School of
COMPUTING
Clemson® University

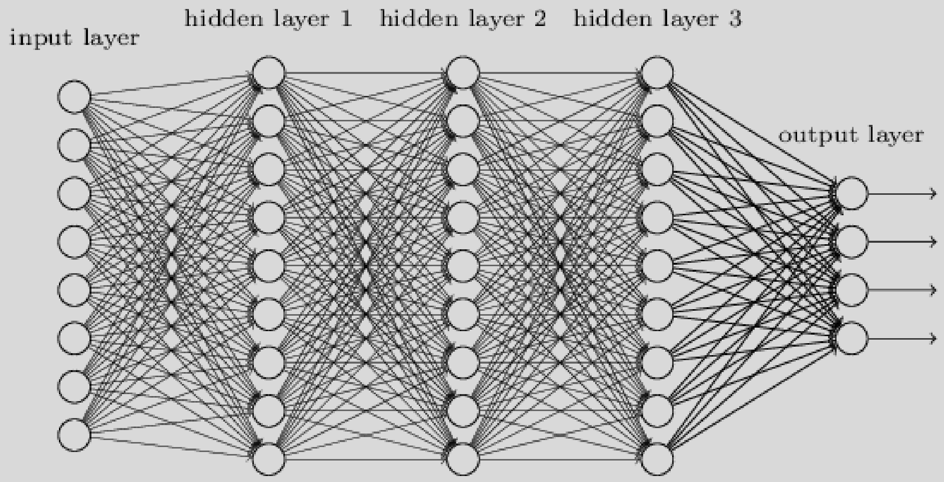
Introduction to Convolutional and Recurrent Neural Networks

Aaron J. Masino, PhD
Associate Professor, School of Computing

Outline

- Convolutional Neural Network
- Recurrent Neural Network (RNN)
  - Vanilla RNN
  - LSTM
  - GRU

Convolutional Neural Network (CNN)



Motivations for CNNs

- Spatial invariant feature learning – some data types (e.g., images) have patterns that are informative regardless of their location
- Learning parameter efficiency
  - Variance (overfitting) – we generally prefer models with fewer parameters to mitigate overfitting
  - Training efficiency – models with fewer parameters tend to be easier to train
- Fully connected ANNs lack these properties

Consider images

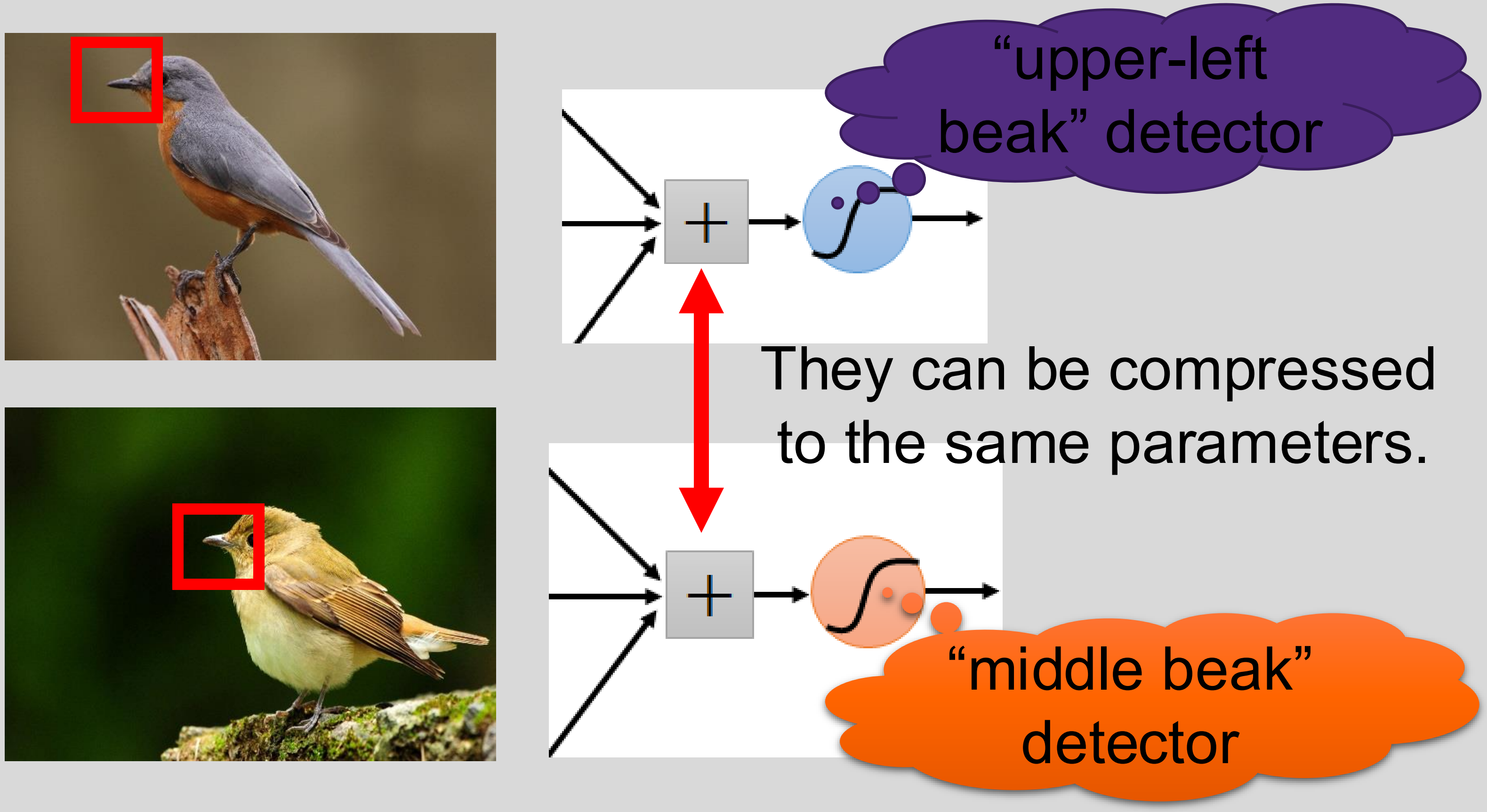
Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters?

"beak" detector

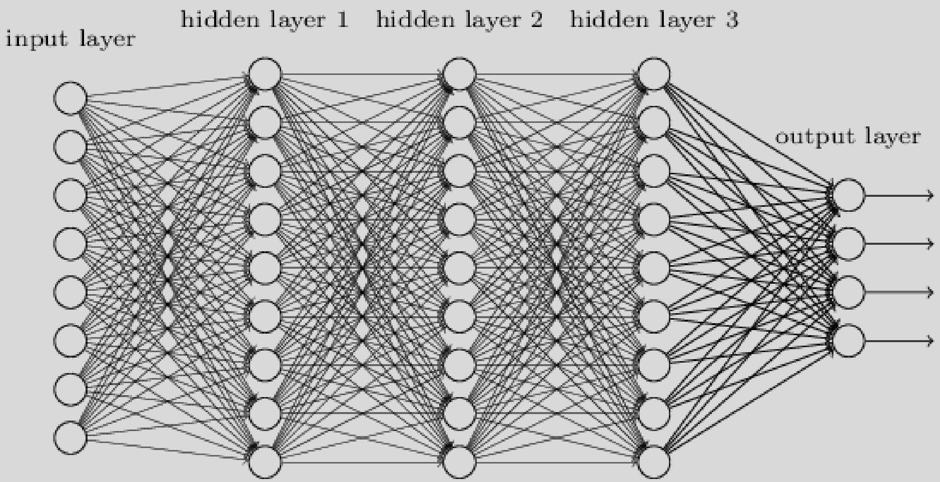
Same pattern appears in different places: They can be compressed!

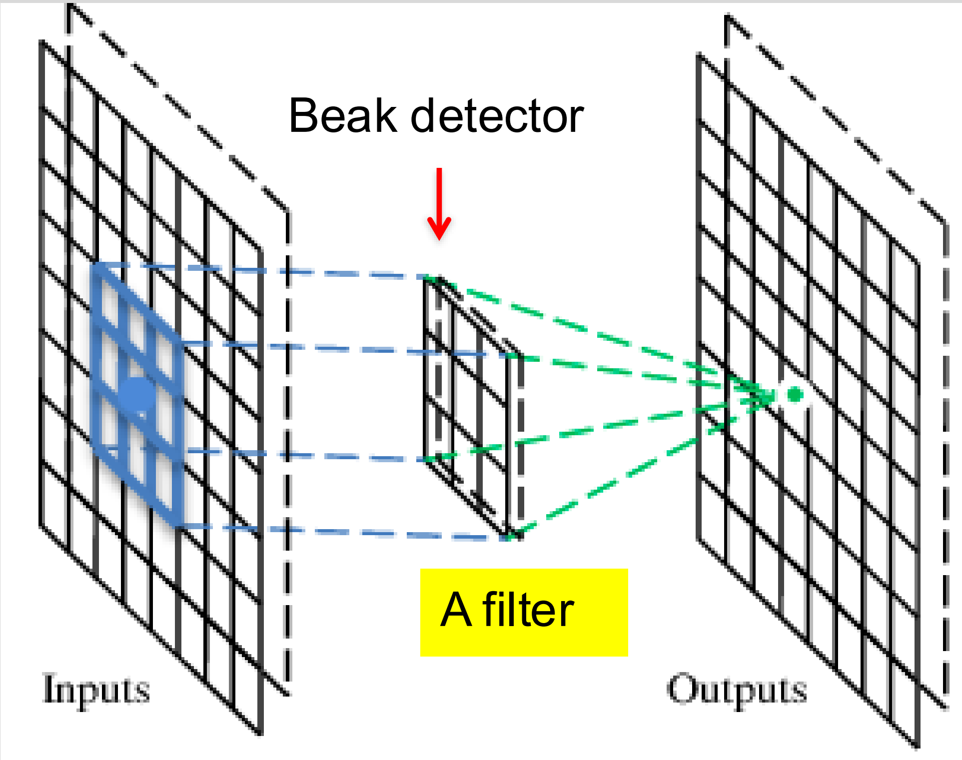
Do we need a separate detector for every location?



