10-701: Introduction to Machine Learning Lecture 12 – RNNs

Henry Chai

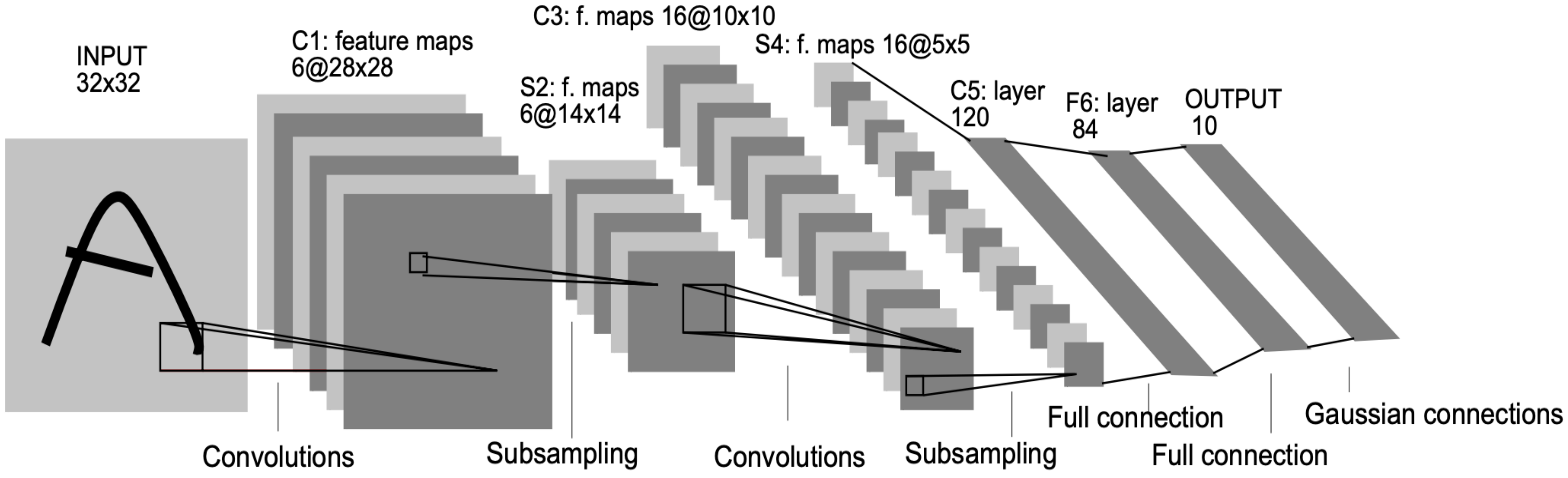
2/26/24

Front Matter

- Announcements
  - HW3 released 2/19, due 2/28 (Wednesday) at 11:59 PM
  - HW4 released 2/28 (Wednesday), due 3/15 (after break) at 11:59 PM
  - Project details will be released 3/1 (Friday)
    - **You must work in groups of 2 or 3 on the project**
- Recommended Readings
  - Zhang, Lipton, Li & Smola, Chapters 9 & 10

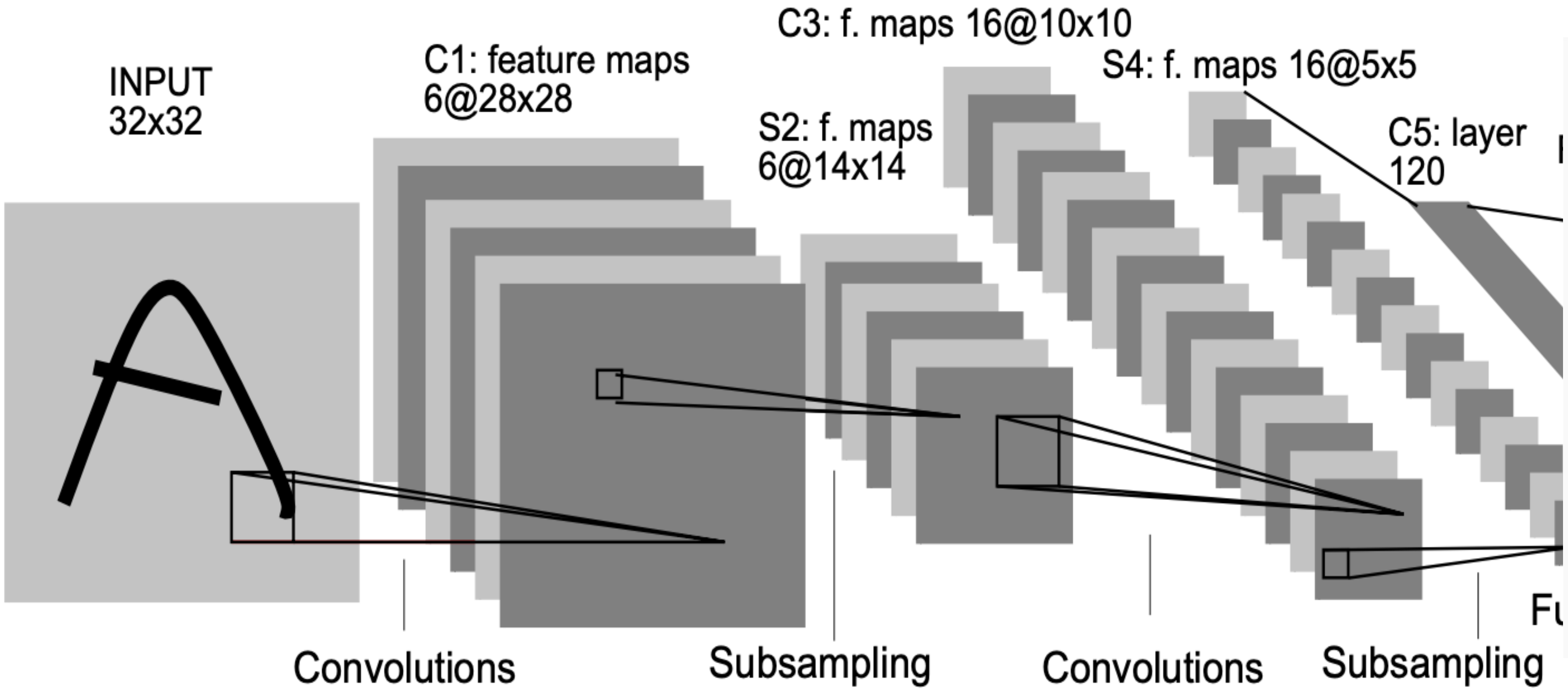
Convolutional Neural Networks

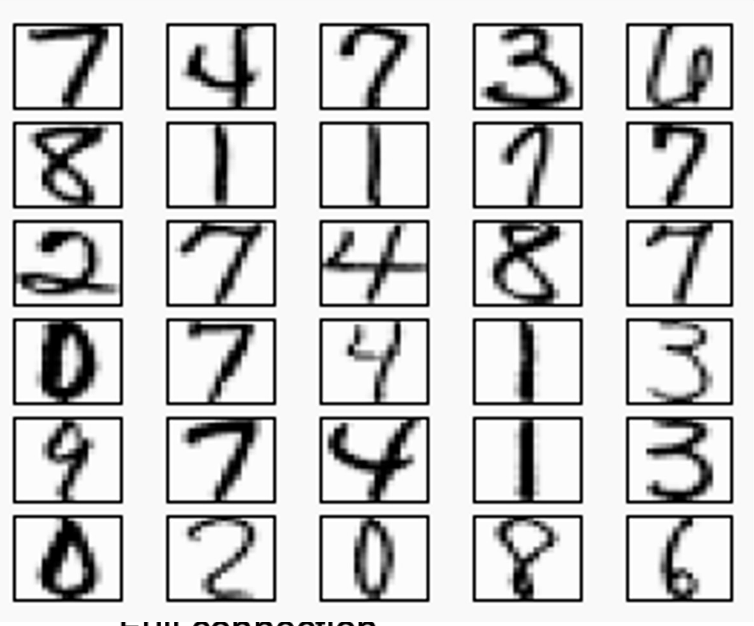
- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Idea: use the first few layers to identify relevant macro-features, e.g., edges
- Insight: for spatially-structured inputs, many useful macro-features are shift or location-invariant, e.g., an edge 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



LeNet (LeCun et al., 1998)

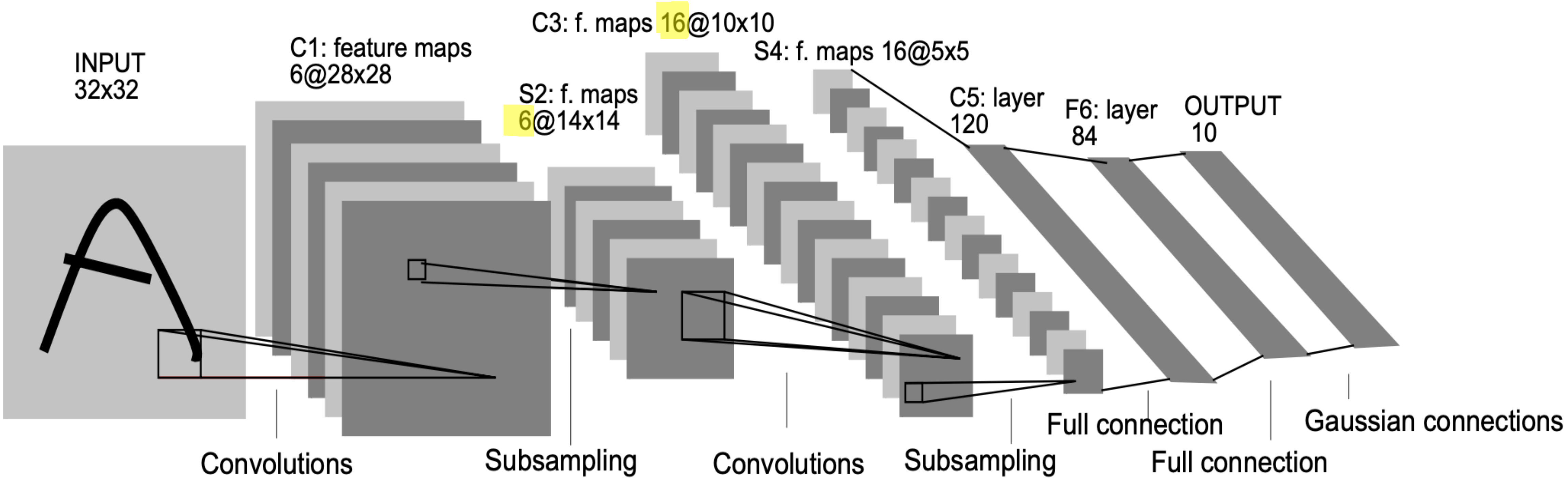
Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf





