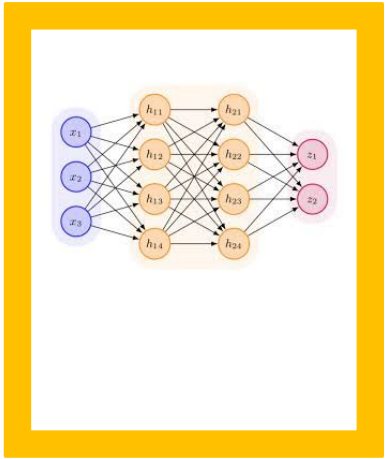


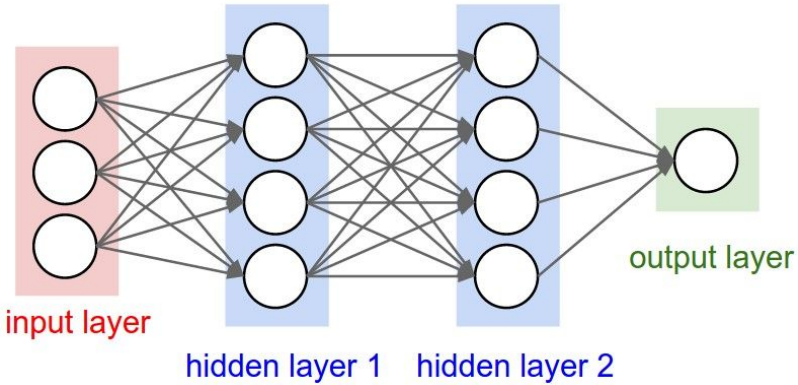
Deep Learning: A Closer Look

Girish Varma

Deep Neural Networks

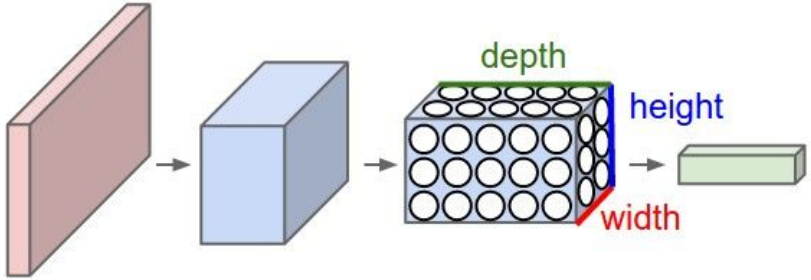
When Input is a vector
(fixed length).

Linear Layer
MLP (Multilayered)



When Input is a
1D/2D/3D sequence.
(variable length)

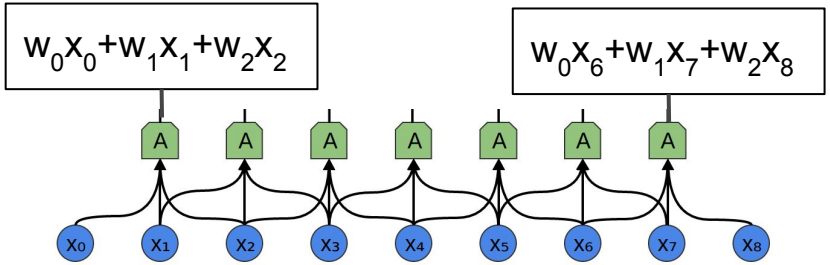
Convolutional Layer
Deep CNN



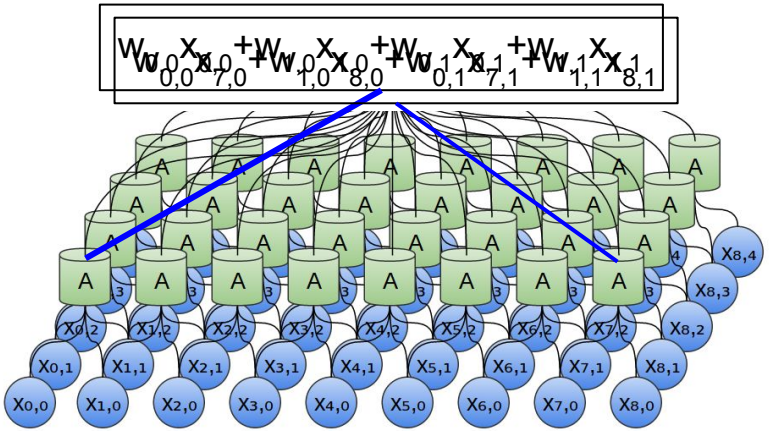
Convolutional Neural Network (CNNs)

For 1D Sequences

weight sharing



For 2D Sequences (Images)

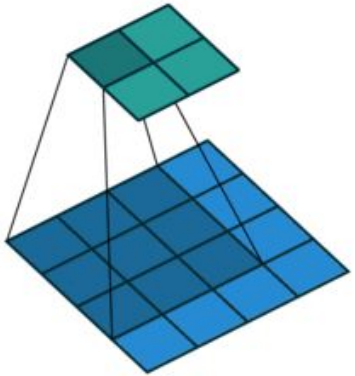


CNNs

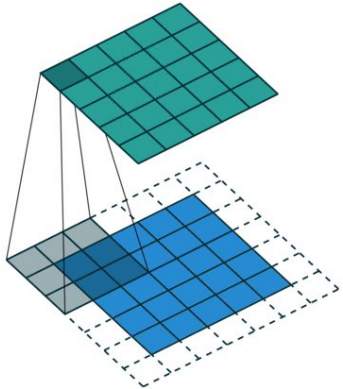
Window size

Stride

Padding



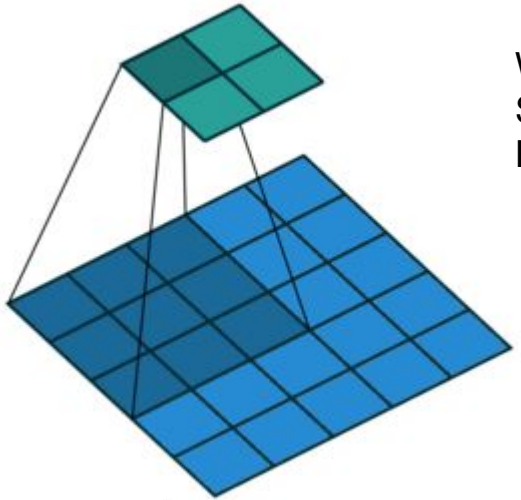
Window size: 3x3
Stride: 1
Padding: 0



Window size: 3x3
Stride: 1
Padding: 1

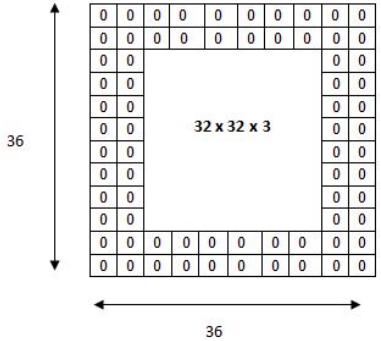
CNNs

Strides reduces dimension

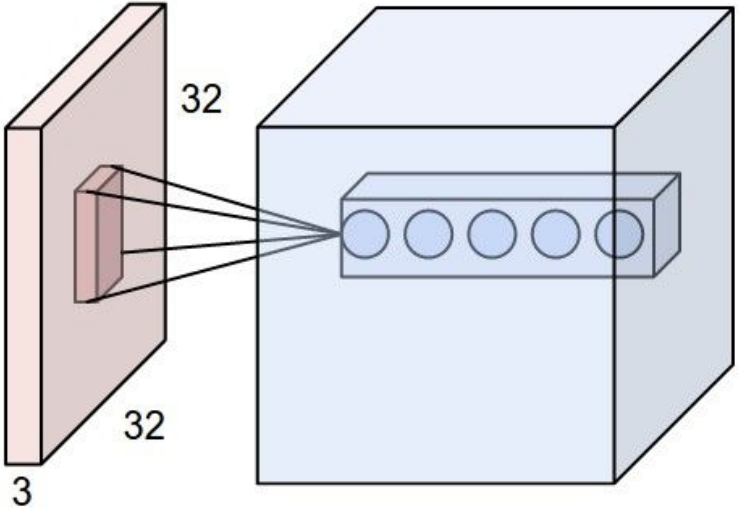


Window size: 3x3
Stride: 2
Padding: 0

$$O = \frac{(W - K + 2P)}{S} + 1$$



Input, Output Channels : Multiple Filters



DEMO: http://cs231n.github.io/convolutional-networks/

Complexity: Parameters

Window Size: F x F

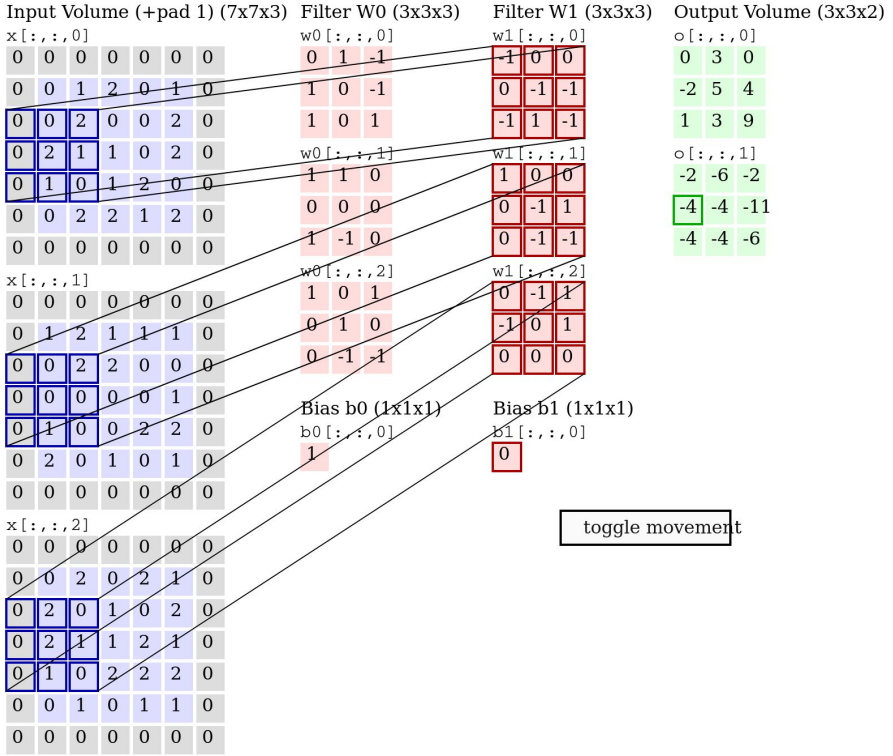
Input Channels: D_1

No. of Filters(Out Channels): K

No. of Parameters =

$$F*F*D_1*K$$

For every pair of input/output channels there is a FxF filter



Complexity: FLOPs

Floating Point Operations

If a FxF convolutions converts an input of size $H_1*W_1*D_1$ to output of size is H_2 x W_2xD_2, then FLOPs is

