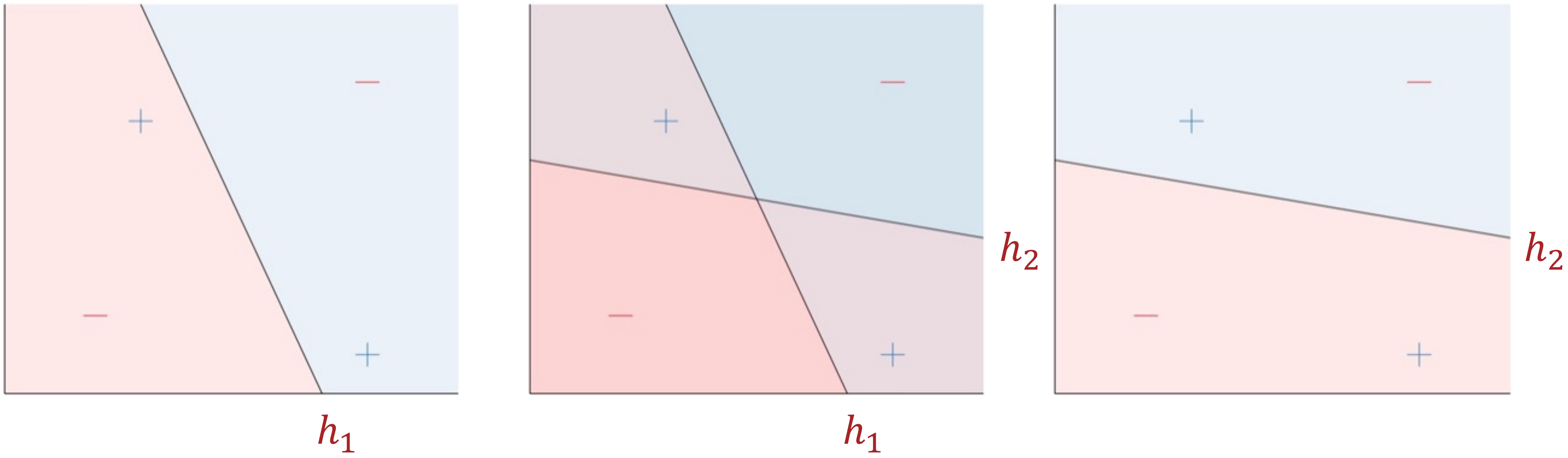
10-301/601: Introduction to Machine Learning Lecture 15 – Deep Learning

Henry Chai

6/21/23

Front Matter

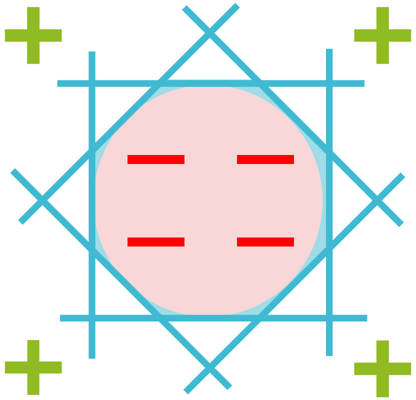
- Announcements
  - PA4 released 6/15, due 7/13 at 11:59 PM
    - We have scheduled this so that **you do not have to be working on PA4 over break!**
  - Midterm on 6/23, this Friday!
    - OH in lieu of recitation on 6/22 (tomorrow)
  - Keep an eye out for a mid-semester grade summary and a feedback poll over the break
    - The poll is required and will count towards your participation grade
- Recommended Readings
  - Various papers linked throughout the slides



Recall: Combining Perceptrons

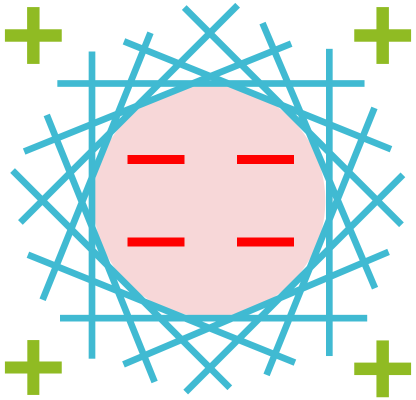
MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP

NNs as Universal Approximators (Cybenko, 1989 & Hornik, 1991)

- Theorem: Any bounded, continuous function can be approximated to an arbitrary precision using a 2-layer (1 hidden layer) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.
- What about unbounded or discontinuous functions?

Source: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F21A09B7475DFB9487990020839A39D2?doi=10.1.1.441.7873&rep=rep1&type=pdf
https://doi.org/10.1016/0893-6080%2891%2990009-T

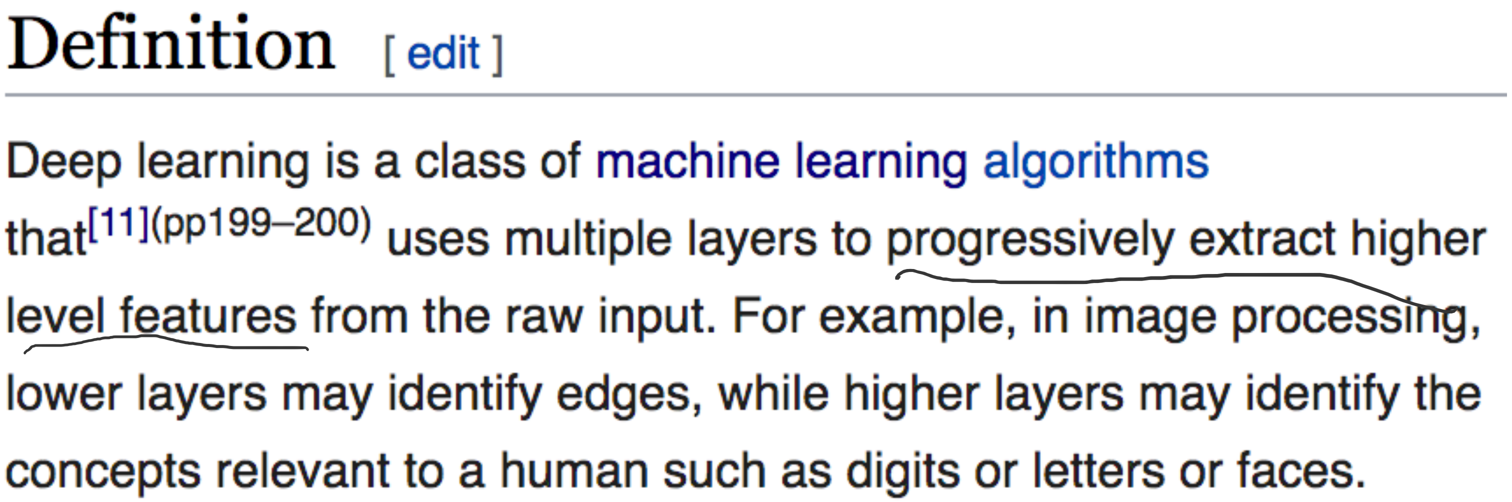
NNs as Universal Approximators (Cybenko, 1988)

- Theorem: Any function can be approximated to an arbitrary precision using a 3-layer (2 hidden layers) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.

Source: G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Dept. of Computer Science, Tufts University, Medford, MA, 1988.

