10-301/601: Introduction to Machine Learning Lecture 17 – Deep Learning

Matt Gormley & Henry Chai

10/28/24

Front Matter

- Announcements
  - HW6 released 10/27, due 11/2 at 11:59 PM
    - **You can only use at most two late days on HW6**
  - Exam 2 on 11/7 (next Thursday) from **6:45 - 8:45 PM**
    - All topics from Lecture 8 to Lecture 16 (inclusive) **+ the portion of today's lecture on MLE/MAP** are in-scope
    - Exam 1 content may be referenced but will not be the primary focus of any question

Recall: Maximum a Posteriori (MAP) Estimation

- Insight: sometimes we have *prior* information we want to incorporate into parameter estimation
- Idea: use Bayes rule to reason about the *posterior* distribution over the parameters
  - MLE finds $\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D}|\theta)$
  - MAP finds $\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\theta|\mathcal{D})$

$$= \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D}|\theta)p(\theta)/p(\mathcal{D})$$

$$= \underset{\theta}{\operatorname{argmax}}\ \underbrace{p(\mathcal{D}|\theta)}_{\text{likelihood}}\underbrace{p(\theta)}_{\text{prior}}$$

$$= \underset{\theta}{\operatorname{argmax}}\ \underbrace{\log p(\mathcal{D}|\theta) + \log p(\theta)}_{\text{log-posterior}}$$

Coin Flipping MAP

- A Bernoulli random variable takes value 1 (or heads) with probability ϕ and value 0 (or tails) with probability $1 - \phi$
- The pmf of the Bernoulli distribution is

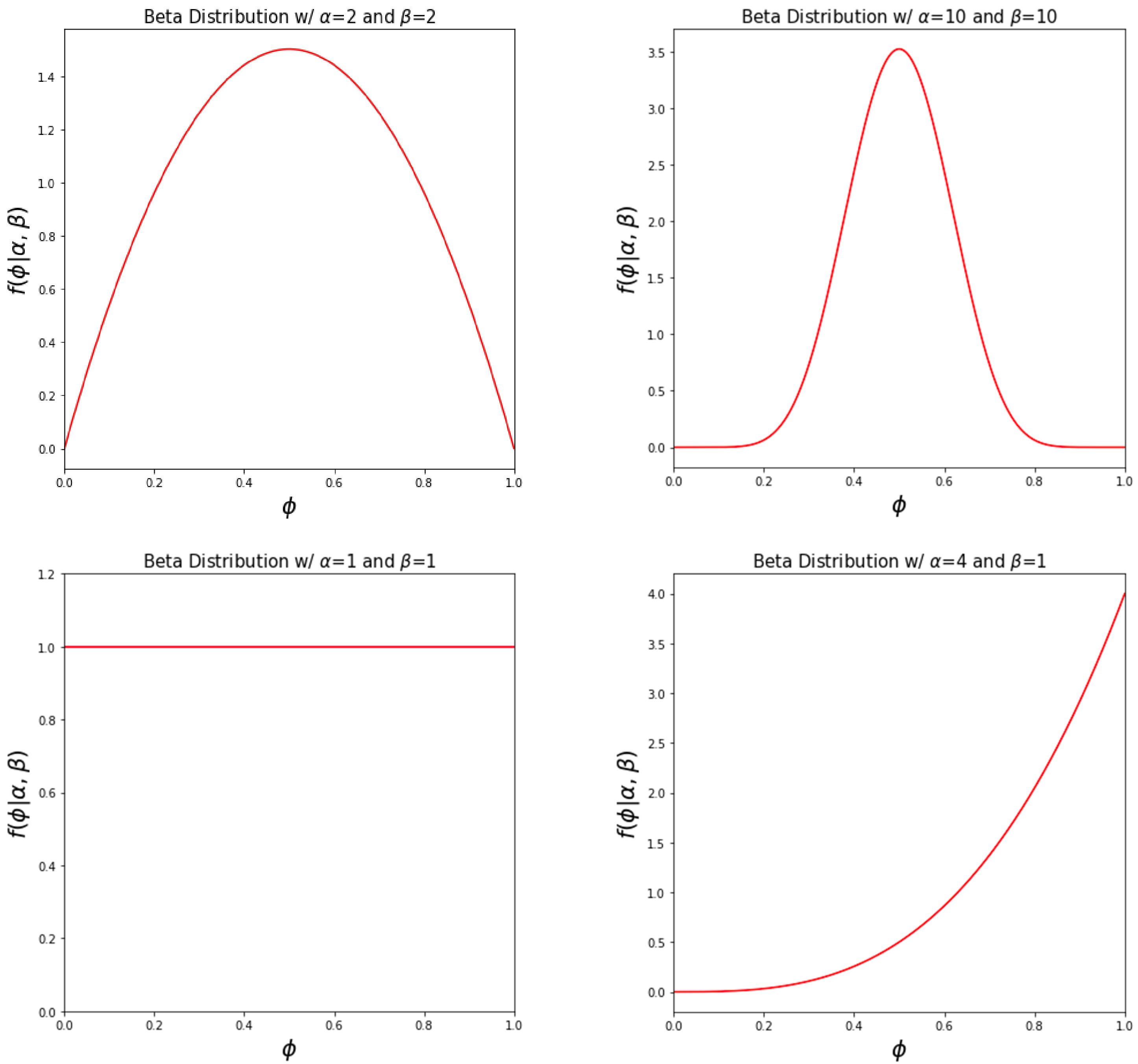
$$p(x|\phi) = \phi^x (1 - \phi)^{1-x}$$

- Assume a Beta prior over the parameter ϕ, which has pdf

$$f(\phi|\alpha, \beta) = \frac{\phi^{\alpha-1}(1 - \phi)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}$$

where $\mathrm{B}(\alpha, \beta) = \int_0^1 \phi^{\alpha-1}(1 - \phi)^{\beta-1} d\phi$ is a normalizing constant to ensure the distribution integrates to 1

Beta Distribution



Why use this strange looking Beta prior?

The Beta distribution is the *conjugate prior* for the Bernoulli distribution!