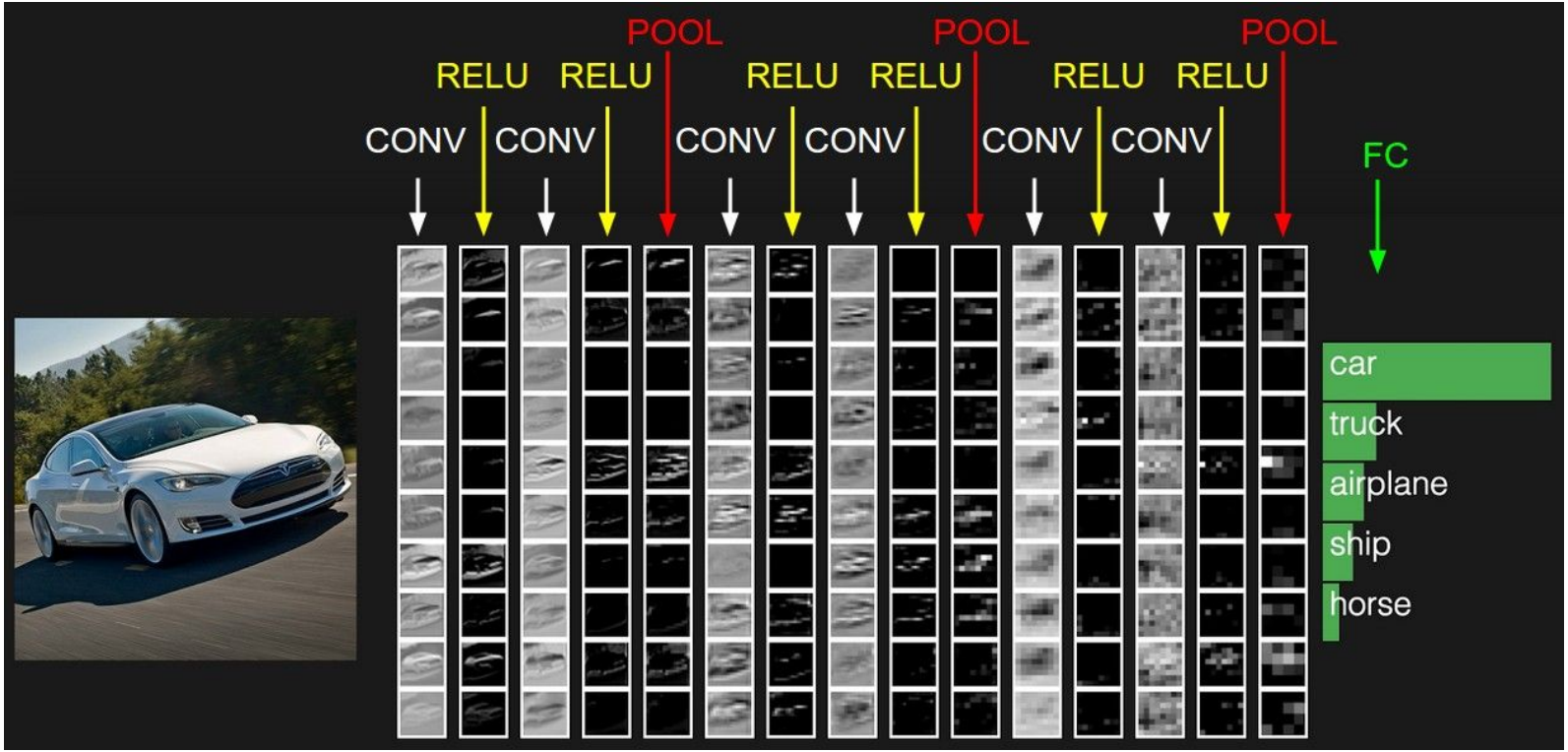
$$H_2*W_2*F*F*D_1*D_2$$

CNN Summary

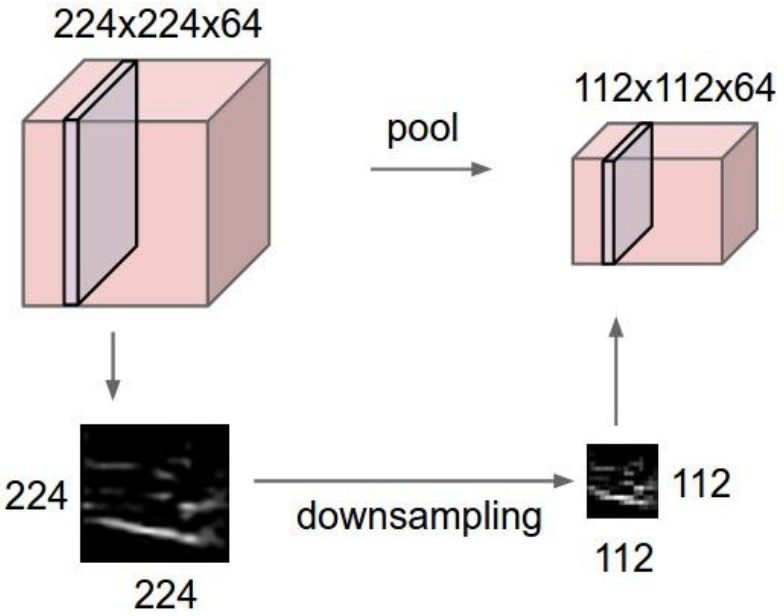
Summary. To summarize, the Conv Layer:

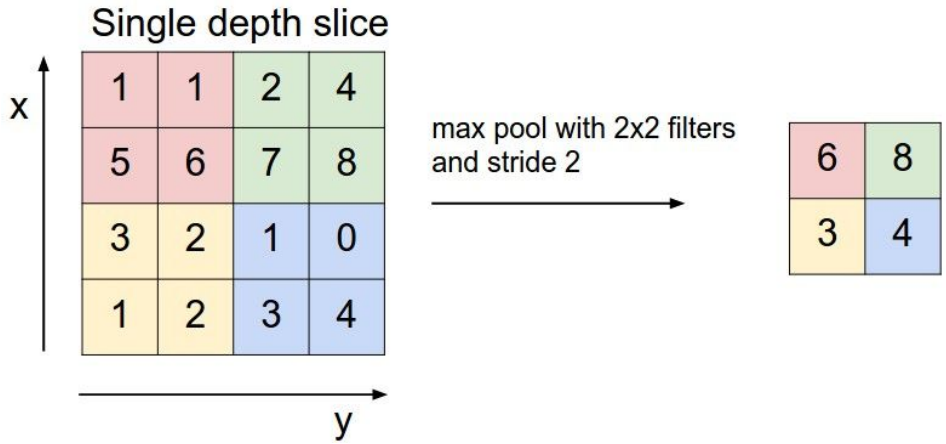
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Deep CNN


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

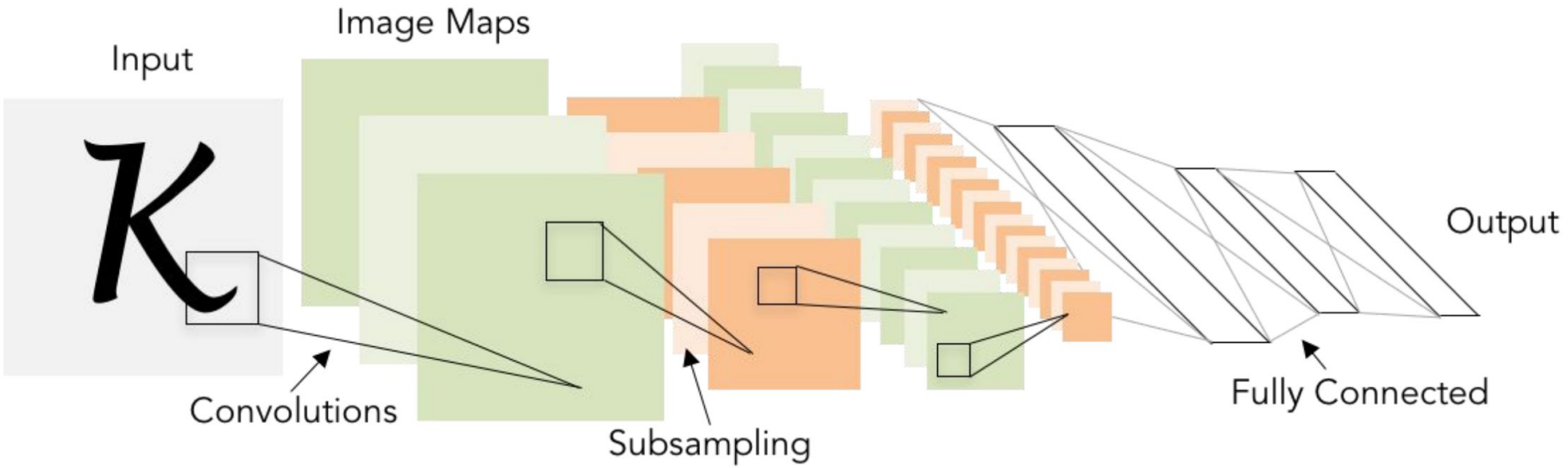
Pooling





No Learnable Parameters

Review: LeNet

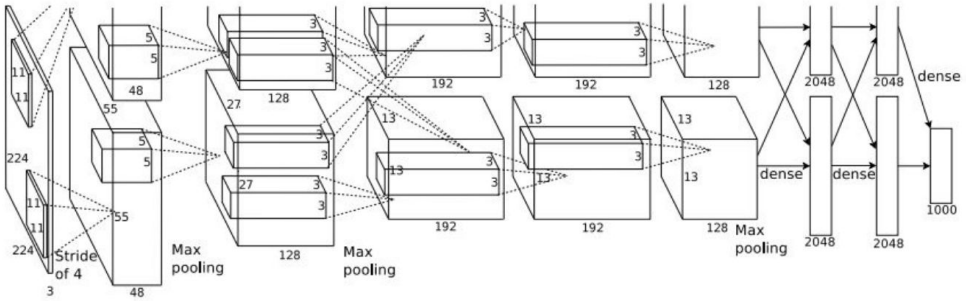


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



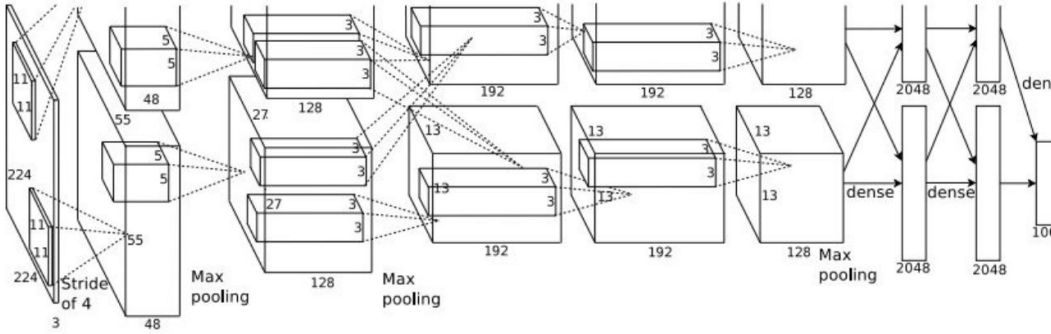
Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