Deep Learning

- From Wikipedia's page on Deep Learning…

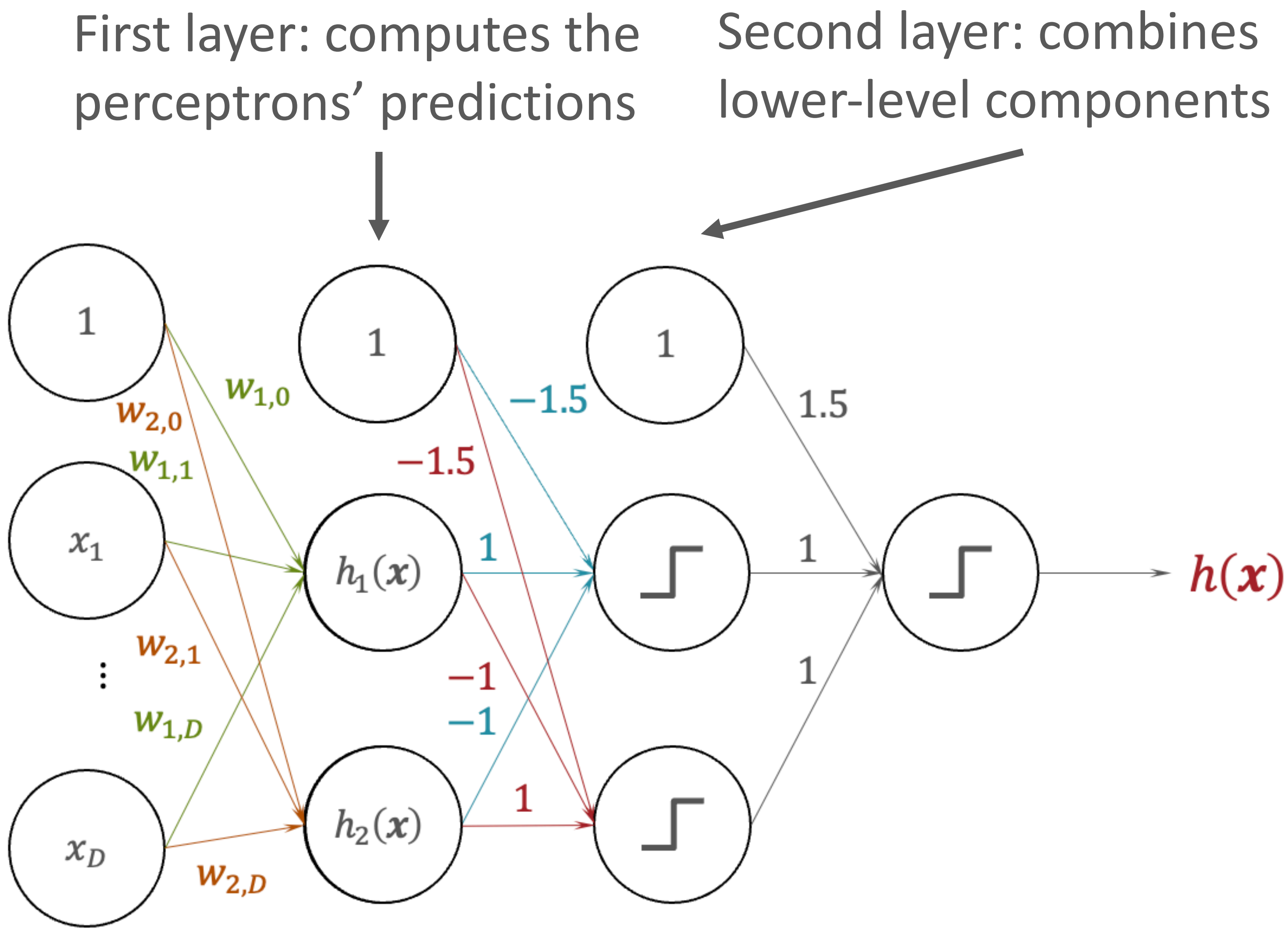


Definition [edit]

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



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

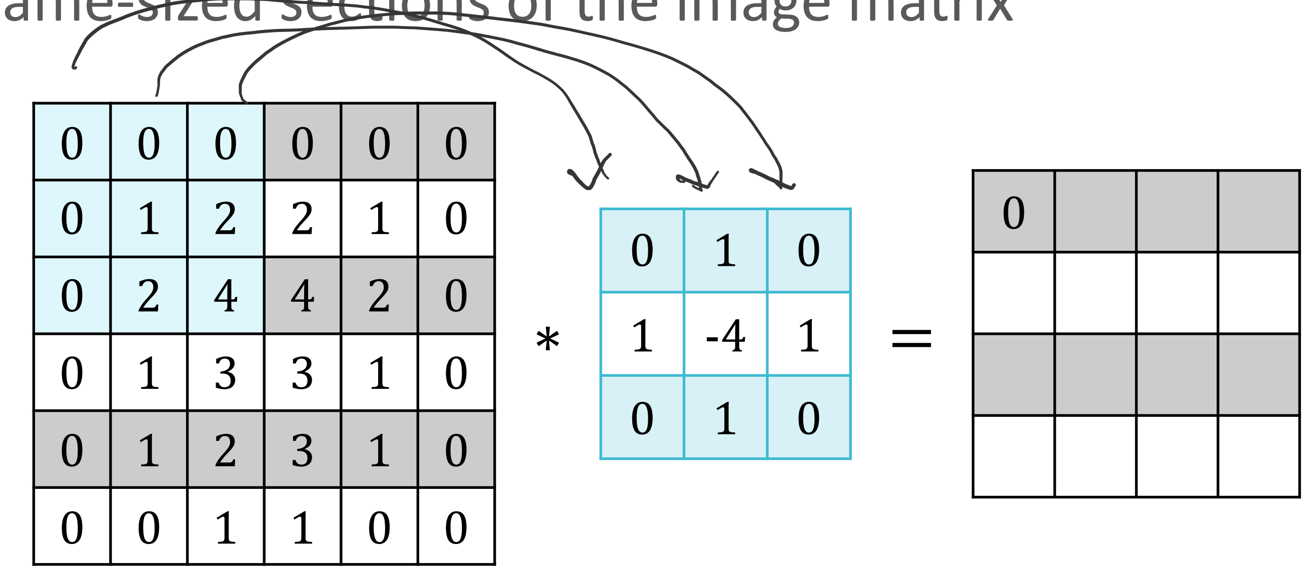
- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

filter

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

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



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

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

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

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & -1 & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

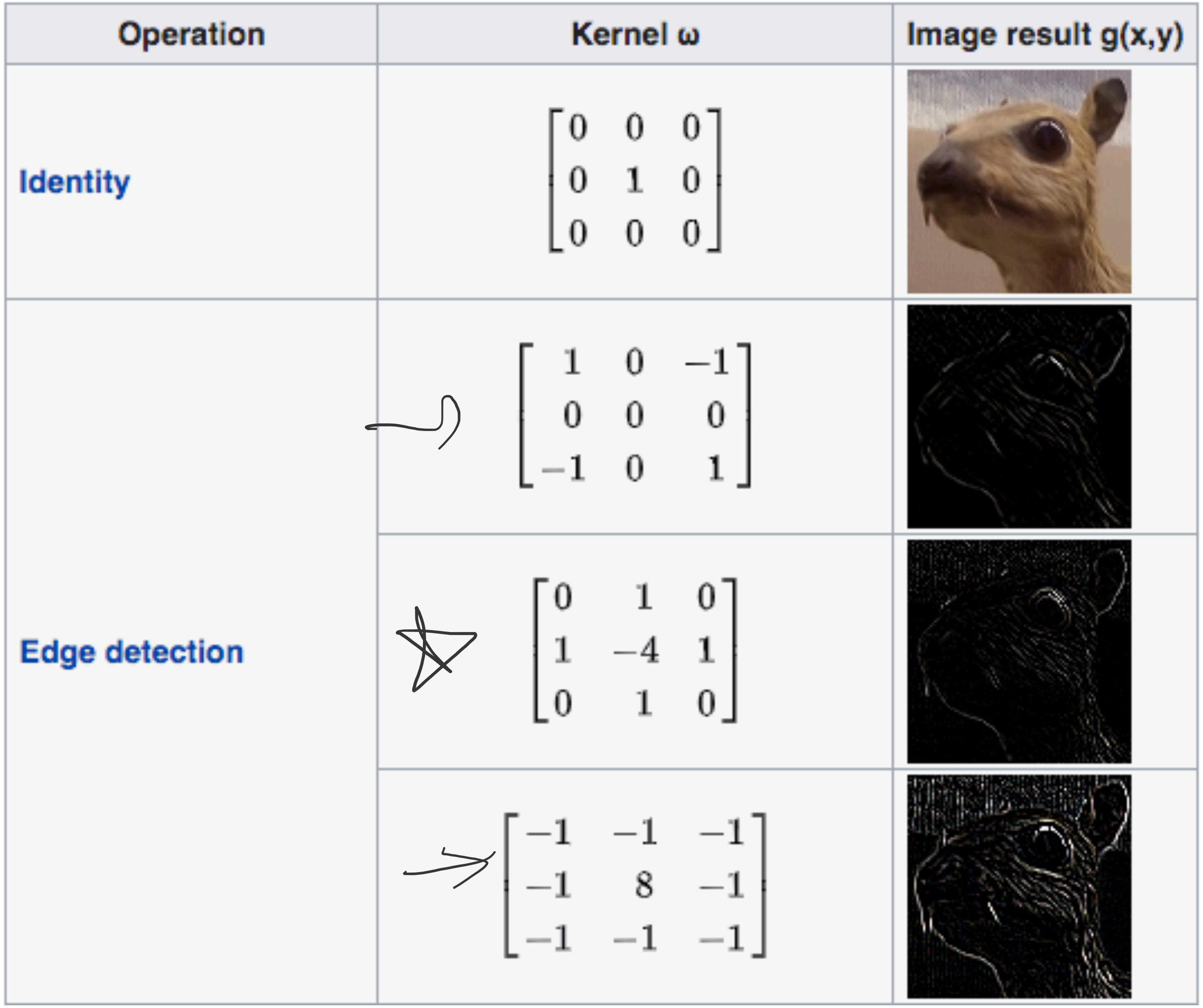
$$(0 * 0) + (0 * 1) + (0 * 0) + (1 * 1) + (2 * -4) \\ + (2 * 1) + (2 * 0) + (4 * 1) + (4 * 0) = -1$$

