Lecture 5: Representation Learning

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

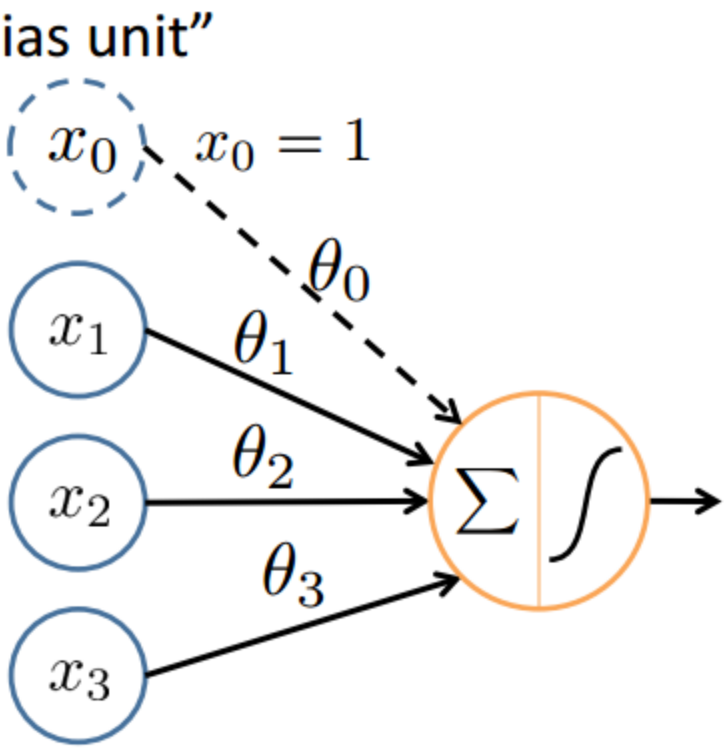
Couse webpage: https://uclanlp.github.io/CS269-17/

UCLA ENGINEERING Computer Science

This lecture

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

Neural Network

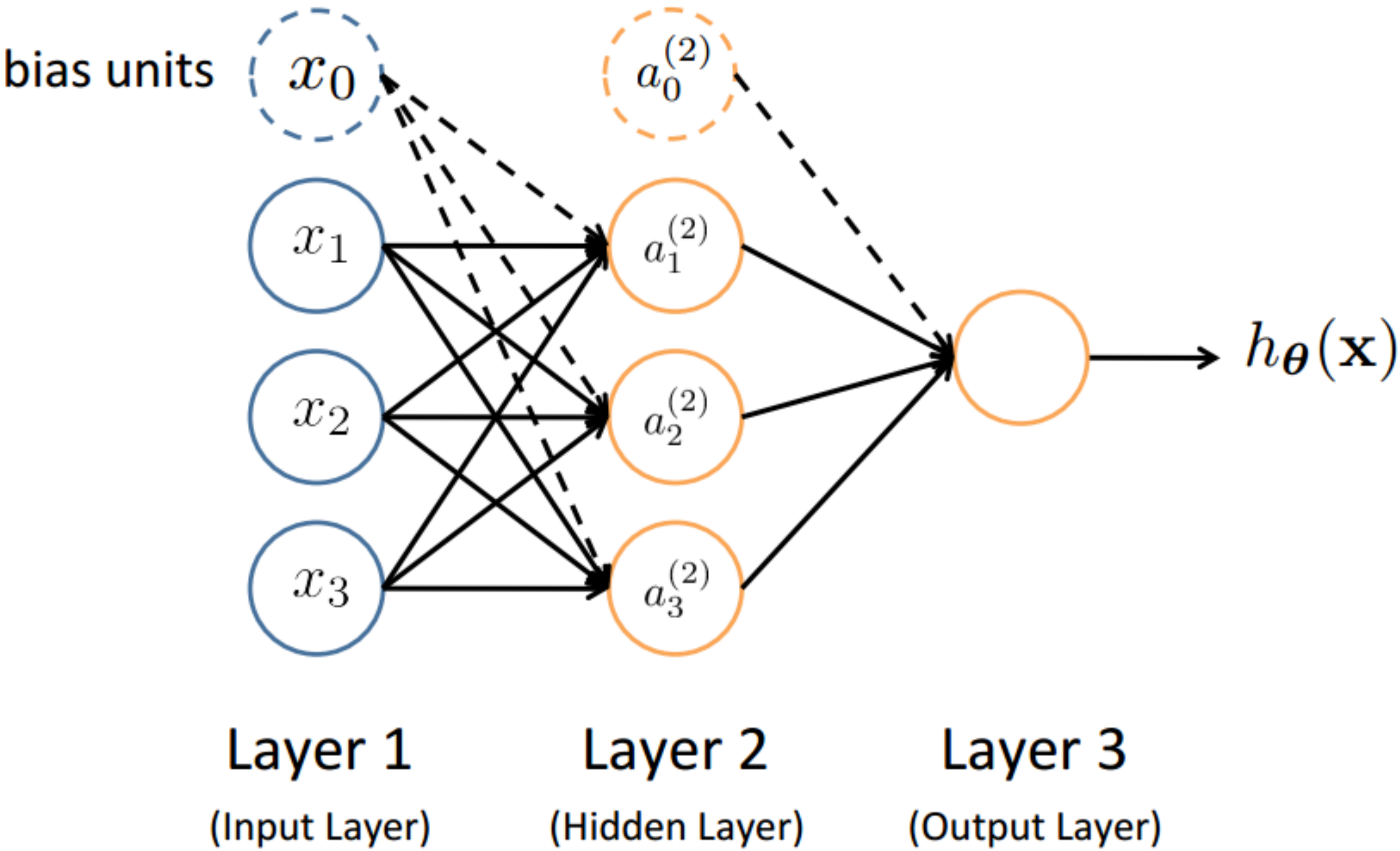


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

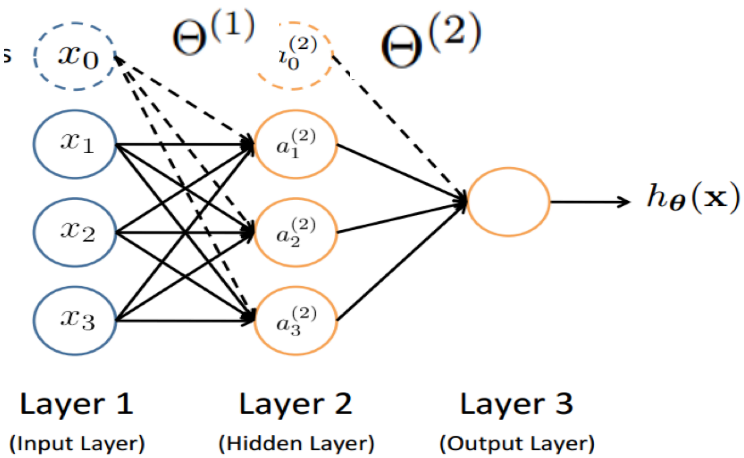
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Neural Network (feed forward)



Feed-Forward Process

- Input layer units are features (in NLP, e.g., words)
  - Usually, one-hot vector or word embedding
- 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 $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

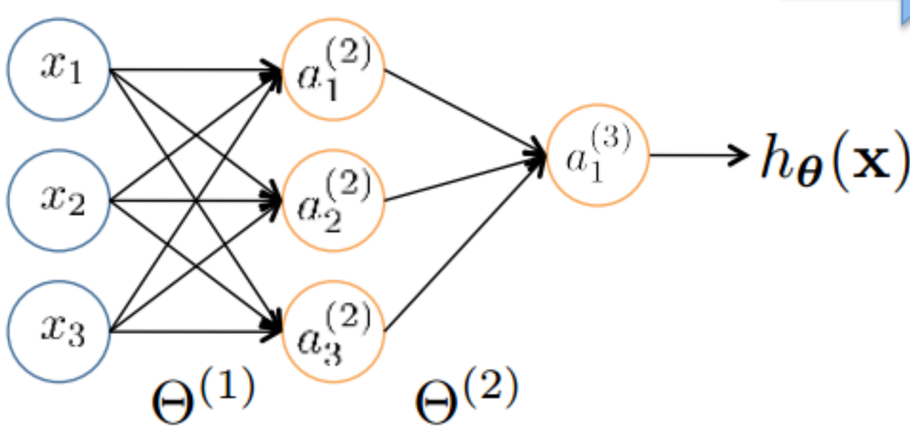
Vector Representation

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

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

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

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



Feed-Forward Steps:

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

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

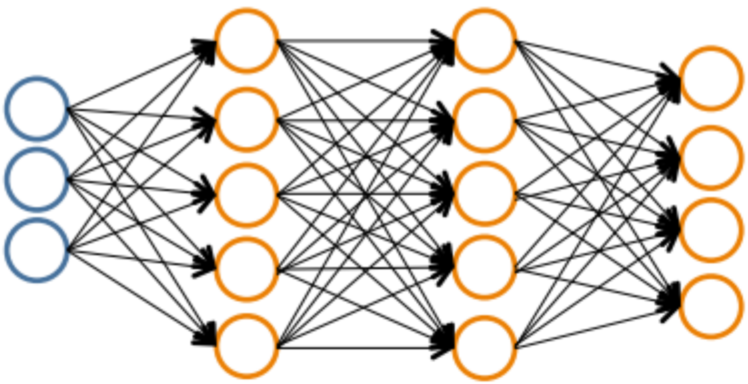
$$\text{Add } a_0^{(2)} = 1$$

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

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

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

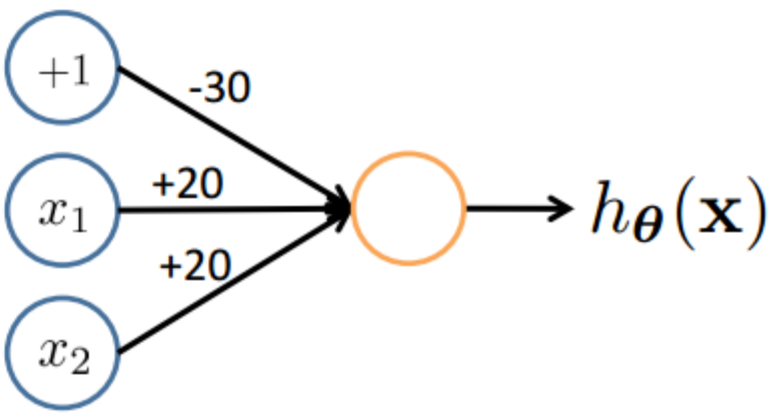
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
when truck

Why staged predictions?