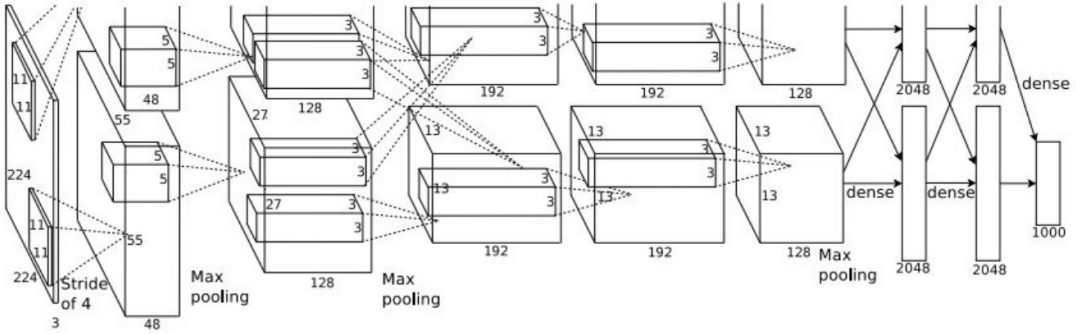
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

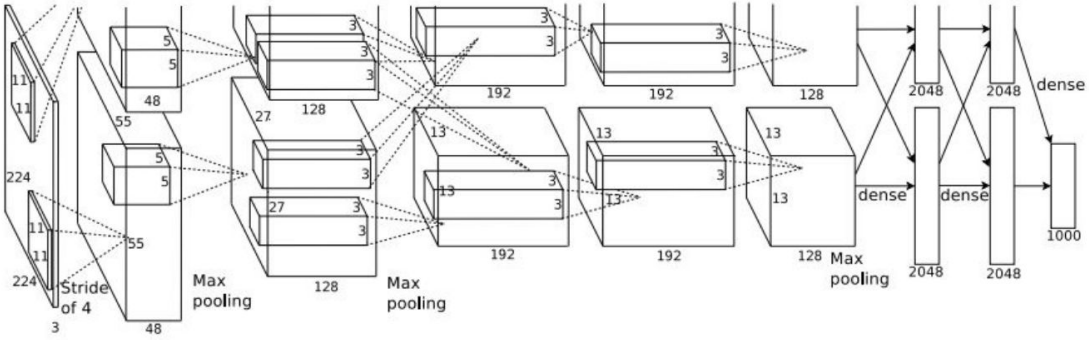
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



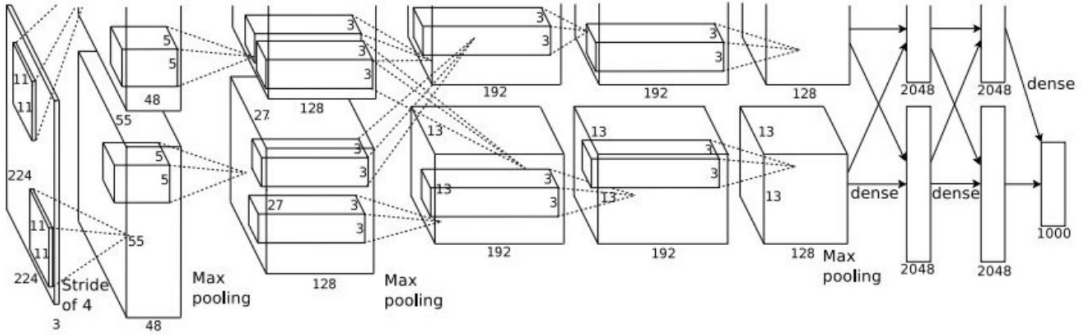
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

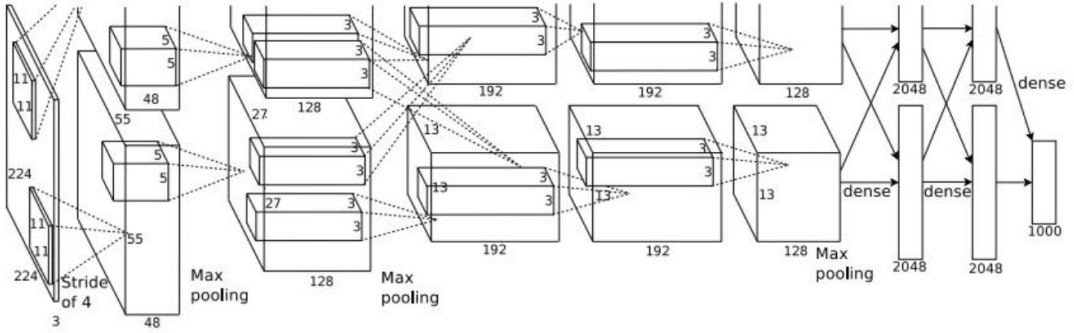
Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

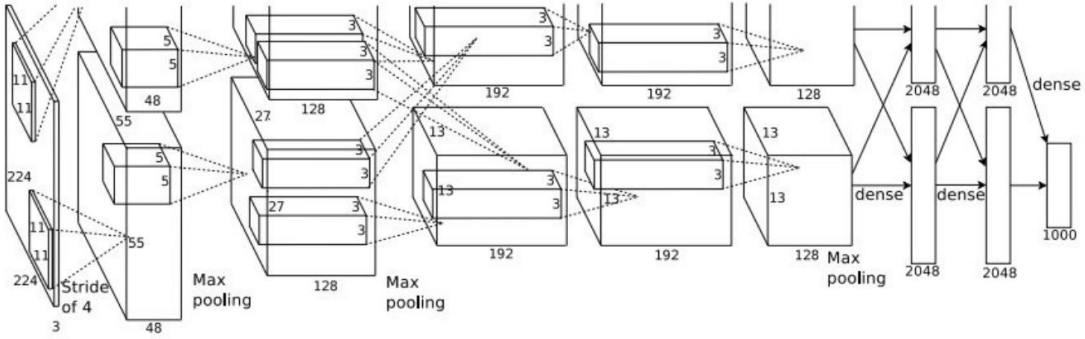
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]



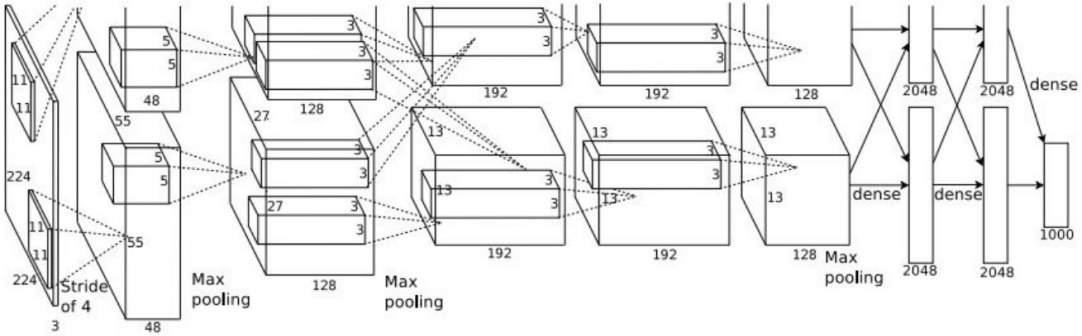
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

CNN Architectures

Case Study: AlexNet

[Krizhevsky et al. 2012]

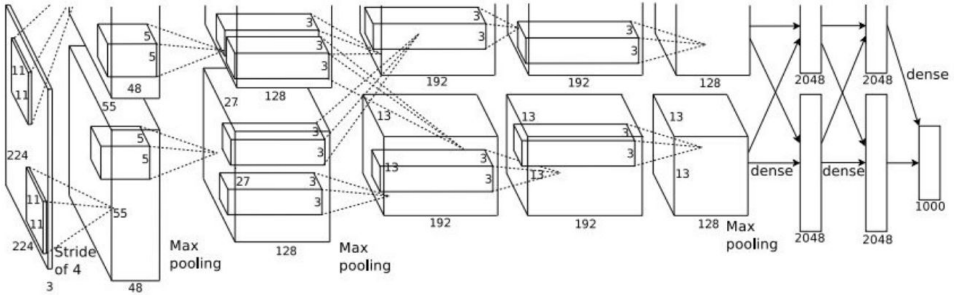


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

CNN Architectures

Case Study: AlexNet

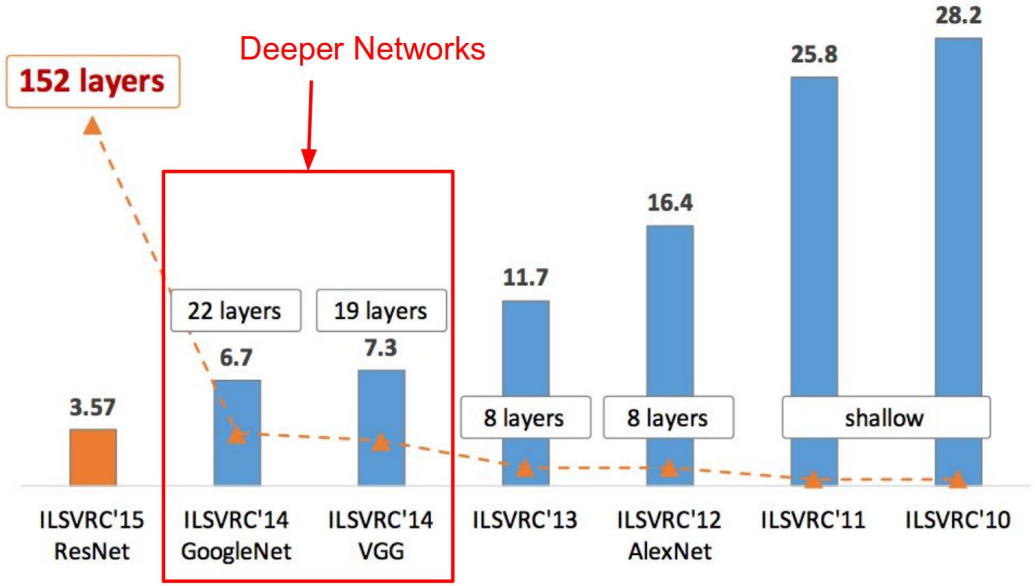
[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