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

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

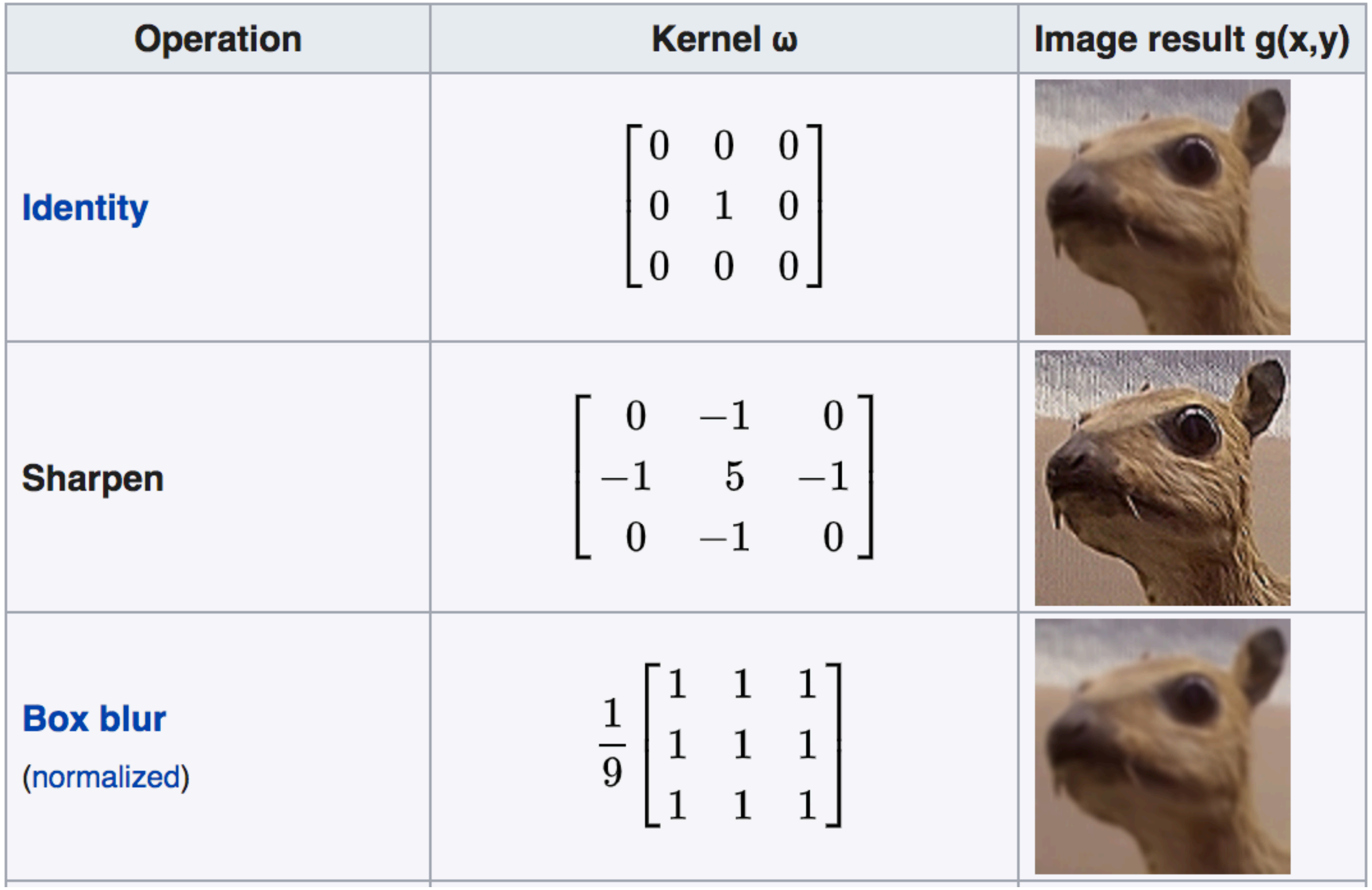
Convolutional Filters



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)

More Filters

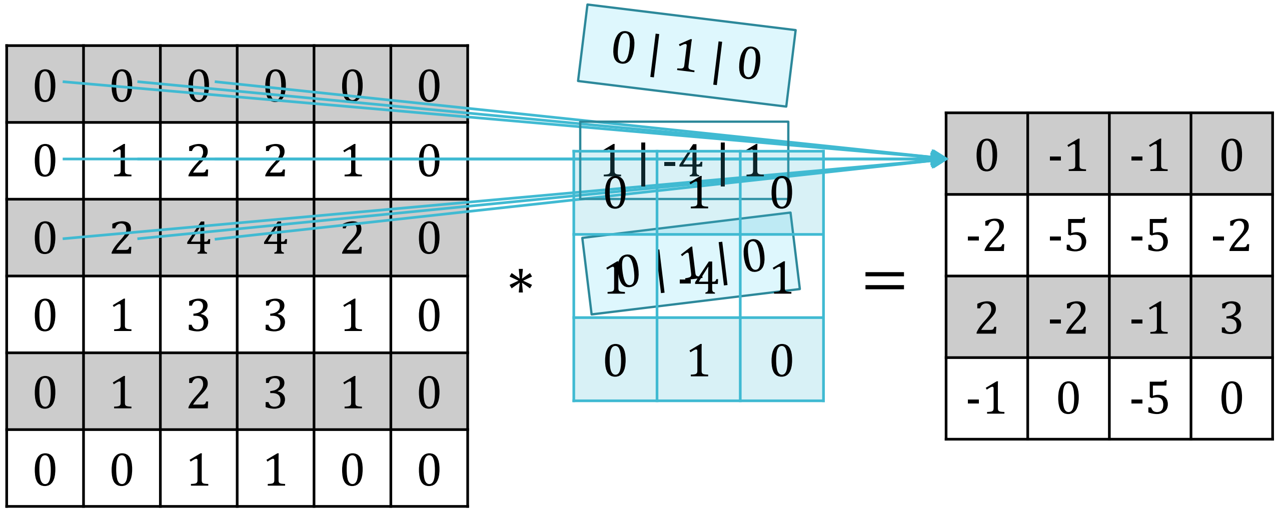


Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

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

Convolutional Filters

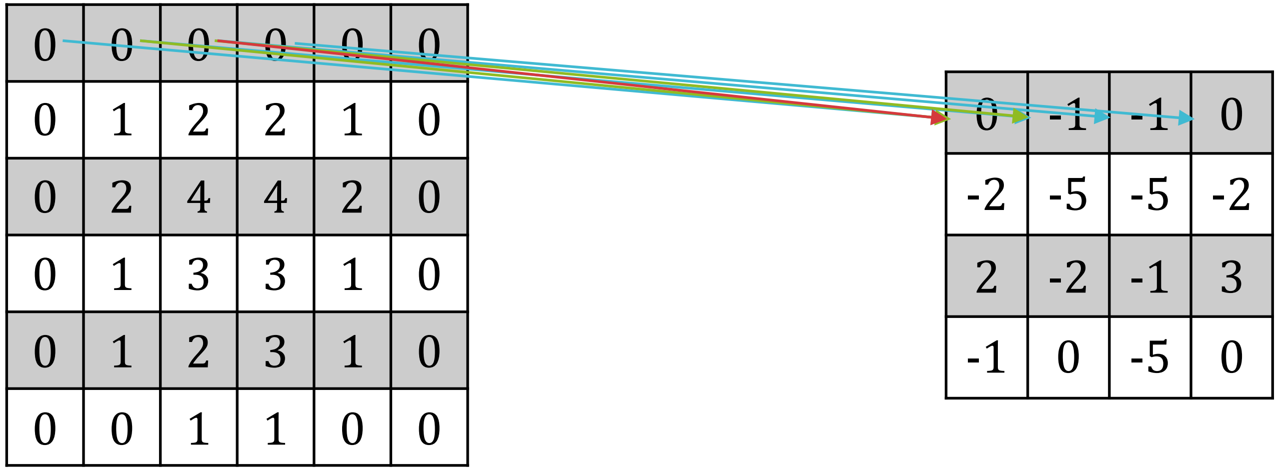
- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix



$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & -1 & -1 & 0 \\ \hline -2 & -5 & -5 & -2 \\ \hline 2 & -2 & -1 & 3 \\ \hline -1 & 0 & -5 & 0 \\ \hline \end{array}$$

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

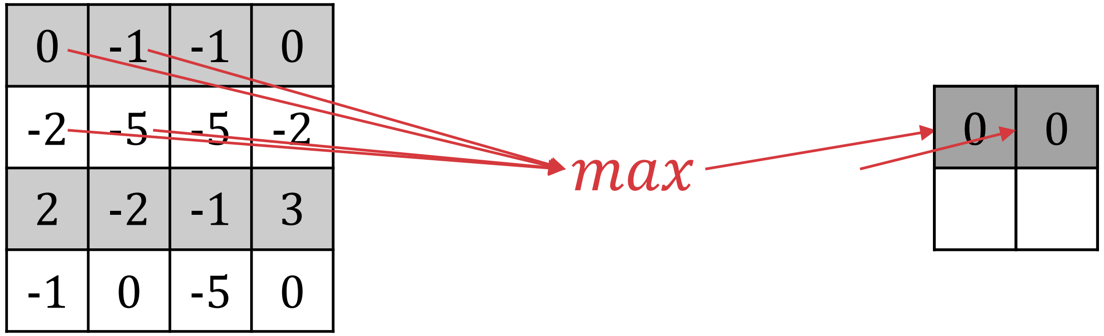
Convolutional Filters: Padding

- 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

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

Downsampling: Pooling

- Combine multiple adjacent nodes into a single node



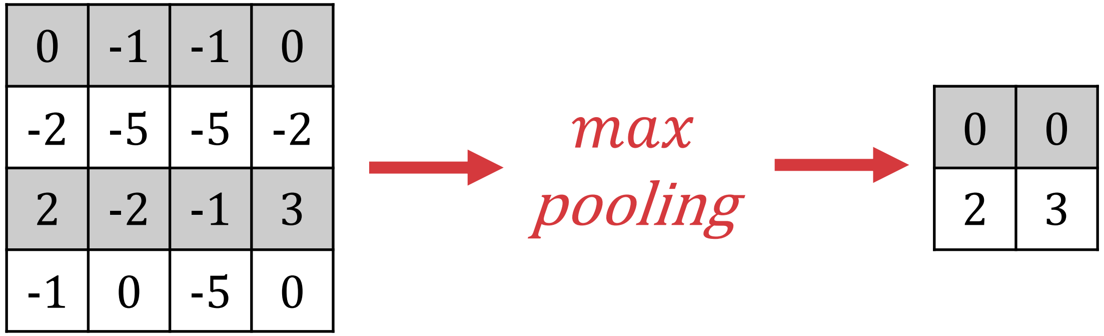
0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

max

0	0

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

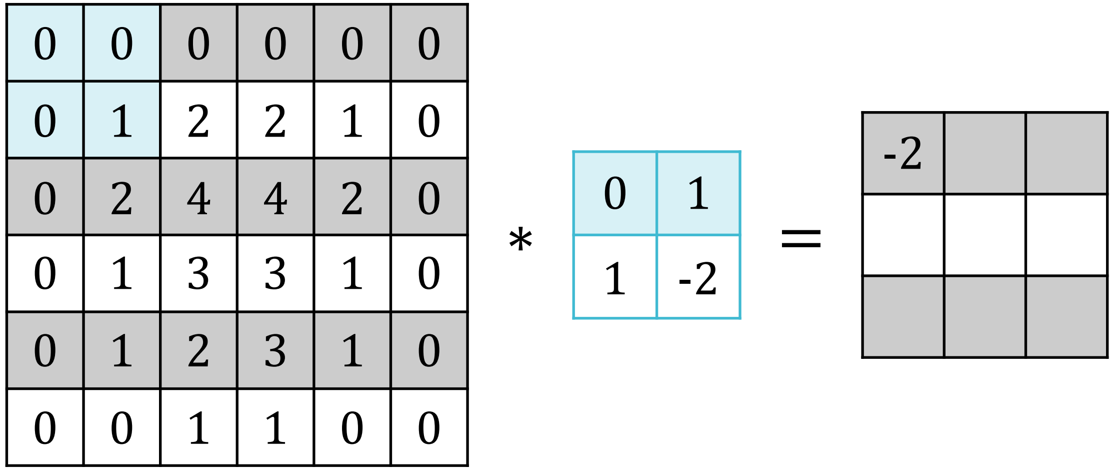
0	0
2	3

- 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

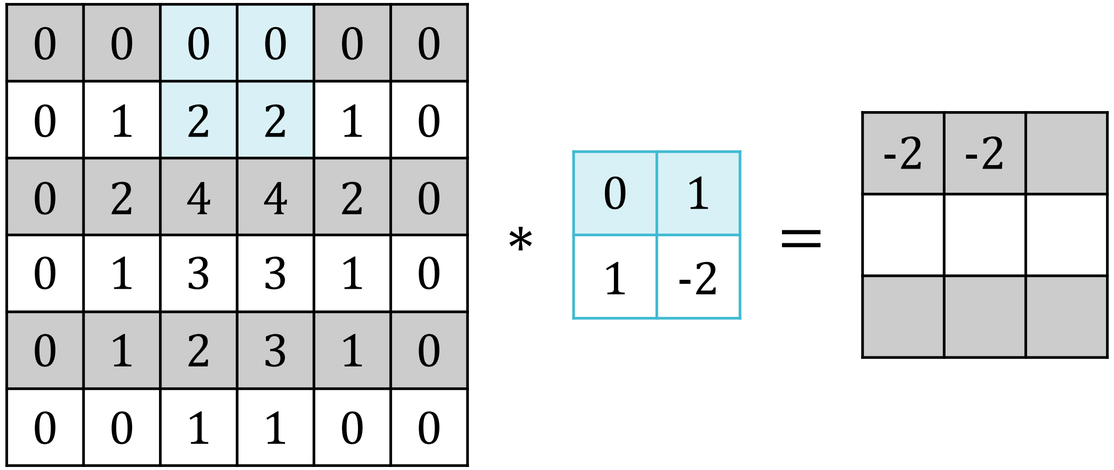


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & & \\ & & \\ & & \end{bmatrix}$$

