Lecture 5: Representation Learning

Kai-Wei Chang
CS @ UCLA
kw@kwchang.net

Couse webpage: https://uclanlp.github.io/CS269-17/

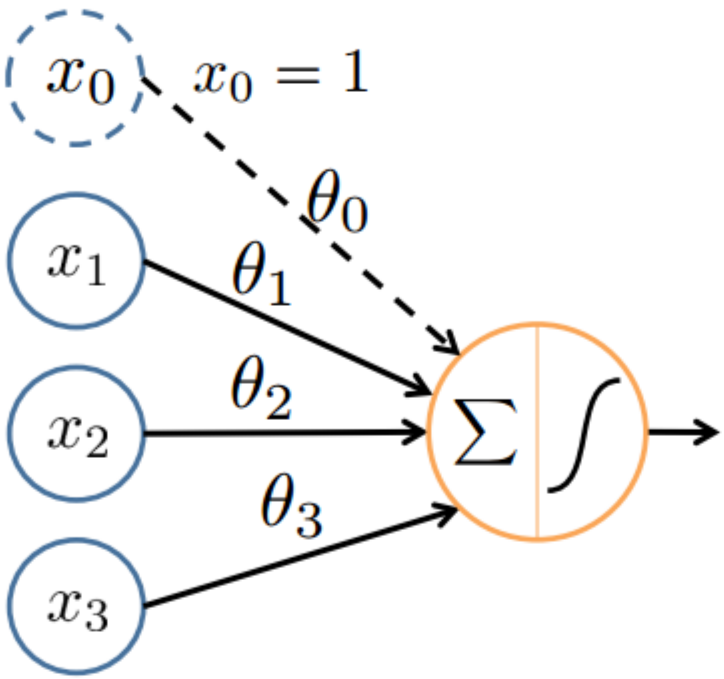
UCLA ENGINEERING
Computer Science

This lecture

- Review: Neural Network
- Recurrent NN
- Representation learning in NLP

Neural Network

"bias unit"

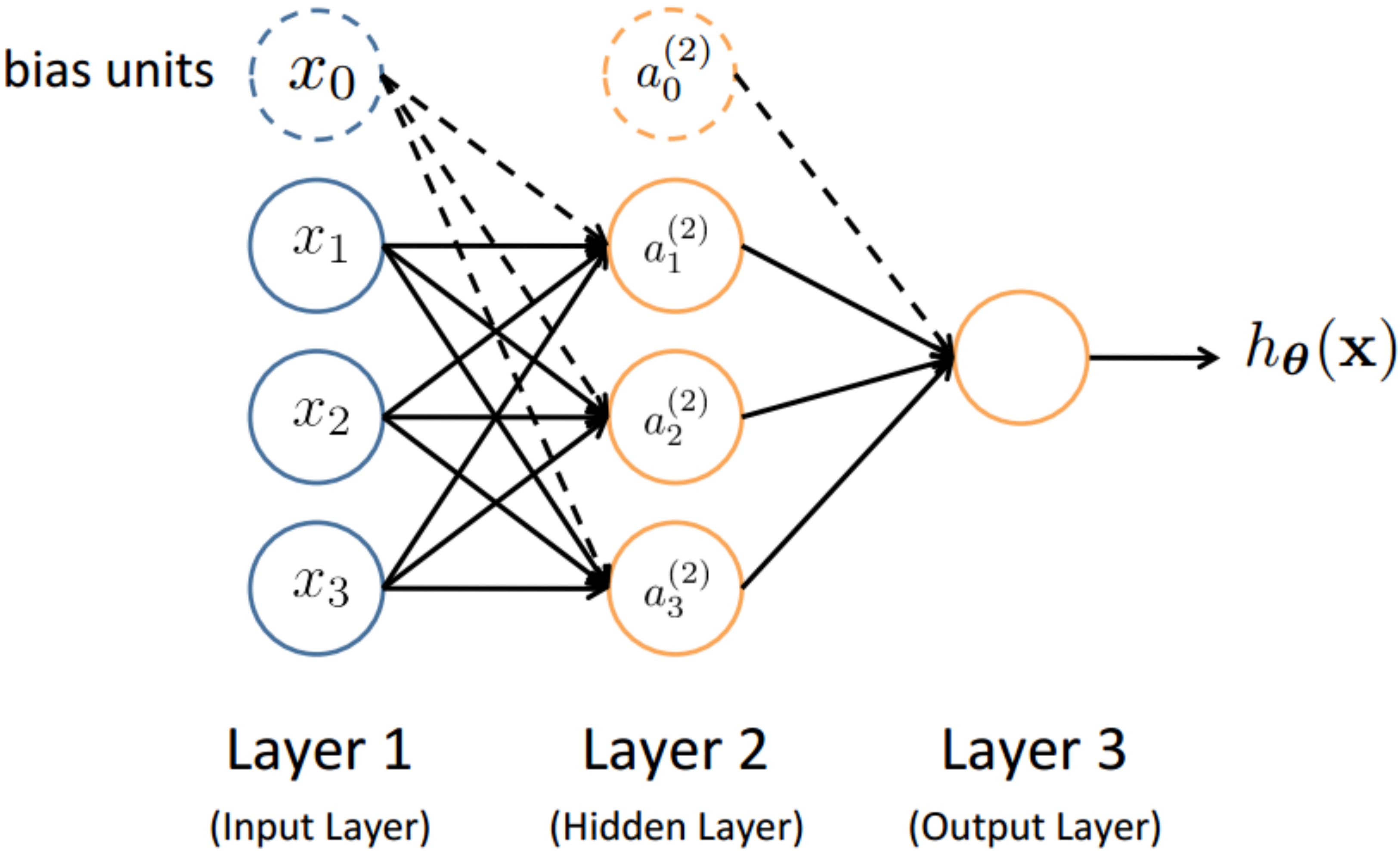


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\intercal}\mathbf{x}}}$$

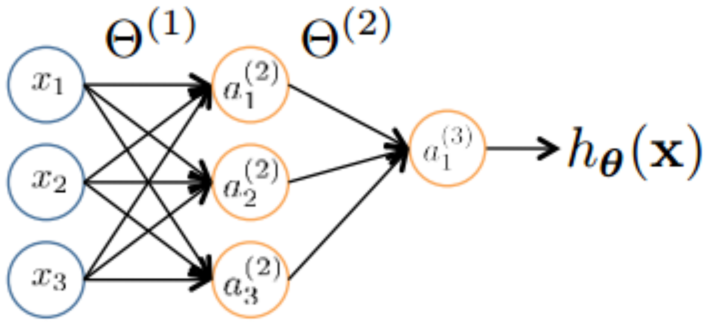
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Neural Network (feed forward)



Feed-Forward Process

- Input layer units are features (in NLP, e.g., words)
- Working forward through the network, the **input function** is applied to compute the input value
  - E.g., weighted sum of the input
- The **activation function** transforms this input function into a final value
  - Typically a **nonlinear** function (e.g, **sigmoid**)