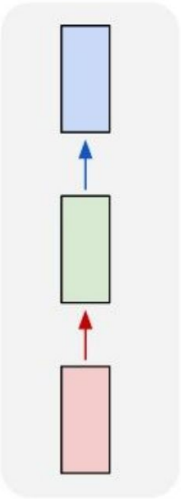
Getting Deeper

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



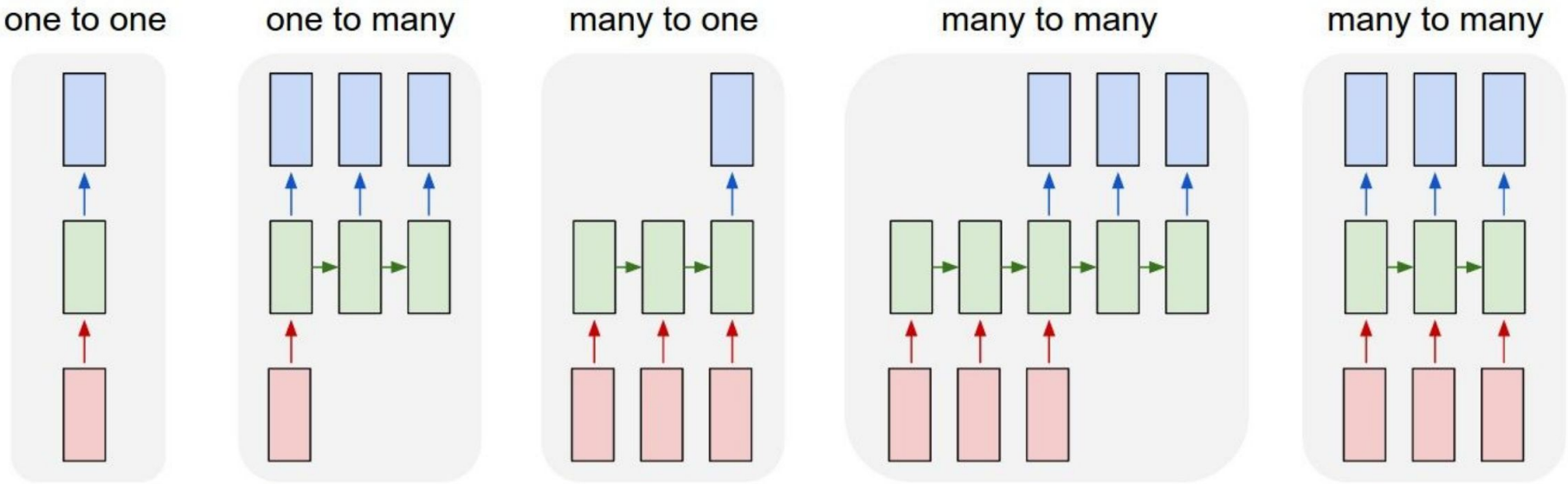
Recurrent Neural Networks (RNNs)

one to one



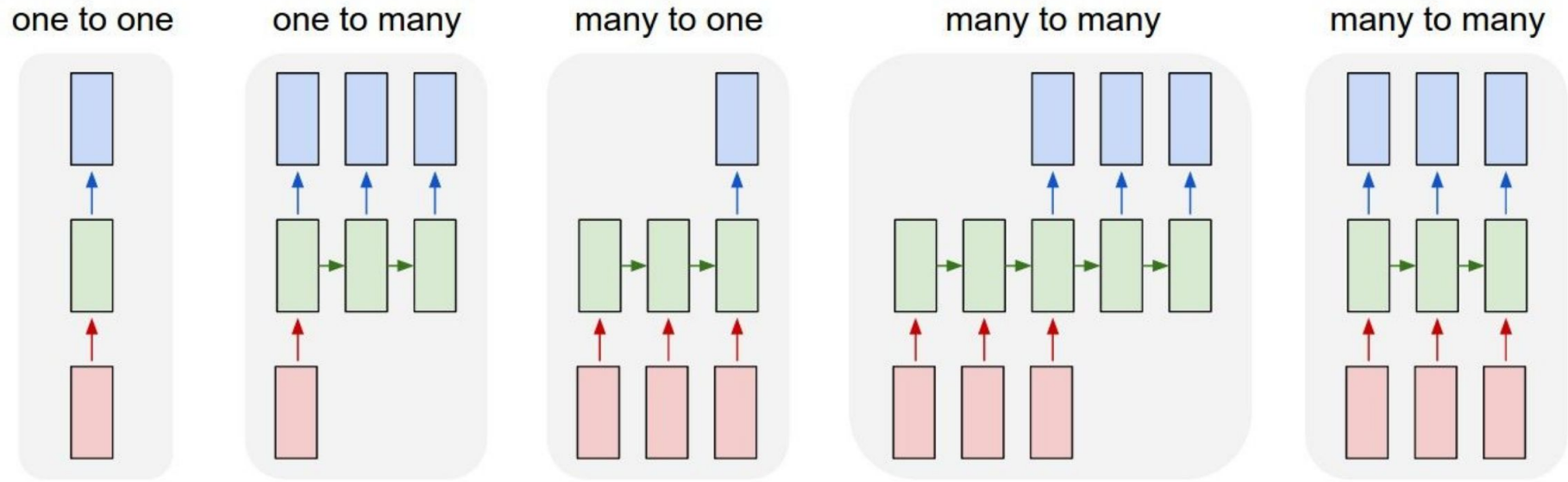
Vanilla Neural Networks

RNNs : Process Sequence



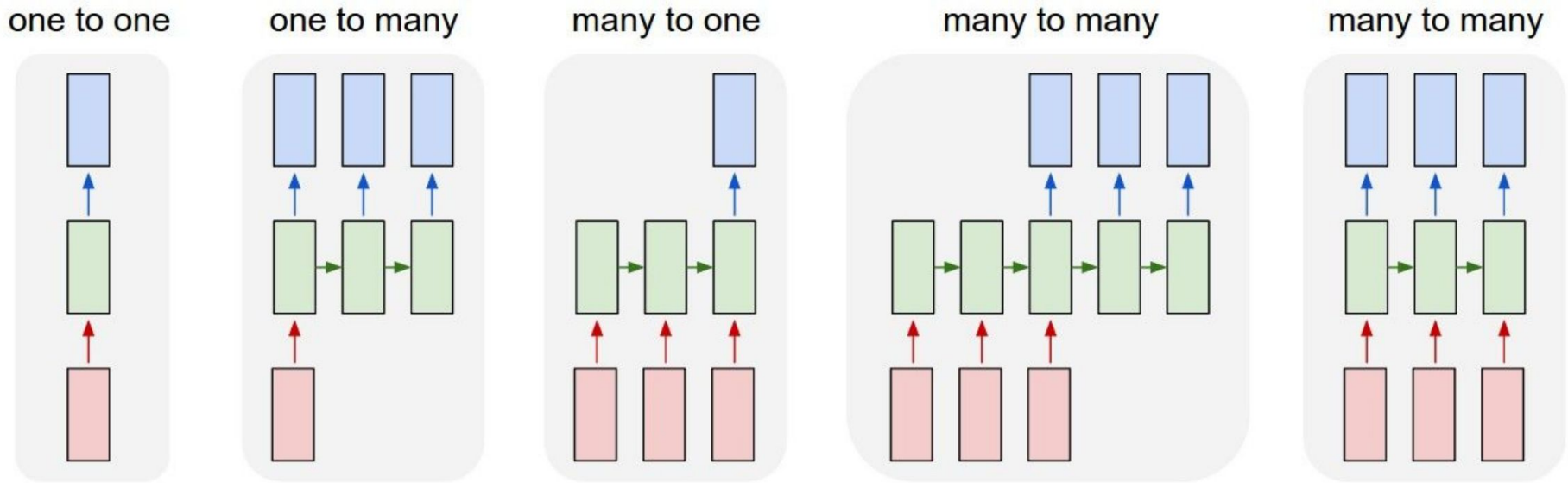
e.g. **Image Captioning**
image -> sequence of words

RNNs : Process Sequence



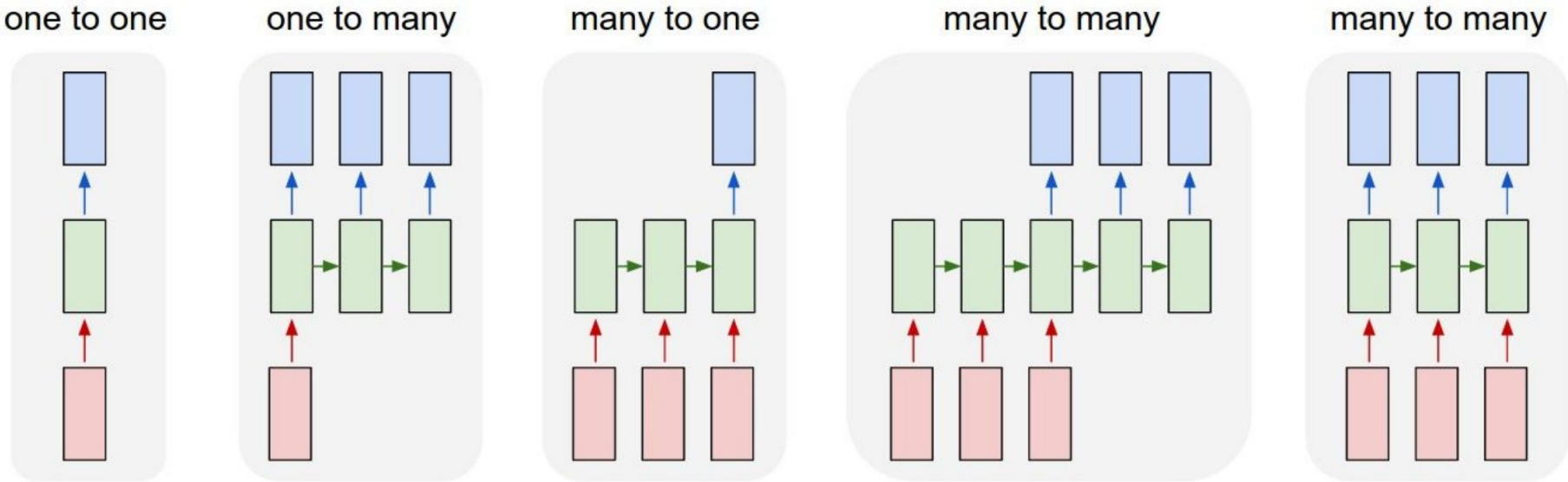
e.g. **Sentiment Classification**
sequence of words -> sentiment

RNNs : Process Sequence



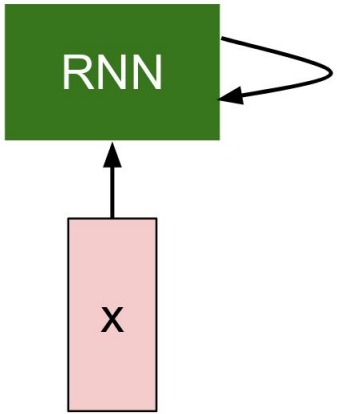
e.g. **Machine Translation**
seq of words -> seq of words

