北京大學

Artificial Neural Networks: Advanced

AIPKU

人工智能引论 第11课

主讲人：刘家瑛

2019年4月15日

Slides Credit

- *Slides modified from Feifei Li.*

Review

- Artificial Neurons
  - **Perceptrons** can do simple classification tasks
  - **Multi-Layer Non-Linear** Neural Network can approximate a wide range of functions
  - **Back-Propagation** with the chain rule to train the network
- Convolutional Neural Networks
  - Convolution operates on **local spatial** area
  - **Fewer parameters** compared with fully connected networks
  - **Automatically** extracts **features** from images

Today

- CNN Architectures
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- Recurrent Neural Network
  - Vanilla RNN
  - Backpropagation through time
  - Long Short-Term Memory
- Beyond CNN and RNN
  - Unsupervised Learning
  - Generative Adversarial Network

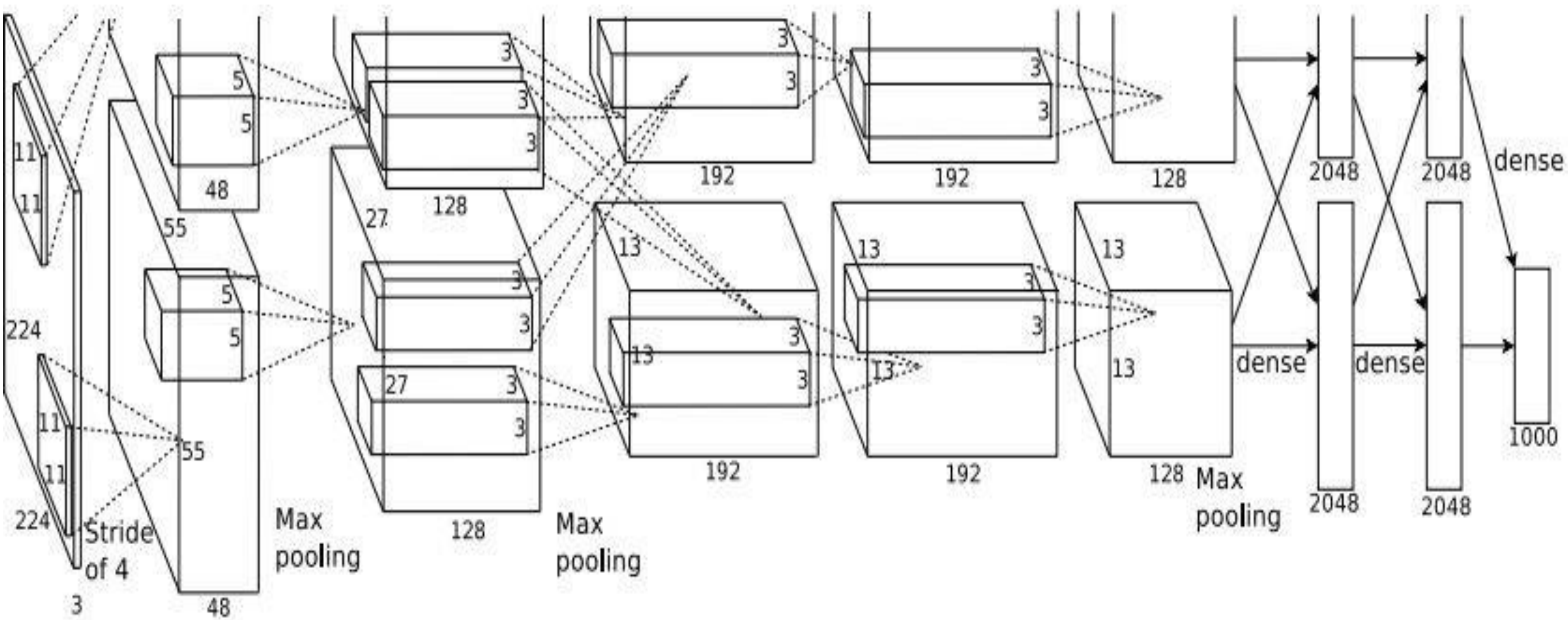
Today

- **CNN Architectures**
  - **AlexNet**
  - **VGG**
  - **GoogLeNet**
  - **ResNet**
- Recurrent Neural Network
  - Vanilla RNN
  - Backpropagation through time
  - Long Short-Term Memory
- Beyond CNN and RNN
  - Unsupervised Learning
  - Generative Adversarial Network

Alex Net [Krizhevsky *et al*. 2012]

Architecture:

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



Alex Net [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)

Details / Retrospectives:

- First use of ReLU
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% → 15.4%

Alex Net [Krizhevsky *et al*. 2012]

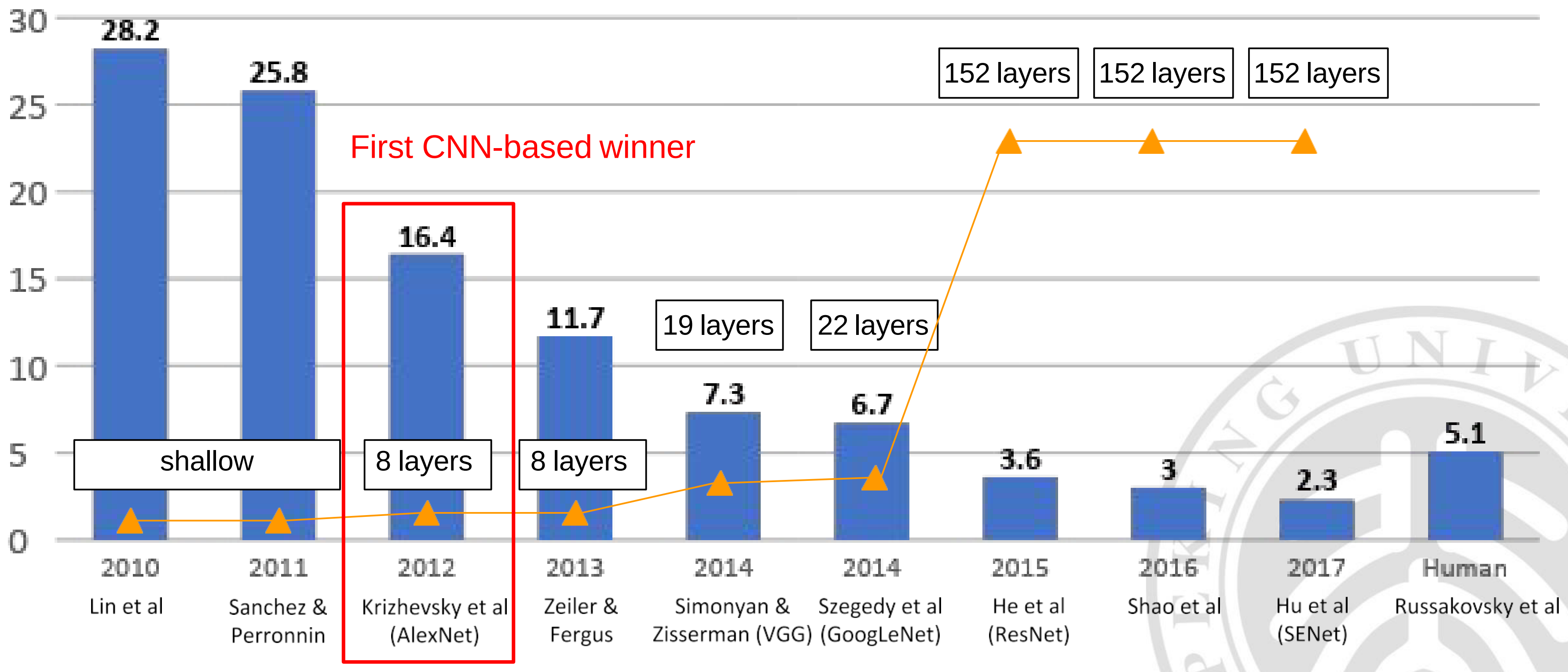
Full (simplified) AlexNet architecture:

[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)

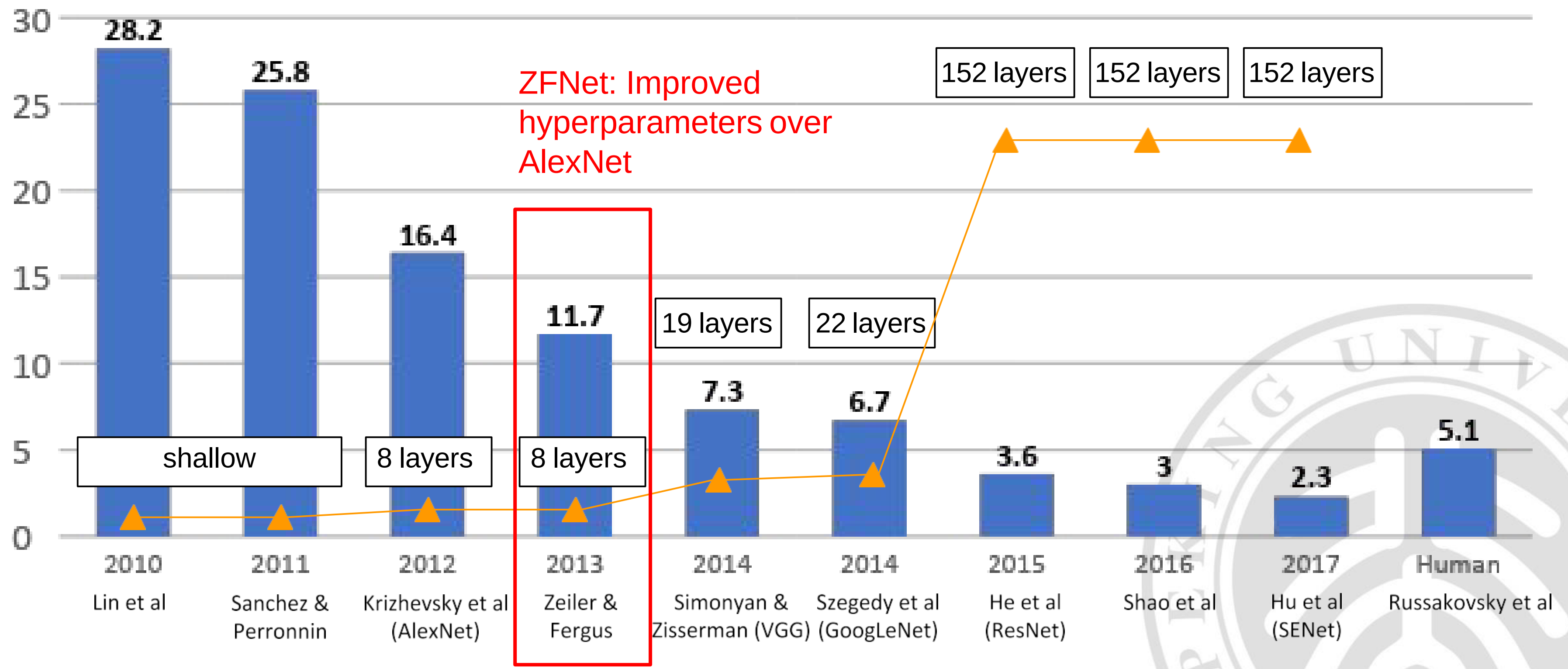
Details/Retrospectives:

- First use of ReLU
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% → 15.4%

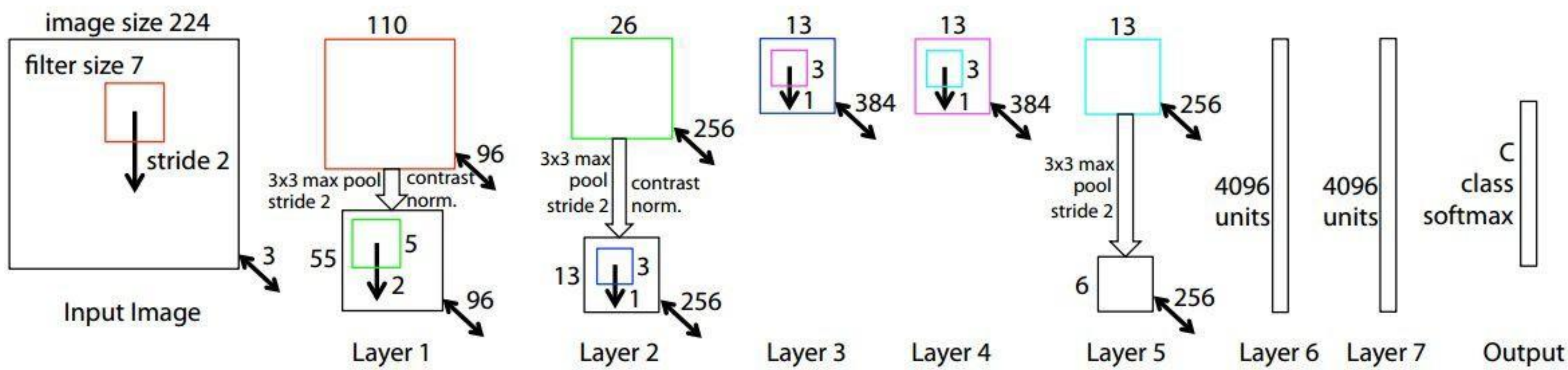
ILSVRC Winners



ILSVRC Winners



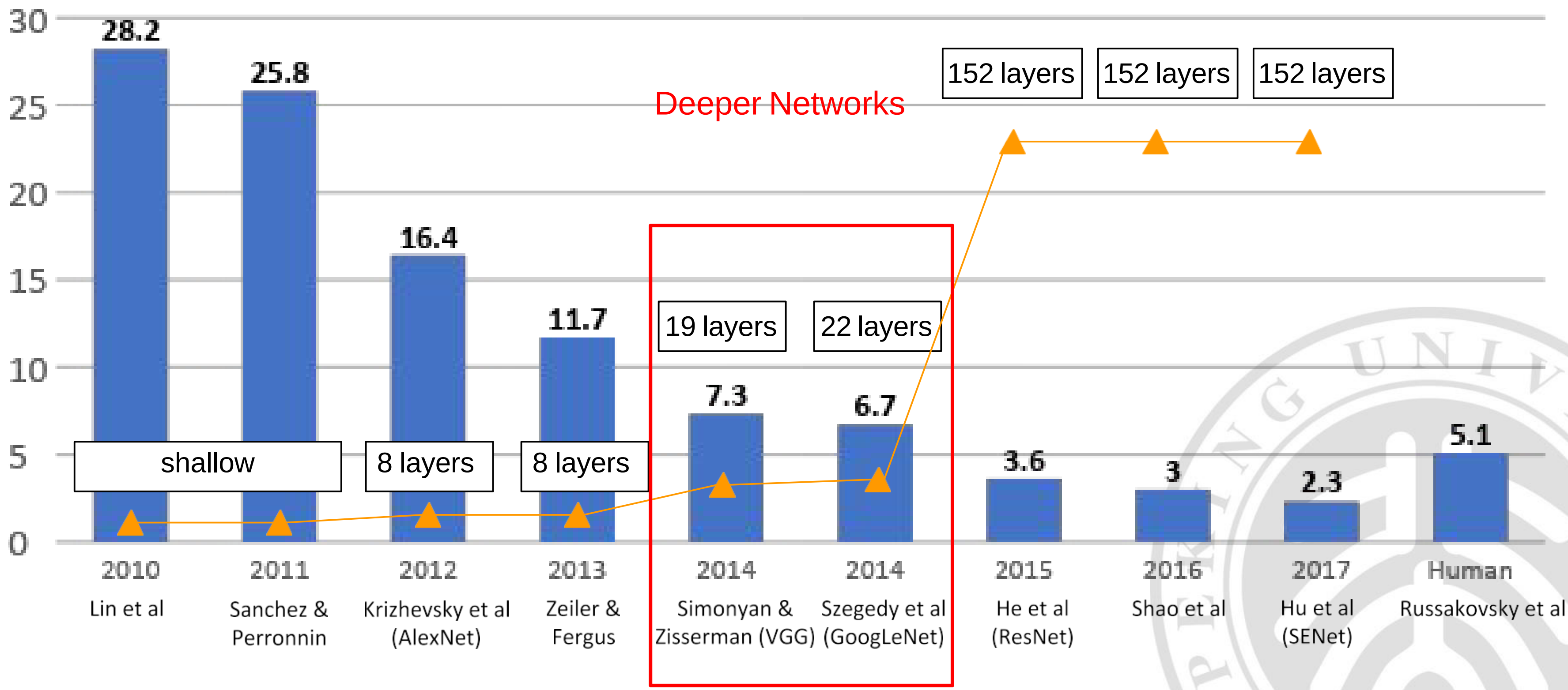
ZFNet [Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) x48 to (7x7 stride 2) x96
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% → 11.7%

ILSVRC Winners



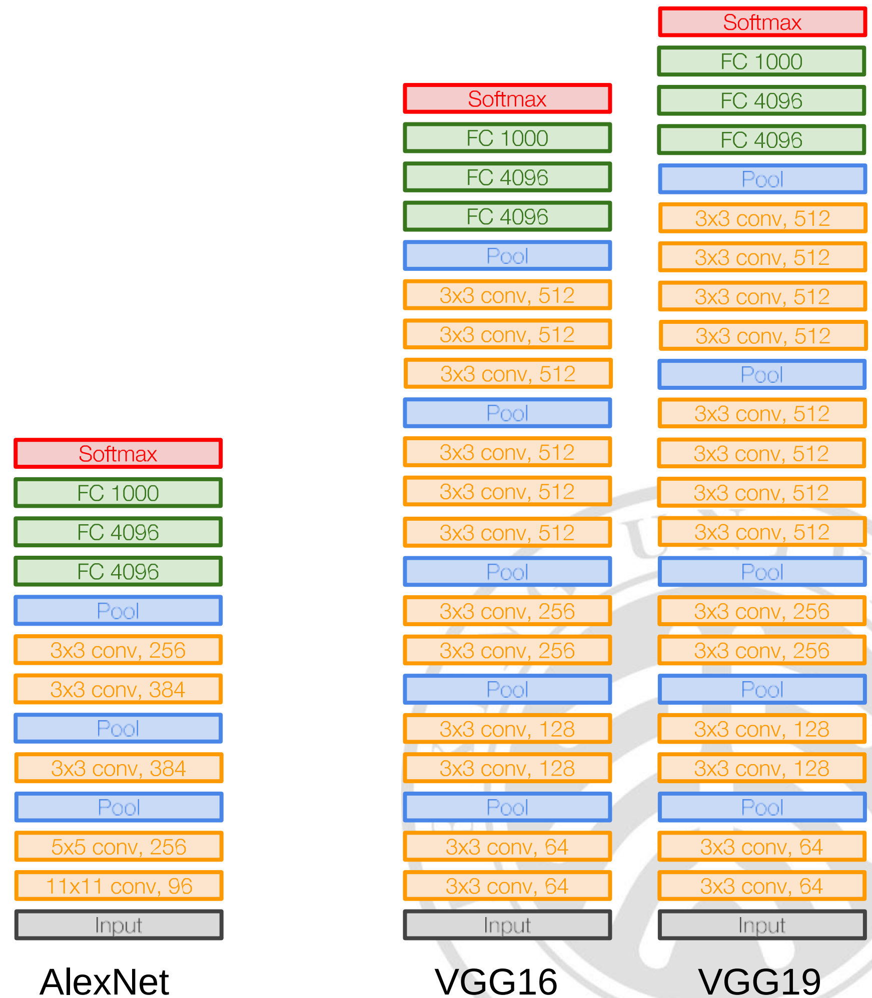
Case Study: VGGNet [Simonyan and Zisserman, 2014]

Small filters, deeper networks

8 layers (AlexNet)
→ 16 - 19 layers (VGG16Net)

Only 3x3 CONV: stride 1, pad 1 and 2x2 MAX POOL: stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
→ 7.3% top 5 error in ILSVRC'14



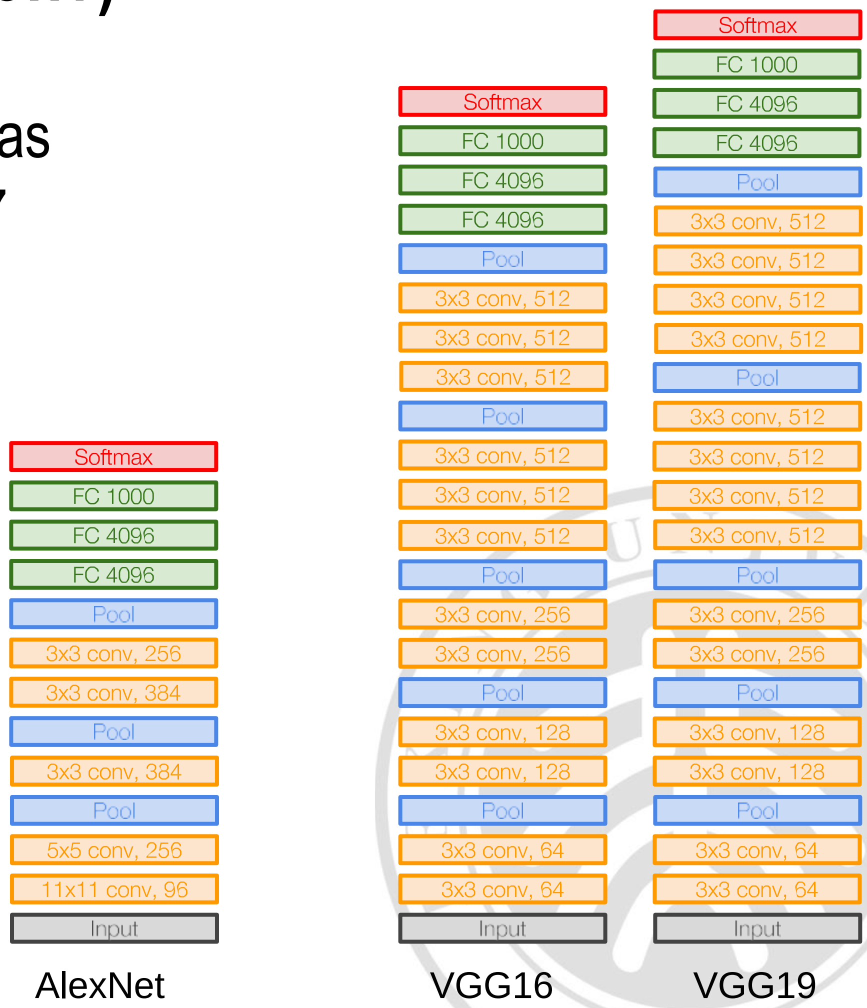
Case Study: VGGNet [Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