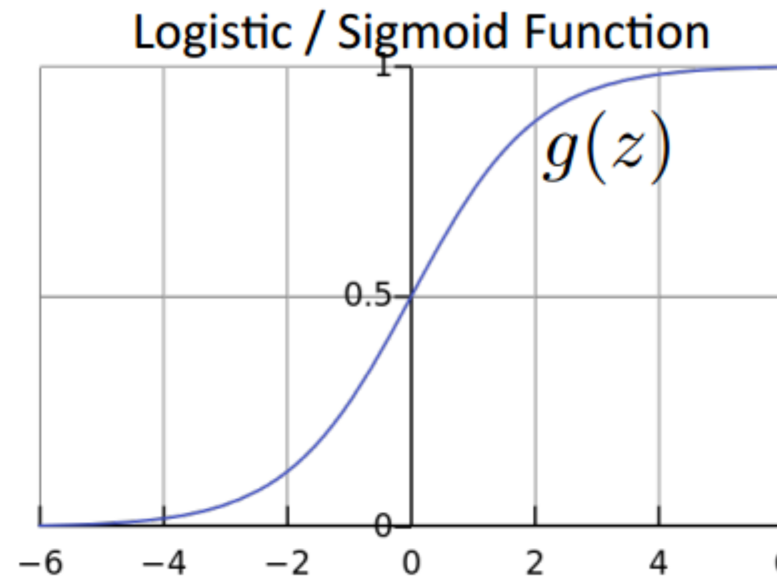
Simple example: AND

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

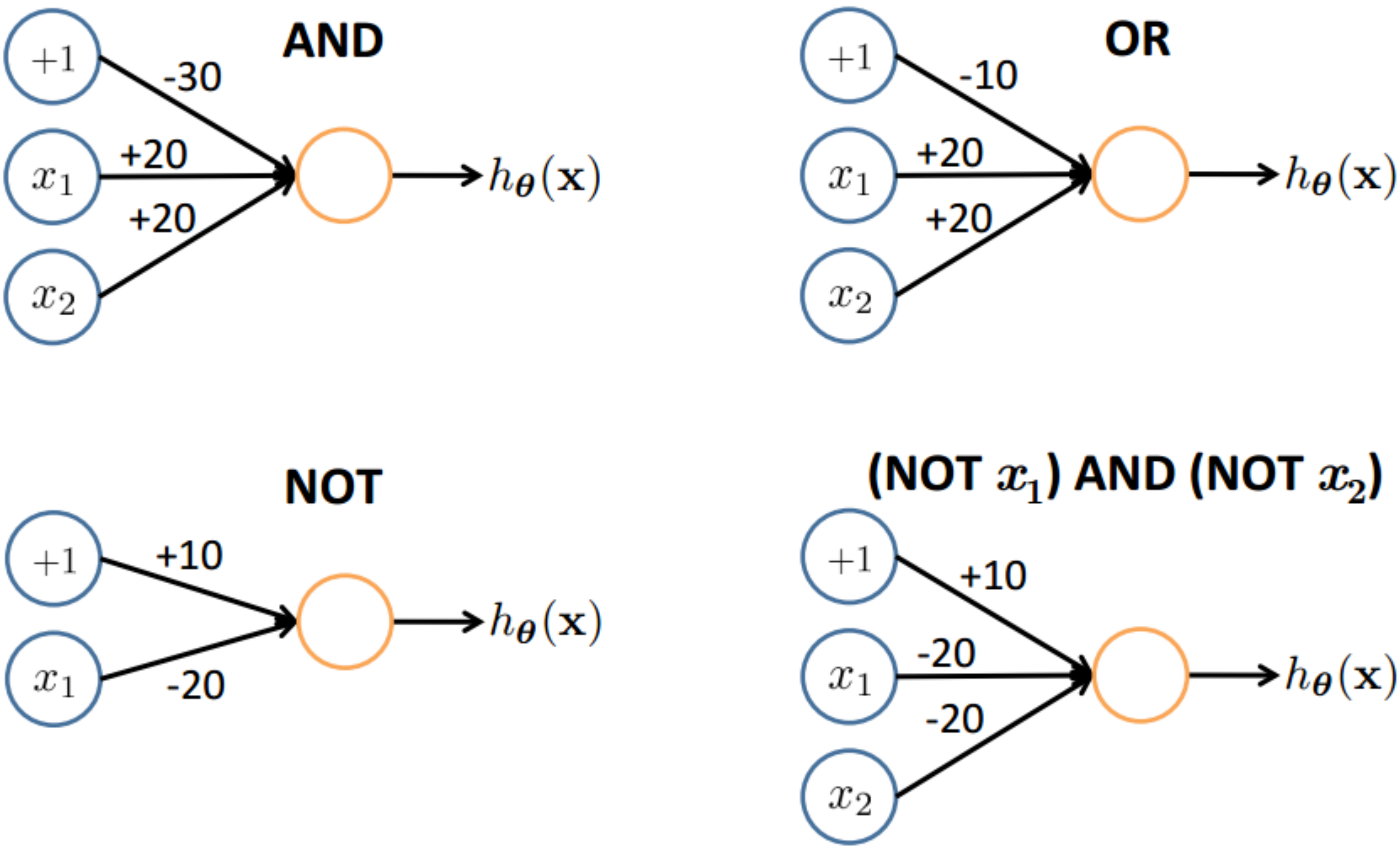
$y = x_1 \text{ AND } x_2$



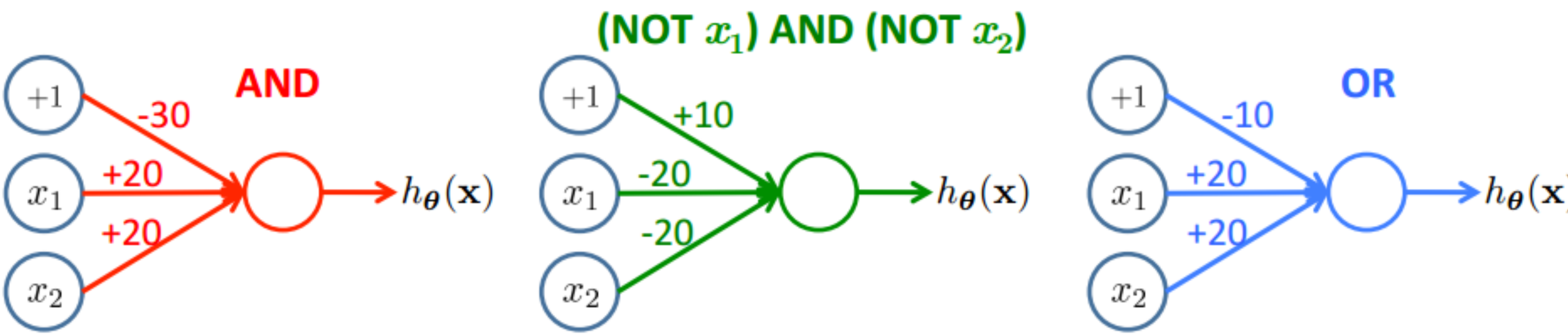
$$h_{\Theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