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

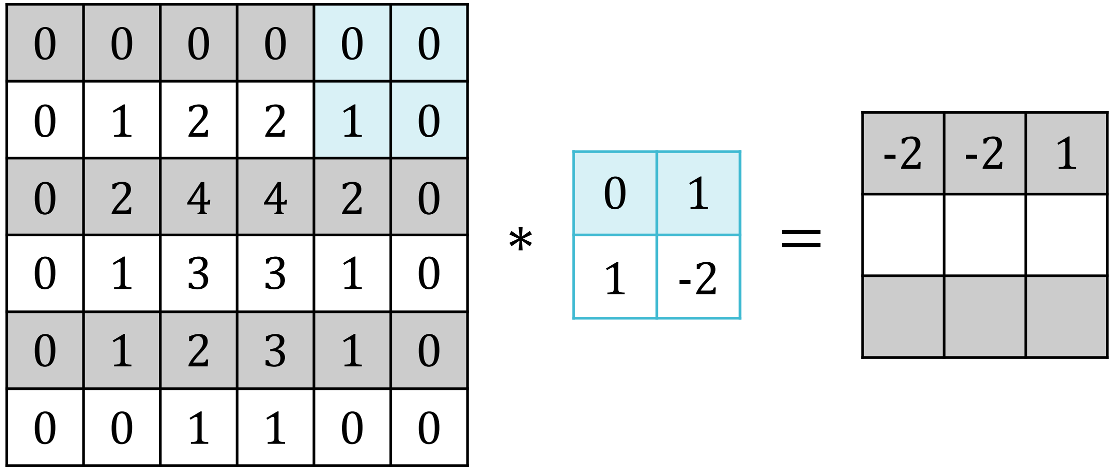
=

-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

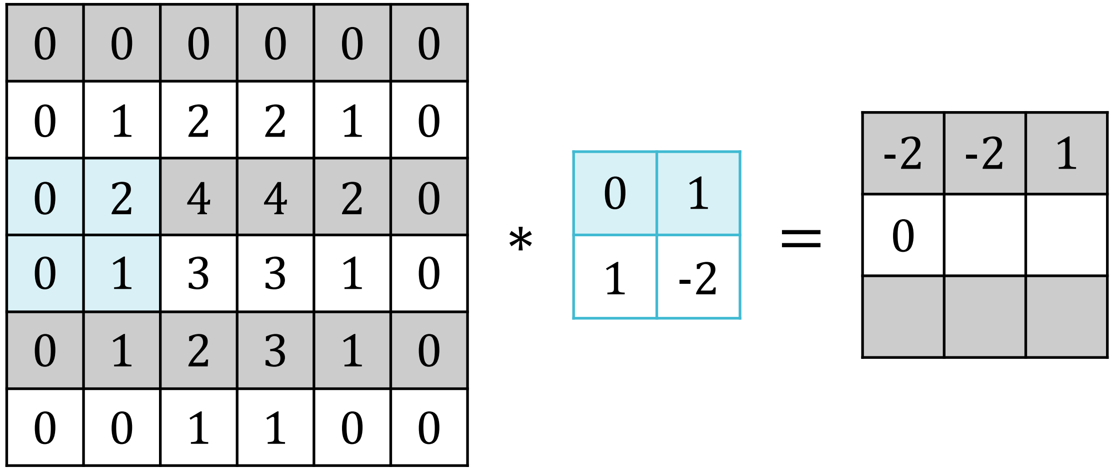


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ & & \\ & & \end{bmatrix}$$

Downsampling: Stride

- Only apply the convolution to some subset of the image

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



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

Downsampling: Stride

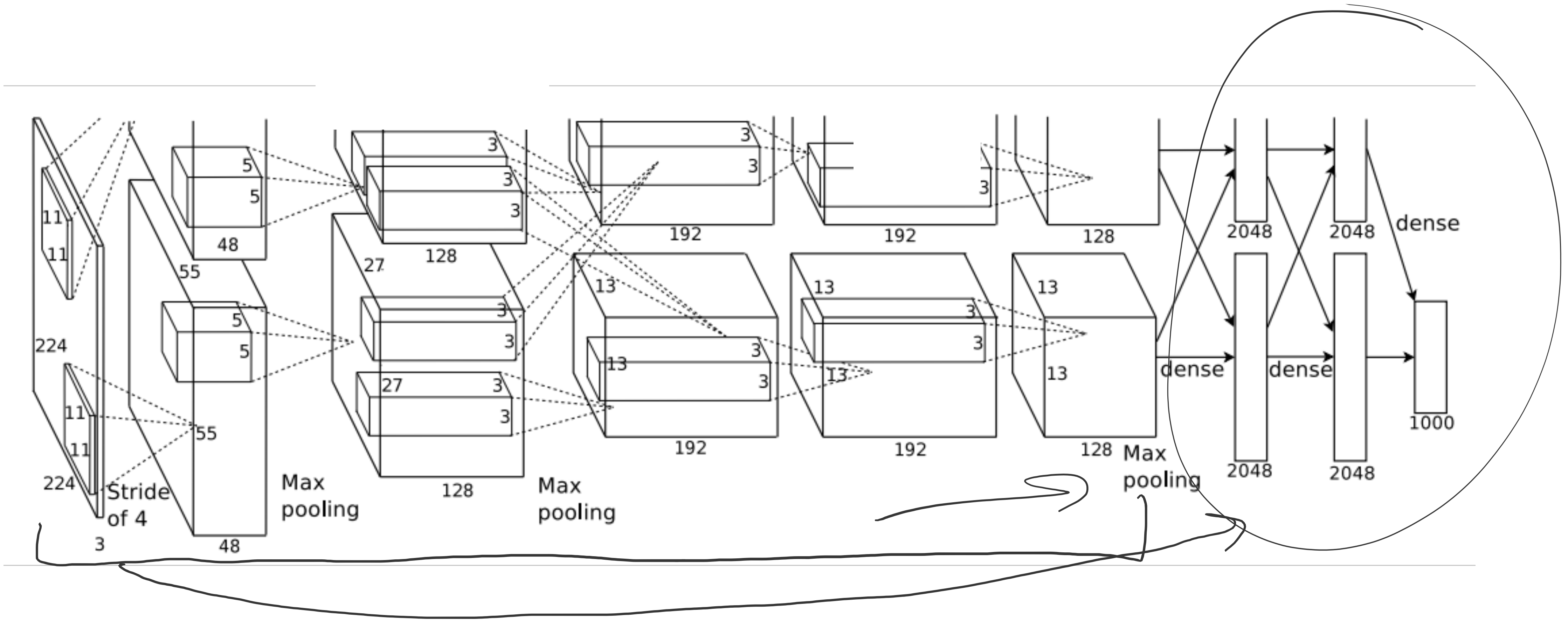
- Only apply the convolution to some subset of the image

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

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

- 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



AlexNet (Krizhevsky et al., 2012)

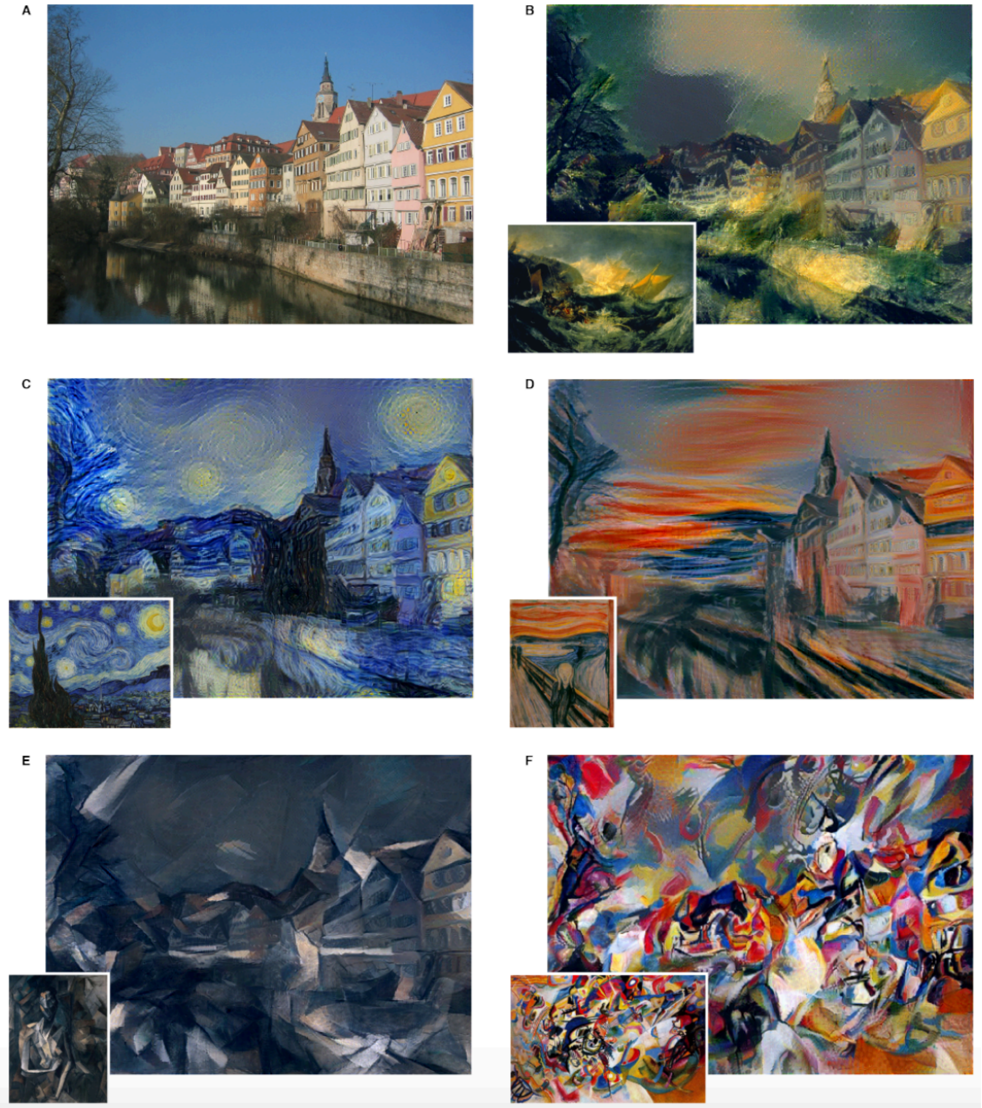
Source: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Cool Example: Style Transfer

