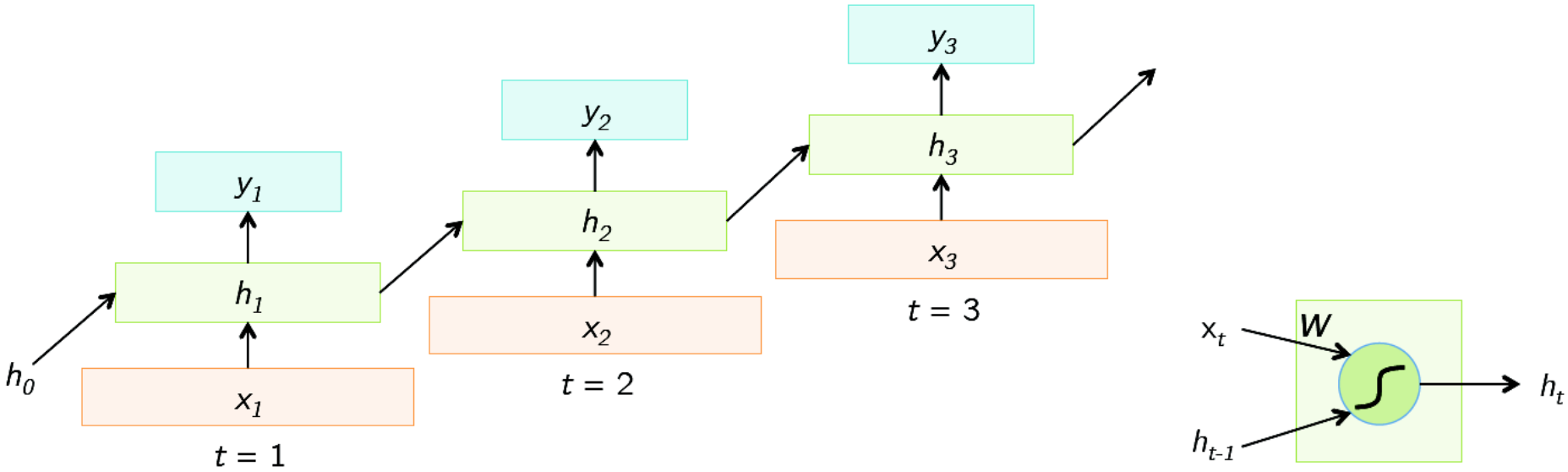
Recurrent Neural Networks



- RNN cell takes current input $\mathbf{x}[n]$ and previous state $\mathbf{h}[n-1]$ as inputs to produce the current state

$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n]) = g\left([\mathbf{W}_{hh}\,\mathbf{W}_{xh}]\begin{bmatrix}\mathbf{h}[n-1]\\ \mathbf{x}[n]\end{bmatrix}\right)$$

Recurrent Neural Networks

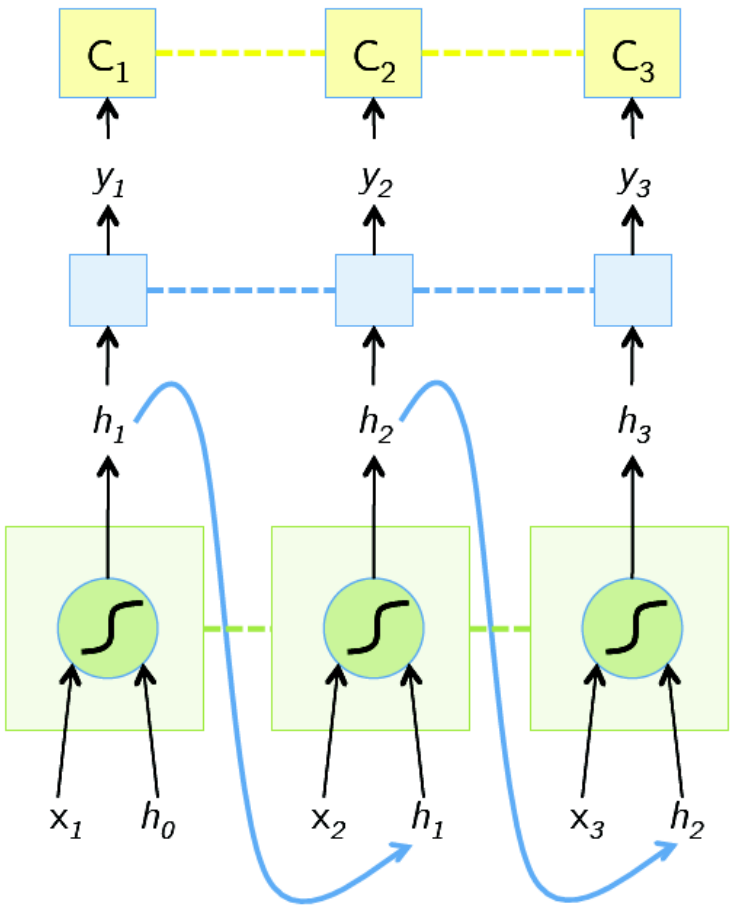


- RNN cell takes current input $\mathbf{x}[n]$ and previous state $\mathbf{h}[n-1]$ as inputs to produce the current state

$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n]) = g\left([\mathbf{W}_{hh}\,\mathbf{W}_{xh}]\begin{bmatrix}\mathbf{h}[n-1]\\ \mathbf{x}[n]\end{bmatrix}\right)$$

- The weights $\mathbf{W} = [\mathbf{W}_{hh}\,\mathbf{W}_{xh}]$ are shared across the time-steps

RNN Forward Propagation



- Forward propagate the inputs

$$\mathbf{h}[n] = g(\mathbf{W}_{hh}\,\mathbf{h}[n-1] + \mathbf{W}_{xh}\,\mathbf{x}[n])$$

$$\mathbf{y}[n] = f(\mathbf{W}_{hy}\,\mathbf{h}[n])$$

$$J_n(\mathbf{W}) = L(\mathbf{y}[n],\,\mathbf{d}[n])$$

$$J(\mathbf{W}) = \sum_n J_n(\mathbf{W})$$

- Estimating $\mathbf{W} = [\mathbf{W}_{hh}\,\mathbf{W}_{xh}\,\mathbf{W}_{hy}]$

$$\mathbf{W}_* = \arg\min_{\mathbf{W}} J(\mathbf{W})$$

- Backpropagate the error to adjust the parameters

RNN Application - Sentiment Classification

- Classify a user review as 'positive' or 'negative'
  - Movie review from IMDB
  - Restaurant review from Swiggy
  - Mobile phone review from Amazon

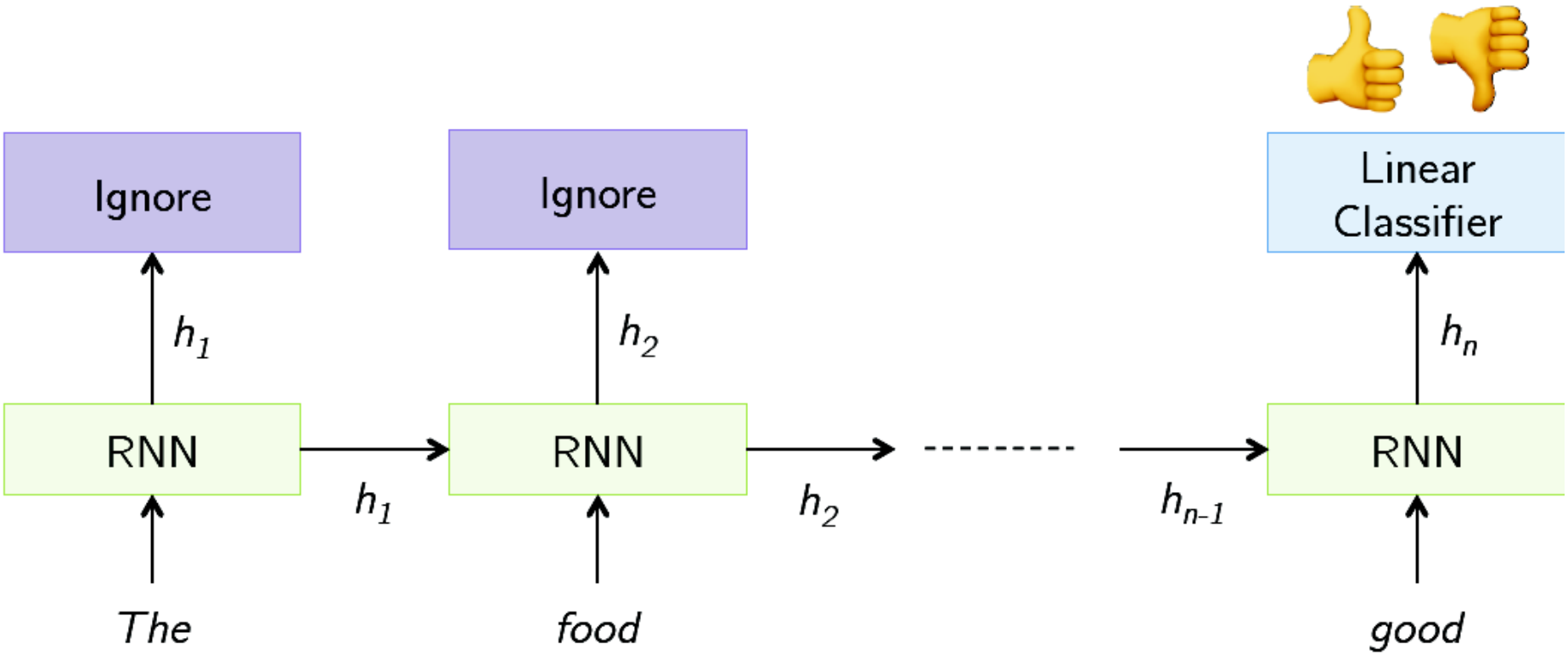
RNN Application - Sentiment Classification

- Classify a user review as 'positive' or 'negative'
  - Movie review from IMDB
  - Restaurant review from Swiggy
  - Mobile phone review from Amazon
- Build a binary classifier for sentiment analysis
  - **Input:** A sequence of words in the review (one or more sentences)
  - **Output:** Positive or Negative review

RNN Application - Sentiment Classification

- Classify a user review as 'positive' or 'negative'
  - Movie review from IMDB
  - Restaurant review from Swiggy
  - Mobile phone review from Amazon
- Build a binary classifier for sentiment analysis
  - **Input:** A sequence of words in the review (one or more sentences)
  - **Output:** Positive or Negative review
  - "The food was really good!"
  - "The chicken crossed the road because it was uncooked"
- Need to analyze the sequence of words to make a decision
- FFNN are not useful as they consider the review as a set of words.

