CS-541: Artificial Intelligence
Deep Learning I

Abdul Rafae Khan
Department of Computer Science
Stevens Institute of Technology
akhan4@stevens.edu

April 25, 2022

A brief history

- 1943: neural networks $\Leftrightarrow$ logical circuits (McCulloch/Pitts)
- 1949: "cells that fire together wire together" learning rule (Hebb)
- 1969: theoretical limitations of neural networks (Minsky/Papert)
- 1974: backpropagation for training multi-layer networks (Werbos)
- 1986: popularization of backpropagation (Rumelhardt, Hinton, Williams)

A brief history

- 1980: Neocognitron, a.k.a. convolutional neural networks (Fukushima)
- 1989: backpropagation on convolutional neural networks (LeCun)
- 1990: recurrent neural networks (Elman)
- 1997: Long Short-Term Memory networks (Hochreiter/Schmidhuber)
- 2006: unsupervised layerwise training of deep networks (Hinton et al.)

Recap: Feedforward Neural Network

Non-linear Neural Network:

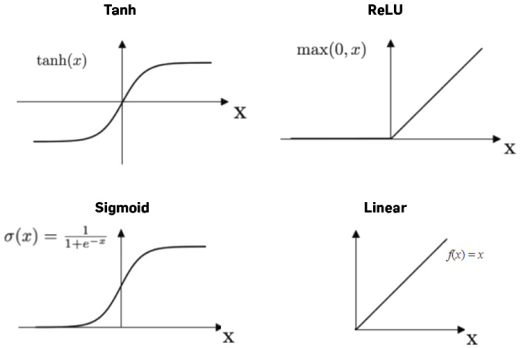
$y = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n)$

$y = \sigma(\sum_{i=1}^{n} w_i x_i)$

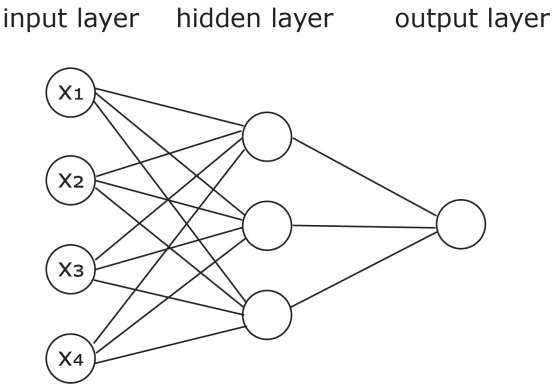
figures/0layer_nn_logistic.jpg

Recap: Feedforward Neural Network

Some commonly used activation functions

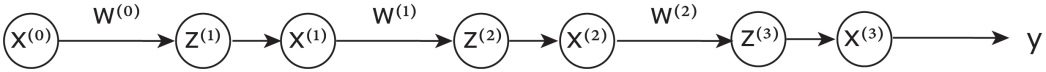


Recap: Feedforward Neural Network

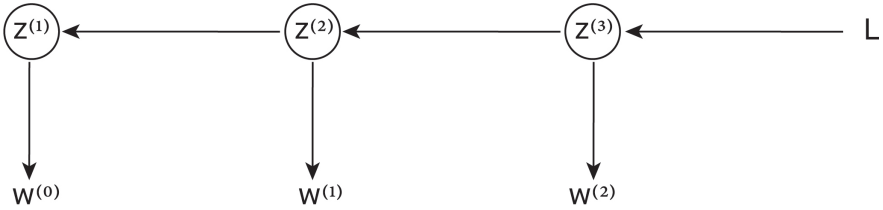


Recap: Feedforward Neural Network

Forward pass



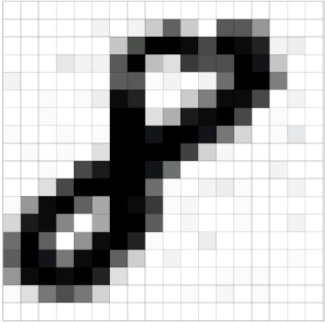
Backward pass



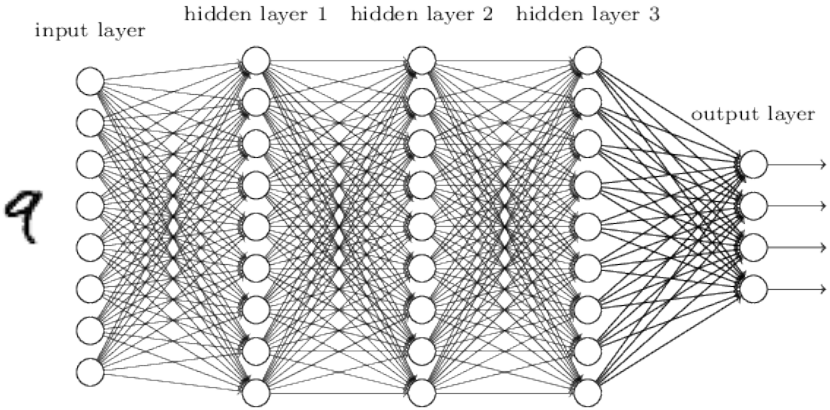
Feedforward Neural Network

MNIST Dataset:

- Each grayscale image, x_{ij}, is a 28×28 matrix
- We reshape it to a 784-dim vector



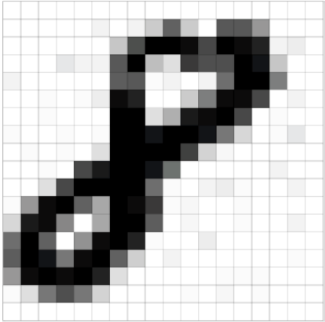
Feedforward Neural Network



Convolutional Neural Network

MNIST Dataset:

- Pixels are not independent
- Pixel in a region depend upon each other



Convolutional Neural Network

Input Image

5×5

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

Filter (Kernel)

3×3

1	0	1
0	1	0
1	0	1

Convolutional Neural Network

Input Image

5×5

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

Filter (Kernel)

3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result

3×3

Convolutional Neural Network

Input Image
5×5

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

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value $\boxed{4}$ **is the inner product of the patch**

1	1	1
0	1	1
0	0	1

and the filter

1	0	1
0	1	0
1	0	1

.