Style Transfer

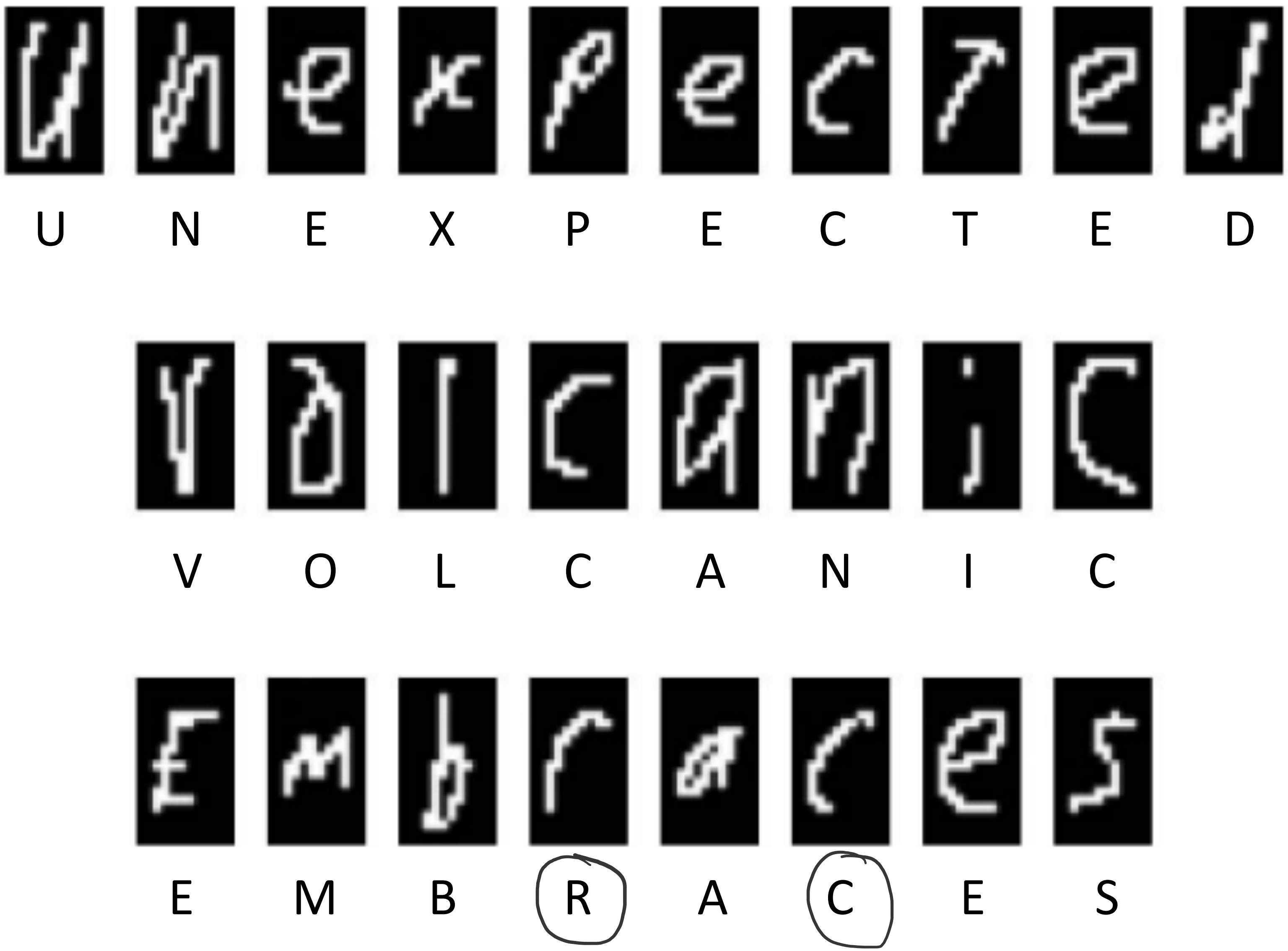
- Basic idea:
  - Learn a content representation for an image using convolutional layers (multiple)
  - Learn a style representation for an image using convolutional layers
  - Compute an image that jointly minimizes the distance from the content image's content representation and the style image's style 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



U N E X P E C T E D

V O L C A N I C

E M B R A C E S

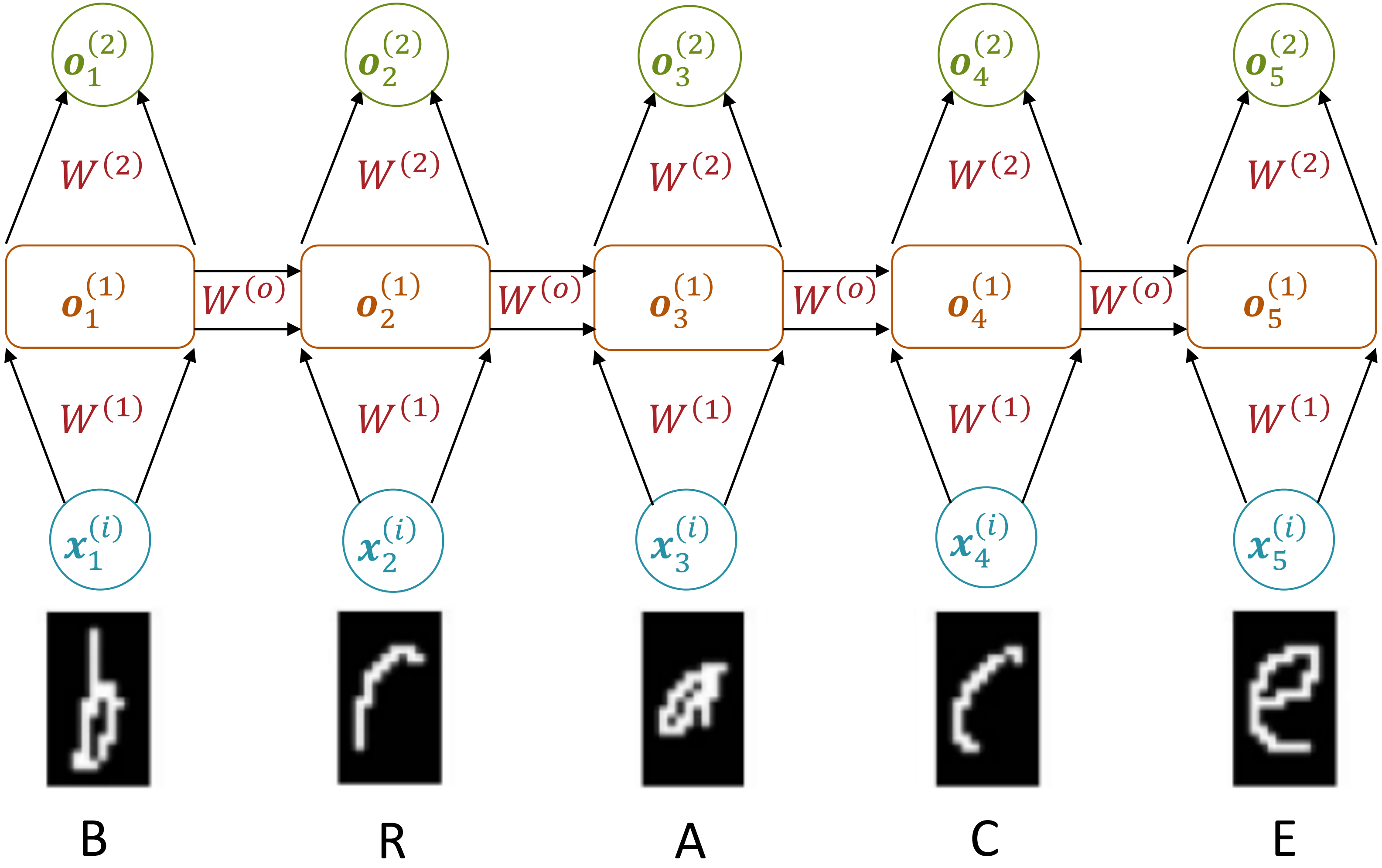
Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6319312

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.