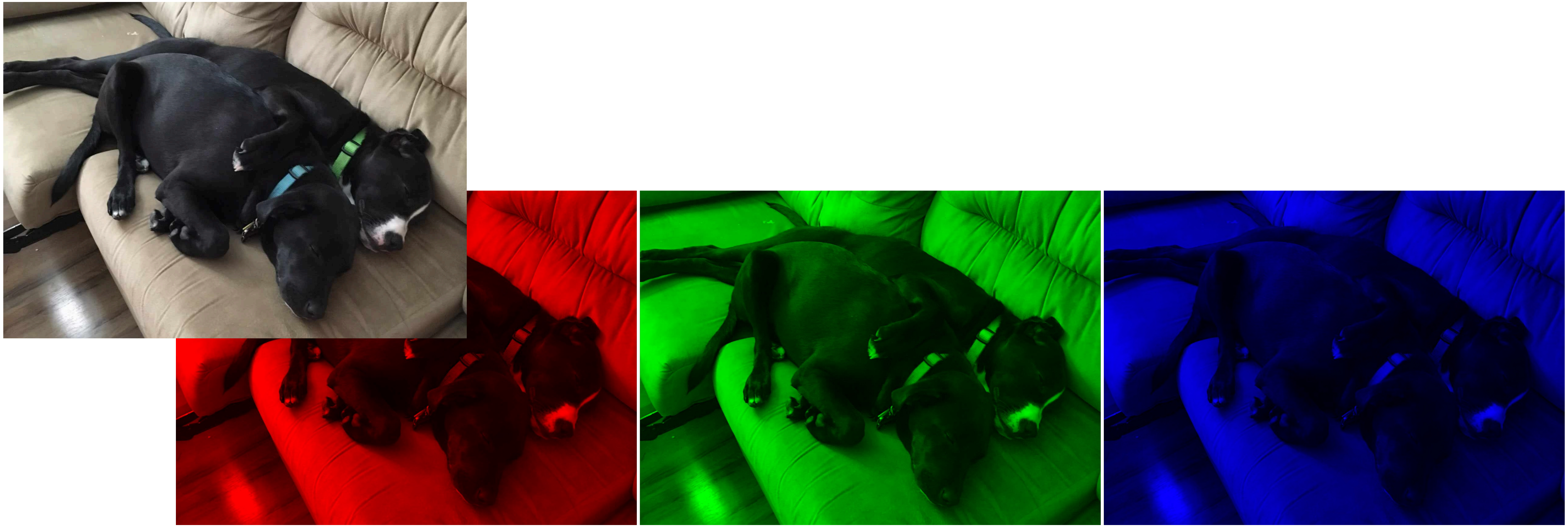
- One of the earliest, most famous deep learning models – achieved remarkable performance at handwritten digit recognition (< 1% test error rate on MNIST)
- Used sigmoid (or logistic) activation functions between layers and mean-pooling, both of which are pretty uncommon in modern architectures

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

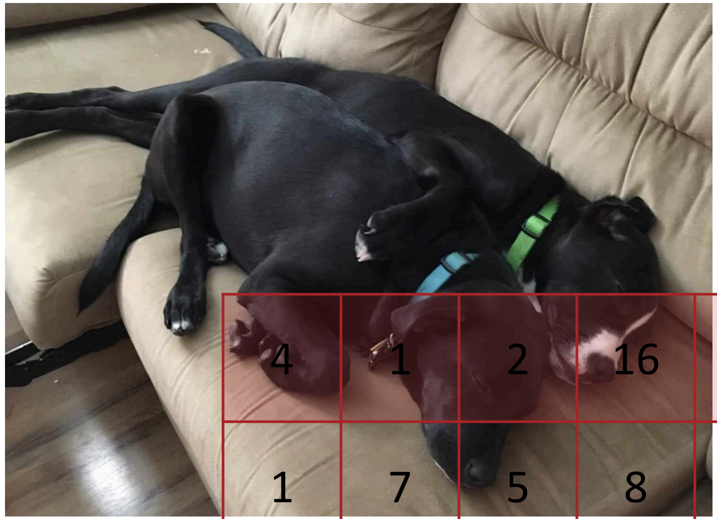


Wait how did we go from 6 to 16?

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf



Channels



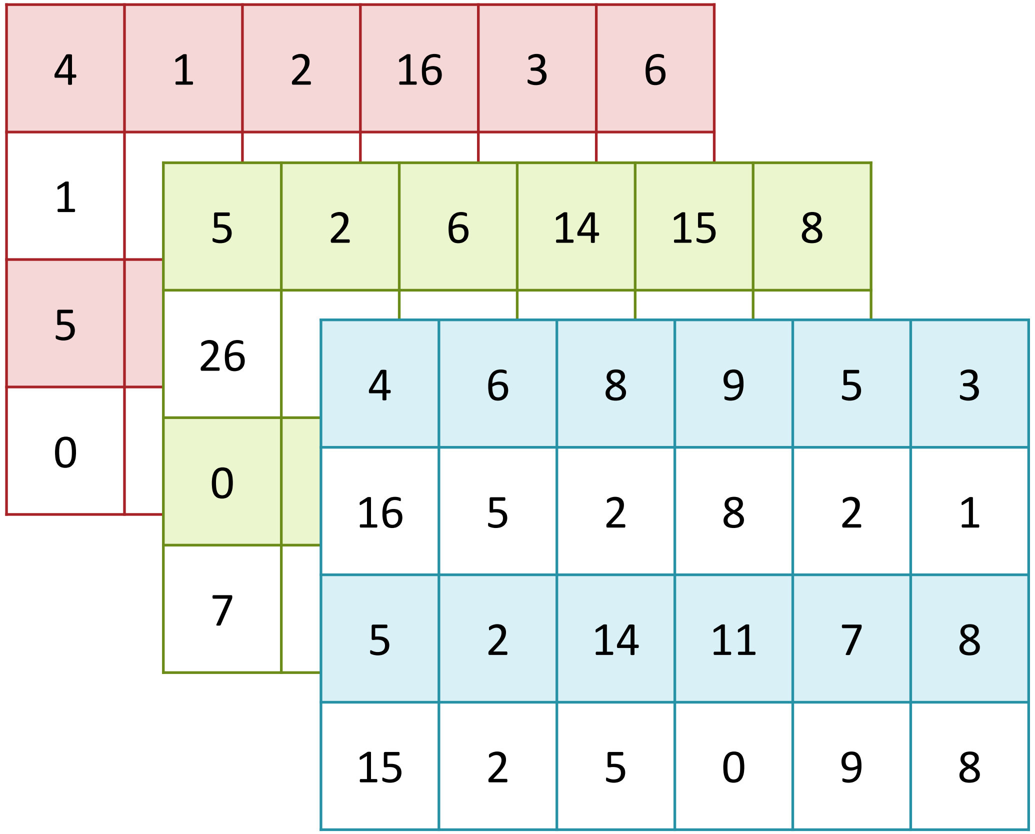
4	1	2	16	3	6
1	7	5	8	19	27
5	2	5	12	17	8
0	4	9	9	6	11

5	2	6	14	15	8
26	3	6	8	4	9
0	15	24	6	1	8
7	4	9	5	24	17

4	6	8	9	5	3
16	5	2	8	2	1
5	2	14	11	7	8
15	2	5	0	9	8

- An image can be represented as the sum of red, green and blue pixel intensities
- Each color corresponds to a *channel*

Example:
$3 \times 4 \times 6$ tensor



4	1	2	16	3	6
1					
5					
0					

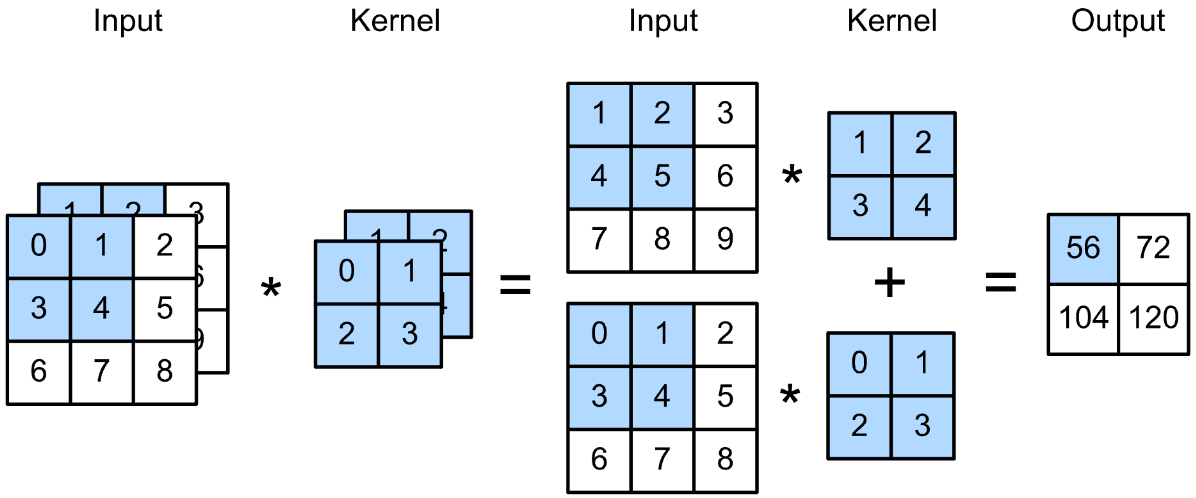
5	2	6	14	15	8
26					
0					
7					

4	6	8	9	5	3
16	5	2	8	2	1
5	2	14	11	7	8
15	2	5	0	9	8

- An image can be represented as a *tensor* or multidimensional array

Convolutions on Multiple Input Channels

- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor

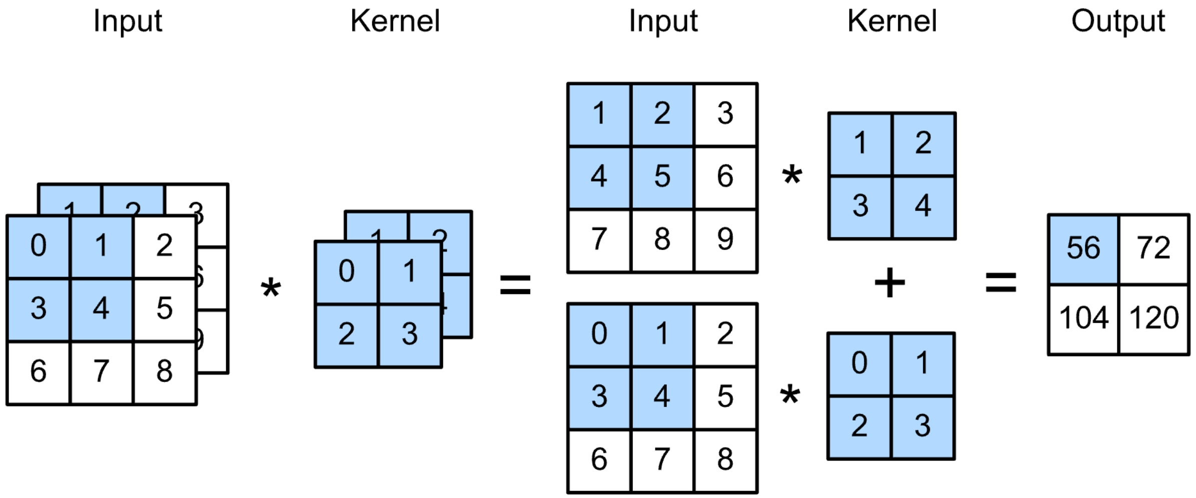


- For c channels and $h \times w$ filters, we have $chw + c$ learnable parameters (each filter has a bias term)