Convolutional Neural Network

Input Image
5×5

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

Filter (Kernel)
3×3

1	0	1
0	1	0
1	0	1

Convolution:

4	3	4
2	4	3
2	3	4

Result
3×3

The value $\boxed{3}$ **is the inner product of the patch**

1	1	0
1	1	1
0	1	1

and the filter

1	0	1
0	1	0
1	0	1

.

Convolutional Neural Network

Input Image		Filters	Outputs
	Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Convolutional Neural Network

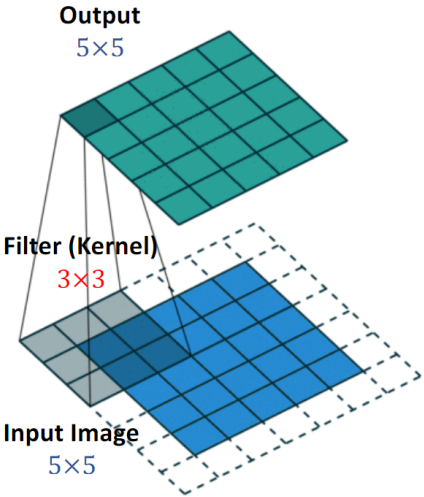
Input Image	Filters	Outputs
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Convolutional Neural Network

Input Image

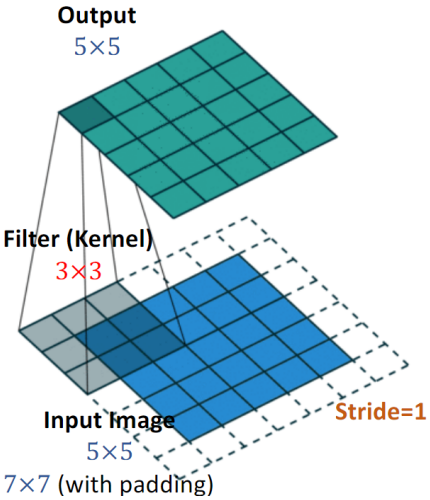
	Filters	**Outputs (or feature map)**
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$	

Convolution: Zero Padding



- Add a "boarder" of all-zeros.
- Increase the input shape:
- From $d_1 \times d_2$ $(d_1 + 2) \times (d_2 + 2)$
- If the filter is 3×3, the output is $d_1 \times d_2$

Convolution: Stride



- In the previous examples, the stride is 1.
- The filter moves 1 step each time.
- The stride can be 2 or even larger.

Dimensions:

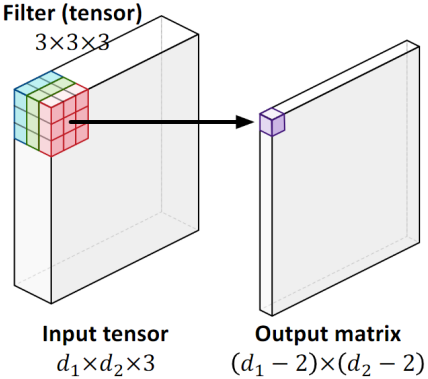
Input: $d_1 \times d_2$

Kernel: $k_1 \times k_2$

Stride: s

Output: $(\lfloor \frac{d_1 - k_1}{s} + 1 \rfloor) \times (\lfloor \frac{d_2 - k_2}{s} + 1 \rfloor)$

Convolution of Tensors

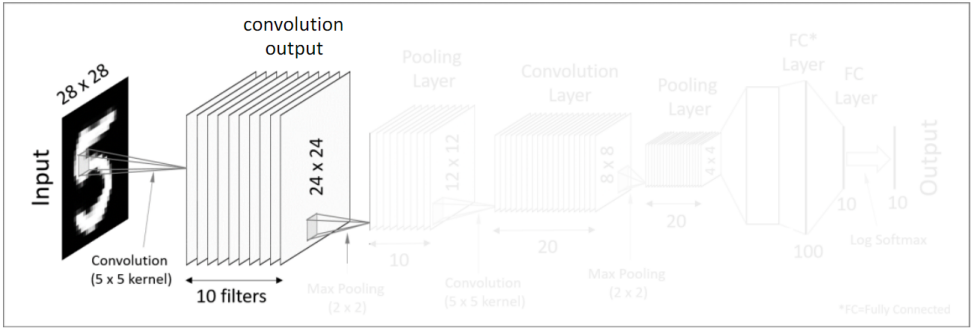


• • The input is the order-3 tensor of shape $d_1 \times d_2 \times 3$ e.g., a color image. • The filter is a order-3 tensor of shape $3 \times 3 \times 3$. • The output is a matrix (order-2 tensor) of shape $(d_1 2) \times (d_2 2)$

Convolutional Neural Network

Use multiple filters (kernels) for the feature mapping. E.g.,

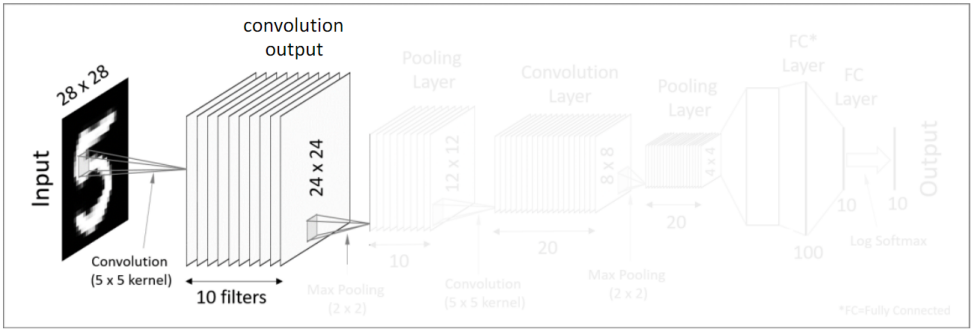
- using ten 5×5 filters, (merely $10\times5\times5 = 250$ parameters!)
- the results are ten 24×24 matrices convolution