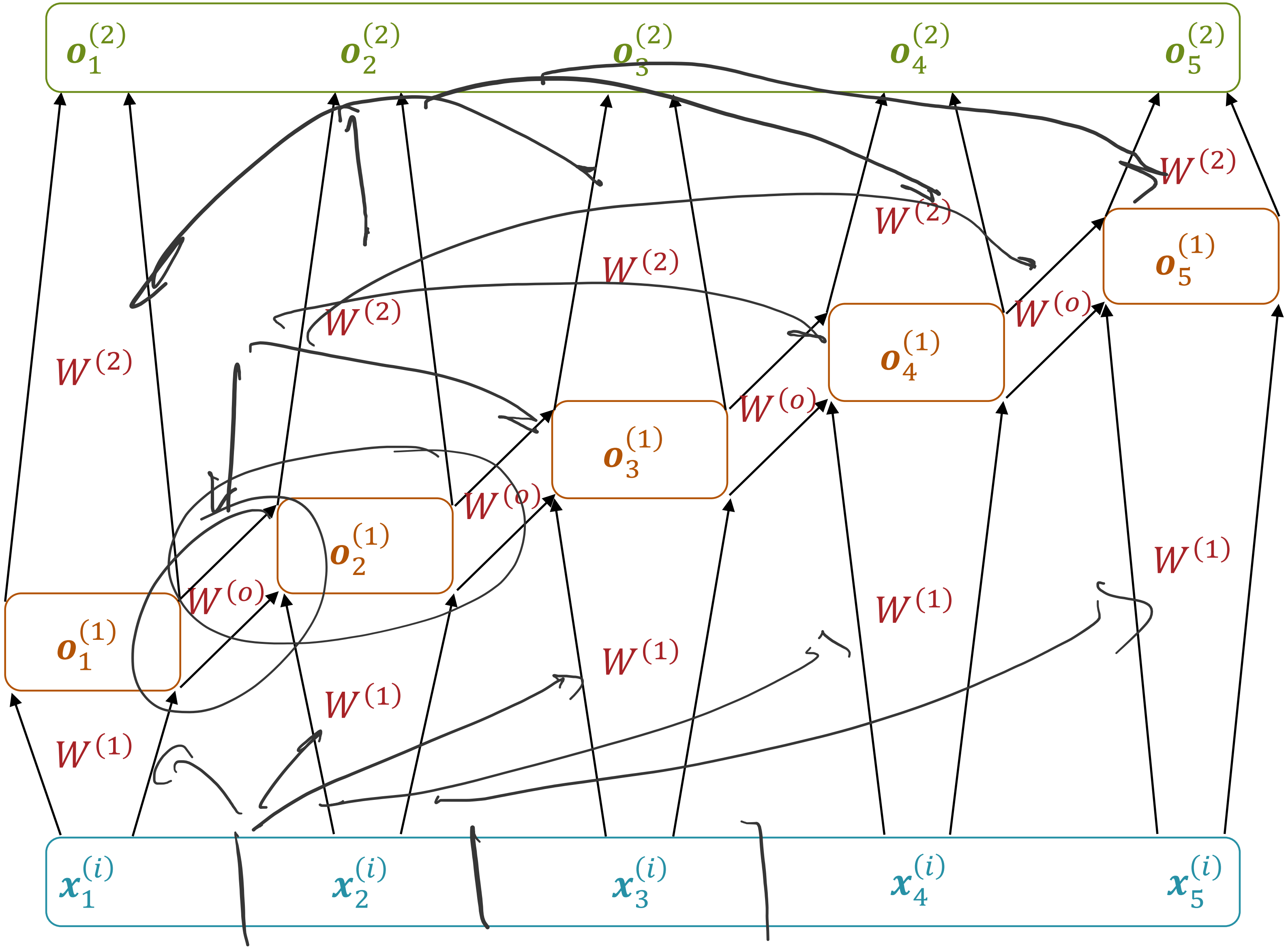
Recurrent Neural Networks

$o_t^{(1)} = \left[1, \theta\left(W^{(1)}x_t^{(i)} + W^{(o)}o_{t-1}^{(1)}\right)\right]^T$ and $o_t^{(2)} = \theta\left(W^{(2)}o_t^{(1)}\right)$



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

Unrolling Recurrent Neural Networks

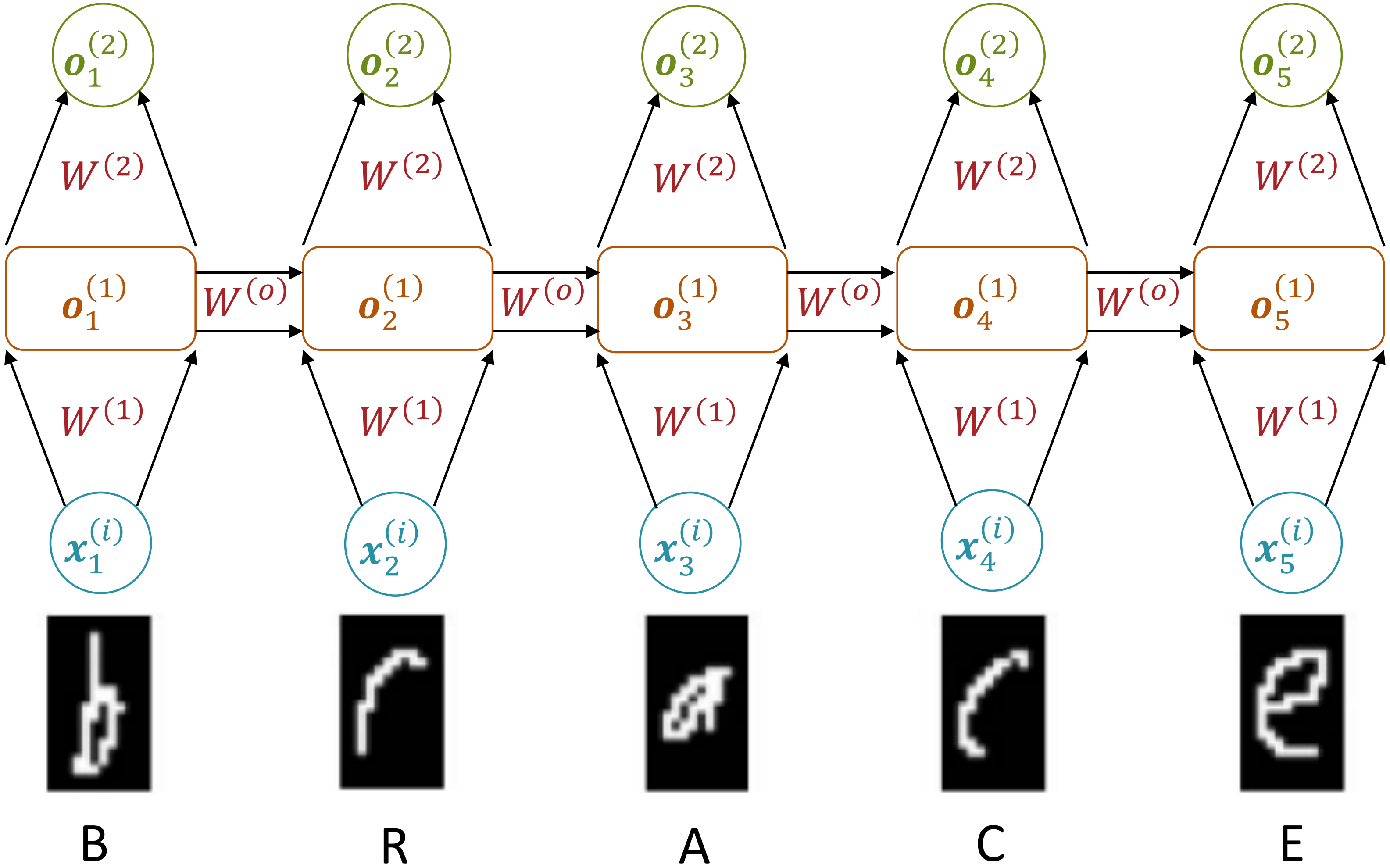


Unrolling Recurrent Neural Networks

- An RNN can be expressed as a feed forward neural network where:
  1. The hidden layers are connected directly to some nodes in the input and output layers
  2. Many of the weights have the same value
- Many fewer weights than a fully connected network!
- Can be trained using gradient descent/backpropagation; sometimes referred to as “backpropagation through time”