- A Bernoulli random variable takes value 1 (or heads) with probability ϕ and value 0 (or tails) with probability $1 - \phi$
- The pmf of the Bernoulli distribution is

$$p(x|\phi) = \phi^x (1 - \phi)^{1-x}$$

- Assume a Beta prior over the parameter ϕ, which has pdf

$$f(\phi|\alpha, \beta) = \frac{\phi^{\alpha-1}(1 - \phi)^{\beta-1}}{\mathrm{B}(\alpha, \beta)}$$

where $\mathrm{B}(\alpha, \beta) = \int_0^1 \phi^{\alpha-1}(1 - \phi)^{\beta-1} d\phi$ is a normalizing constant to ensure the distribution integrates to 1

$\log(a^b c^d) = b \log a + d \log c$

Coin Flipping MAP

- Given N iid samples $\{x^{(1)}, \dots, x^{(N)}\}$, the log-posterior is

$$\ell(\phi) = \log f(\phi|\alpha, \beta) + \sum_{n=1}^{N} \log p\left(x^{(n)}|\phi\right)$$

$$= \log \frac{\phi^{\alpha-1}(1-\phi)^{\beta-1}}{B(\alpha,\beta)} + N_1 \log \phi + N_0 \log 1-\phi$$

where N_i = # of i's in my dataset

$$= (\alpha-1)\log\phi + (\beta-1)\log(1-\phi) - \log B(\alpha,\beta)$$
$$+ N_1 \log \phi + N_0 \log(1-\phi)$$

$$= (\alpha-1+N_1)\log\phi + (\beta-1+N_0)\log(1-\phi) - \log B(\alpha,\beta)$$

Coin Flipping MAP

- Given N iid samples $\{x^{(1)}, \ldots, x^{(N)}\}$, the partial derivative of the log-posterior is

$$\frac{\partial \ell}{\partial \phi} = \frac{(\alpha - 1 + N_1)}{\phi} - \frac{(\beta - 1 + N_0)}{1 - \phi}$$

$$\vdots$$

$$\rightarrow \hat{\phi}_{MAP} = \frac{(\alpha - 1 + N_1)}{(\beta - 1 + N_0) + (\alpha - 1 + N_1)}$$

- $\alpha - 1$ is a “pseudocount” of the number of 1’s (or heads) you’ve “observed”
- $\beta - 1$ is a “pseudocount” of the number of 0’s (or tails) you’ve “observed”

Coin Flipping MAP: Example

- Suppose $\mathcal{D}$ consists of ten 1's or heads ($N_1 = 10$) and two 0's or tails ($N_0 = 2$):

$$\phi_{MLE} = \frac{10}{10+2} = \frac{10}{12}$$

- Using a Beta prior with $\alpha = 101$ and $\beta = 101$, then

$$\phi_{MAP} = \frac{(101-1+10)}{(101-1+2)+(101-1+10)} = \frac{110}{212} \approx \frac{1}{2}$$

Coin Flipping MAP: Example

- Suppose $\mathcal{D}$ consists of ten 1's or heads ($N_1 = 10$) and two 0's or tails ($N_0 = 2$):

$$\phi_{MLE} = \frac{10}{10+2} = \frac{10}{12}$$

- Using a Beta prior with $\alpha = 1$ and $\beta = 1$, then

$$\phi_{MAP} = \frac{(1-1+10)}{(1-1+2)+(1-1+10)} = \frac{10}{12} = \phi_{MLE}$$

MLE/MAP Learning Objectives

You should be able to…

- Recall probability basics, including but not limited to: discrete and continuous random variables, probability mass functions, probability density functions, events vs. random variables, expectation and variance, joint probability distributions, marginal probabilities, conditional probabilities, independence, conditional independence
- State the principle of maximum likelihood estimation and explain what it tries to accomplish
- State the principle of maximum a posteriori estimation and explain why we use it
- Derive the MLE or MAP parameters of a simple model in closed form

Deep Learning

- From Wikipedia's page on Deep Learning…

Definition [edit]

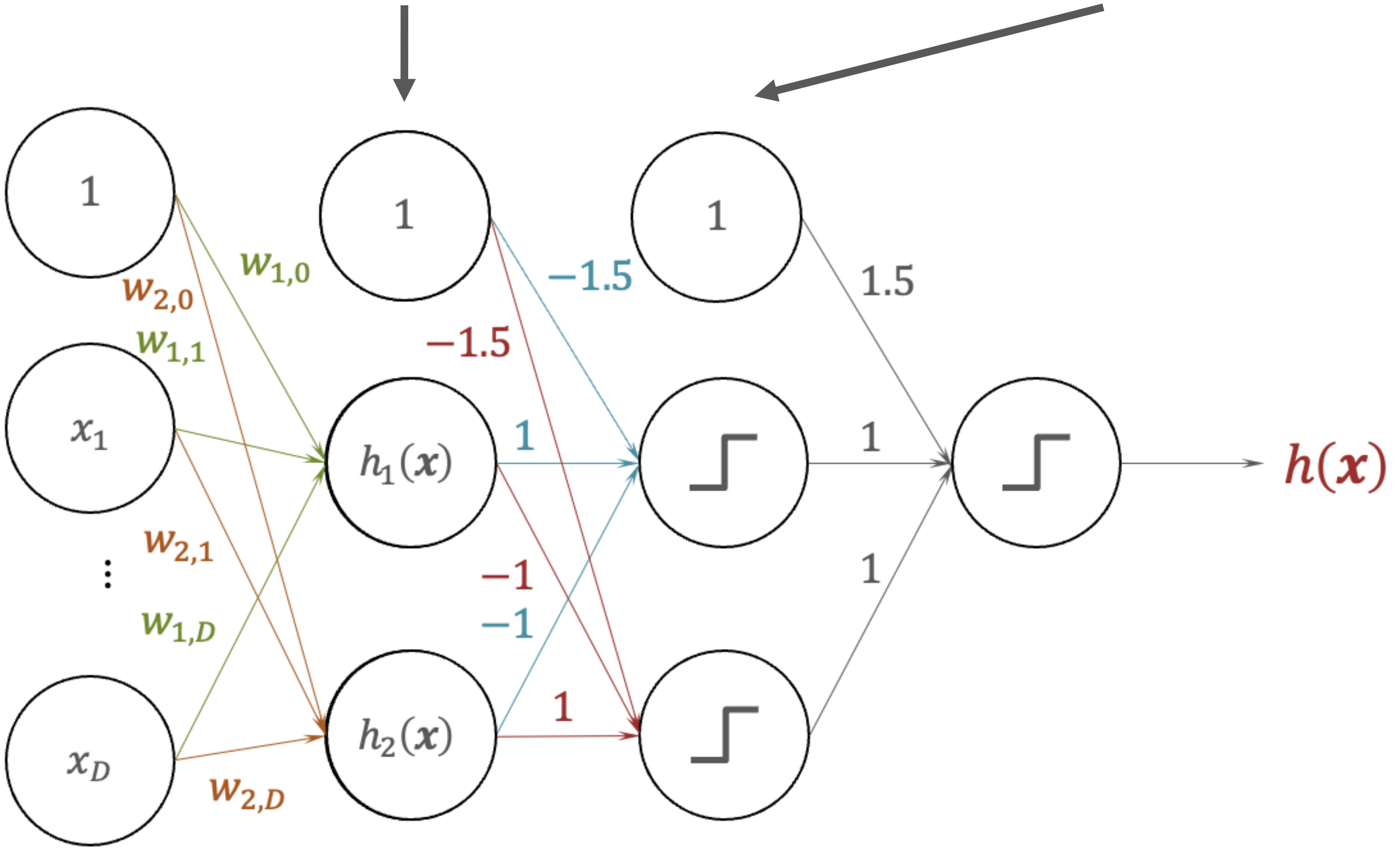
Deep learning is a class of machine learning algorithms that[11](pp199–200) uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