Convolutional Neural Network

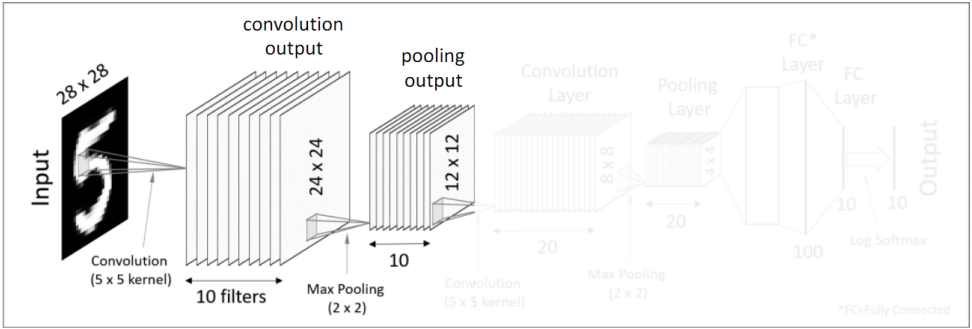
Use multiple filters (kernels) for the feature mapping.

- Apply an activation function (e.g., ReLU).
- A Convolutional layer = convolution + activation.



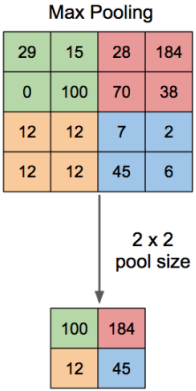
Convolutional Neural Network

- Pooling reduces the dimensionality of each feature map.
- E.g., Max Pooling, Average Pooling, etc.



Convolutional Neural Network

- Pooling reduces the dimensionality of each feature map.
- E.g., Max Pooling, Average Pooling, etc.

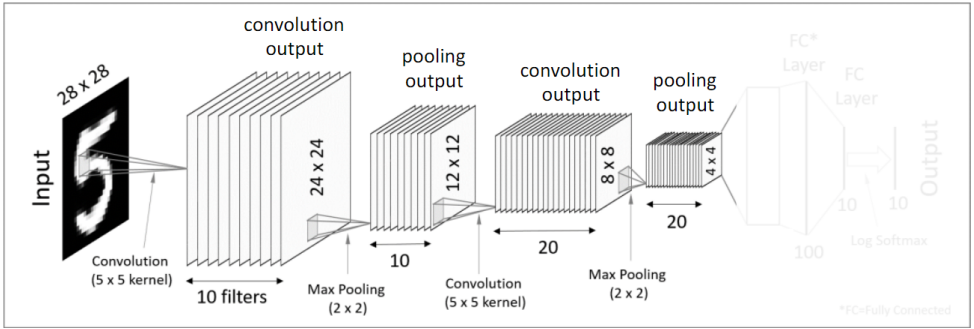


Settings:

- Max Pooling
- Pool size $= 2 \times 2$
- No overlap (stride $= 2 \times 2$)

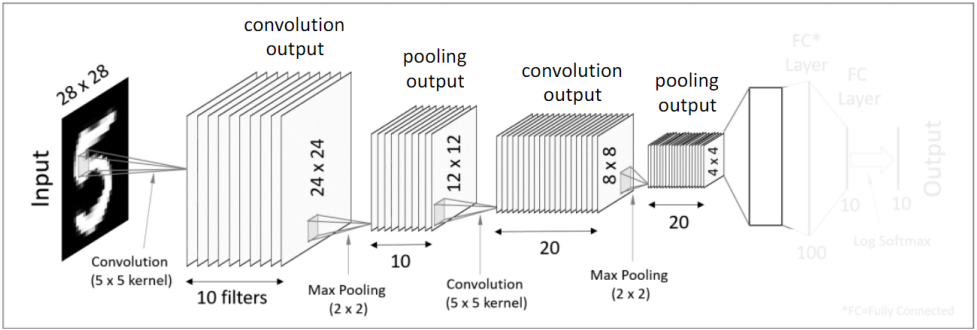
Convolutional Neural Network

- Perform tensor convolution
- Apply $5 \times 5 \times 10$ tensor filter to the $12 \times 12 \times 10$ tensor input
- Twenty such filters $\rightarrow$ twenty 8×8 feature maps
- 2×2 Max Pooling applied to an 8×8 matrix $\rightarrow$ a 4×4 matrix



