



GE 461

Introduction to Data Science

Spring 2023

Deep Learning

Hamdi Dibekliolu

Slide Credits: F. Li, A. Karpathy, J. Johnson, G. Cinbis

So, What is DEEP Machine Learning

A few different ideas:

- **(Hierarchical) Compositionality**
 - Cascade of non-linear transformations
 - Multiple layers of representations
- **End-to-End Learning**
 - Learning (goal-driven) representations
 - Learning feature extraction
- **Distributed Representations**
 - No single neuron “encodes” everything
 - Groups of neurons work together

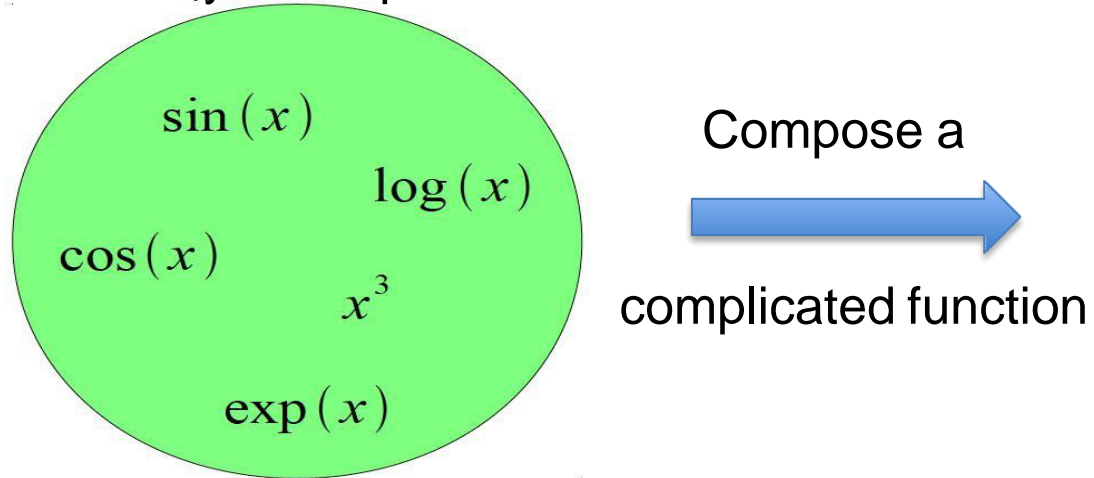
So, What is DEEP Machine Learning

A few different ideas:

- **(Hierarchical) Compositionality**
 - Cascade of non-linear transformations
 - Multiple layers of representations
- **End-to-End Learning**
 - Learning (goal-driven) representations
 - Learning feature extraction
- **Distributed Representations**
 - No single neuron “encodes” everything
 - Groups of neurons work together