Source: https://en.wikipedia.org/wiki/Deep_learning

Deep Learning

First layer: computes the perceptrons' predictions

Second layer: combines lower-level components



Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Idea: use the first few layers to identify relevant macro-features, e.g., edges
- Insight: for spatially-structured inputs, many useful macro-features are shift or location-invariant, e.g., an edge in the upper left corner of a picture looks like an edge in the center
- Strategy: learn a filter for macro-feature detection in a small window and apply it over the entire image

Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

*

0	1	0
1	-4	1
0	1	0

Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1	0
1	-4	1
0	1	0

$=$

0			

$$(0*0)+(0*1)+(0*0)+(0*1)+(1*-4)+(2*1)+(0*0)+(2*1)+(4*0)=0$$

Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1	0
1	-4	1
0	1	0

$=$

0	-1		

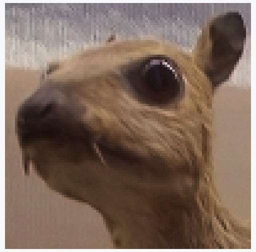
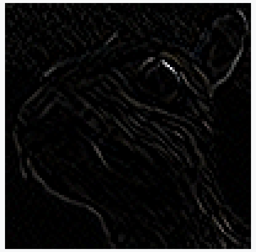
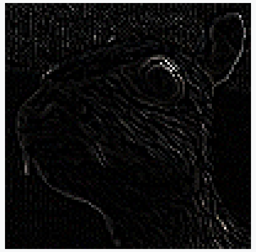
$$(0*0)+(0*1)+(0*0)+(1*1)+(2*-4)$$
$$+(2*1)+(2*0)+(4*1)+(4*0)=-1$$

Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 & 0 \\ -2 & -5 & -5 & -2 \\ 2 & -2 & -1 & 3 \\ -1 & 0 & -5 & 0 \end{bmatrix}$$

Convolutional Filters

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Poll Question 1:

What effect do you think the following filter will have on an image?

A. Sharpen the image

B. Blur the image

C. Shift the image left

D. Rotate the image clockwise

E. Nothing (TOXIC)

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

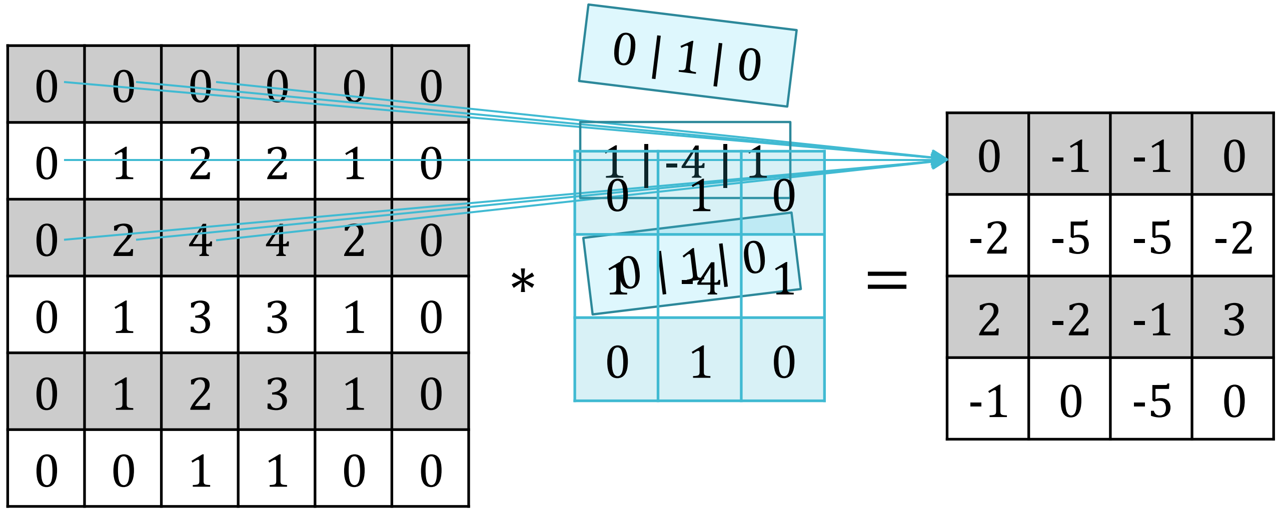
More Filters

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix



Convolutional Filters

- Convolutions can be represented by a feed forward neural network where:
  1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
  2. Many of the weights have the same value.

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

- Many fewer weights than a fully connected layer!
- Convolution weights are learned using gradient descent/ backpropagation, not prespecified

Convolutional Filters: Padding

- What if relevant features exist at the border of our image?
- Add zeros around the image to allow for the filter to be applied "everywhere" e.g. a *padding* of 1 with a 3x3 filter preserves image size and allows every pixel to be the center

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	2	2	1	0	0
0	0	2	4	4	2	0	0
0	0	1	3	3	1	0	0
0	0	1	2	3	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

$*$

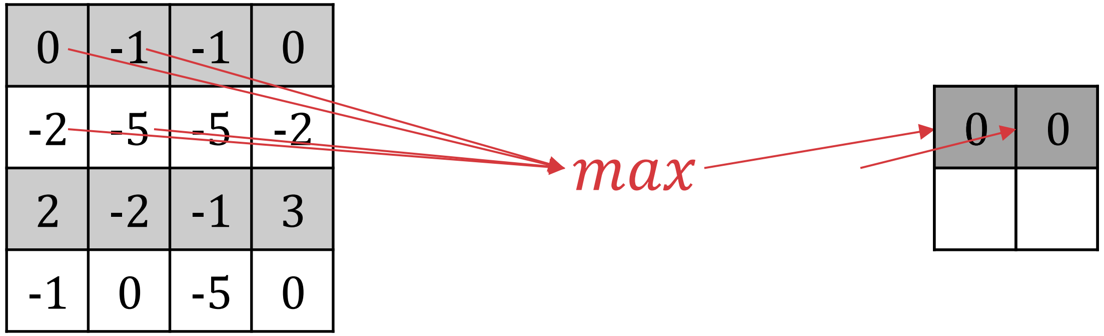
0	1	0
1	-4	1
0	1	0

$=$

0	1	2	2	1	0
1	0	-1	-1	0	1
2	-2	-5	-5	-2	2
1	2	-2	-1	3	1
1	-1	0	-5	0	1
0	2	-1	0	2	0

Downsampling: Pooling

- Combine multiple adjacent nodes into a single node



Downsampling: Pooling

- Combine multiple adjacent nodes into a single node

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

max pooling

0	0
2	3

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
  - Protects the network from (slightly) noisy inputs

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1
1	-2

$=$

-2		

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1
1	-2

$=$

-2	-2	

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1
1	-2

$=$

-2	-2	1

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

$*$

0	1
1	-2

$=$

-2	-2	1
0		

Downsampling: Stride

- Only apply the convolution to some subset of the image e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