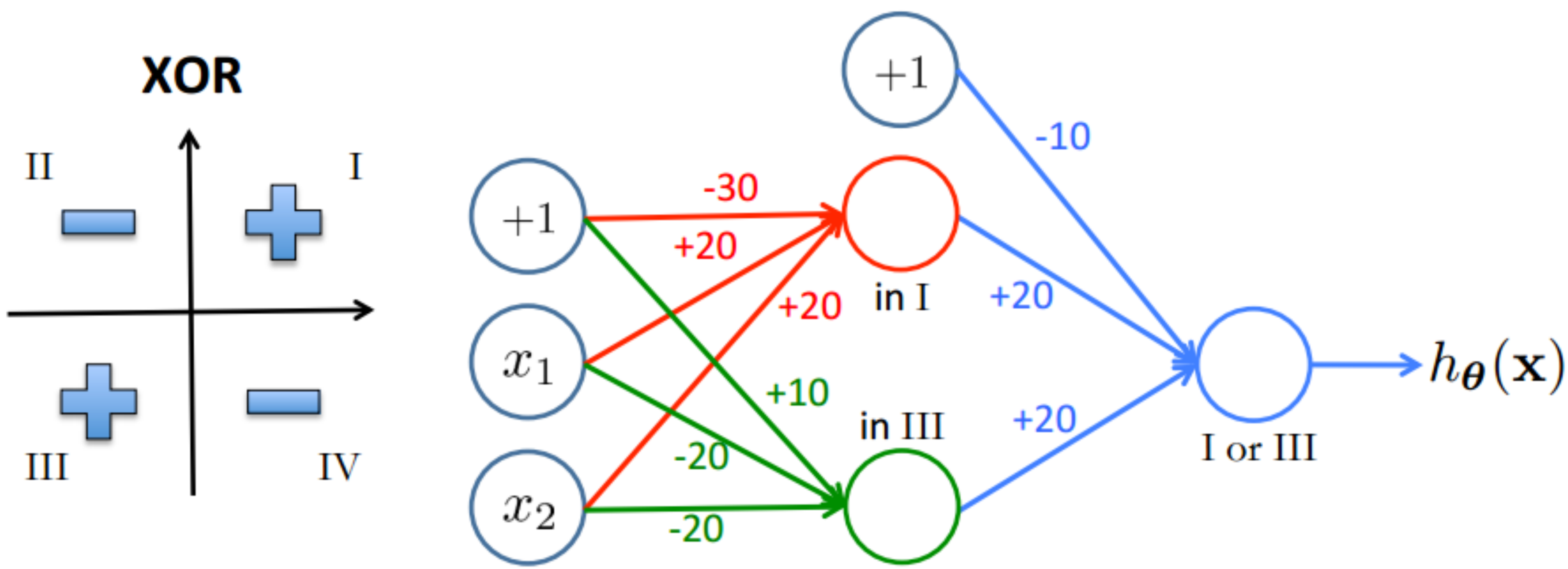


x_1	x_2	$h_{\Theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

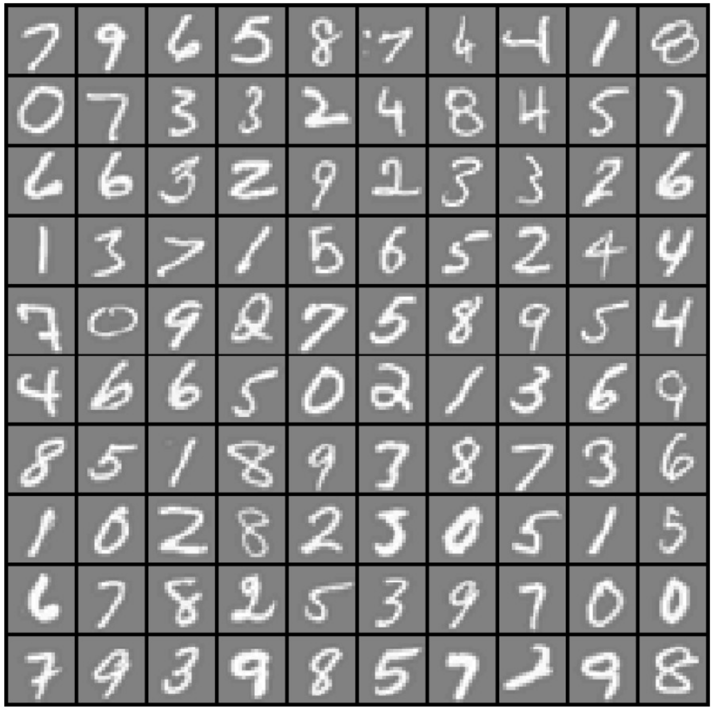

AND
+1
-30
x_1
+20
+20
x_2
$h_\theta(\mathbf{x})$
OR
+1
-10
x_1
+20
+20
x_2
$h_\theta(\mathbf{x})$
NOT
+1
+10
x_1
-20
$h_\theta(\mathbf{x})$
(NOT x_1) AND (NOT x_2)
+1
+10
x_1
-20
-20
x_2
$h_\theta(\mathbf{x})$

Combining Representations to Create Non-Linear Functions



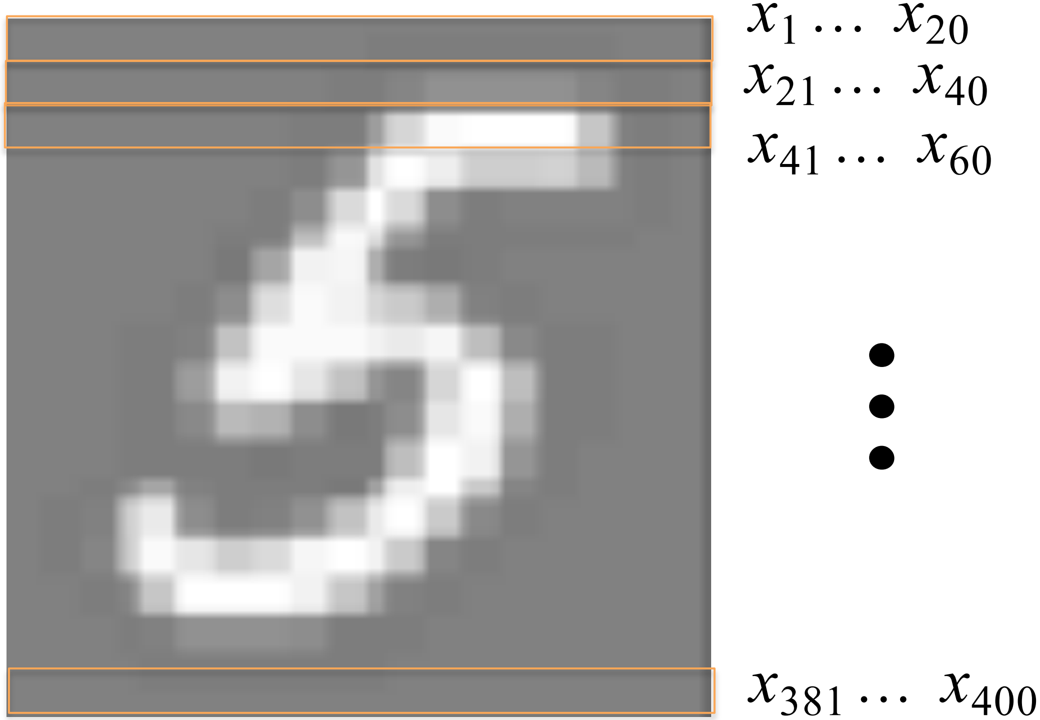


Layering Representations



20×20 pixel images

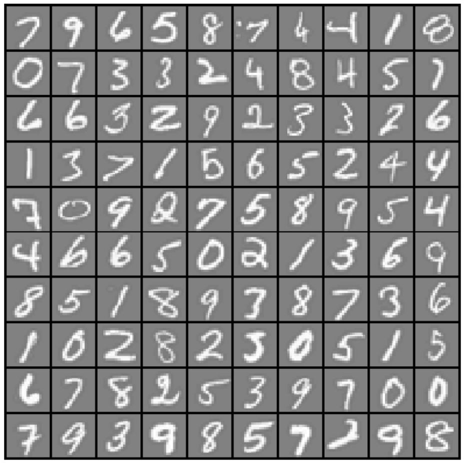
$d = 400$ 10 classes

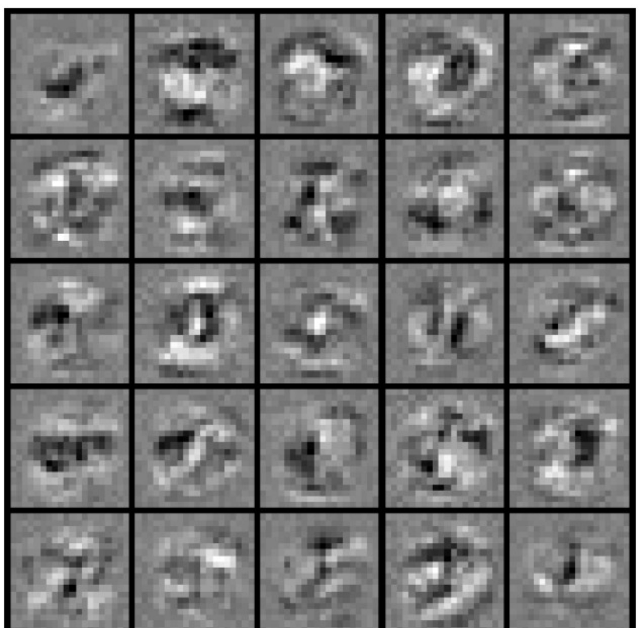


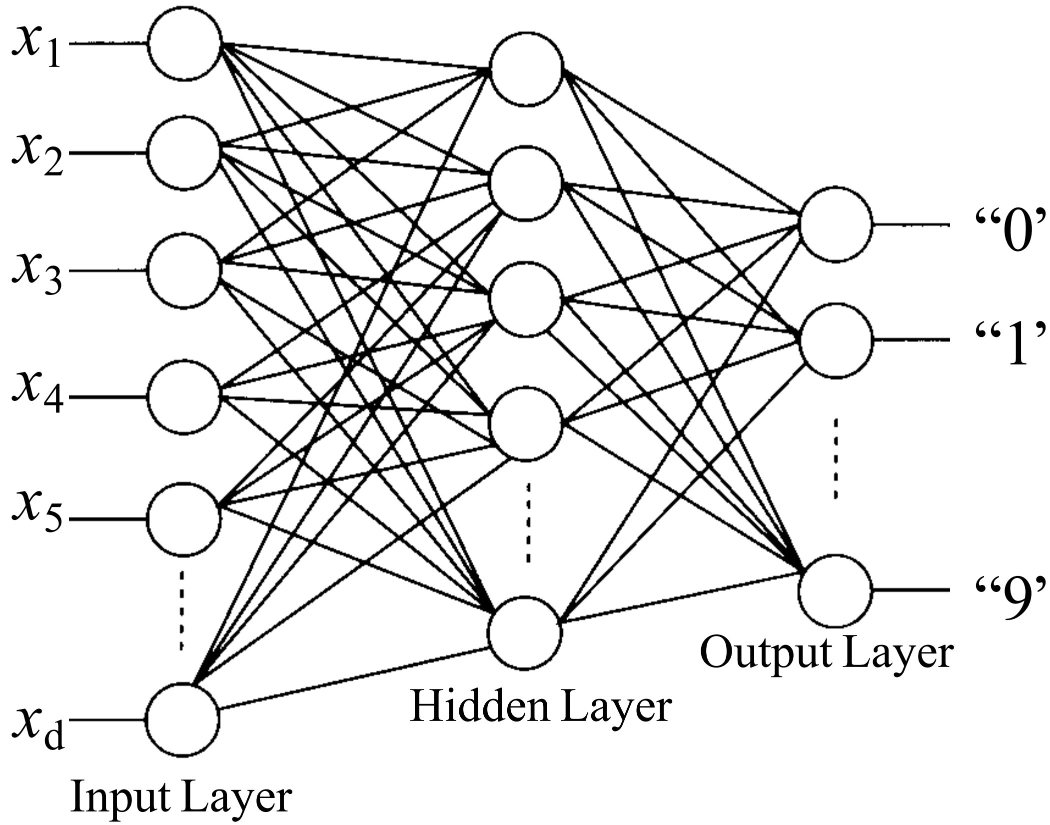
Each image is "unrolled" into a vector x of pixel intensities