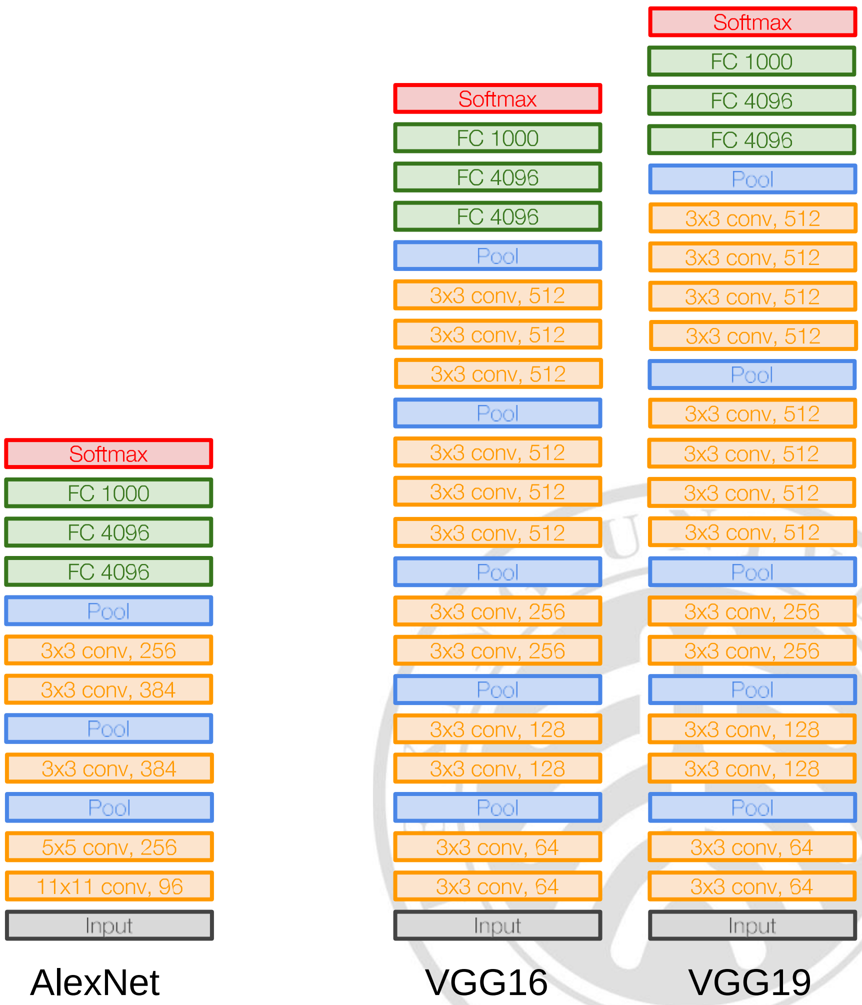
But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

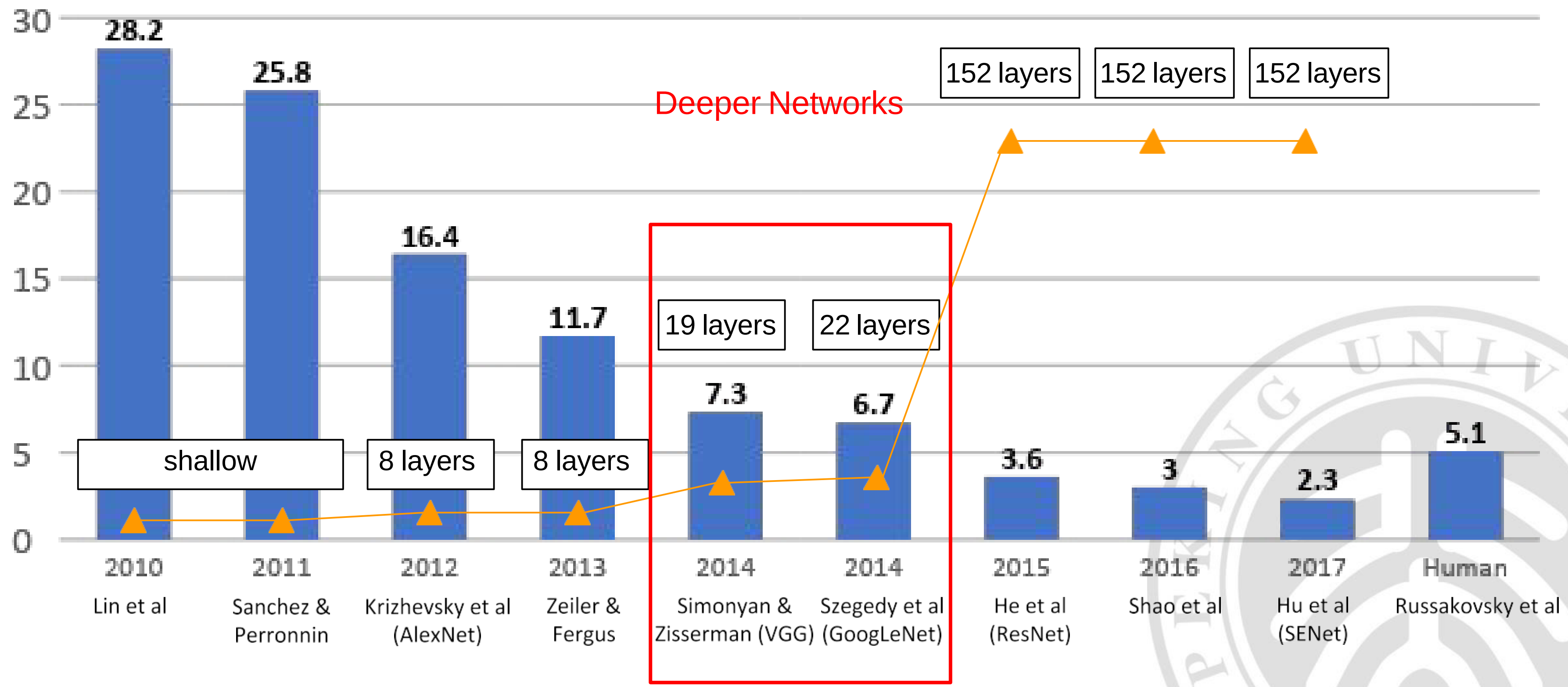


Case Study: VGGNet *[Simonyan and Zisserman, 2014]*

- **Details:**
  - ILSVRC'14 2nd in classification, 1st in localization
  - Similar training procedure as Krizhevsky 2012
  - No Local Response Normalisation (LRN)
  - Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
  - Use ensembles for best results
  - FC7 features generalize well to other tasks



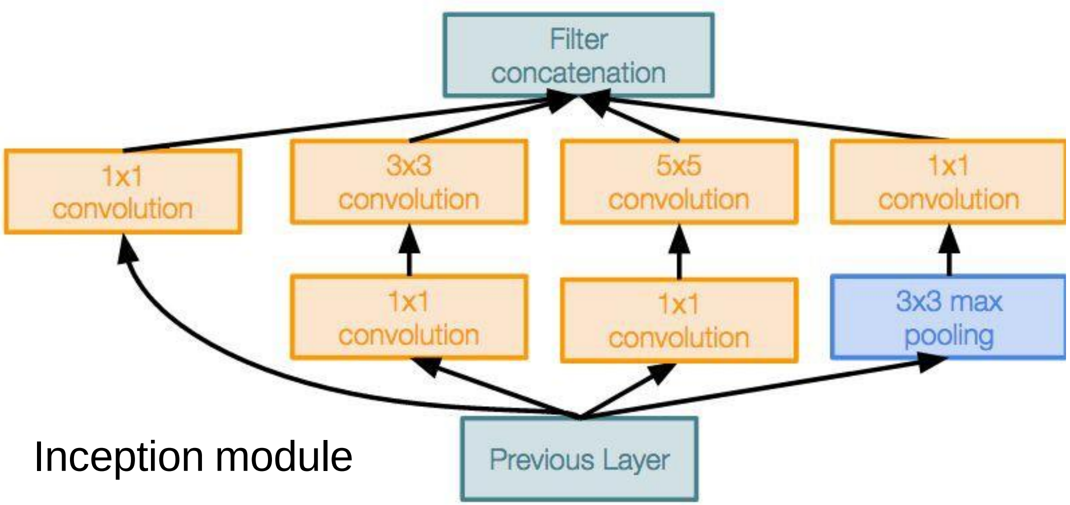
ILSVRC Winners



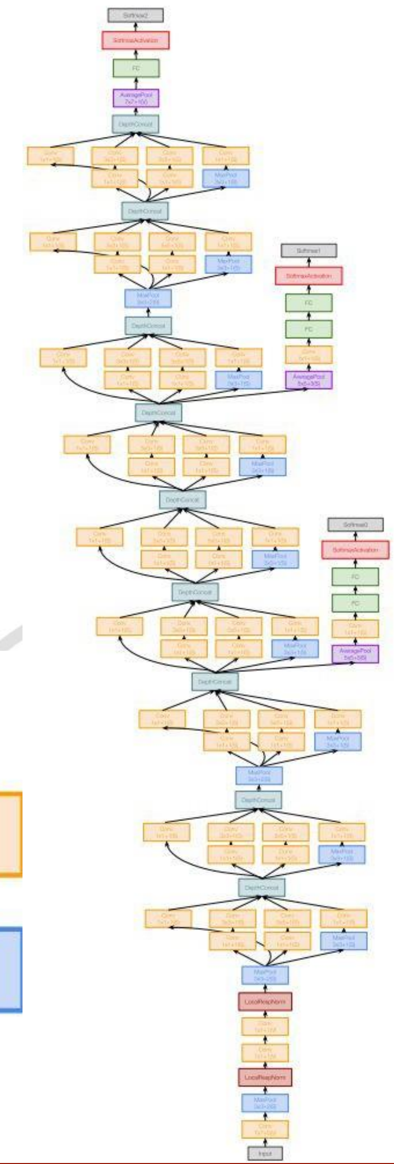
GoogLeNet [Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

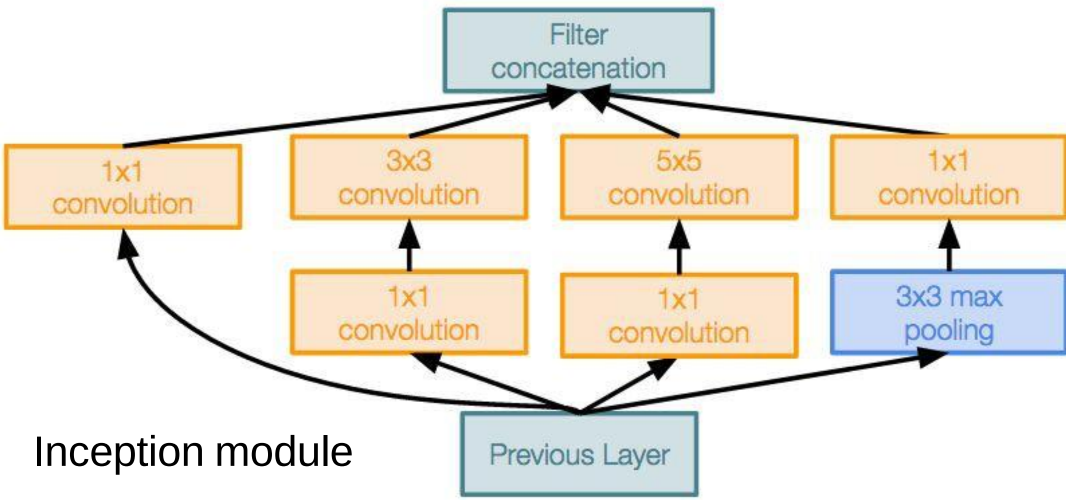


Inception module

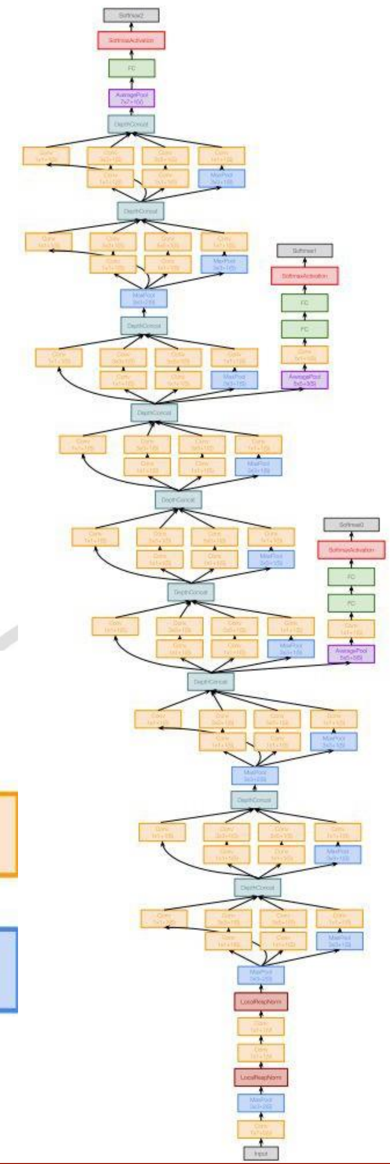


GoogLeNet [Szegedy et al., 2014]

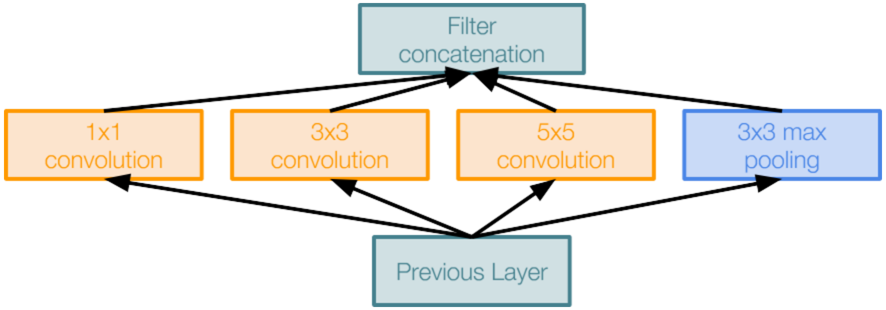
"Inception module":
Design a good local network topology (network within a network),
and then stack these modules on top of each other.



Inception module



GoogLeNet [Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

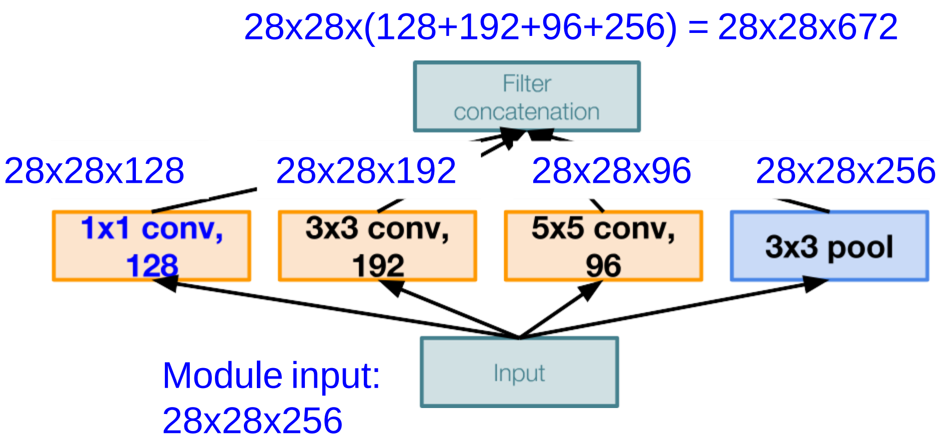
Concatenate all filter outputs together depth-wise

Q: What is the problem with this? [Hint: Computational complexity]

GoogLeNet [Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

Q: What is the problem with this? [Hint: Computational complexity]

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x192x3x3x256

[5x5 conv, 96] 28x28x96x5x5x256

Total: 854M ops

Very expensive compute

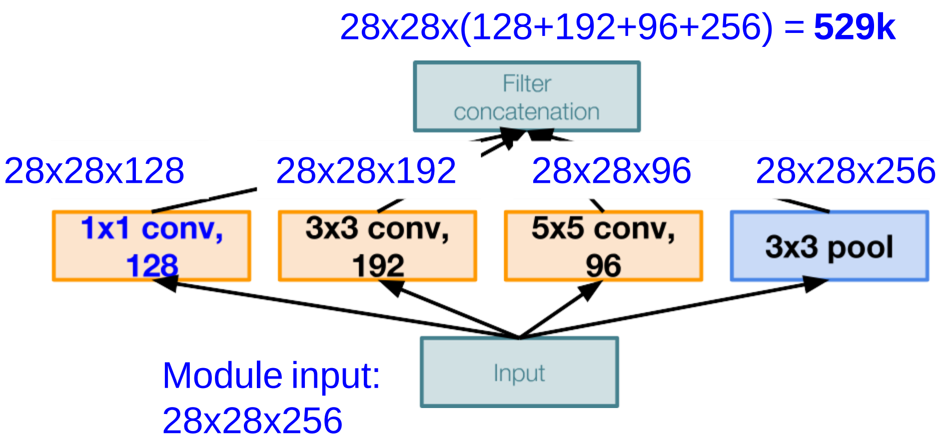
Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

GoogLeNet [Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:

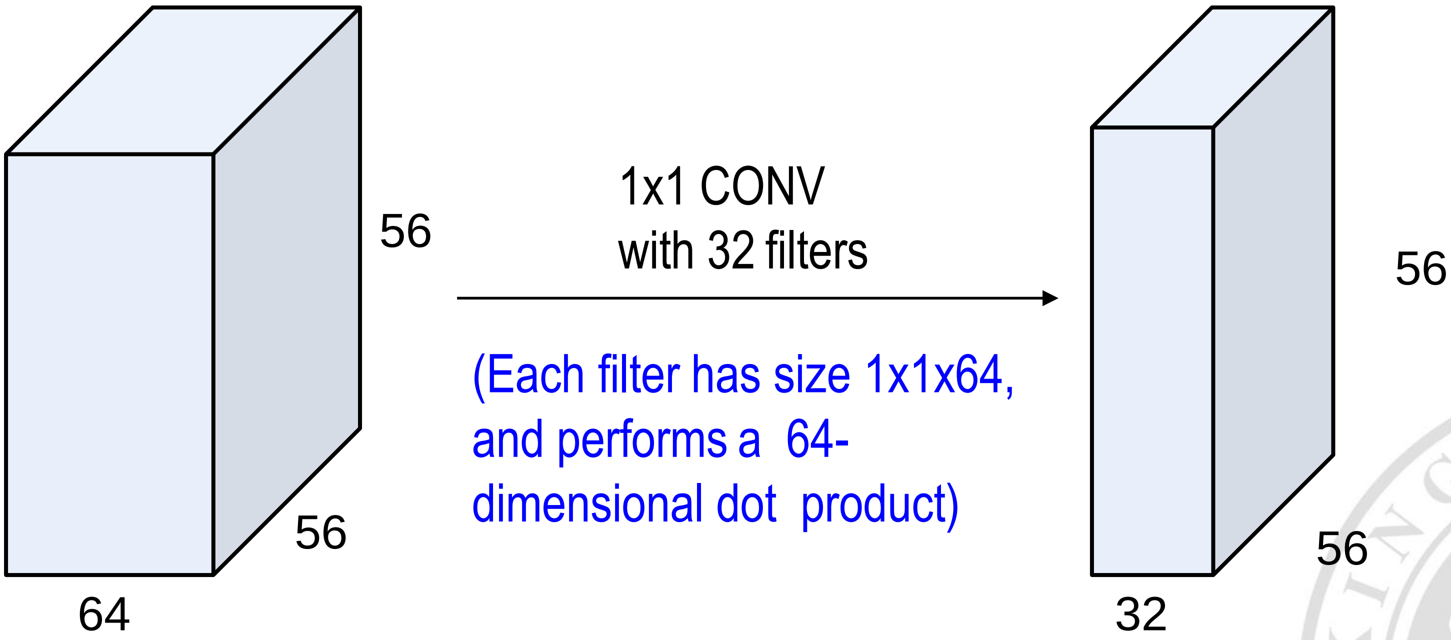
Q3:What is output size after filter concatenation?



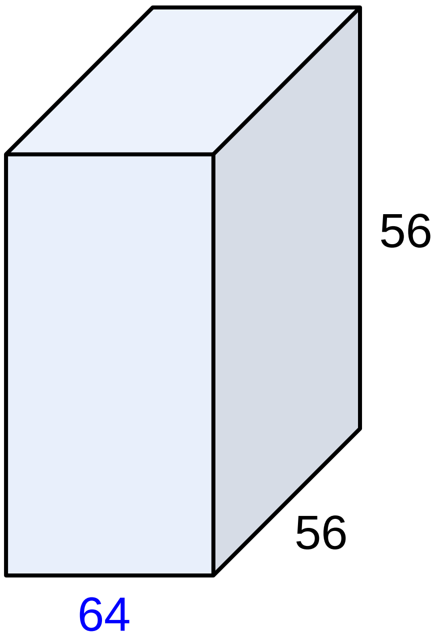
Naive Inception module

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth

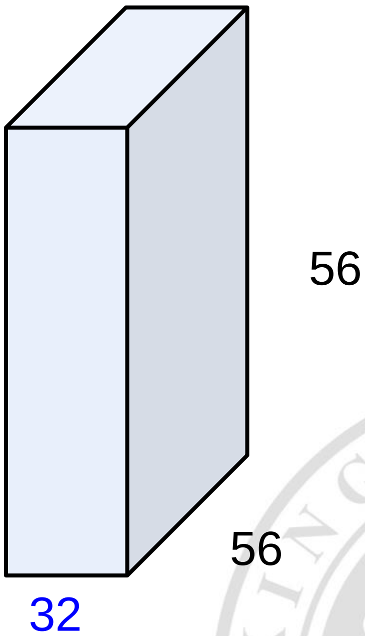
Reminder: 1x1 Convolution



Reminder: 1x1 Convolution



1x1 CONV
with 32 filters

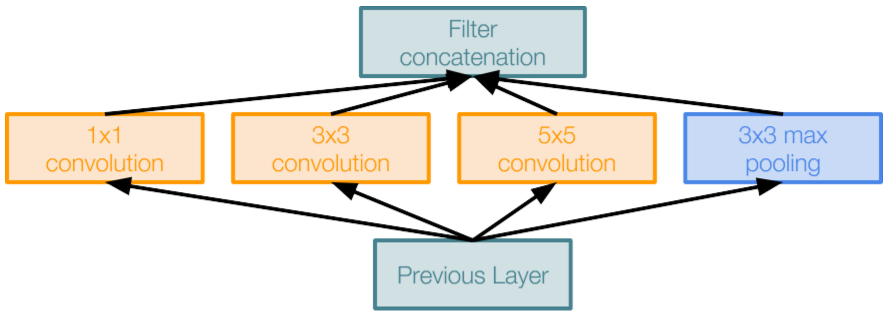


Preserves spatial dimensions, reduces depth!