Can we reuse our “detector”

- We want an architecture that will allow us to apply the detector over the entire input
- Fully connected layer does NOT do this
- Enter *convolution*





School of COMPUTING
Clemson University

Convolution Operator

$$\begin{bmatrix} X_{0,0} & X_{0,1} & X_{0,2} \\ X_{1,0} & X_{1,1} & X_{1,2} \\ X_{2,0} & X_{2,1} & X_{2,2} \end{bmatrix} * \begin{bmatrix} k_{0,0} & k_{0,1} \\ k_{1,0} & k_{1,1} \end{bmatrix} = \begin{bmatrix} y_{0,0} & y_{0,1} \\ y_{1,0} & y_{1,1} \end{bmatrix}$$

Input | Filter | Output

Convolution: Filter, $\boldsymbol{k}$, is convolved with input by sliding it across the input with stride, s. At each location of overlap, the dot product of the filter and input region is computed to from the output.

Output for 2 X 2 kernel with stride of 1

$$y_{0,0} = k_{0,0}x_{0,0} + k_{0,1}x_{0,1} + k_{1,0}x_{1,0} + k_{1,1}x_{1,1}$$

$$y_{0,1} = k_{0,0}x_{0,1} + k_{0,1}x_{0,2} + k_{1,0}x_{1,1} + k_{1,1}x_{1,2}$$

$$y_{1,0} = k_{0,0}x_{1,0} + k_{0,1}x_{1,1} + k_{1,0}x_{2,0} + k_{1,1}x_{2,1}$$

$$y_{0,0} = k_{0,0}x_{1,1} + k_{0,1}x_{1,2} + k_{1,0}x_{2,1} + k_{1,1}x_{2,2}$$

Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Dot product → 3 -1

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

Color image: RGB 3 channels

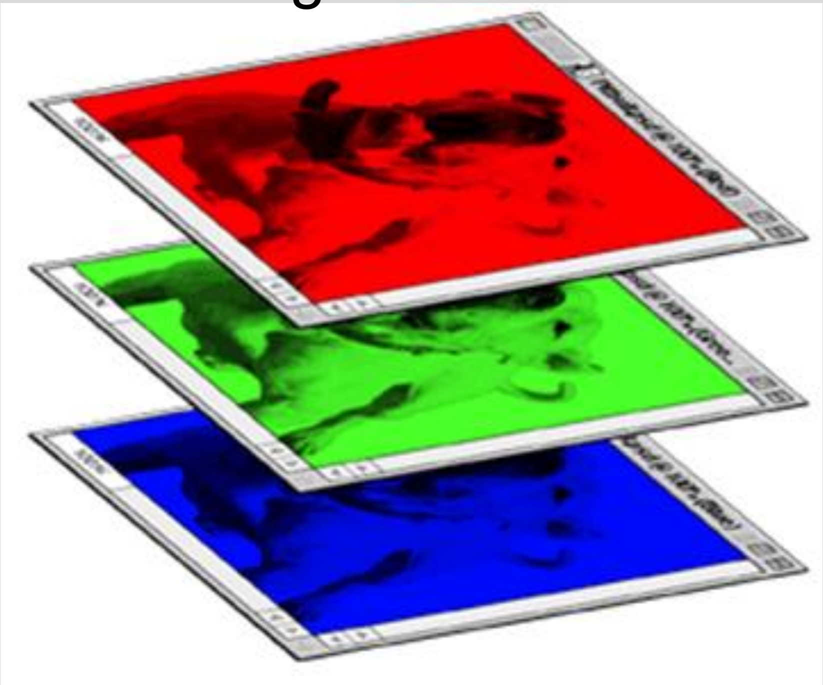
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

Color image



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

More efficiency with pooling

Subsampling pixels will not change the overall scene

bird

Subsampling

bird

We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

Max Pooling

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

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

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

Convolution and Pooling

Original Image

1	1	1	1	1	1
1	1	1	1	1	1
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0
0	0	1	1	0	0