Source: http://preview.d2l.ai/d2l-en/master/chapter_convolutional-neural-networks/channels.html

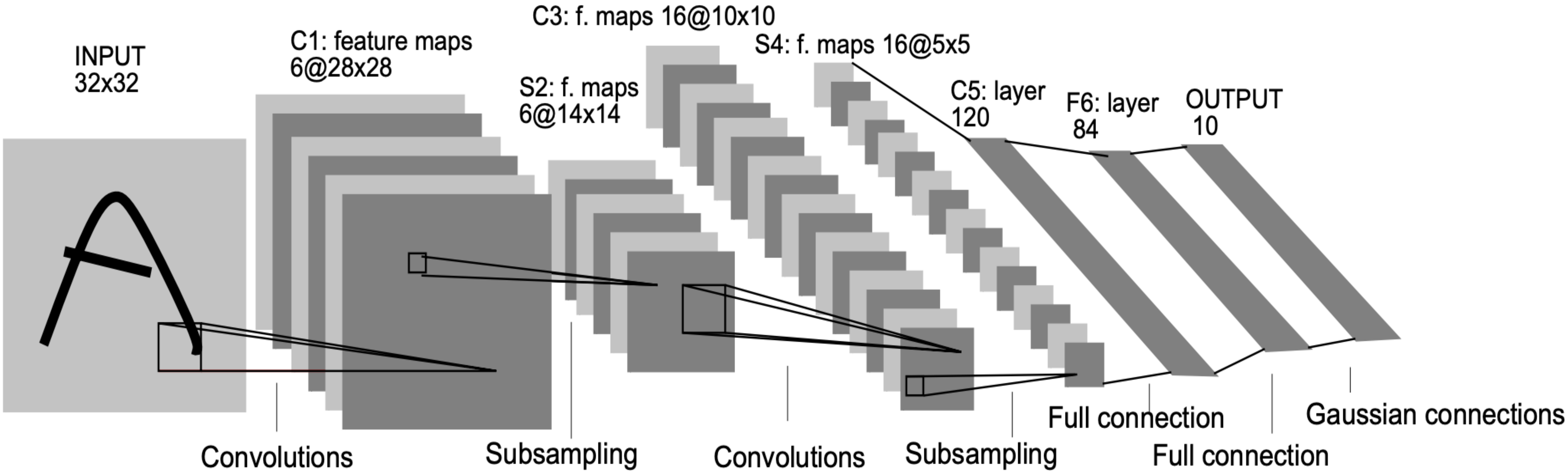
Convolutions on Multiple Input Channels

- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor



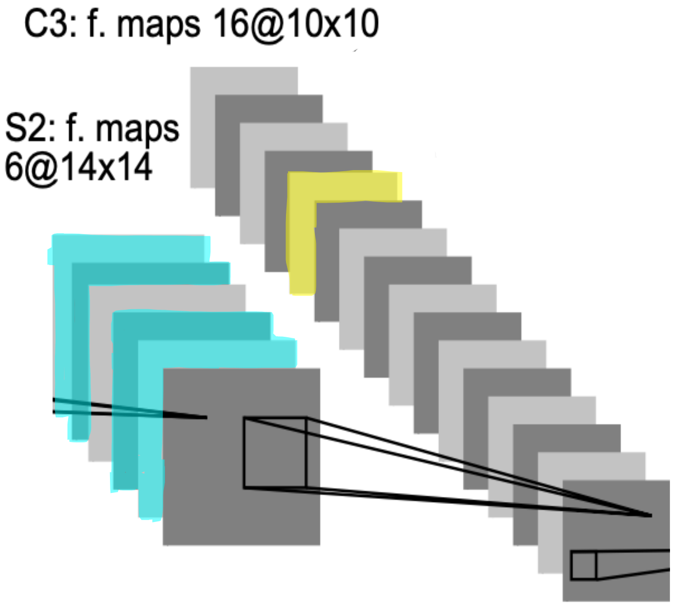
- Questions:
  1. Why might we want a different filter for each input?
  2. Why do we combine them together into a single output channel?

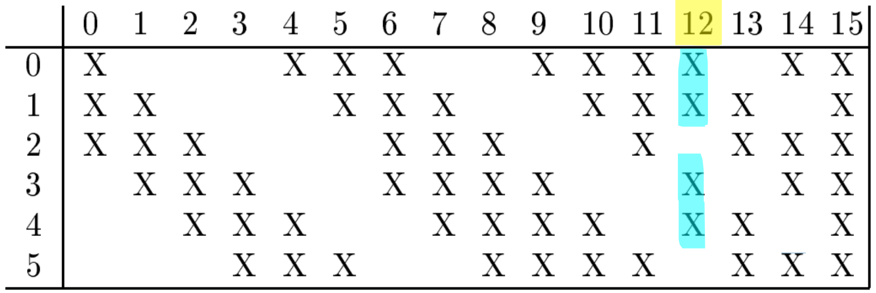
Source: http://preview.d2l.ai/d2l-en/master/chapter_convolutional-neural-networks/channels.html



- Channels in hidden layers correspond to different macro-features, which we might want to manipulate differently → one filter per channel

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf





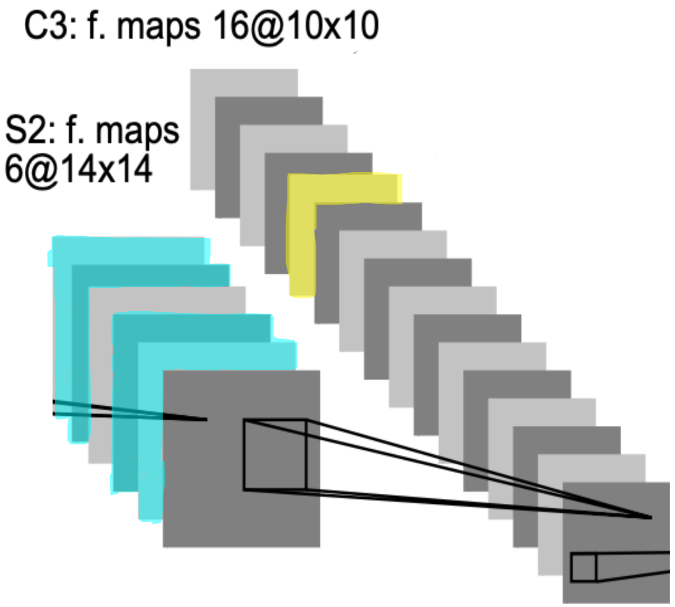
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

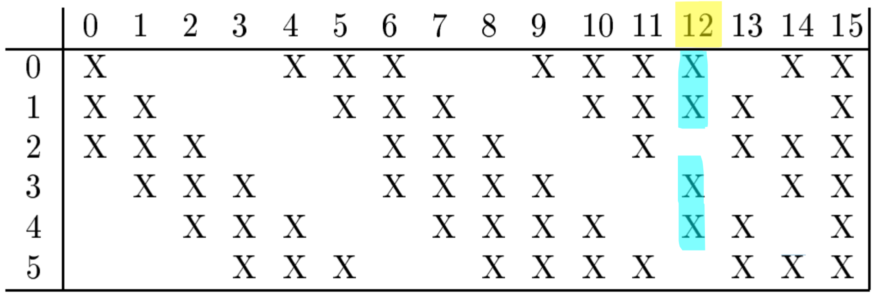
TABLE I

Each column indicates which feature map in S2 are combined by the units in a particular feature map of C3.

- We can combine these macro-features into a new, interesting, "higher-level" feature
  - But we don't always need to combine all of them!
  - Different combinations → multiple output channels
  - Common architecture: more output channels and smaller outputs in deeper layers

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf





	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

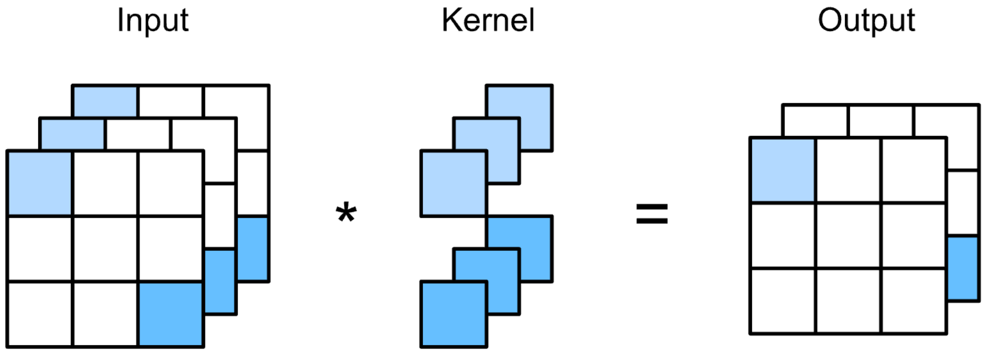
Each column indicates which feature map in S2 are combined by the units in a particular feature map of C3.

Okay, but what if our layers become too big in the channel dimension?

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Downsampling: 1×1 Convolutions

- Convolutional layers can be represented as 4-D tensors of size $c_o \times c_i \times h \times w$ where c_o is the number of output channels and c_i is the number of input channels
- Layers of size $c_o \times c_i \times 1 \times 1$ can condense many input channels into fewer output channels (if $c_o < \ c_i$)



- Practical note: 1×1 convolutions are typically followed by a nonlinear activation function; otherwise, they could simply be folded into other convolutions

Source: http://preview.d2l.ai/d2l-en/master/chapter_convolutional-neural-networks/channels.html