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times4}$$

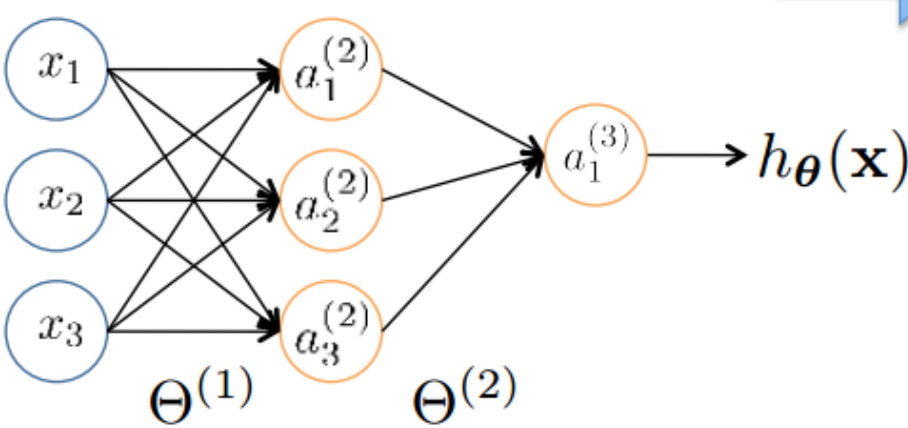
Vector Representation

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

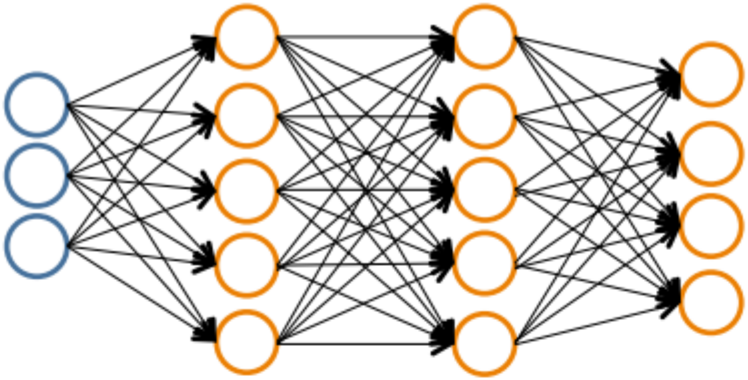
Can extend to multi-class


Pedestrian


Car


Motorcycle


Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$
when motorcycle

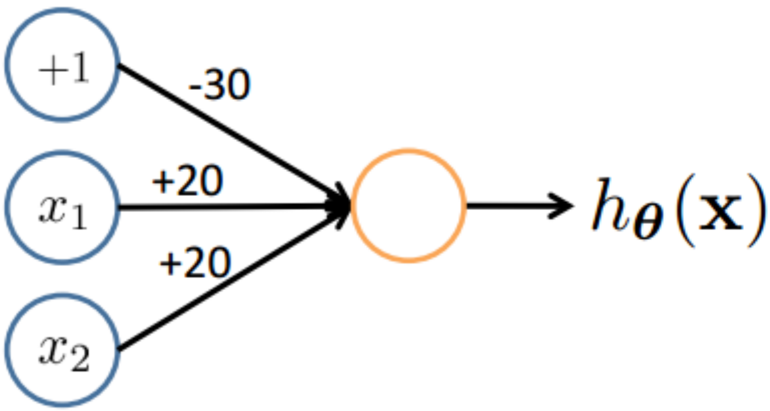
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
when truck

Why staged predictions?

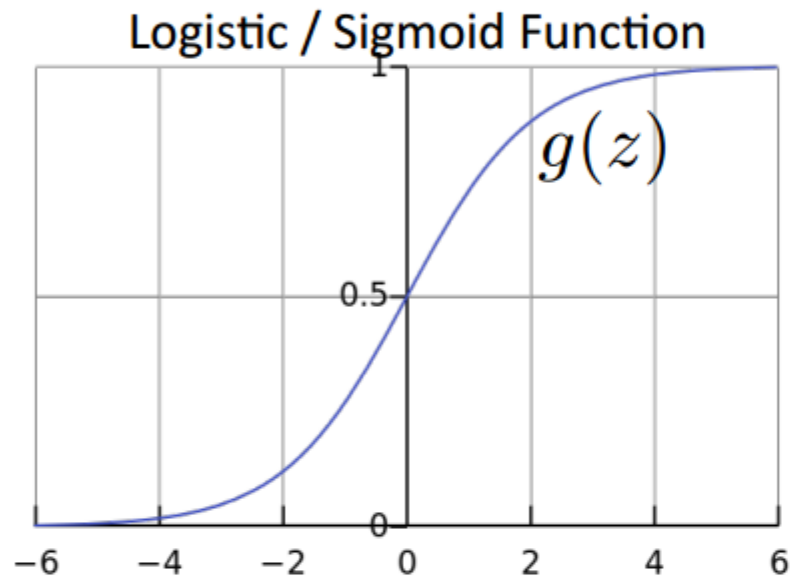
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

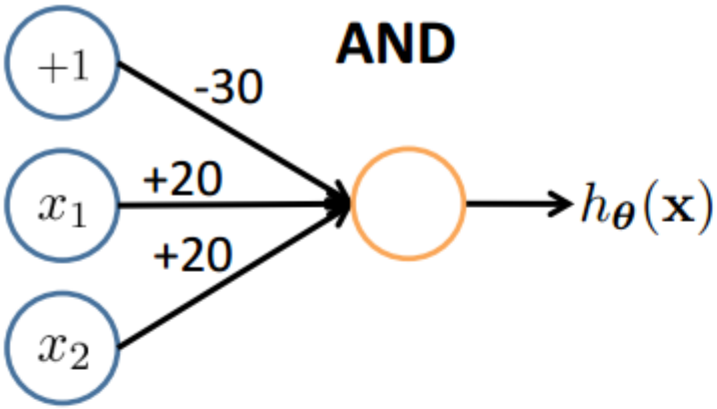
$y = x_1 \text{ AND } x_2$

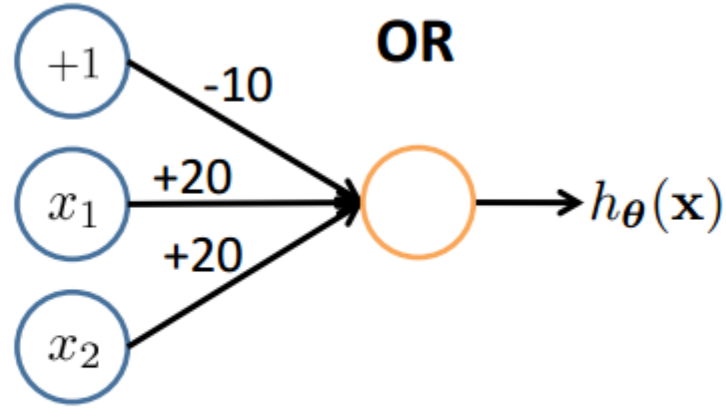


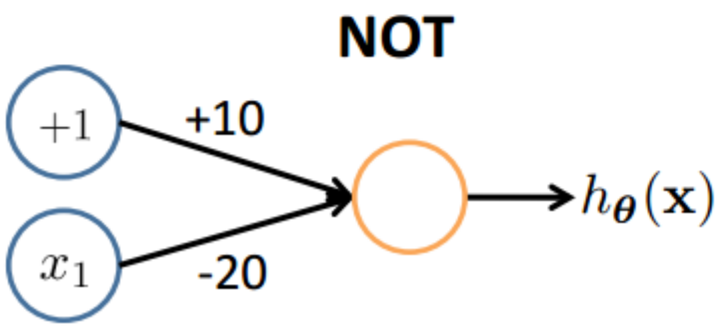
$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