Recurrent Neural Networks

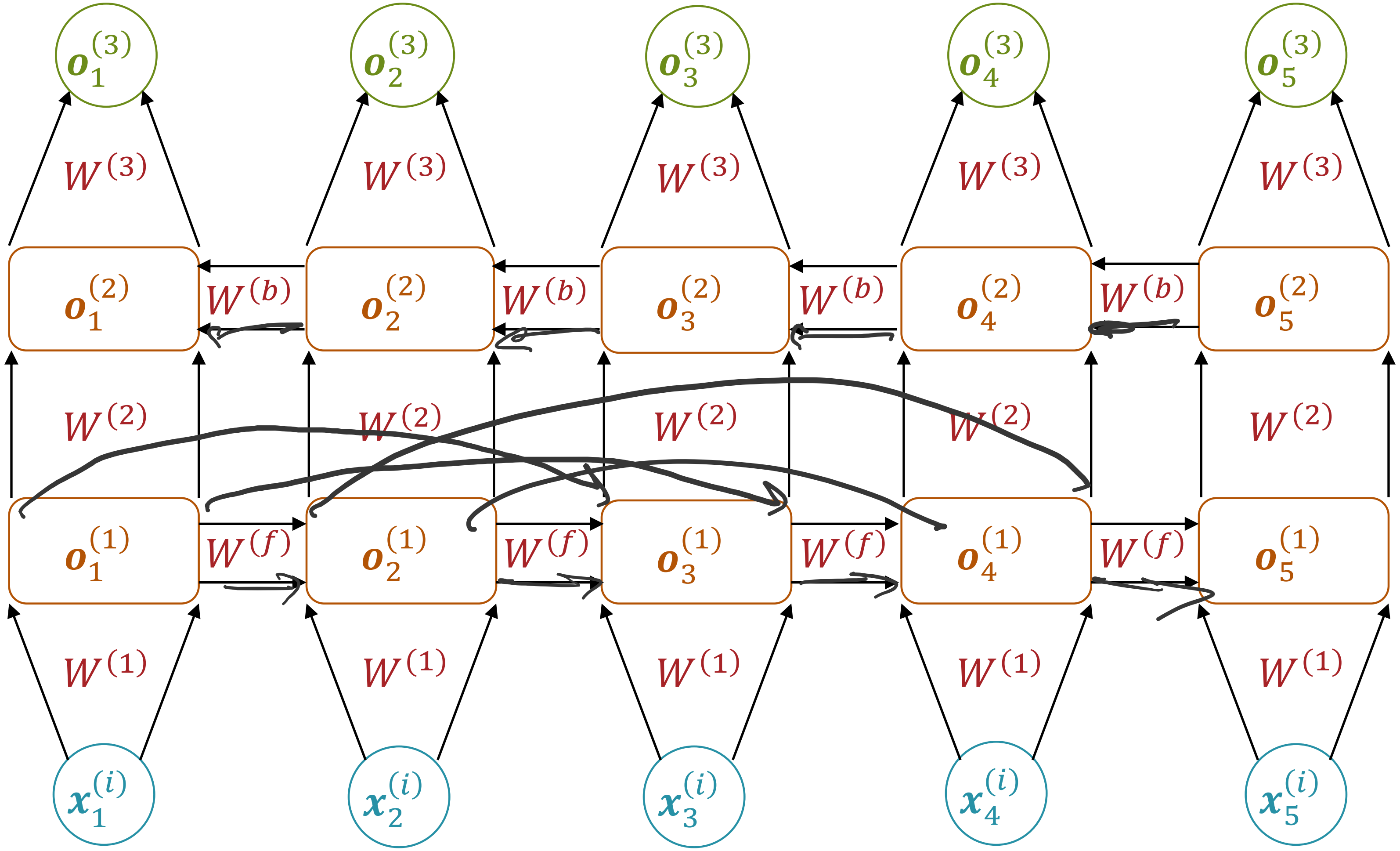
$$o_t^{(1)} = \left[1, \theta\left(W^{(1)} x_t^{(i)} + W^{(o)} o_{t-1}^{(1)}\right)\right]^T \text{ and } o_t^{(2)} = \theta\left(W^{(2)} o_t^{(1)}\right)$$



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

Bidirectional Recurrent Neural Networks

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

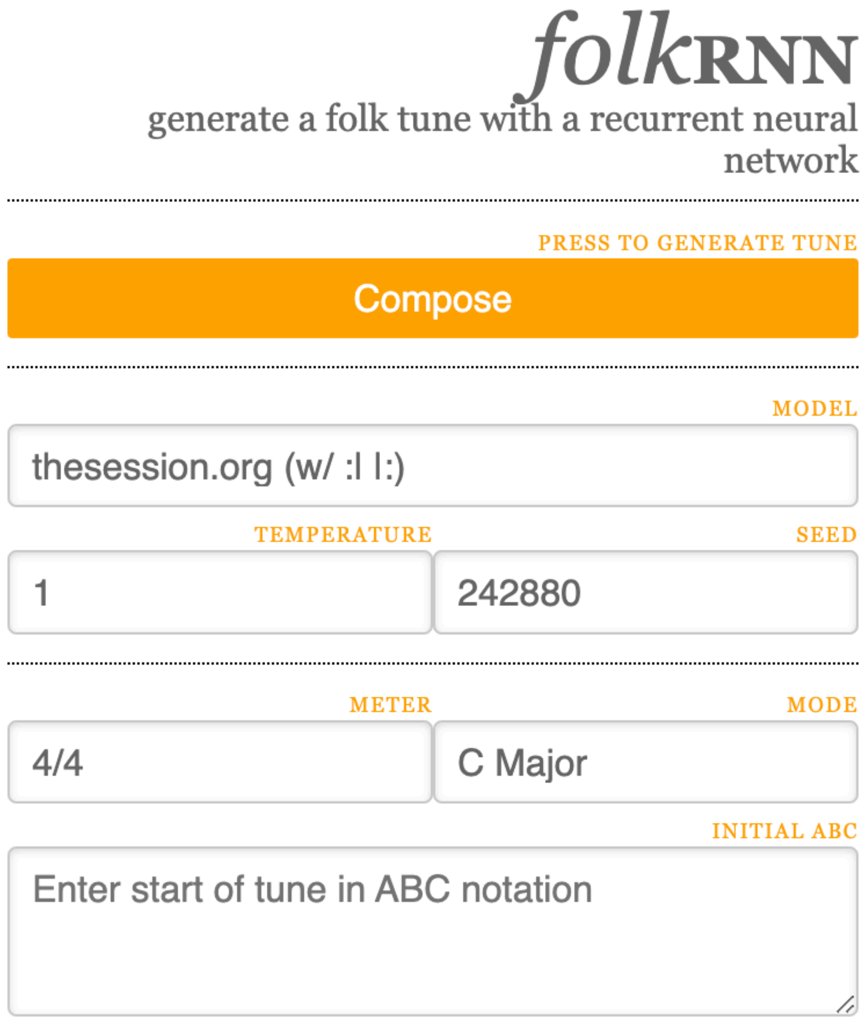


Cool Example: Music Composition

Music Composition

- Basic idea:
  - Music can be represented as a sequence of tokens i.e., notes or chords
  - In general, music, especially compositions from certain genres, have a predictable (learnable) structure
  - Generate a piece by sampling from the distribution over subsequent tokens given the previous tokens, learned by minimizing the training cross-entropy loss
  - For complete details, see https://arxiv.org/pdf/1604.08723.pdf

Cool Example: Music Composition



folkRNN

generate a folk tune with a recurrent neural network

PRESS TO GENERATE TUNE

MODEL: thesession.org (w/ :| |:)

TEMPERATURE: 1

SEED: 242880

METER: 4/4

MODE: C Major

INITIAL ABC: Enter start of tune in ABC notation

Source: https://folkrnn.org/tune/102645

Key Takeaways

- MLPs and neural networks of sufficient depth are universal approximators
- Deep learning
  - Convolutional neural networks use convolutions to learn macro-features
  - Recurrent neural networks use contextual information to reason about sequential data
  - Both can be thought of as slight modifications to the fully-connected feed-forward neural network
    - Both can still be learned using (stochastic) gradient descent