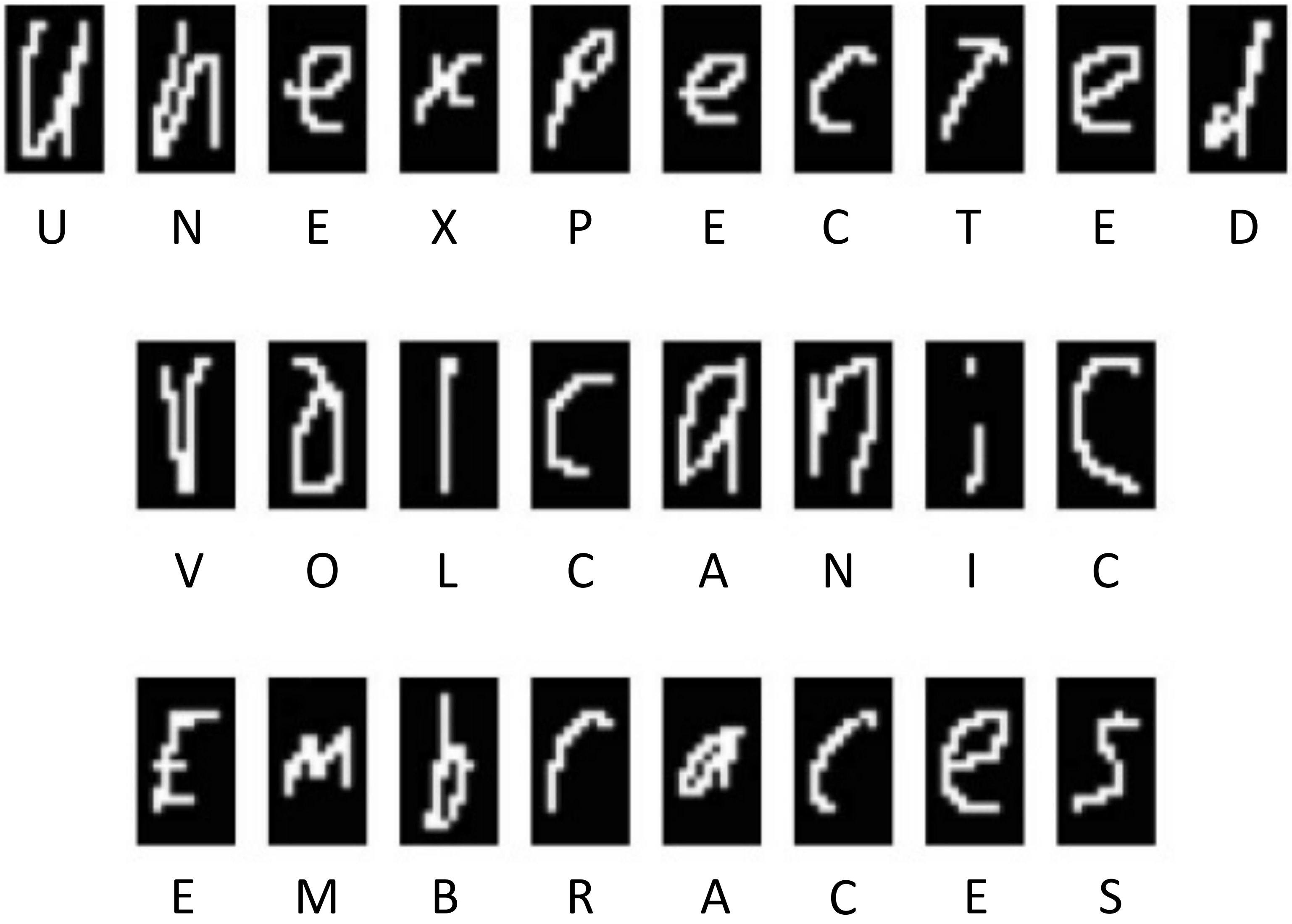
Key Takeaways

- The loss function for neural networks is non-convex!
  - Momentum can help break out of local minima
  - Adaptive gradients help when parameters behave differently w.r.t. step sizes
  - Random restarts can improve the changes of finding better local minima
  - Jitter & dropout act like regularization for neural networks by preventing them fitting the training dataset perfectly
- MLPs and neural networks of sufficient depth are universal approximators

Key Takeaways

- Convolutional neural networks use convolutions to learn macro-features
  - Can be thought of as slight modifications to the fully-connected feed-forward neural network
  - Can still be learned using SGD
  - Padding is used to preserve spatial dimensions while pooling, stride and 1×1 convolutions are used to downsample intermediate representations

Example: Handwriting Recognition

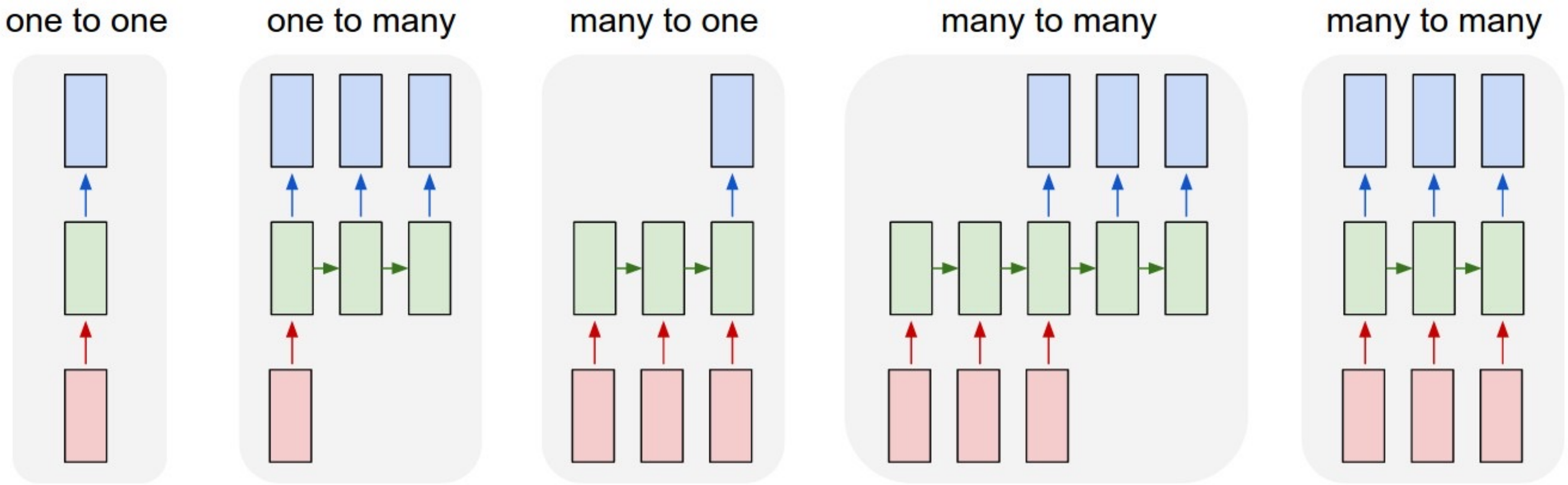


Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

$$\boldsymbol{y}^{(i)} = \left[\boldsymbol{y}_1^{(i)}, \boldsymbol{y}_2^{(i)}, \dots, \boldsymbol{y}_{T_i}^{(i)}\right]$$

Outputs, $y^{(i)}$

Inputs, $\boldsymbol{x}^{(i)}$



$$\boldsymbol{x}^{(i)} = \left[\boldsymbol{x}_1^{(i)}, \boldsymbol{x}_2^{(i)}, \dots, \boldsymbol{x}_{T_i}^{(i)}\right]$$

Sequential Data

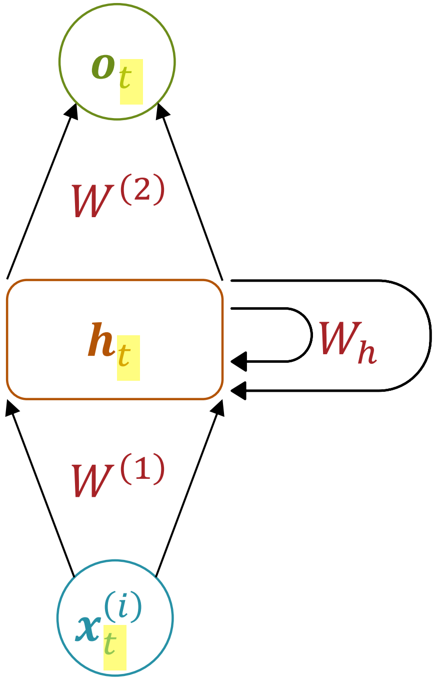
Source: https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

- Neural networks are frequently applied to inputs with some inherent temporal or sequential structure (e.g., text or video) of variable length
- 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)
- Approach: 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 $\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$



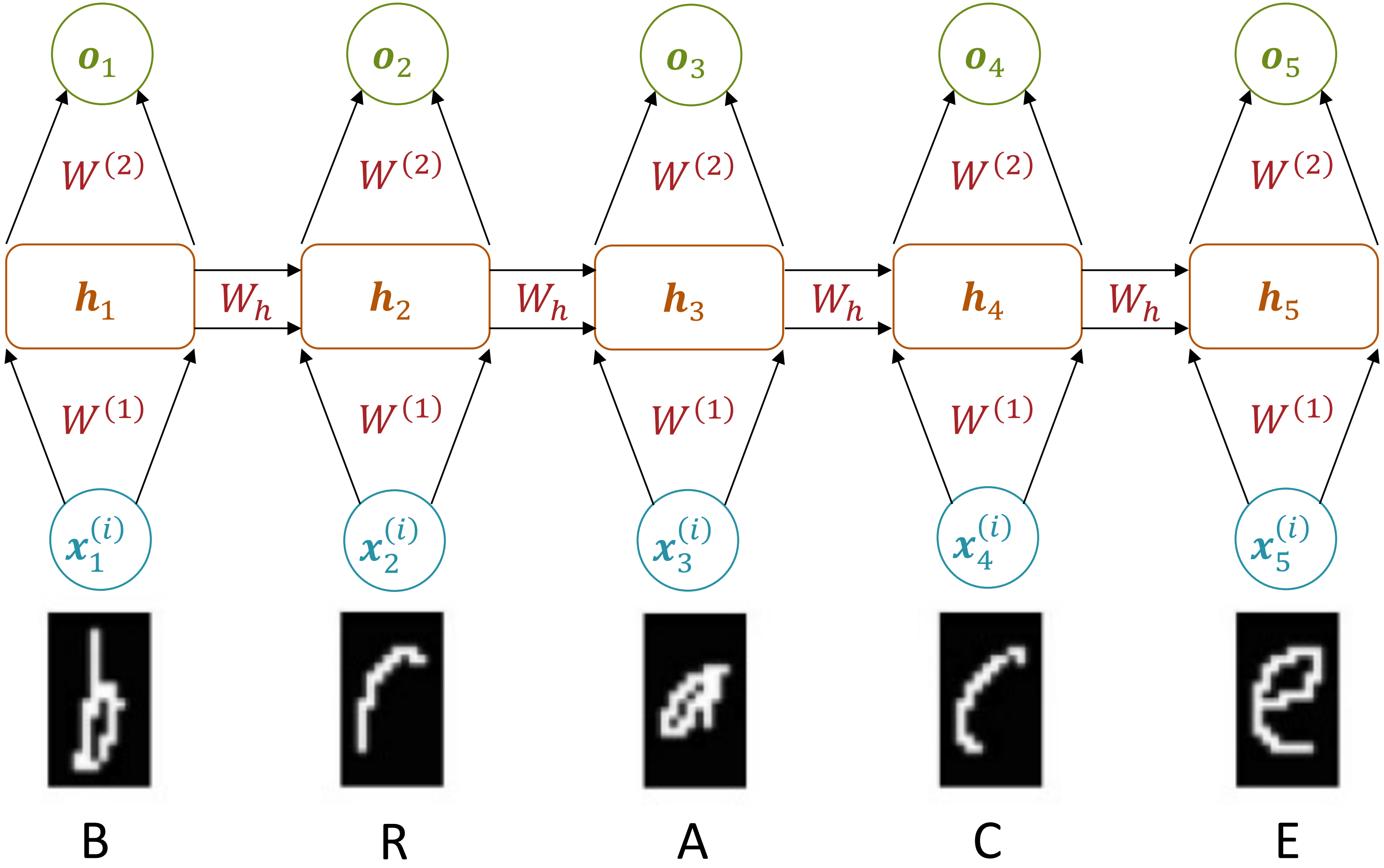
- Training dataset consists of (input **sequence**, label **sequence**) pairs, potentially of varying lengths

$$\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\right)\right\}_{n=1}^{N}$$
$$\boldsymbol{x}^{(n)} = \left[\boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_{T_n}^{(n)}\right]$$
$$\boldsymbol{y}^{(n)} = \left[\boldsymbol{y}_1^{(n)}, \dots, \boldsymbol{y}_{T_n}^{(n)}\right]$$