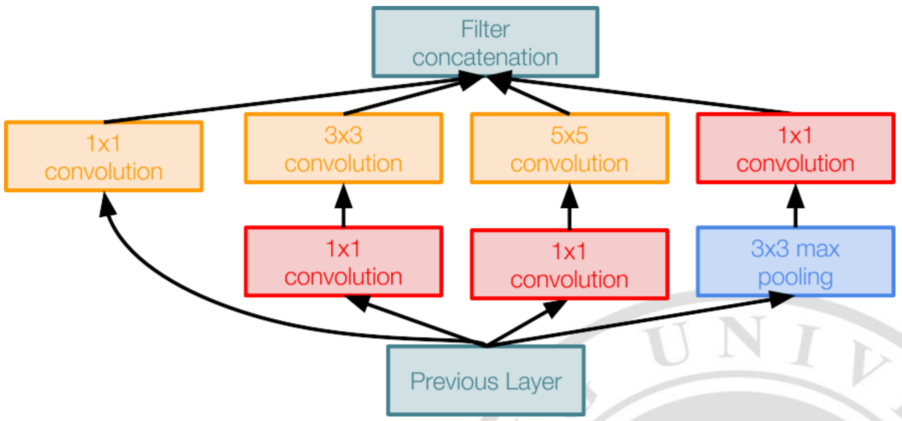
Projects depth to lower dimension (combination of feature maps)

GoogLeNet [Szegedy et al., 2014]

1x1 conv "bottleneck" layers

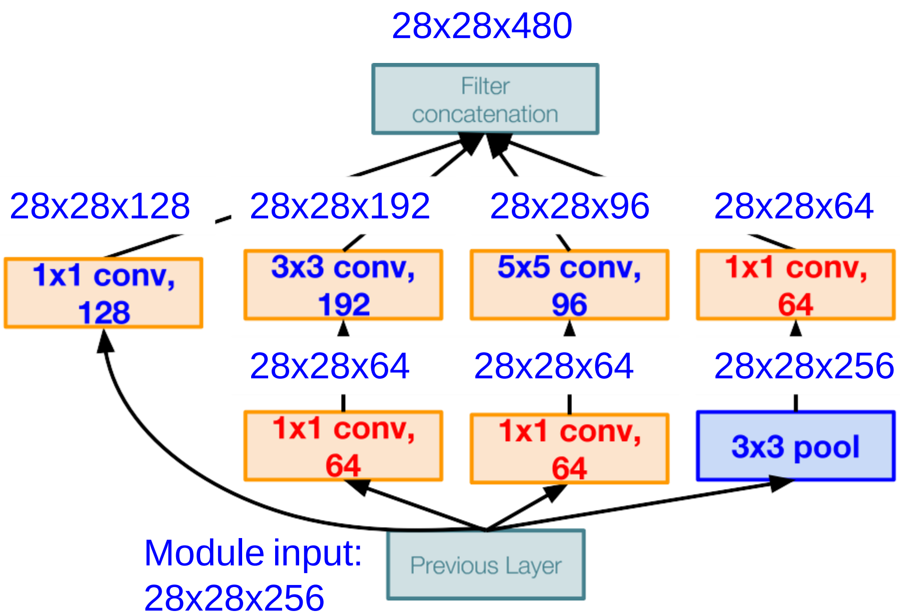


Naive Inception module



Inception module with dimension reduction

GoogLeNet [Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

Conv Ops:

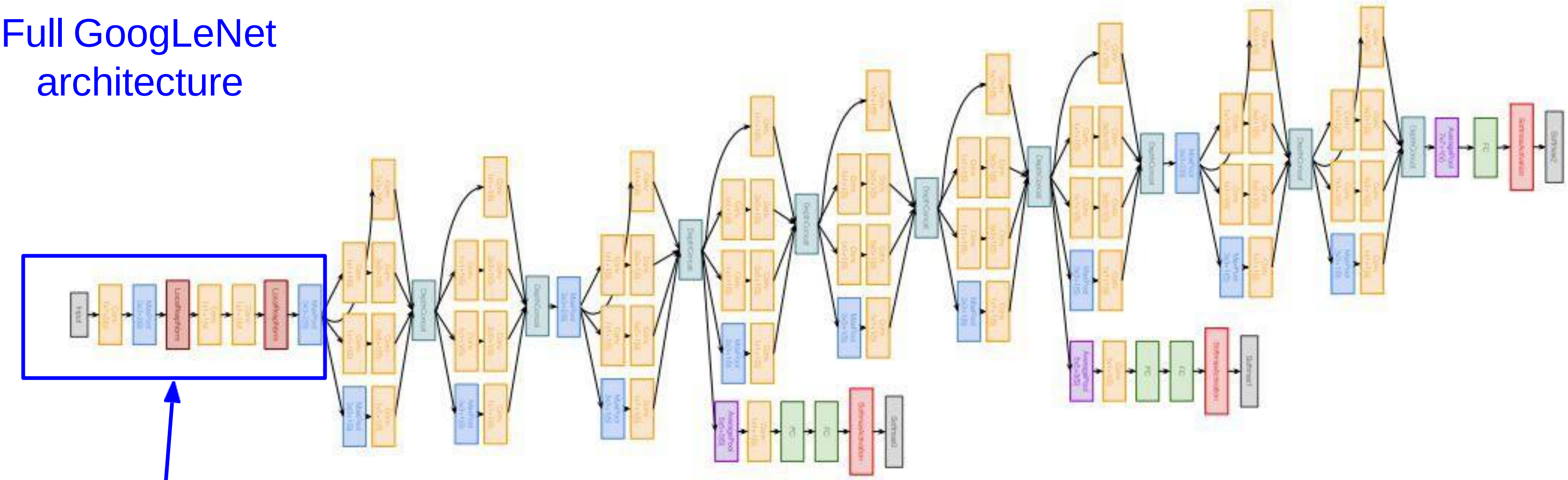
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

Total: 358M ops

Compared to 854M ops for naive version, bottleneck can also reduce depth after pooling layer

GoogLeNet [Szegedy et al., 2014]

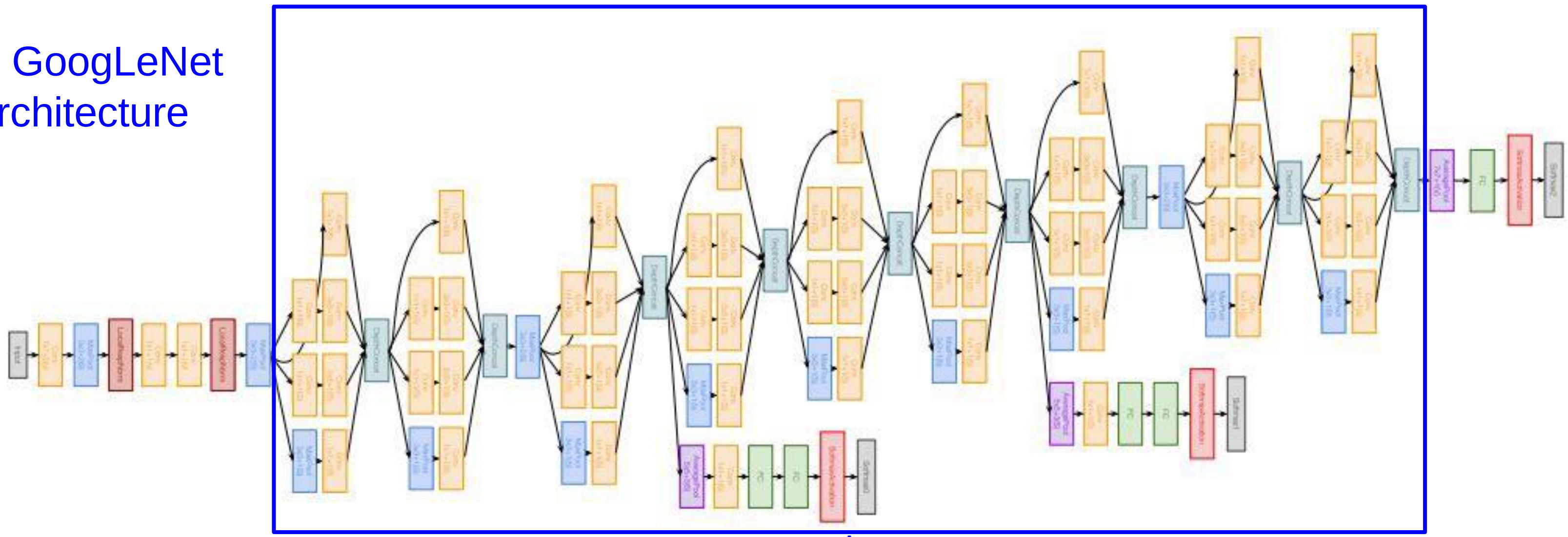
Full GoogLeNet architecture



Stem Network:
Conv-Pool-
2x Conv-Pool

GoogLeNet [Szegedy et al., 2014]

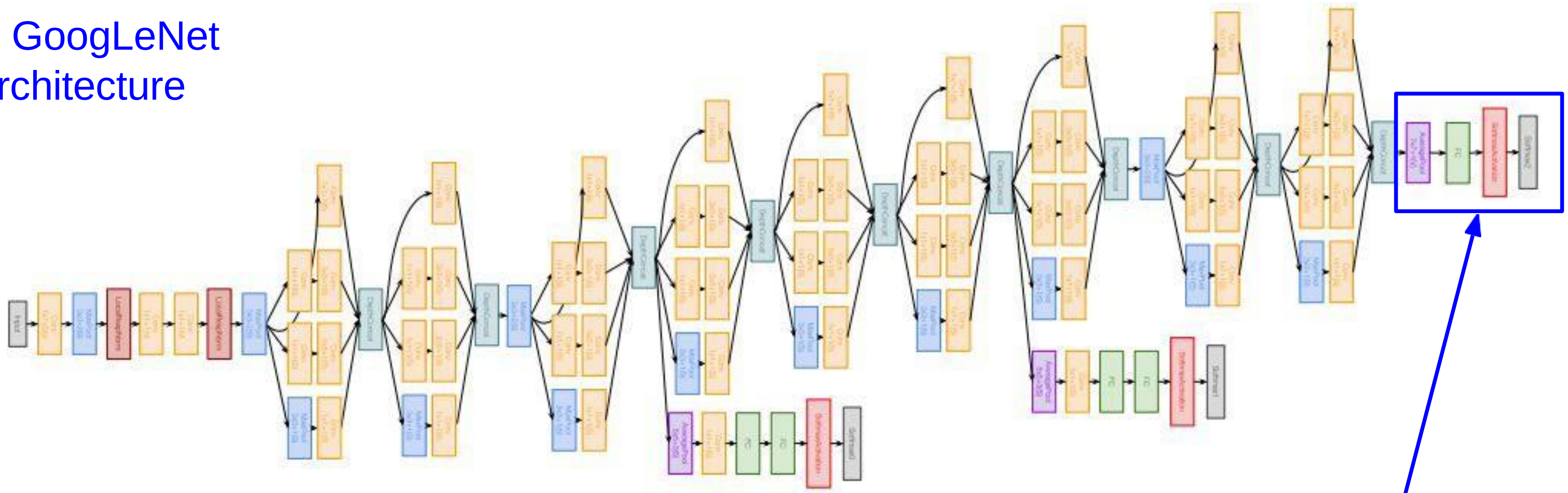
Full GoogLeNet architecture



Stacked Inception Modules

GoogLeNet [Szegedy et al., 2014]

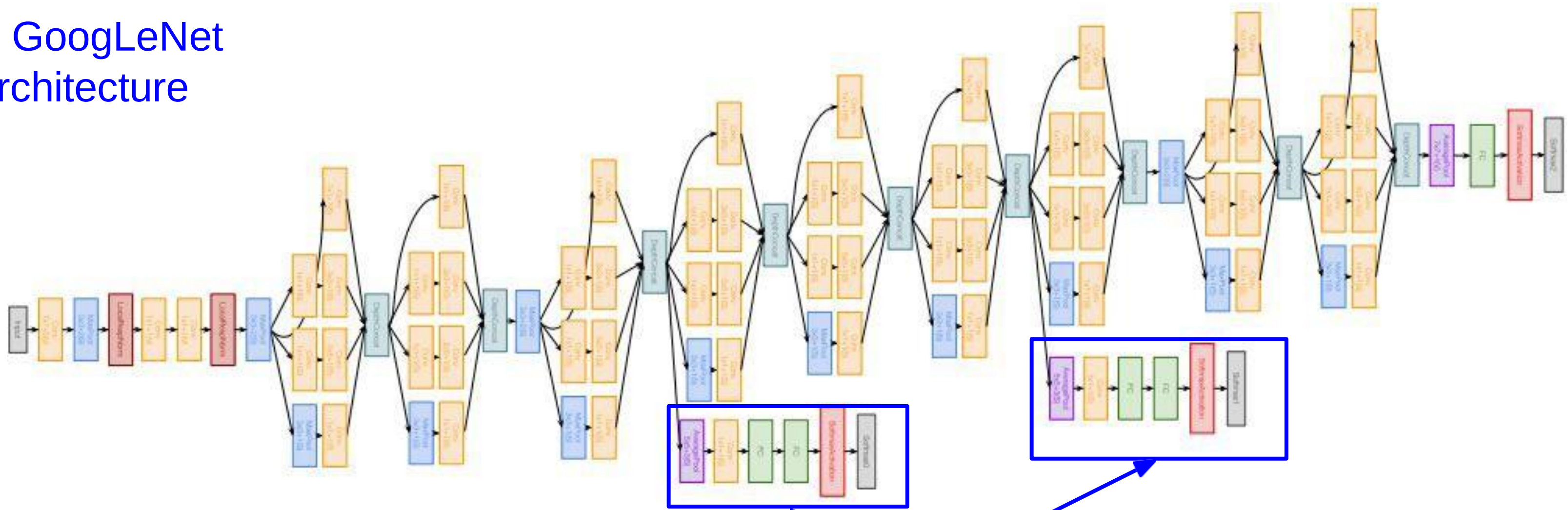
Full GoogLeNet architecture



Classifier output
(removed expensive FC layers!)

GoogLeNet [Szegedy et al., 2014]

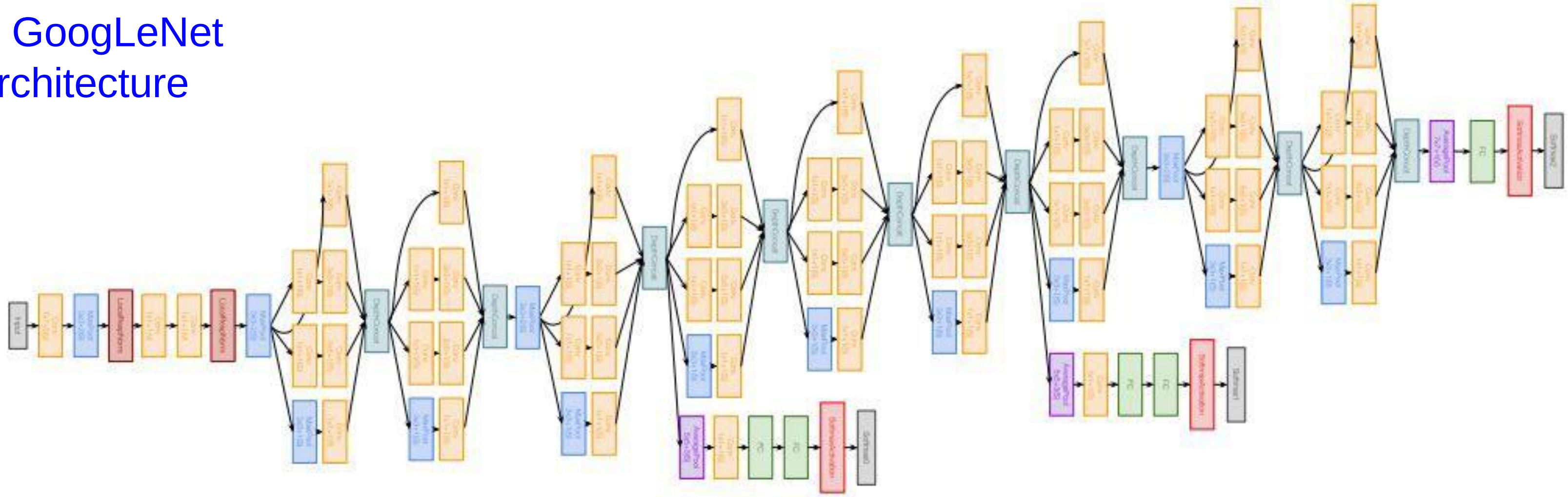
Full GoogLeNet architecture



Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

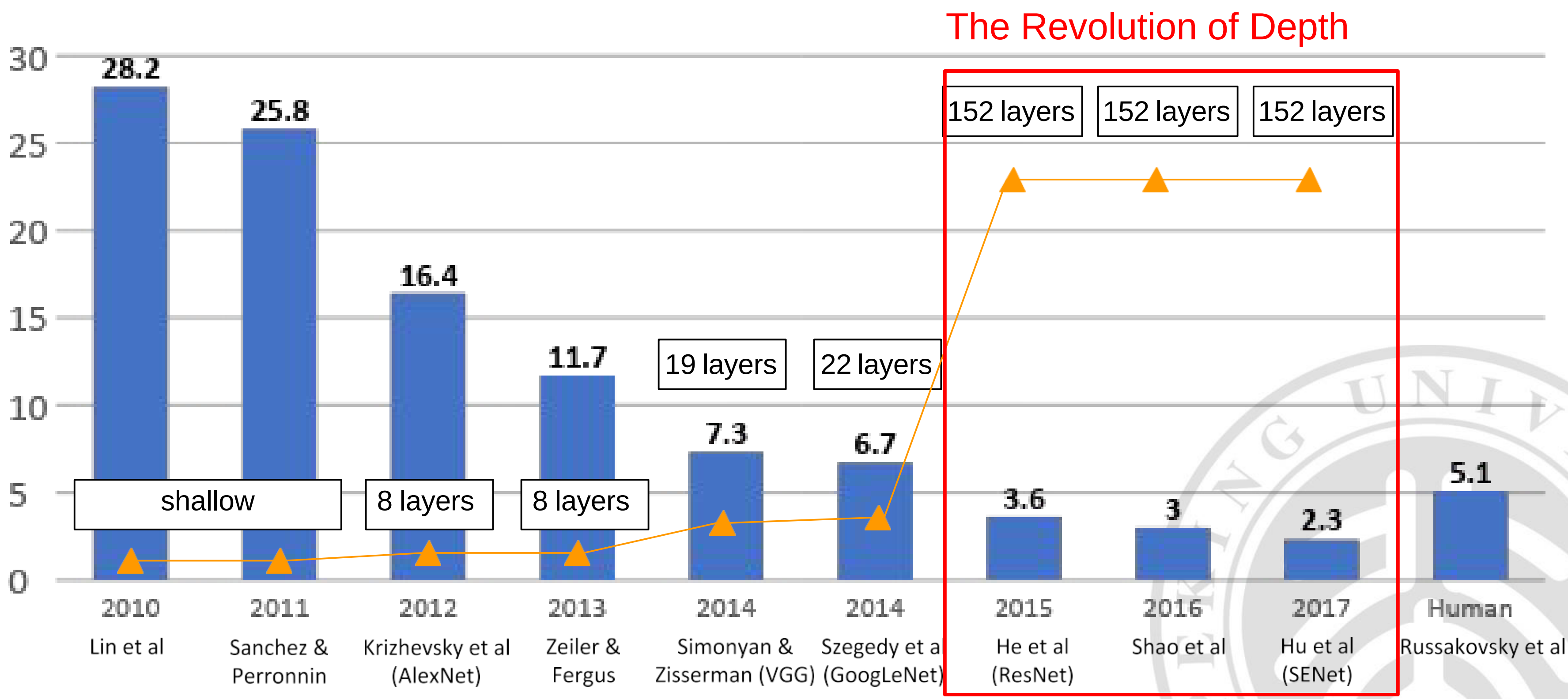
GoogLeNet [Szegedy et al., 2014]

Full GoogLeNet architecture

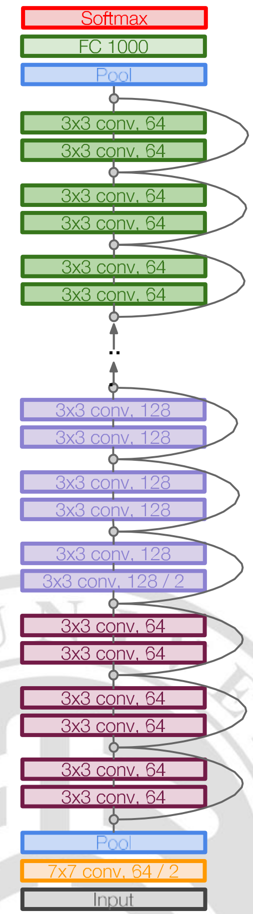


22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don’t count auxiliary output layers)

ILSVRC Winners

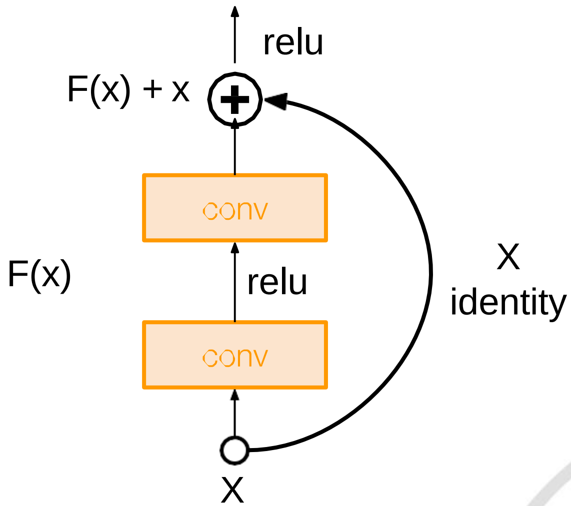


ResNet [He et al., 2015]



Very deep networks using **residual** connections

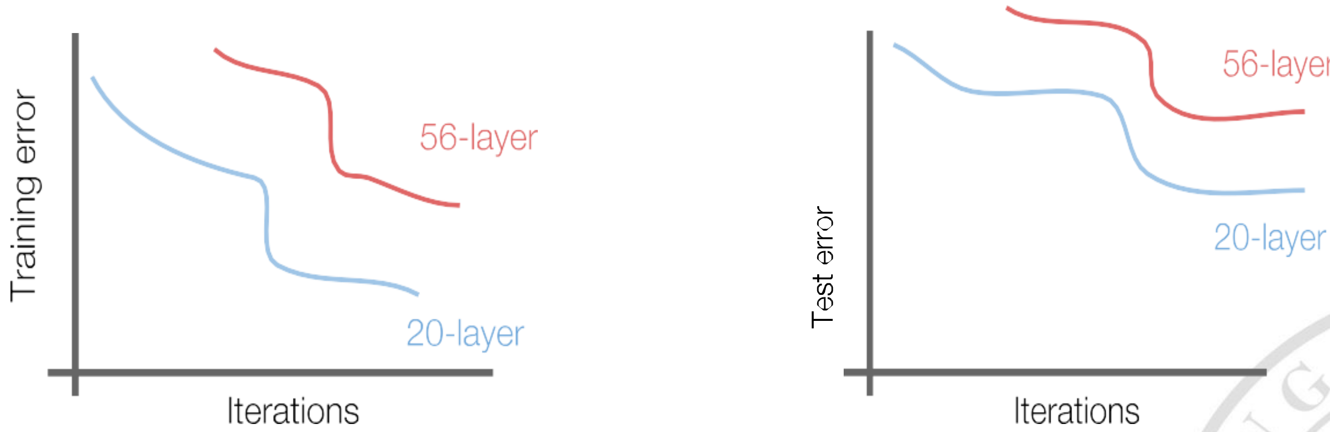
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



Residual block

ResNet [He et al., 2015]

What happens when we continue stacking deeper layers on a "plain" convolutional neural network?