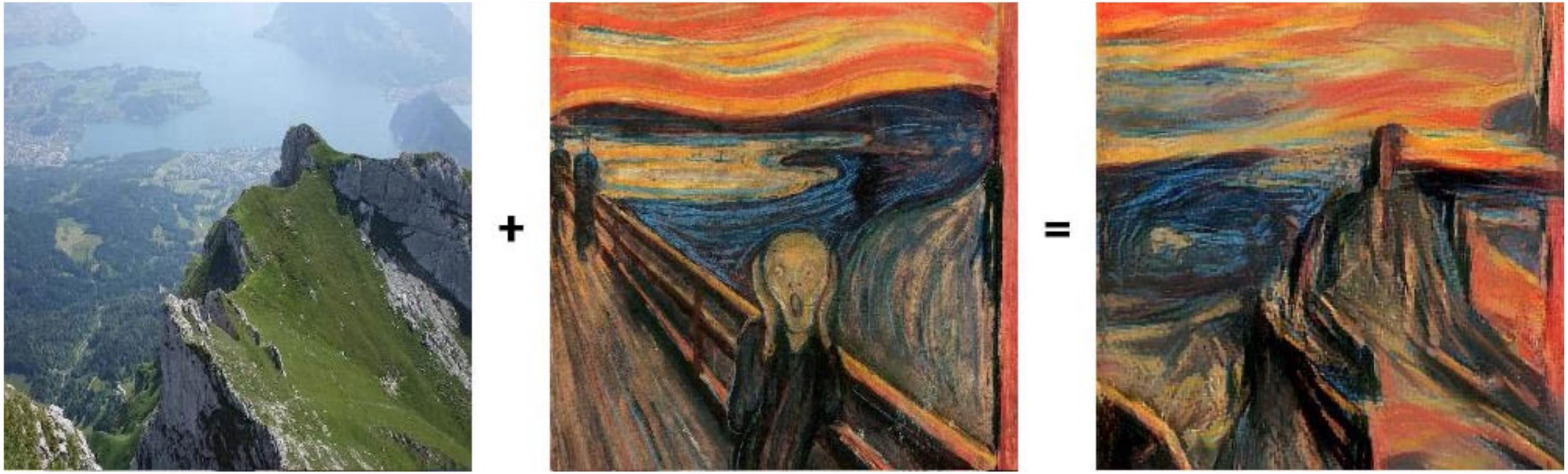
$*$

0	1
1	-2

$=$

-2	-2	1
0	1	1
1	2	0

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
- Many relevant macro-features will tend to span large portions of the image, so taking strides with the convolution tends not to miss out on too much



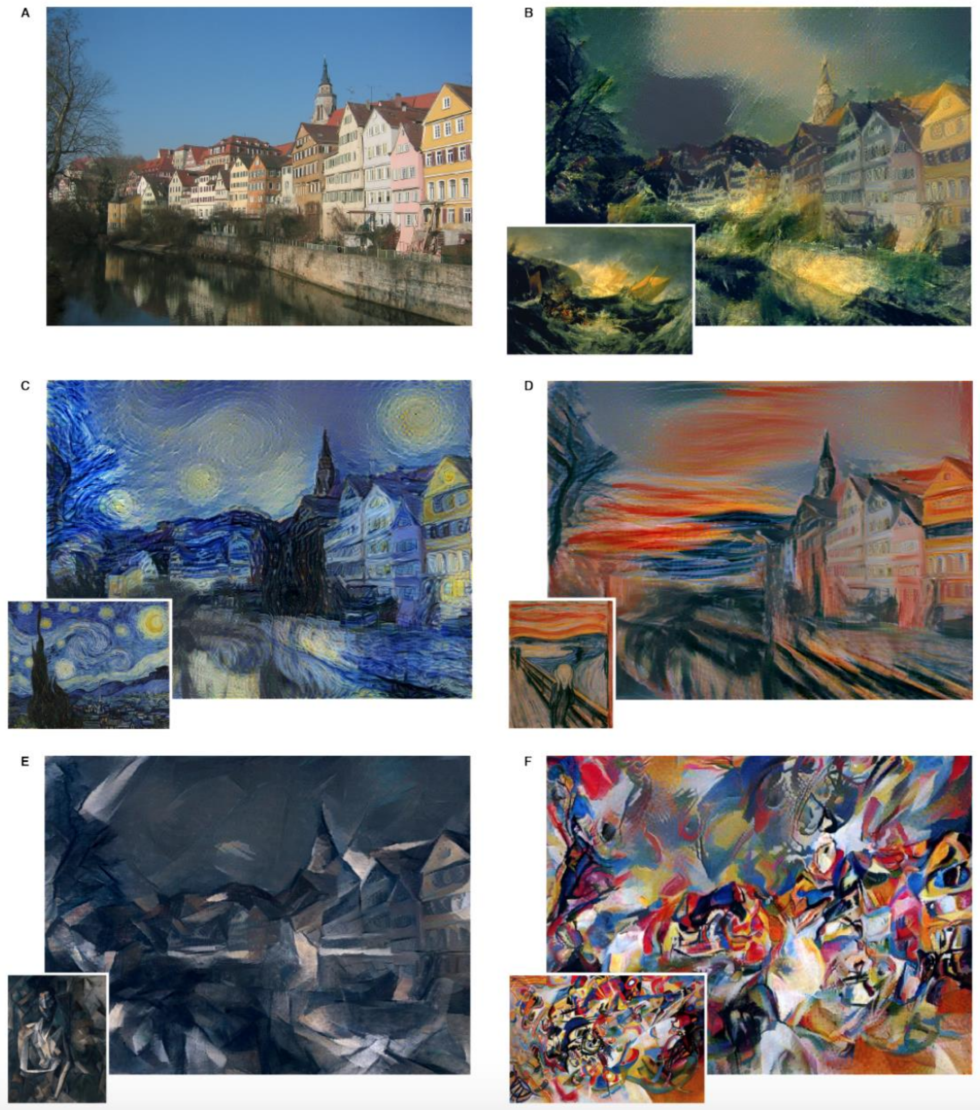
Cool Example: Style Transfer

Source: https://medium.com/data-science-group-iitr/artistic-style-transfer-with-convolutional-neural-network-7ce2476039fd

Style Transfer

- Basic idea:
  - Learn a content representation for an image using convolutional layers
  - Learn a style representation for an image using convolutional layers
  - Compute an image that jointly minimizes the distance from the content image's content representation and the style image's style representation
  - For complete details, see https://arxiv.org/pdf/1508.06576.pdf

Cool Example: Style Transfer



Source: https://arxiv.org/pdf/1508.06576.pdf

Example: Handwriting Recognition

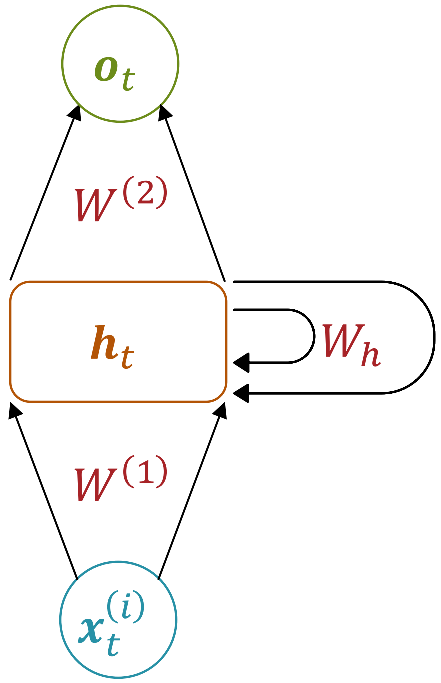
Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

Recurrent Neural Networks

- Neural networks are frequently applied to inputs with some inherent temporal or sequential structure, e.g., text or words
- Idea: use the information from previous parts of the input to inform subsequent predictions
- Insight: the hidden layers learn a useful representation (relative to the task)
- Strategy: incorporate the output from earlier hidden layers into later ones.