Feature Map

x1	x1	x1
x1	x1	x1
x0	x0	x0

Feature Map

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Max Pooling

Average Pooling

Sum Pooling

Max pooling is common, but average and sum pooling are also used

The whole CNN

Convolution

Non-linear Activation

Max Pooling

Convolution

Non-linear Activation

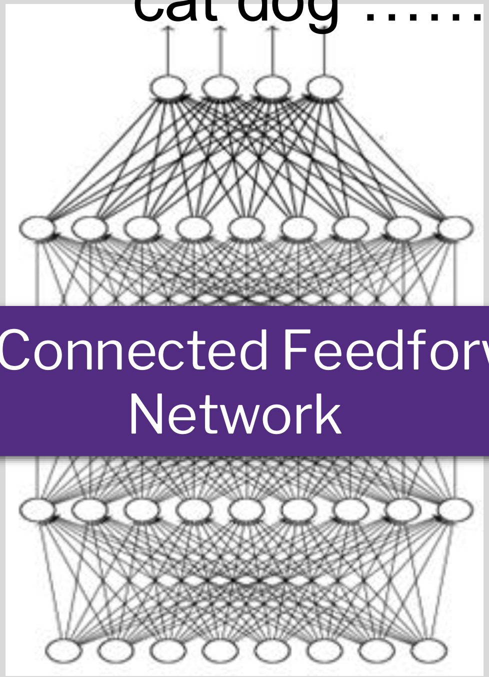
Max Pooling

Many kernels

Can repeat many times

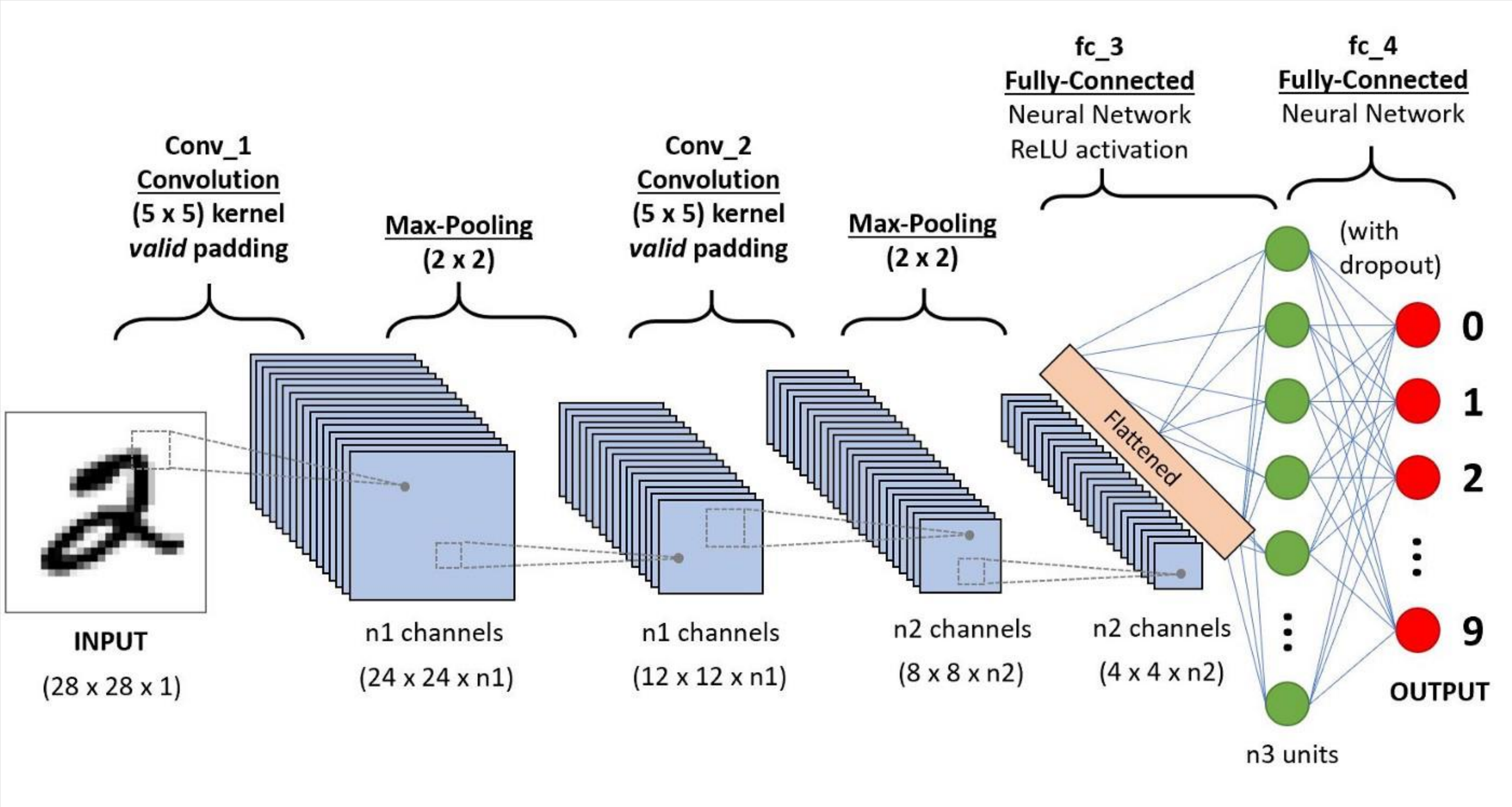
Flattened

Fully Connected Feedforward Network



Class Probabilities

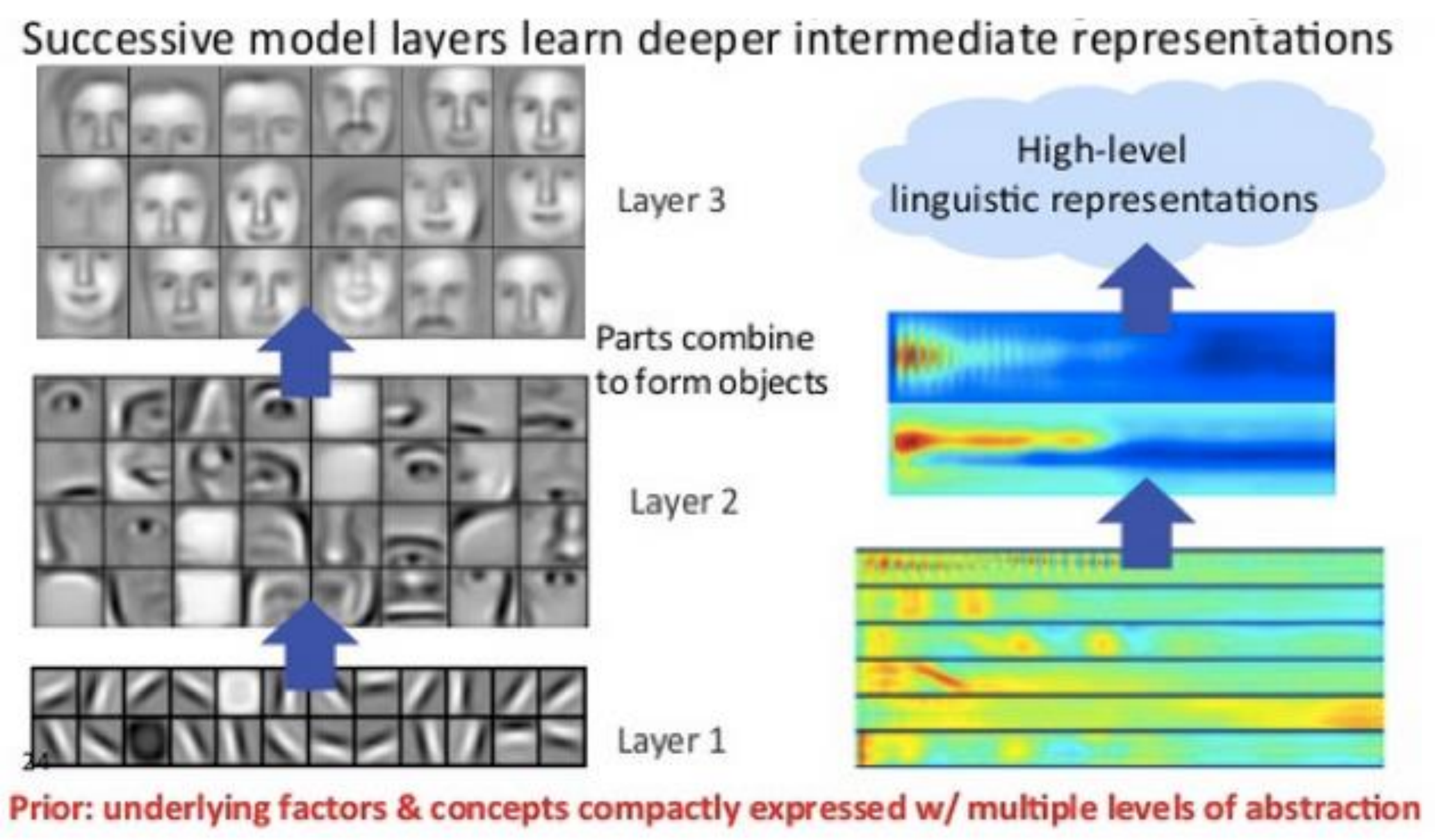
Deep CNN Architecture



- Multiple convolution layers stacked in sequence
- Each layer can have an arbitrary number of convolution kernels
- Pooling layers for down sampling

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Feature hierarchy

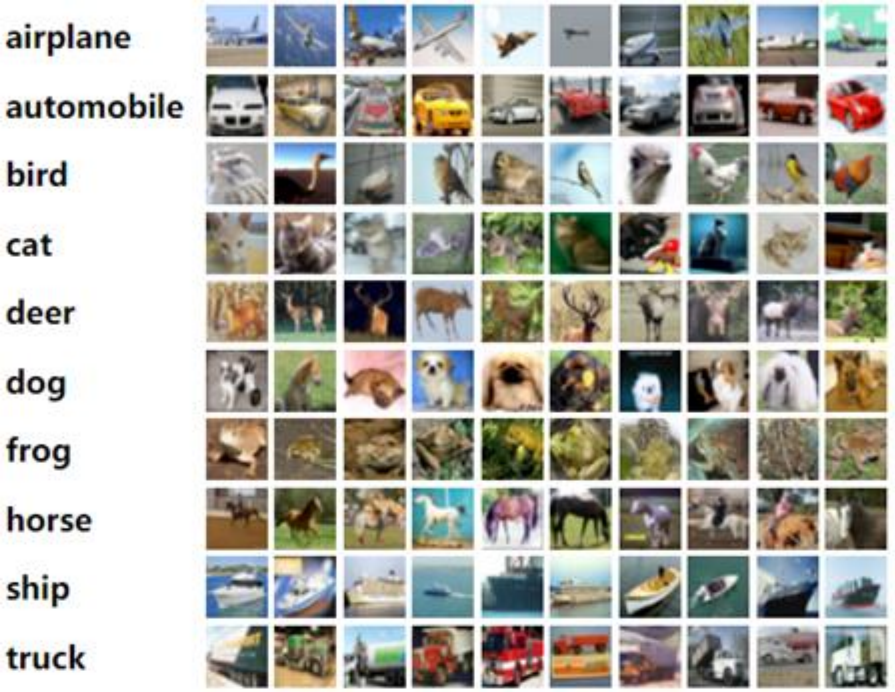


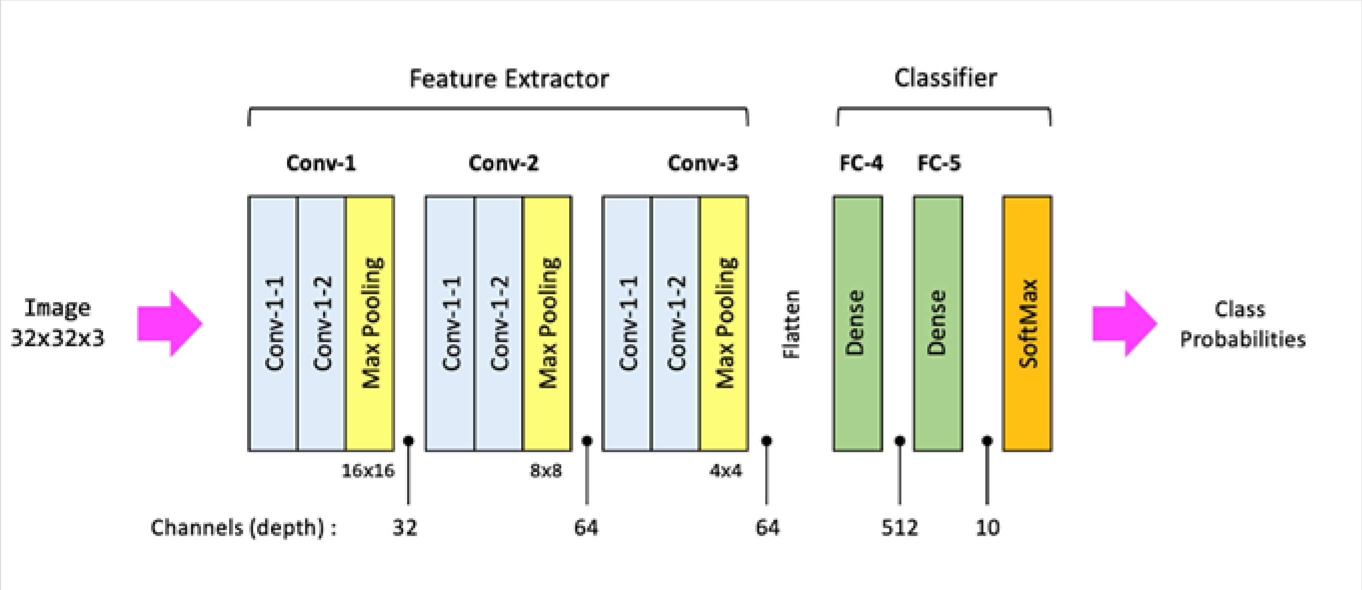
- Early layers tend to learn simple features such as edges
- Feature learning is spatially invariant
- Later layers learn features composed of simpler features to recognize more complex patterns
- Final layers are effectively "object models"