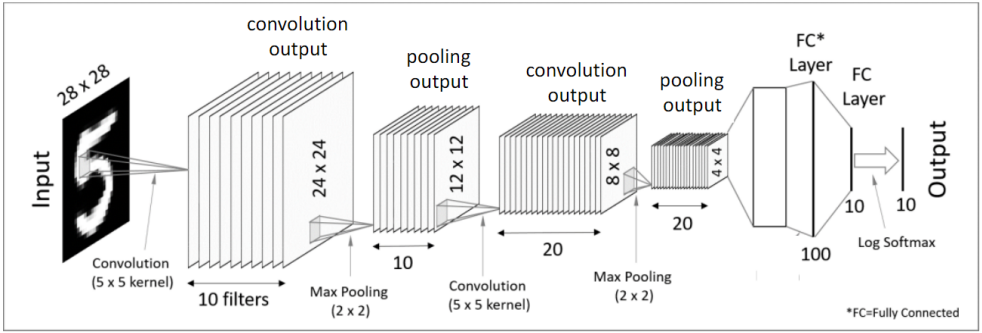
Convolutional Neural Network

- Flatten the $4 \times 4 \times 20$ tensor
- Get a 320-dim vector



Convolutional Neural Network

Add one or more fully-connected layers



Convolutional Neural Network

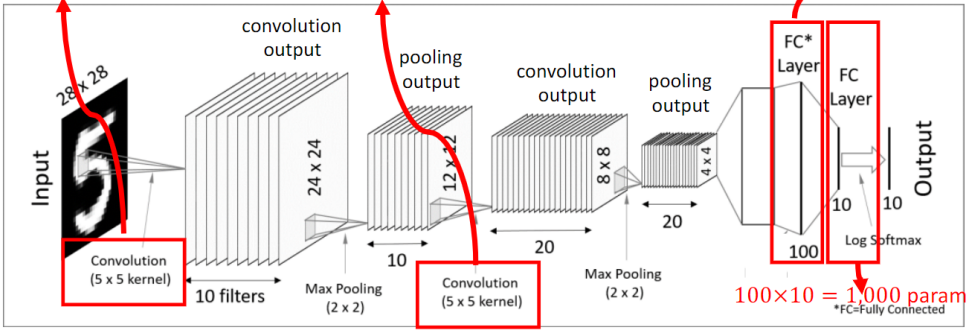
$10 \times 5 \times 5 = 250$ parameters

- 10: number of filters
- 5×5: size of filters

$20 \times 5 \times 5 \times 10 = 5000$ parameters

- 20: number of filters
- $5 \times 5 \times 10$: size of filters

$320 \times 100 = 32{,}000$ parameters

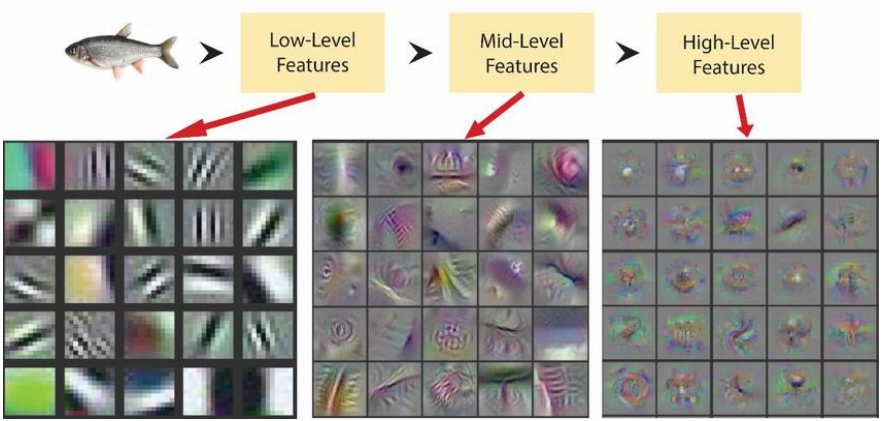


$100 \times 10 = 1{,}000$ parameters

Convolutional Neural Network

Question: Why convolutional layers?

Answer: detect features like edge, shape, pattern, etc.



CNN: Hyper-Parameters

- Convolutional layers

No. of Filters, Filter shape, Stride, Zero-padding

- Pooling layers

MaxPool or AvgPool, Pool size, Pool stride

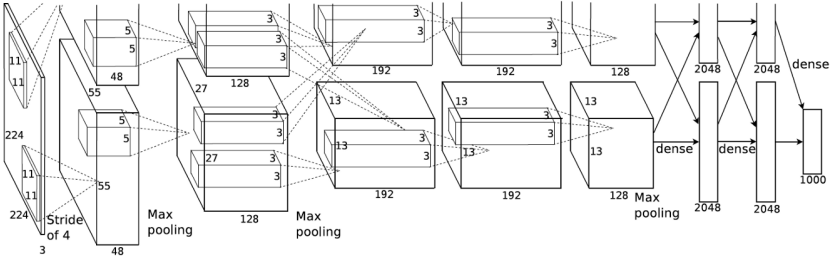
- Fully-connected layers

Width

- Activation functions

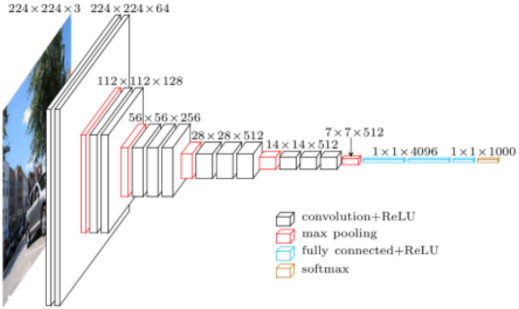
ReLU, SoftMax, Sigmoid

AlexNet [Simoyan/Zisserman, 2014]



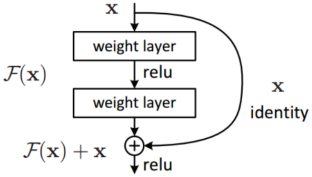
- Non-linearity: use RelU ($\max(z, 0)$) instead of logistic
- Data augmentation: translate, horizontal reflection, vary intensity, dropout (guard against overfitting)
- Computation: parallelize across two GPUs (6 days)
- Results on ImageNet: 16.4% error (next best was 25.8%)

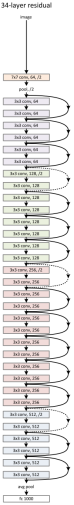
VGGNet [Krizhevsky et al., 2012]



- Architecture: deeper but smaller filters; uniform
- Computation: 4 GPUs for 2-3 weeks
- Results on ImageNet: 7.3% error (AlexNet: 16.4%)

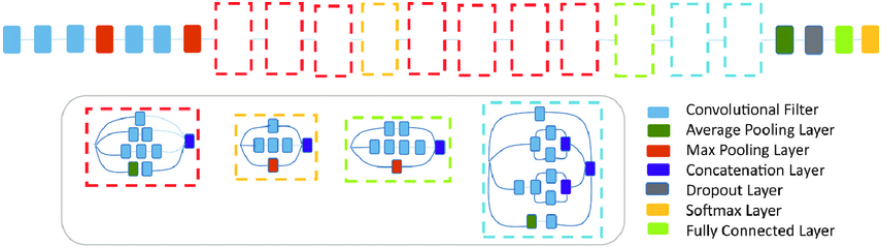
Residual Network [He et al. 2015]





- Key idea: make it easy to learn the identity (good inductive bias)
- Enables training 152 layer networks
- Results on ImageNet: 3.6% error

Inception V3 [Szegedy et al., 2016]



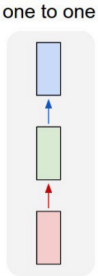
Inception Modules

- Architecture: deeper but smaller filters; uniform
- Computation: 4 GPUs for 2-3 weeks
- Results on ImageNet: 7.3% error (AlexNet: 16.4%)

Summary so for

- Locality of connections, capture spatial structure
- Filters have parameter sharing; most parameters in last fully connected layers
- Depth really matters

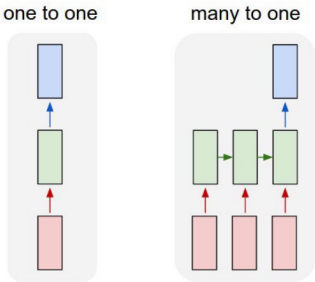
How to model sequential data?