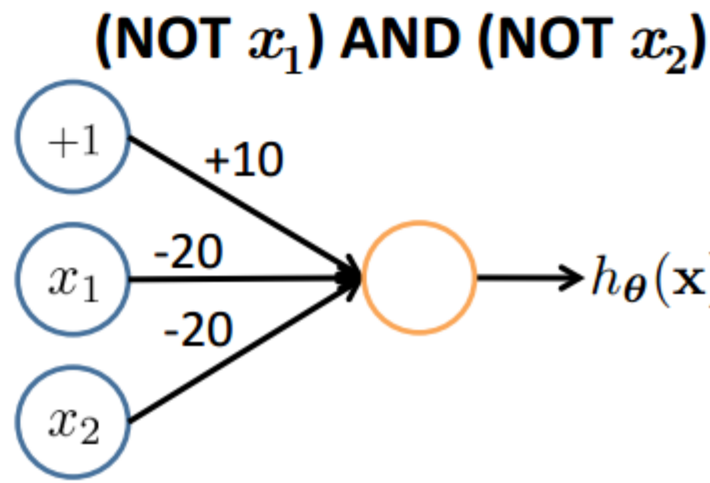


x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

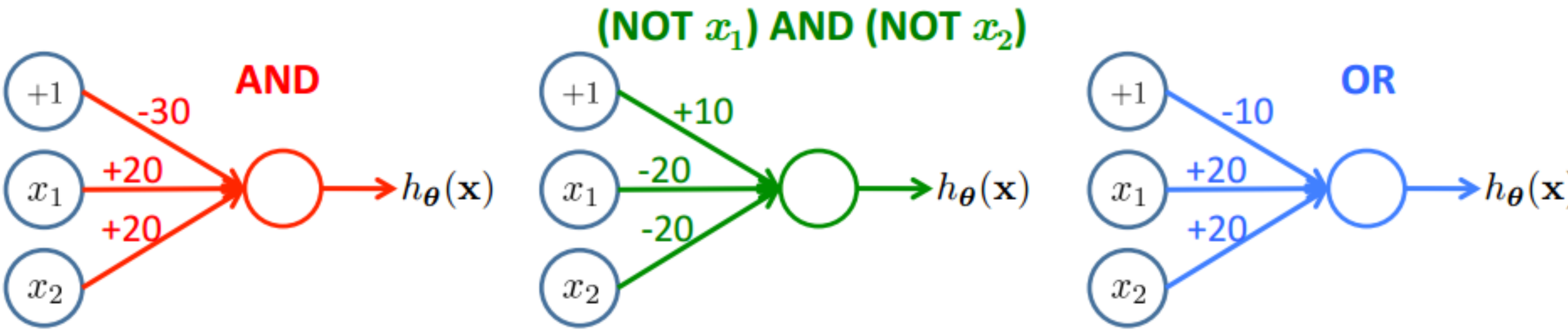

AND
+1
x_1
x_2
-30
+20
+20
$h_\theta(\mathbf{x})$

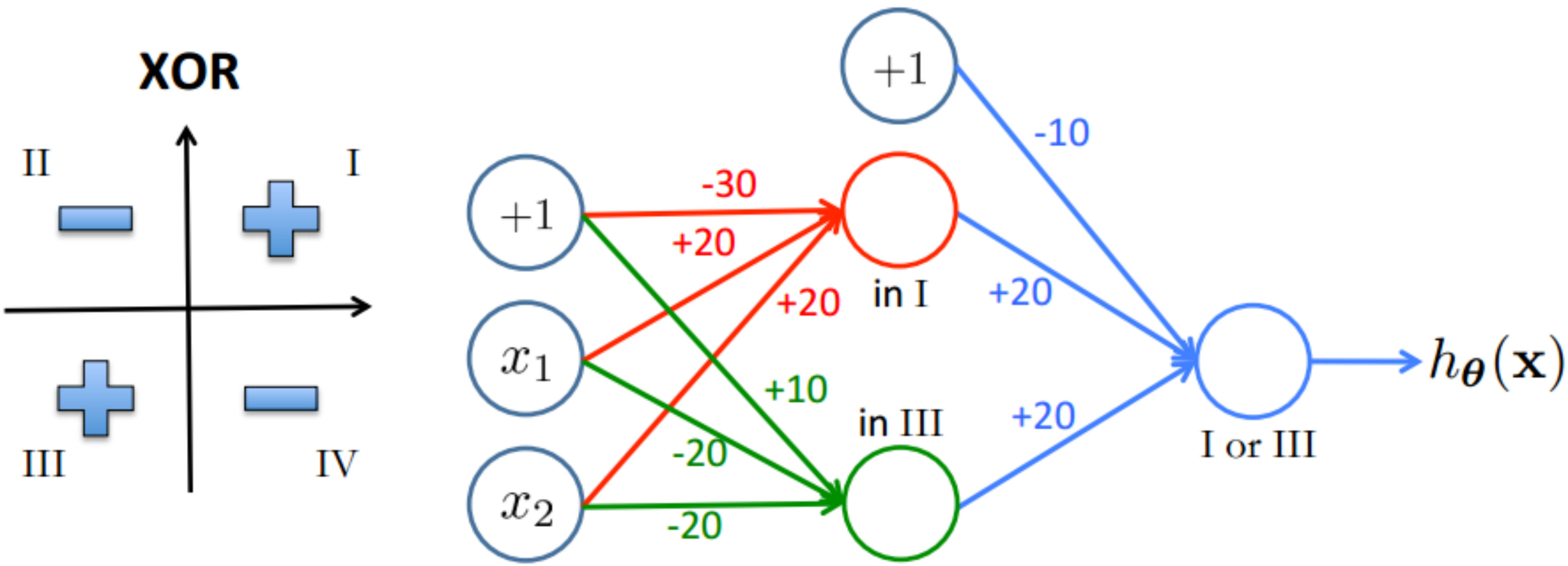

OR
+1
x_1
x_2
-10
+20
+20
$h_\theta(\mathbf{x})$


NOT
+1
x_1
+10
-20
$h_\theta(\mathbf{x})$

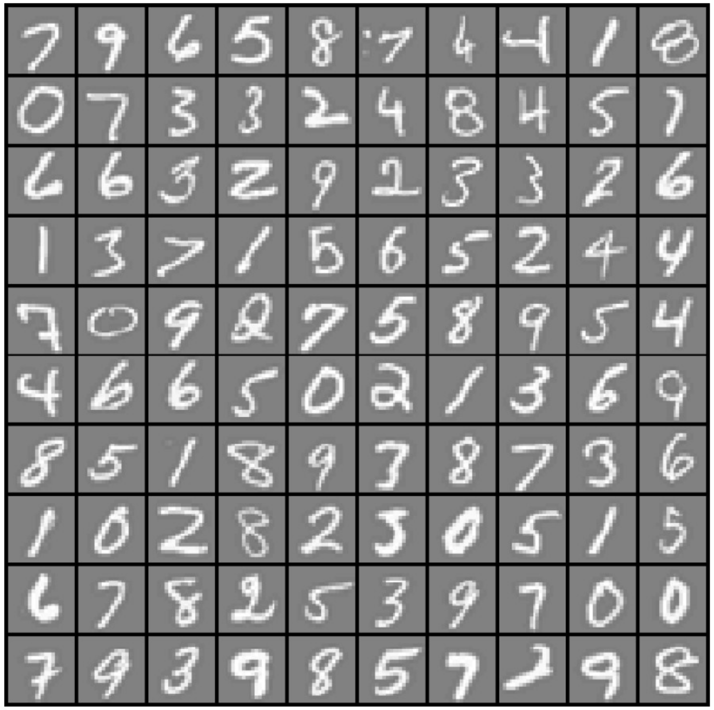

(NOT x_1) AND (NOT x_2)
+1
x_1
x_2
+10
-20
-20
$h_\theta(\mathbf{x})$

Combining Representations to Create Non-Linear Functions



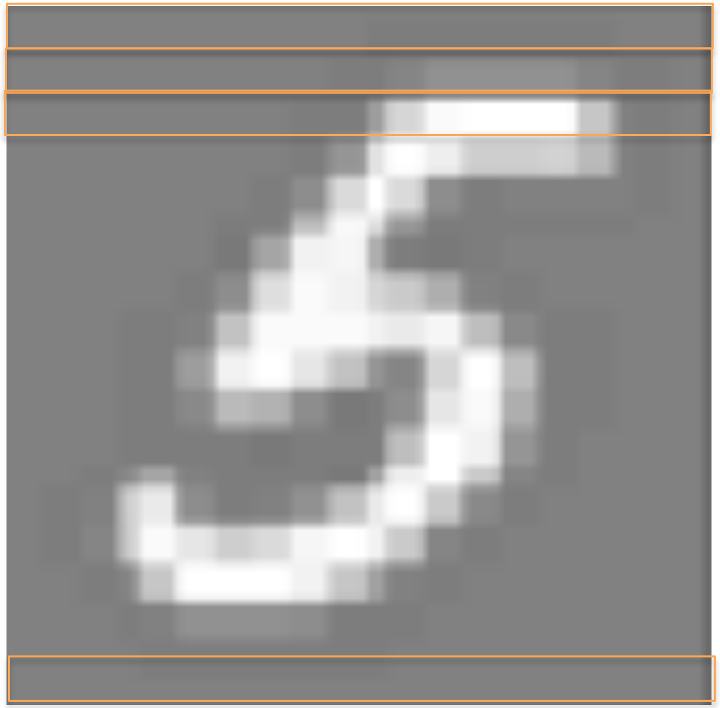


Layering Representations



20×20 pixel images

$d = 400$ 10 classes



$x_1 \ldots x_{20}$

$x_{21} \ldots x_{40}$

$x_{41} \ldots x_{60}$

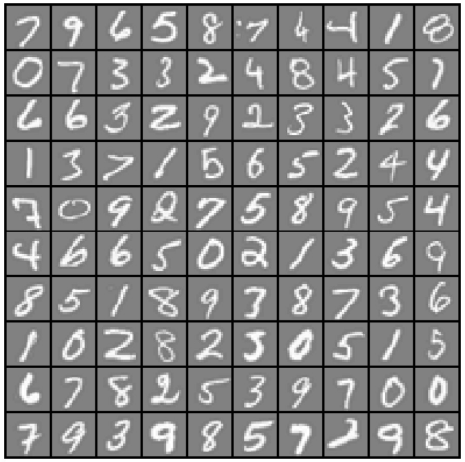
$\vdots$

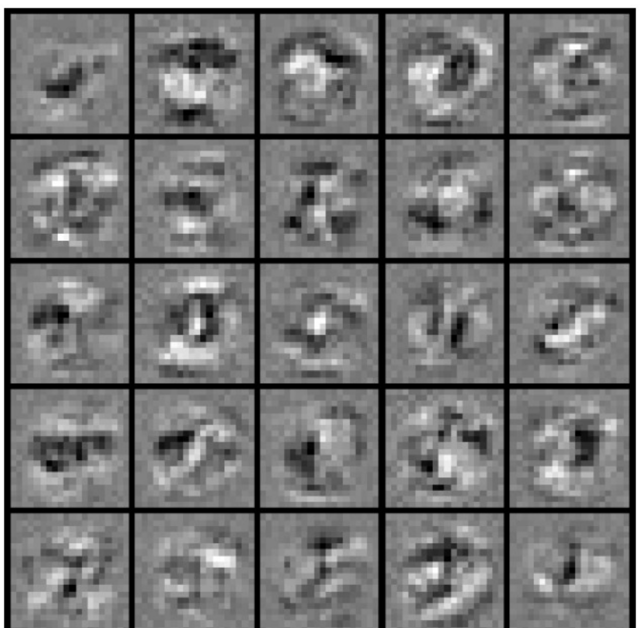
$x_{381} \ldots x_{400}$

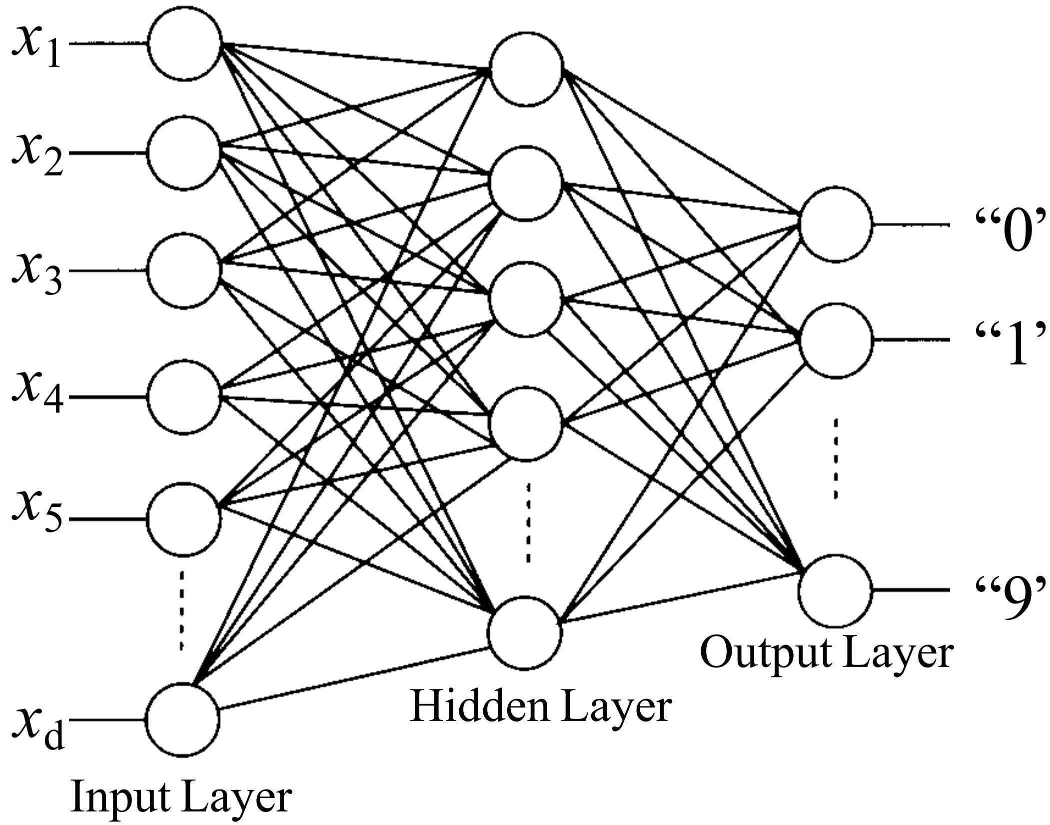
Each image is "unrolled" into a vector x of pixel intensities

UCLA ENGINEERING
Computer Science

Layering Representations







Visualization of Hidden Layer

This lecture

- Review: Neural Network
  - Learning NN