Recurrent Neural Networks

$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h \boldsymbol{h}_{t-1}\right)\right]^T$ and $\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta(W^{(2)}\boldsymbol{h}_t)$



- Training dataset consists of (input **sequence**, label **sequence**) pairs, potentially of varying lengths

$$\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\right)\right\}_{n=1}^{N}$$

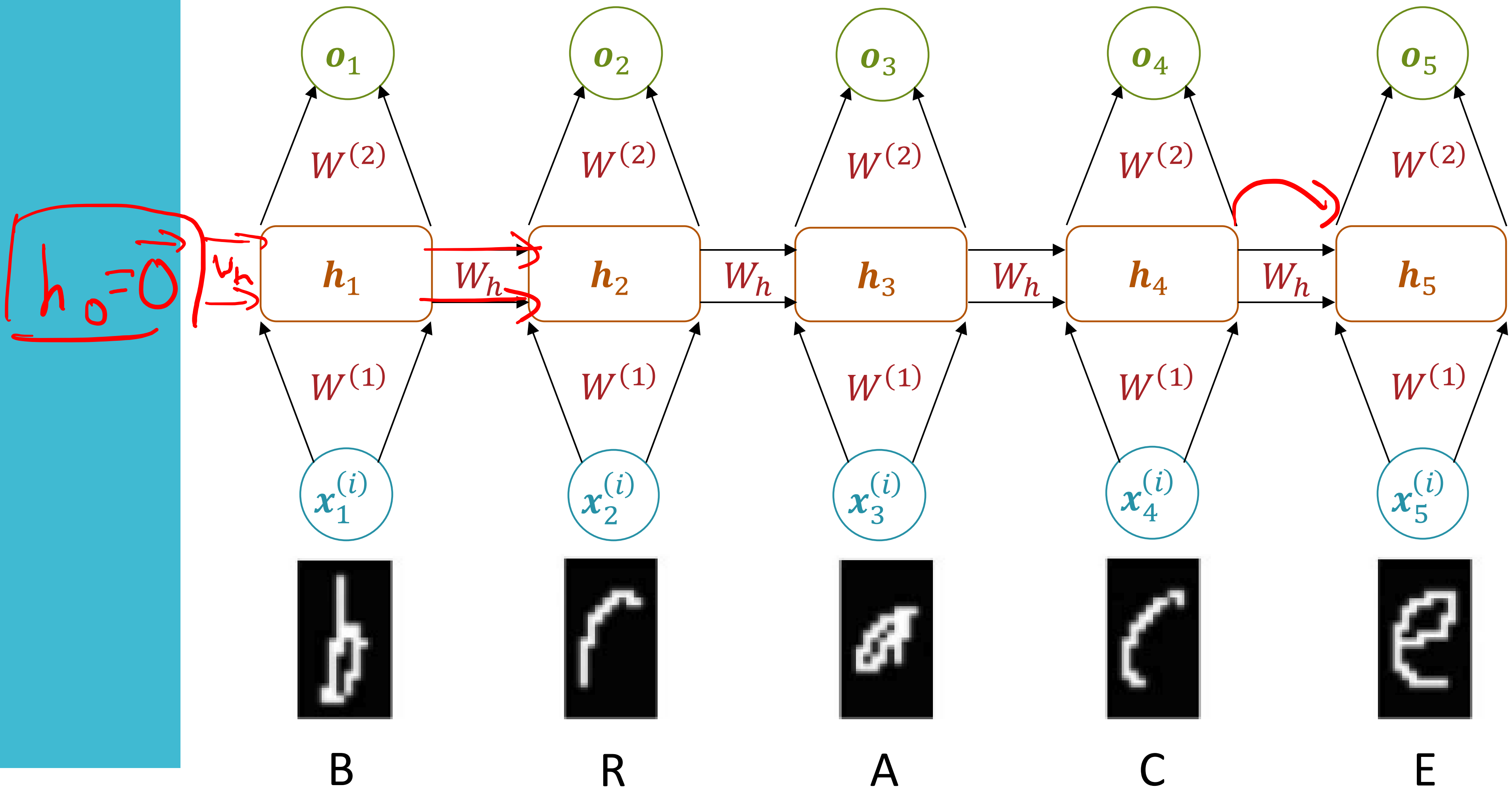
$$\boldsymbol{x}^{(n)} = \left[\boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_{T_n}^{(n)}\right]$$

$$\boldsymbol{y}^{(n)} = \left[\boldsymbol{y}_1^{(n)}, \dots, \boldsymbol{y}_{T_n}^{(n)}\right]$$

- This model requires an initial value for the hidden representation, $\boldsymbol{h}_0$, typically a vector of all zeros