Sentiment Analysis



Sentiment Analysis

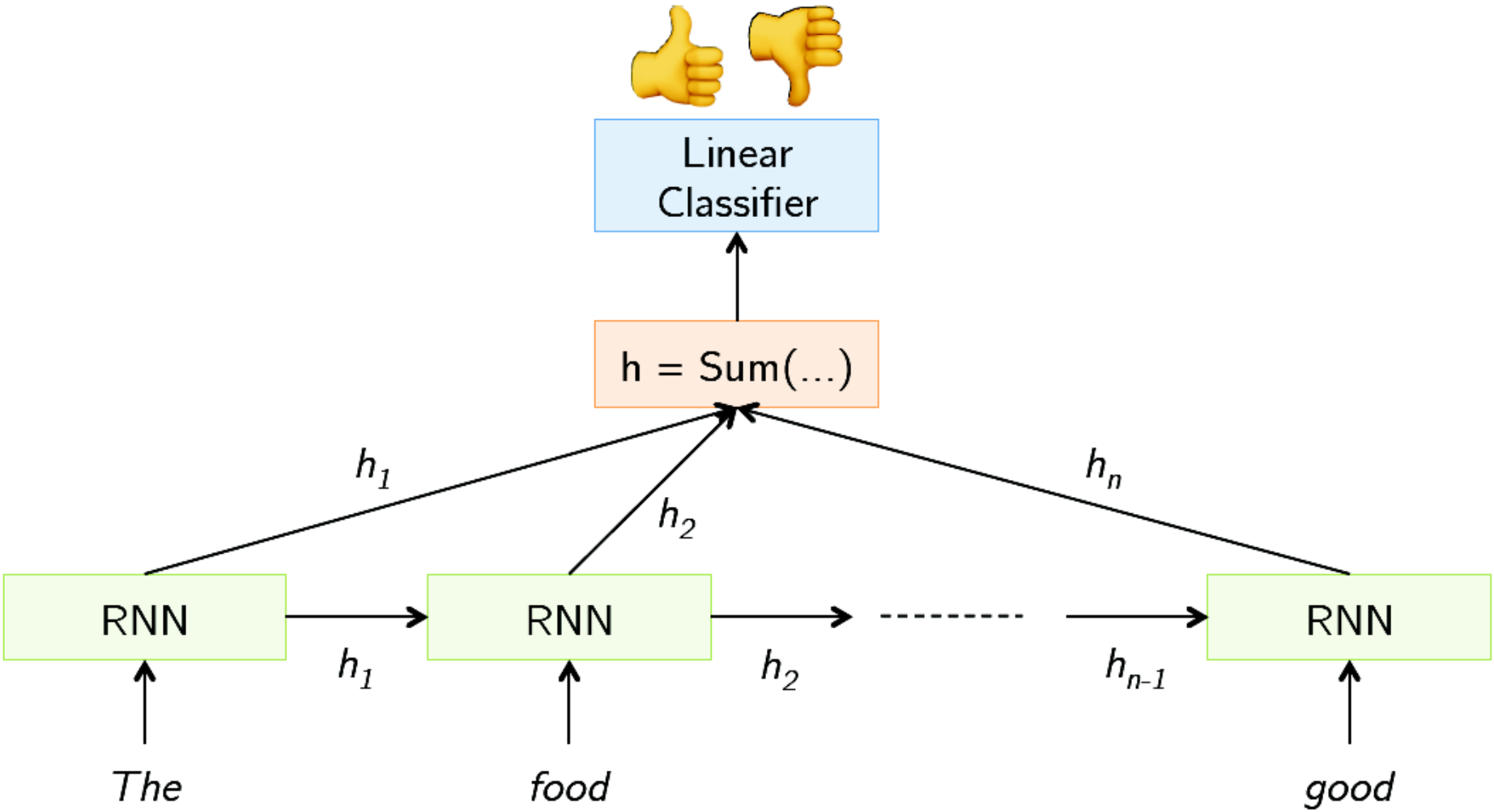
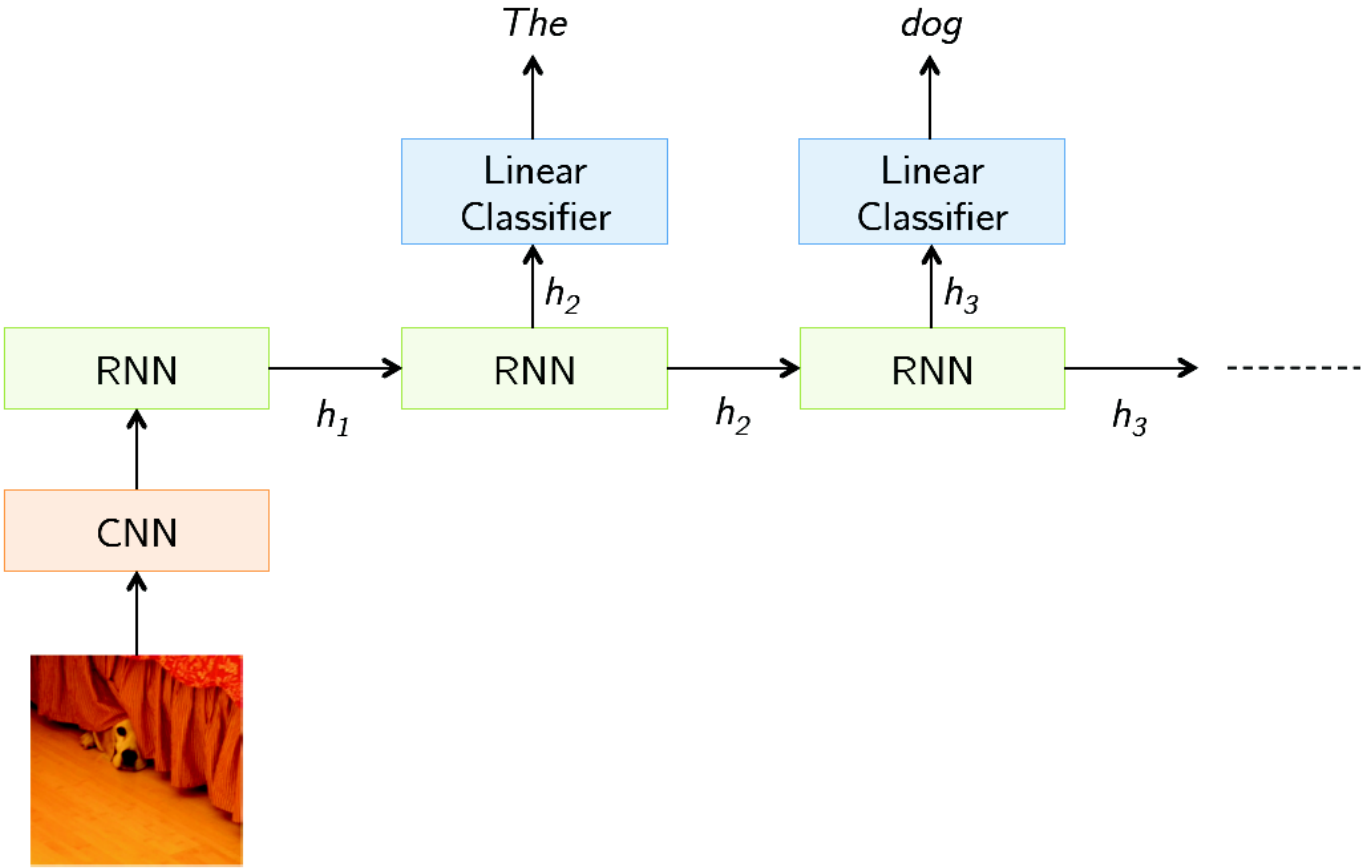


Image Captioning

- Given an image, generate a sentence describing the image content
  - **Input:** Image
  - **Output:** A sentence describing the scene in the mage

Image Captioning

- Given an image, generate a sentence describing the image content
  - **Input:** Image
  - **Output:** A sentence describing the scene in the mage



Neural Image Caption Generator

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A close up of a cat laying on a couch.

Language Modelling

- Predict the next word in a sentence from the previous words
  - **Input:** Last K words in a sentence: $W_{N-K}, \cdots W_N$
  - Output: Predict W_{N+1}

$$P[W_{N+1}/W_N, W_{N-1}, \cdots W_{N-K}]$$

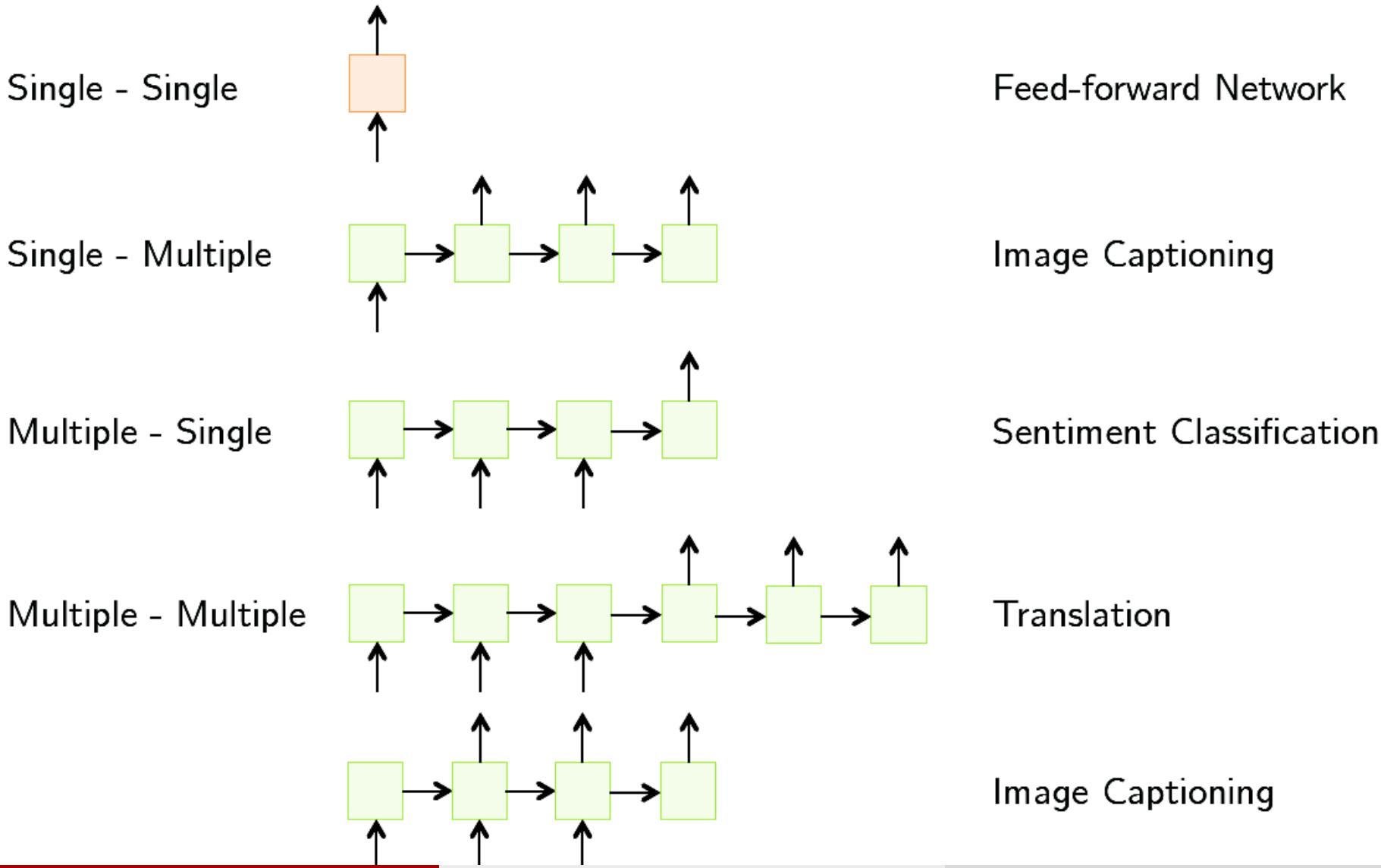
Language Modelling

- Predict the next word in a sentence from the previous words
  - **Input:** Last K words in a sentence: $W_{N-K}, \cdots W_N$
  - Output: Predict W_{N+1}