- Recursive and Recurrent NN
- Representation learning in NLP

Stochastic Sub-gradient Descent

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For epoch $1 \ldots T$:
3. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
4. Update $w \leftarrow w - \eta \nabla f(\theta)$
5. Return θ

Cost Function

$$f(\theta) = J(\theta) + g(\theta), \qquad g(\theta) = \gamma\, \theta^T \theta$$

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1-y_i)\log(1-h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\left(1-(h_\Theta(\mathbf{x}_i))_k\right)\right]$$
$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right]$$

$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_\Theta J(\Theta)$

$J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

- But, tends to work well in practice

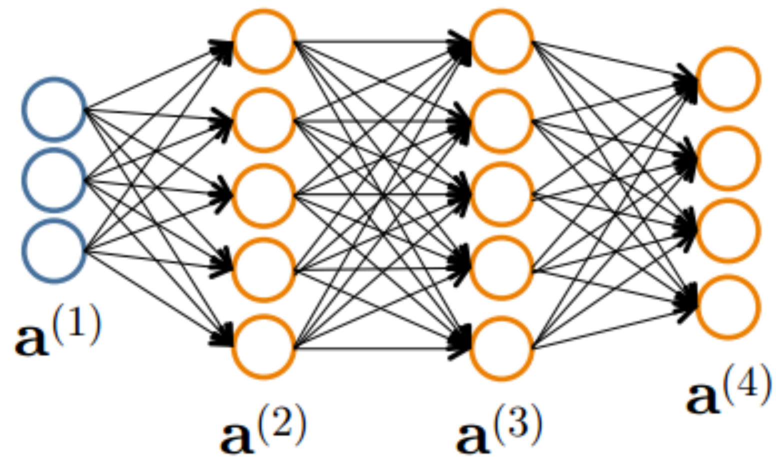
Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

UCLA ENGINEERING Computer Science

Forward Propagation

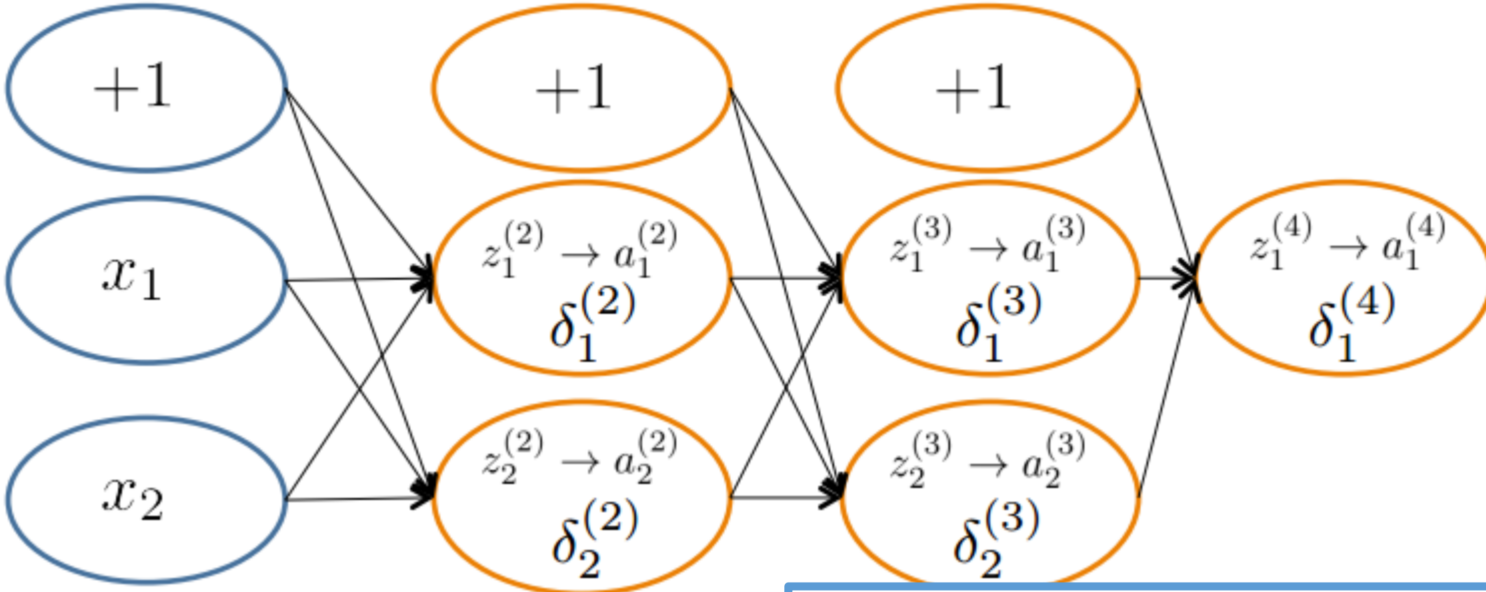
- Given one labeled training instance $(\mathbf{x},\ y)$:



<u>Forward Propagation</u>

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0{}^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0{}^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Backpropagation: Compute Gradient



$$\boxed{\frac{d}{dt} f(g(t)) = f'(g(t))g'(t) = \frac{df}{dg} \cdot \frac{dg}{dt}}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

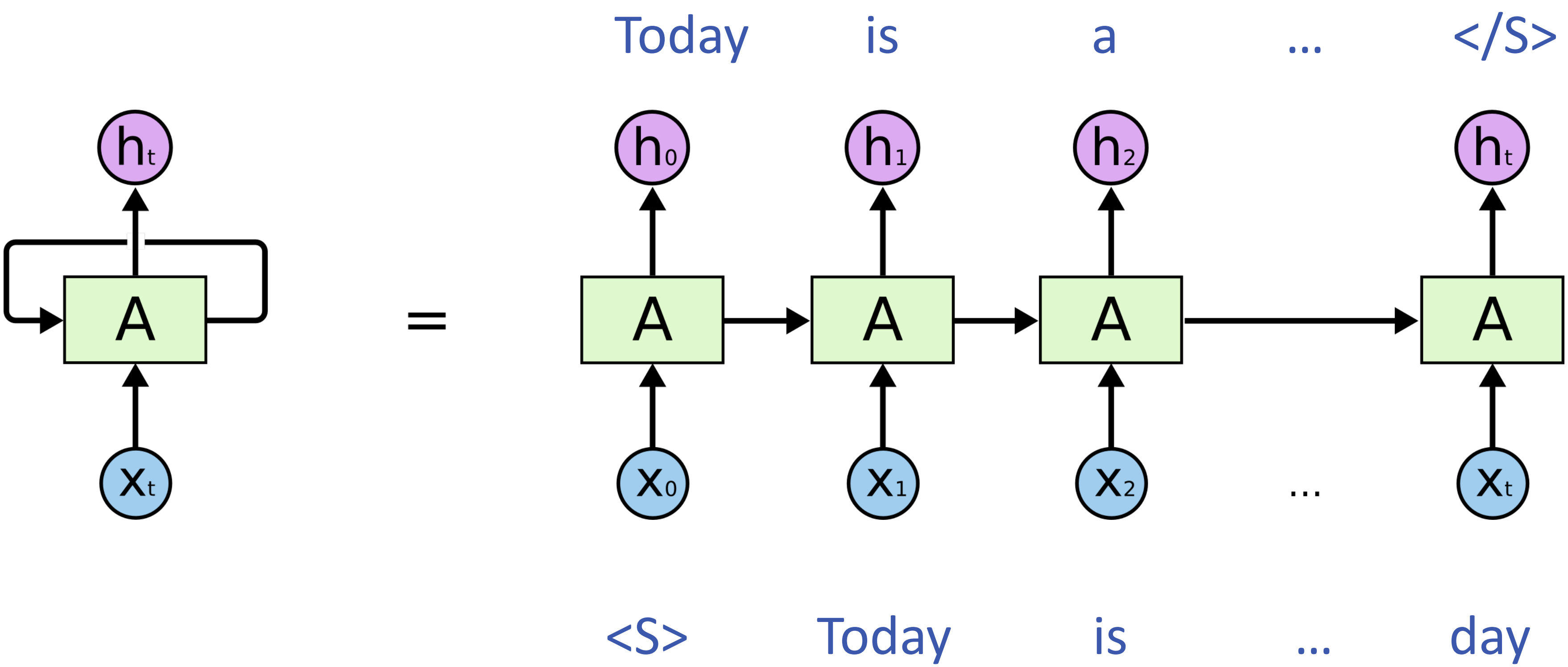
$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

This lecture

- Review: Neural Network
- Recurrent NN
- Representation learning in NLP

How to deal with input with variant size?

- Use same parameters

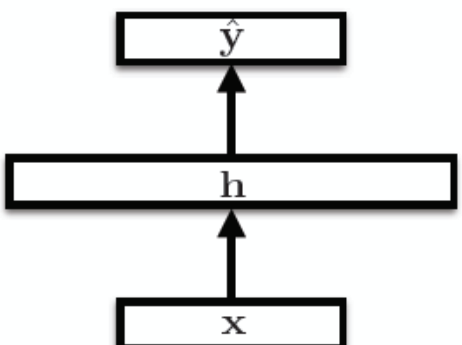


Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

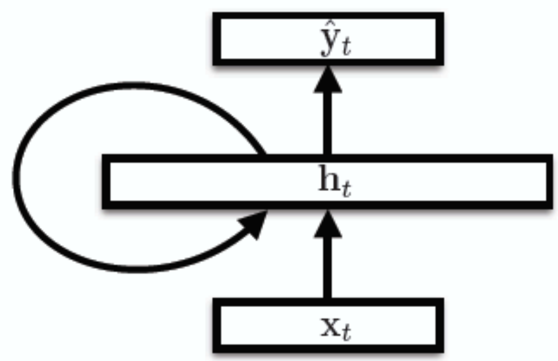
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