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

Unrolling Recurrent Neural Networks

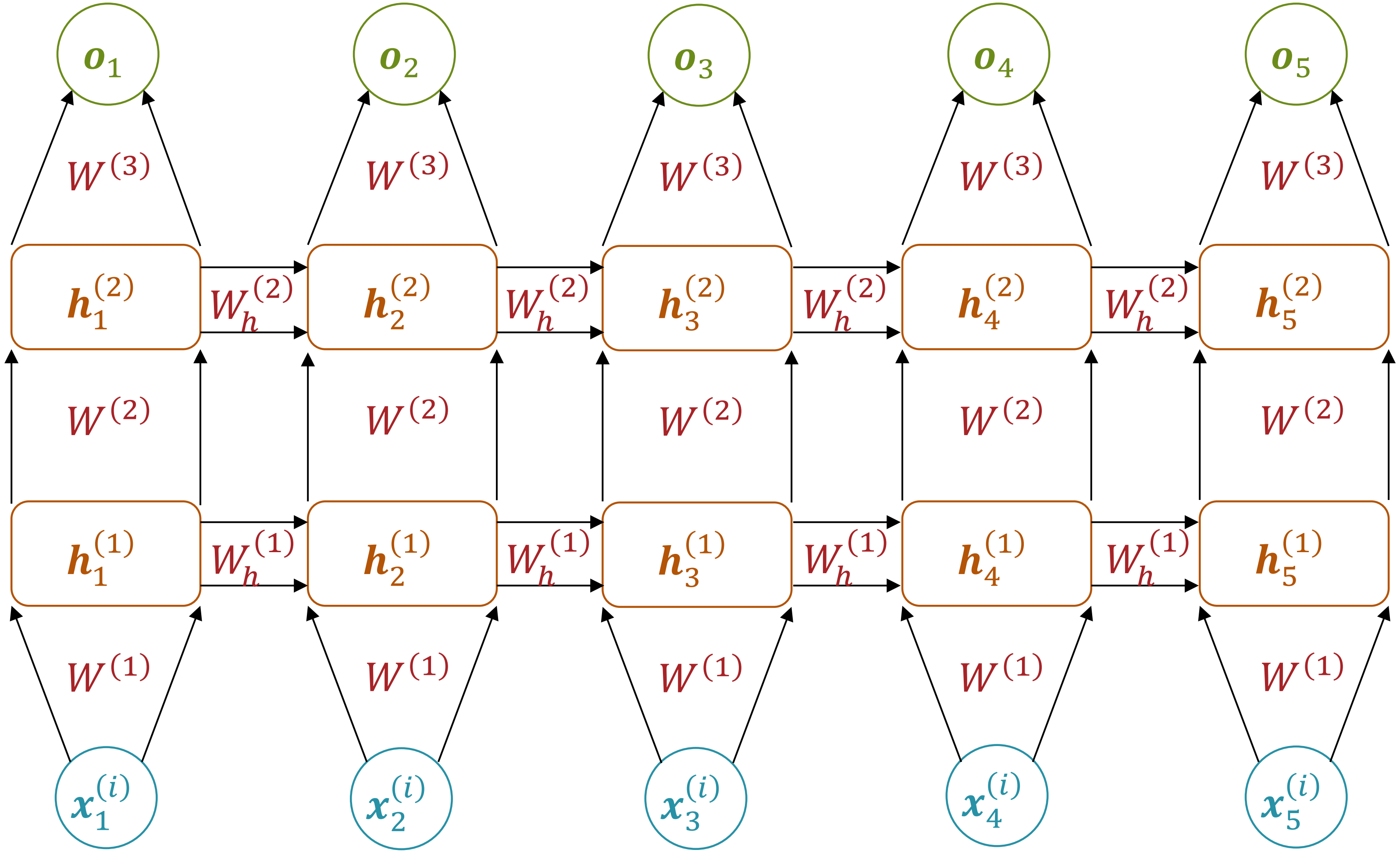
$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h \boldsymbol{h}_{t-1}\right)\right]^T$ and $\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$



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

Deep Recurrent Neural Networks

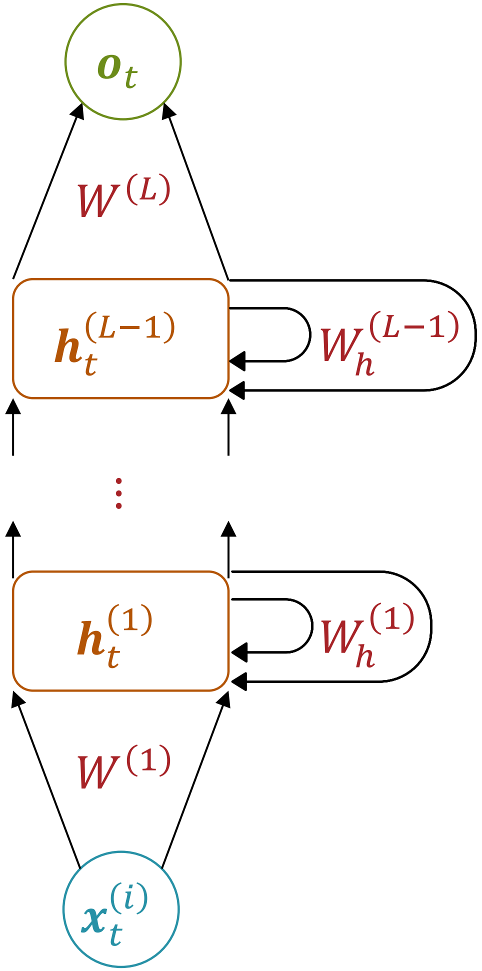
$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T$ and $\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$



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

Deep Recurrent Neural Networks

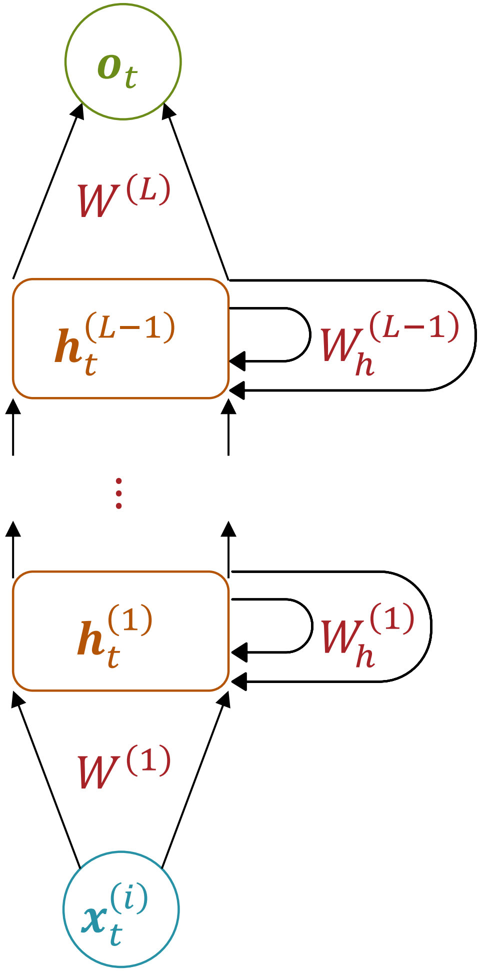
$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T$ and $\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$



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?

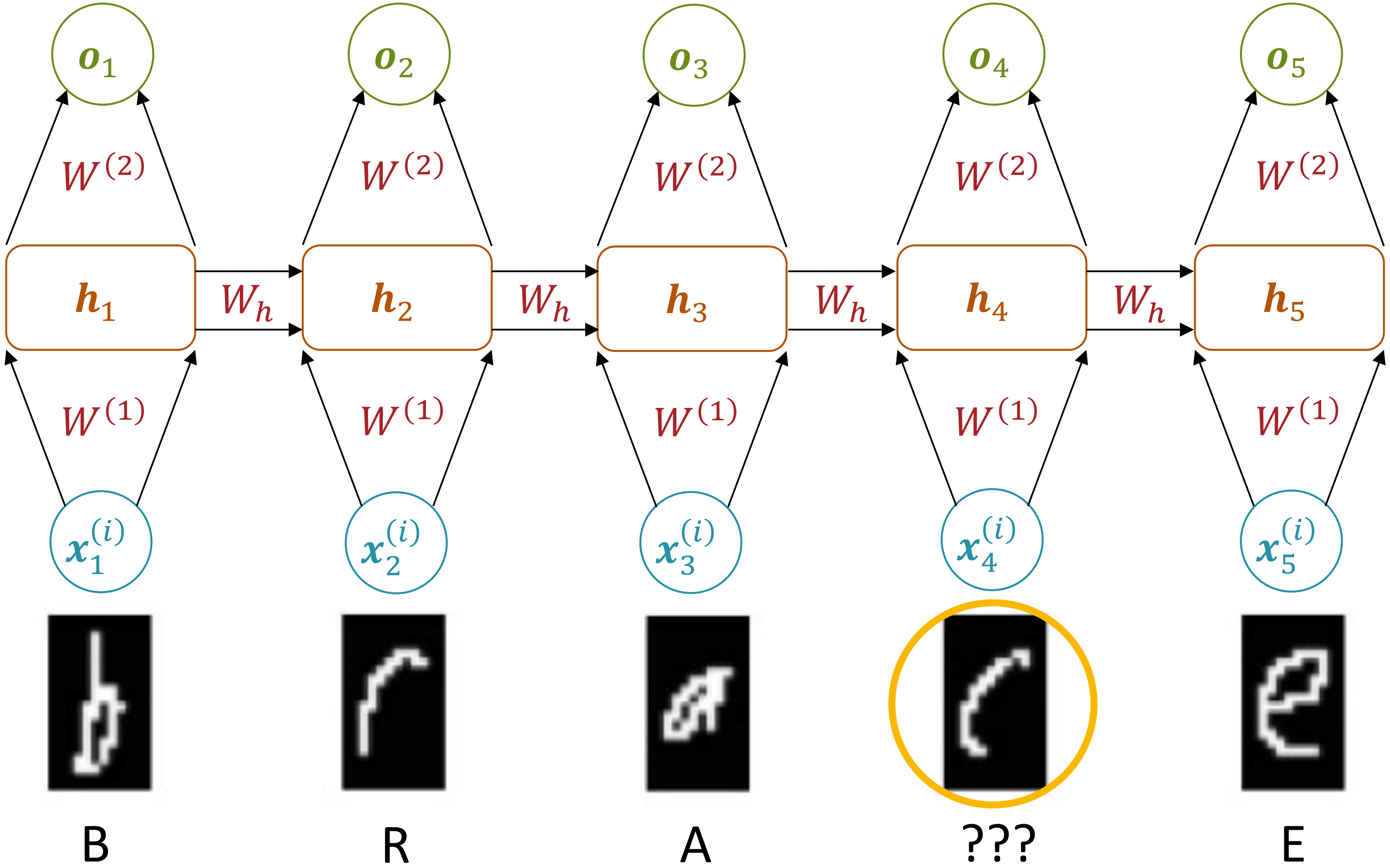
$$\boldsymbol{h}_t^{(l)} = \left[1, \theta\left(W^{(l)}\boldsymbol{h}_t^{(l-1)} + W_h^{(l)}\boldsymbol{h}_{t-1}^{(l)}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(L)}\boldsymbol{h}_t^{(L-1)}\right)$$



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?