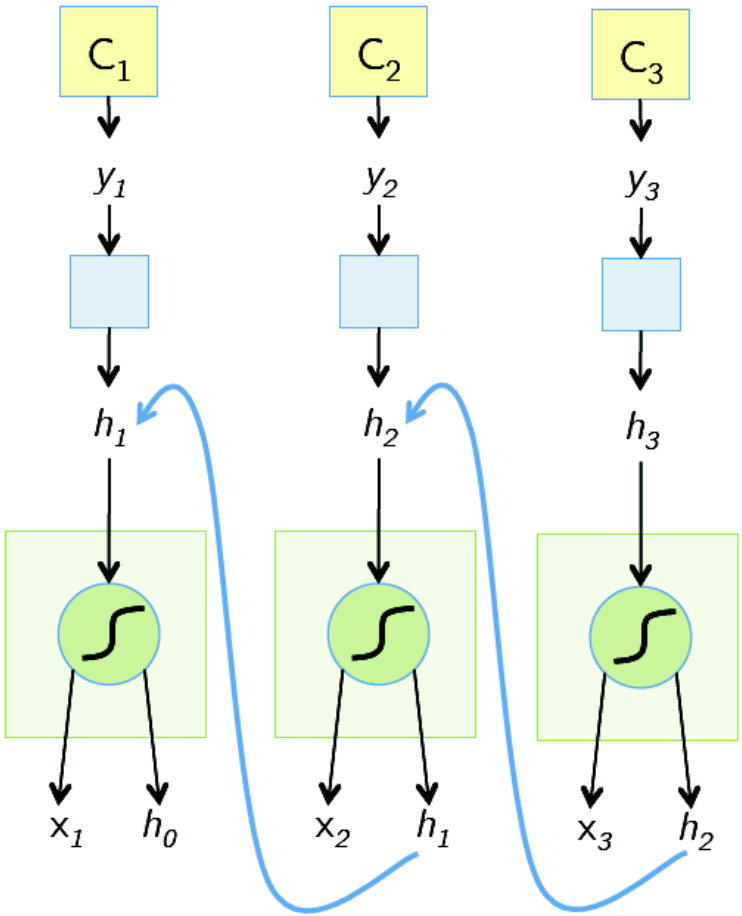
$$P[W_{N+1}/W_N, W_{N-1}, \cdots W_{N-K}]$$

Why, Salisbury must find his flesh and thought. That which I am not aps, not a man and in fire. To show the reining of the raven and the wars. To grace my hand reproach within, and not a fair are hand. That Caesar and my goodly father's world;! When I was heaven of presence and our fleets. We spare with hours, but cut thy council I am great,! Murdered and by thy master's ready there. My power to give thee but so much as hell Some service in the noble bondman here, Would show him to her wine.

RNN Architectures

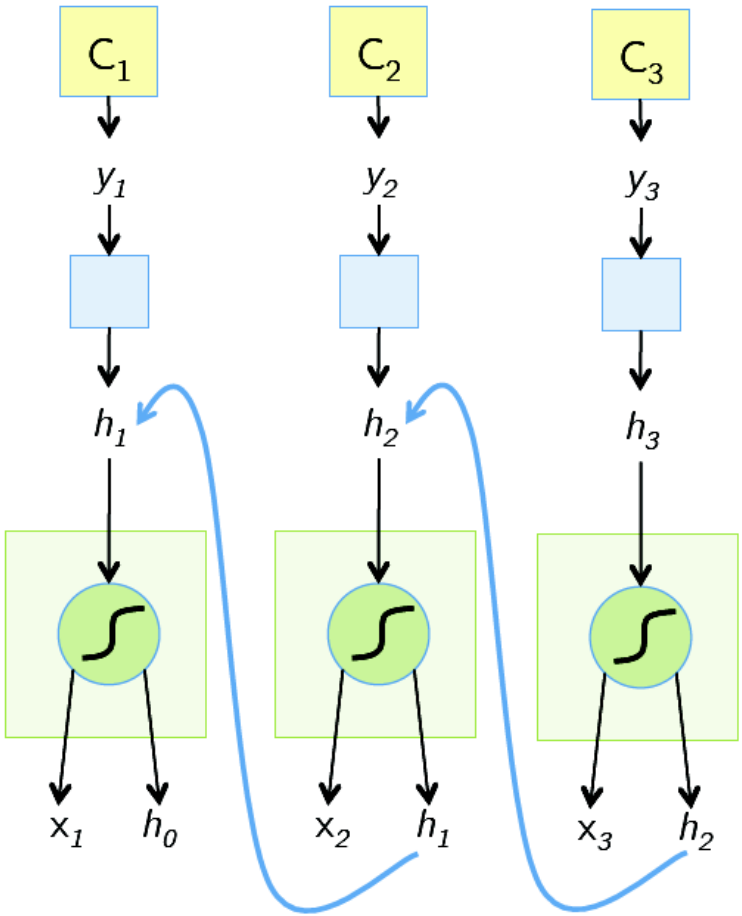


Temporal Backpropagation



- Unfold the network during forward pass
- Treat unfolded network as a big FFN
- This big FFN takes entire time-series as input

Temporal Backpropagation



- Unfold the network during forward pass
- Treat unfolded network as a big FFN
- This big FFN takes entire time-series as input
- Compute gradients through usual backpropagation
- Update the shared weights

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

- Error at n^{th} instant depends on all the previous hidden states $\mathbf{h}[k]$, $k <= n$

Issues with RNNs

Issues with RNNs

Effect of 1^{st} hidden state on error at the n^{th} instant

$$\frac{\partial J_n(\mathbf{W})}{\partial \mathbf{h}[1]} = \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[1]}$$

Issues with RNNs

Effect of 1^{st} hidden state on error at the n^{th} instant

$$\frac{\partial J_n(\mathbf{W})}{\partial \mathbf{h}[1]} = \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[1]}$$

$$= \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[n]} \frac{\partial \mathbf{h}[n]}{\partial \mathbf{h}[n-1]} \cdots \frac{\partial \mathbf{h}[3]}{\partial \mathbf{h}[2]} \frac{\partial \mathbf{h}[2]}{\partial \mathbf{h}[1]}$$

Issues with RNNs

Effect of 1^{st} hidden state on error at the n^{th} instant

$$\frac{\partial J_n(\mathbf{W})}{\partial \mathbf{h}[1]} = \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[1]}$$

$$= \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[n]} \frac{\partial \mathbf{h}[n]}{\partial \mathbf{h}[n-1]} \cdots \frac{\partial \mathbf{h}[3]}{\partial \mathbf{h}[2]} \frac{\partial \mathbf{h}[2]}{\partial \mathbf{h}[1]}$$

- The product of n real numbers can shrink to zero or explode to infinity
- For tanh() and $\sigma()$ activations, the gradients will be less than 1.

Issues with RNNs

Effect of 1^{st} hidden state on error at the n^{th} instant

$$\frac{\partial J_n(\mathbf{W})}{\partial \mathbf{h}[1]} = \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[1]}$$

$$= \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[n]} \frac{\partial \mathbf{h}[n]}{\partial \mathbf{h}[n-1]} \cdots \frac{\partial \mathbf{h}[3]}{\partial \mathbf{h}[2]} \frac{\partial \mathbf{h}[2]}{\partial \mathbf{h}[1]}$$

- The product of n real numbers can shrink to zero or explode to infinity
- For tanh() and $\sigma()$ activations, the gradients will be less than 1.
- For relu(), outputs are not bounded, causing instability
- RNNs suffer from either vanishing gradients or exploding gradients

Issues with RNNs

Effect of 1^{st} hidden state on error at the n^{th} instant

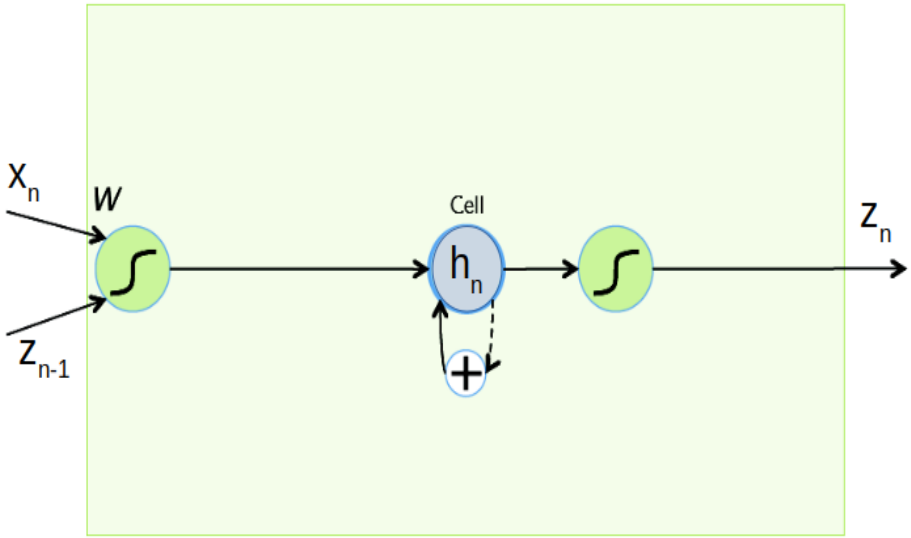
$$\frac{\partial J_n(\mathbf{W})}{\partial \mathbf{h}[1]} = \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[1]}$$

$$= \frac{\partial J_n(\mathbf{W})}{\partial \mathbf{y}[n]} \frac{\partial \mathbf{y}[n]}{\partial \mathbf{h}[n]} \frac{\partial \mathbf{h}[n]}{\partial \mathbf{h}[n-1]} \cdots \frac{\partial \mathbf{h}[3]}{\partial \mathbf{h}[2]} \frac{\partial \mathbf{h}[2]}{\partial \mathbf{h}[1]}$$