Unrolling Recurrent Neural Networks

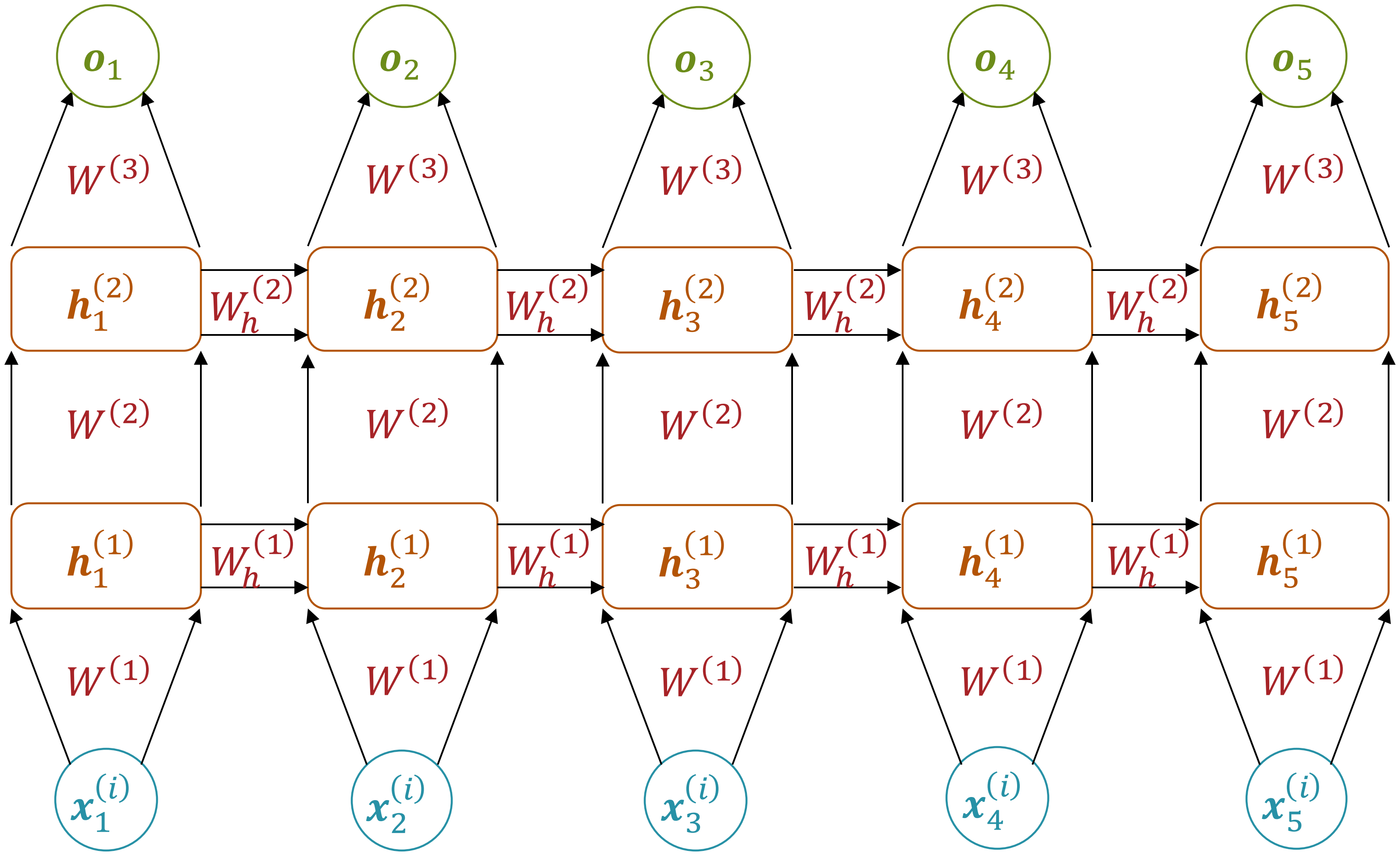
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h \boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