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

Recap: Logistic regression

$$\min_{\boldsymbol{\theta}} \quad \frac{\lambda}{2n}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{n}\sum_i \log(1 + e^{-\mathrm{y_i}(\boldsymbol{\theta}^\mathrm{T}\mathbf{x}_\mathrm{i})})$$

Let $\mathrm{h}_\theta(x_i) = 1/(1 + e^{-\theta^T x_i})$ (probability $y = 1$ given x_i)

$$\frac{\lambda}{2n}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{n}\sum_i \mathrm{y_i} \log(h_\theta(x_i)) + (1 - y_i)\,(\log(1 - h_\theta(x_i))$$

Cost Function

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

Logistic Regression:

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

Neural Network:

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

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

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

Based on slide by Andrew Ng

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:

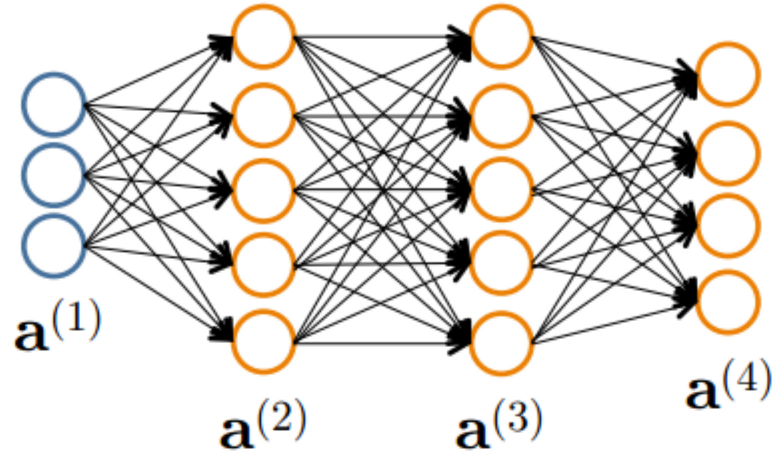
- $J(\Theta)$
- $\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$

UCLA ENGINEERING Computer Science

Based on slide by Andrew Ng

Forward Propagation

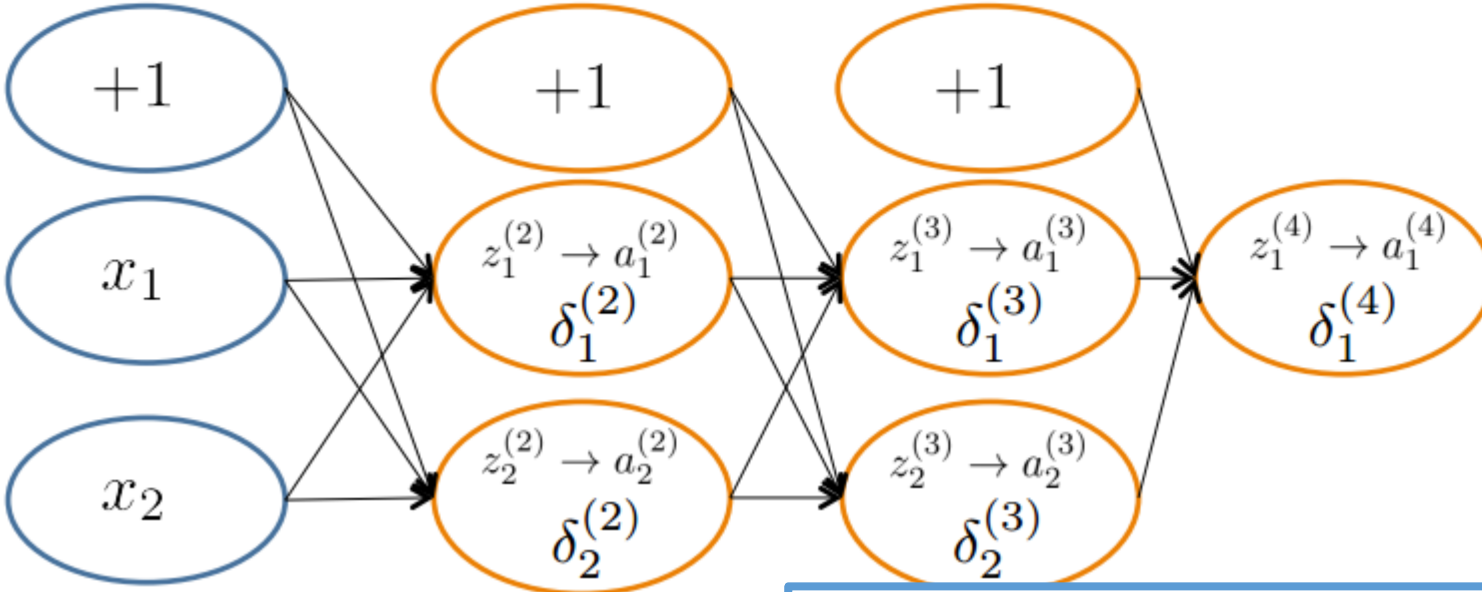
- Given one labeled training instance $(\mathbf{x}, y)$:



<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



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

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

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

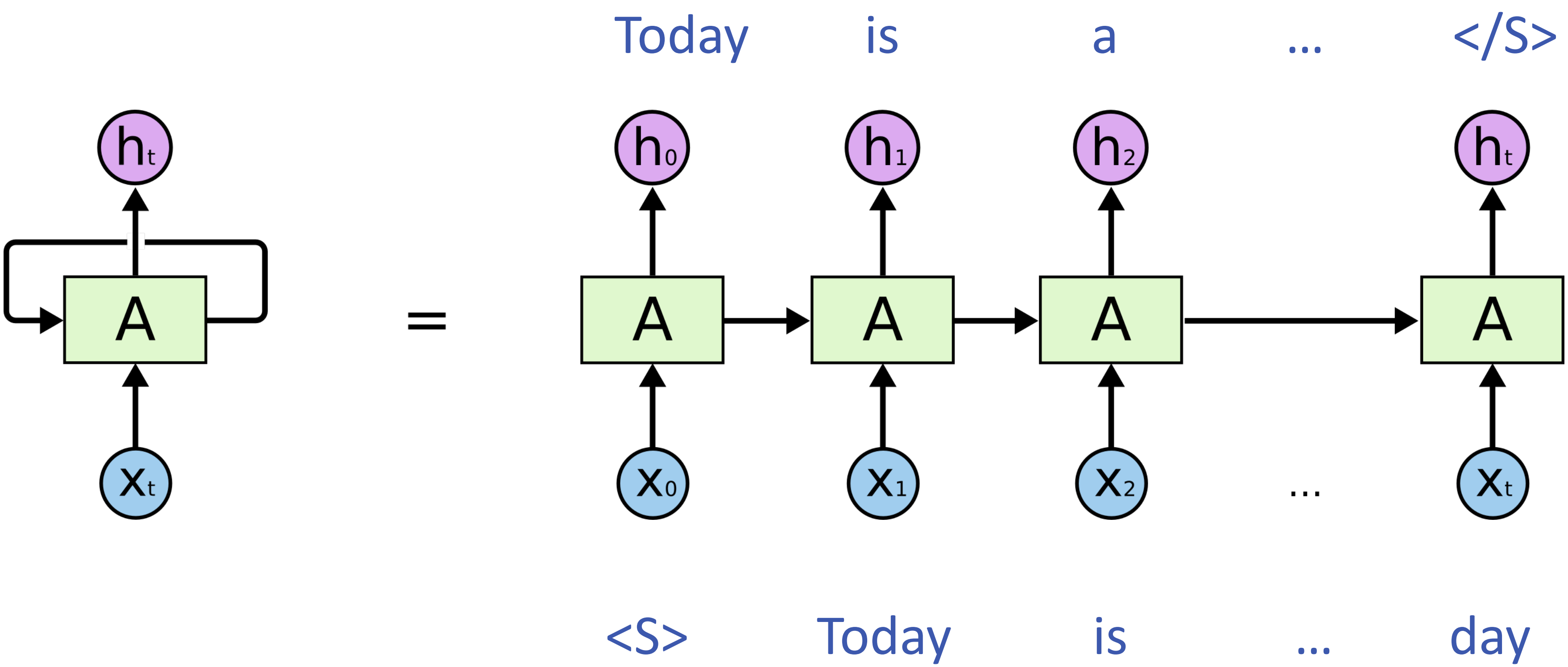
where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

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

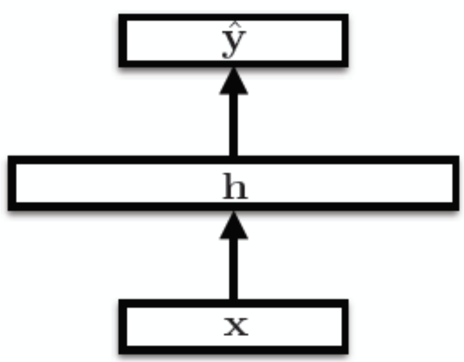
$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

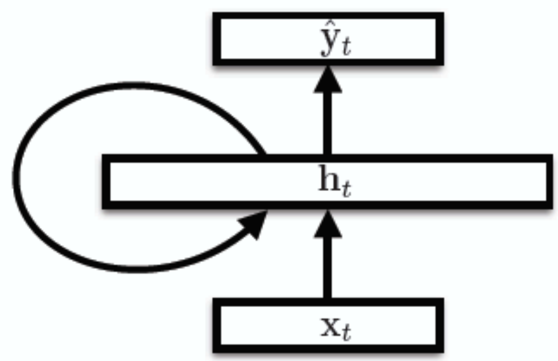
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

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

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

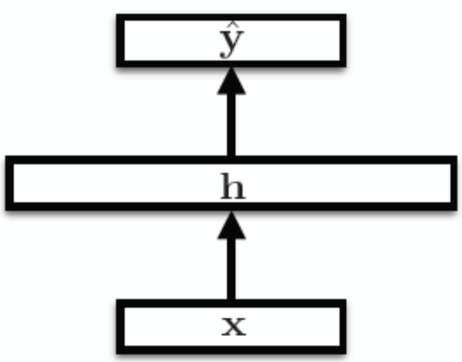
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