Limitations of FC Nets and ConvNets:

- Process a paragraph as a whole
- Fixed-size input (e.g., image)
- Fixed-size output (e.g., predicted probabilities)

How to model sequential data?



Limitations of FC Nets and ConvNets:

- Process a paragraph as a whole
- Fixed-size input (e.g., image)
- Fixed-size output (e.g., predicted probabilities)

- RNNs are better ways to model the sequential data (e.g., text, speech, and time series).

How to convert words to vectors?

- Neural networks use vectors as inputs
- Convert words to fixed length vectors

How to convert words to vectors?

- We have already seen TFIDF
- Another simpler idea is one-hot encoding

Rome Paris word V

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

How to convert words to vectors?

- We have already seen TFIDF
- Another simpler idea is one-hot encoding

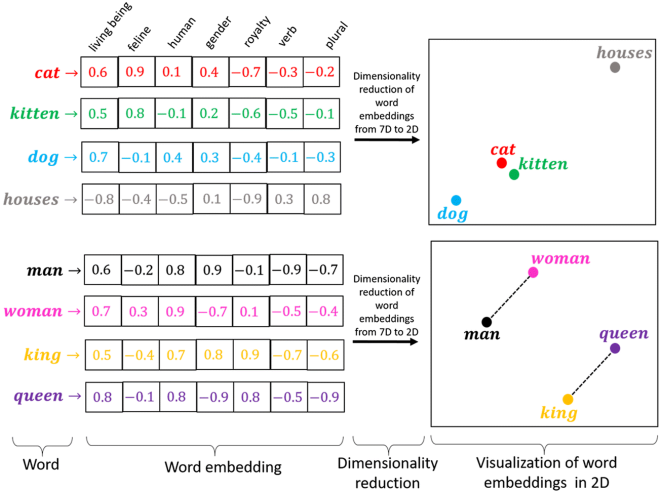
Rome Paris word V

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

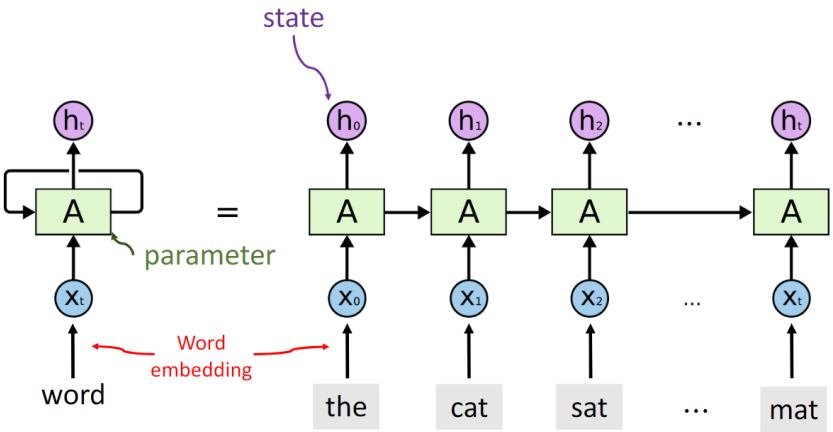
- The vector size is as large as the vocabulary
- No relation between similar word vectors

How to convert words to vectors?

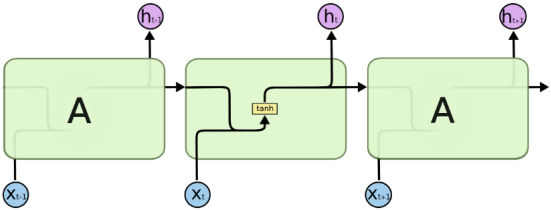
- Semantic relationship between words is preserved

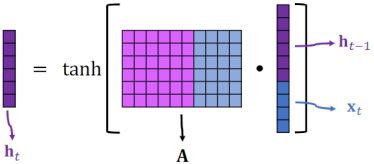


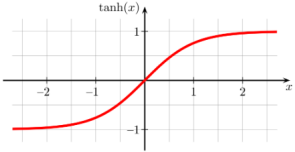
Simple RNN


state
h_t
A
=
parameter
x_t
Word
embedding
word
h_0
h_1
h_2
...
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t
the
cat
sat
...
mat

