The question of whether computers can think is like the question of whether submarines can swim.

- E. W. Dijsktra

CSE455 & CSE552
Machine Learning

Spring 2025

Neural Networks

1

Supervised Machine Learning

$$\arg\max_{f \in F} \sum_{i=1}^{n} g(y_i, f(x_i))$$

2

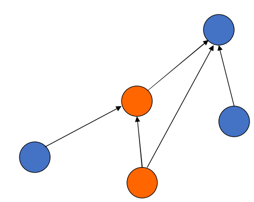
Supervised Machine Learning

$$\arg\max_{W} \sum_{i=1}^{n} g(y_i, f_{nn}(W, x_i))$$

3

Neural Networks

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10}
- Large connectitivity: 10^5
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures



4

Understanding the Brain

- Levels of analysis (Marr, 1982)
  1. Computational theory
  2. Representation and algorithm
  3. Hardware implementation
- Reverse engineering: From hardware to theory
- Parallel processing: SIMD vs MIMD

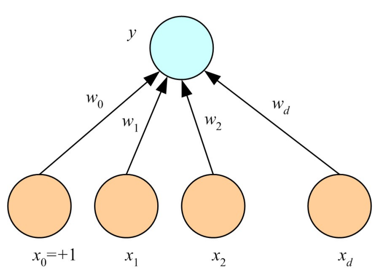
  Neural net: SIMD with modifiable local memory

  Learning: Update by training/experience

Supervised Machine Learning

$$\arg\max_{W} \sum_{i=1}^{n} g(\, y_i, f_{nn}(W, x_i))$$

Perceptron



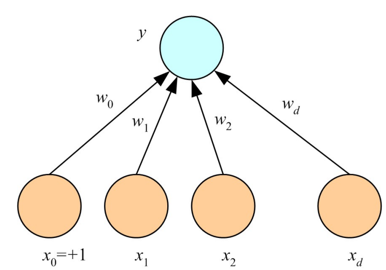
$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w}^T\mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$

(Rosenblatt, 1962)

Perceptron

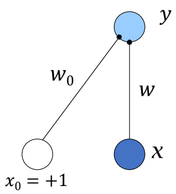


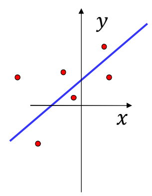
$$y^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)}$$

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{n} g(\, y^{(i)}, \mathbf{w}^T\mathbf{x}^{(i)})$$

What a Perceptron Does

Regression: $y = wx + w_0$

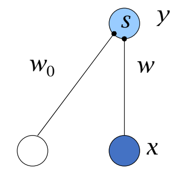


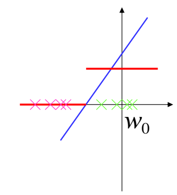


9

What a Perceptron Does

Classification: $y = 1(wx + w_0 > 0)$





$$y = \text{sigmoid}(o) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

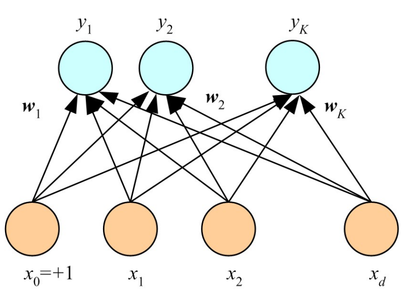
10

Regression with K Outputs

Regression:

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \boldsymbol{w}_i{}^T \boldsymbol{x}$$

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$$



11

Classification with K Outputs

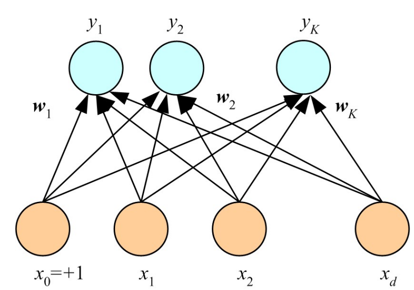
Classification:

$$o_i = \boldsymbol{w}_i{}^T \boldsymbol{x}$$

$$y_i = \frac{e^{o_i}}{\sum_k e^{o_k}}$$

Choose C_i

if $y_i = \max_k y_k$



12

Training

Training

- Online (instances seen one by one) vs batch (whole sample) learning:
  - No need to store the whole sample
  - Problem may change in time
  - Wear and degradation in system components
- Stochastic gradient-descent: Update after a single pattern
- Generic update rule (LMS rule):

$$\Delta w_{ij}{}^{t} = \eta(r_i{}^{t} - y_i{}^{t})x_j{}^{t}$$

  - Update = Learning Factor * (Desired Output – Actual Output) * Input

Training a Perceptron: Regression

- Regression (Linear output):

$$E^t(\boldsymbol{w} \mid \boldsymbol{x}^t, r^t) = \frac{1}{2}(r^t - y^t)^2 = \frac{1}{2}\left[r^t - (\boldsymbol{w}^T\boldsymbol{x}^t)\right]^2$$