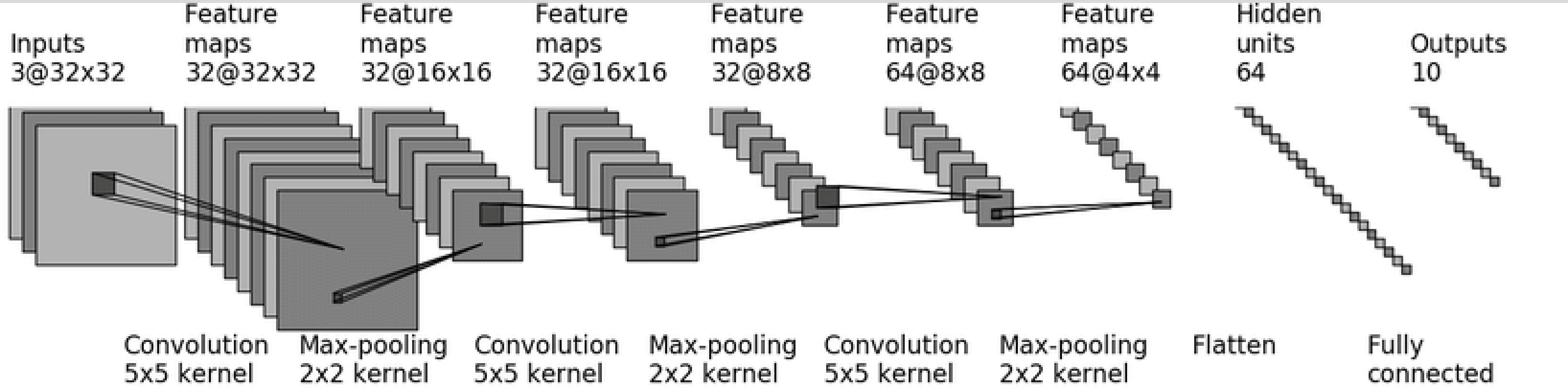
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/handouts/CS224N_DeepNLP_Week7_lecture1.pdf

CNN for Image Classification

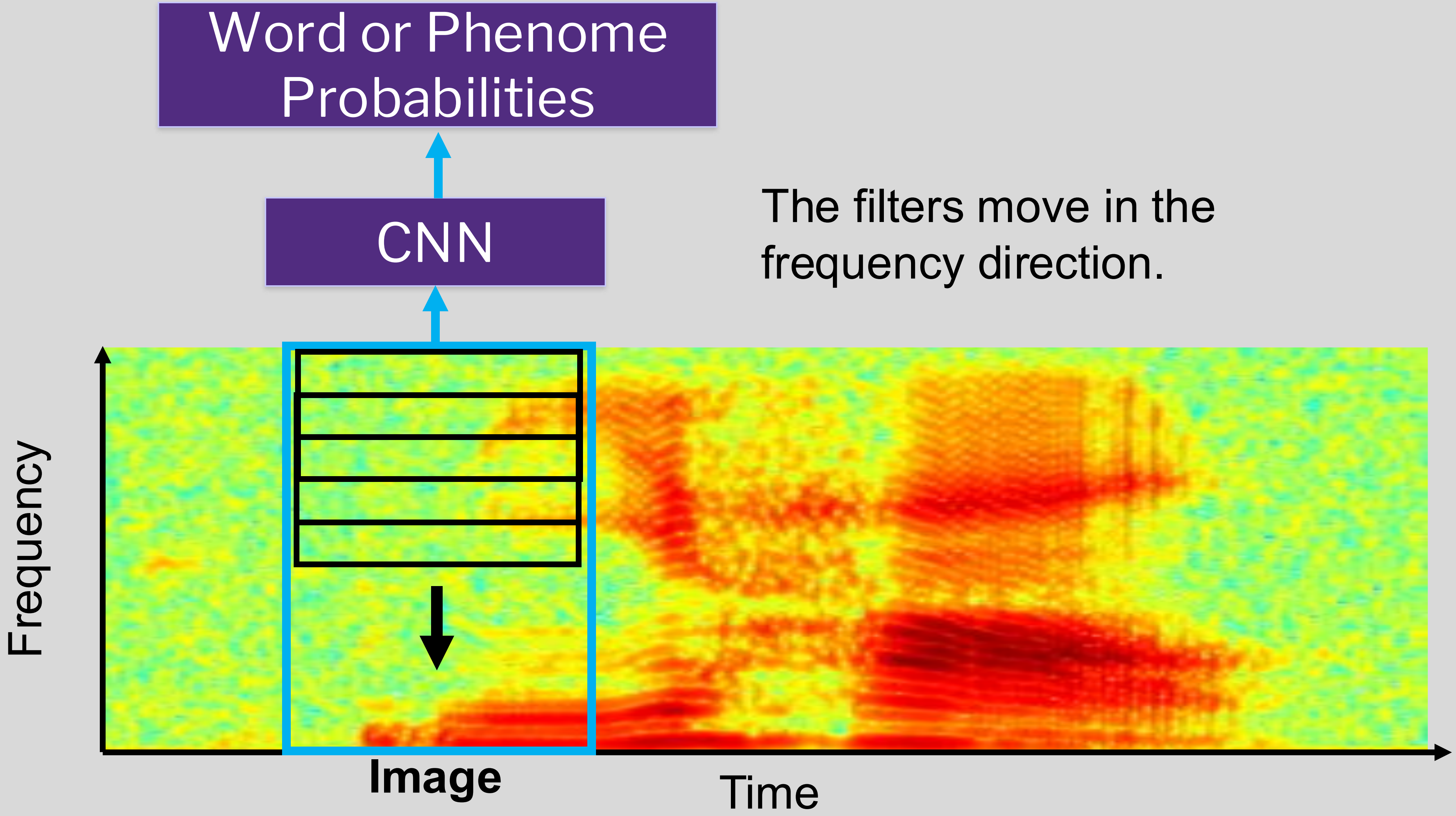
Example using the CIFAR10 dataset, 32 x 32