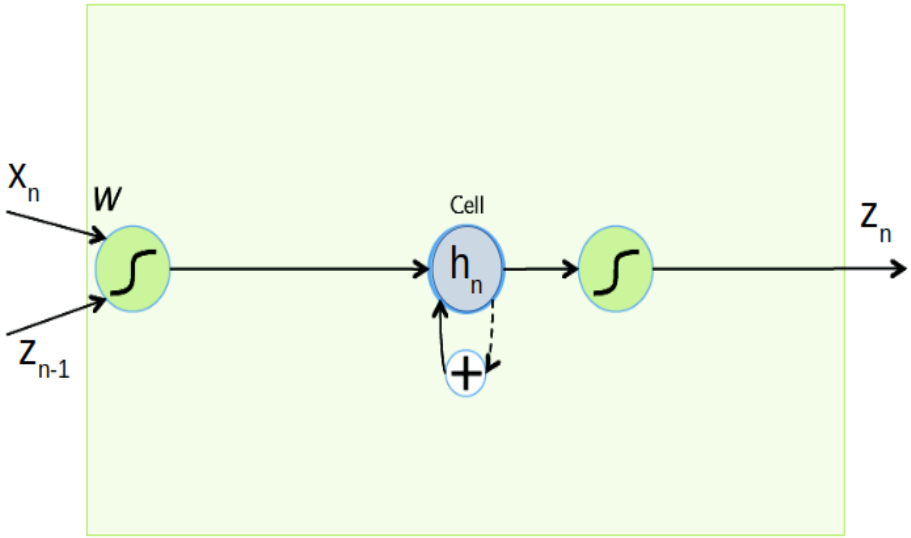
- The product of n real numbers can shrink to zero or explode to infinity
- For tanh() and $\sigma()$ activations, the gradients will be less than 1.
- For relu(), outputs are not bounded, causing instability
- RNNs suffer from either vanishing gradients or exploding gradients
- Introduce identity relation between the hidden states

$$\mathbf{h}[n] = \mathbf{h}[n-1] + g(\mathbf{x}[n]) \qquad \frac{\partial \mathbf{h}[n]}{\partial \mathbf{h}[n-1]} = 1$$

LSTM Idea

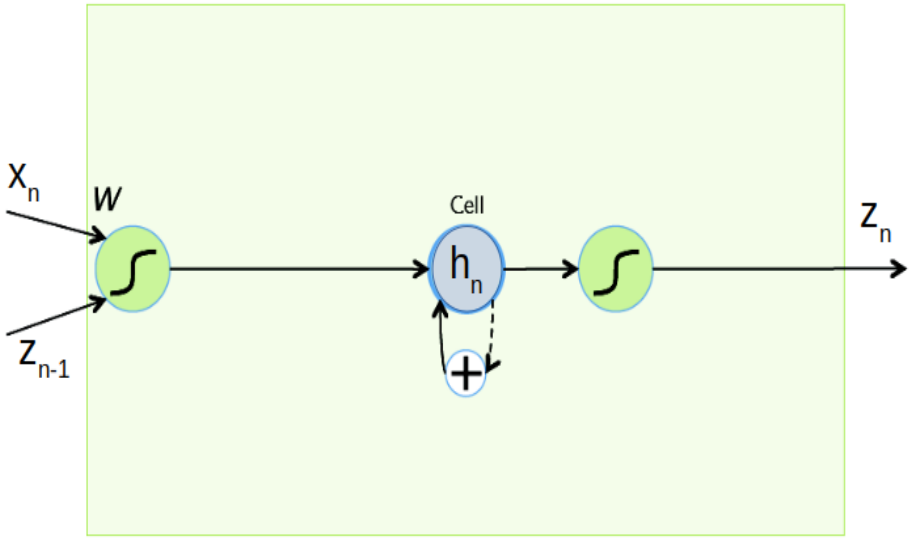


LSTM Idea



- $\mathbf{x}_n$ - current input
- $\mathbf{z}_{n-1}$ - previous output
- $\mathbf{h}_n$ - current cell state

LSTM Idea



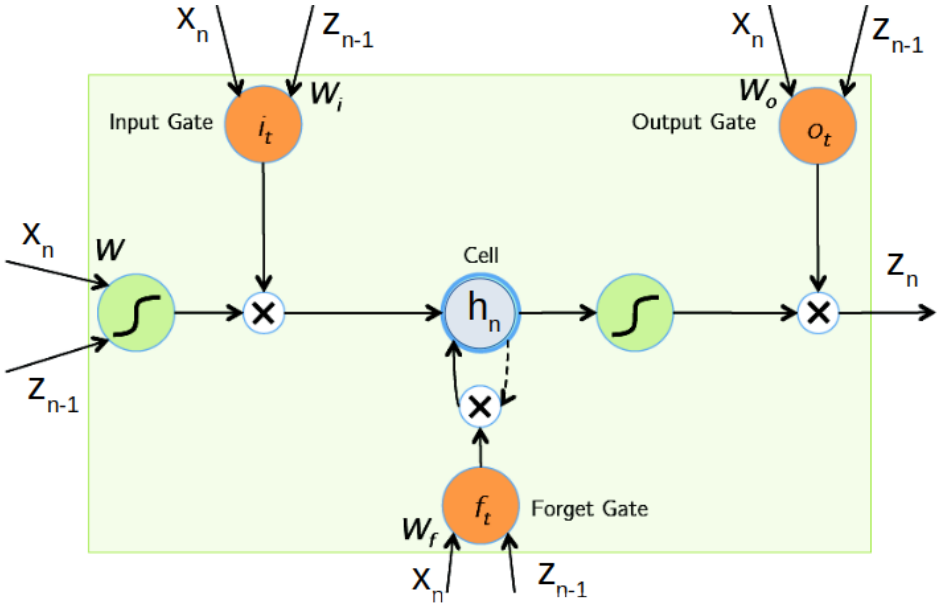
$$\mathbf{h}_n = \mathbf{h}[n-1] + \tanh(\mathbf{W}_{xh}\mathbf{x}_n + \mathbf{W}_{zh}\mathbf{z}_{n-1})$$

$$\mathbf{z}_n = \tanh(\mathbf{h}_n)$$

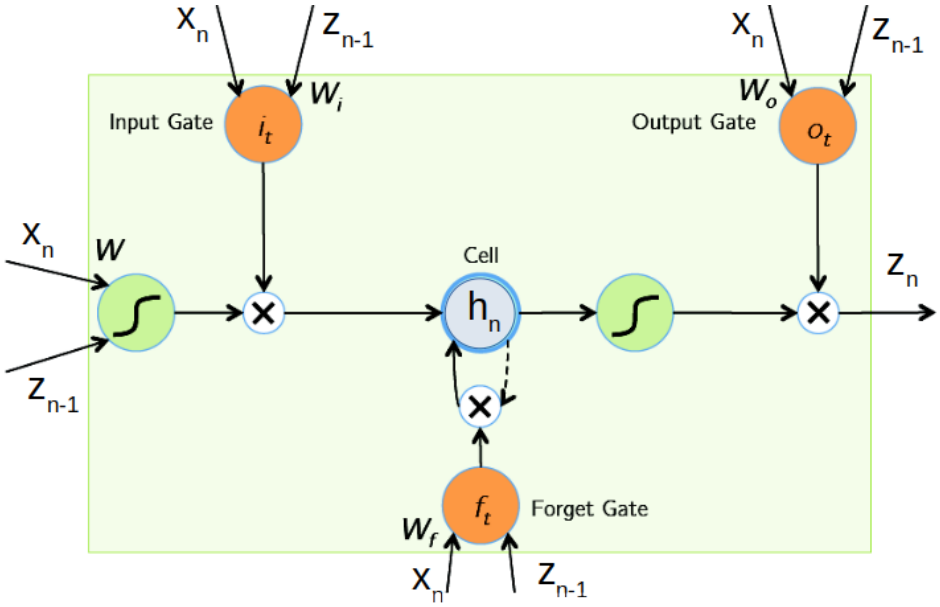
- $\mathbf{x}_n$ - current input
- $\mathbf{z}_{n-1}$ - previous output
- $\mathbf{h}_n$ - current cell state

- State and output are different
- Creates highway path for state (to ensure gradient flow)
- State acts like an accumulator over time (memory)
- Additive update for the state

LSTM Cell

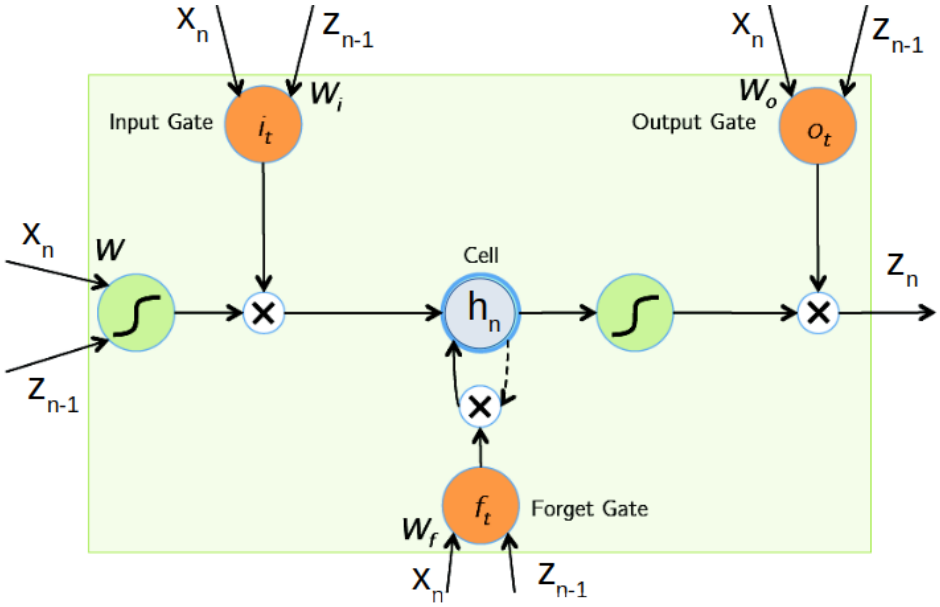

x_n
z_{n-1}
W_i
Input Gate
i_t
x_n
z_{n-1}
W_o
Output Gate
o_t
x_n
W
Cell
h_n
z_n
z_{n-1}
f_t
Forget Gate
W_f
x_n
z_{n-1}

LSTM Cell



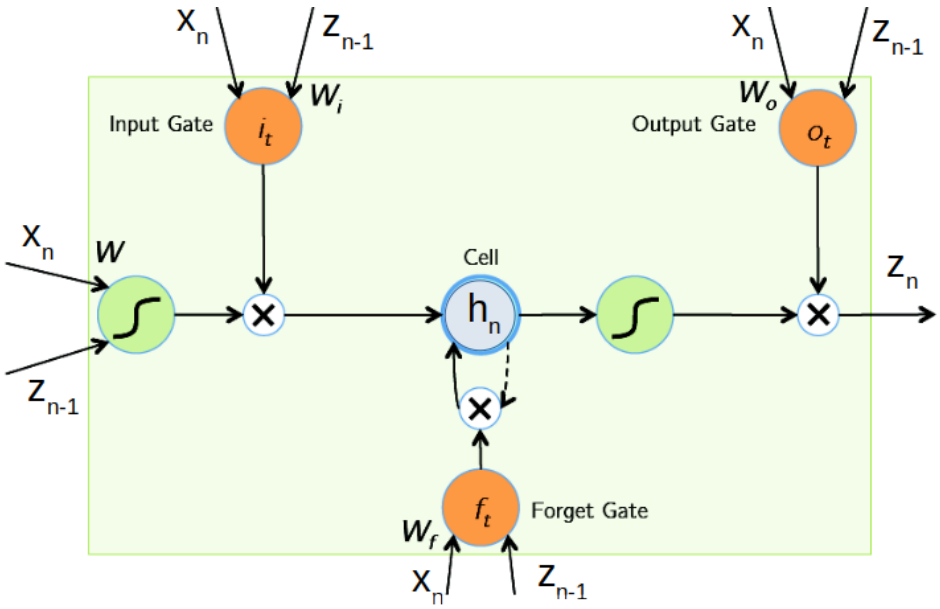
- Gates control flow of info.
- Sigmoid functions for gating