Simple RNN

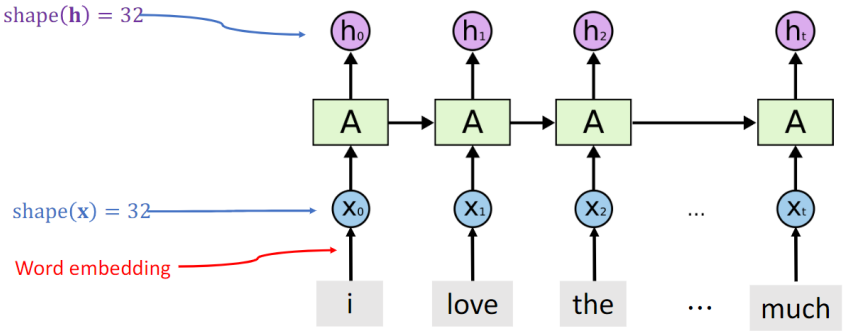




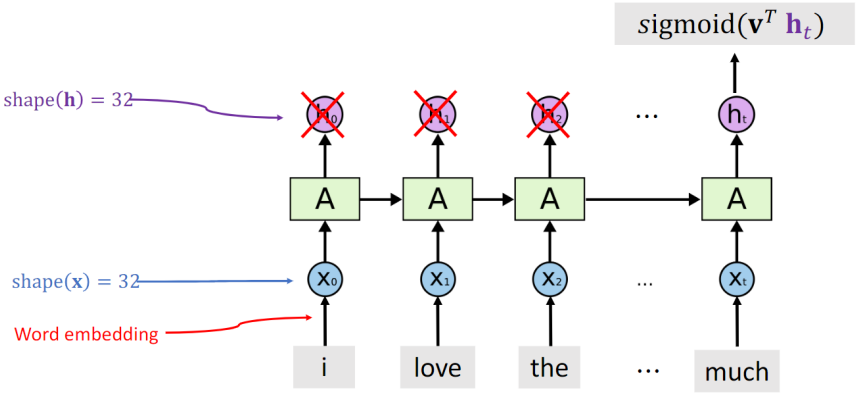


hyperbolic tangent function

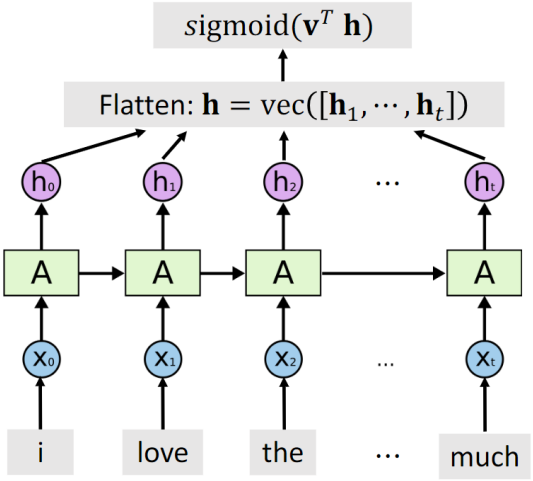
Simple RNN


shape(h) = 32
shape(x) = 32
Word embedding
h0
h1
h2
ht
A
A
A
A
x0
x1
x2
...
xt
i
love
the
...
much

Simple RNN

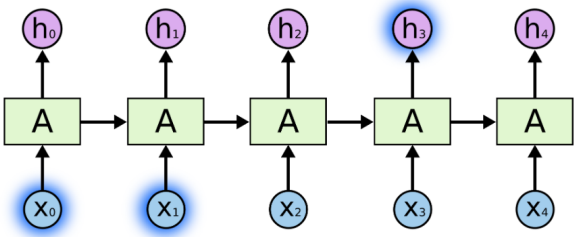

$\text{sigmoid}(\mathbf{v}^T\ \mathbf{h}_t)$
$\text{shape}(\mathbf{h}) = 32$
h_0
h_1
h_2
...
h_t
A
A
A
A
$\text{shape}(\mathbf{x}) = 32$
x_0
x_1
x_2
...
x_t
Word embedding
i
love
the
...
much

Simple RNN


$\text{sigmoid}(\mathbf{v}^T\ \mathbf{h})$
Flatten: $\mathbf{h} = \text{vec}([\mathbf{h}_1, \cdots, \mathbf{h}_t])$
h_0
h_1
h_2
...
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t
i
love
the
...
much

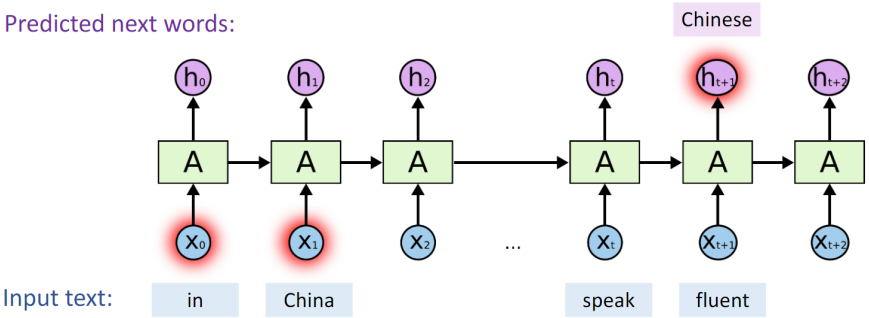
SimpleRNN is good at short-term dependence

Predicted next words: sky

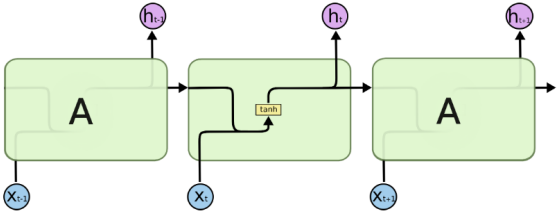


Input text: clouds are in the

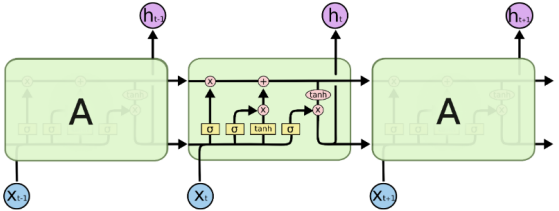
Simple RNN


Predicted next words:
Chinese
h_0
h_1
h_2
h_t
h_{t+1}
h_{t+2}
A
A
A
A
A
A
x_0
x_1
x_2
...
x_t
x_{t+1}
x_{t+2}
Input text:
in
China
speak
fluent

LSTM RNN [Hochreiter/Schmidhuber, 1997]