RNNs : Process Sequence

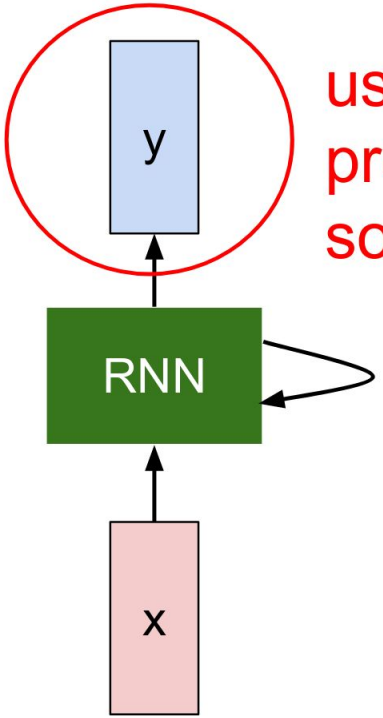


e.g. **Video classification on frame level**

RNNs



RNNs



usually want to predict a vector at some time steps

RNNs

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

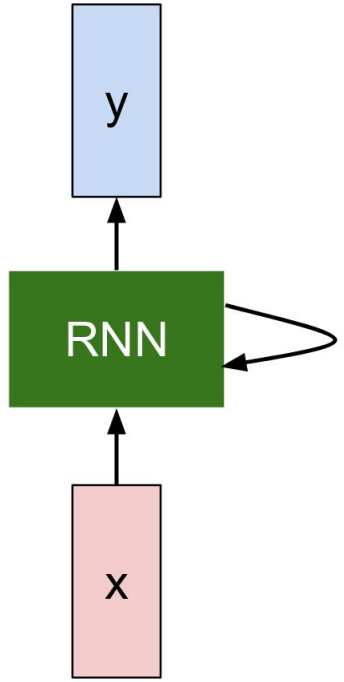
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

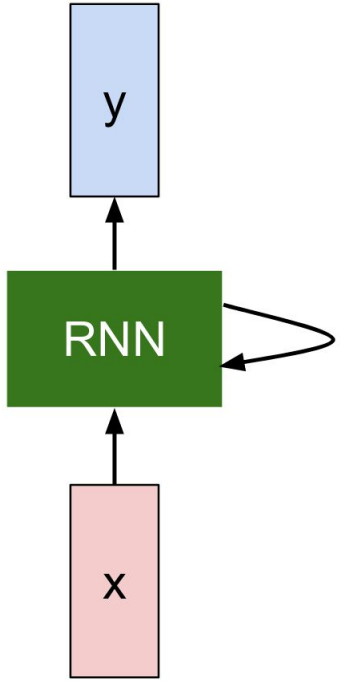
some function with parameters W



RNNs

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

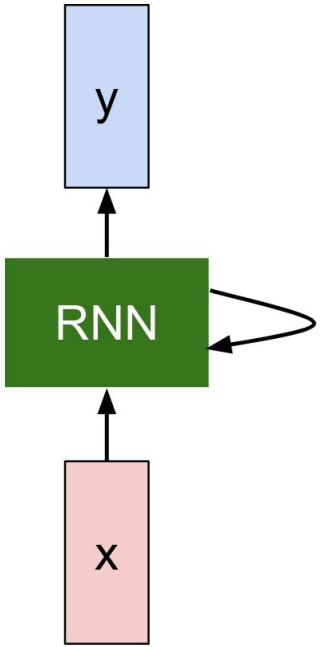
$$h_t = f_W(h_{t-1}, x_t)$$



Notice: the same function and the same set of parameters are used at every time step.

RNNs

The state consists of a single *"hidden"* vector **h**:

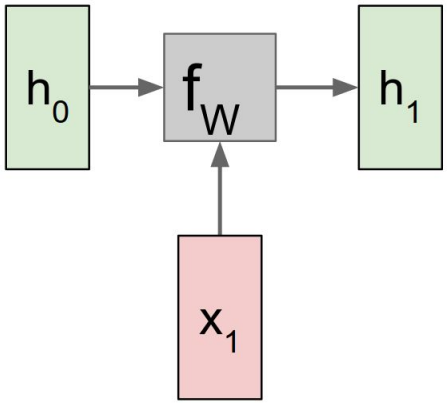


$$h_t = f_W(h_{t-1}, x_t)$$

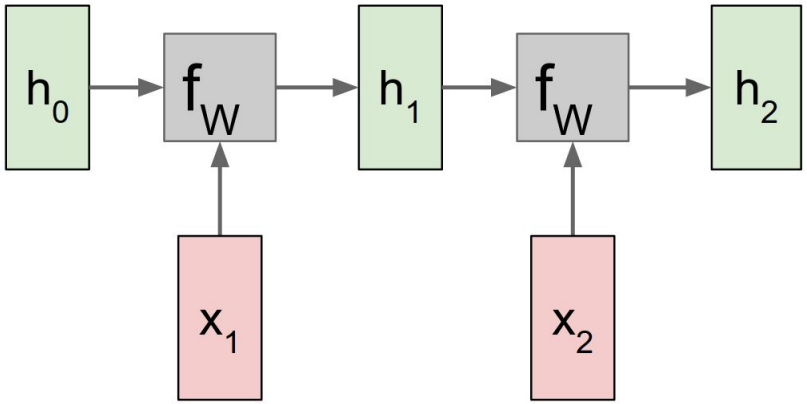
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