LSTM Cell



- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

LSTM Cell

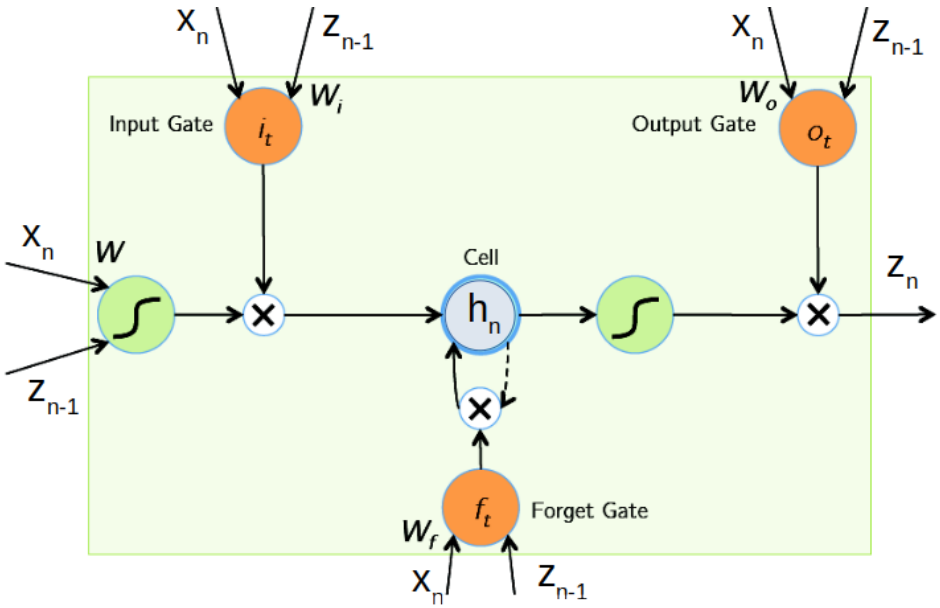


- Gating functions

$$\mathbf{f}_n = \sigma(\mathbf{W}_x^f \mathbf{x}_n + \mathbf{W}_z^f \mathbf{z}_{n-1})$$

- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

LSTM Cell



- Gating functions

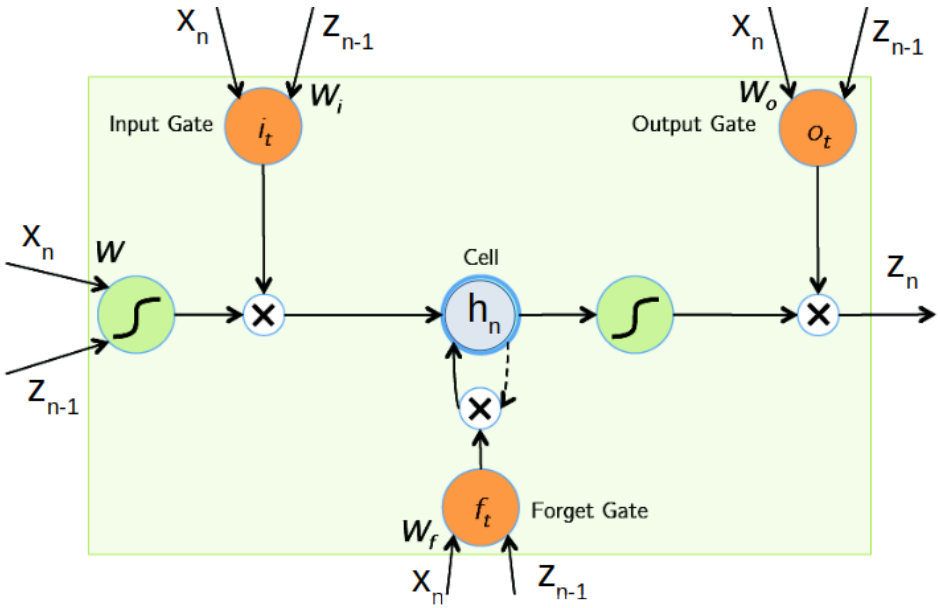
$$\mathbf{f}_n = \sigma(\mathbf{W}_x^f \mathbf{x}_n + \mathbf{W}_z^f \mathbf{z}_{n-1})$$

$$\mathbf{i}_n = \sigma(\mathbf{W}_x^i \mathbf{x}_n + \mathbf{W}_z^i \mathbf{z}_{n-1})$$

$$\mathbf{o}_n = \sigma(\mathbf{W}_x^o \mathbf{x}_n + \mathbf{W}_z^o \mathbf{z}_{n-1})$$

- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

LSTM Cell



- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

- Gating functions

$$\mathbf{f}_n = \sigma(\mathbf{W}_x^f \mathbf{x}_n + \mathbf{W}_z^f \mathbf{z}_{n-1})$$

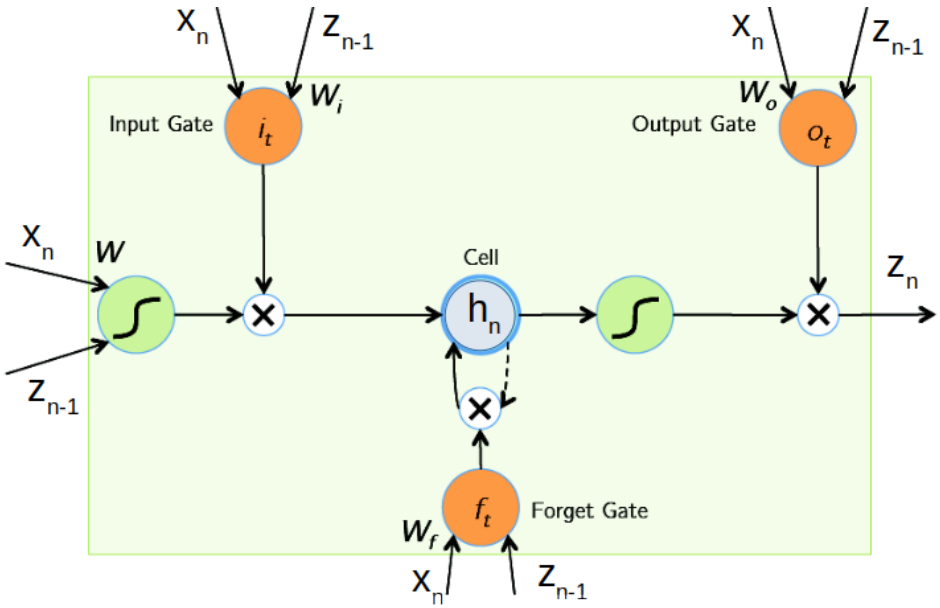
$$\mathbf{i}_n = \sigma(\mathbf{W}_x^i \mathbf{x}_n + \mathbf{W}_z^i \mathbf{z}_{n-1})$$

$$\mathbf{o}_n = \sigma(\mathbf{W}_x^o \mathbf{x}_n + \mathbf{W}_z^o \mathbf{z}_{n-1})$$

- Cell input for state update

$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x \mathbf{x}_n + \mathbf{W}_z \mathbf{z}_{n-1})$$

LSTM Cell



- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

- Gating functions

$$\mathbf{f}_n = \sigma(\mathbf{W}_x^f \mathbf{x}_n + \mathbf{W}_z^f \mathbf{z}_{n-1})$$

$$\mathbf{i}_n = \sigma(\mathbf{W}_x^i \mathbf{x}_n + \mathbf{W}_z^i \mathbf{z}_{n-1})$$

$$\mathbf{o}_n = \sigma(\mathbf{W}_x^o \mathbf{x}_n + \mathbf{W}_z^o \mathbf{z}_{n-1})$$

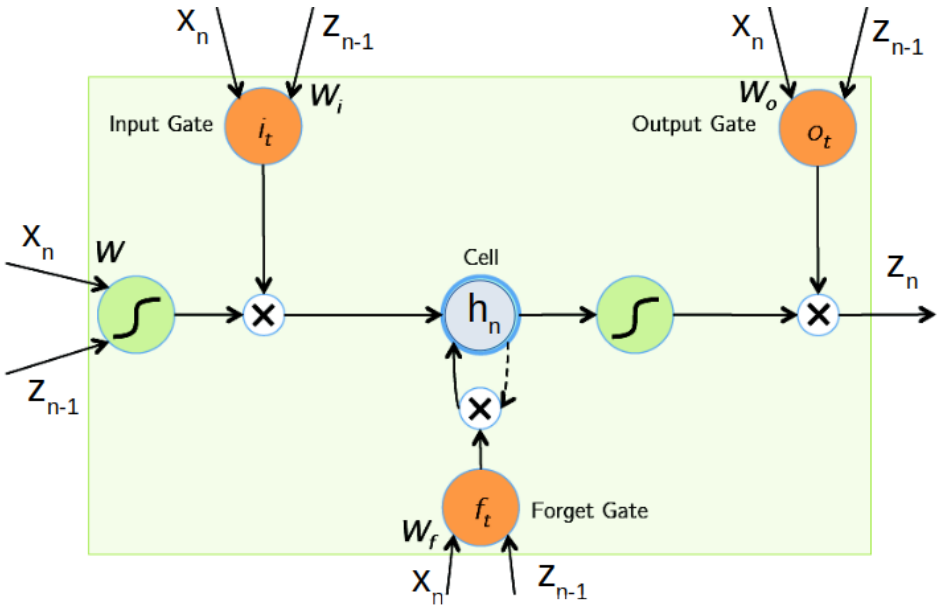
- Cell input for state update

$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x \mathbf{x}_n + \mathbf{W}_z \mathbf{z}_{n-1})$$

- State Update

$$\mathbf{h}_n = \mathbf{h}_{n-1} \odot \mathbf{f}_n + \tilde{\mathbf{h}}_n \odot \mathbf{i}_n$$

LSTM Cell



- Gates control flow of info.
- Sigmoid functions for gating
- Forget, input and output gates
- 3-fold increase in parameters

- Gating functions

$$\mathbf{f}_n = \sigma(\mathbf{W}_x^f \mathbf{x}_n + \mathbf{W}_z^f \mathbf{z}_{n-1})$$

$$\mathbf{i}_n = \sigma(\mathbf{W}_x^i \mathbf{x}_n + \mathbf{W}_z^i \mathbf{z}_{n-1})$$

$$\mathbf{o}_n = \sigma(\mathbf{W}_x^o \mathbf{x}_n + \mathbf{W}_z^o \mathbf{z}_{n-1})$$

- Cell input for state update

$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x \mathbf{x}_n + \mathbf{W}_z \mathbf{z}_{n-1})$$

- State Update

$$\mathbf{h}_n = \mathbf{h}_{n-1} \odot \mathbf{f}_n + \tilde{\mathbf{h}}_n \odot \mathbf{i}_n$$