56-layer model performs worse on both training and test error → The deeper model performs worse, but it's not caused by overfitting!

ResNet [He et al., 2015]

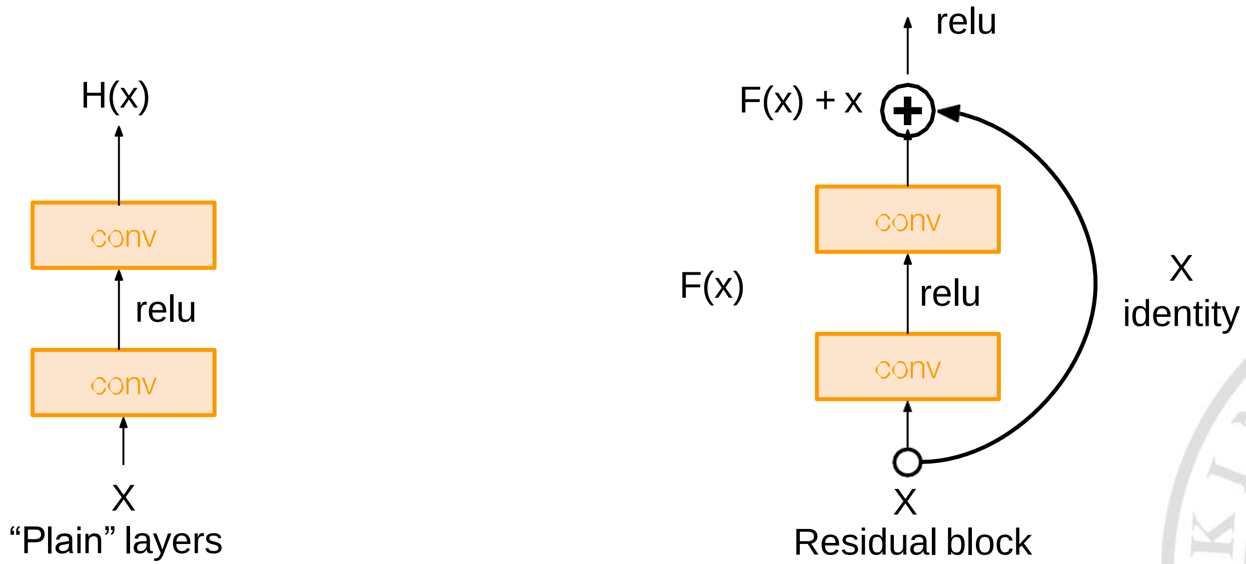
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

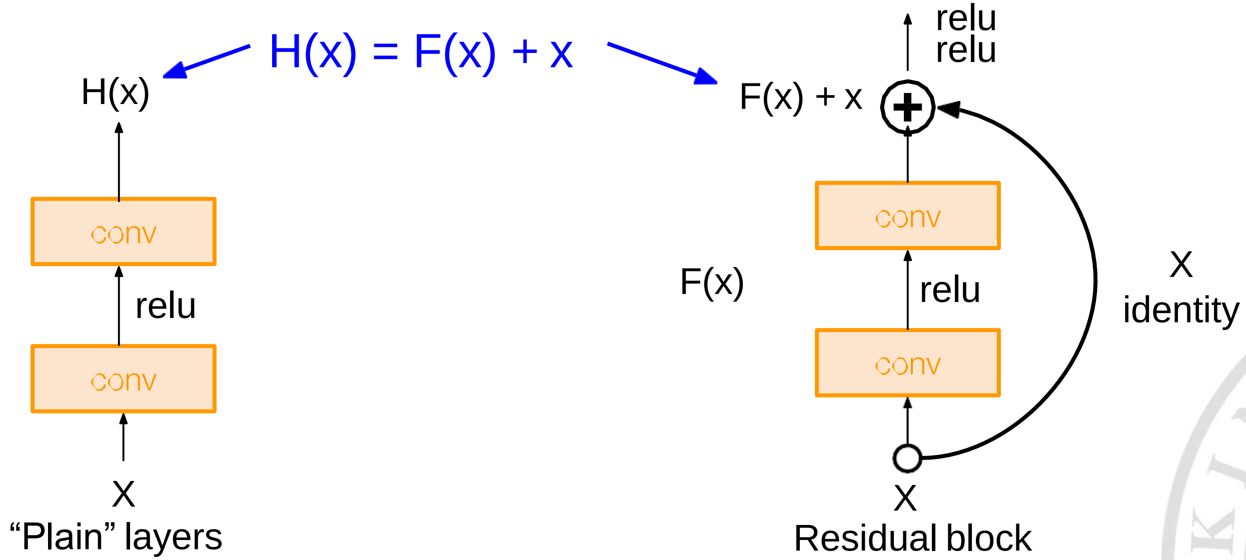
ResNet [He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet [He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

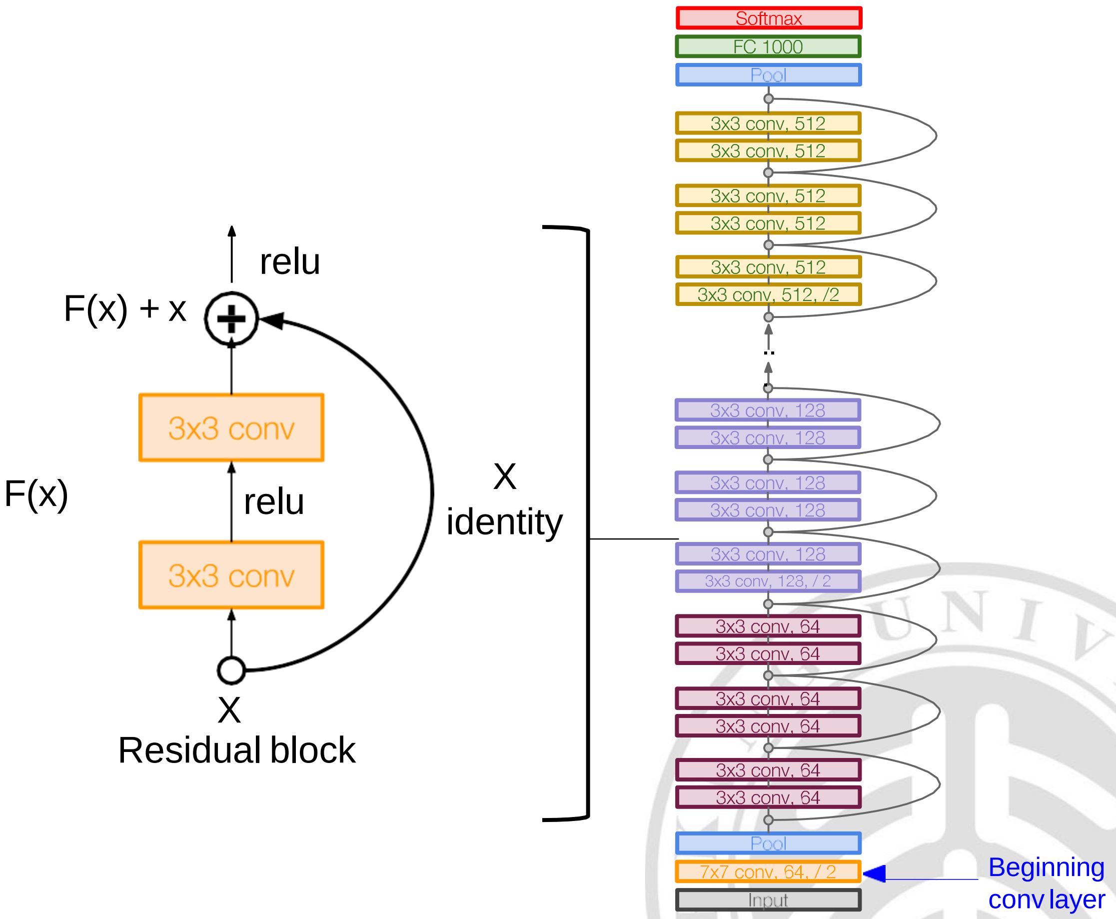


Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

ResNet [He et al., 2015]

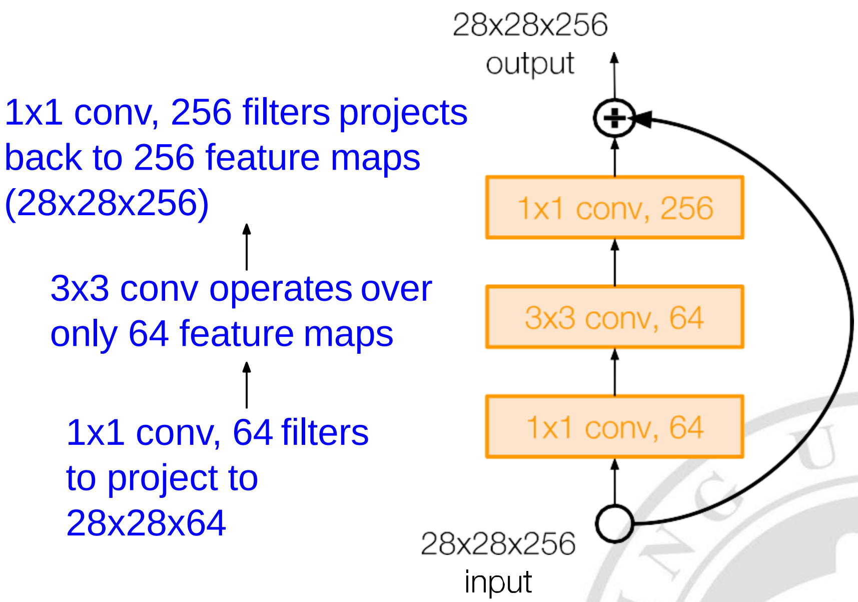
Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



ResNet [He et al., 2015]

For deeper networks (ResNet-50+),
use “bottleneck” layer to improve efficiency
(similar to GoogLeNet)



ResNet [He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier 2/ initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

ResNet [He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

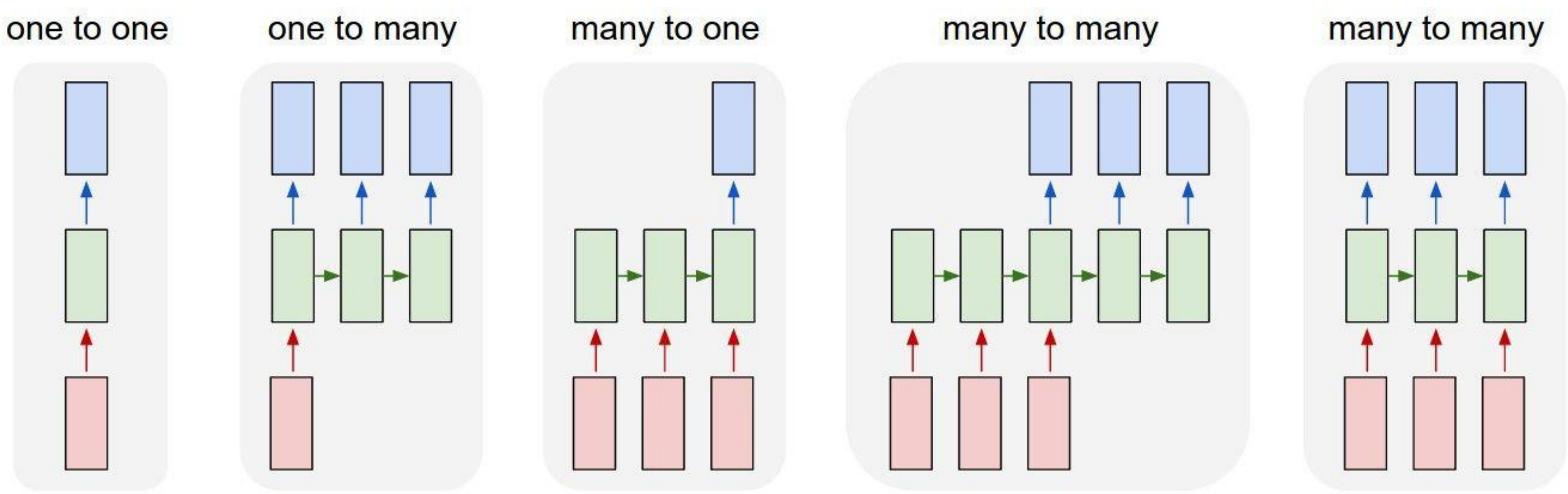
CNN Summary

- **VGG, GoogLeNet, ResNet** all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Efforts to investigate necessity of depth Vs. width and residual connections

Today

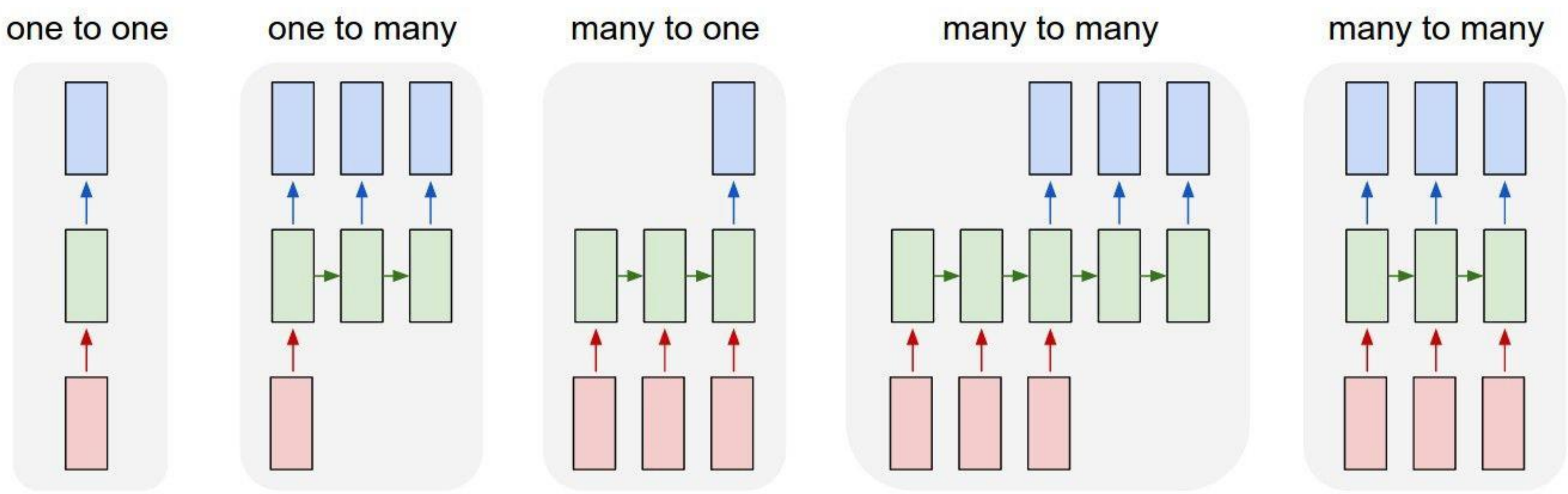
- CNN Architectures
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- **Recurrent Neural Network**
  - **Vanilla RNN**
  - **Backpropagation through time**
  - **Long Short-Term Memory**
- Beyond CNN and RNN
  - Unsupervised Learning
  - Generative Adversarial Network

Recurrent Neural Networks: Process Sequences



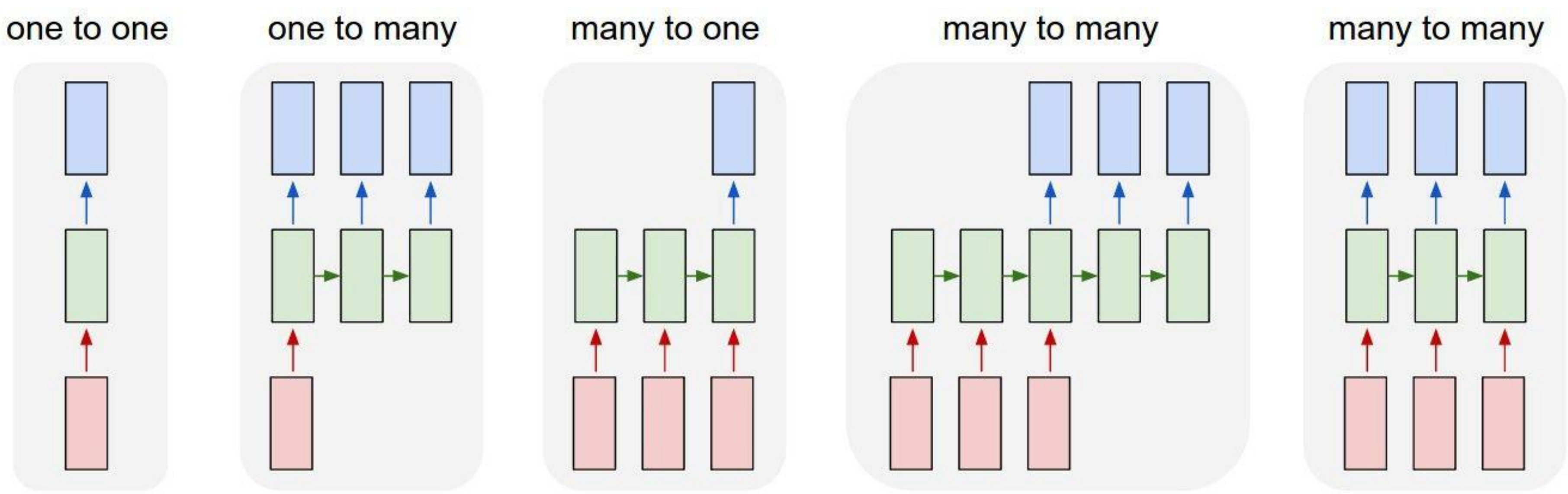
e.g. **Image Captioning**
image → sequence of words

Recurrent Neural Networks: Process Sequences



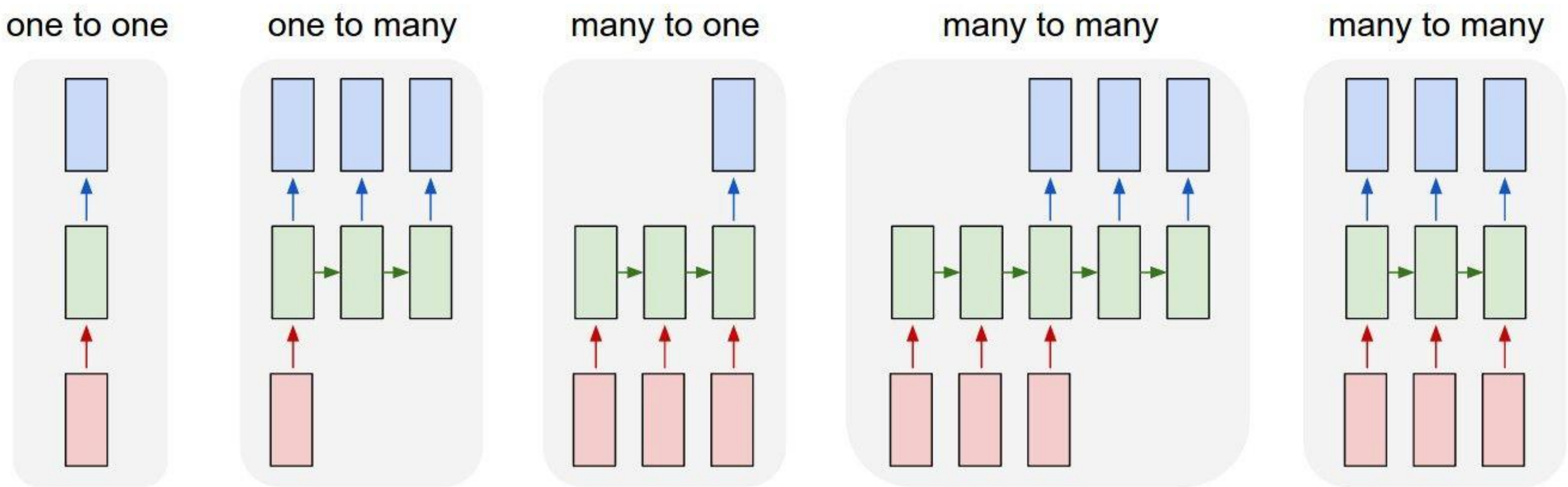
e.g. **Sentiment Classification**
sequence of words → sentiment

Recurrent Neural Networks: Process Sequences



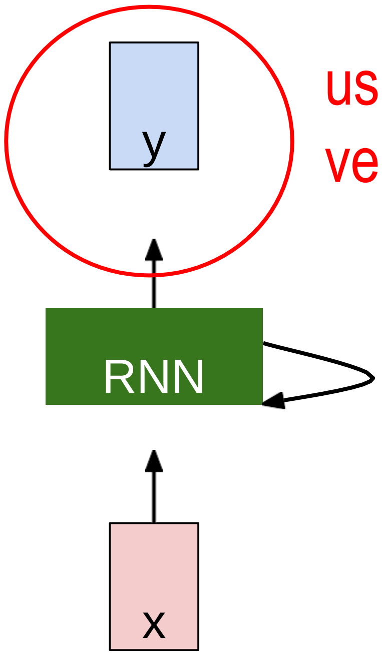
e.g. **Machine Translation**
seq of words → seq of words