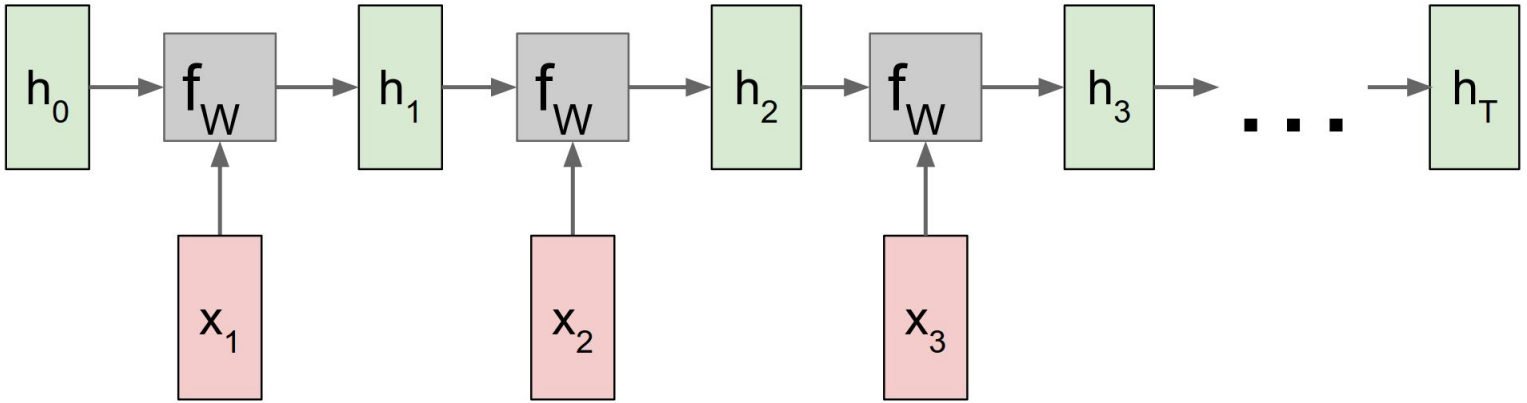
RNN : Computation Graph



RNN : Computation Graph

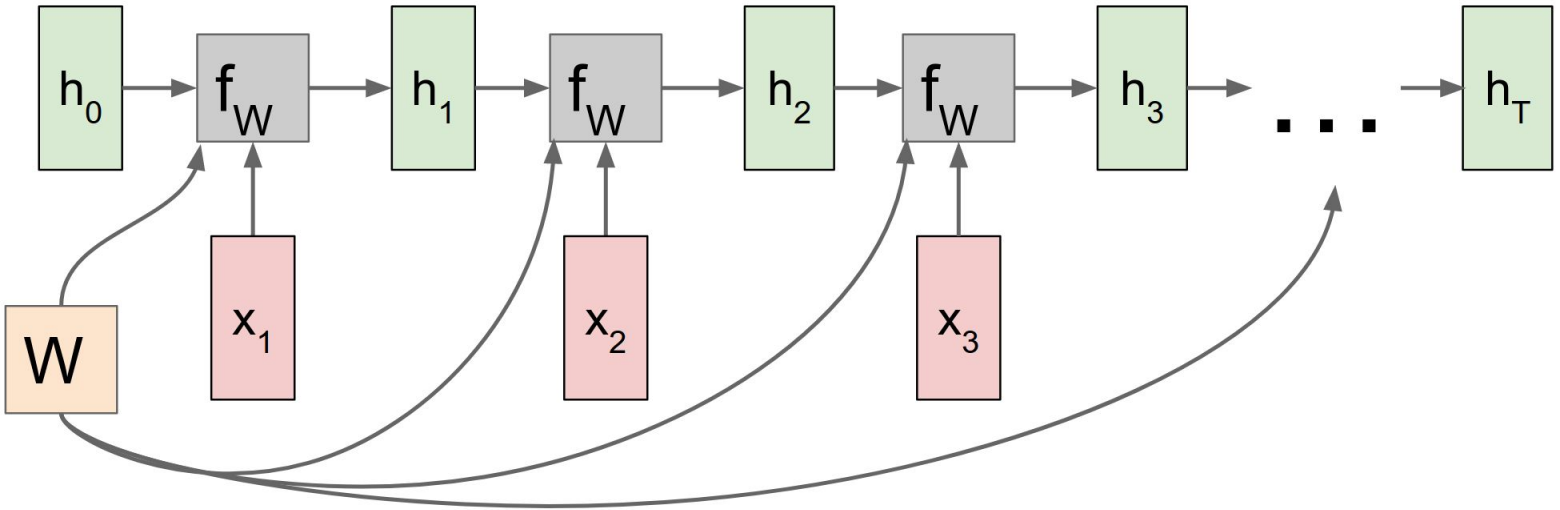


RNN : Computation Graph

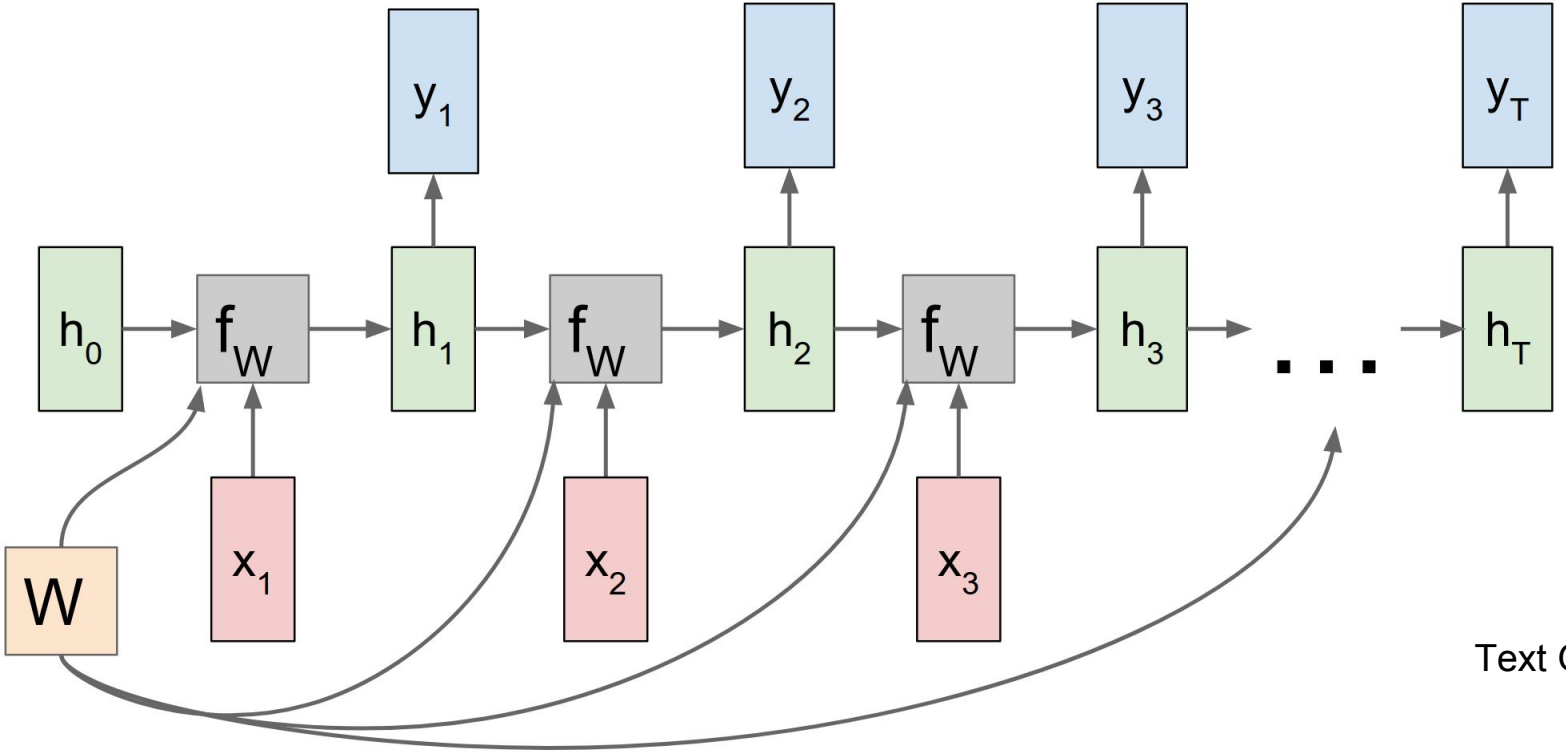


RNN : Computation Graph

Re-use the same weight matrix at every time-step

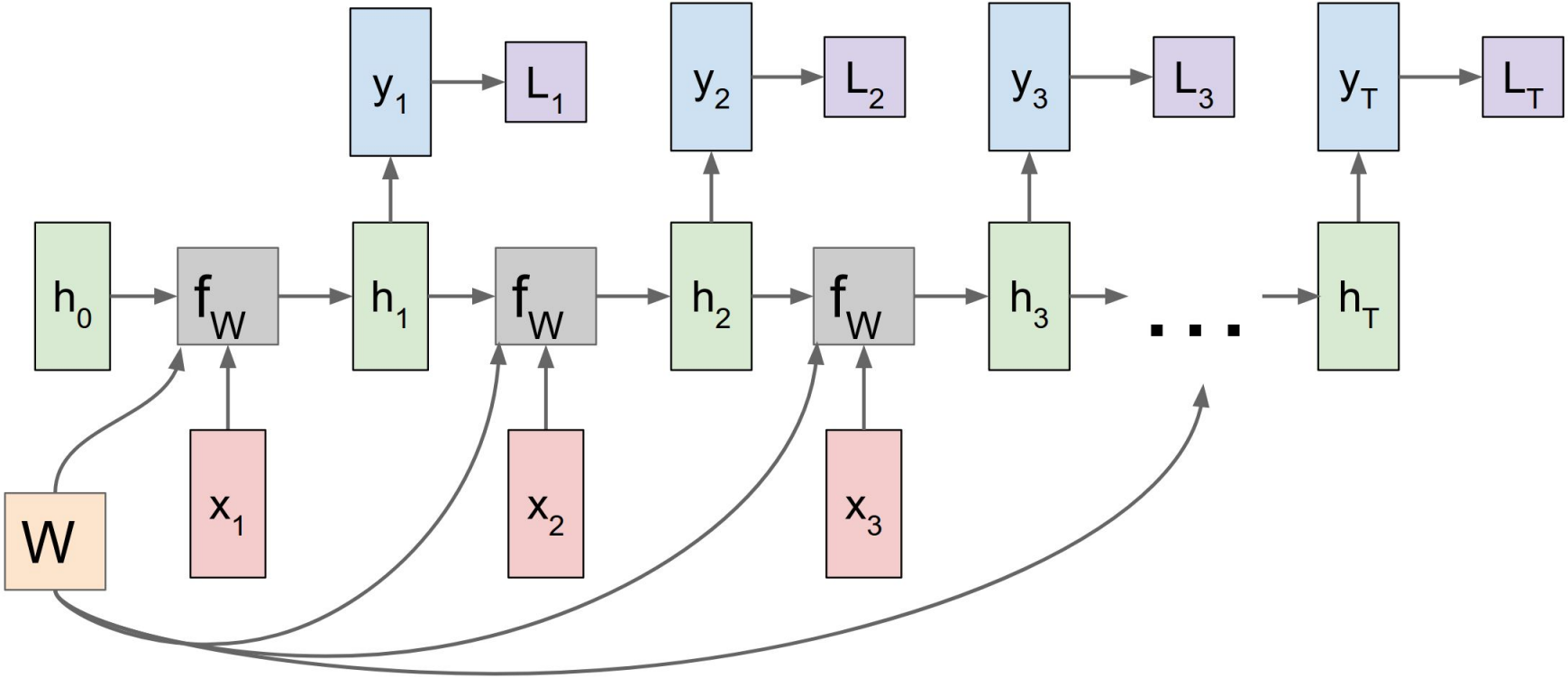


RNN: Many to Many

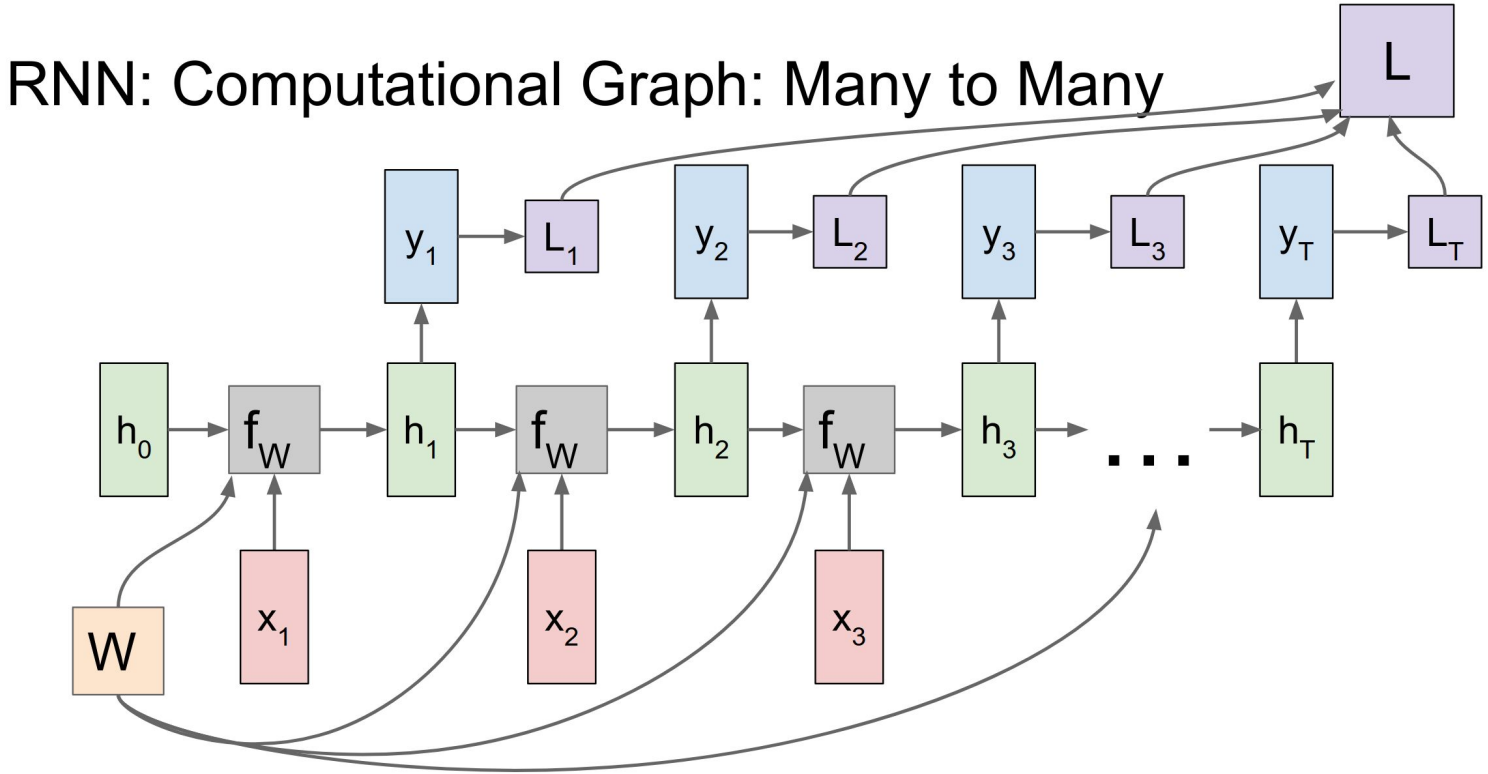


Text Generation

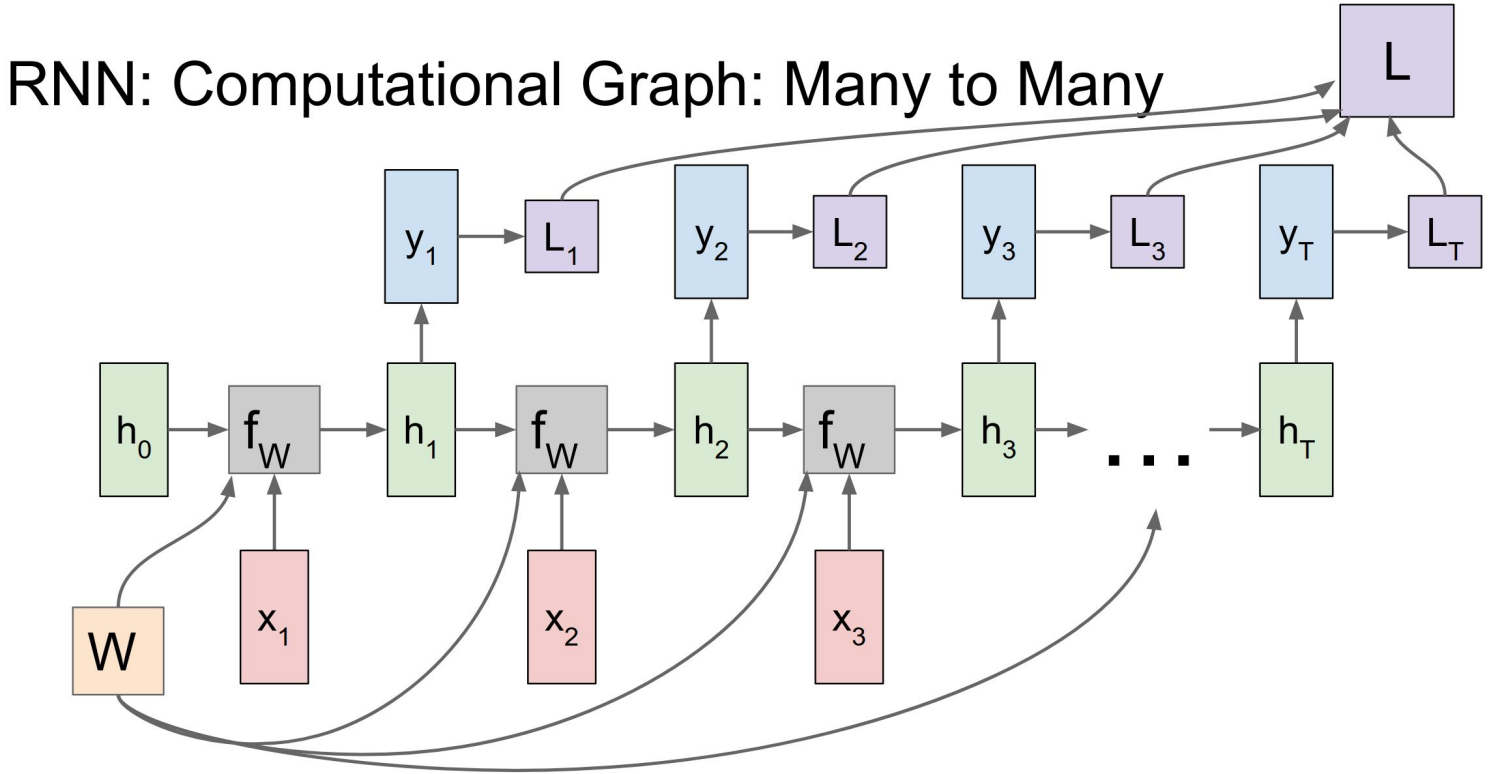
Loss Function for Many to Many



Loss Function for Many to Many

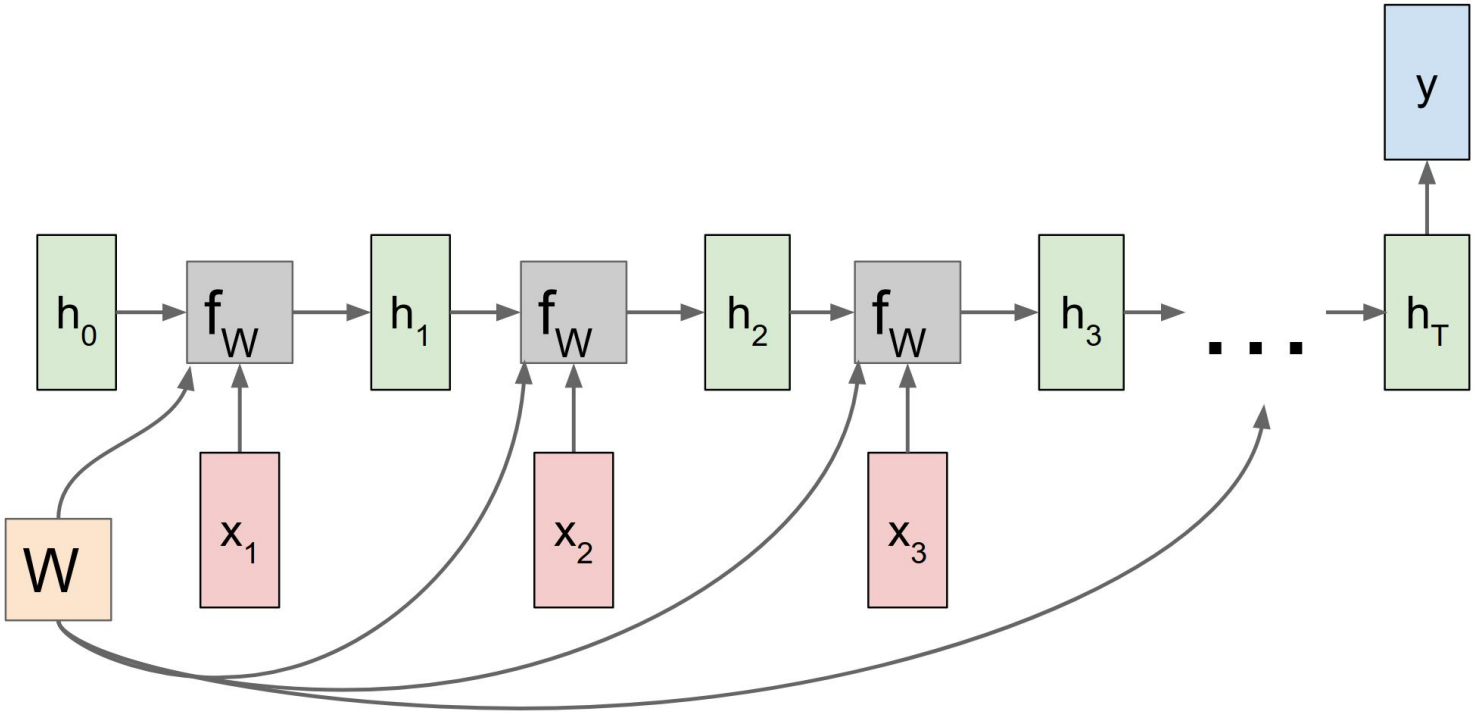


Loss Function for Many to Many



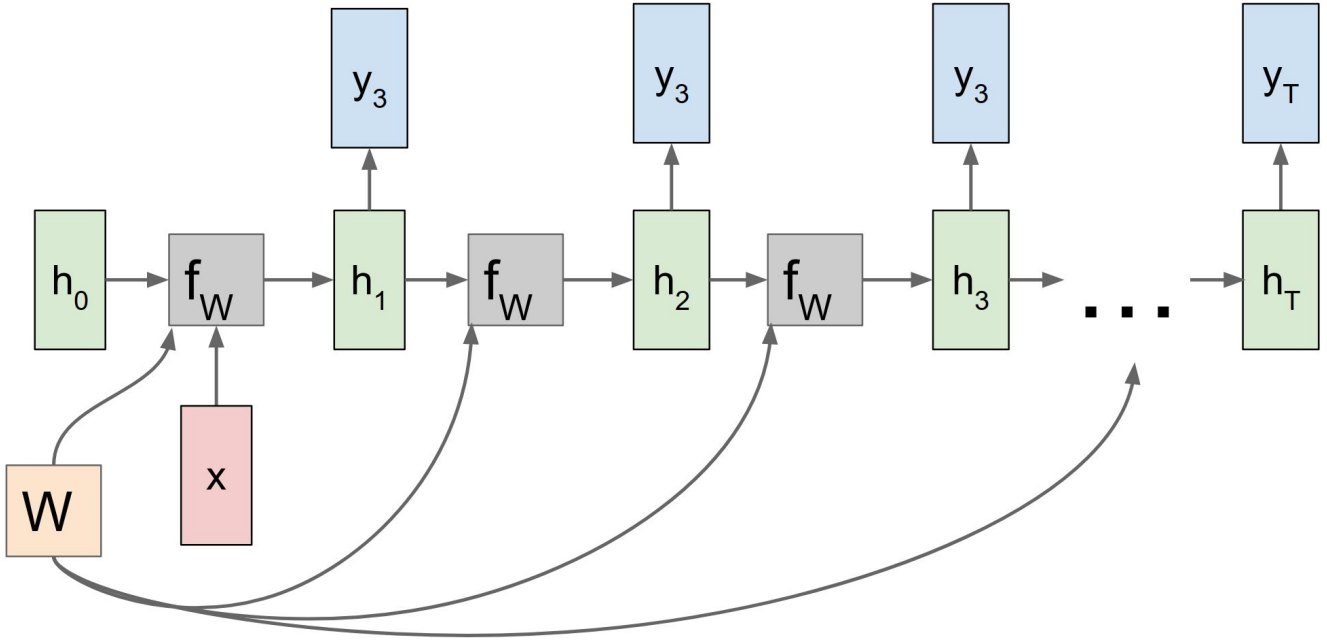