Building A Complicated Function

Given a library of simple functions

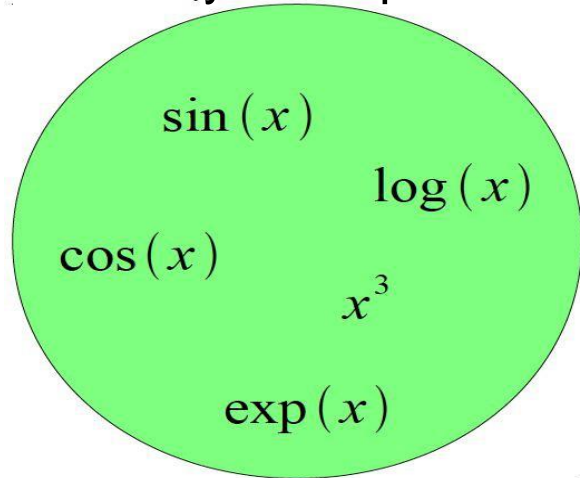


Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



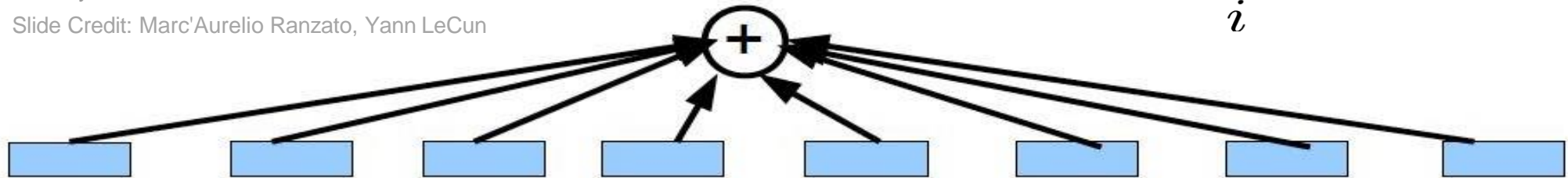
Compose a
→
complicated function

Idea 1: Linear Combinations

$$f(x) = \sum_i \alpha_i g_i(x)$$

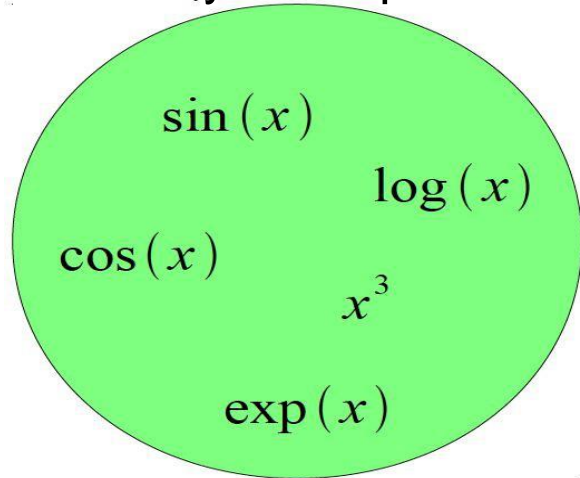
Slide by Dhruv Batra


Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Building A Complicated Function

Given a library of simple functions



Compose a

complicated function

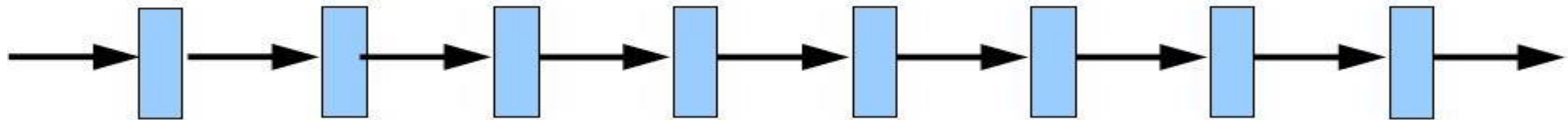
Idea 2: Compositions

- Deep Learning

$$f(x) = g_1(g_2(\dots(g_n(x)\dots)))$$

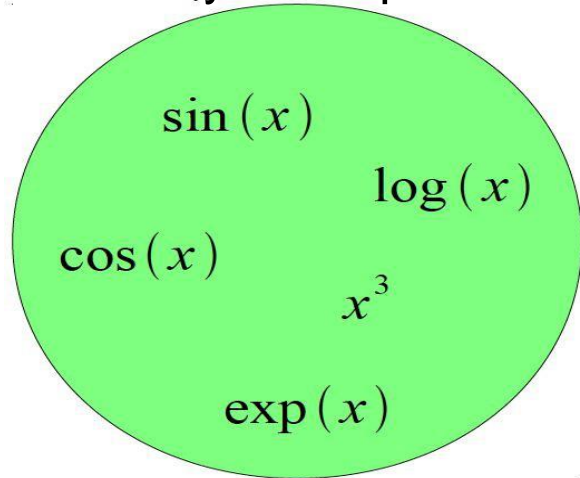
Slide by Dhruv Batra


Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Building A Complicated Function