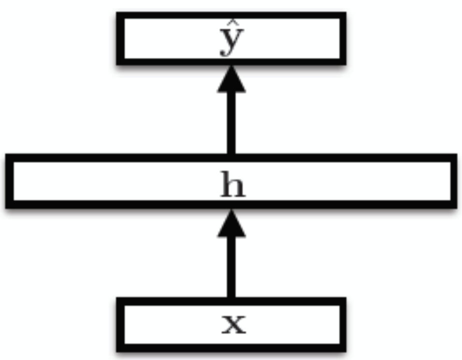


Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

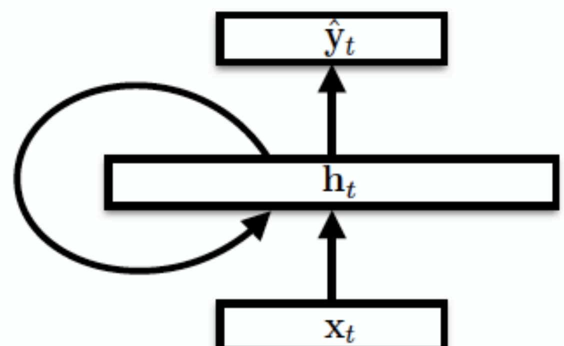


Recurrent NN

$$\cancel{\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})}$$

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

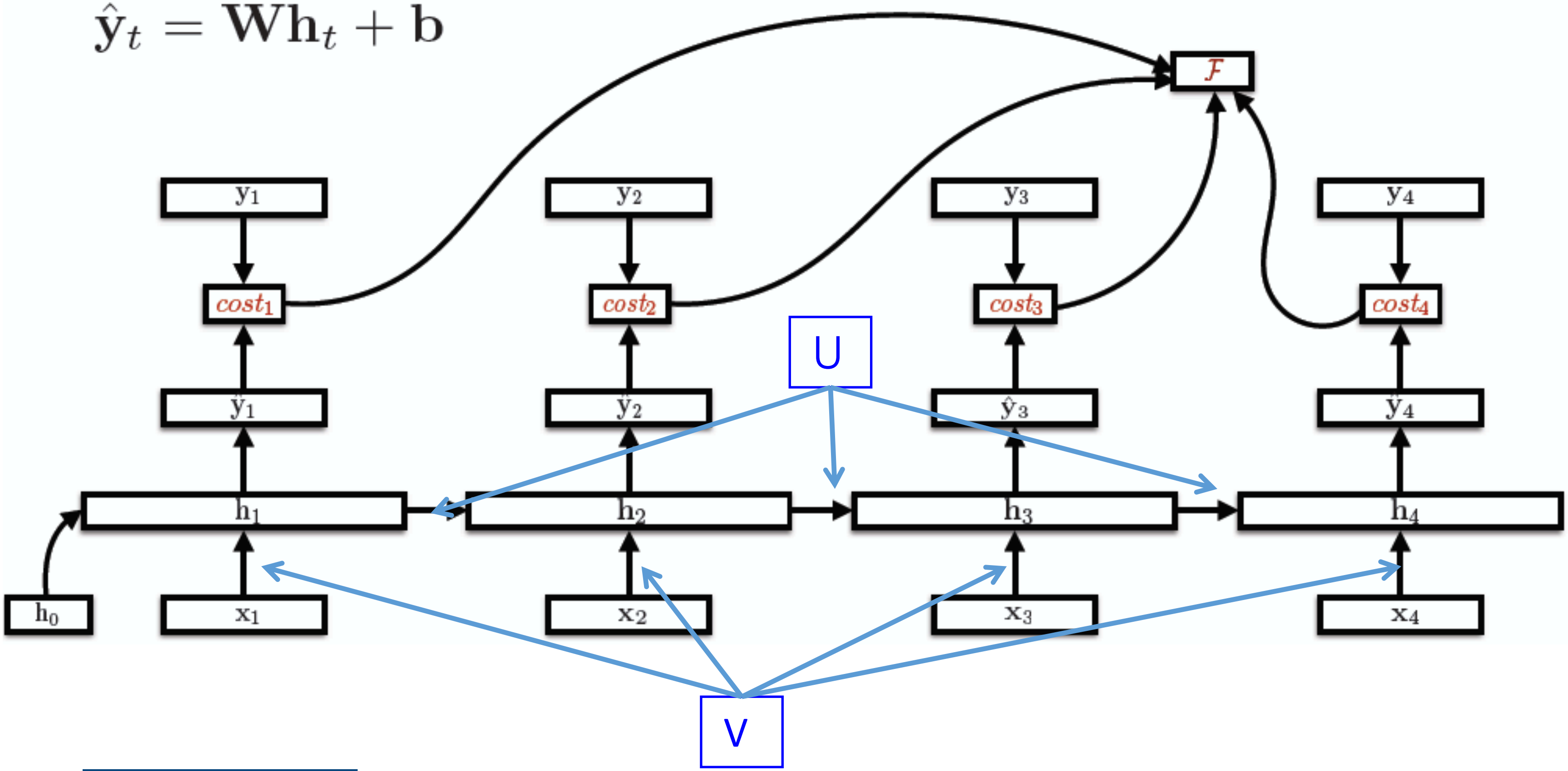


UCLA ENGINEERING
Computer Science

Unroll RNNs

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



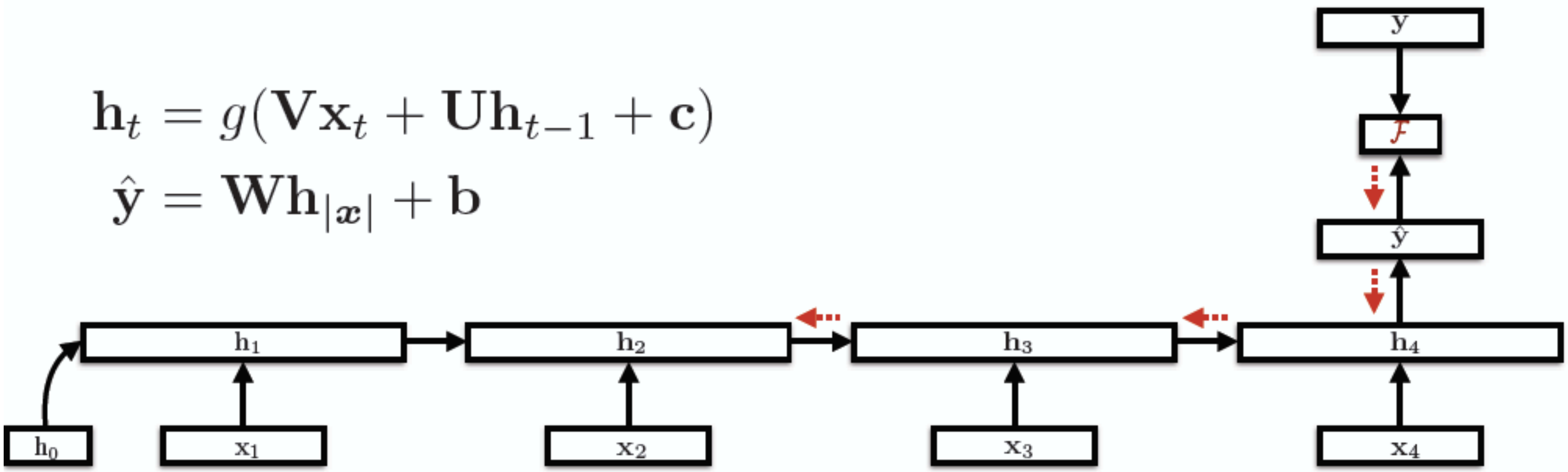
UCLA ENGINEERING
Computer Science

RNN training

- Back-propagation over time

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

UCLA ENGINEERING
Computer Science

Vanishing Gradients

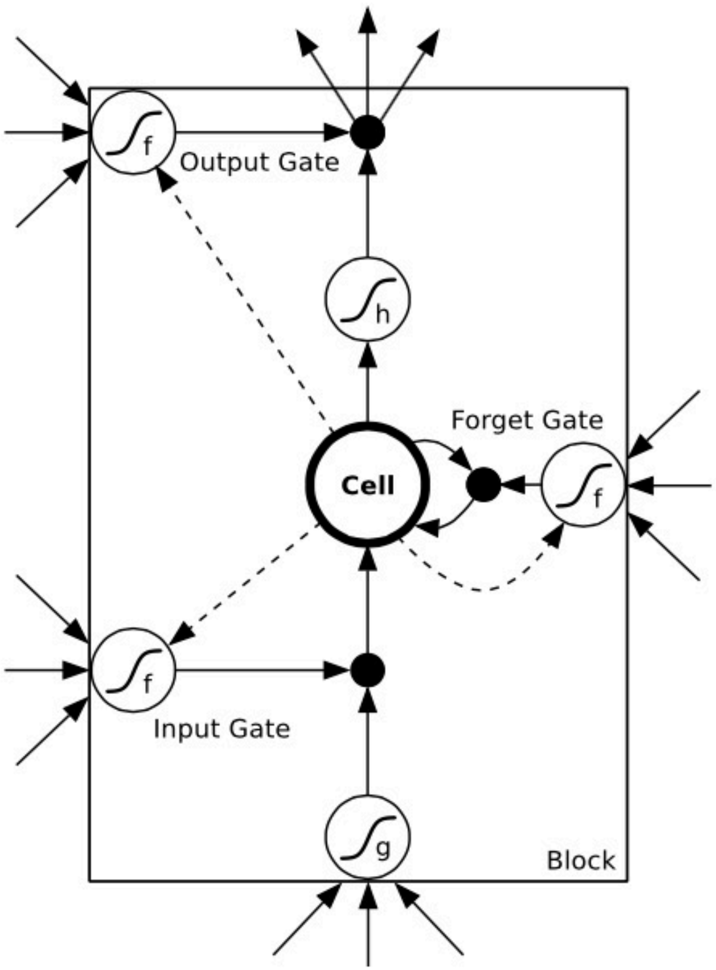
- For the traditional activation functions, each gradient term has the value in range (-1, 1).
- Multiplying *n* of these small numbers to compute gradients
- The longer the sequence is, the more severe the problems are.

UCLA ENGINEERING
Computer Science

RNNs characteristics

- Model hidden states (input) dependencies
- Errors “back propagation over time”
- Feature learning methods
- Vanishing gradient problem:
  cannot model long-distant dependencies of the hidden states.