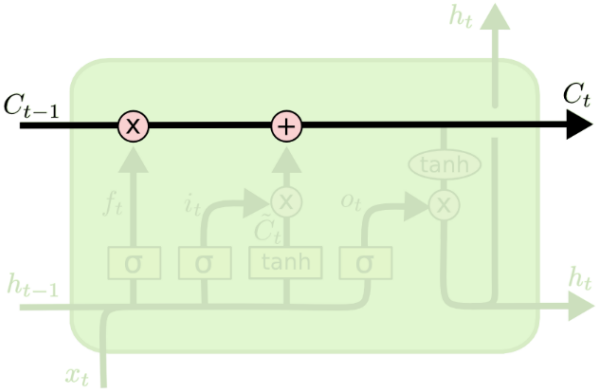
Simple RNN



LSTM

LSTM RNN

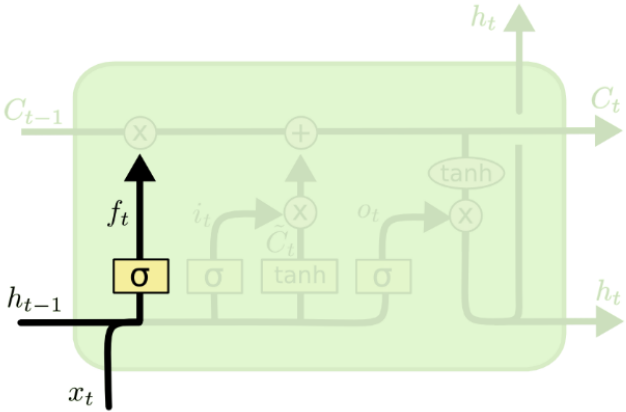
Conveyor belt: the past information directly flows to the future



LSTM RNN

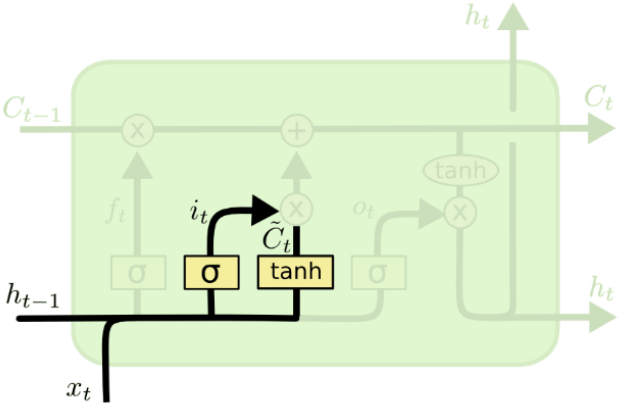
Forget gate (f): a vector (the same shape as c and h).

- A value of zero means "let nothing through".
- A value of one means "let everything through!"



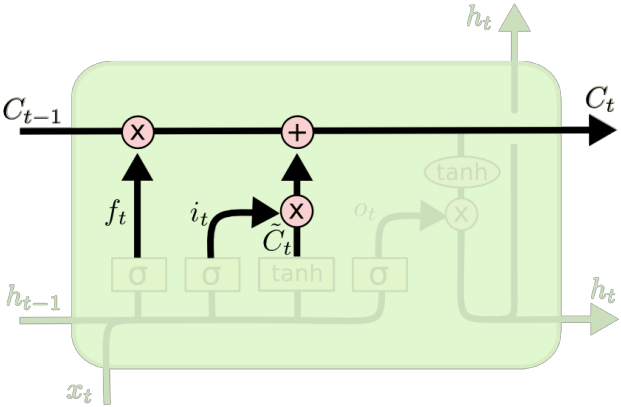
LSTM RNN

Input gate (i): decides which values of the conveyor belt we'll update



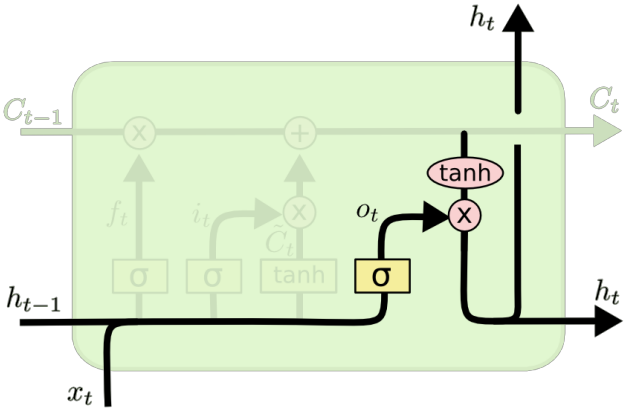
LSTM RNN

New value ($\tilde{c}_t$): to be added to the conveyor belt



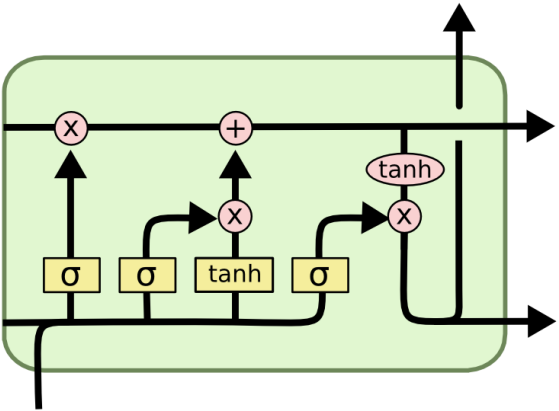
LSTM RNN

Output gate (o_t): decide what flows from the conveyor belt c_{t-1} to the state h_t



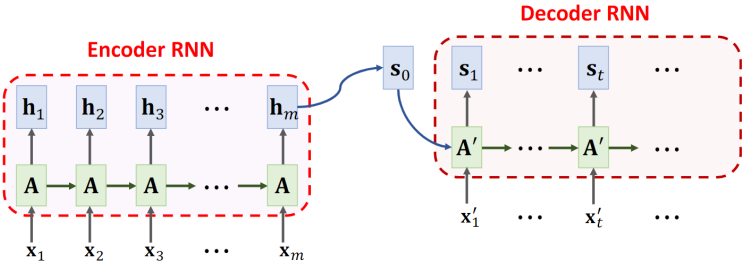
LSTM RNN

Number of parameters: 4 times as many as SimpleRNN



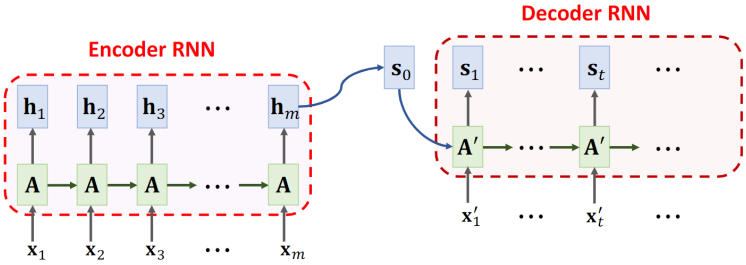
RNN: Sequence-to-Sequence (Seq2Seq)