Many to One (Sentiment Analysis)

RNN: Computational Graph: Many to One



One to Many (Image Captioning)

RNN: Computational Graph: One to Many



Summary

- CNNs
  a. Window size, Stride, Padding
  b. Input/Output channels
  c. Parameters and FLOPs
  d. Pooling
- CNN Architectures (LeNet, AlexNet)
- RNN
  a. Definition Formulas
  b. Different Modes of Operation

References

- http://cs231n.github.io/convolutional-networks/
- http://colah.github.io/posts/2014-07-Conv-Nets-Modular/
-

GPU and Deep Learning

CPU vs GPU

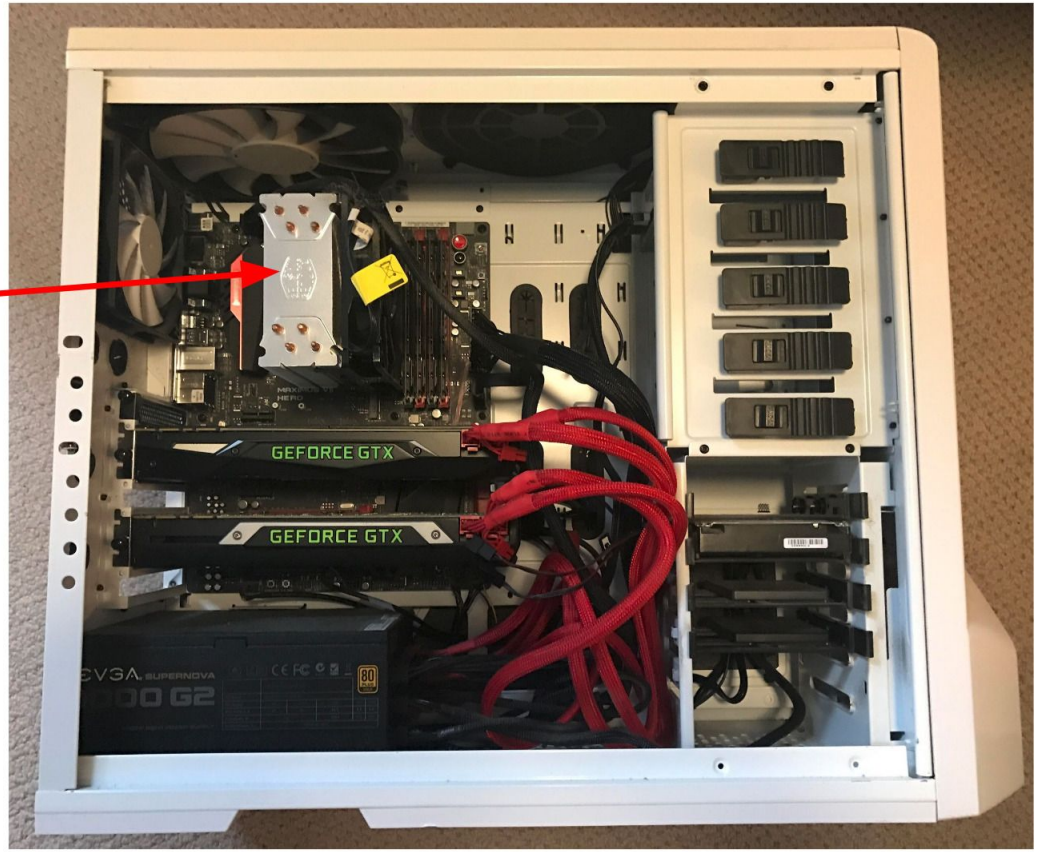
GPU and Deep Learning

My computer

GPU and Deep Learning

Spot the CPU!