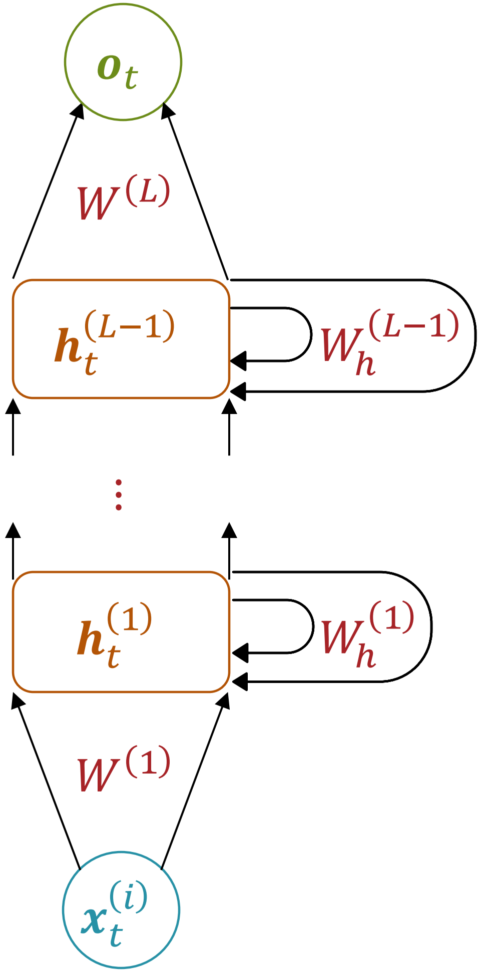
Deep Recurrent Neural Networks

$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



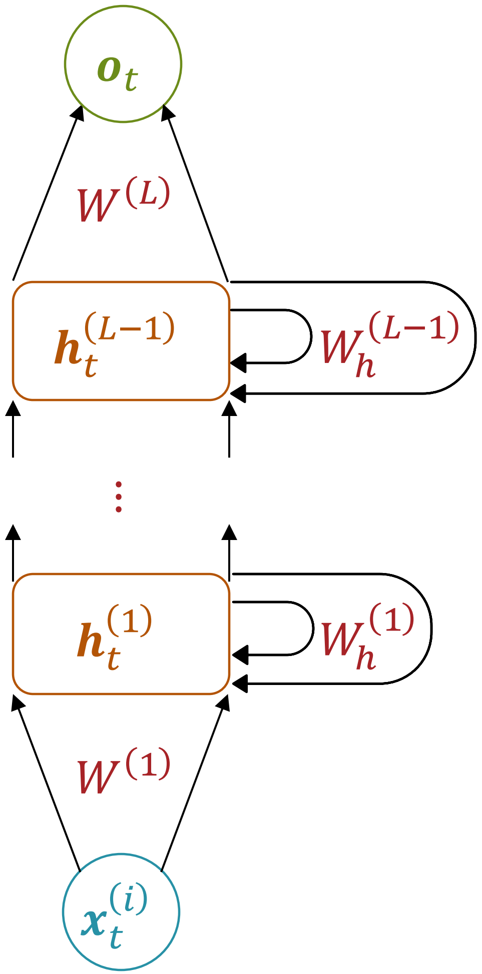
Deep Recurrent Neural Networks

$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



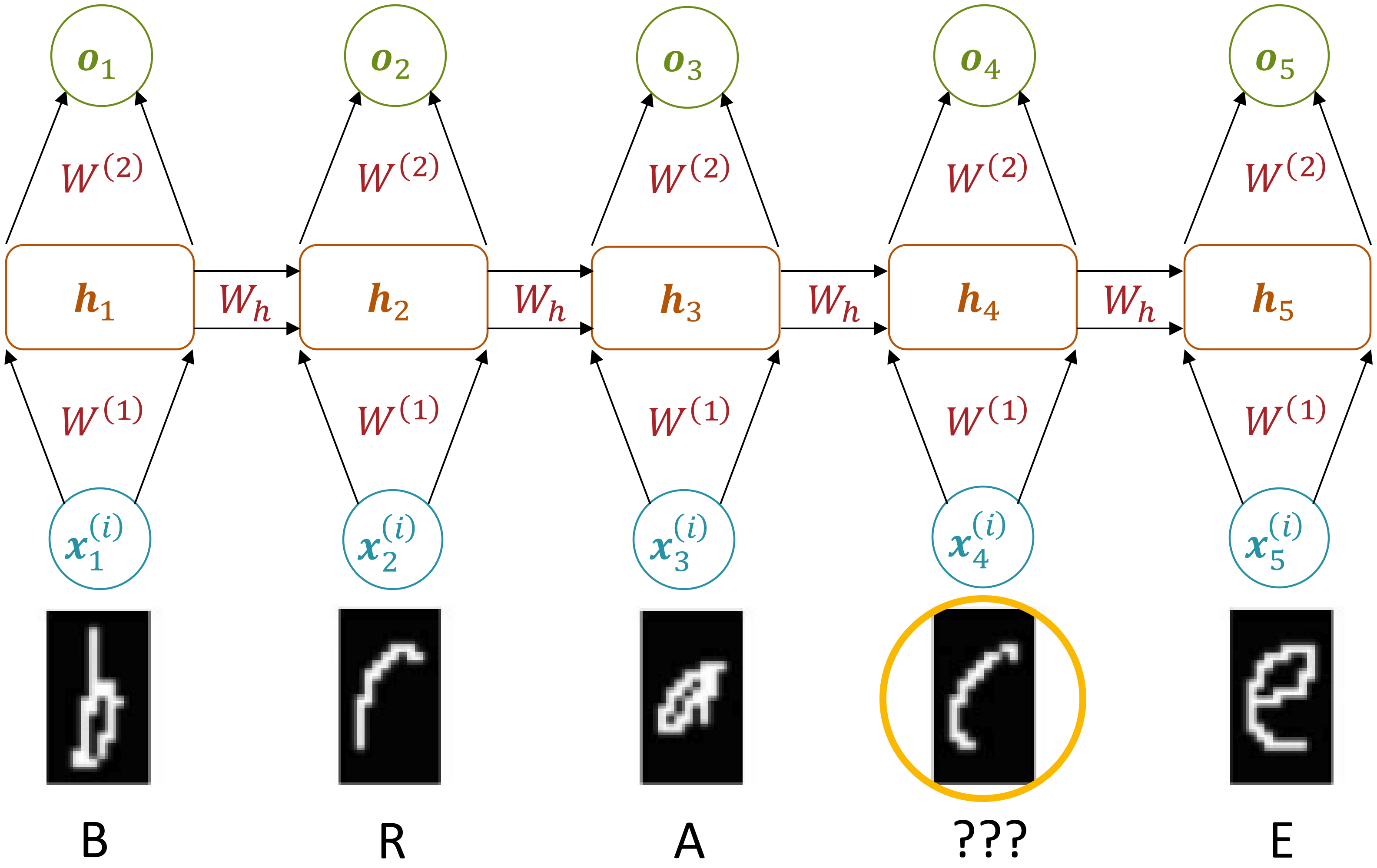
But why do we only pass information forward? What if later time steps have useful information as well?

$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



But why do we only pass information forward? What if later time steps have useful information as well?

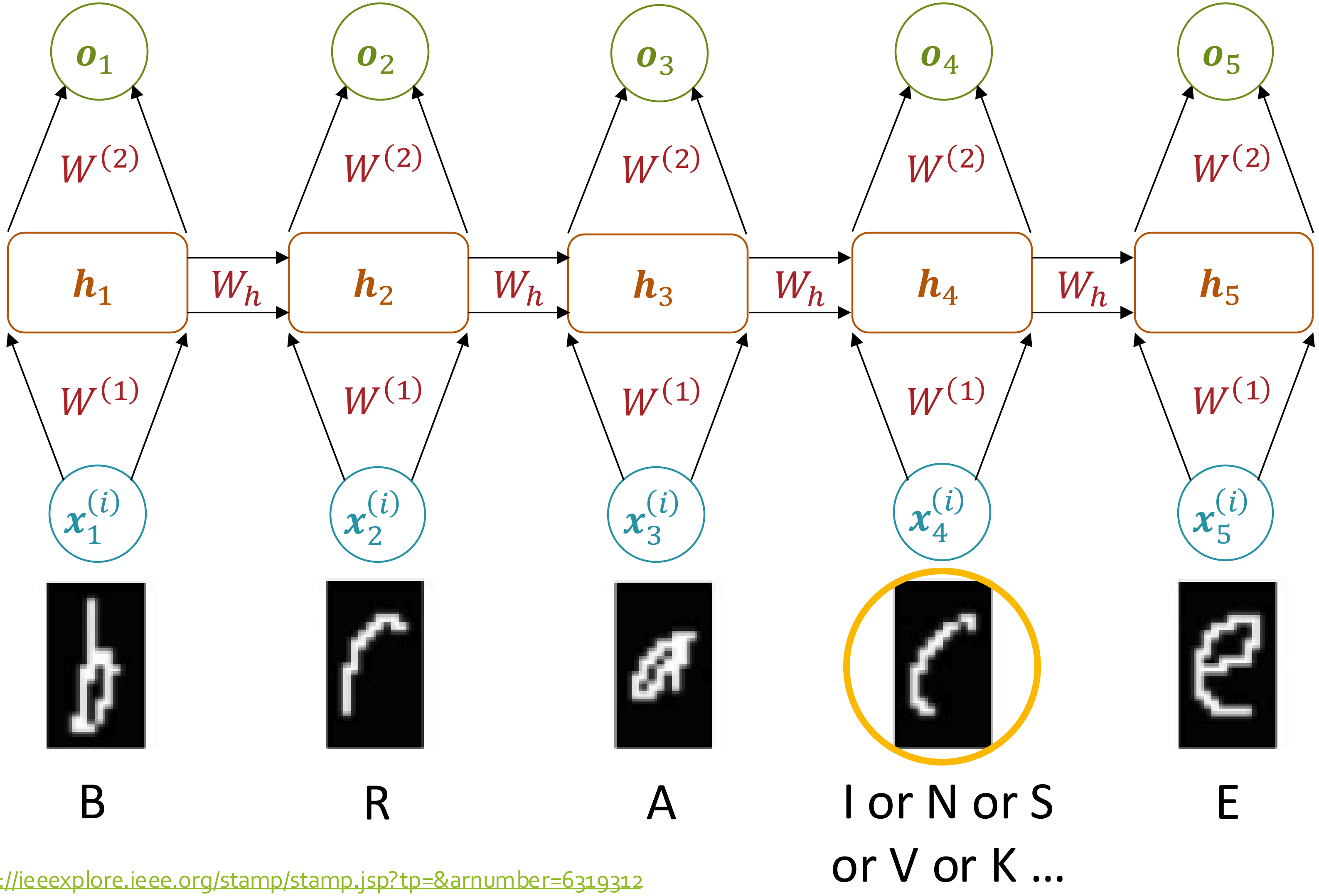
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

But why do we only pass information forward? What if later time steps have useful information as well?