- Special cases of RNN to solve complex language problems
- Input and Output is a sequence of words/tokens



RNN: Seq2Seq [Sutskever et al 2014]

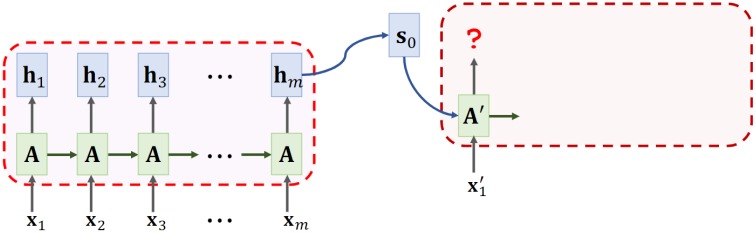
- x_i is the ith word in the input
- x'_j is the jth word in the output



RNN: Seq2Seq

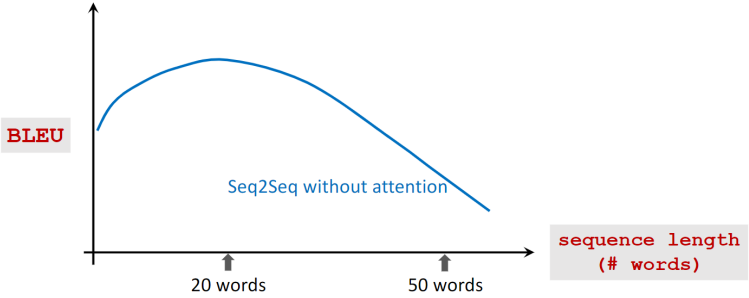
Predicting the next output word

$$s_1 = \tanh(A' \begin{bmatrix} x_1' \\ s_0 \end{bmatrix} + b)$$



RNN: Seq2Seq

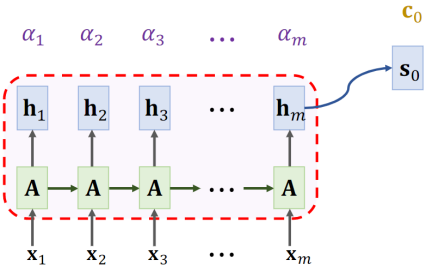
As the length of the input sequence increases, the performance decreases



RNN: Seq2Seq + attention

$$\text{Weight: } \alpha_i = \mathit{align}(h_i, s_0)$$

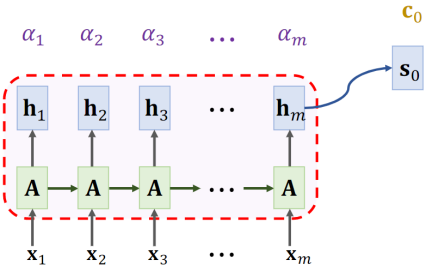
where 'align' is another small neural network
α_i represents how important word x_i is for the prediction of the word



RNN: Seq2Seq + attention

$$\text{Weight: } \alpha_i = \mathit{align}(h_i, s_0)$$

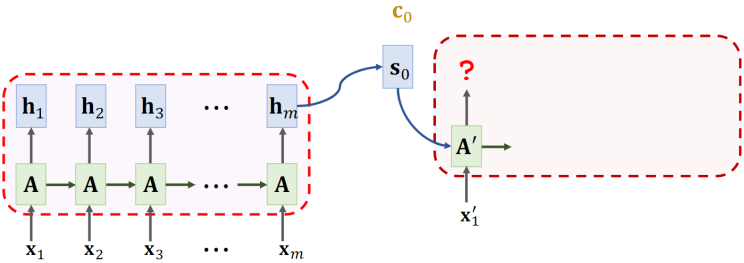
$$\text{Context vector: } c_0 = \alpha_i h_i + \cdots + \alpha_m h_m$$



RNN: Seq2Seq + attention

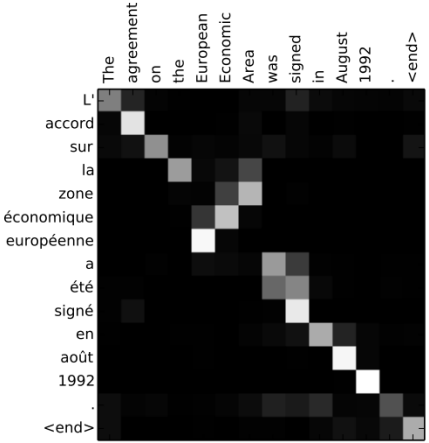
Predicting the next output word (with attention)

$$s_1 = \tanh(A' \begin{bmatrix} x_1' \\ s_0 \\ c_0 \end{bmatrix} + b)$$



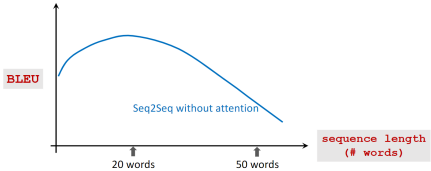
RNN: Seq2Seq + attention

Word alignment using attention values



RNN: Seq2Seq + attention

Increasing the input length does not affect the performance



CNN + attention

Attention mechanism has also been used for CNNs

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

Summary so far

- Recurrent neural networks: model sequences (non-linear version of HMM)
- Logic intuition: learning a program with a for loop (reduce)
- Word embeddings learn word vectors while keeping semantic relationship
- LSTMs mitigate the vanishing gradient problem

Recap

Phenomena	**Ideas**
Fixed vectors	Feedforward NNs
Spatial structure	Convolutional NNs
Sequence	Recurrent NNs, LSTMs
Sequence-to-sequence	Encoder-Decoder

References

Stuart Russell and Xiaodong Song (2021)
CS 188 — Introduction to Artificial Intelligence
University of California, Berkeley

Chelsea Finn and Nima Anari (2021)
CS221 — Artificial Intelligence: Principles and Techniques
Stanford University

The End