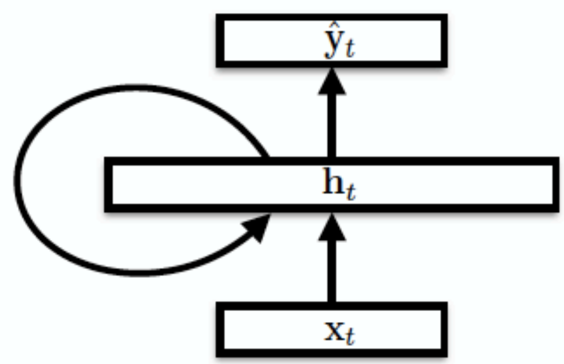
Recurrent NN

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

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

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

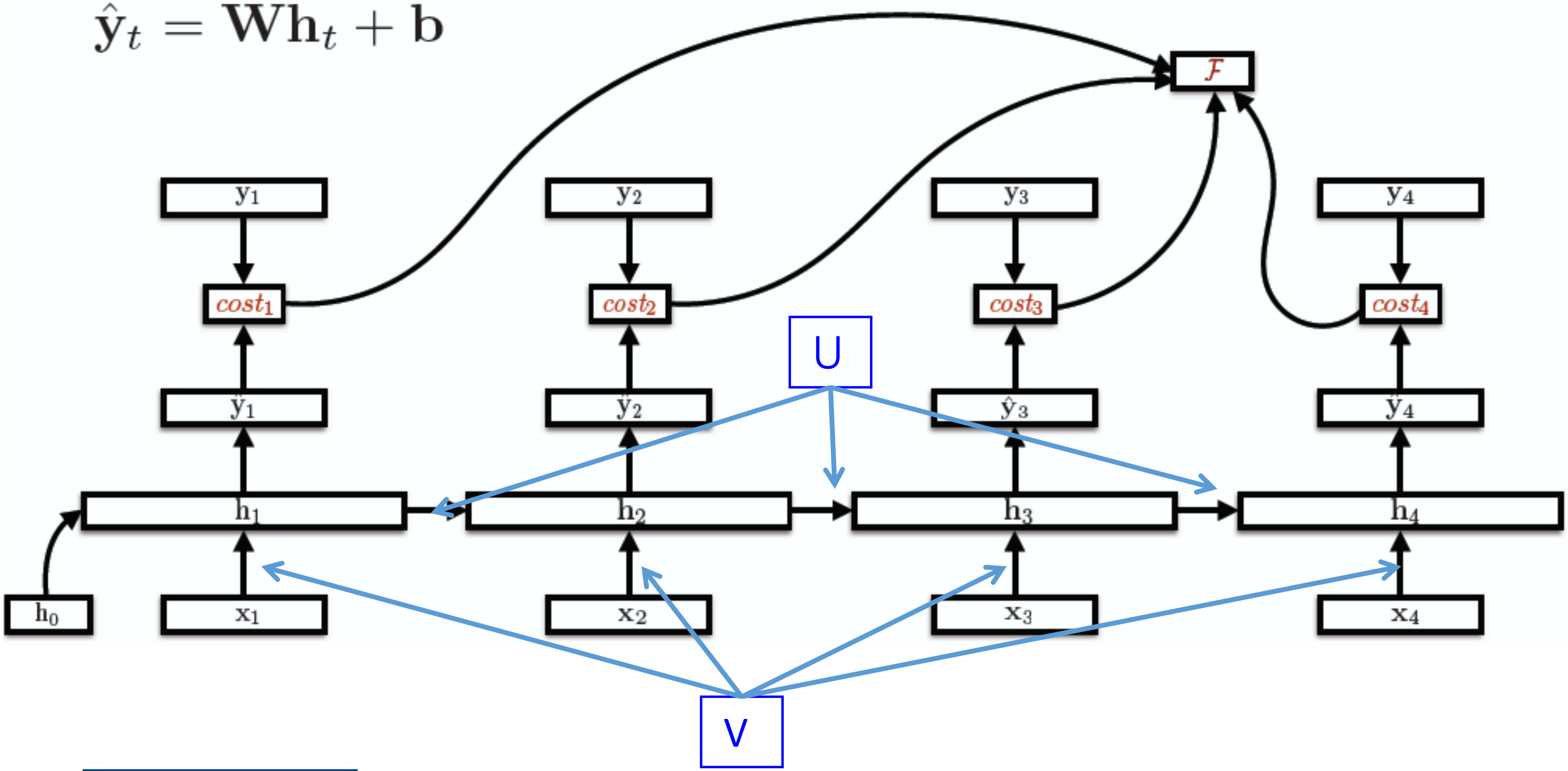




Unroll RNNs

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

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



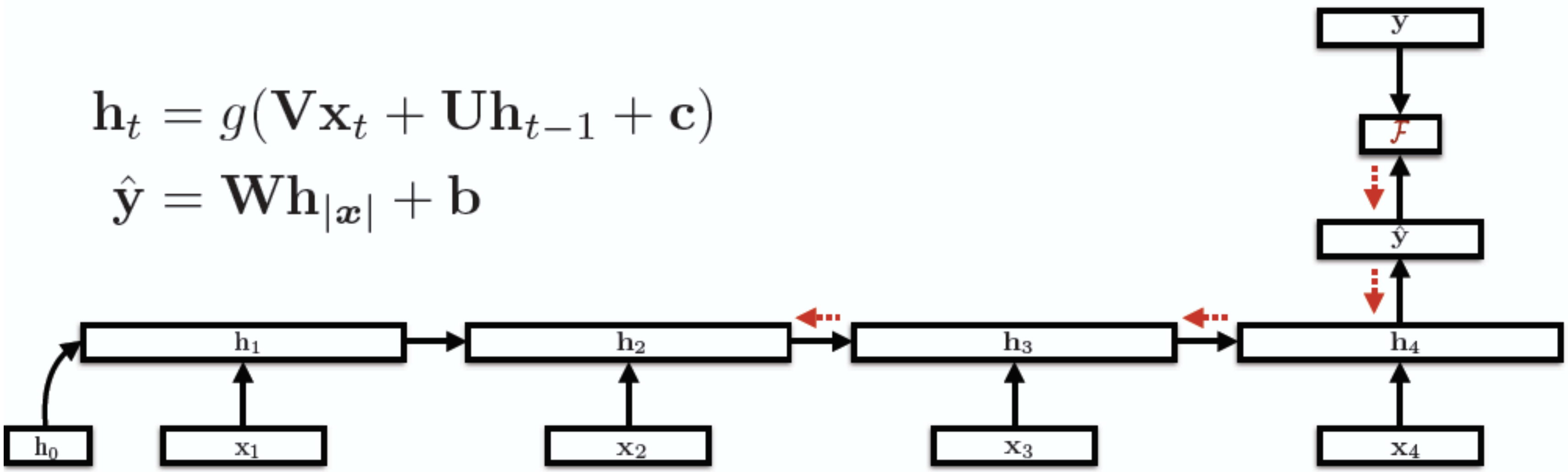
UCLA ENGINEERING
Computer Science

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

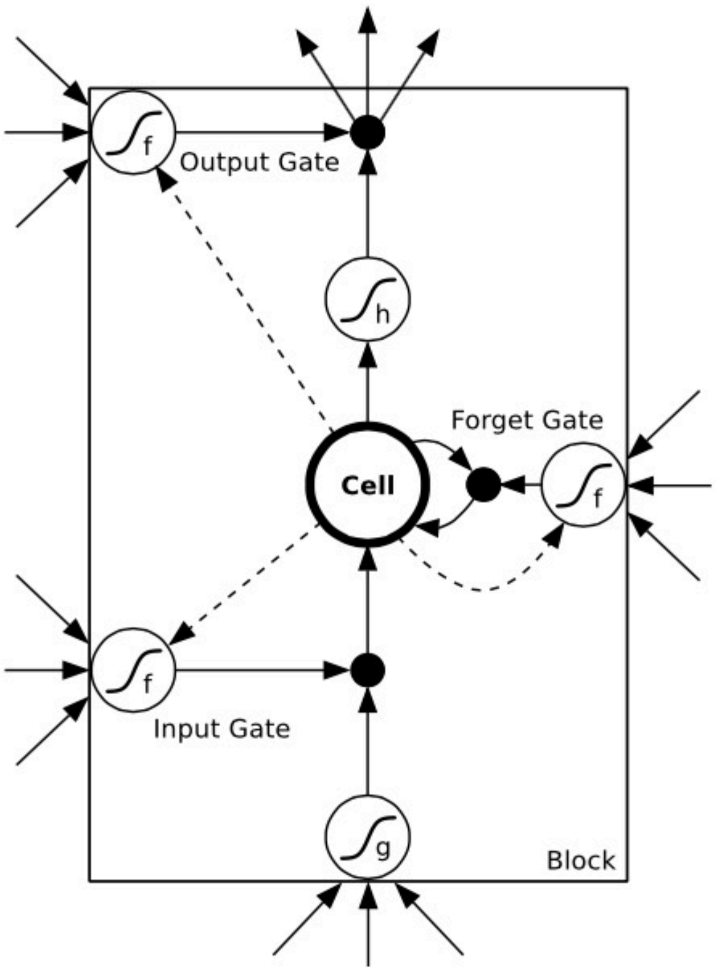
Vanishing Gradients

- 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.