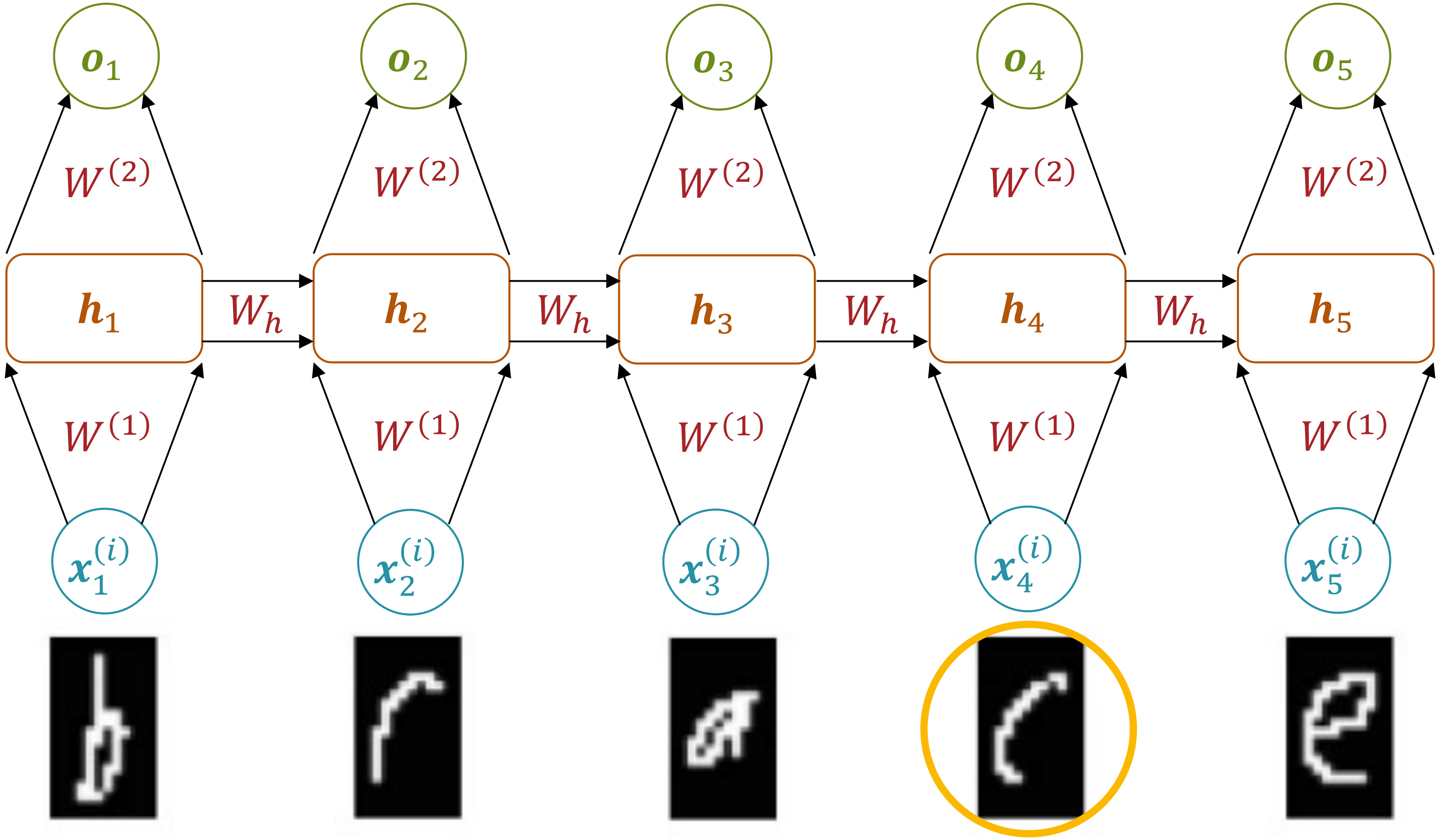
$$\boldsymbol{h}_t = \left[1, \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h \boldsymbol{h}_{t-1}\right)\right]^T \text{ and } \boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W^{(2)}\boldsymbol{h}_t\right)$$



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

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

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



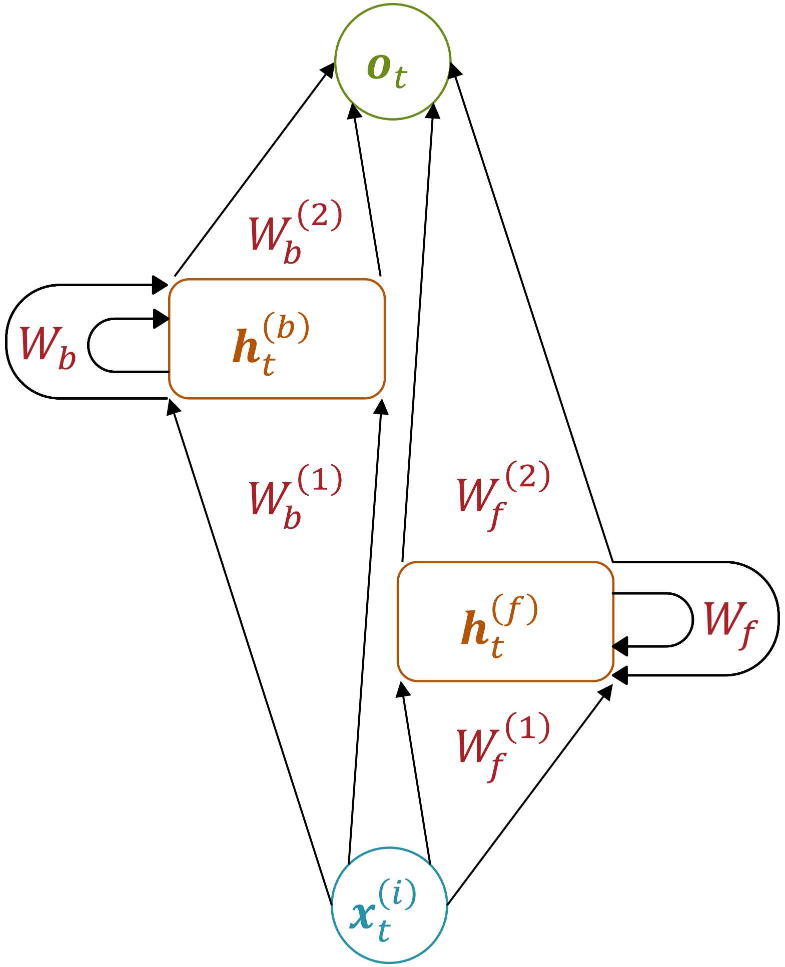
B R A I or N or S or V or K … E

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

Bidirectional Recurrent Neural Networks

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

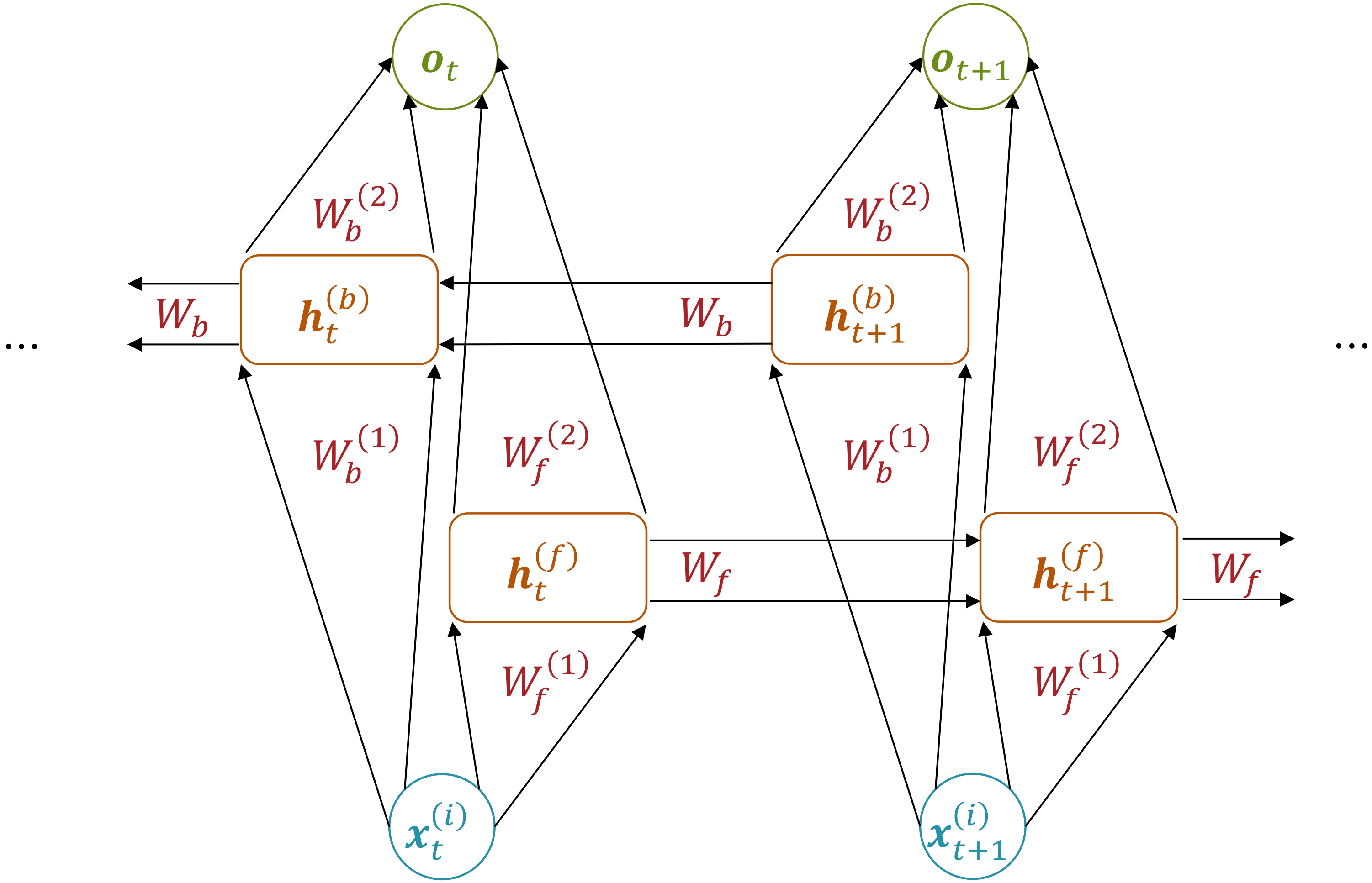
$$\boldsymbol{o}_t = \hat{y}_t^{(i)} = \theta\left(W_f^{(2)}\boldsymbol{h}_t^{(f)} + W_b^{(2)}\boldsymbol{h}_t^{(b)}\right)$$