(central processing unit)



GPU and Deep Learning

Spot the GPUs!

(graphics processing unit)

This image is in the public domain

GPU and Deep Learning

CPU vs GPU

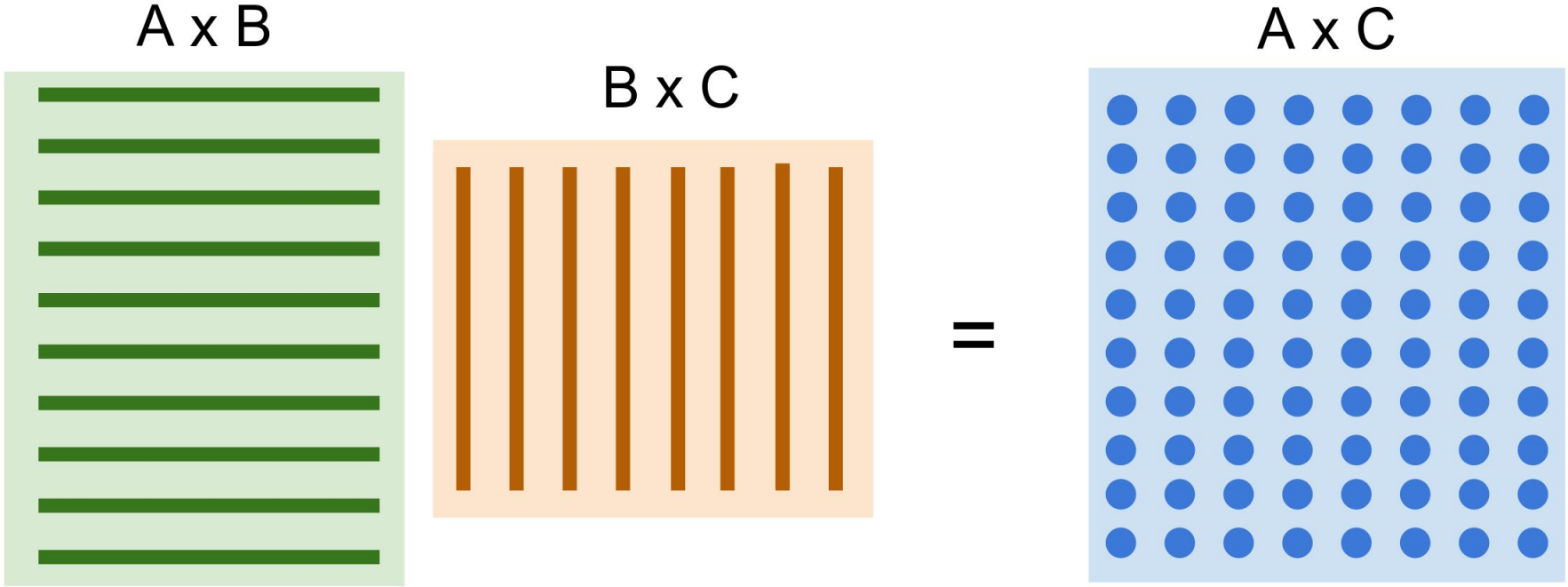
	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and “dumber”; great for parallel tasks

GPU and Deep Learning

Example: Matrix Multiplication



GPU and Deep Learning

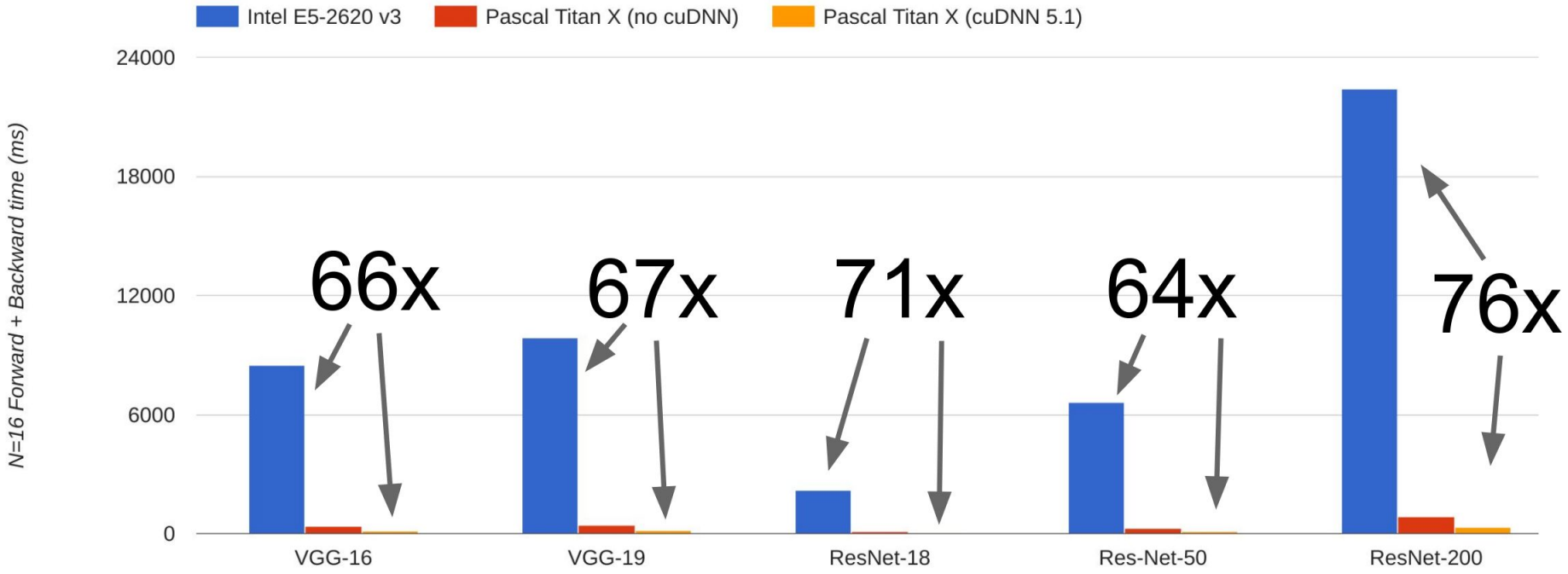
Programming GPUs

- CUDA (NVIDIA only)
  - Write C-like code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower :(

GPU and Deep Learning

CPU vs GPU in practice

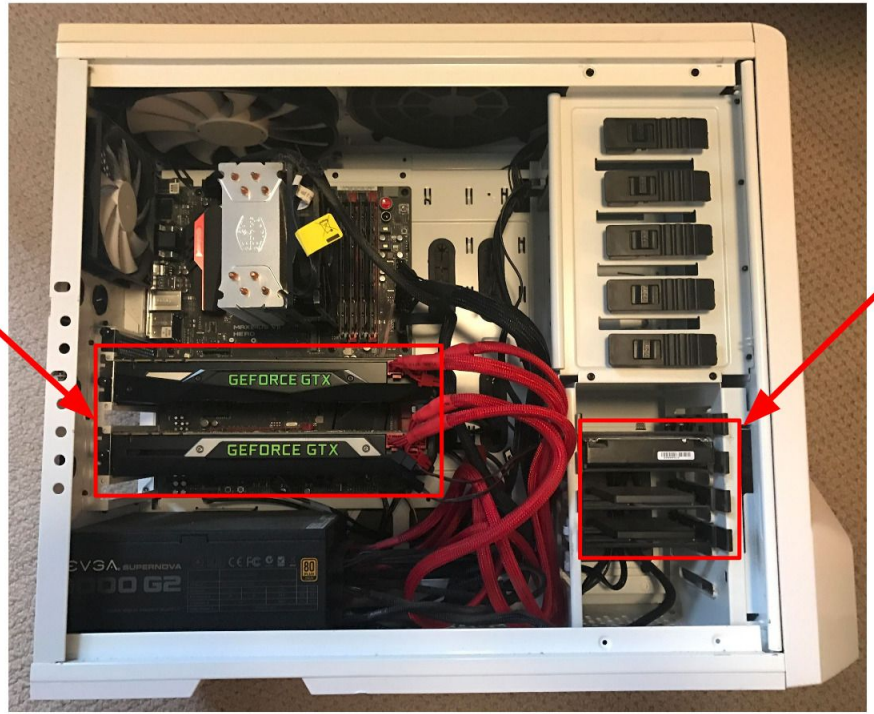
(CPU performance not well-optimized, a little unfair)



GPU and Deep Learning

CPU / GPU Communication

Model is here

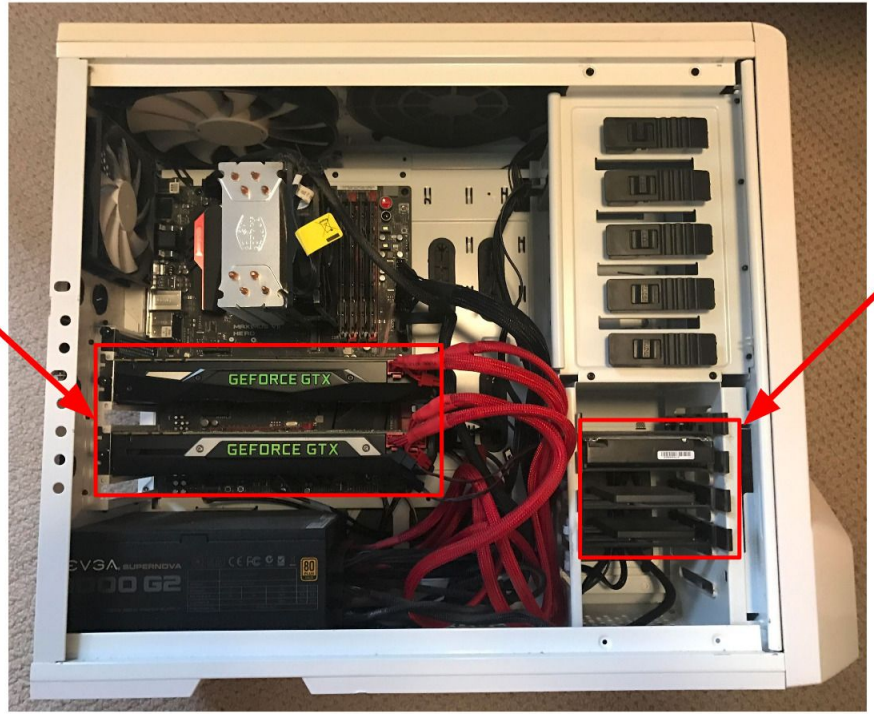


Data is here

GPU and Deep Learning

CPU / GPU Communication

Model is here



Data is here

If you aren't careful, training can bottleneck on reading data and transferring to GPU!

Solutions:

- Read all data into RAM
- Use SSD instead of HDD
- Use multiple CPU threads to prefetch data


AIML