- Cell output

$$\mathbf{z}_n = \tanh(\mathbf{h}[n]) \odot \mathbf{o}_n$$

Variants of LSTMs

- Several LSTM variants were proposed in the literature
  - No input gate, no forget gate, no output gate, no input activation, no output activation, introducing peepholes, coupling input and forget gates, full gate recurrence

Variants of LSTMs

- Several LSTM variants were proposed in the literature
  - No input gate, no forget gate, no output gate, no input activation, no output activation, introducing peepholes, coupling input and forget gates, full gate recurrence
- All these variants were evaluated on some standard tasks
  - TIMIT speech recognition (phoneme recognition)
  - IAM online handwriting recognition
  - JSB Chorales: Next-frame music prediction

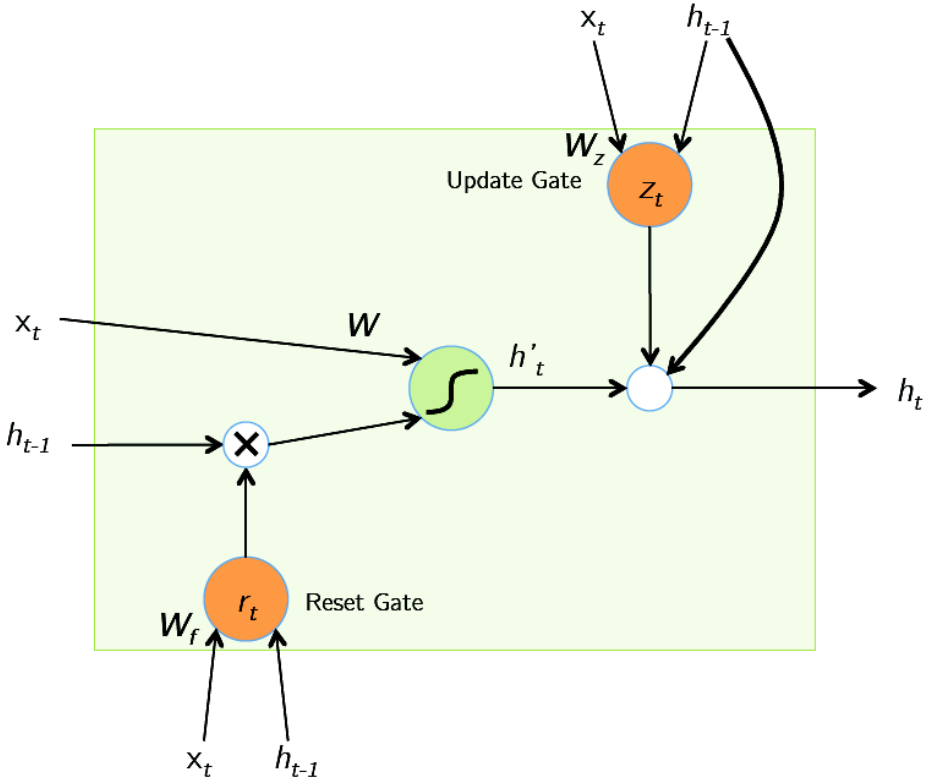
Variants of LSTMs

- Several LSTM variants were proposed in the literature
  - No input gate, no forget gate, no output gate, no input activation, no output activation, introducing peepholes, coupling input and forget gates, full gate recurrence
- All these variants were evaluated on some standard tasks
  - TIMIT speech recognition (phoneme recognition)
  - IAM online handwriting recognition
  - JSB Chorales: Next-frame music prediction
- Standard LSTM performed reasonably well on all the datasets, and none of the modifications significantly improved the performance
- Forget gate and output activation were found to be crucial

Variants of LSTMs

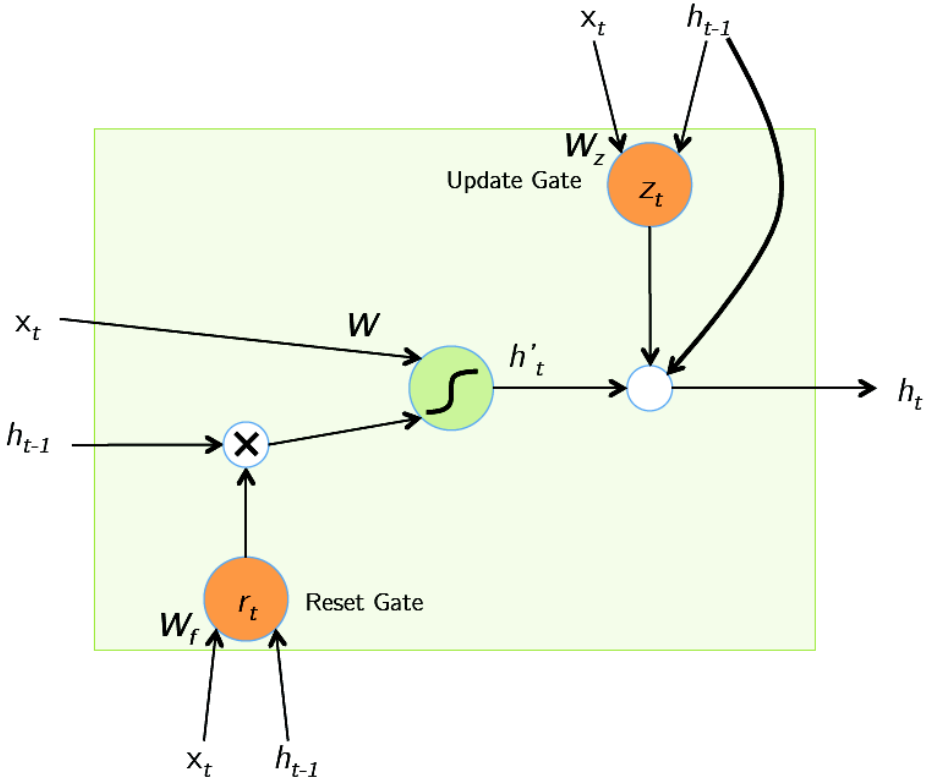
- Several LSTM variants were proposed in the literature
  - No input gate, no forget gate, no output gate, no input activation, no output activation, introducing peepholes, coupling input and forget gates, full gate recurrence
- All these variants were evaluated on some standard tasks
  - TIMIT speech recognition (phoneme recognition)
  - IAM online handwriting recognition
  - JSB Chorales: Next-frame music prediction
- Standard LSTM performed reasonably well on all the datasets, and none of the modifications significantly improved the performance
- Forget gate and output activation were found to be crucial
- LSTM: A Search Space Odyssey, Greff. *et al.*, 2015.

Gated Recurrent Unit (GRU)



- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

Gated Recurrent Unit (GRU)



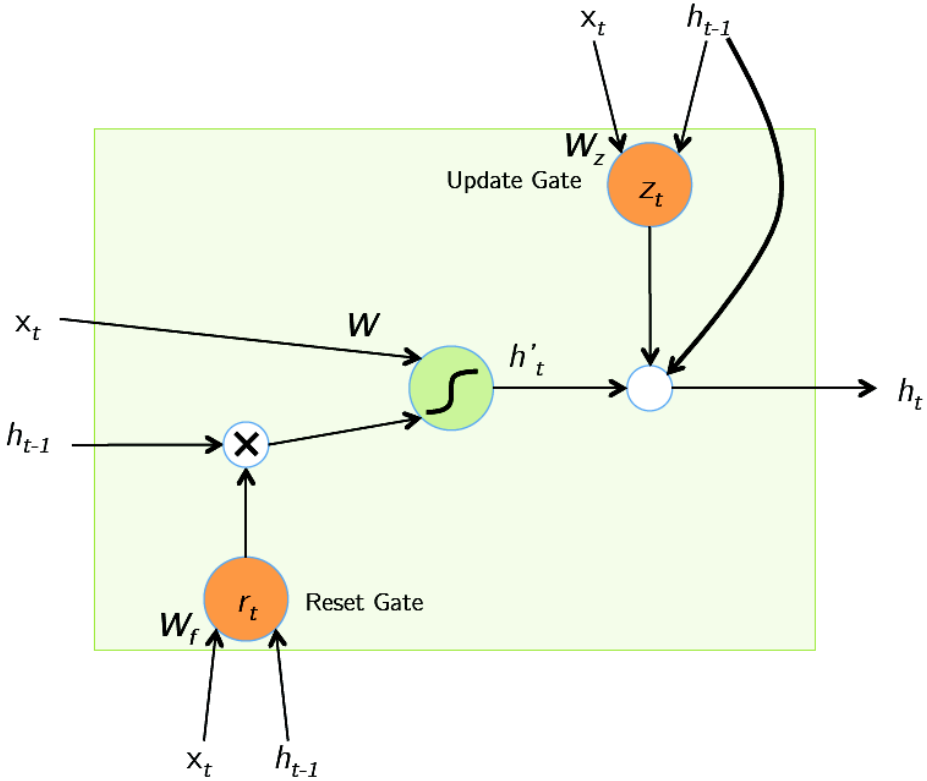
- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

- Gating functions

$$\mathbf{r}_n = \sigma(\mathbf{W}_x^r \mathbf{x}_n + \mathbf{W}_z^r \mathbf{z}_{n-1})$$

$$\mathbf{u}_n = \sigma(\mathbf{W}_x^u \mathbf{x}_n + \mathbf{W}_z^u \mathbf{z}_{n-1})$$

Gated Recurrent Unit (GRU)



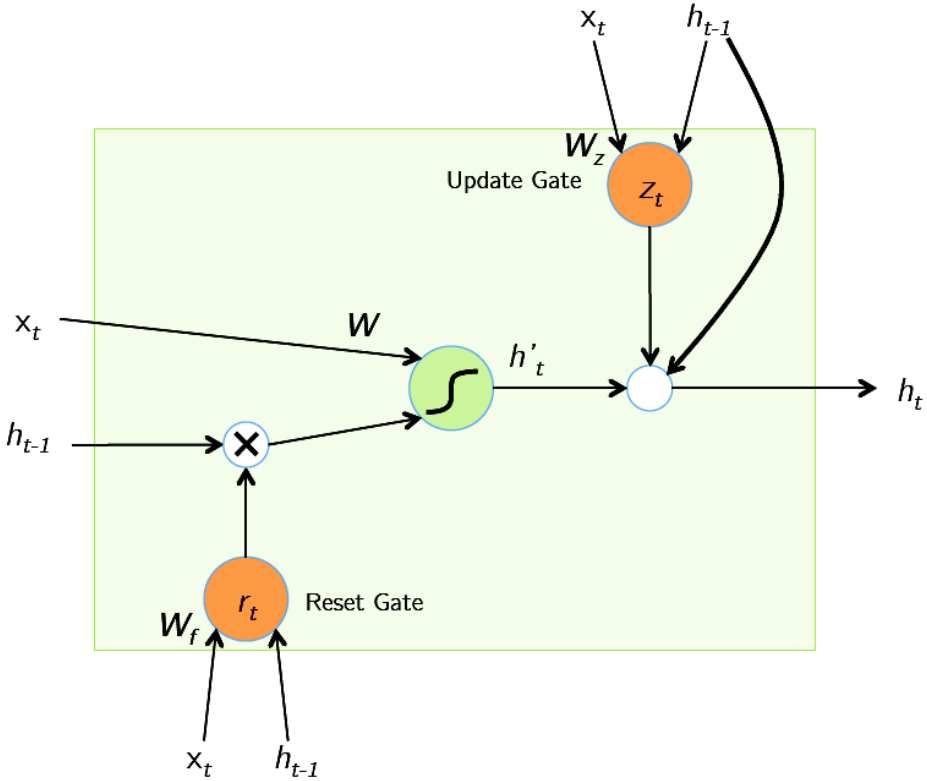
- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