Recurrent Neural Networks: Process Sequences



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

Recurrent Neural Network



usually want to predict a vector at some time steps

Recurrent Neural Network

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

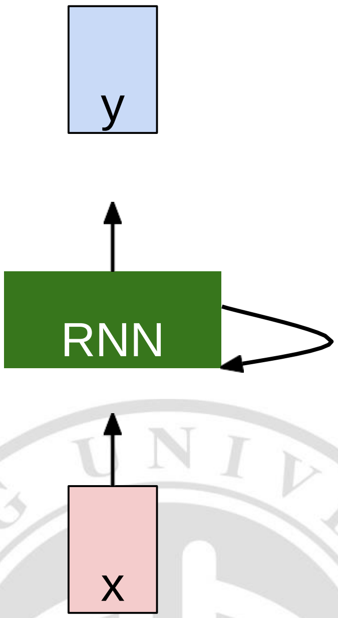
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

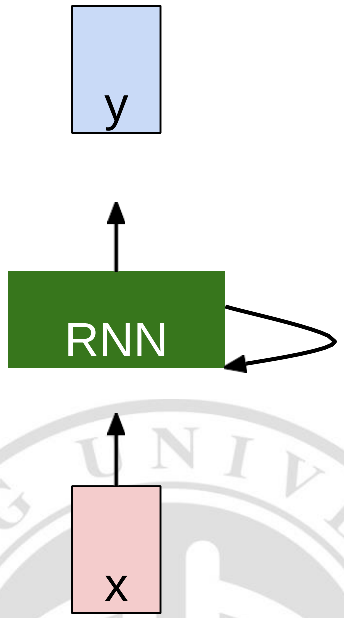
some function with parameters W



Recurrent Neural Network

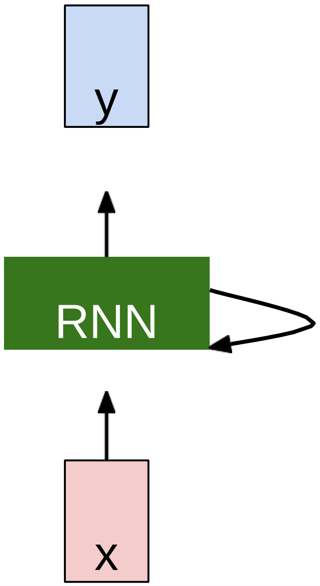
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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



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

(Simple) Recurrent Neural Network



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

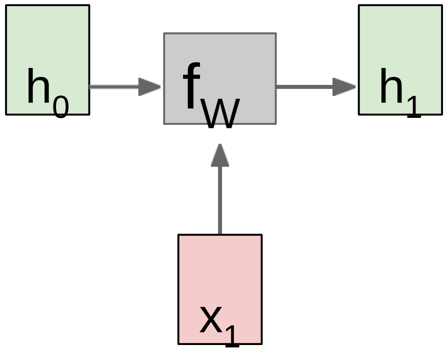
$$h_t = f_W(h_{t-1}, x_t)$$

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

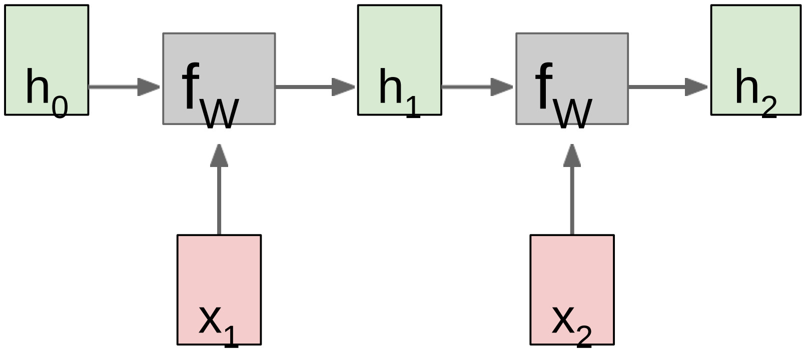
$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

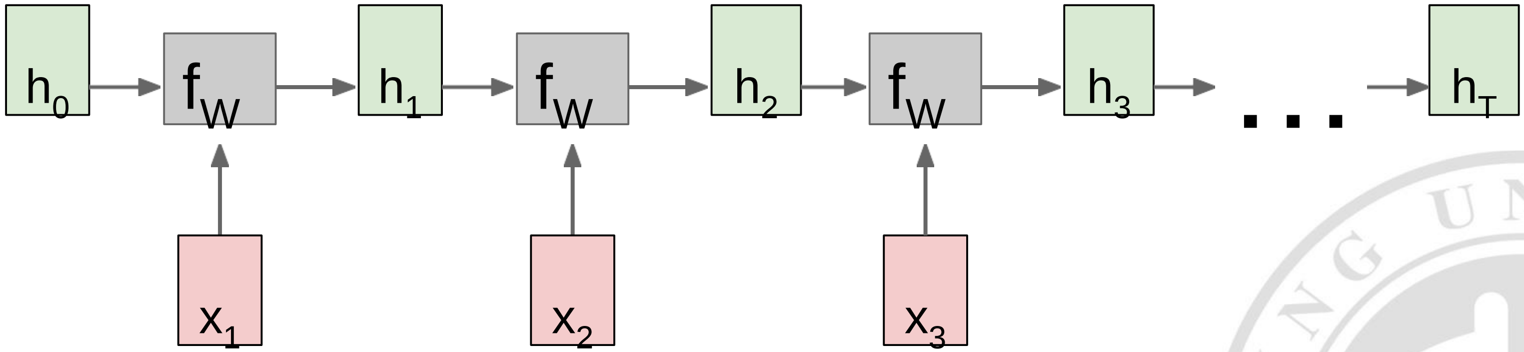
RNN: Computational Graph



RNN: Computational Graph

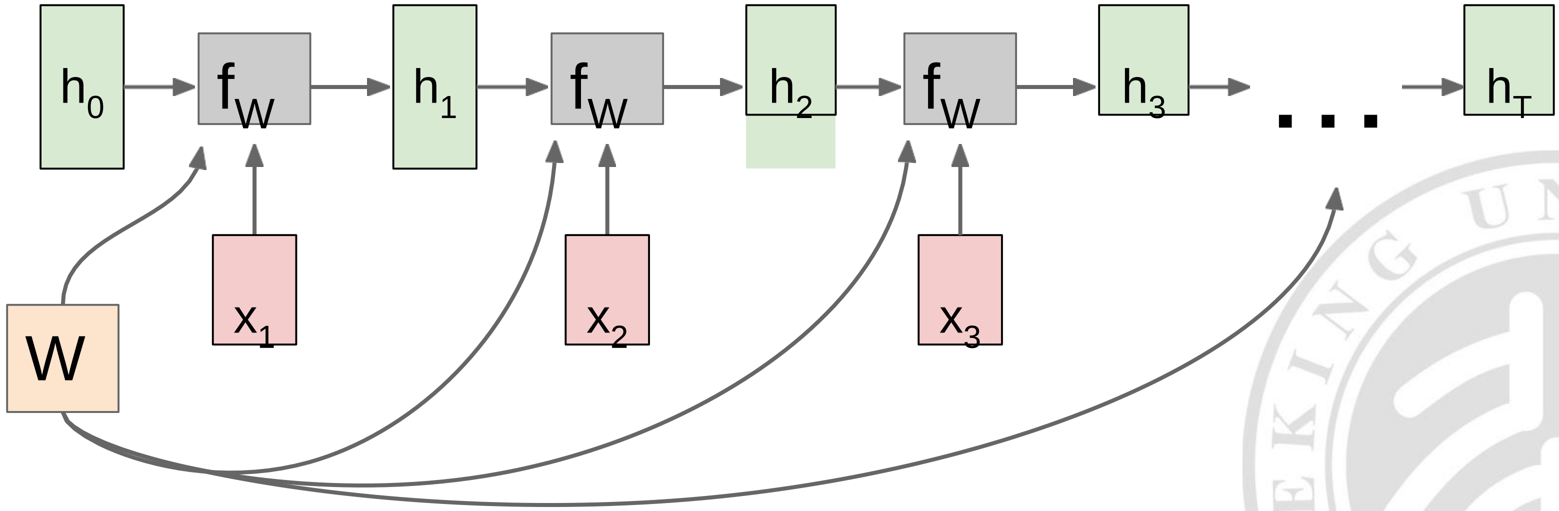


RNN: Computational Graph

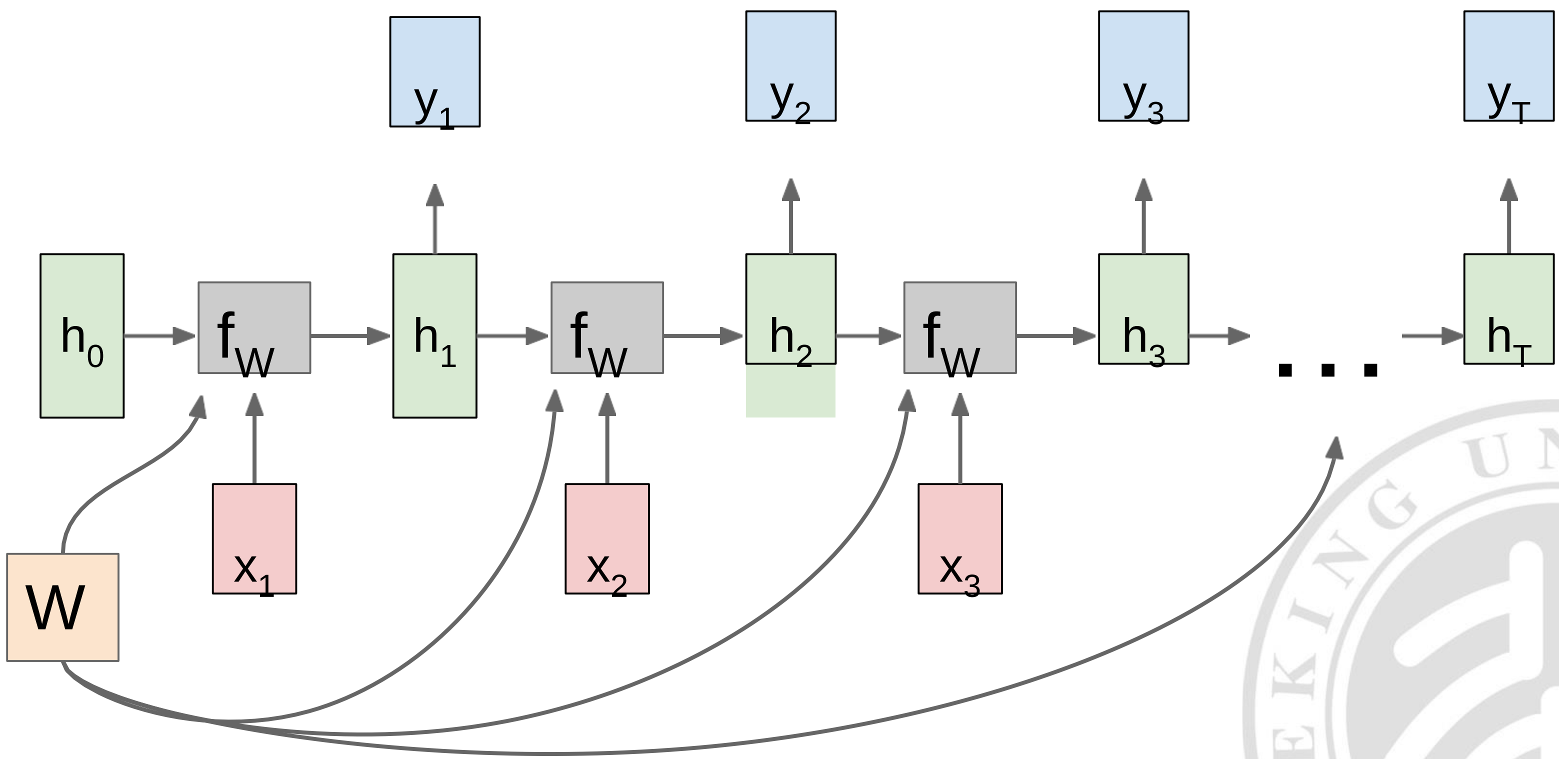


RNN: Computational Graph

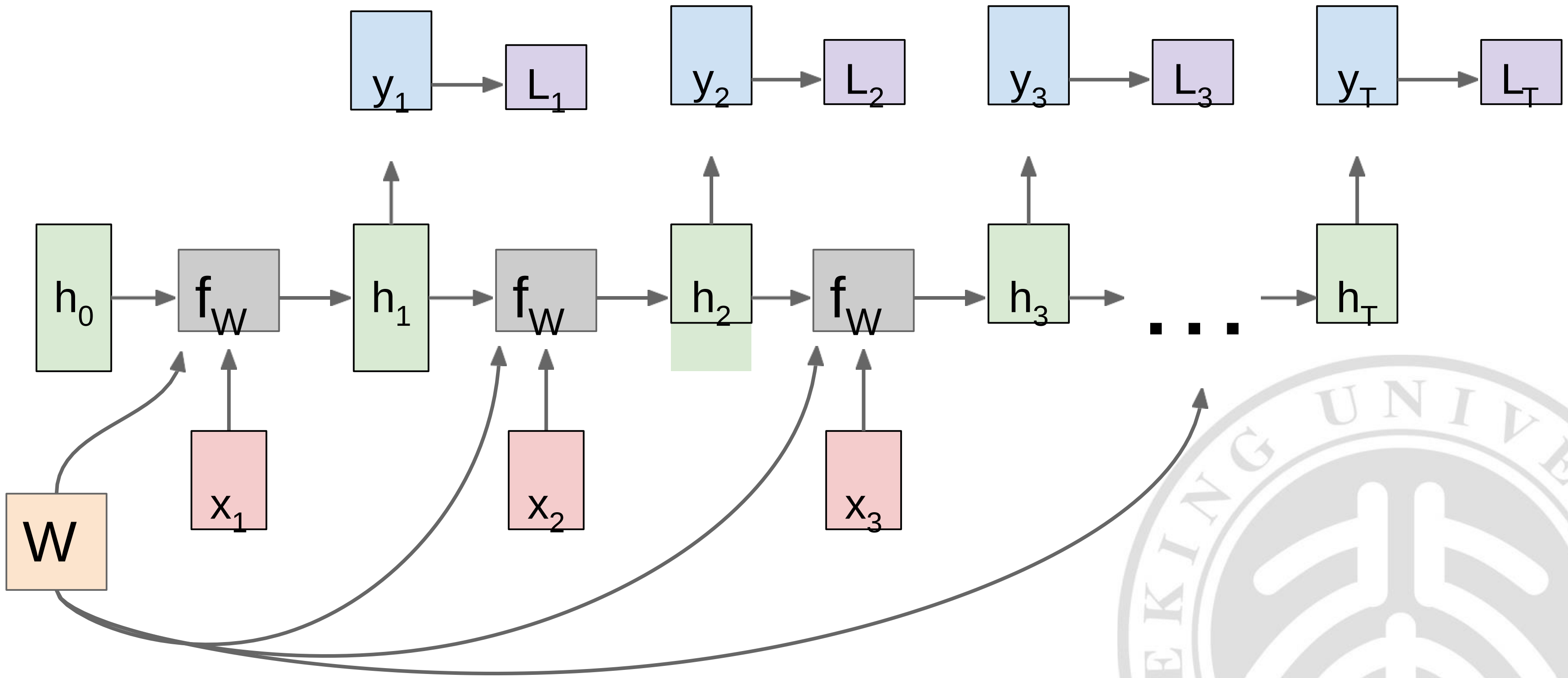
Re-use the same weight matrix at every time-step