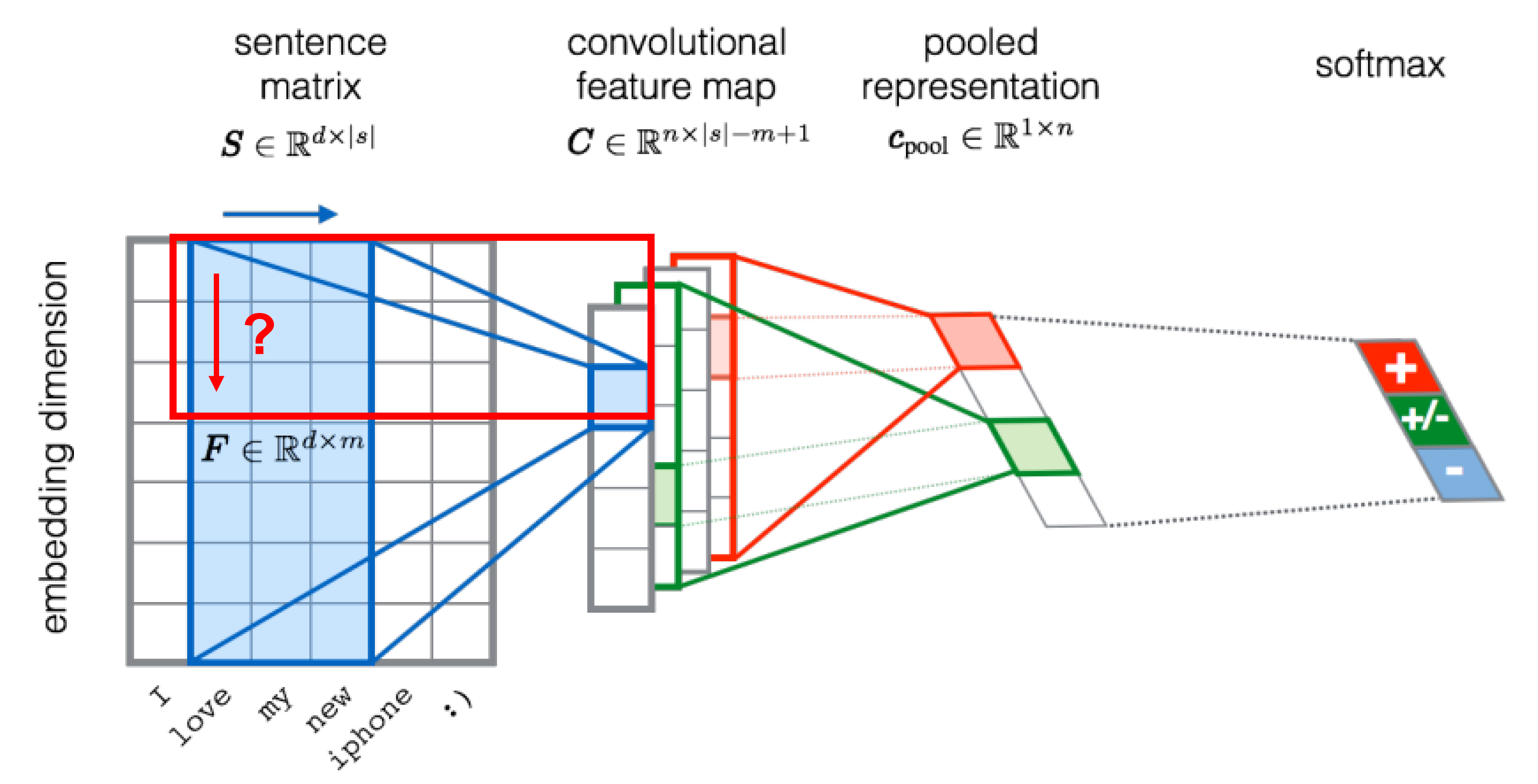
CNN for speech recognition



The filters move in the frequency direction.

Spectrogram

CNN in text classification



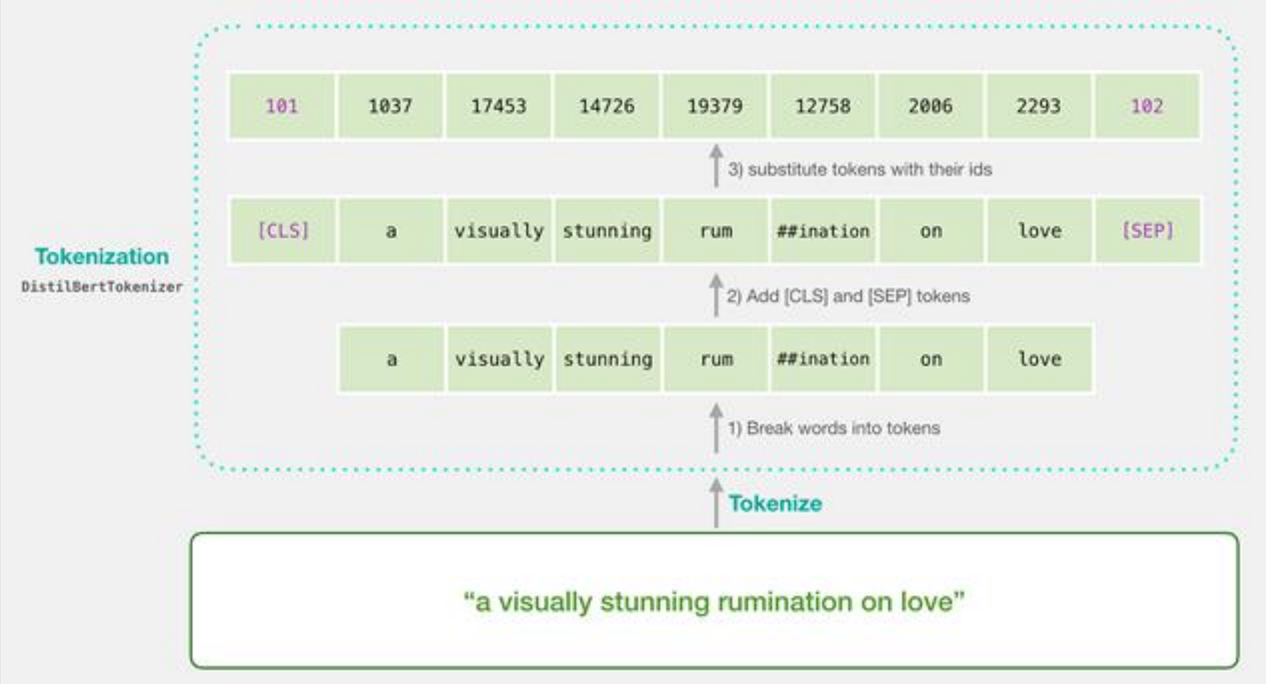
Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

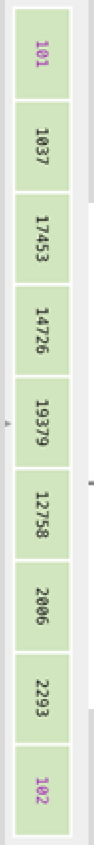
Recurrent Neural Networks (RNN)

ANN vs RNN

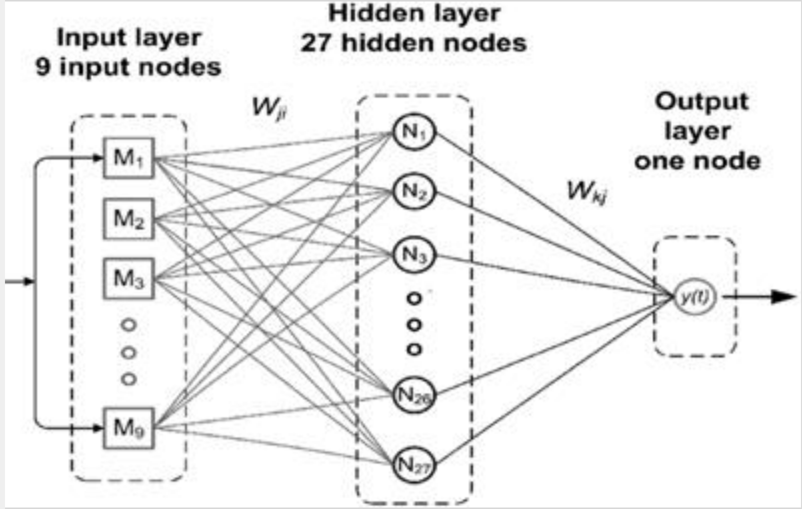
ANNs do not work well for data where the relative order of inputs is informative and dynamic such as text (sequence of words) or videos (sequence of images)

We can divide sentences in tokens





ANN will intermix the features, missing contextual information based in their position.



Sequence Learning Applications

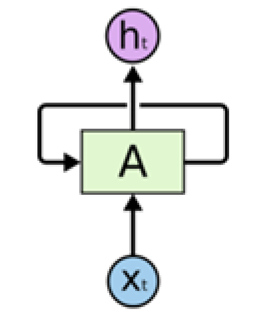
RNNs can be applied to various type of sequential data to learn the temporal patterns that are informative to a given task.

Many applications

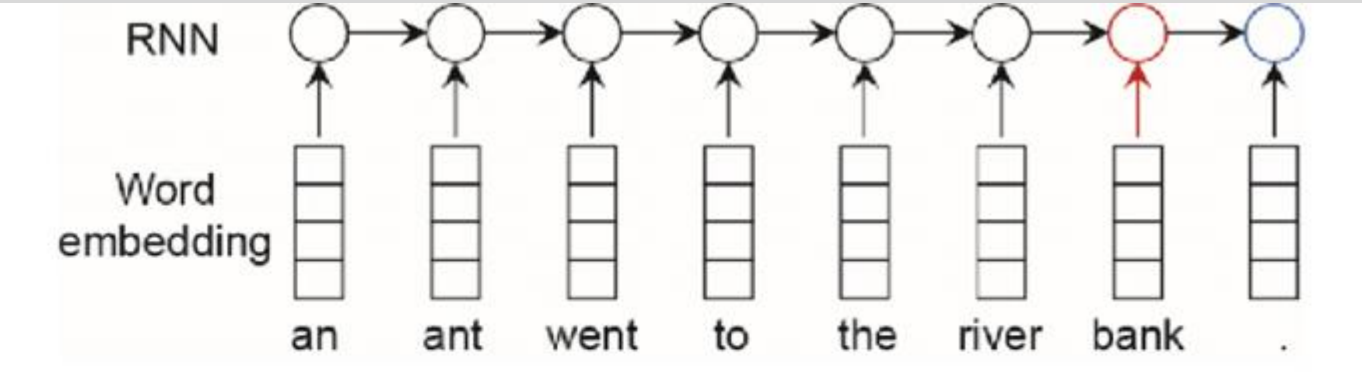
- Time-series data (e.g., stock price) ⟶prediction
- Raw sensor data (e.g., signal, voice, handwriting) ⟶sequence of class labels
- Text document ⟶ single label (document level), multiple labels (document partition labels)
- Image and video ⟶ Text description (e.g., captions, scene interpretation)

Recurrent Neural Network (RNN)

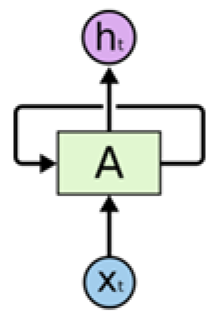
- RNN units (or cells) add loops to retain prior information
- Inputs are processed in sequential order