- Gating functions

$$\mathbf{r}_n = \sigma(\mathbf{W}_x^r \mathbf{x}_n + \mathbf{W}_z^r \mathbf{z}_{n-1})$$

$$\mathbf{u}_n = \sigma(\mathbf{W}_x^u \mathbf{x}_n + \mathbf{W}_z^u \mathbf{z}_{n-1})$$

Gated Recurrent Unit (GRU)



- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

- Gating functions

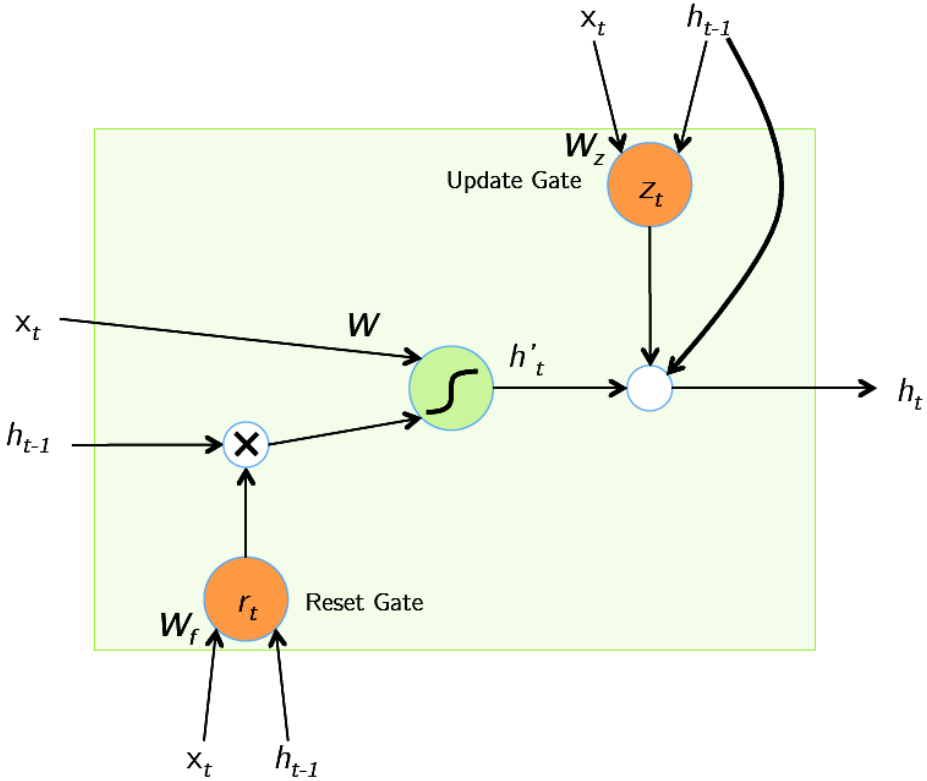
$$\mathrm{r}_n = \sigma(\mathbf{W}_x^r\, \mathbf{x}_n + \mathbf{W}_z^r\, \mathbf{z}_{n-1})$$

$$\mathbf{u}_n = \sigma(\mathbf{W}_x^u\, \mathbf{x}_n + \mathbf{W}_z^u\, \mathbf{z}_{n-1})$$

- Input for state update

$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x\, \mathbf{x}_n + \mathbf{W}_z\, \mathbf{z}_{n-1} \odot \mathrm{r}_n)$$

Gated Recurrent Unit (GRU)



- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

- Gating functions

$$\mathrm{r}_n = \sigma(\mathbf{W}_x^r\, \mathbf{x}_n + \mathbf{W}_z^r\, \mathbf{z}_{n-1})$$

$$\mathbf{u}_n = \sigma(\mathbf{W}_x^u\, \mathbf{x}_n + \mathbf{W}_z^u\, \mathbf{z}_{n-1})$$

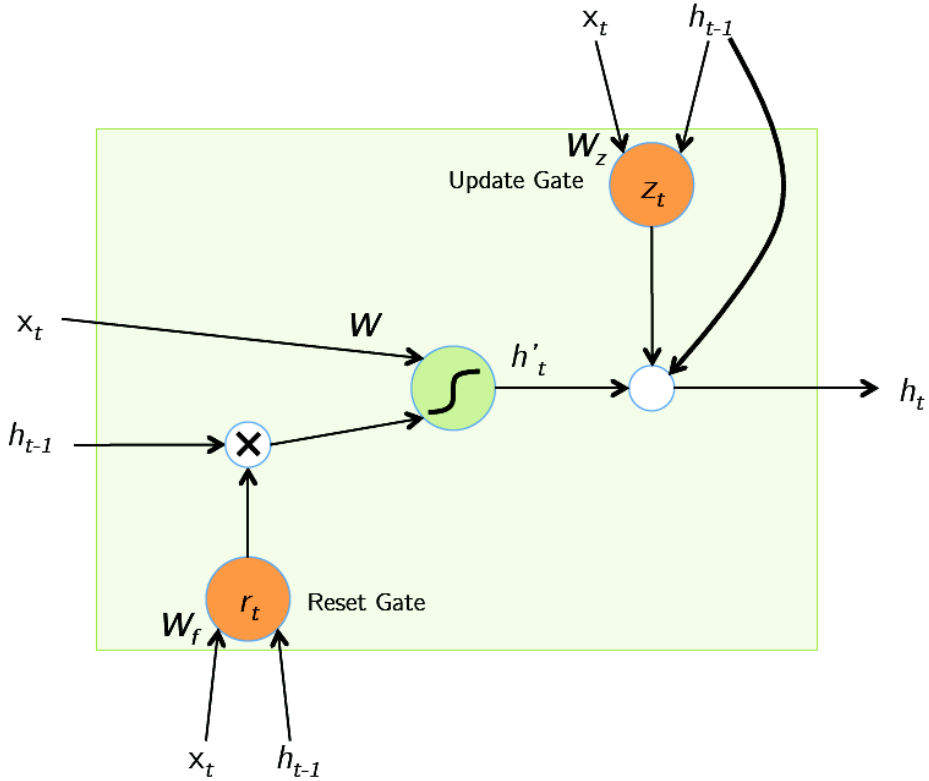
- Input for state update

$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x\, \mathbf{x}_n + \mathbf{W}_z\, \mathbf{z}_{n-1} \odot \mathrm{r}_n)$$

- State Update

$$\mathbf{h}_n = \mathbf{h}_{n-1} \odot (1 - \mathbf{u}_n) + \tilde{\mathbf{h}}_n \odot \mathbf{u}_n$$

Gated Recurrent Unit (GRU)



- Simplified version of LSTM
- Merges input and forget gates
- Merges cell state and output

- Gating functions

$$\mathrm{r}_n = \sigma(\mathbf{W}_x^r\, \mathbf{x}_n + \mathbf{W}_z^r\, \mathbf{z}_{n-1})$$

$$\mathbf{u}_n = \sigma(\mathbf{W}_x^u\, \mathbf{x}_n + \mathbf{W}_z^u\, \mathbf{z}_{n-1})$$

- Input for state update

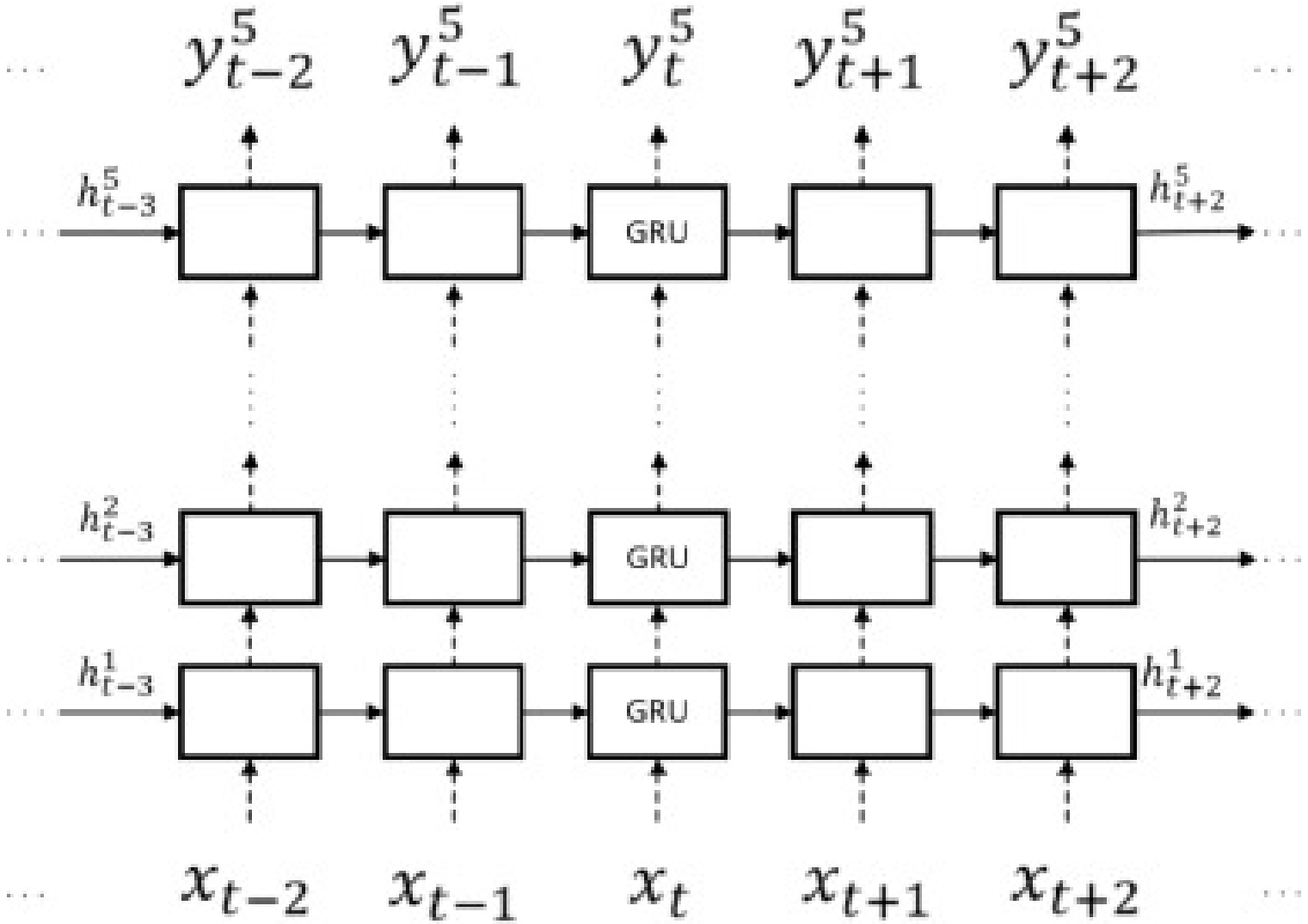
$$\tilde{\mathbf{h}}_n = \tanh(\mathbf{W}_x\, \mathbf{x}_n + \mathbf{W}_z\, \mathbf{z}_{n-1} \odot \mathrm{r}_n)$$