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)



$$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \boldsymbol{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\boldsymbol{W}_i[\boldsymbol{x}_t, \boldsymbol{h}_t] + \boldsymbol{b}_i) \\ \sigma(\boldsymbol{W}_f[\boldsymbol{x}_t, \boldsymbol{h}_t] + \boldsymbol{b}_f) \\ \sigma(\boldsymbol{W}_o[\boldsymbol{x}_t, \boldsymbol{h}_t] + \boldsymbol{b}_o) \\ f(\boldsymbol{W}_g[\boldsymbol{x}_t, \boldsymbol{h}_t] + \boldsymbol{b}_g) \end{pmatrix}$$

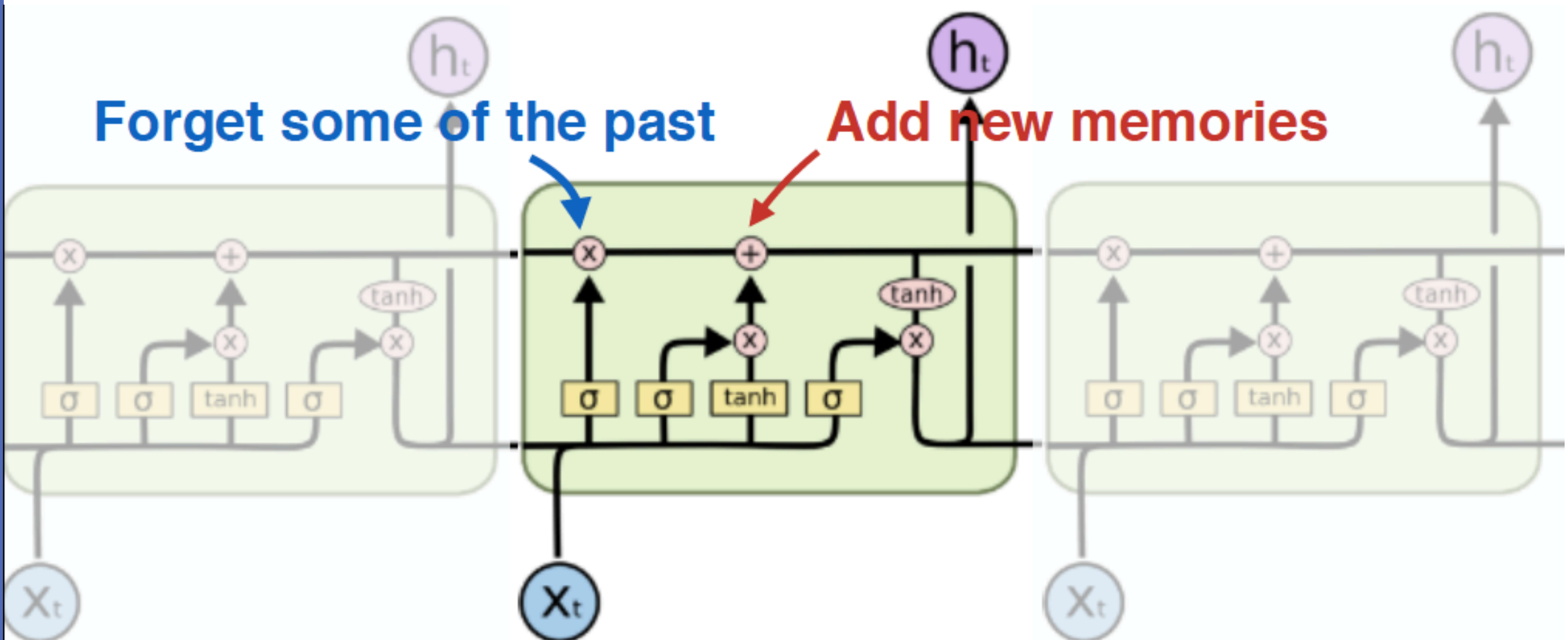
$$\boldsymbol{c}_t = \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \boldsymbol{g}_t$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t * f(\boldsymbol{c}_t)$$

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

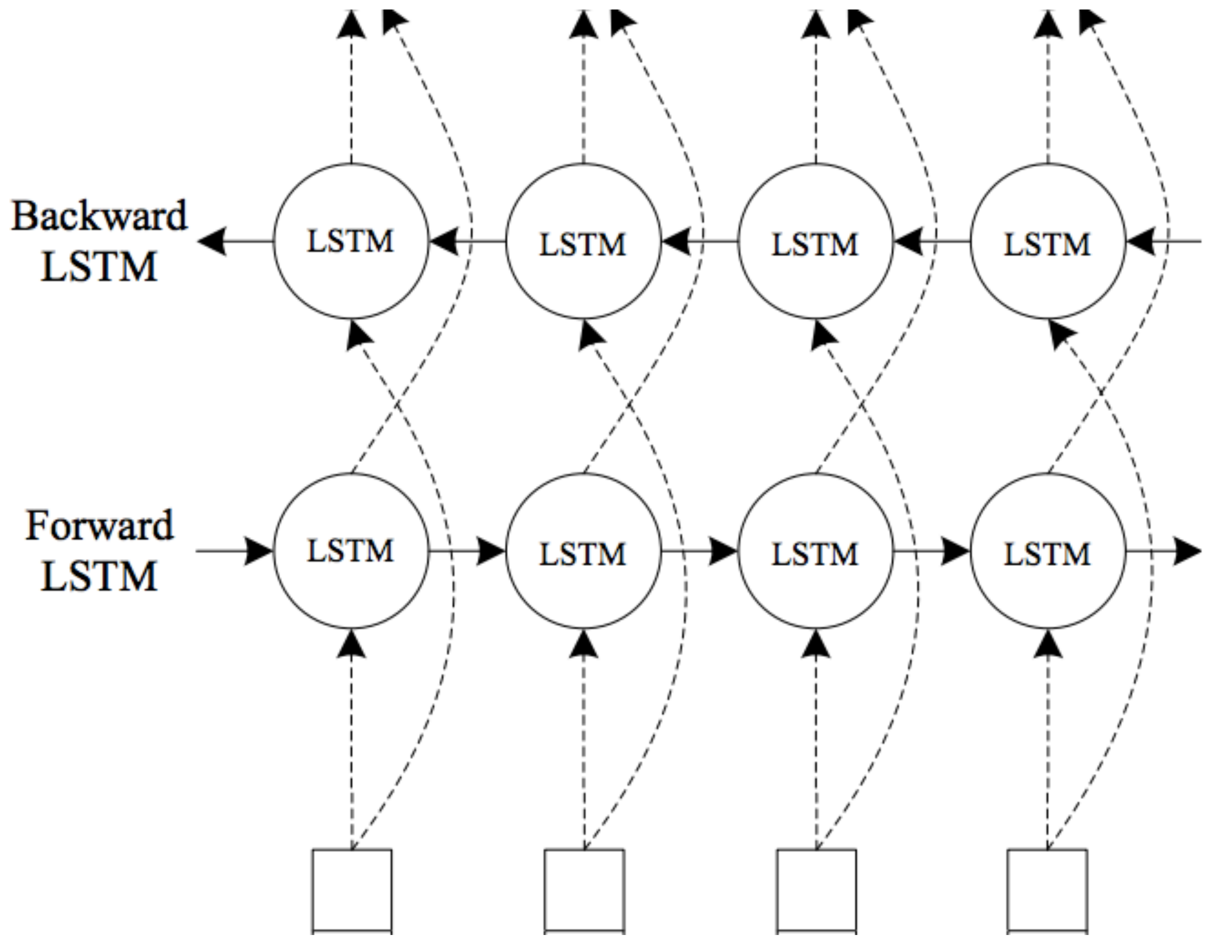


Capable of modeling long-distant dependencies between states.

UCLA ENGINEERING Computer Science

Figure credit: Christopher Olah

Bidirectional LSTMs

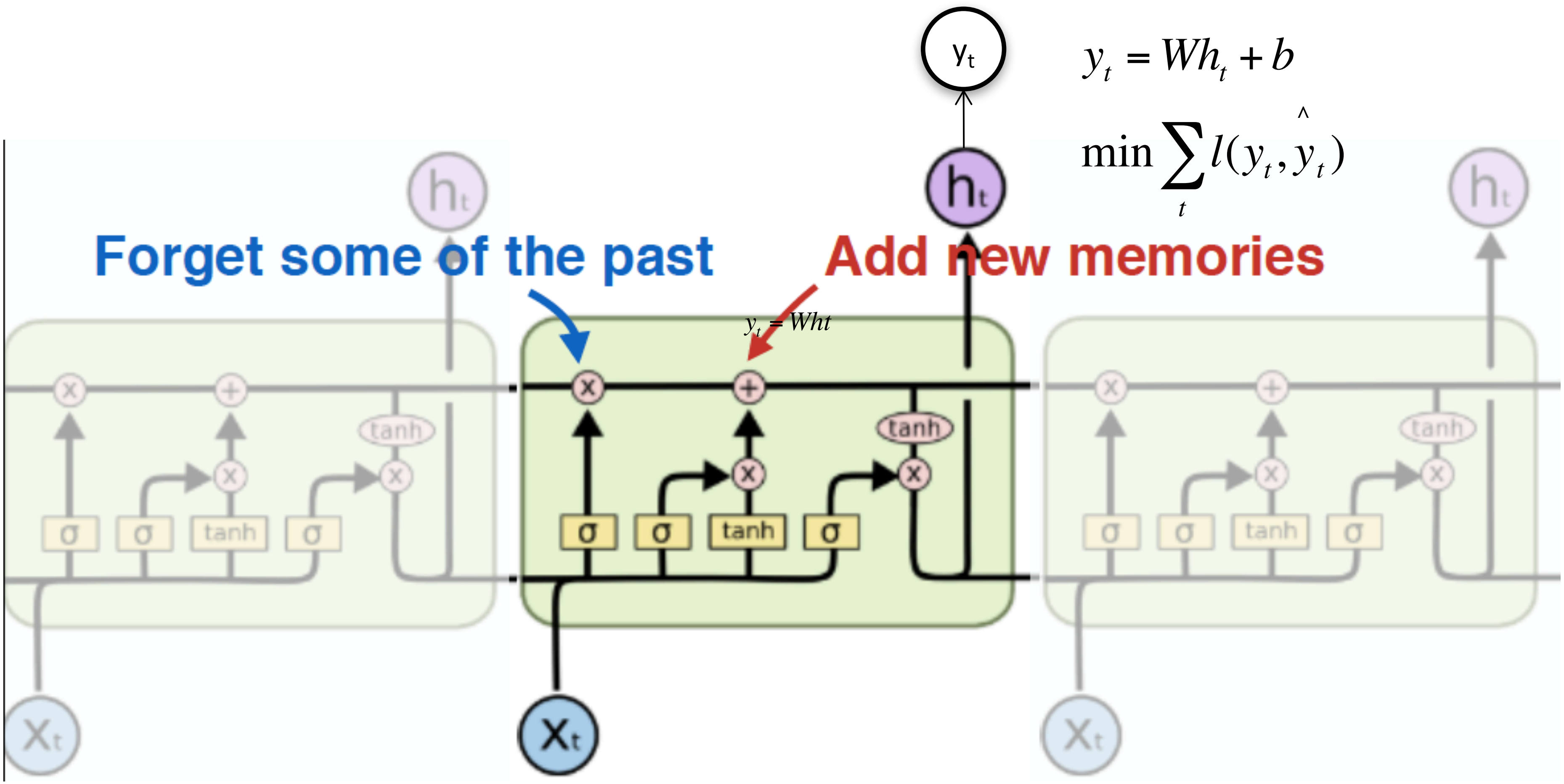


How to deal with sequence output?