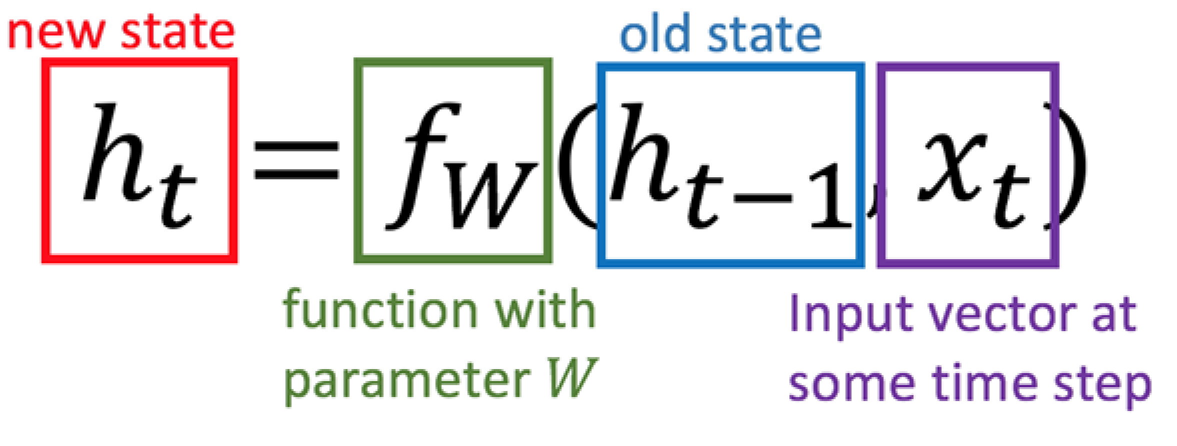
Recurrent Neural Networks have loops.



Recurrent Neural Networks

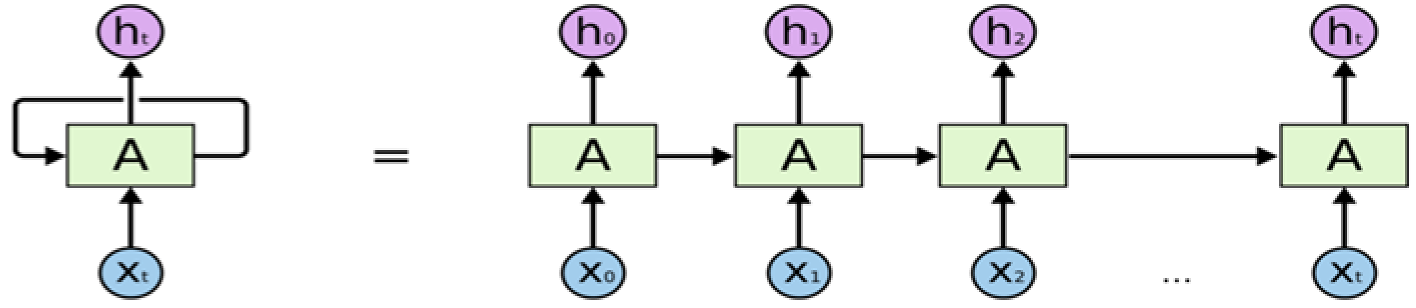


Recurrent Neural Networks have loops.



In the above diagram, a chunk of neural network, $\boldsymbol{A} = \boldsymbol{f_W}$, looks at some input $\boldsymbol{x_t}$ and outputs a value $\boldsymbol{h_t}$. A loop allows information to be passed from one step of the network to the next.

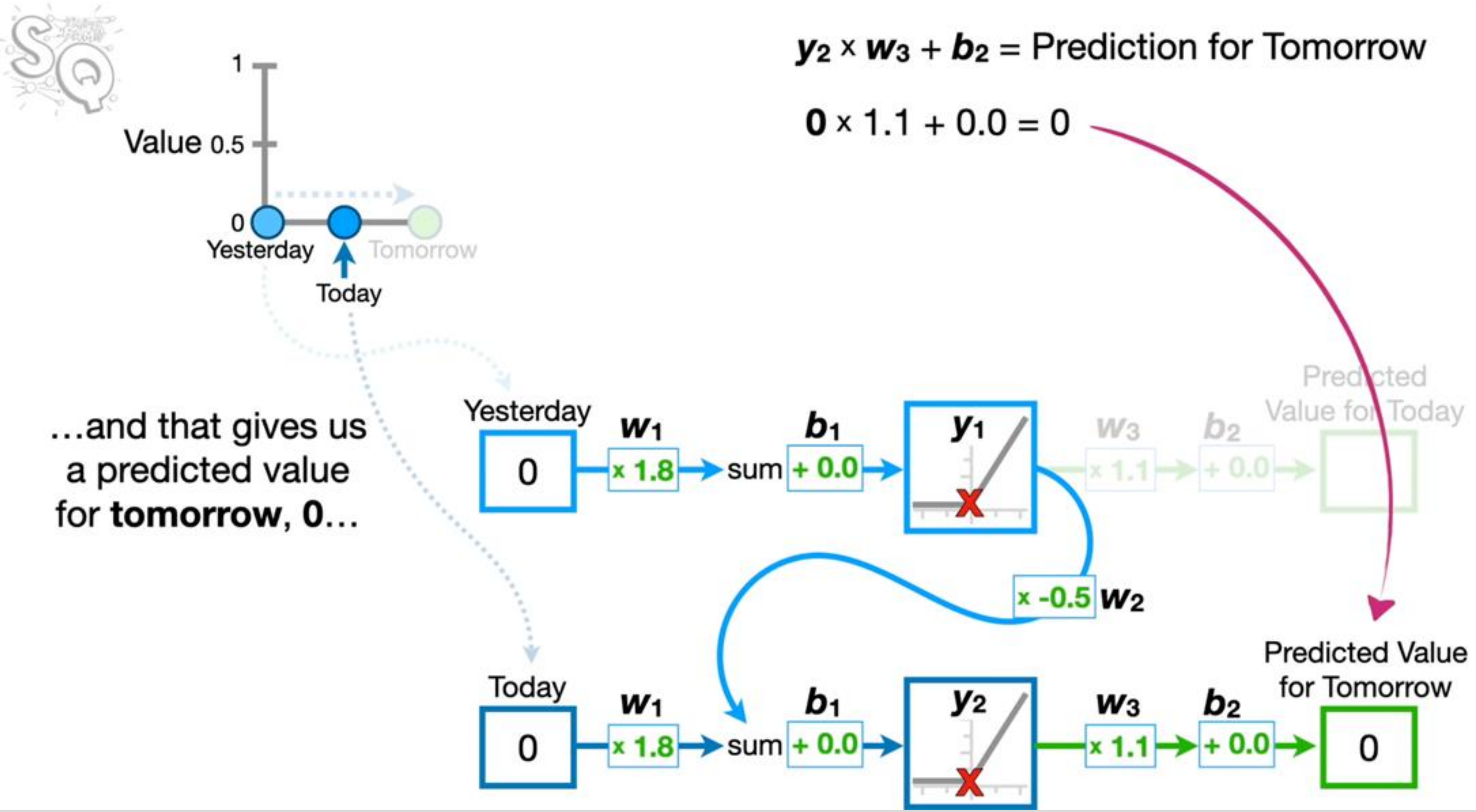
“Unrolling” Recurrent Neural Networks



An unrolled recurrent neural network.

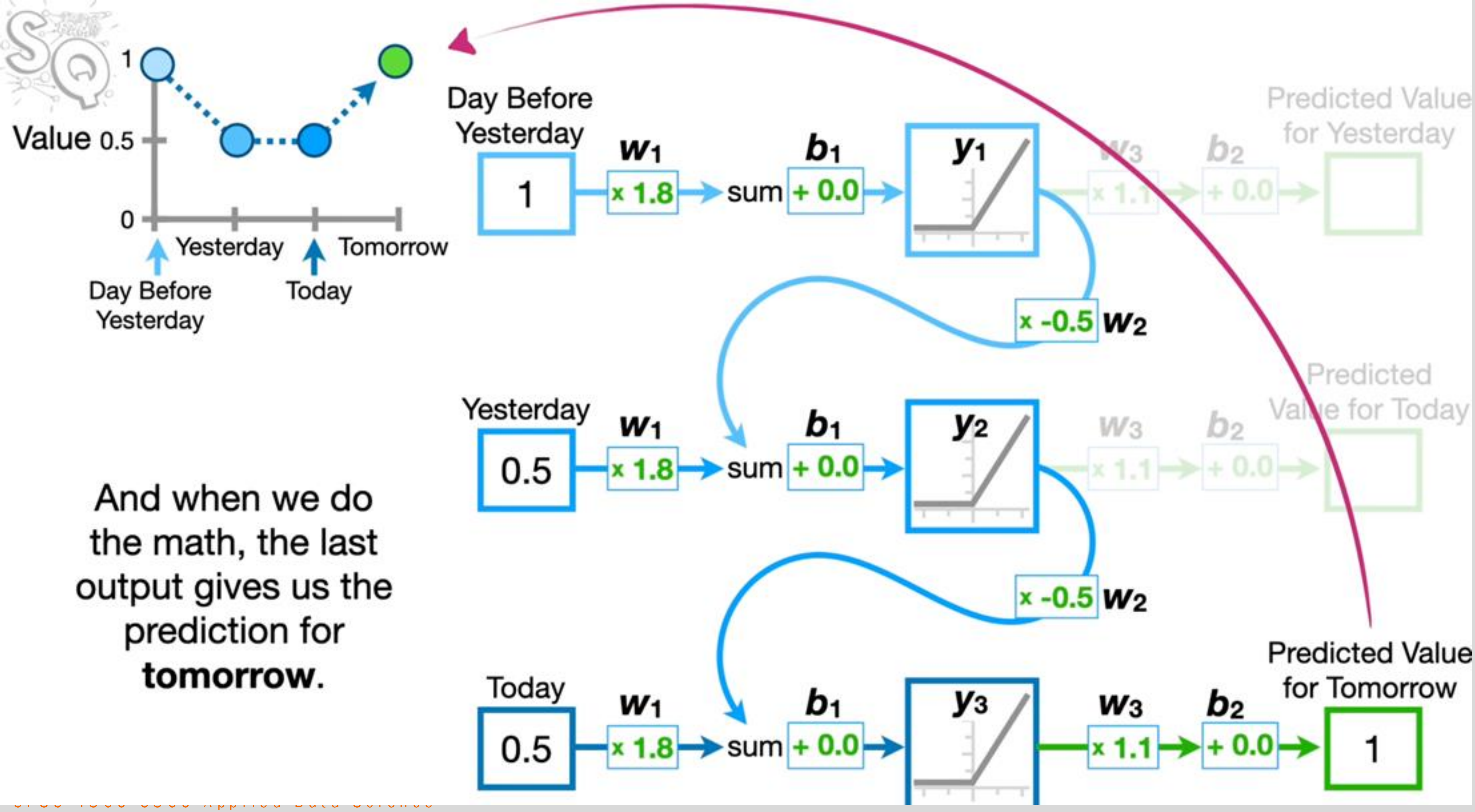
A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The diagram above shows what happens if we **unroll the loop**.