Long-Short Term Memory Networks (LSTMs)



$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma(W_i[x_t, h_t] + b_i) \\ \sigma(W_f[x_t, h_t] + b_f) \\ \sigma(W_o[x_t, h_t] + b_o) \\ f(W_g[x_t, h_t] + b_g) \end{pmatrix}$$

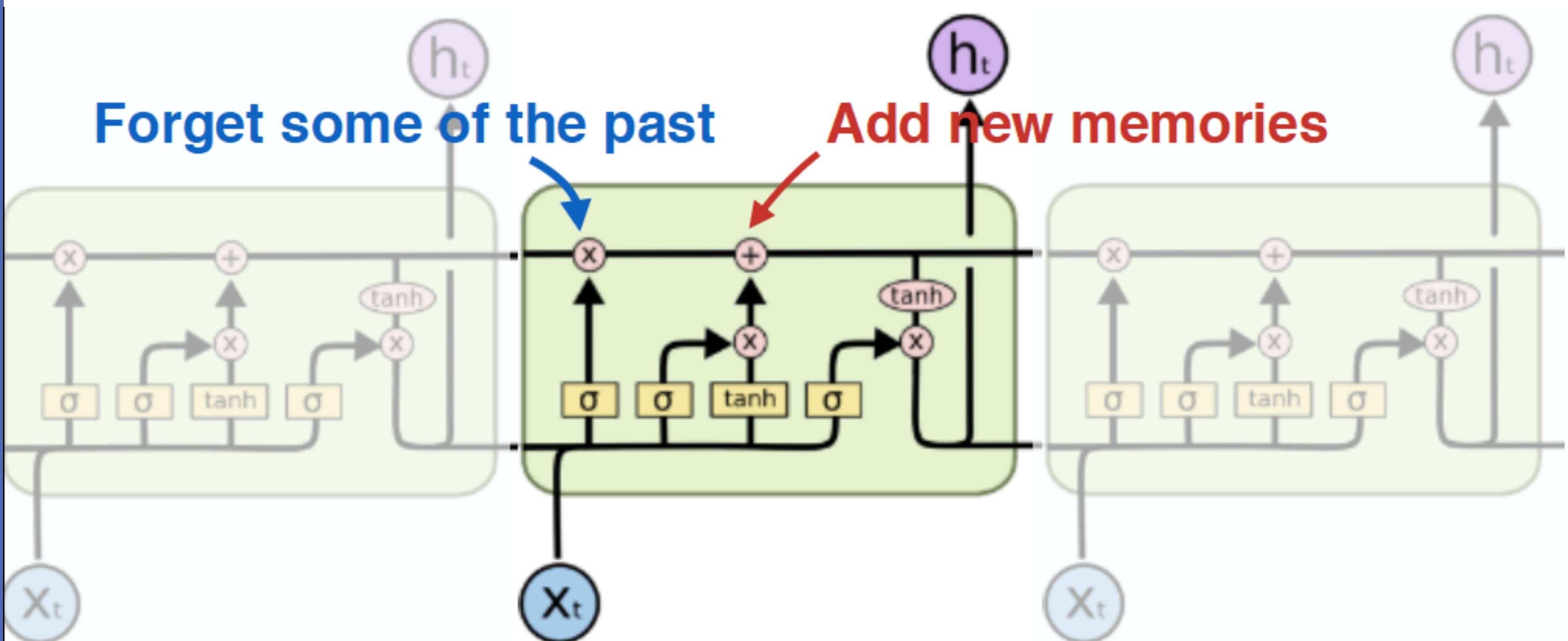
$$c_t = f_t * c_{t-1} + i_t * g_t$$

$$h_t = o_t * f(c_t)$$

Use gates to control the information to be added from the input, forgot from the previous memories, and outputted.
σ and f are *sigmoid* and *tanh* function respectively, to map the value to [-1, 1]

UCLA ENGINEERING
Computer Science

Another Visualization

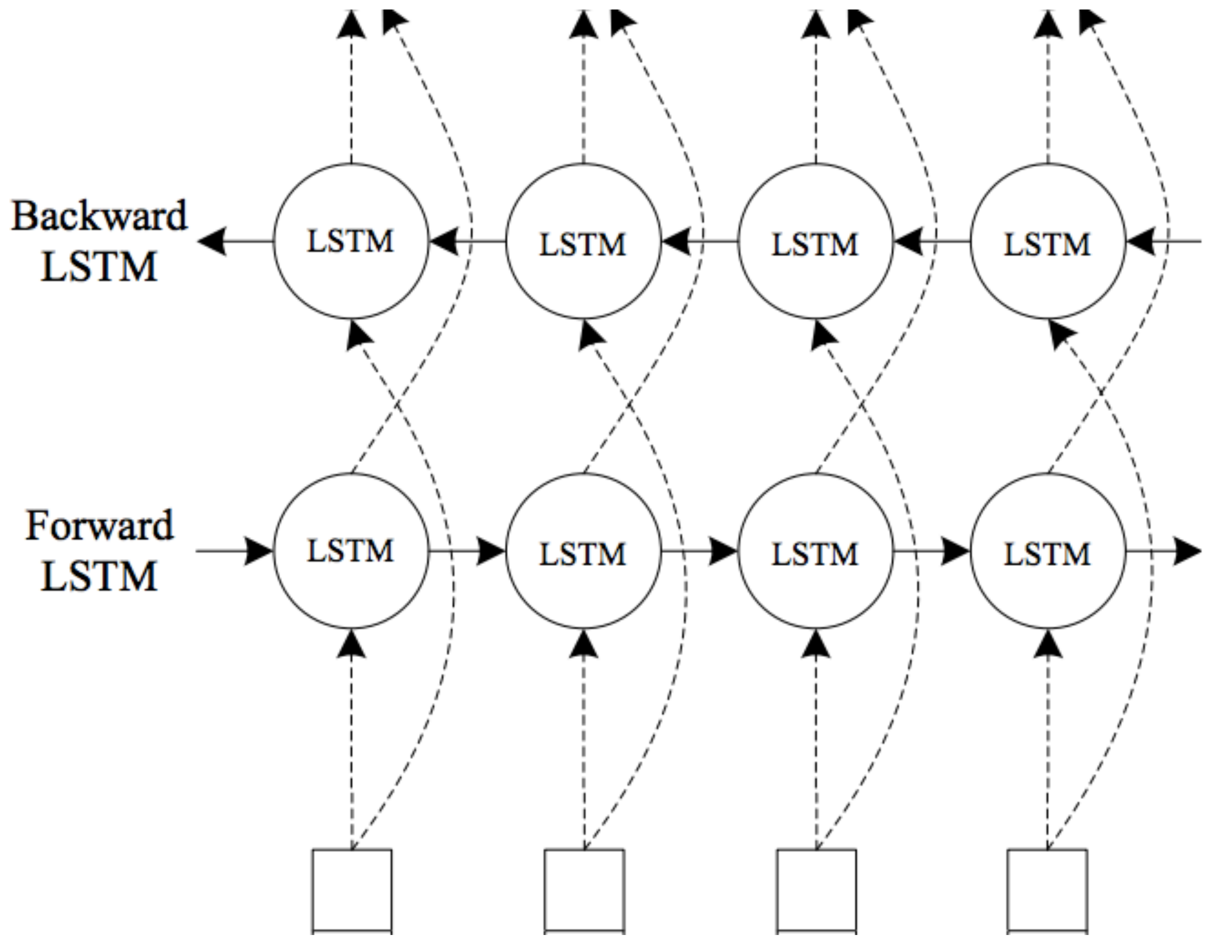


Capable of modeling long-distant dependencies between states.