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t$$

Classification

- Single sigmoid output

$$y^t = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x}^t)$$

$$E^t(\boldsymbol{w} \mid \boldsymbol{x}^t, \boldsymbol{r}^t) = -r^t \log y^t - (1 - r^t)\log(1 - y^t)$$

$$\Delta w_j^t = \eta(r^t - y^t)x_j^t$$

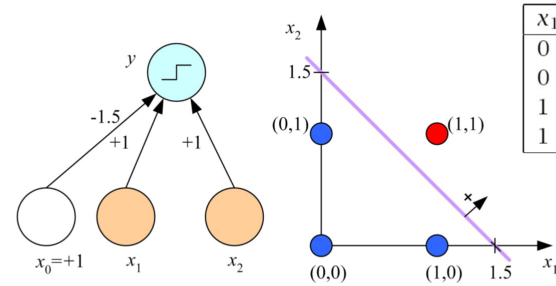
Classification

- *K*>2 softmax outputs

$$y^t = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}^t}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}^t} \quad E^t\left(\{\boldsymbol{w}_i\}_i \mid \boldsymbol{x}^t, \boldsymbol{r}^t\right) = -\sum_i r_i^t \log y_i^t$$

$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$

Learning Boolean AND



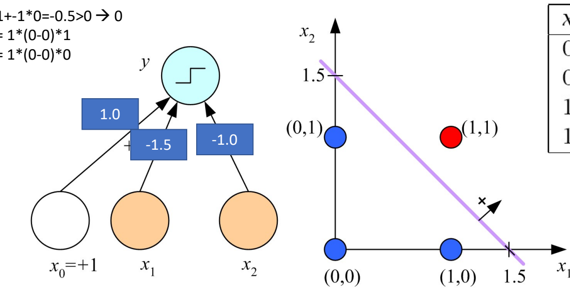
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

Learning Boolean AND

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

Learning Boolean AND

1*1-1.5*1+-1*0=-0.5>0 → 0
Deltaw1 = 1*(0-0)*1
Deltaw2 = 1*(0-0)*0



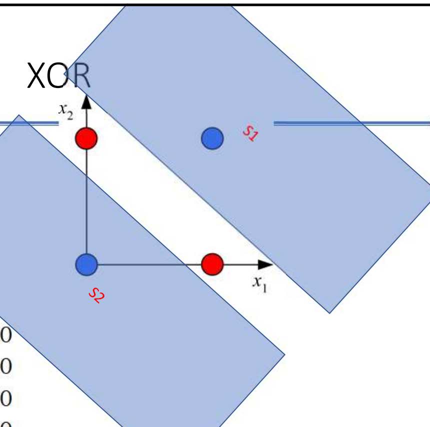
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

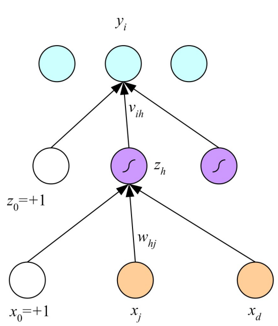


- No w_0, w_1, w_2 satisfy:

$$
\begin{aligned}
w_0 &\le 0 \\
w_2 + w_0 &> 0 \\
w_1 + w_0 &> 0 \\
w_1 + w_2 + w_0 &\le 0
\end{aligned}
$$

(Minsky and Papert, 1969)