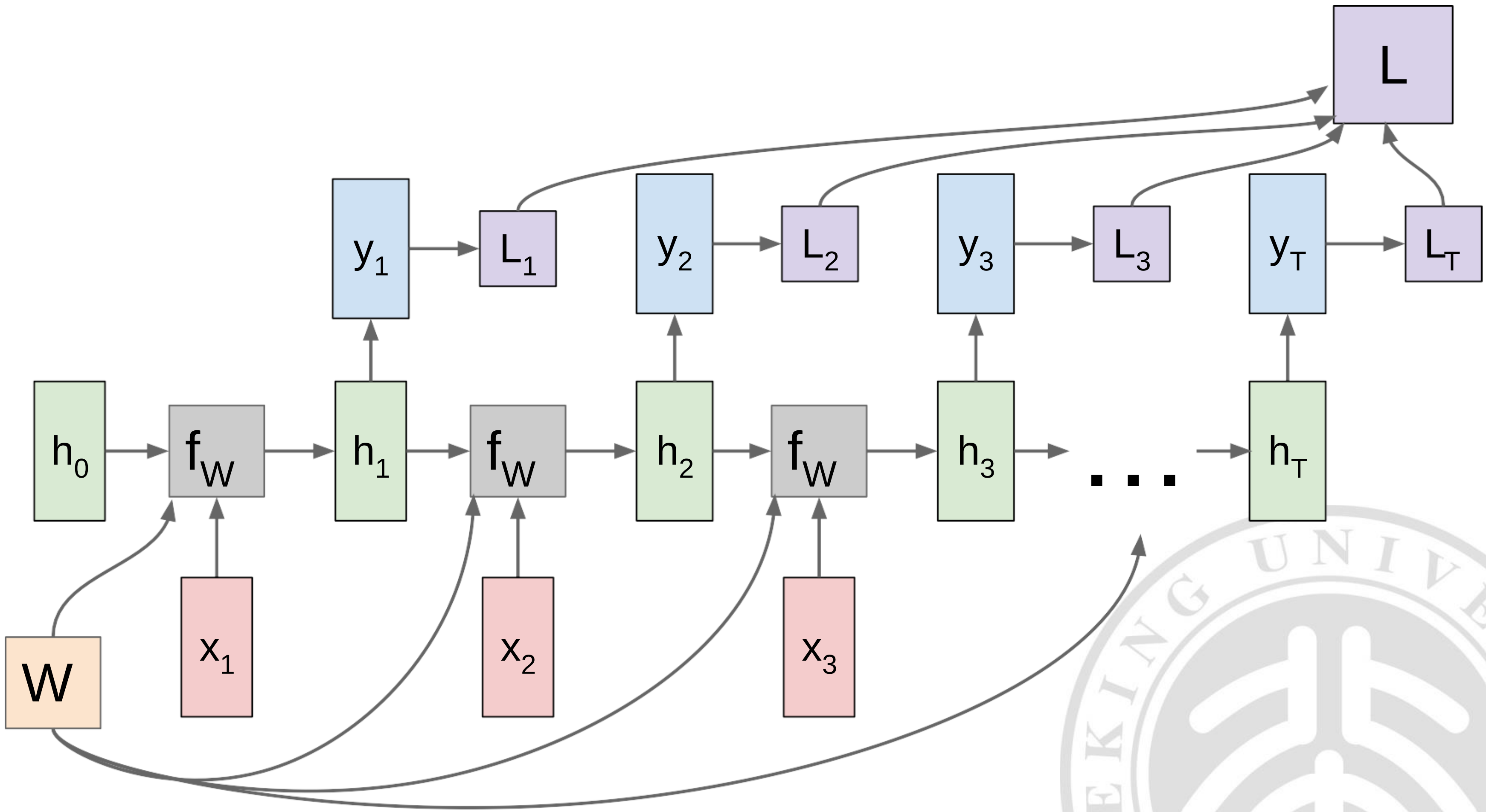
Many to Many



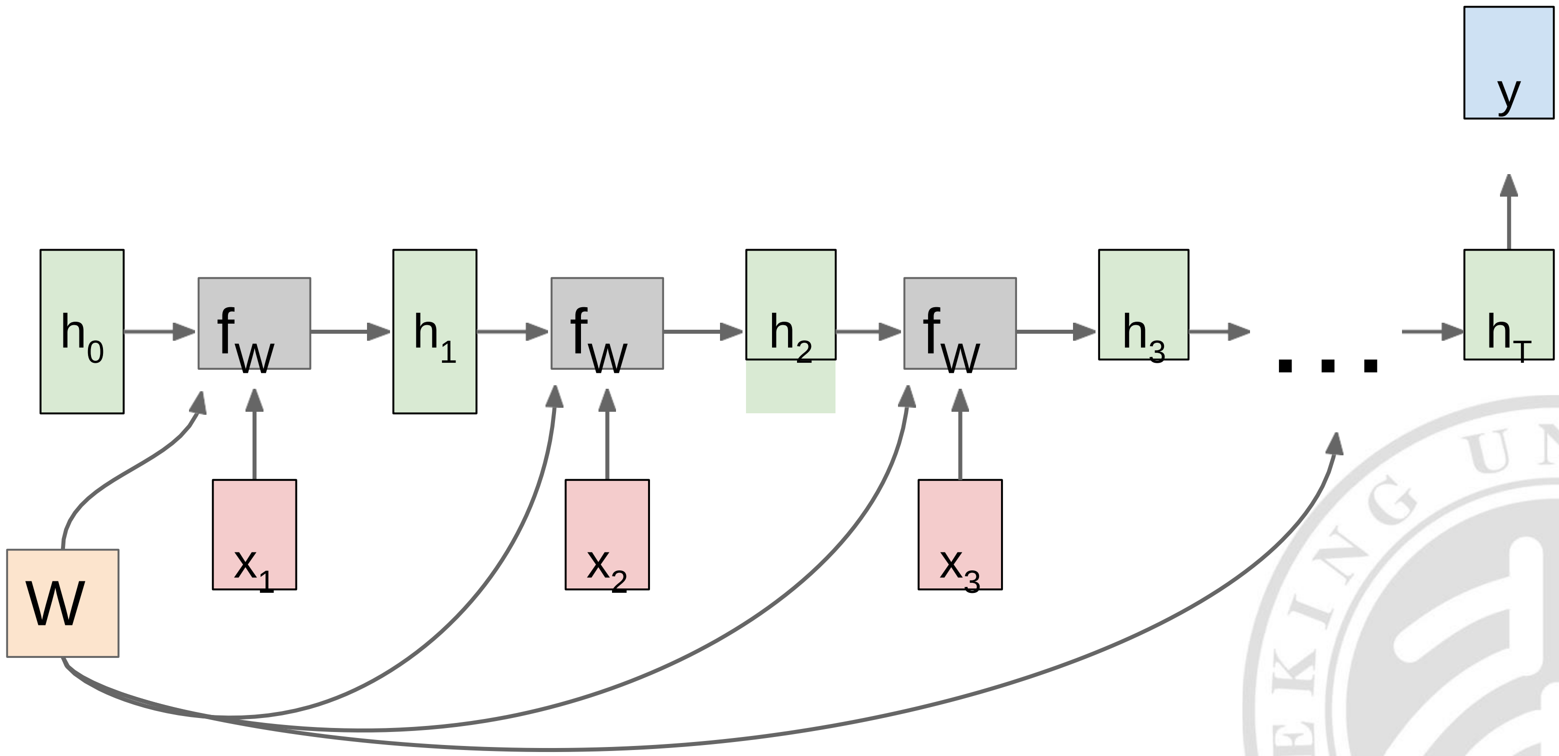
Many to Many



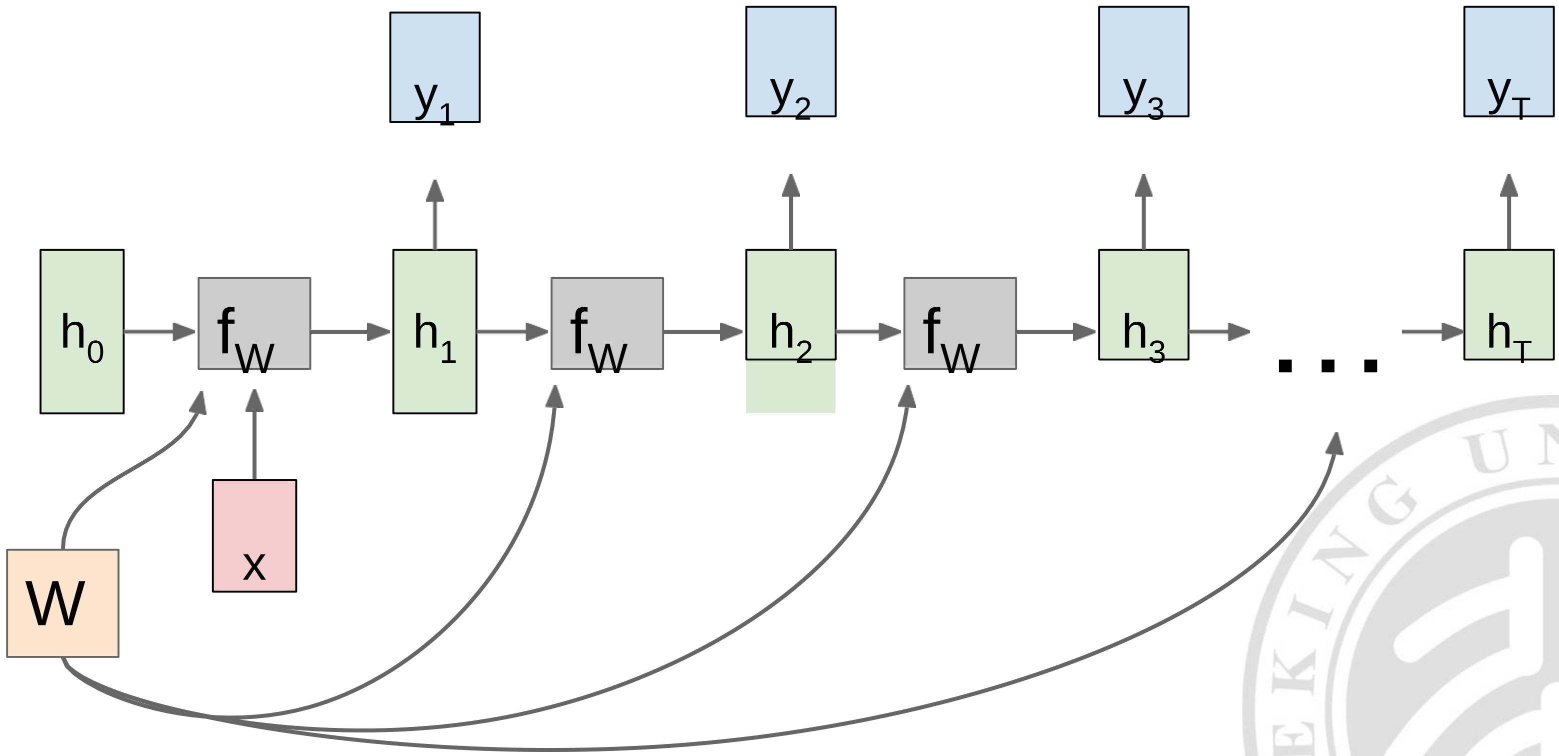
Many to Many



Many to One

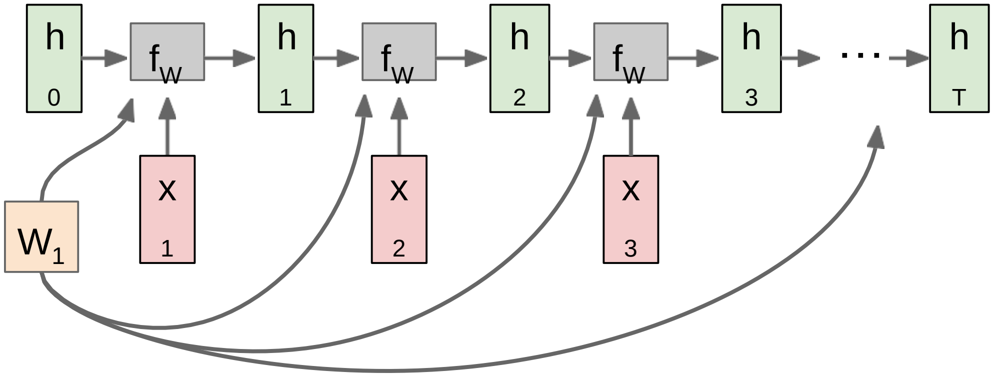


One to Many



Sequence to Sequence: Many-to-one + One-to-many

Many to one: Encode input sequence in a single vector

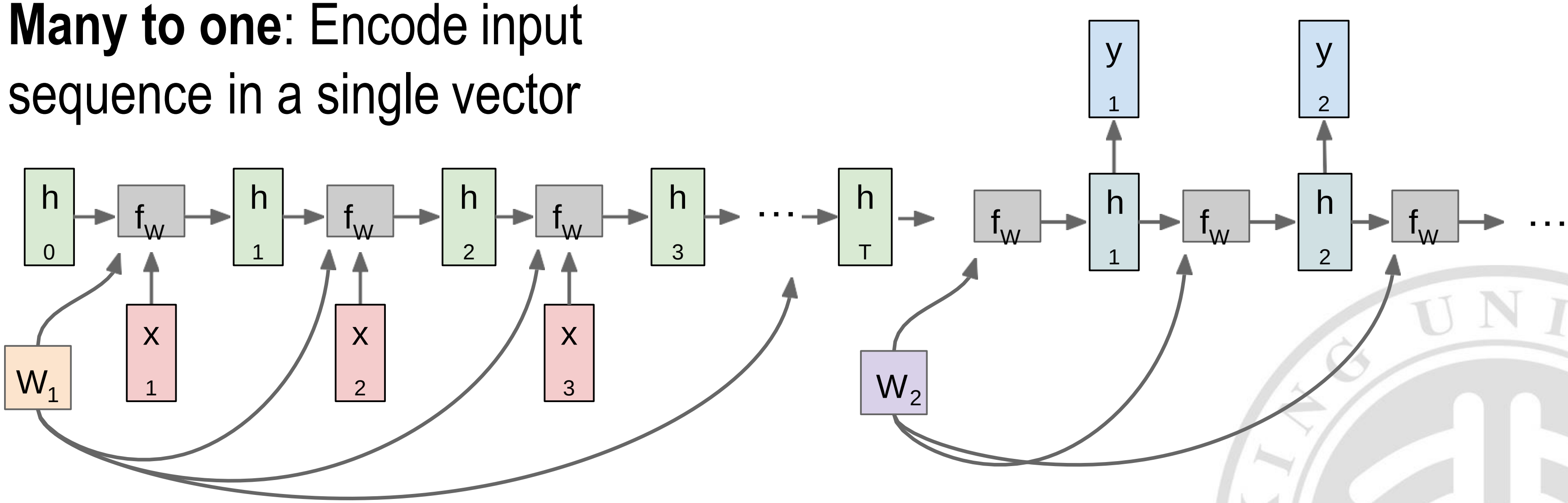


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence to Sequence: Many-to-one + One-to-many

One to many: Produce output sequence from single input vector

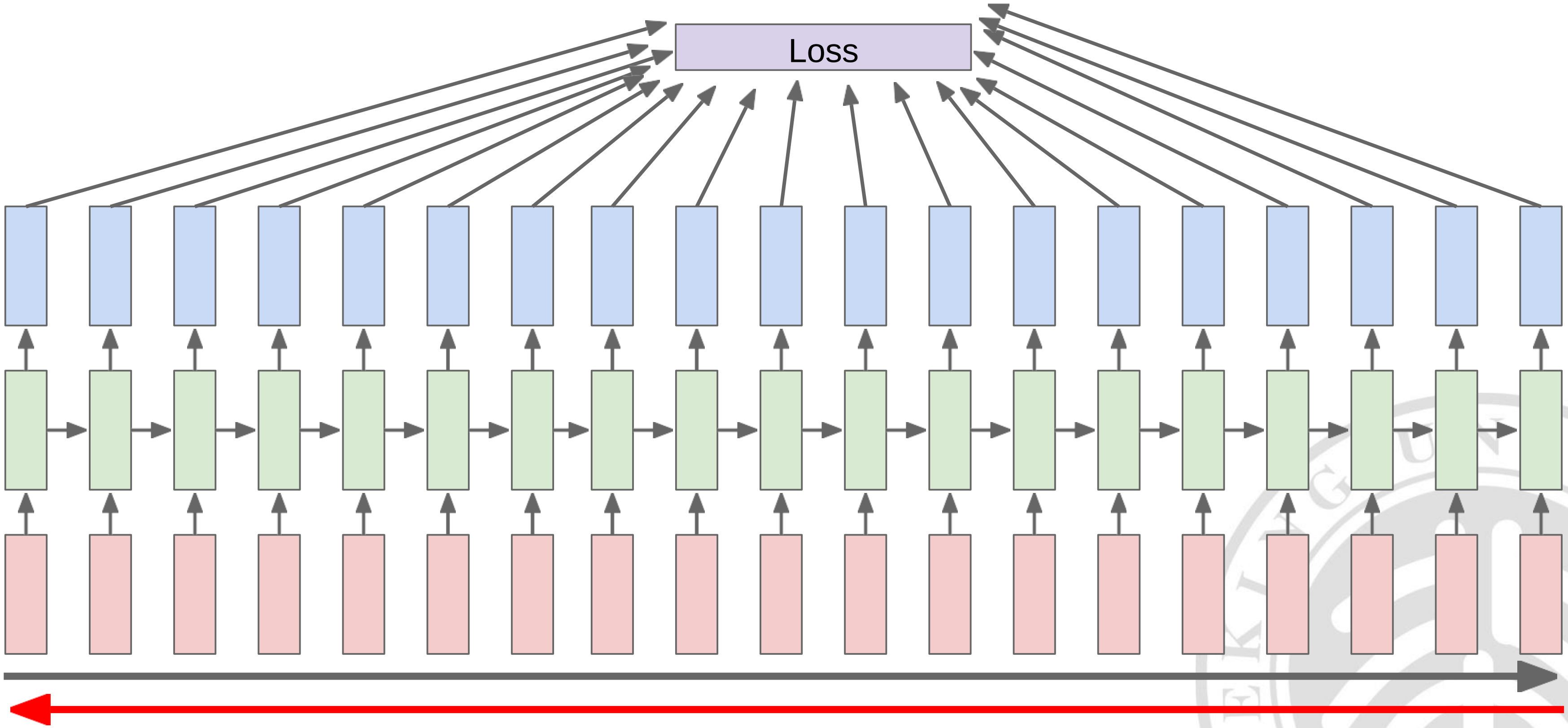
Many to one: Encode input sequence in a single vector



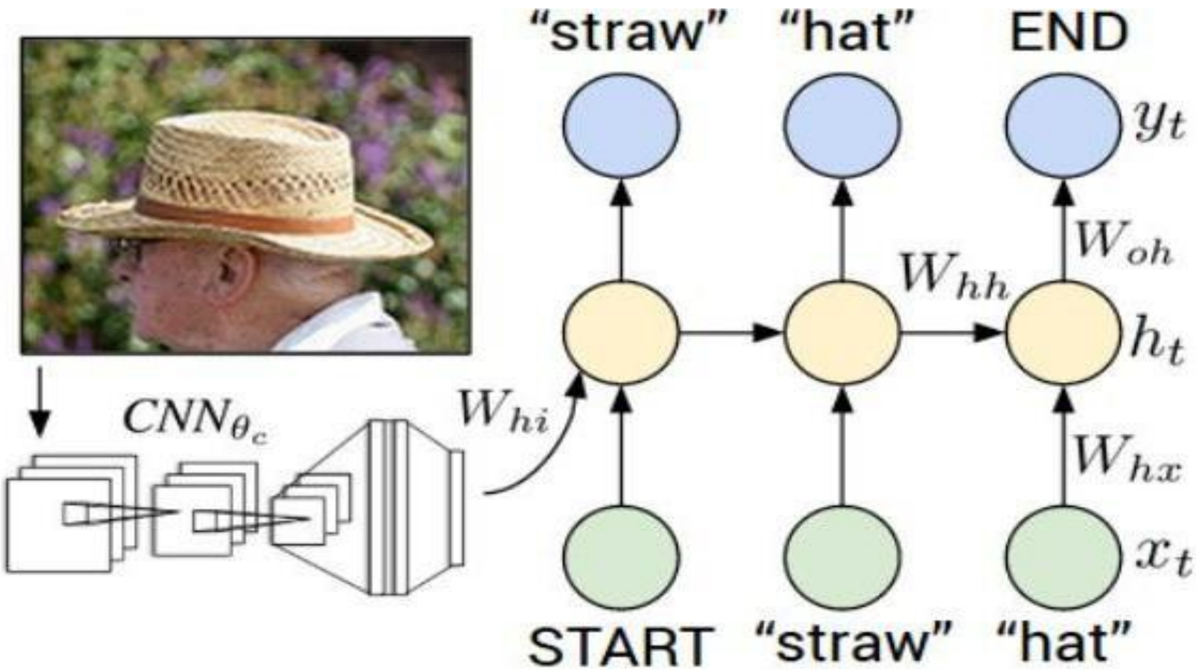
Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Backpropagation Through Time

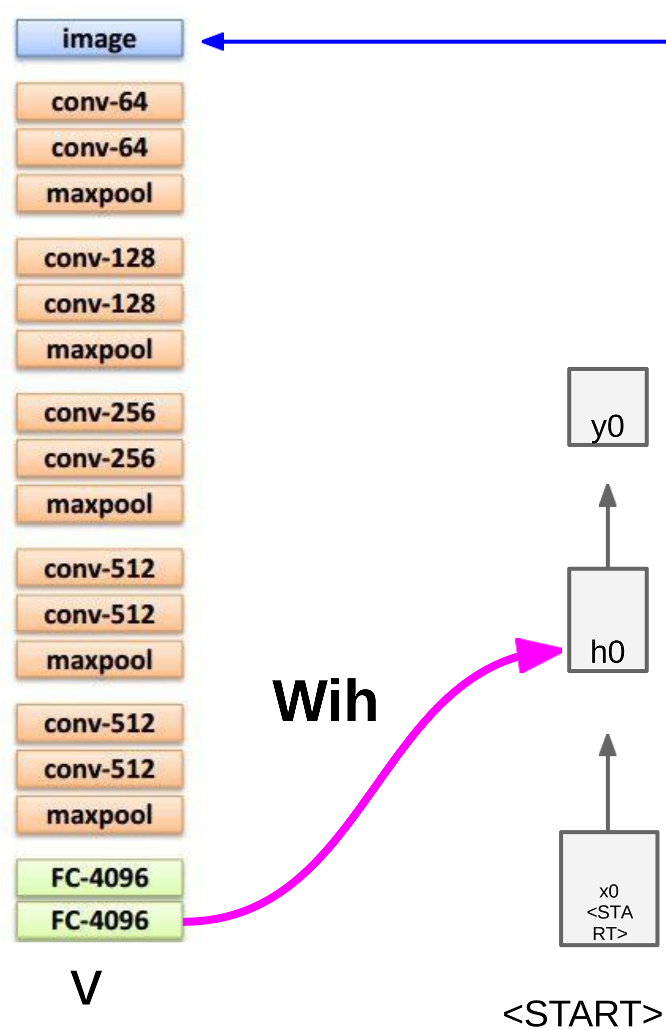
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Example: Image Captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick



test image

before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h **+ Wih * v**)

test image

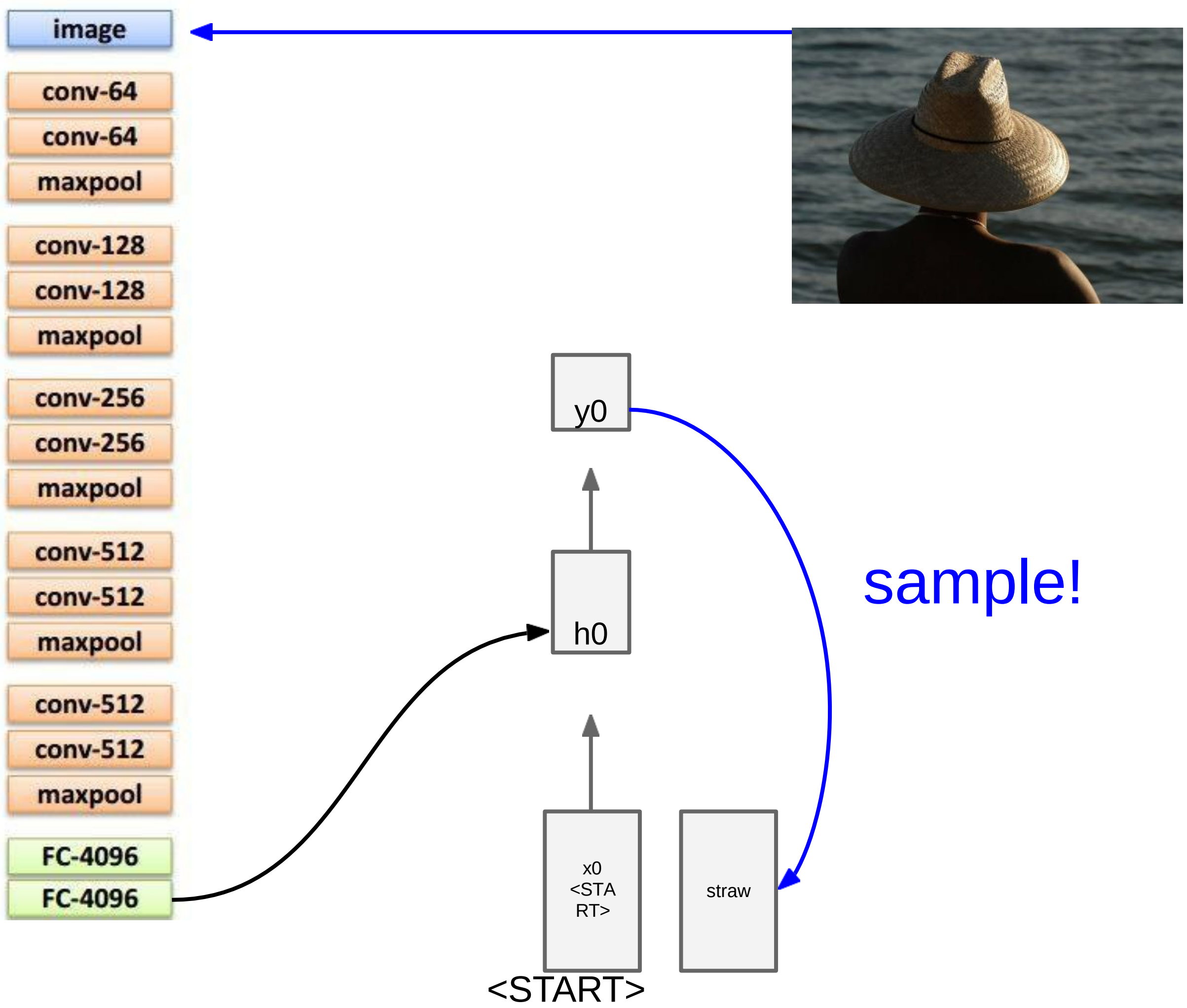


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
h0
x0
<START>
straw
<START>
sample!