UCLA ENGINEERING
Computer Science

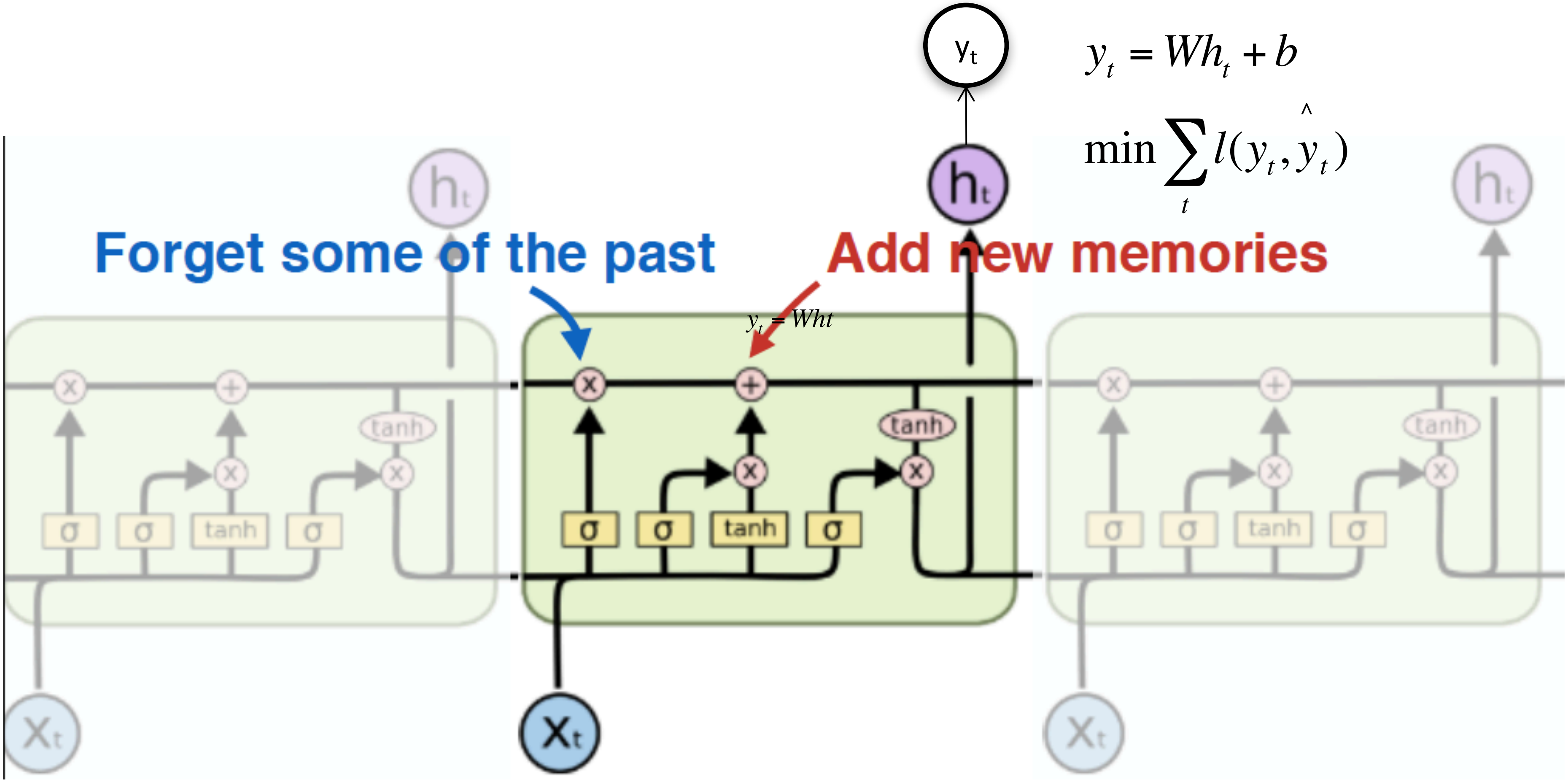
Figure credit: Christopher Olah

Bidirectional LSTMs



UCLA ENGINEERING
Computer Science

LSTMs for Sequential Tagging

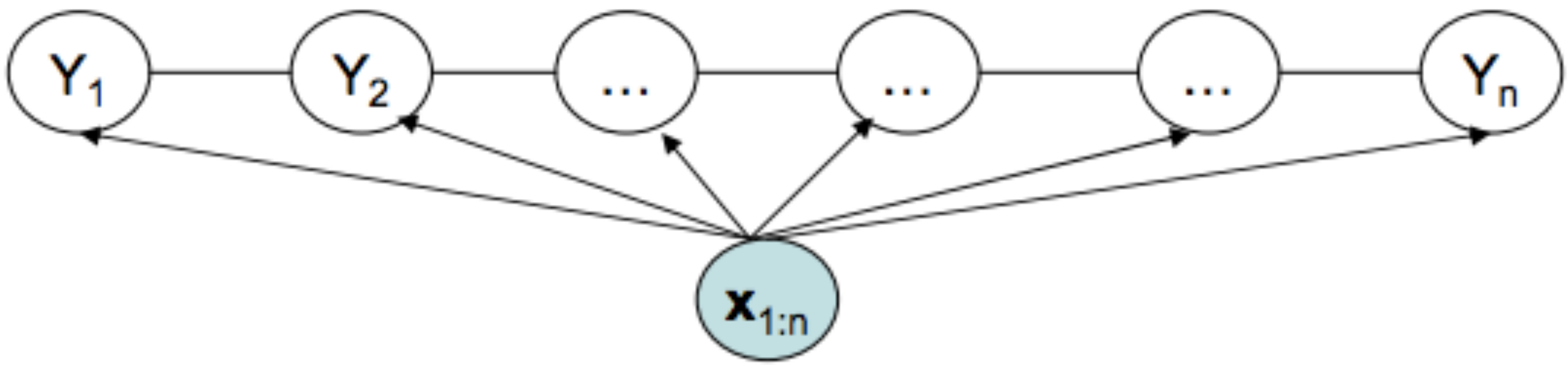


$$y_t = Wh_t + b$$

$$\min \sum_t l(y_t, \hat{y}_t)$$

Sophisticated model of input + local predictions.

Recall CRFs for Sequential Tagging



Arbitrary features on the input side

Markov assumption on the output side

UCLA ENGINEERING
Computer Science

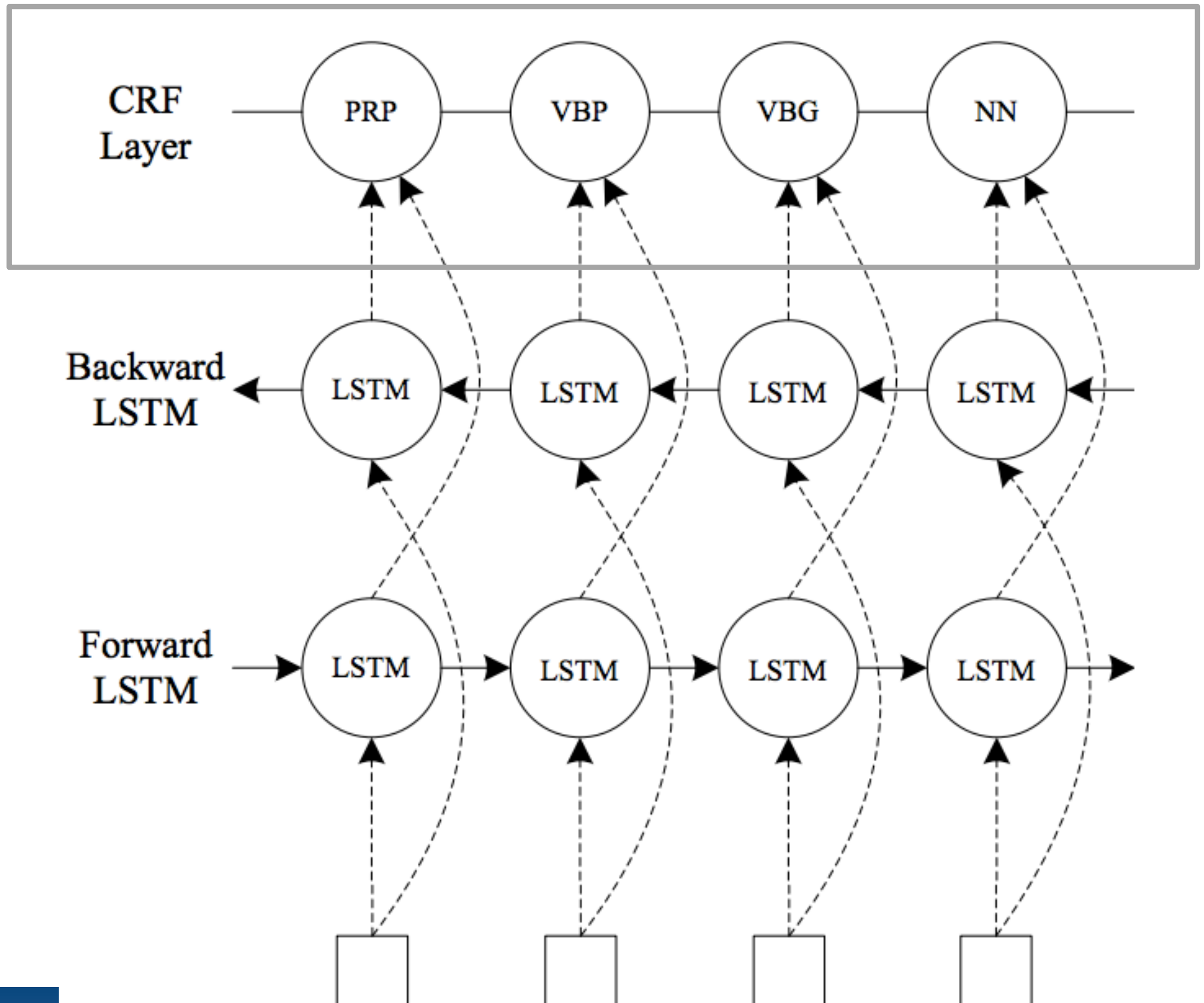
LSTMs for Sequential Tagging

- Completely ignored the interdependencies of the outputs

  - Will this work? Yes.
    - Liang et. al. (2008), *Structure Compilation: Trading Structure for Features*

  - Is this the best model? Not necessarily.

Combining CRFs with LSTMs



Combining Two Benefits

- Directly model output dependencies by CRFs.
- Powerful automatic feature learning using biLSTMs.
- Jointly training all the parameters to “share the modeling responsibilities”