Given a library of simple functions



Compose a

complicated function

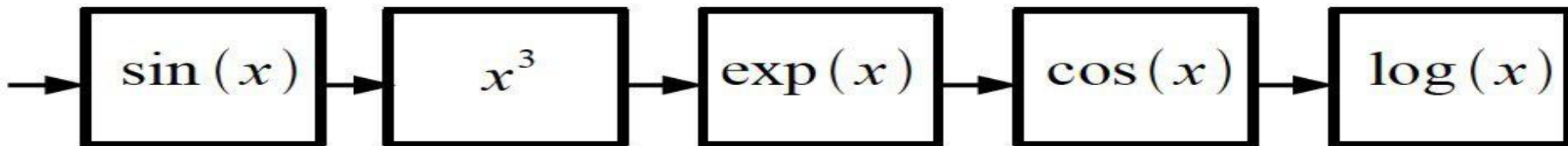
Idea 2: Compositions

- Deep Learning

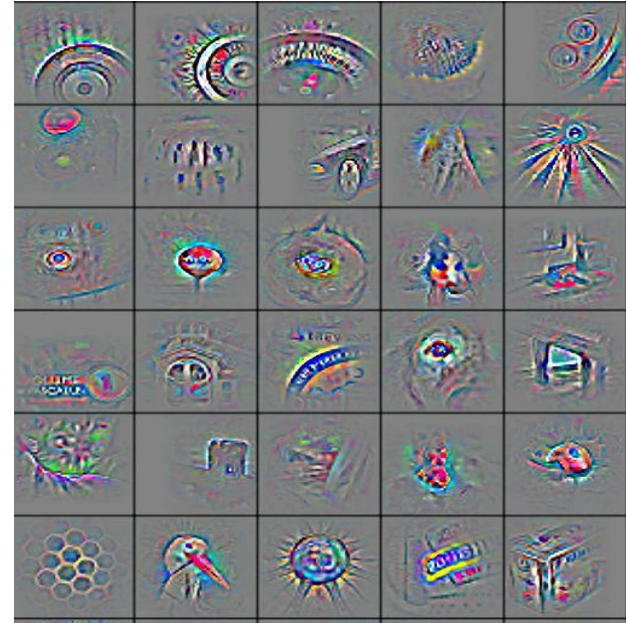
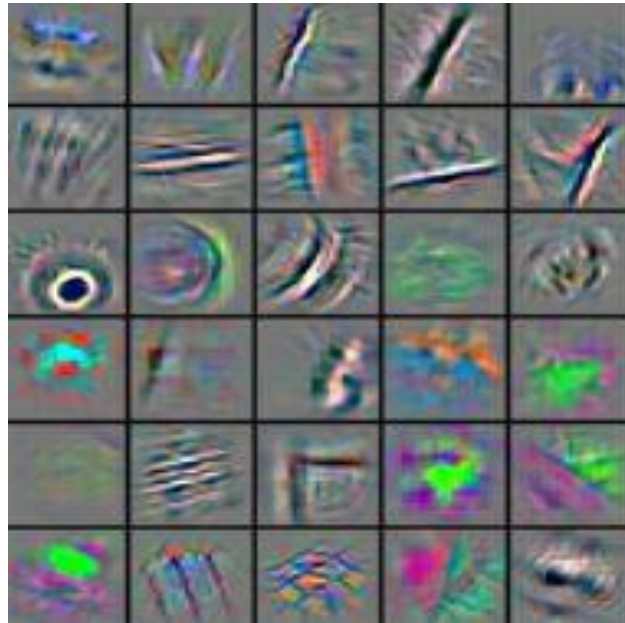
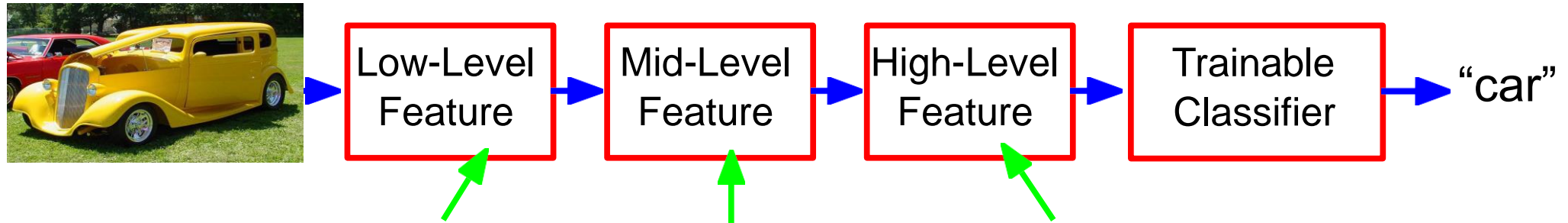
$$f(x) = \log(\cos(\exp(\sin^3(x))))$$

Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Deep Learning = Hierarchical Compositionality



Slide by Dhruv Batra

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

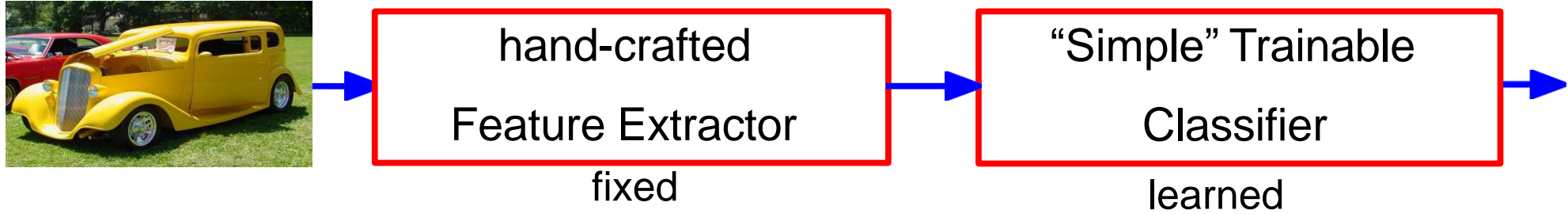
So, What is DEEP Machine Learning

A few different ideas:

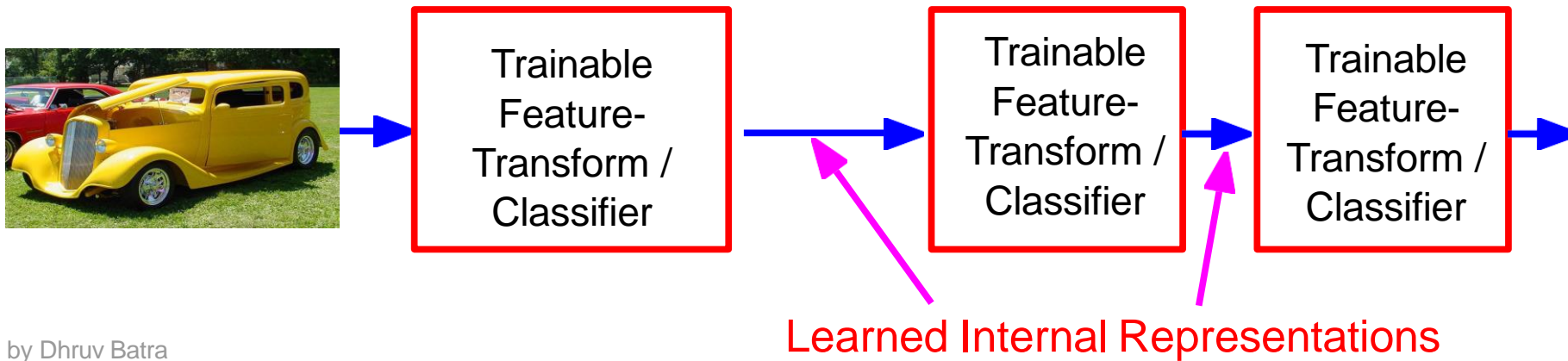
- **(Hierarchical) Compositionality**
 - Cascade of non-linear transformations
 - Multiple layers of representations
- **End-to-End Learning**
 - Learning (goal-driven) representations
 - Learning feature extraction
- **Distributed Representations**
 - No single neuron “encodes” everything
 - Groups of neurons work together