test image

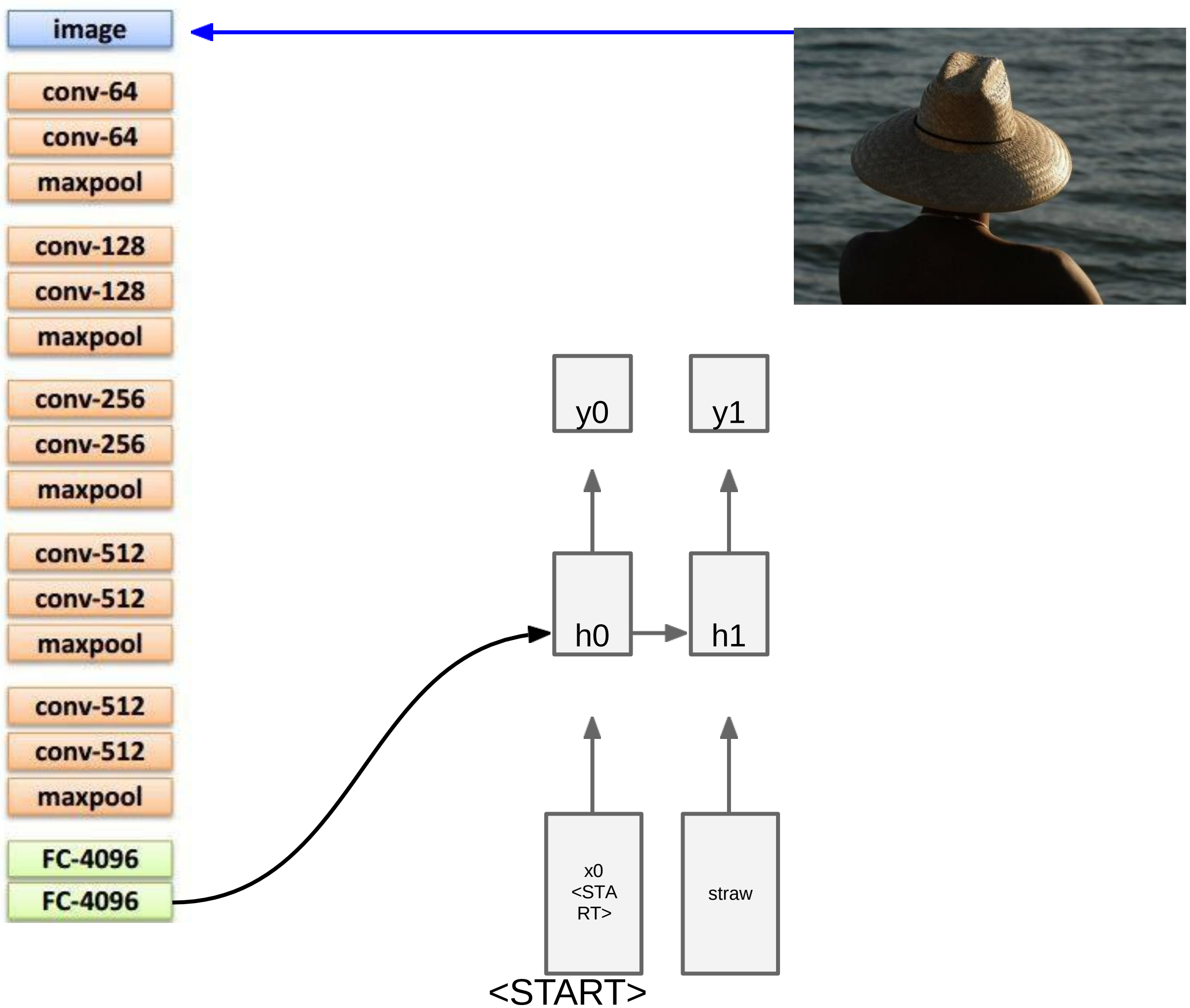


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0 <START>
straw
<START>

test image

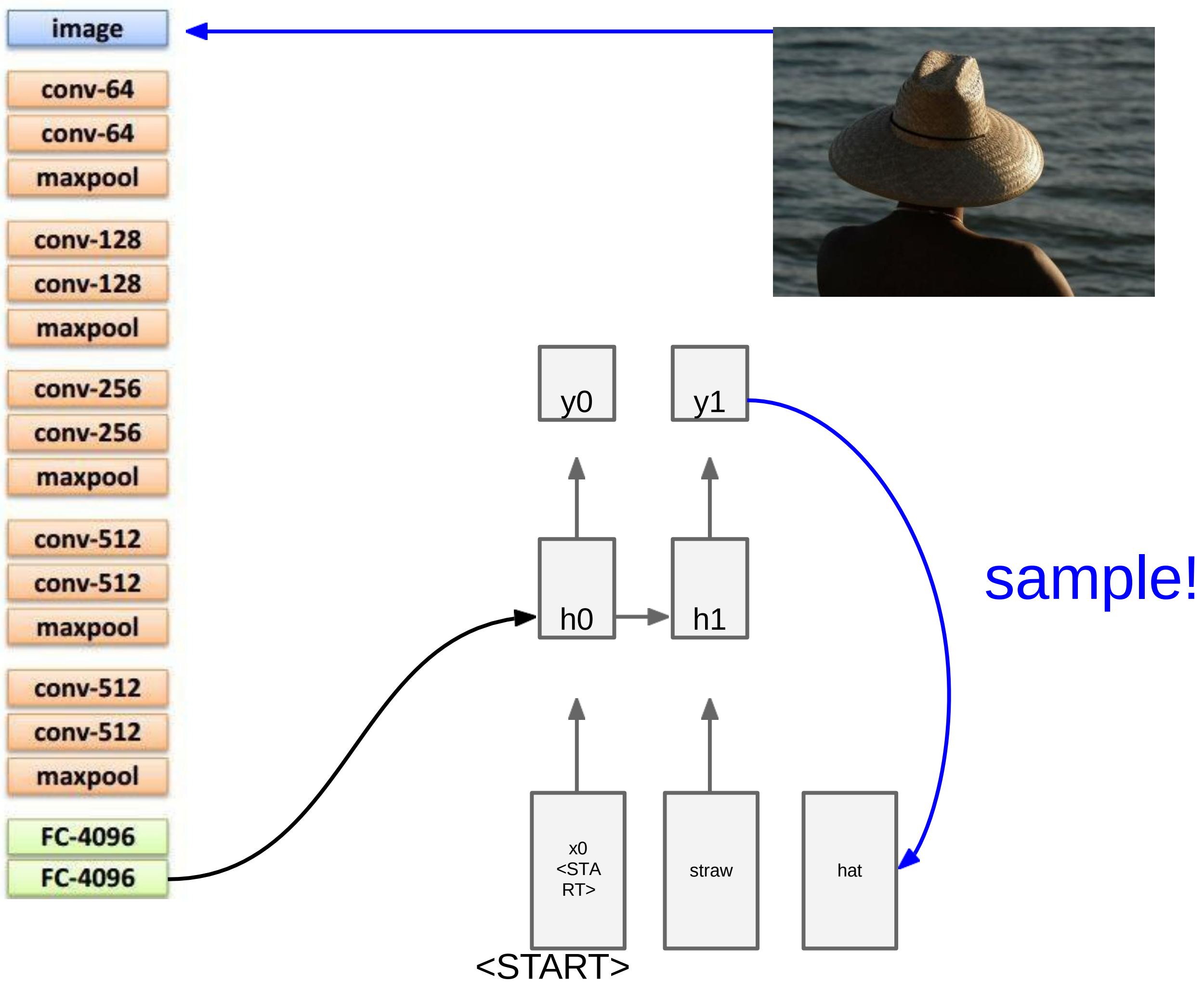


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0 <STA RT>
straw
hat
<START>
sample!

test image

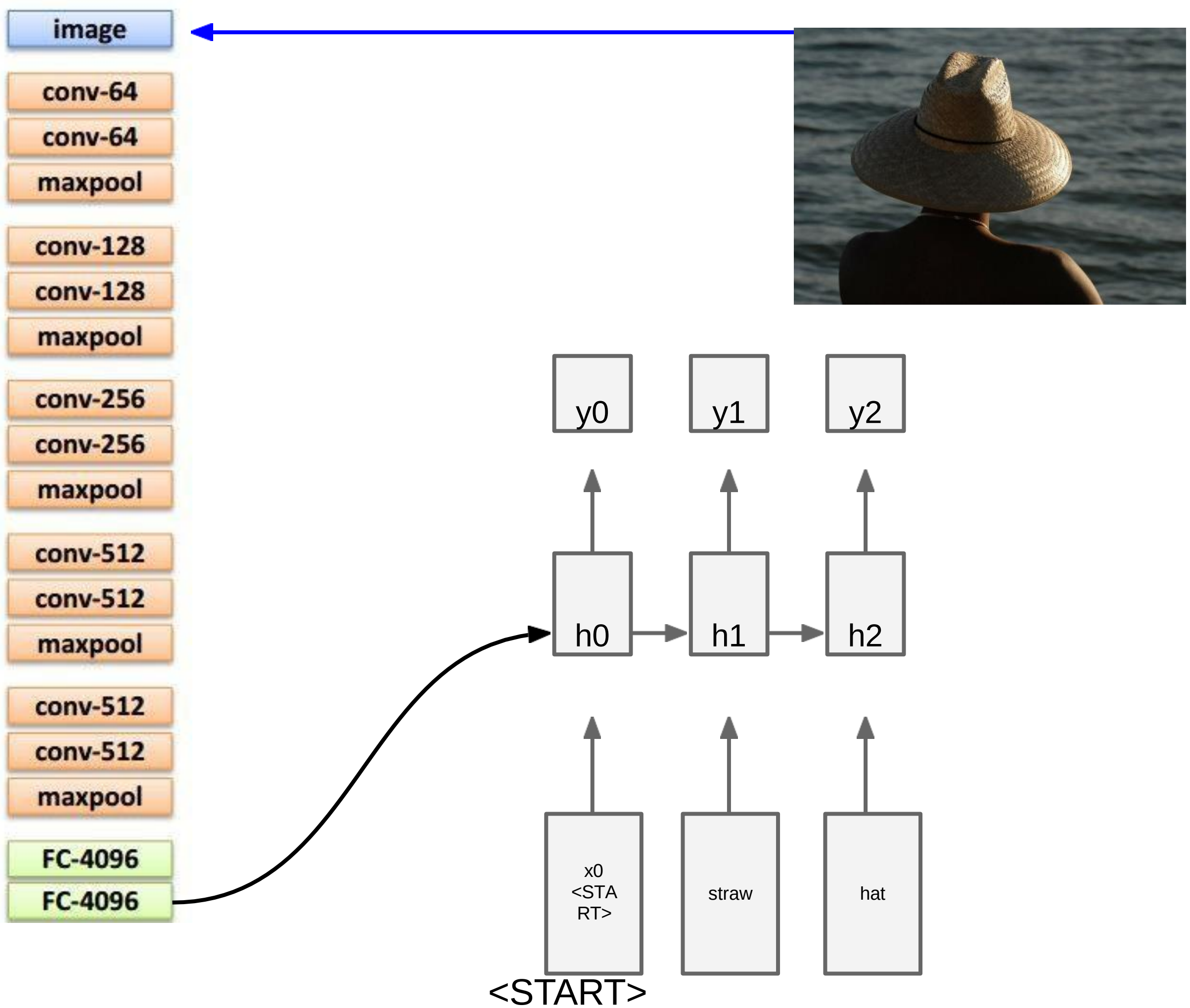
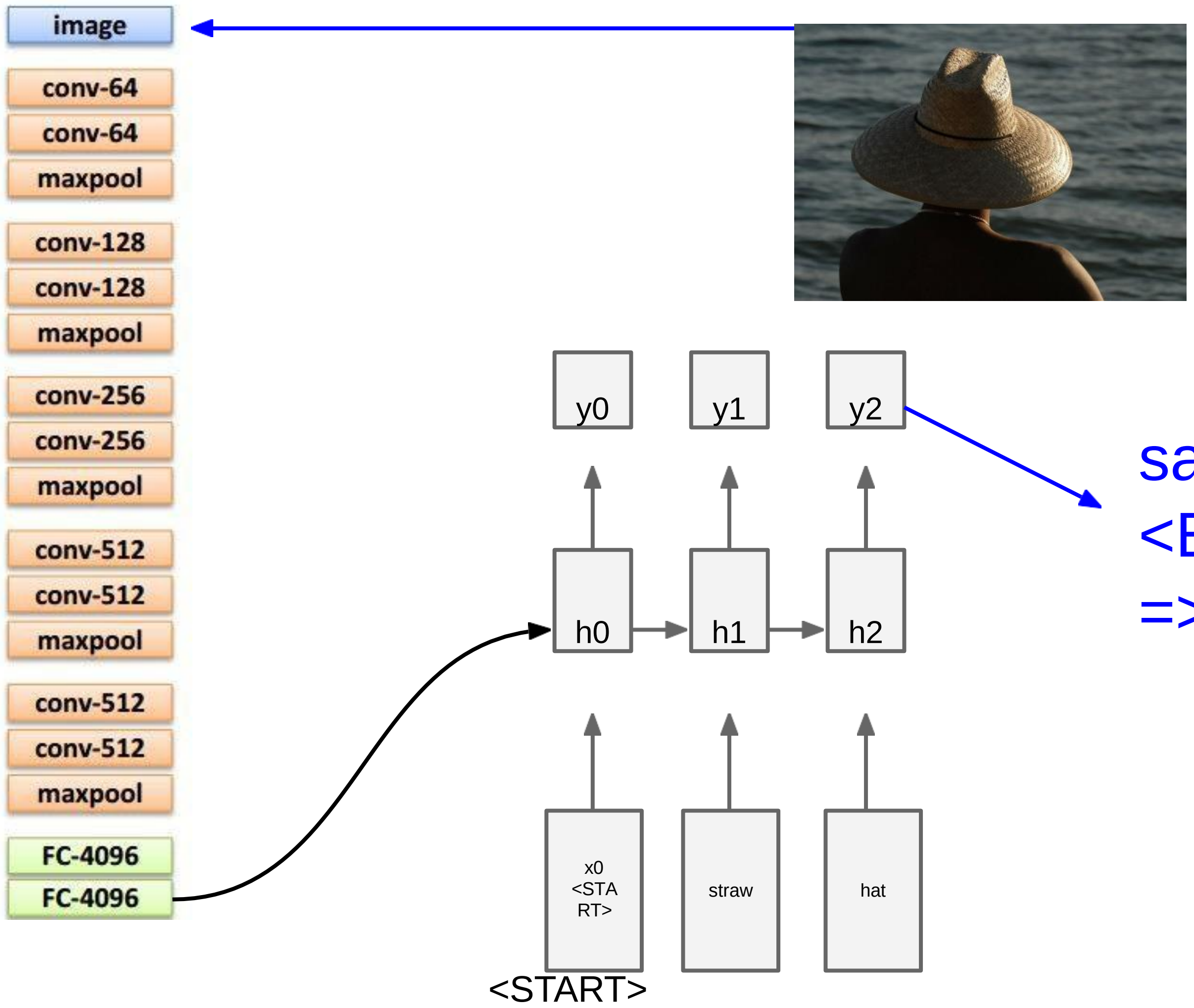


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
y2
h0
h1
h2
x0 <START>
straw
hat
<START>

test image



sample
<END> token
=> finish.

Image Captioning: Example Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

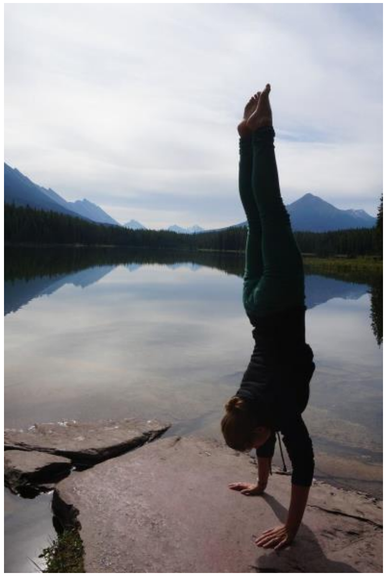
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Multilayer RNNs

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

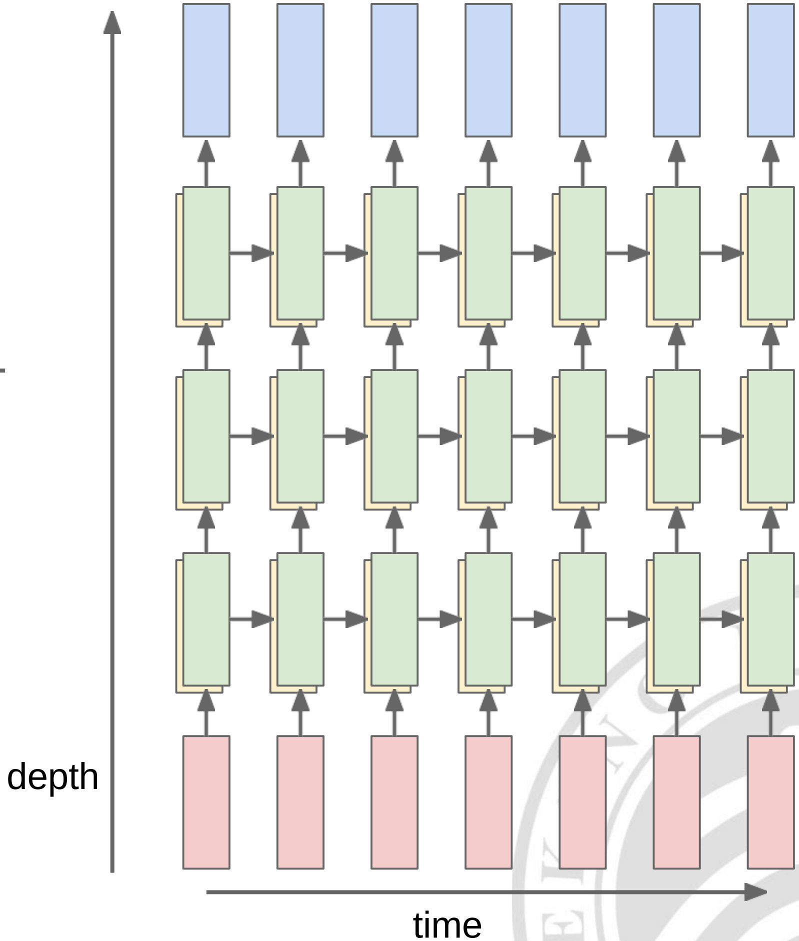
LSTM:

$W^l \ [4n \times 2n]$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

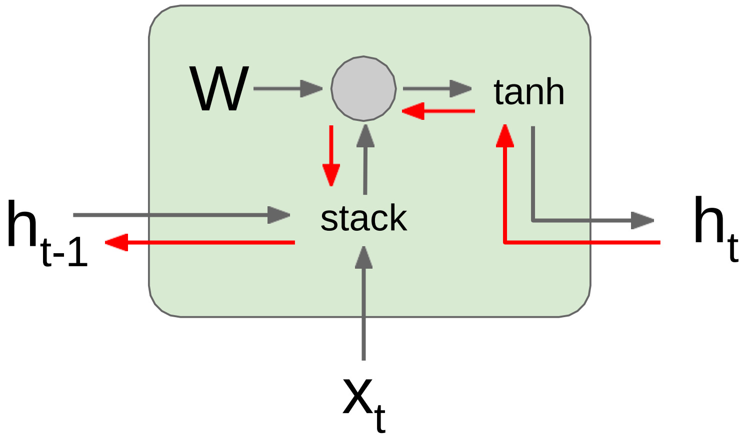
$$h_t^l = o \odot \tanh(c_t^l)$$



Vanilla RNN Gradient Flow

Backpropagation from h_t to h_{t-1} multiplies by W

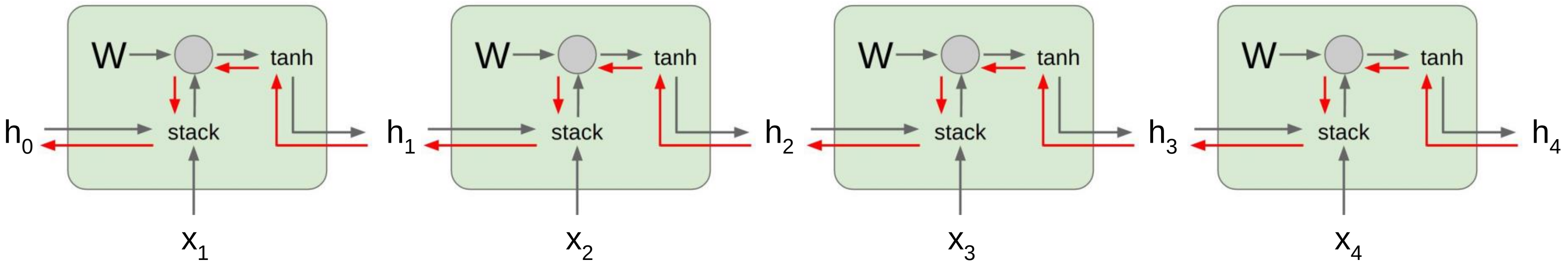
(actually W_{hh}^T)



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

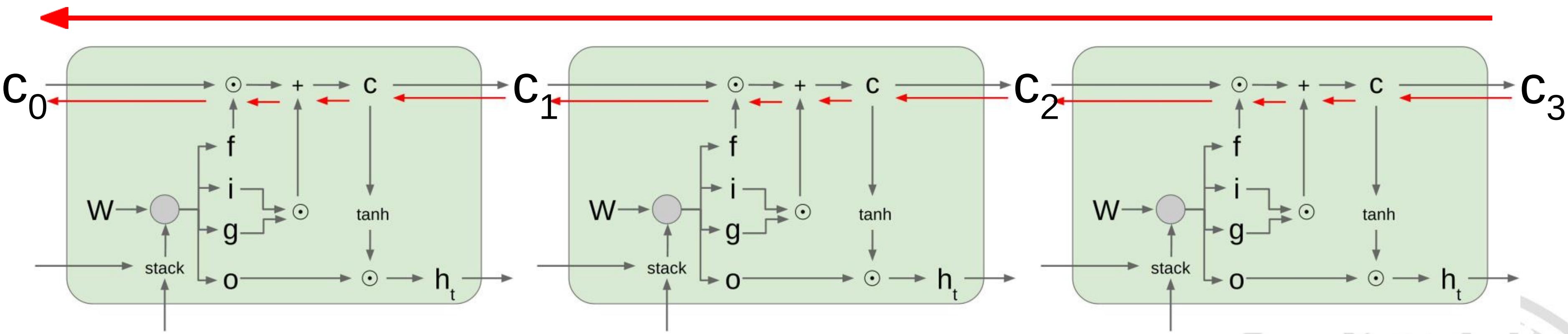
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

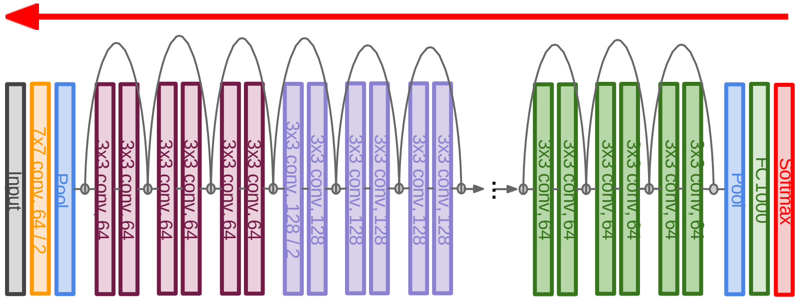
Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet !



[Hochreiter et al., 1997]

Today

- CNN Architectures
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- Recurrent Neural Network
  - Vanilla RNN
  - Backpropagation through time
  - Long Short-Term Memory
- **Beyond CNN and RNN**
  - **Unsupervised Learning**
  - **Generative Adversarial Network**

Supervised Vs. Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

→ Cat

Classification

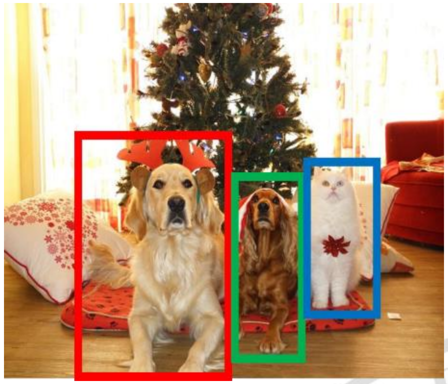
Supervised Vs. Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.