Multilayer Perceptrons

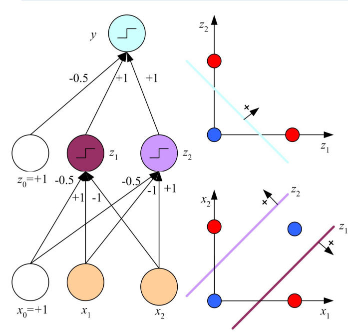


$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

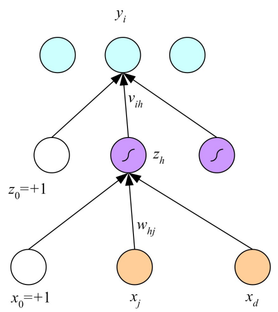
(Rumelhart et al., 1986)

XOR with Multilayer Perceptrons



x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

Backpropagation



$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$\boxed{\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}}$$

Regression

$$E(\mathbf{W}, \boldsymbol{v} \mid \mathcal{X}) = \frac{1}{2} \sum_t \left(r^t - y^t\right)^2$$

$$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0$$

$$\boxed{\Delta v_h = \sum_t \left(r^t - y^t\right) z_h^t}$$

Backward

Forward

$$z_h = \boxed{\text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right)}$$

$\boldsymbol{x}$

$$\begin{aligned}
\Delta w_{hj} &= -\eta \frac{\partial E}{\partial w_{hj}} \\
&= -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}} \\
&= -\eta \sum_t -\left(r^t - y^t\right) v_h \boxed{z_h^t\left(1 - z_h^t\right)} x_j^t \\
&= \eta \sum_t \left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t
\end{aligned}$$

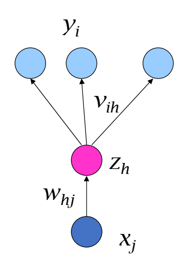
Regression with Multiple Outputs

$$E(\mathbf{W}, \mathbf{V} \mid \mathcal{X}) = \frac{1}{2} \sum_t \sum_i \left(r_i^t - y_i^t\right)^2$$

$$y_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0}$$

$$\Delta v_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

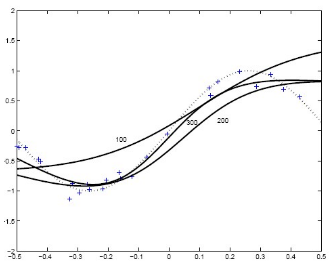
$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih}\right] z_h^t \left(1 - z_h^t\right) x_j^t$$

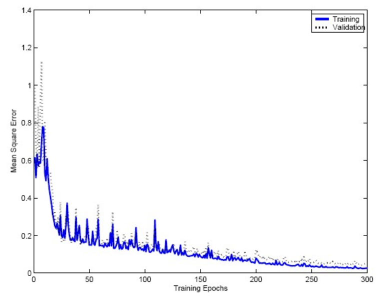


Algorithm

```
Initialize all v_ih and w_hj to rand(−0.01, 0.01)
Repeat
    For all (x^t, r^t) ∈ X in random order
        For h = 1, ..., H
            z_h ← sigmoid(w_h^T x^t)
        For i = 1, ..., K
            y_i = v_i^T z
        For i = 1, ..., K
            Δv_i = η(r_i^t − y_i^t) z
        For h = 1, ..., H
            Δw_h = η(Σ_i (r_i^t − y_i^t) v_ih) z_h (1 − z_h) x^t
        For i = 1, ..., K
            v_i ← v_i + Δv_i
        For h = 1, ..., H
            w_h ← w_h + Δw_h
Until convergence
```

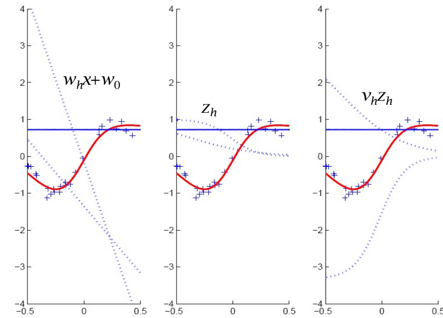
Results





29

Results



30

Two-Class Discrimination

One sigmoid output y^t for $P(C_1|\boldsymbol{x}^t)$ and $P(C_2|\boldsymbol{x}^t) \equiv 1\text{-}y^t$

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0\right)$$

$$E(\mathbf{W},\boldsymbol{v} \mid \mathcal{X}) = -\sum_t r^t \log y^t + \left(1 - r^t\right)\log\left(1 - y^t\right)$$

$$\Delta v_h = \eta \sum_t \left(r^t - y^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t$$