“Shallow” vs Deep Learning

- “Shallow” models



- Deep models (especially supervised deep learning)



Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

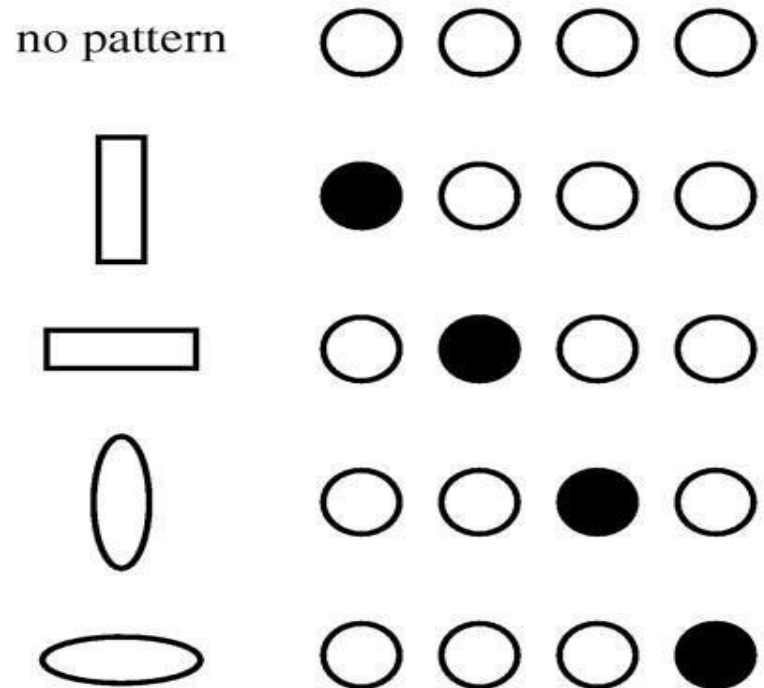
So, What is DEEP Machine Learning

A few different ideas:

- **(Hierarchical) Compositionality**
 - Cascade of non-linear transformations
 - Multiple layers of representations
- **End-to-End Learning**
 - Learning (goal-driven) representations
 - Learning feature extraction
- **Distributed Representations**
 - No single neuron “encodes” everything
 - Groups of neurons work together