- Idea 1: combine DL with CRF

- Idea 2: introduce structure in DL

LSTMs for Sequential Tagging

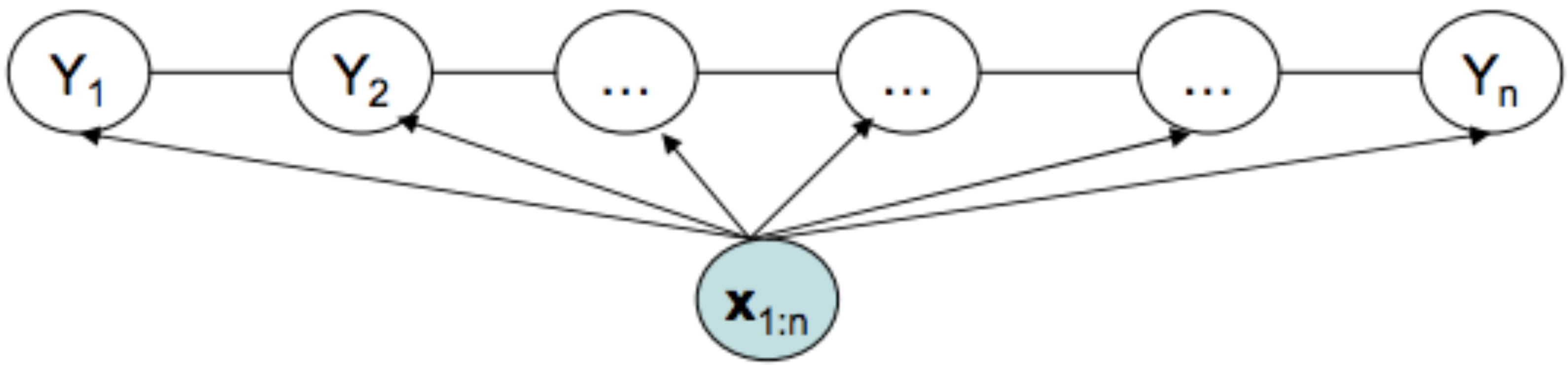


$$y_t = Wh_t + b$$

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

Sophisticated model of input + local predictions.

Recall CRFs for Sequential Tagging



Arbitrary features on the input side

Markov assumption on the output side

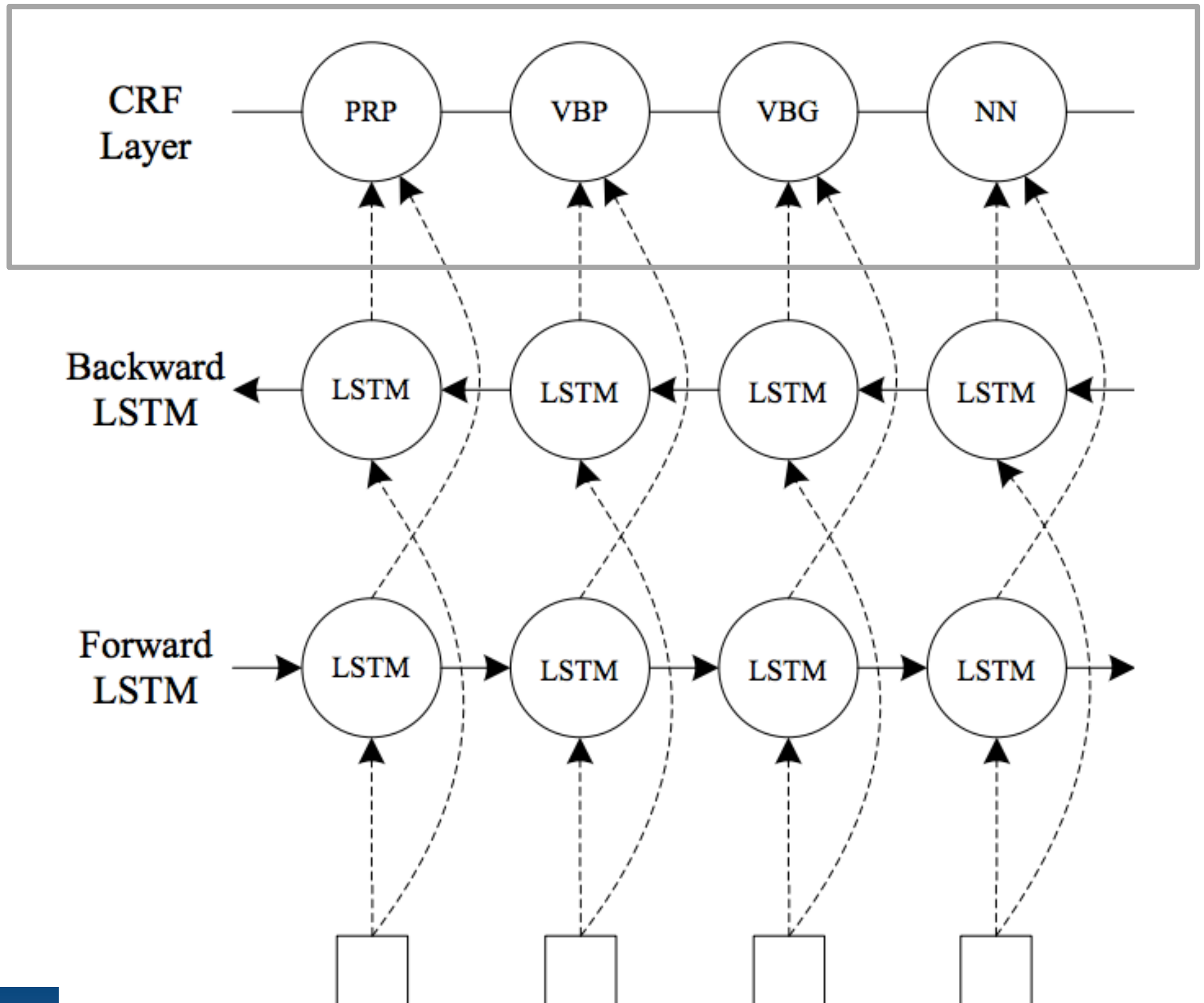
LSTMs for Sequential Tagging

- 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



UCLA ENGINEERING
Computer Science

Traditional CRFs v.s. LSTM-CRFs

- Traditional CRFs:

$$P(Y \mid X; \theta) = \frac{1}{\sum_{Y} \prod_{n=1}^{n} \exp(\theta f(y_i, y_{i-1}, x_{1:n}))} \prod_{n=1}^{n} \exp(\theta f(y_i, y_{i-1}, x_{1:n}))$$

- LSTM-CRFs:

$$P(Y \mid X; \Theta) = \frac{1}{\sum_{Y} \prod_{n=1}^{n} \exp(\lambda f(y_i, y_{i-1}, LSTM(x_{1:n})))} \prod_{n=1}^{n} \exp(\lambda f(y_i, y_{i-1}, LSTM(x_{1:n})))$$