“Message” passing in RNNs

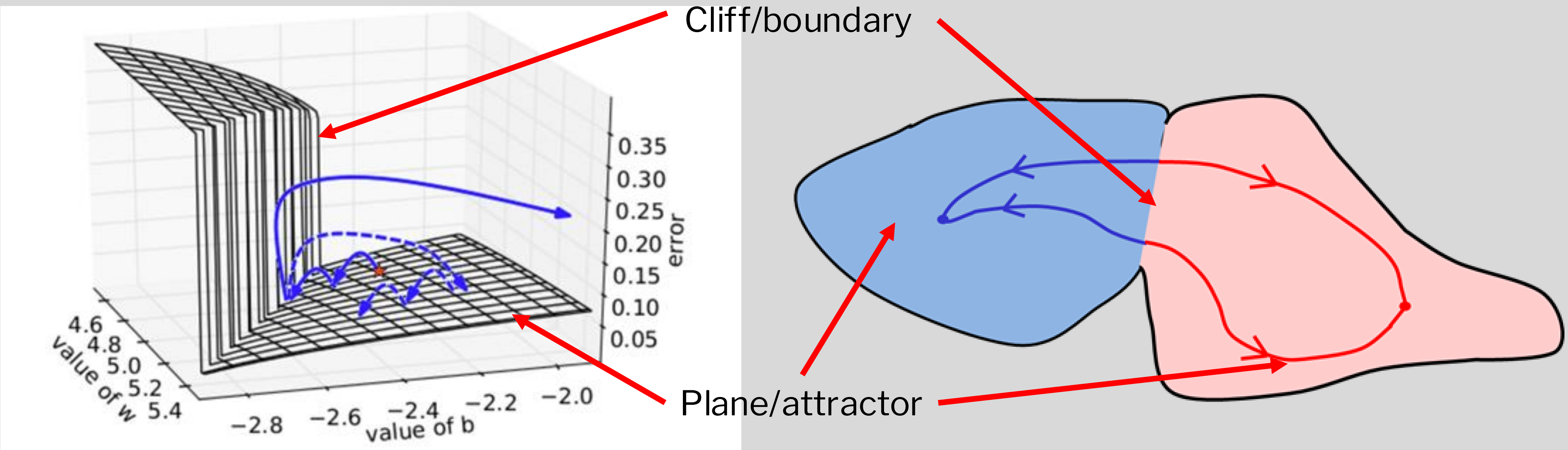


https://www.youtube.com/watch?v=AsNTP8Kwu80

“Message” passing in RNNs



Exploding and Vanishing Gradients



Exploding: If we start almost exactly on the boundary (cliff), tiny changes can make a huge difference.

Vanishing: If we start a trajectory within an attractor (plane, flat surface), small changes in where we start make no difference to where we end up.

Both cases hinder the learning process.

Exploding gradient problem

Large numbers of W2 can rapidly explode.

And that means the first input value is amplified **16** times before it gets to the final copy of the network.

$$\textbf{Input}_1 \times 2 \times 2 \times 2 \times 2$$

$$= \textbf{Input}_1 \times 2^4 = \textbf{Input}_1 \times 16$$

$$= \textbf{Input}_1 \times w_2^{\text{Num. Unroll}}$$

Vanishing gradient problem

Small values of W2 can rapidly vanish

And that means the first input value is amplified **16** times before it gets to the final copy of the network.

$$Input_1 \times \left(1/2\right)^4 = \frac{Input_1}{16}$$

Networks with Memory

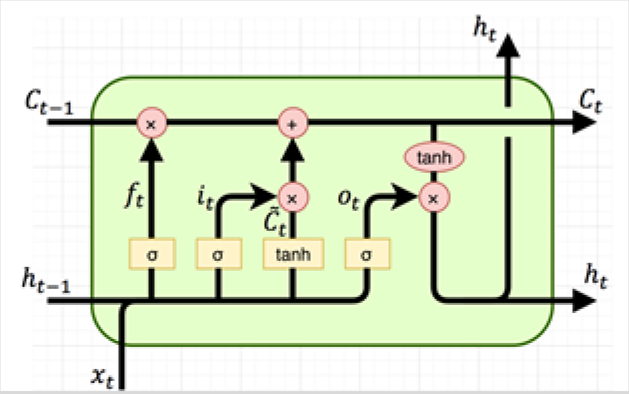
Standards (aka vanilla) RNN operates in a “multiplicative” way leading to vanishing/exploding gradients

Two recurrent cell designs have proposed and widely adopted:

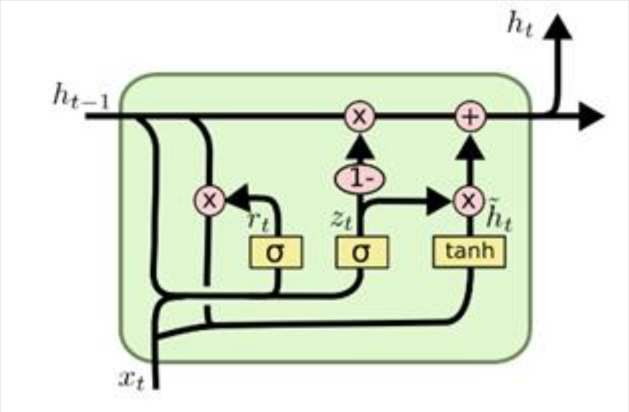
- Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997)
- Gated Recurrent Unit (GRU) (Cho et al. 2014)

Both designs process information in an “additive” way with gates to control information flow.

- Sigmoid gate outputs numbers between 0 and 1, describing how much of each component should be let through.