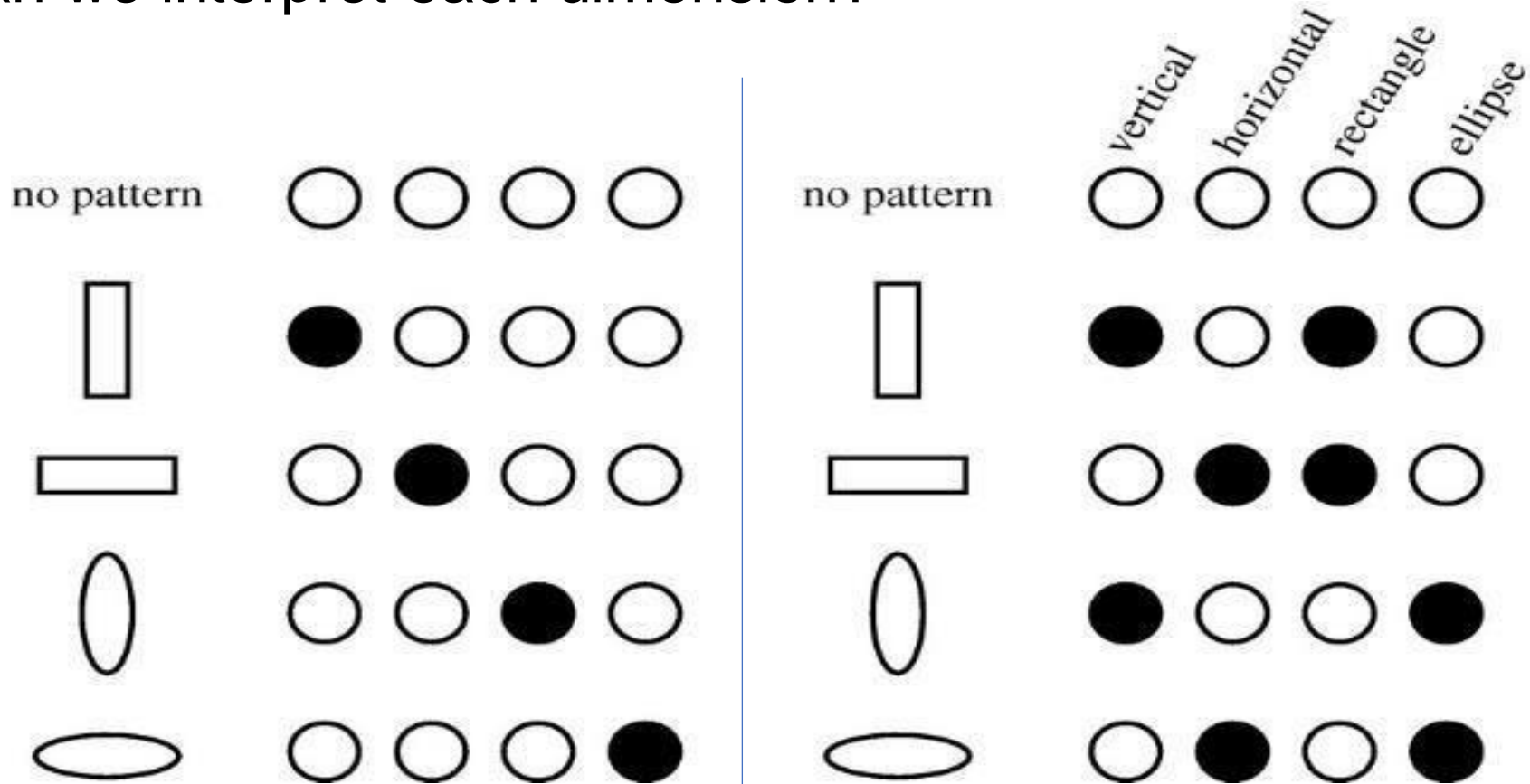
Distributed Representations Toy Example

- Local vs Distributed

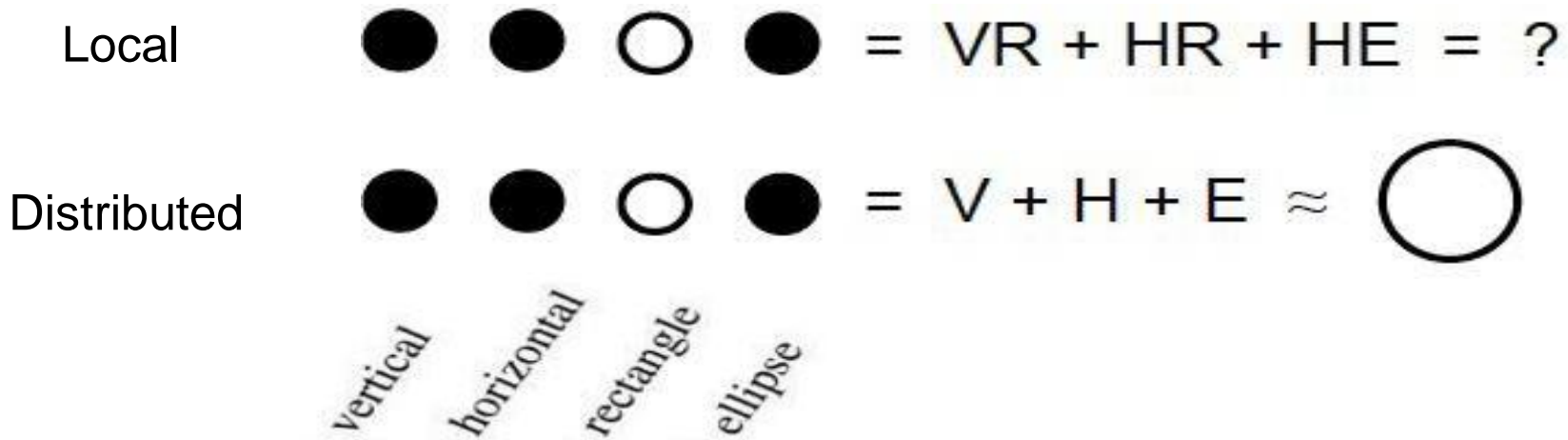


Distributed Representations Toy Example

- Can we interpret each dimension?



Power of distributed representations!



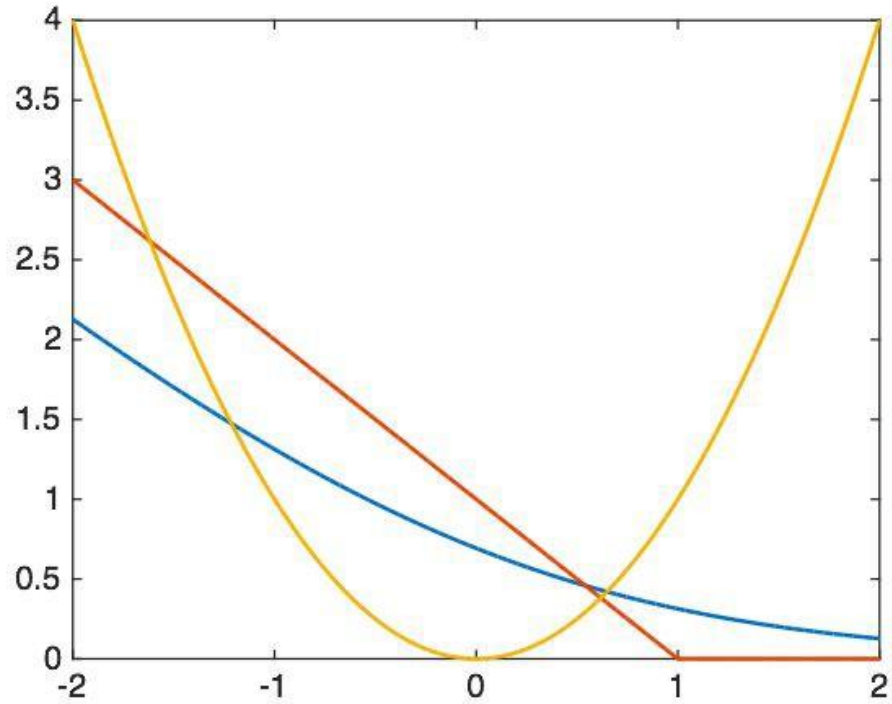
$$f(x, W) = Wx$$

- Loss function
- Optimization
- Convolutional Nets
- Recurrent Nets

Loss Functions

Loss functions

- There are many different loss functions



- Log Loss / Cross Entropy
- Hinge Loss
- Square Loss

Classification Losses

Hinge Loss/Multi class SVM Loss

$$SVM\ Loss = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

- s_j - Computed score of the training example for jth class.
- $y(i)$ - Ground truth label for ith training example.

Classification Losses

Cross Entropy Loss/Negative Log Likelihood

$$\text{CrossEntropyLoss} = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

- s_j – Computed score of the training example for jth class.
- $y(i)$ - Ground truth label for ith training example.

Regression Losses

Mean Square Error/Quadratic Loss/L2 Loss

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

- n - Number of training examples.
- i - i th training example in a data set.
- $y(i)$ - Ground truth label for i th training example.
- $y_{\text{hat}}(i)$ - Prediction for i th training example.

Regression Losses

Mean Absolute Error/L1 Loss

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

- n - Number of training examples.
- i - i th training example in a data set.
- $y(i)$ - Ground truth label for i th training example.
- $y_{\text{hat}}(i)$ - Prediction for i th training example.