$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$

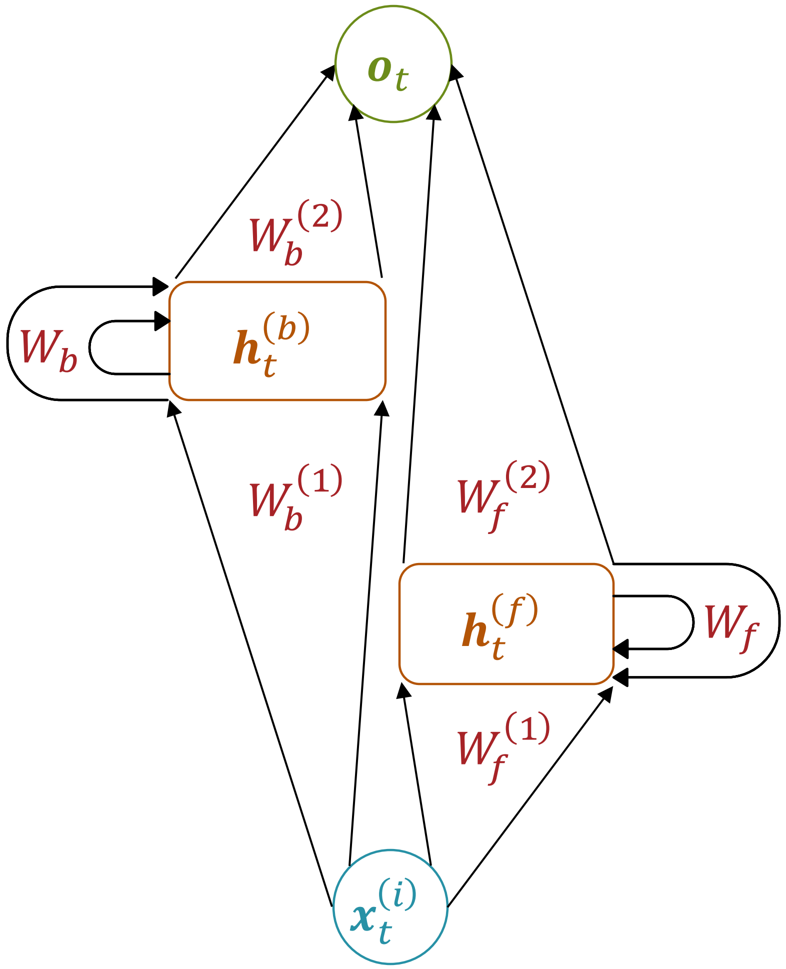


Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

Bidirectional Recurrent Neural Networks

$$\boldsymbol{h}_t^{(f)} = \left[1, \theta\left(W_f^{(1)} \boldsymbol{x}_t^{(i)} + W_f \boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{h}_t^{(b)} = \left[1, \theta\left(W_b^{(1)} \boldsymbol{x}_t^{(i)} + W_b \boldsymbol{h}_{t+1}\right)\right]^T$$

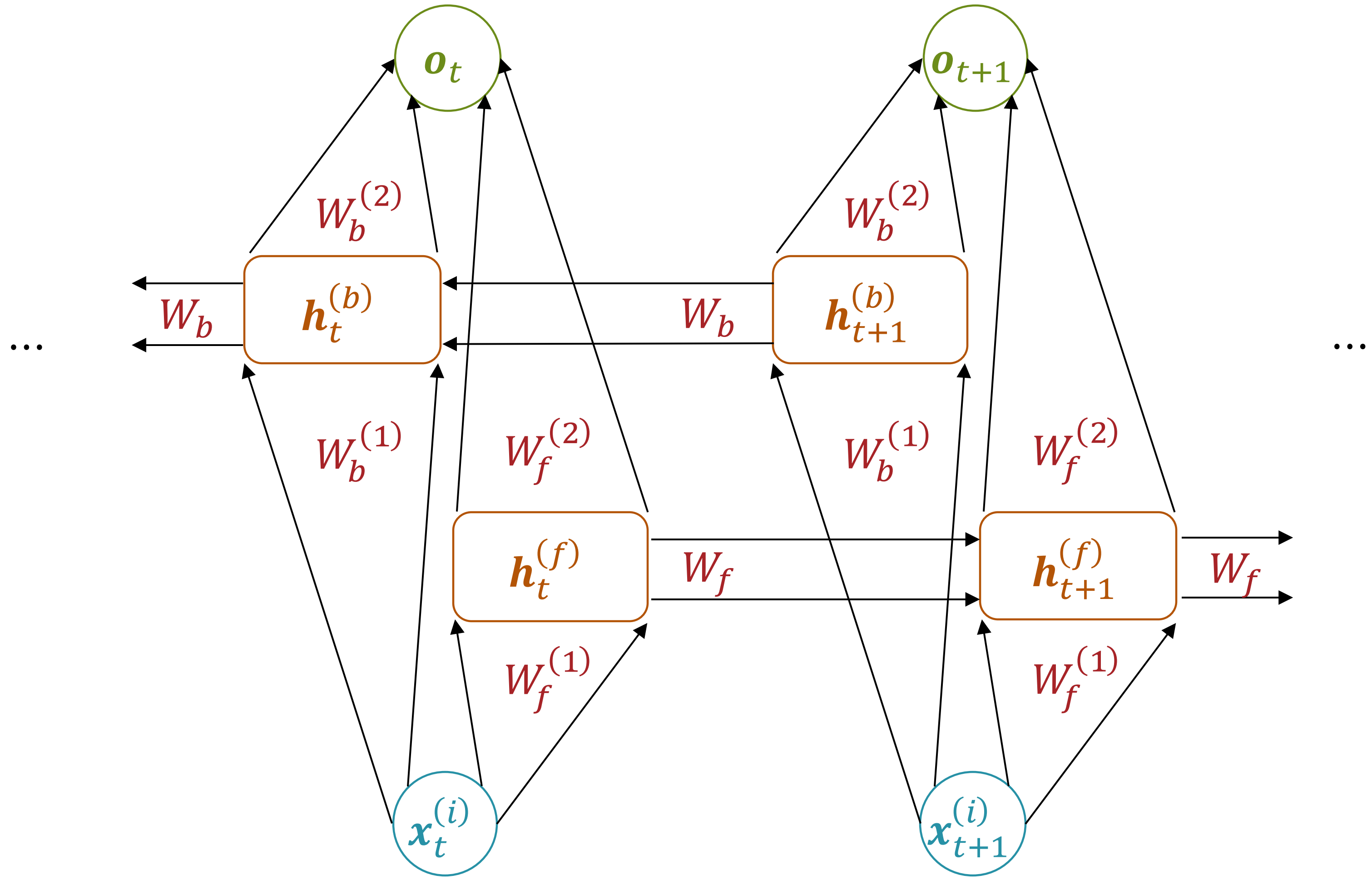
$$\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W_f^{(2)} \boldsymbol{h}_t^{(f)} + W_b^{(2)} \boldsymbol{h}_t^{(b)}\right)$$



$$\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W_f^{(2)}\boldsymbol{h}_t^{(f)} + W_b^{(2)}\boldsymbol{h}_t^{(b)}\right)$$

Unrolling Bidirectional Recurrent Neural Networks

$$\boldsymbol{h}_t^{(f)} = \left[1, \theta\left(W_f^{(1)}\boldsymbol{x}_t^{(i)} + W_f\boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{h}_t^{(b)} = \left[1, \theta\left(W_b^{(1)}\boldsymbol{x}_t^{(i)} + W_b\boldsymbol{h}_{t+1}\right)\right]^T$$



Training RNNs

- A (deep/bidirectional) RNN simply represents a (somewhat complicated) computation graph
  - Weights are shared between different timesteps, significantly reducing the number of parameters to be learned!
- Can be trained using (stochastic) gradient descent/ backpropagation → “backpropagation through time”