31

K>2 Classes

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \qquad y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} \equiv P\left(C_i \mid \boldsymbol{x}^t\right)$$

$$E(\mathbf{W},\boldsymbol{v} \mid \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t$$

$$\Delta v_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih}\right] z_h^t \left(1 - z_h^t\right) x_j^t$$

32

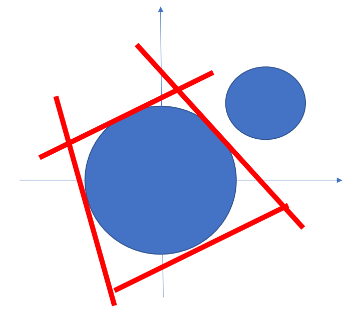
Multiple Hidden Layers

- MLP with one hidden layer is a universal approximator (Hornik et al., 1989), but using multiple layers may lead to simpler networks

$$z_{1h} = \text{sigmoid}\left(\boldsymbol{w}_{1h}^T \boldsymbol{x}\right) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), h = 1, \ldots, H_1$$

$$z_{2l} = \text{sigmoid}\left(\boldsymbol{w}_{2l}^T \boldsymbol{z}_1\right) = \text{sigmoid}\left(\sum_{h=1}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), l = 1, \ldots, H_2$$

$$y = \boldsymbol{v}^T \boldsymbol{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0$$



Improving Convergence

- Momentum

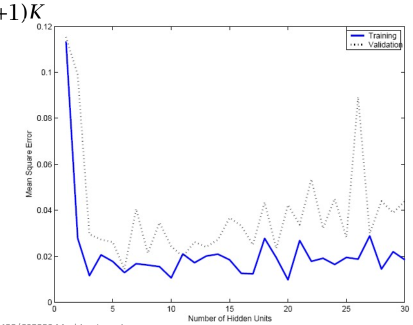
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

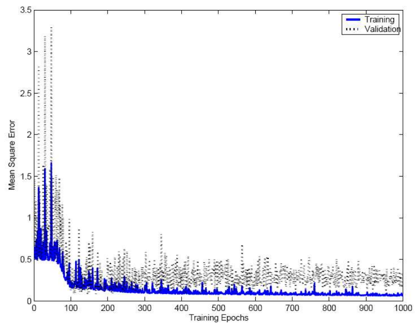
- Adaptive learning rate

$$\Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

Overfitting/Overtraining

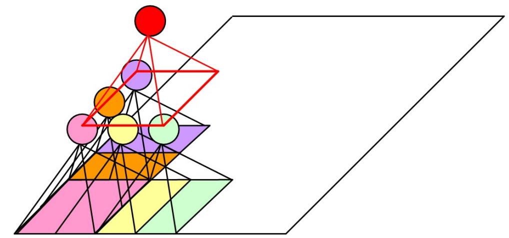
Number of weights: $H(d+1)+(H+1)K$





37

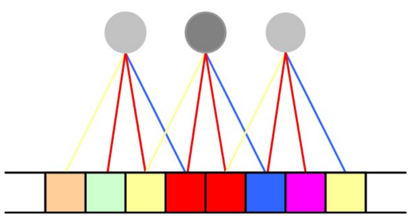
Structured MLP



(Le Cun et al, 1989)

38

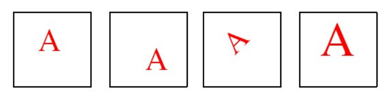
Weight Sharing



39

Hints

- Invariance to translation, rotation, size (Abu-Mostafa, 1995)



- Virtual examples
- Augmented error: $E'=E+\lambda_h E_h$

If ***x'*** and ***x*** are the "same": $E_h=[g(x|\theta)- g(x'|\theta)]^2$

Approximation hint:

$$E_h = \begin{cases} 0 & \text{if } g(x|\theta) \in [a_x, b_x] \\ (g(x|\theta)-a_x)^2 & \text{if } g(x|\theta) < a_x \\ (g(x|\theta)-b_x)^2 & \text{if } g(x|\theta) > b_x \end{cases}$$

40

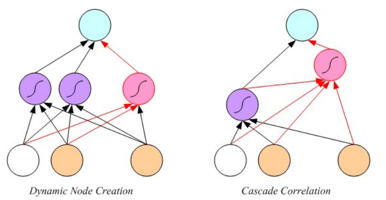
Tuning the Network Size

- Destructive
- Weight decay:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i$$

$$E' = E + \frac{\lambda}{2}\sum_i w_i^2$$

- Constructive
- Growing networks



(Ash, 1989) (Fahlman and Lebiere, 1989)