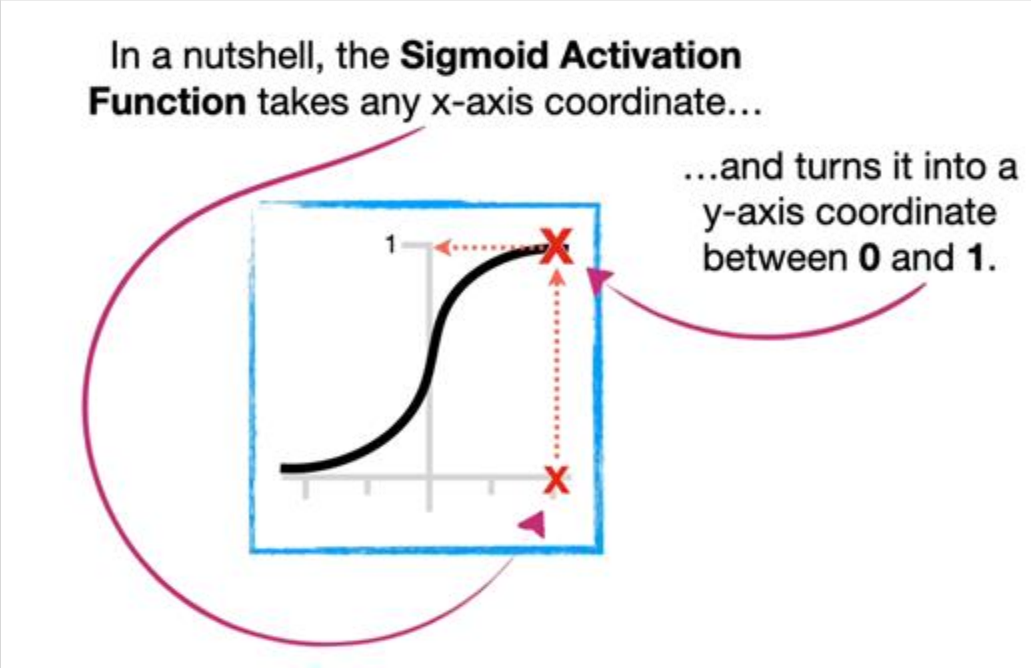


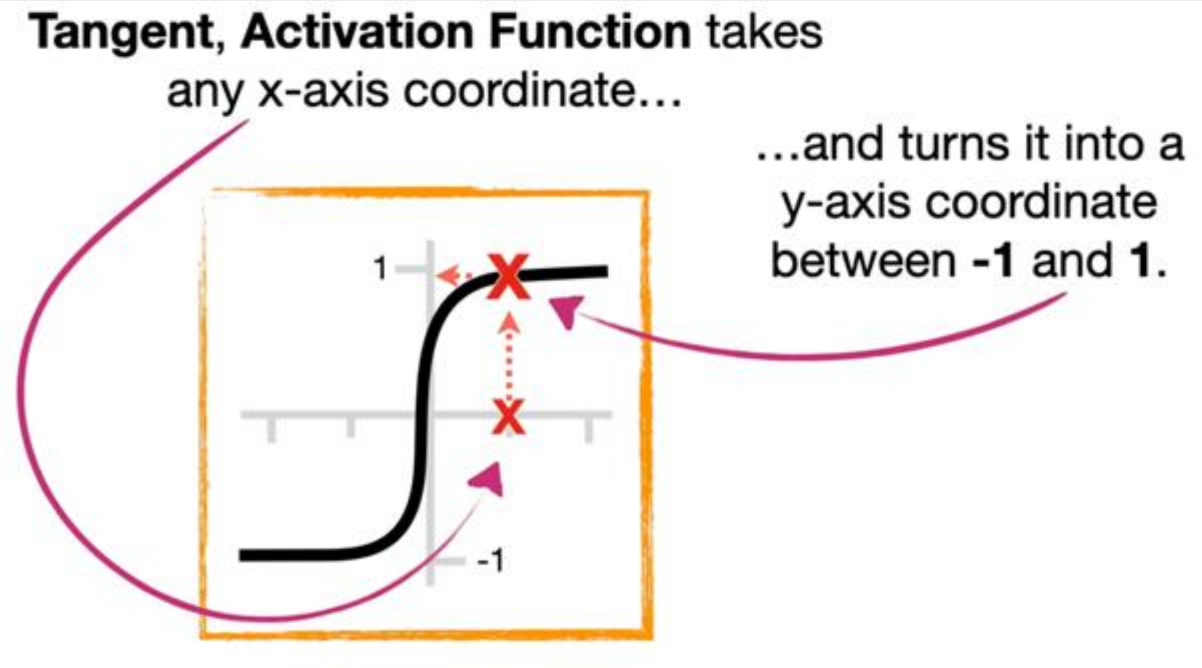
Standard LSTM Cell



GRU Cell

Activation functions

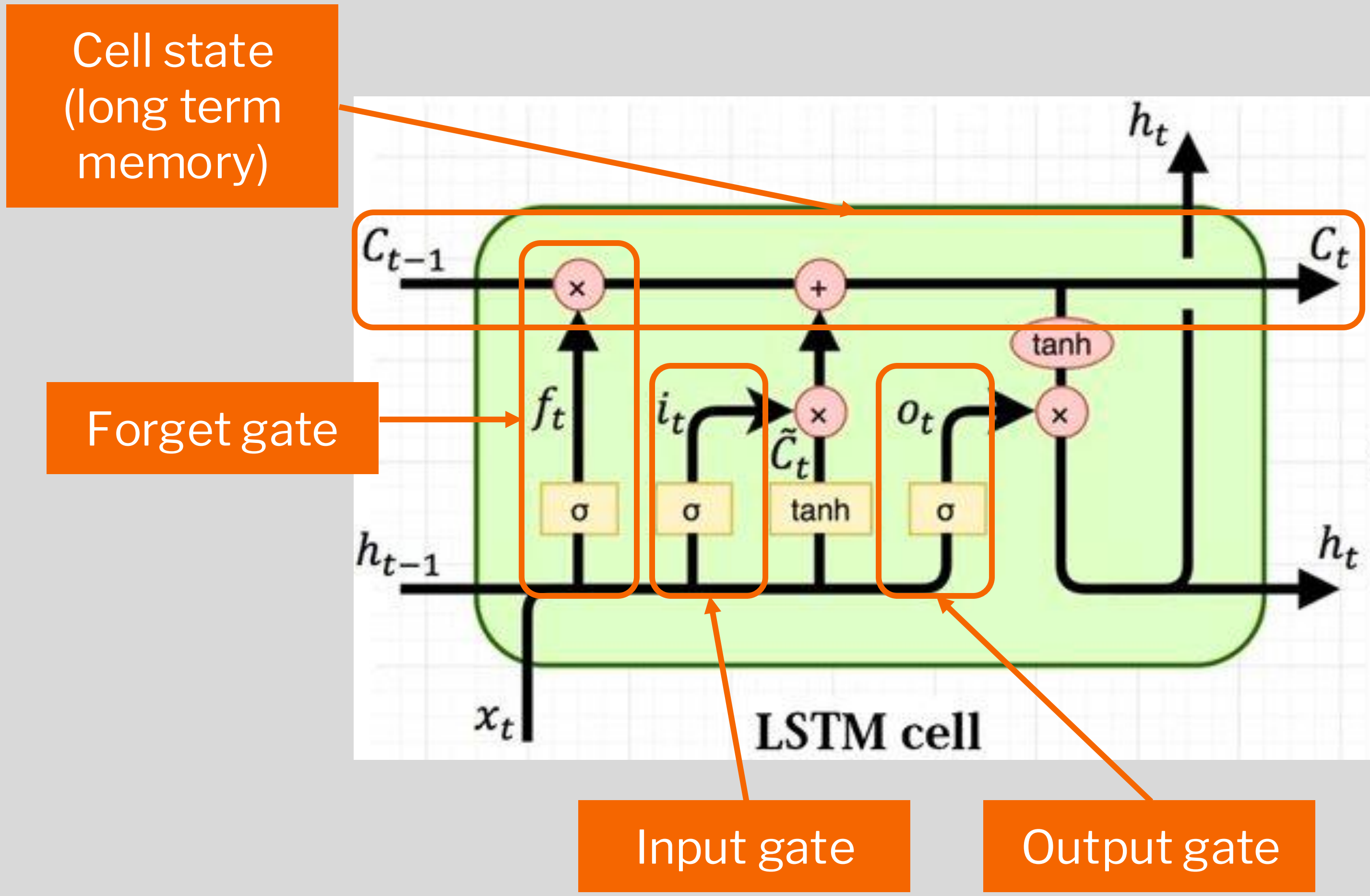




Useful for gates, where we want no (0) or all (1) information to flow

Useful for cell outputs where we want to avoid vanishing gradients (ReLU can also be used)

LSTM



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$

$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$

$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

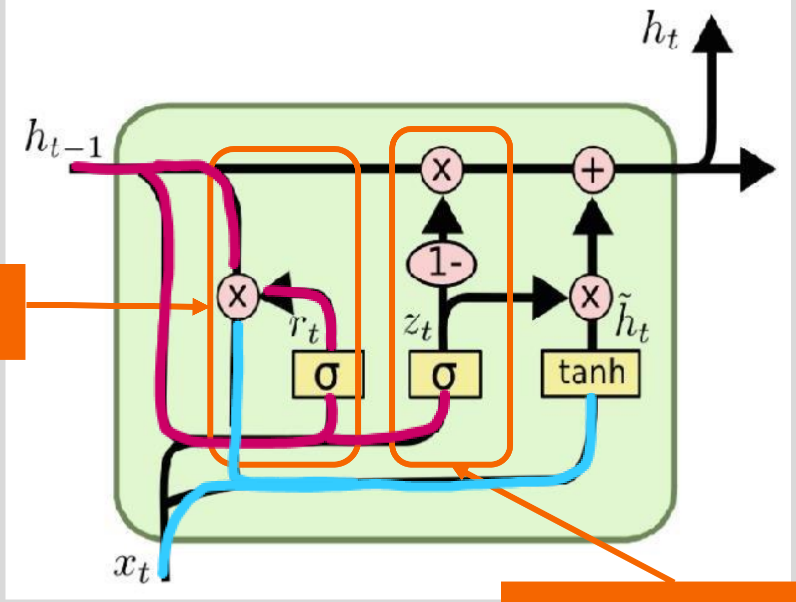
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

$$h_t = \tanh(C_t) * o_t$$

https://www.youtube.com/watch?v=YCzL96nL7j0

GRU

Reset Gate



Update gate

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

https://www.youtube.com/watch?v=YCzL96nL7j0

Exploding & vanishing gradients revisited

- The exploding / vanishing gradient problem also occurs in other networks types when many layers are used (e.g., several CNN layers)
- Mitigation strategies include:
  - ReLU (and its variants) activation function
  - Weight initialization via Xavier, He, Kaiming methods
  - Gradient clipping
  - Batch normalization

Questions?