$\Theta = \{\lambda, \Omega\}$ where Ω is LSTM parameters

UCLA ENGINEERING Computer Science

Combining Two Benefits

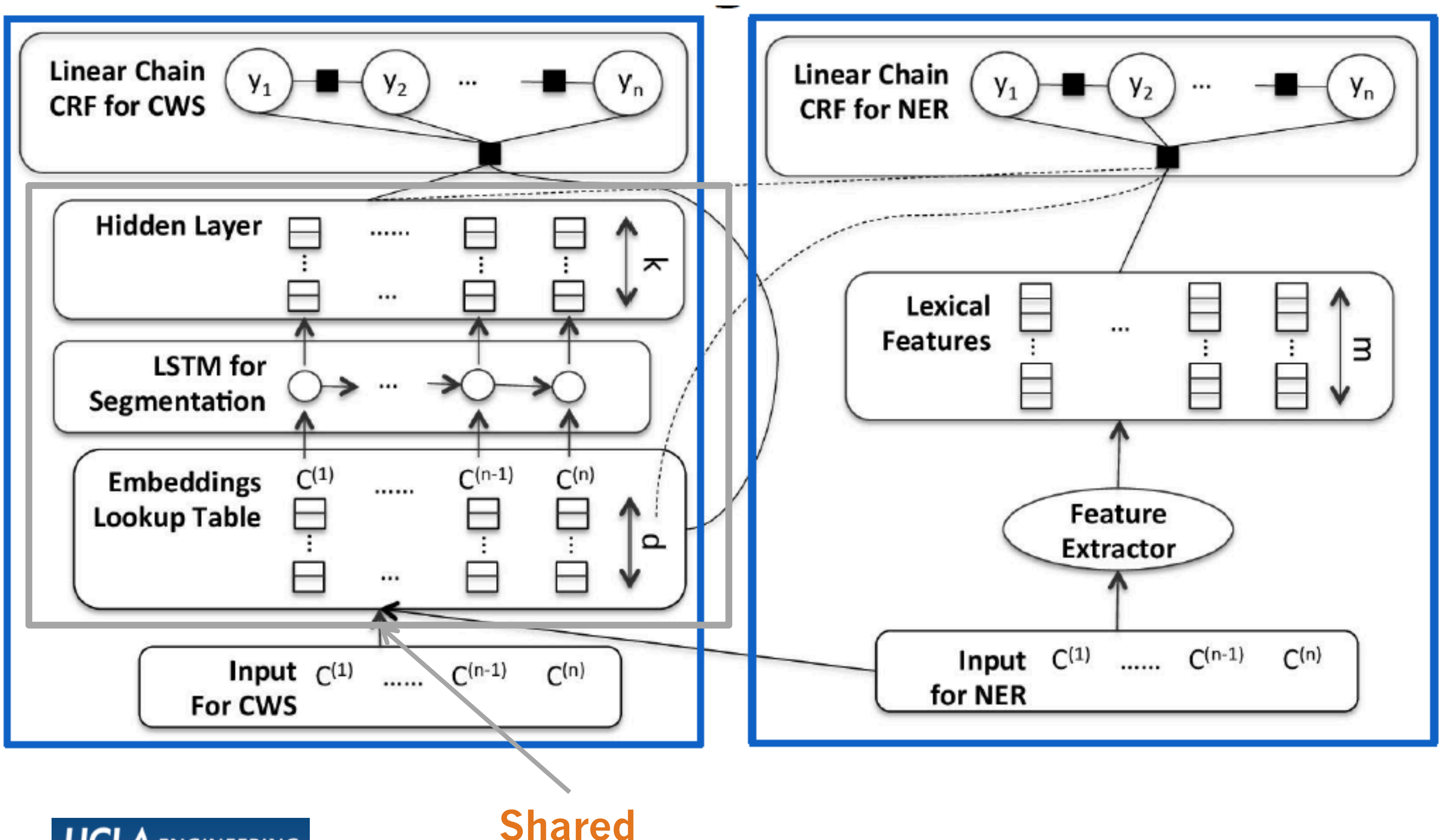
- Directly model output dependencies by CRFs.
- Powerful automatic feature learning using biLSTMs.
- Jointly training all the parameters to “share the modeling responsibilities”

Transfer Learning with LSTM-CRFs

- Neural networks as feature learner
- Share the feature learner for different tasks
- Jointly train the feature learners so that it learns the *common features*.
- Use different CRFs for different tasks to encode task-specific information
  - Going forward, one can imagine using other graphical models besides linear chain CRFs.

UCLA ENGINEERING
Computer Science

Transfer Learning CWS + NER



UCLA ENGINEERING
Computer Science

Joint Training

- Simply linearly combine two objectives.

$$\mathcal{L}_{joint}(\Theta) = \lambda \mathcal{L}_s(\boldsymbol{y}_s; \boldsymbol{x}_s, \Theta) + \mathcal{L}_n(\boldsymbol{y}_n; \boldsymbol{x}_n, \Theta)$$

- Alternative updates for each module's parameters.

How to deal with sequence output?

- Idea 1: combine DL with CRF

- Idea 2: introduce structure in DL

Sequence to Sequence Learning with Neural Networks

NIPS 2014

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT-14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous state of the art. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

Sequence to Sequence Learning with Neural Networks

NIPS 2014

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

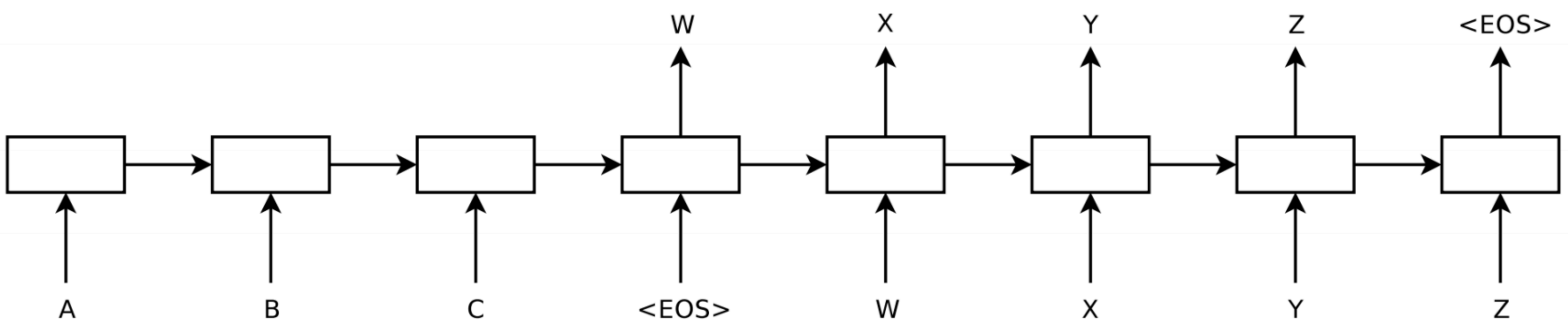


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Neural Machine Translation by Jointly Learning to Align and Translate

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ICLR 2015

Abstract

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ICLR 2015

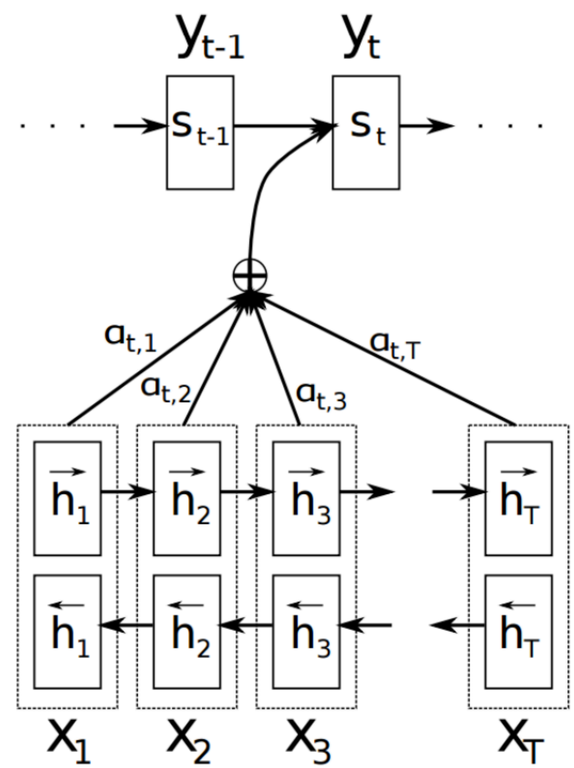
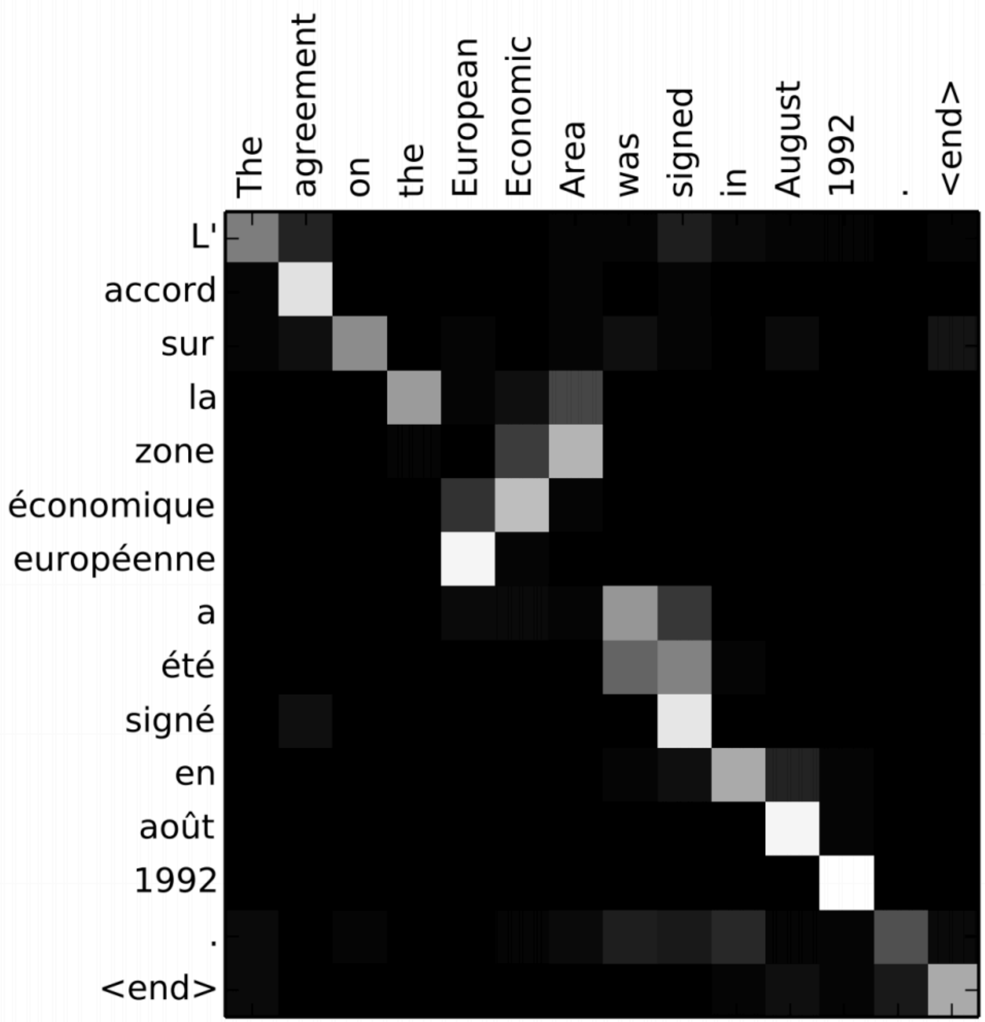


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Neural Machine Translation by Jointly Learning to Align and Translate

ICLR 2015



Save
javas