Unrolling Bidirectional Recurrent Neural Networks

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

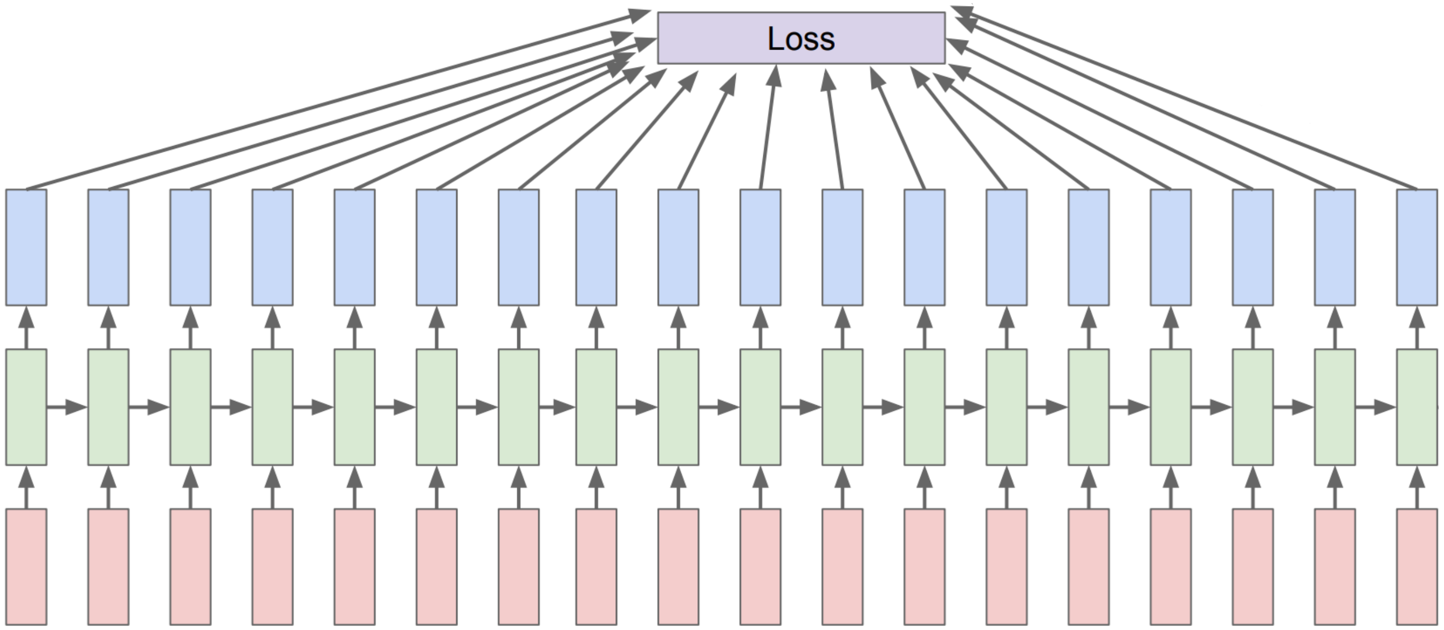
$\boldsymbol{h}_t^{(f)} = \left[1, \theta\left(W_f^{(1)}\boldsymbol{x}_t^{(i)} + W_f\boldsymbol{h}_{t-1}\right)\right]^T$ and $\boldsymbol{h}_t^{(b)} = \left[1, \theta\left(W_b^{(1)}\boldsymbol{x}_t^{(i)} + W_b\boldsymbol{h}_{t+1}\right)\right]^T$



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"

Training RNNs

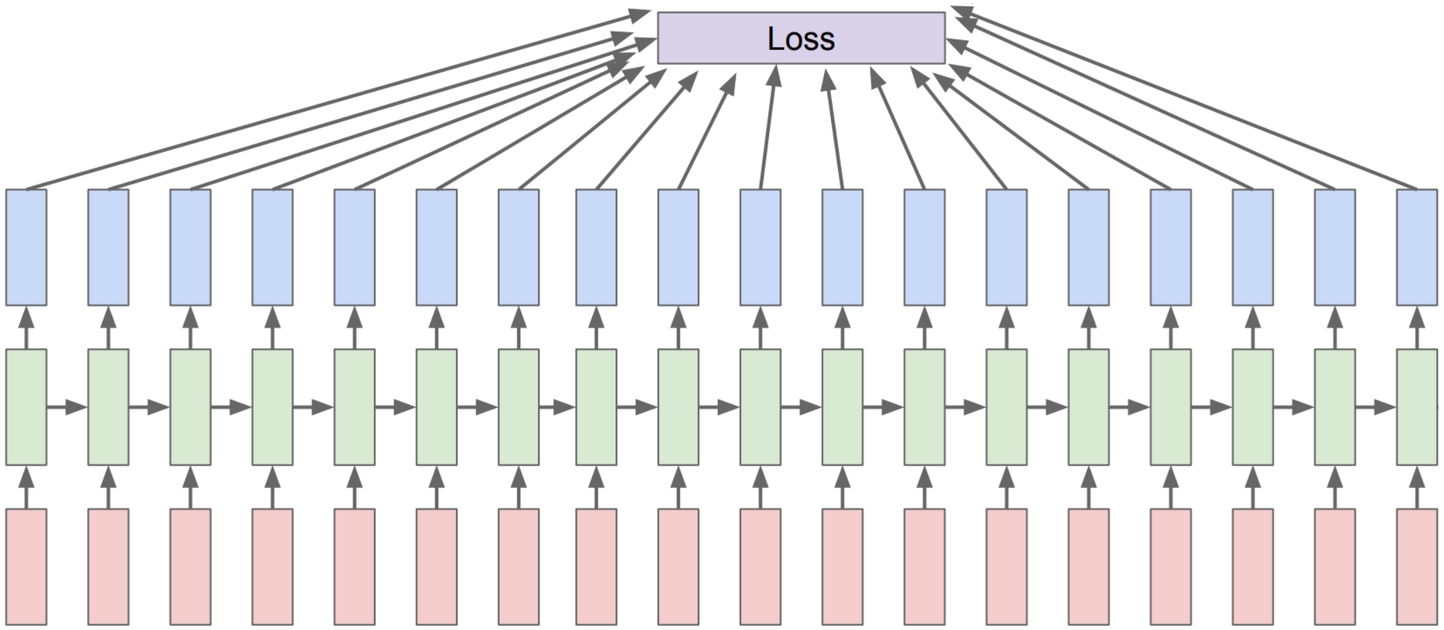


Forward pass to compute outputs and hidden representations

Backward pass to compute gradients

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

Training RNNs: Challenges



Forward pass to compute outputs and hidden representations

Backward pass to compute gradients

- Issue: as the sequence length grows, the gradient is more likely to explode or vanish

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

Recall: Vanishing Gradients

Insight: $s_b^{(l)}$ *only* affects $\ell^{(i)}$ via $o_b^{(l)}$

Chain rule: $\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$

$$o_b^{(l)} = \theta\left(s_b^{(l)}\right) \rightarrow \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta\left(s_b^{(l)}\right)}{\partial s_b^{(l)}}$$

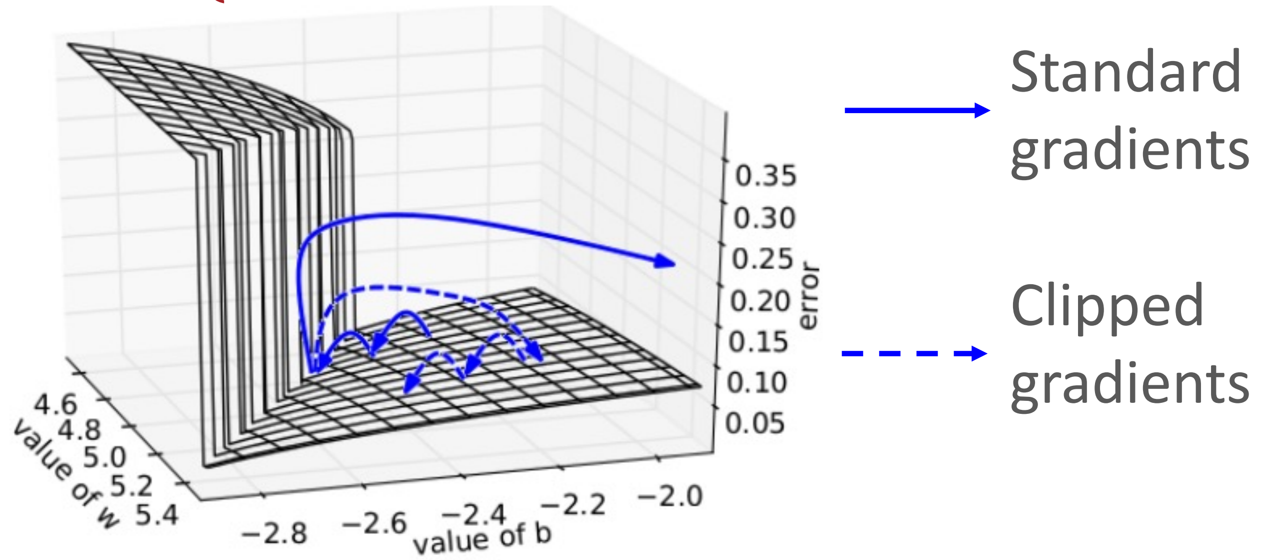
$$= 1 - \left(\tanh\left(s_b^{(l)}\right)\right)^2 \leq 1$$

when $\theta(\cdot) = \tanh(\cdot)$

Gradient Clipping (Pascanu et al., 2013)