- State Update

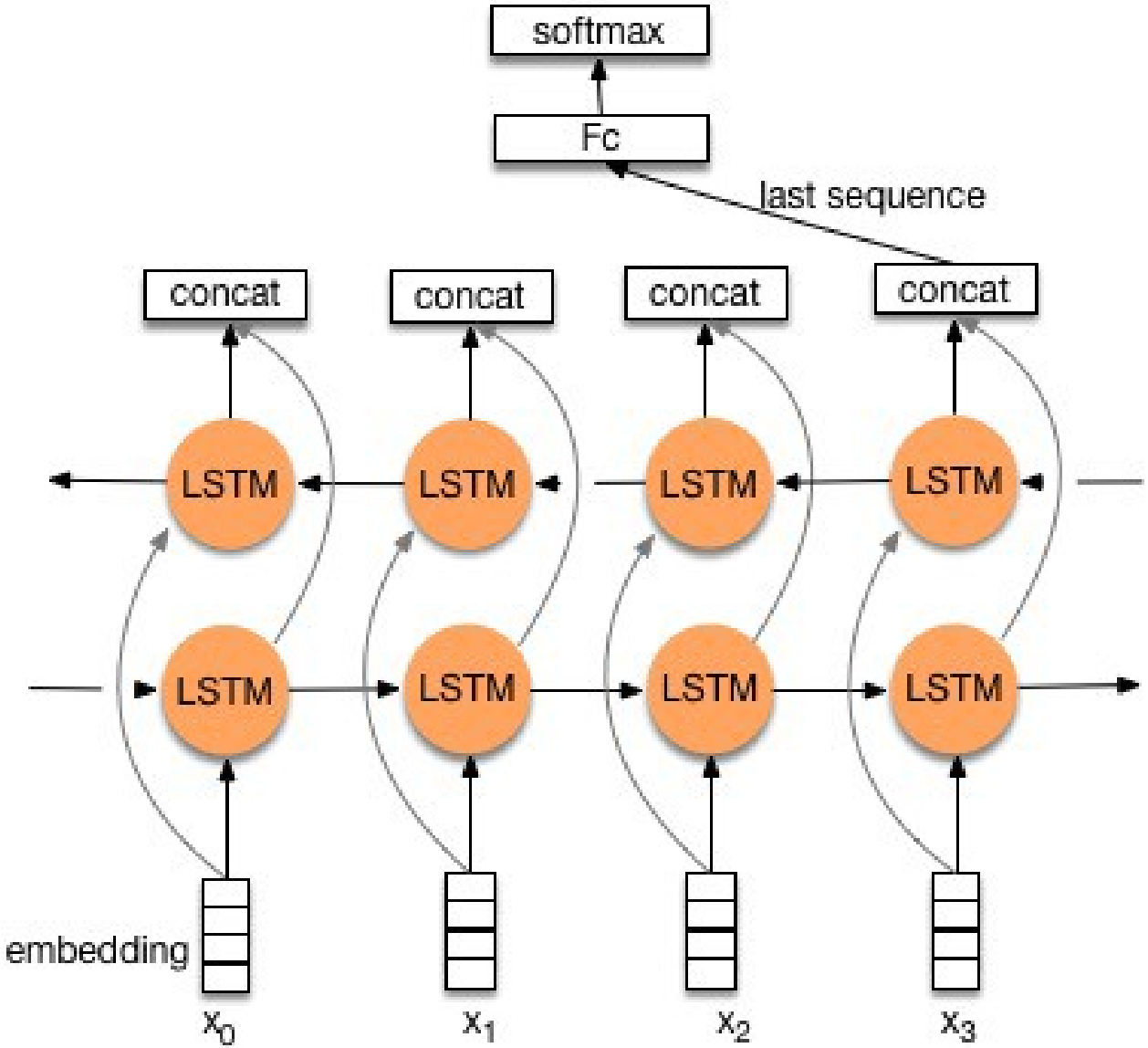
$$\mathbf{h}_n = \mathbf{h}_{n-1} \odot (1 - \mathbf{u}_n) + \tilde{\mathbf{h}}_n \odot \mathbf{u}_n$$

- GRUs were shown to outperform LSTMs in machine translation

Stacked RNN Layers



Bidirectional RNNs



Summary of Recurrent Models

- State vector is used to compactly represent the past history
- State vector is updated as $\mathbf{h}_n = g(\mathbf{h}_{n-1}, \mathbf{x}_n)$

Summary of Recurrent Models

- State vector is used to compactly represent the past history
- State vector is updated as $\mathbf{h}_n = g(\mathbf{h}_{n-1}, \mathbf{x}_n)$
- The parameters are estimated using temporal backpropagation
  - In theory, gradients can be backpropagated from any time-step T to 0
  - Implementations truncates the model to fixed K time steps.

Summary of Recurrent Models

- State vector is used to compactly represent the past history
- State vector is updated as $\mathbf{h}_n = g(\mathbf{h}_{n-1}, \mathbf{x}_n)$
- The parameters are estimated using temporal backpropagation
  - In theory, gradients can be backpropagated from any time-step T to 0
  - Implementations truncates the model to fixed K time steps.
- The *infinite memory* advantage of RNNs is largely absent in practice
  - Recurrent models trained in practice are effectively feed-forward.
  - The unlimited context offered by recurrence is strictly not necessary

Summary of Recurrent Models

- State vector is used to compactly represent the past history
- State vector is updated as $\mathbf{h}_n = g(\mathbf{h}_{n-1}, \mathbf{x}_n)$
- The parameters are estimated using temporal backpropagation
  - In theory, gradients can be backpropagated from any time-step T to 0
  - Implementations truncates the model to fixed K time steps.
- The *infinite memory* advantage of RNNs is largely absent in practice
  - Recurrent models trained in practice are effectively feed-forward.
  - The unlimited context offered by recurrence is strictly not necessary
- Feed-forward alternates for sequence-to-sequence modeling
  - Autoregressive feed-forward models and Transformers
  - Advantages: Parallelization, Training stability, faster inference
  - FFNNs and stable RNNs offer similar performance

Autoregressive Feedforward Models

- Feedforward models to predict next sample from the past k samples
- Autoregressive models are generative models and capture the joint pdf

$$p(x_0, x_1, x_2, \cdots, x_n) = \prod_{k=0}^{n} p(x_k / x_0, x_1, x_2, \cdots x_{k-1})$$

Autoregressive Feedforward Models

- Feedforward models to predict next sample from the past k samples
- Autoregressive models are generative models and capture the joint pdf

$$p(x_0, x_1, x_2, \cdots, x_n) = \prod_{k=0}^{n} p(x_k / x_0, x_1, x_2, \cdots x_{k-1})$$

- Estimate predictive distribution of next sample given the past samples
  - If x is discrete, softmax can be used to model the pmf.
  - If x is continuous, the parameters of the mixture model are estimated

Autoregressive Feedforward Models

- Feedforward models to predict next sample from the past k samples
- Autoregressive models are generative models and capture the joint pdf

$$p(x_0, x_1, x_2, \cdots, x_n) = \prod_{k=0}^{n} p(x_k / x_0, x_1, x_2, \cdots x_{k-1})$$

- Estimate predictive distribution of next sample given the past samples
  - If x is discrete, softmax can be used to model the pmf.
  - If x is continuous, the parameters of the mixture model are estimated
- AR models are amenable to conditioning - local ($\mathbf{v}_l$) and global ($\mathbf{v}_g$)

$$p(x_k / x_0, x_1, x_2, \cdots x_{k-1}, \mathbf{v}_g, \mathbf{v}_l)$$

- AR models can model multiple time scales (ms to minutes)
- Slow inference is their main drawback (Pixelnets and Wavenets)

Transformers

Transformers

- Feedforward architectures incorporating context information explicitly

Transformers

- Feedforward architectures incorporating context information explicitly
- The context vector of $\mathbf{x}_n$ is derived from its previous samples as

$$\mathbf{x}_n^c = \sum_{i=n-N}^{n} a_{ni} \left(\mathbf{W}_v \mathbf{x}_k \right)$$

Transformers

- Feedforward architectures incorporating context information explicitly
- The context vector of $\mathbf{x}_n$ is derived from its previous samples as

$$\mathbf{x}_n^c = \sum_{i=n-N}^{n} a_{ni} \left(\mathbf{W}_v \mathbf{x}_k \right)$$

- a_{ni} denotes the attention needed from i^{th} frame at the n^{th} instant

Transformers

- Feedforward architectures incorporating context information explicitly
- The context vector of $\mathbf{x}_n$ is derived from its previous samples as

$$\mathbf{x}_n^c = \sum_{i=n-N}^{n} a_{ni} \left(\mathbf{W}_v \mathbf{x}_k \right)$$

- a_{ni} denotes the attention needed from i^{th} frame at the n^{th} instant
- Attention is analogous to similarity, but need not be symmetric

$$\mathbf{q} = \mathbf{W}_q \mathbf{x}_n \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i, \qquad n - N \leq i \leq n$$

$$a_{ni} = \text{softmax}_N \left(\frac{\mathbf{q}^\mathsf{T} \mathbf{k}_i}{\sqrt{M}} \right)$$

Transformers

- Feedforward architectures incorporating context information explicitly
- The context vector of $\mathbf{x}_n$ is derived from its previous samples as

$$\mathbf{x}_n^c = \sum_{i=n-N}^{n} a_{ni} \left(\mathbf{W}_v \mathbf{x}_k \right)$$

- a_{ni} denotes the attention needed from i^{th} frame at the n^{th} instant
- Attention is analogous to similarity, but need not be symmetric
  - Compute the similarity between two different projections (query, key)

$$\mathbf{q} = \mathbf{W}_q \mathbf{x}_n \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i, \qquad n - N \leq i \leq n$$

$$a_{ni} = \text{softmax}_N \left(\frac{\mathbf{q}^\mathsf{T} \mathbf{k}_i}{\sqrt{M}} \right)$$

Transformers

- Feedforward architectures incorporating context information explicitly
- The context vector of $\mathbf{x}_n$ is derived from its previous samples as

$$\mathbf{x}_n^c = \sum_{i=n-N}^{n} a_{ni} \left(\mathbf{W}_v \mathbf{x}_k \right)$$

- a_{ni} denotes the attention needed from i^{th} frame at the n^{th} instant
- Attention is analogous to similarity, but need not be symmetric
  - Compute the similarity between two different projections (query, key)

$$\mathbf{q} = \mathbf{W}_q \mathbf{x}_n \qquad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i, \qquad n - N \leq i \leq n$$

$$a_{ni} = \text{softmax}_N \left(\frac{\mathbf{q}^\mathsf{T} \mathbf{k}_i}{\sqrt{M}} \right)$$

- The context-dependent vector x_n^c is added to context-independent vector x_n and processed further

Thank You!