Bayesian Learning

- Consider weights w_i as random vars, prior $p(w_i)$

$$p(\mathbf{w} \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \mathbf{w})p(\mathbf{w})}{p(\mathcal{X})} \quad \hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log p(\mathbf{w} \mid \mathcal{X})$$

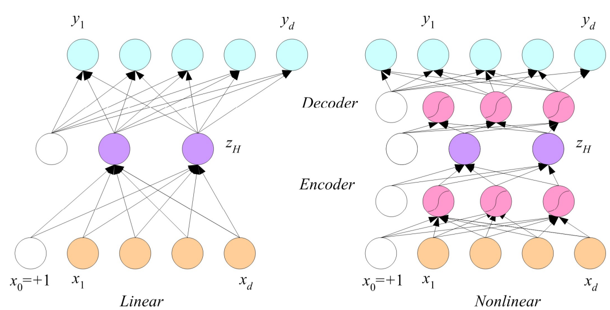
$$\log p(\mathbf{w} \mid \mathcal{X}) = \log p(\mathcal{X} \mid \mathbf{w}) + \log p(\mathbf{w}) + C$$

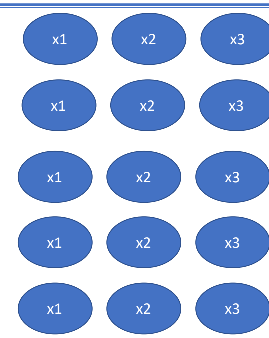
$$p(\mathbf{w}) = \prod_i p(w_i) \text{ where } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right]$$

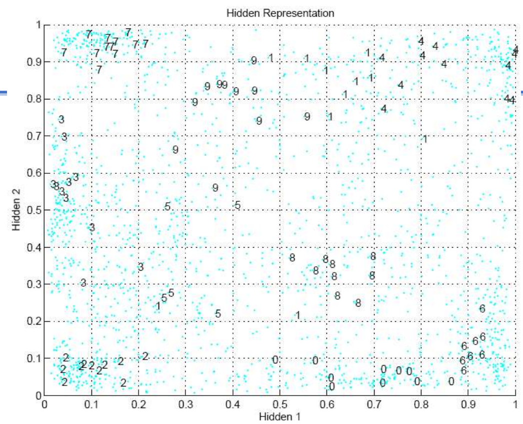
$$E' = E + \lambda \|\mathbf{w}\|^2$$

- Weight decay, ridge regression, regularization
  cost=data-misfit + λ complexity

Dimensionality Reduction



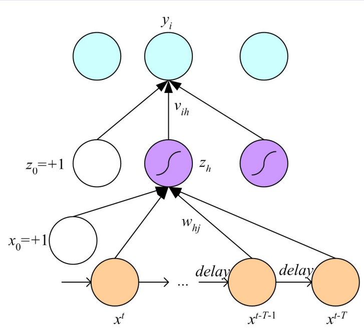




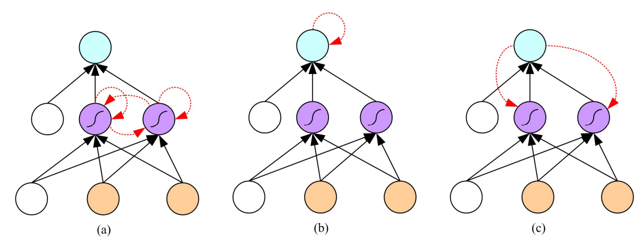
Learning Time

- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

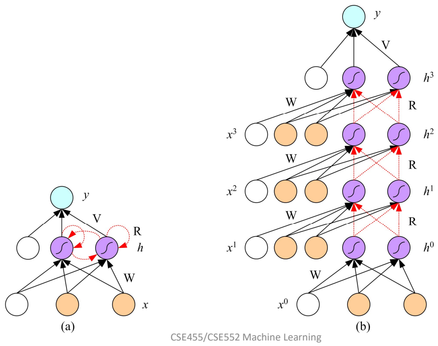
Time-Delay Neural Networks



Recurrent Networks



Unfolding in Time



Thanks for listening!