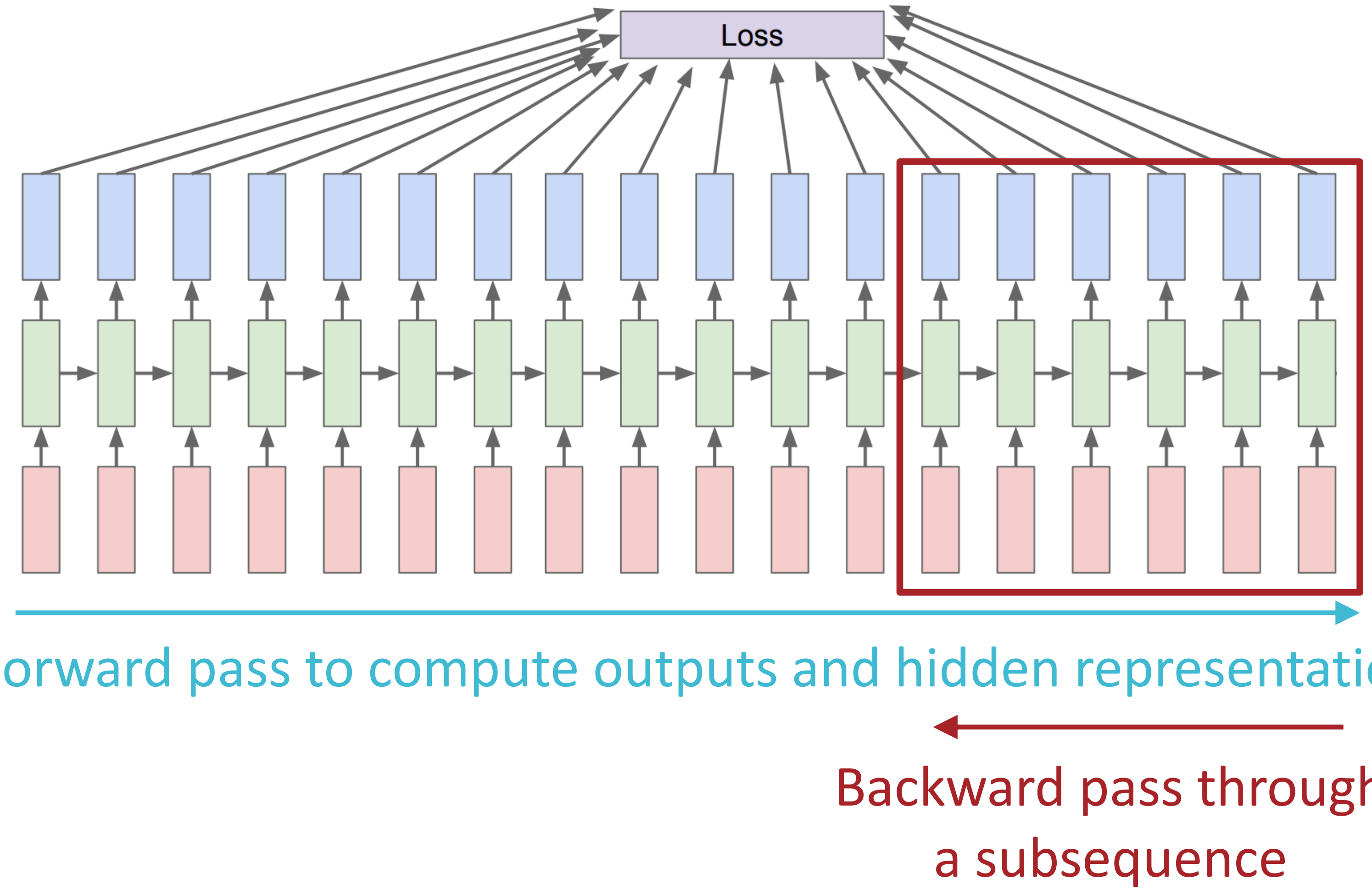
- Common strategy to deal with exploding gradients: if the magnitude of the gradient ever exceeds some threshold, simply scale it down to the threshold

$$G = \begin{cases} \nabla_W \ell^{(i)} & \text{if } \left\|\nabla_W \ell^{(i)}\right\|_2 \le \tau \\ \left(\frac{\tau}{\left\|\nabla_W \ell^{(i)}\right\|_2}\right) \nabla_W \ell^{(i)} & \text{otherwise} \end{cases}$$



Source: https://arxiv.org/pdf/1211.5063.pdf

Truncated Backpropagation Through Time



Forward pass to compute outputs and hidden representations

Backward pass through a subsequence

- Idea: limit the number of time steps to backprop through

Source: http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

- LSTM networks address the vanishing gradient problem by replacing hidden layers with *memory cells*
- Each cell still computes a hidden representation but also maintains a separate internal *state,* C_t
- The flow of information through a cell is manipulated by three *gates*:
    - An input gate, I_t, that controls how much the state looks like the normal RNN hidden layer
    - An output gate, O_t, that “releases” the hidden representation to later timesteps
    - A forget gate, F_t, that determines if the previous memory cell’s state affects the current internal state

Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

- LSTM networks address the vanishing gradient problem by replacing hidden layers with *memory cells*
- Each cell still computes a hidden representation but also maintains a separate internal *state,* C_t
- Gates are implemented as sigmoids: a value of 0 would be a fully closed gate and 1 would be fully open

$$I_t = \sigma\left(W_{ix}\boldsymbol{x}_t^{(i)} + W_{ih}\boldsymbol{h}_{t-1}\right)$$

$$O_t = \sigma\left(W_{ox}\boldsymbol{x}_t^{(i)} + W_{oh}\boldsymbol{h}_{t-1}\right)$$

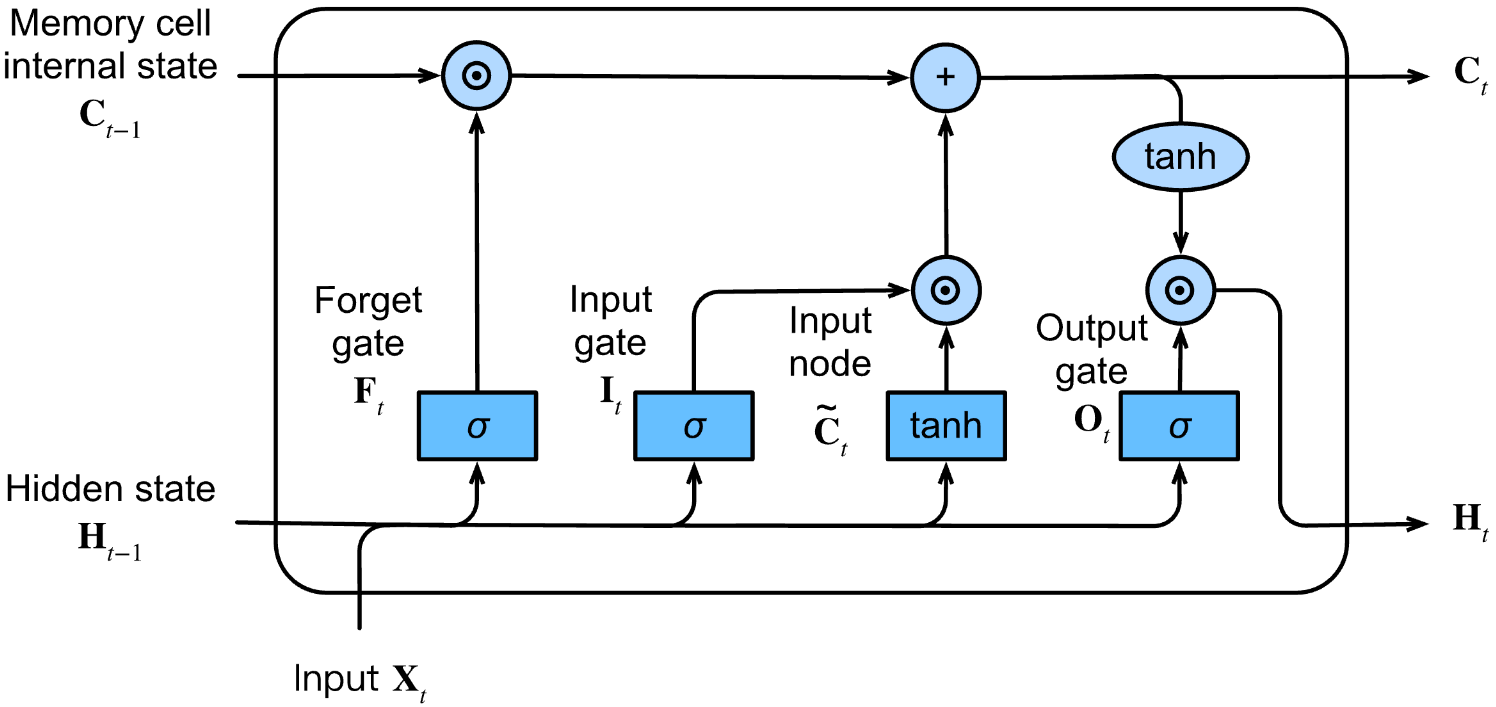
$$F_t = \sigma\left(W_{fx}\boldsymbol{x}_t^{(i)} + W_{fh}\boldsymbol{h}_{t-1}\right)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \theta\left(W^{(1)}\boldsymbol{x}_t^{(i)} + W_h\boldsymbol{h}_{t-1}\right)$$

$$\boldsymbol{h}_t = C_t \odot O_t$$

Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

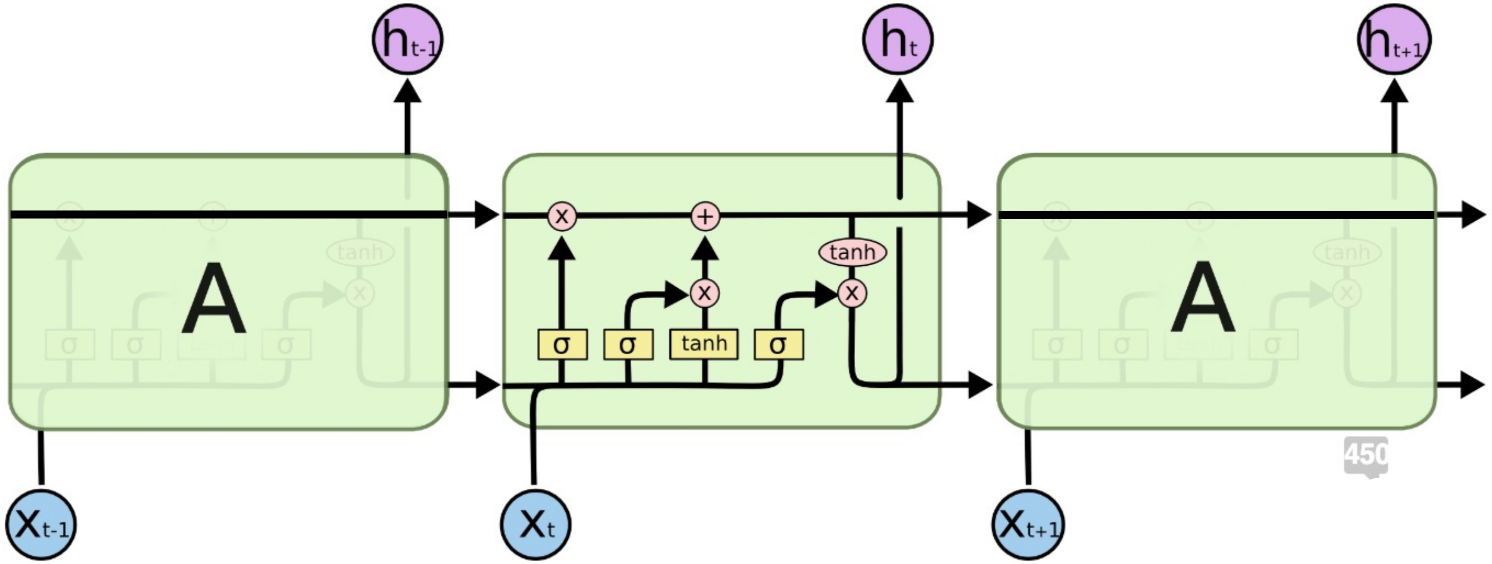
- LSTM networks address the vanishing gradient problem by replacing hidden layers with *memory cells*
- Each cell still computes a hidden representation but also maintains a separate internal *state,* C_t



Source: https://d2l.ai/chapter_recurrent-modern/lstm.html

Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

- LSTM networks address the vanishing gradient problem by replacing hidden layers with *memory cells*
- Each cell still computes a hidden representation but also maintains a separate internal *state*, C_t



- The internal state allows information to move through time without needing to affect the hidden representations!

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/