DOG, DOG, CAT

Object Detection

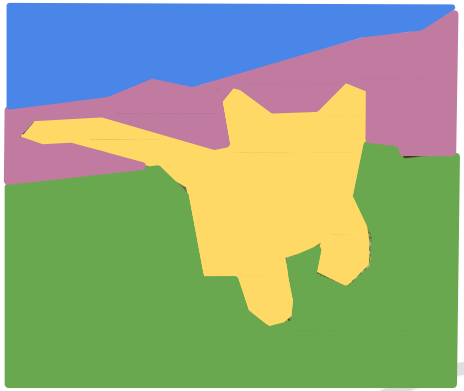
Supervised Vs. Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.



Semantic Segmentation

Supervised Vs. Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

A cat sitting on a suitcase on the floor

Image Captioning

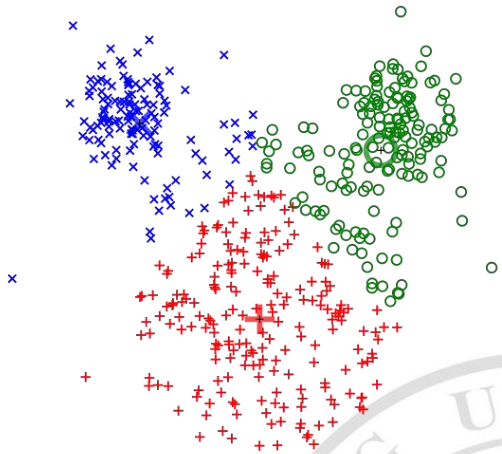
Supervised Vs. Unsupervised Learning

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



K-means clustering

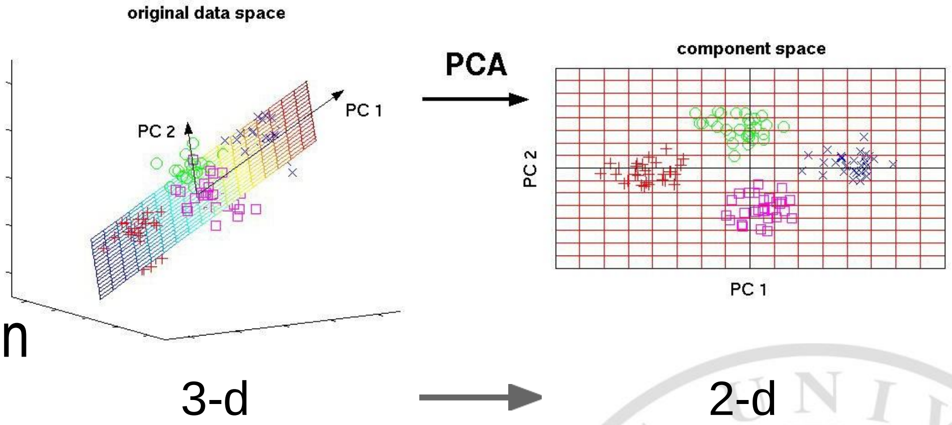
Supervised Vs. Unsupervised Learning

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)

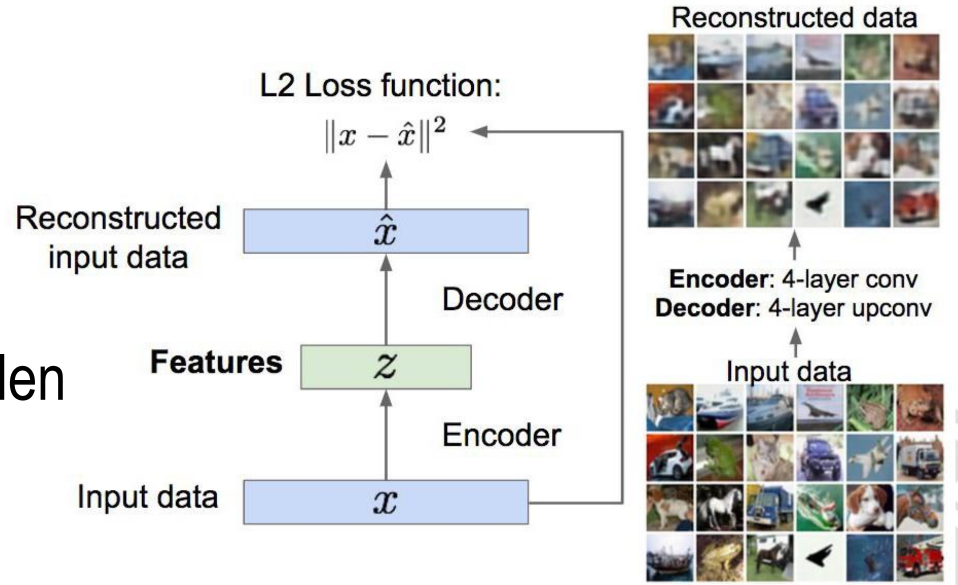
Supervised Vs. Unsupervised Learning

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

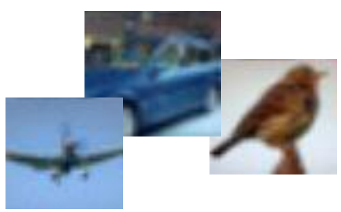
Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Autoencoders
(Feature learning)

Generative Models

- Given training data, generate new samples from same distribution
- **Several flavors:**
  - Explicit density estimation: explicitly define and solve for $p_{model}(x)$
  - Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it



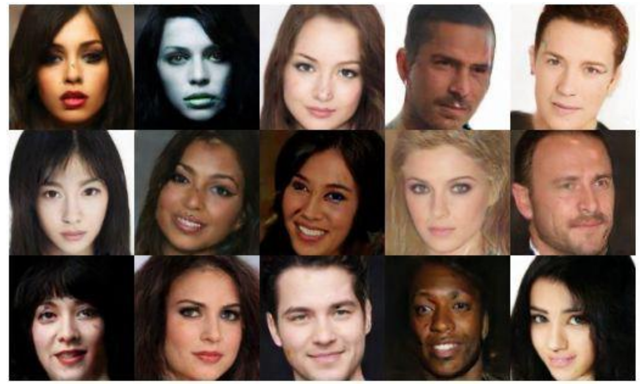
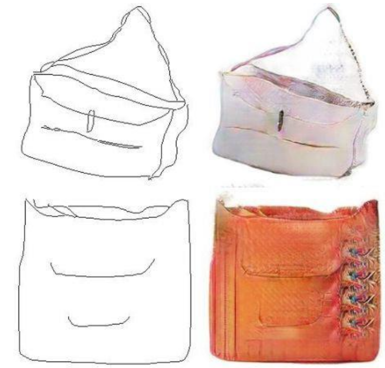
Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Why Generative Models

- Realistic samples for artwork, super-resolution, colorization, etc.

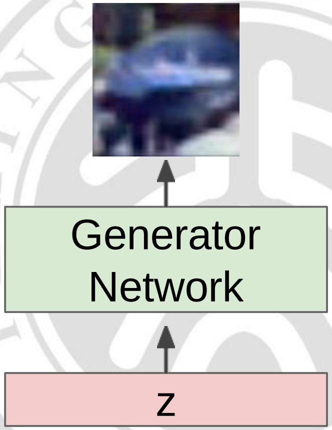




- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

Generative Adversarial Networks

- Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.
- Q: What can we use to represent this complex transformation?
- A: A neural network!

Output: Sample from training distribution



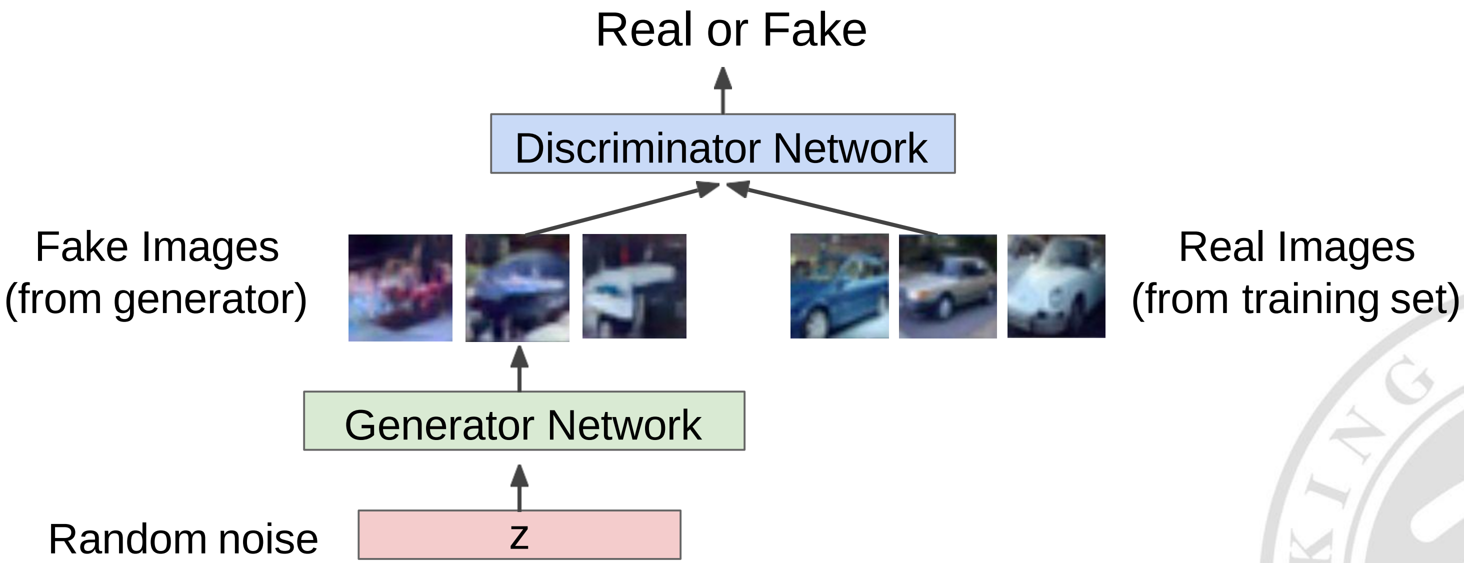
Input: Random noise

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images



Training GANs: Two-player game

Generator network: try to fool the discriminator by generating real-looking images

Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

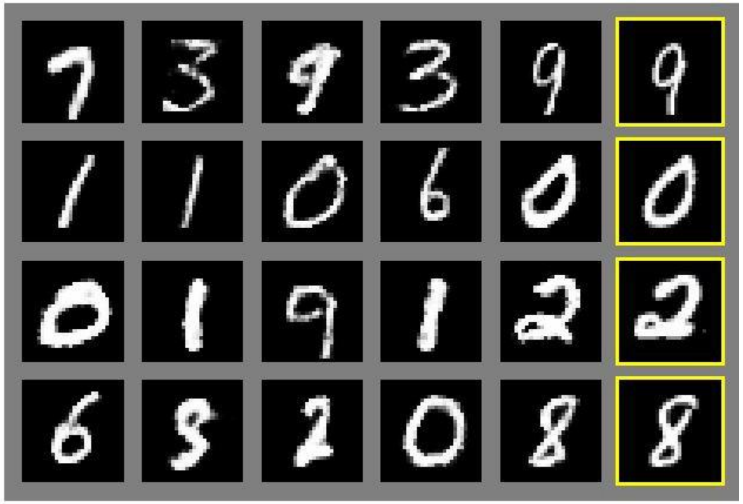
Minimax objective function:

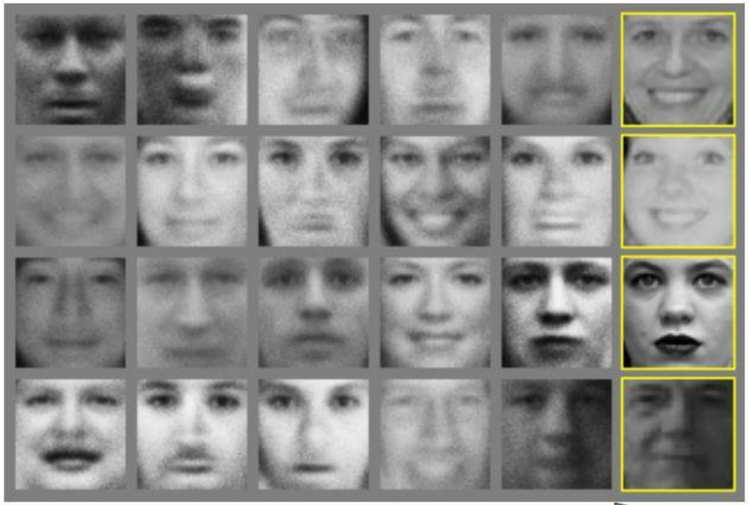
$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Generative Adversarial Nets

Generated samples

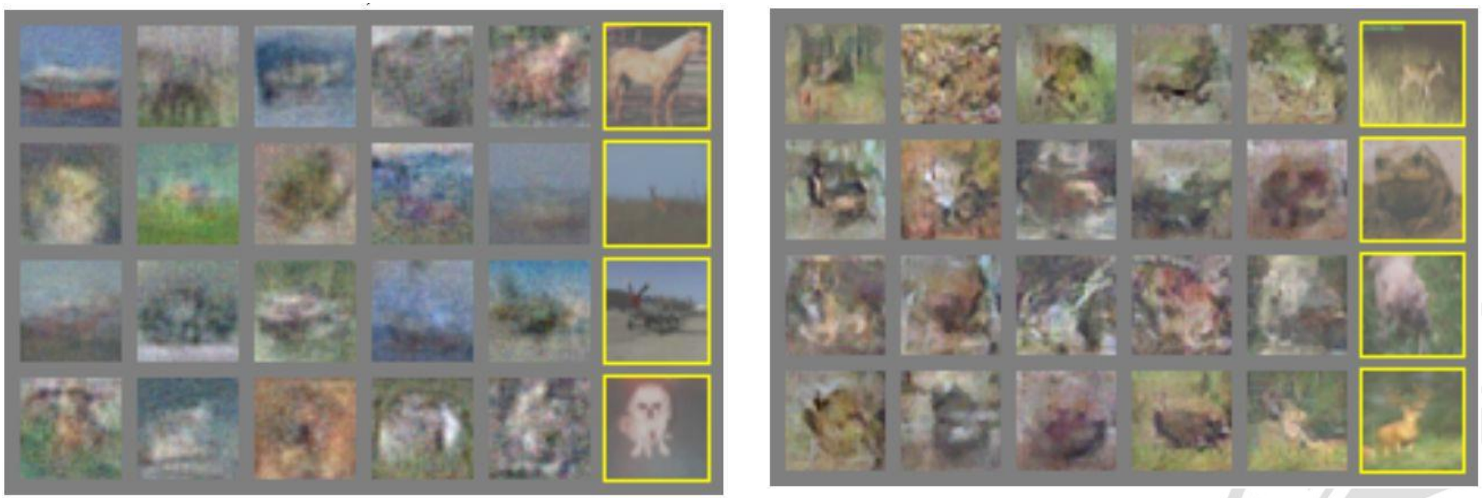




Nearest neighbor from training set

Generative Adversarial Nets

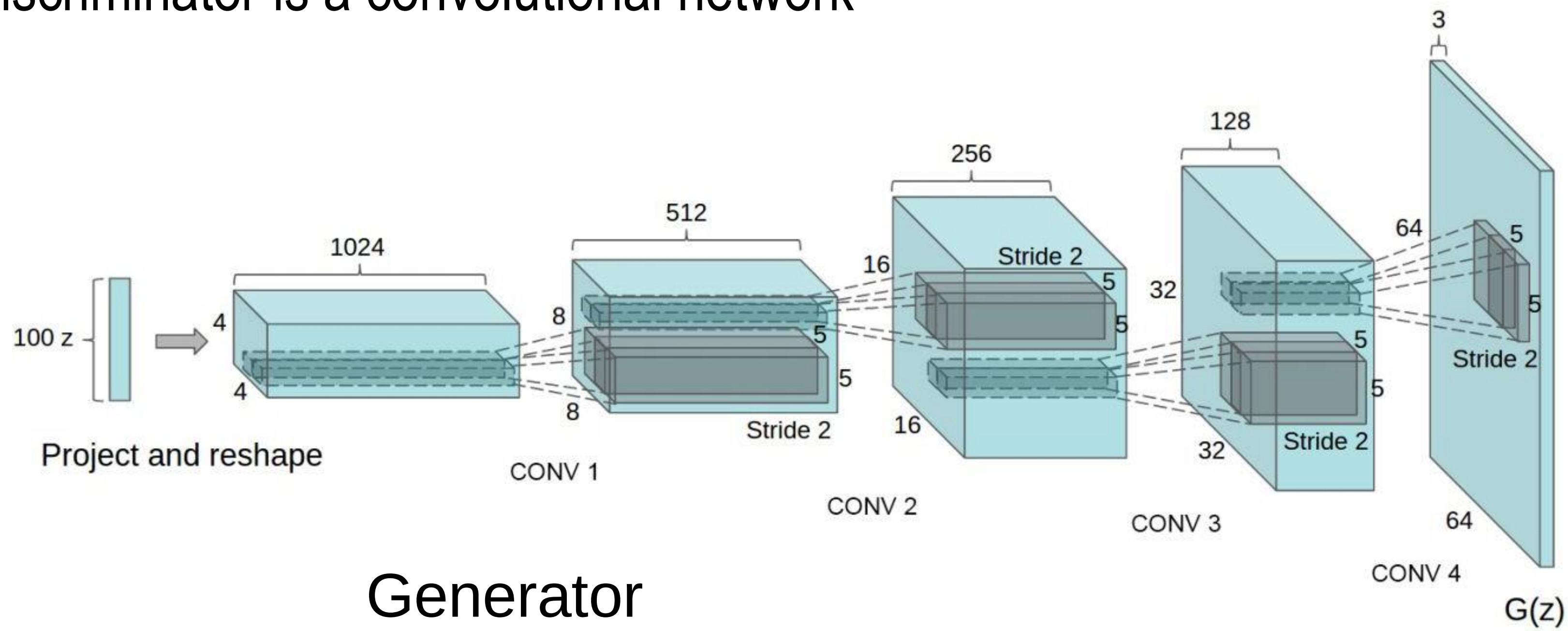
Generated samples (CIFAR-10)



Nearest neighbor from training set

Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network



Generator

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Convolutional Architectures

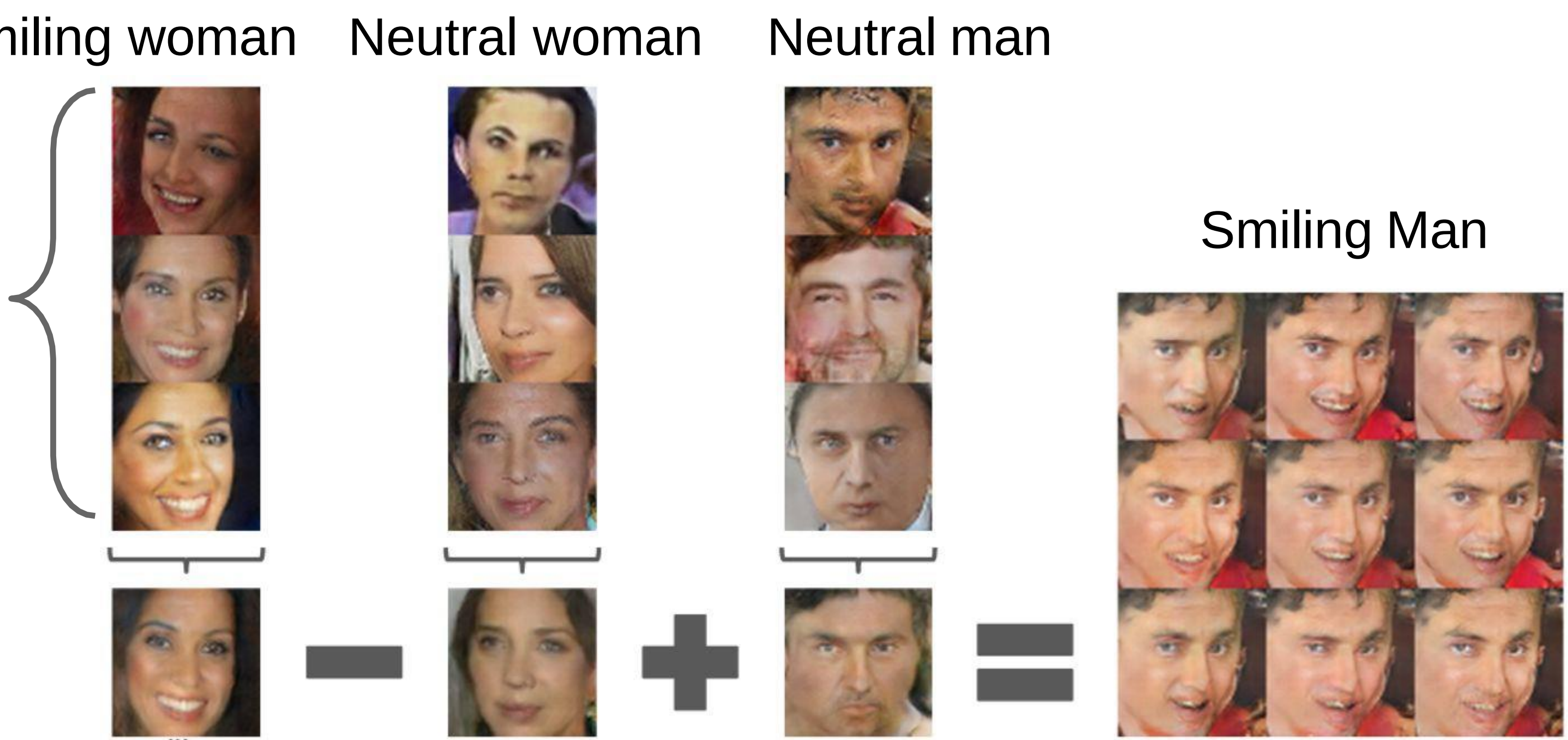
Samples from the model look much better!

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Interpretable Vector Math

Samples from the model

Average Z vectors, do arithmetic



Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

GANs

- Don’t work with an explicit density function
- Take game-theoretic approach: learn to generate from training distribution through 2-player game
- Pros:
  - Beautiful, state-of-the-art samples!
- Cons:
  - Trickier / more unstable to train
- Active areas of research:
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
  - Conditional GANs, GANs for all kinds of applications