Regression Losses

Mean Bias Error

$$MBE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)}{n}$$

- n - Number of training examples.
- i - i th training example in a data set.
- $y(i)$ - Ground truth label for i th training example.
- $y_{\text{hat}}(i)$ - Prediction for i th training example.

Weight Regularization

λ = regularization strength
(hyperparameter)

$$L = \frac{1}{N} \sum_{i=1}^N \text{Loss}_i + \lambda R(W)$$

Some reg. types:

L2 regularization

L1 regularization

Elastic net (L1 + L2)

...

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Which one does L2 regularization choose?

$$w_1^T x = w_2^T x = 1$$

L2 regularization: motivation

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

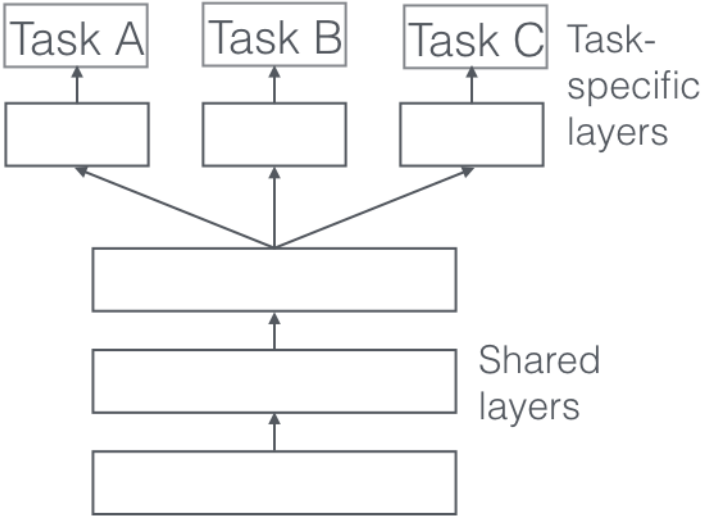
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Why does it make sense?

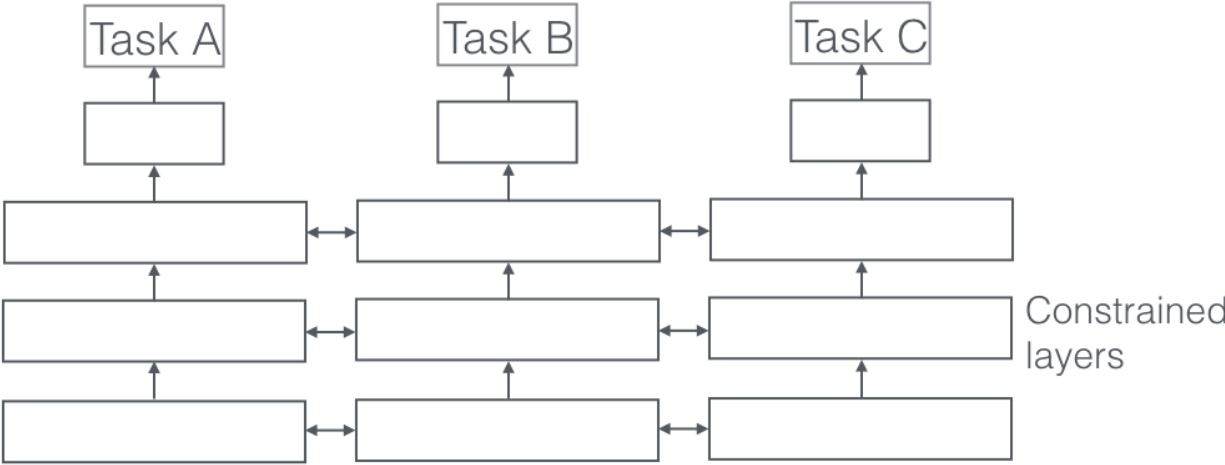
$$w_1^T x = w_2^T x = 1$$

Multi-task Learning

Jointly minimize the losses of different tasks



Hard parameter sharing for multi-task learning in deep neural networks

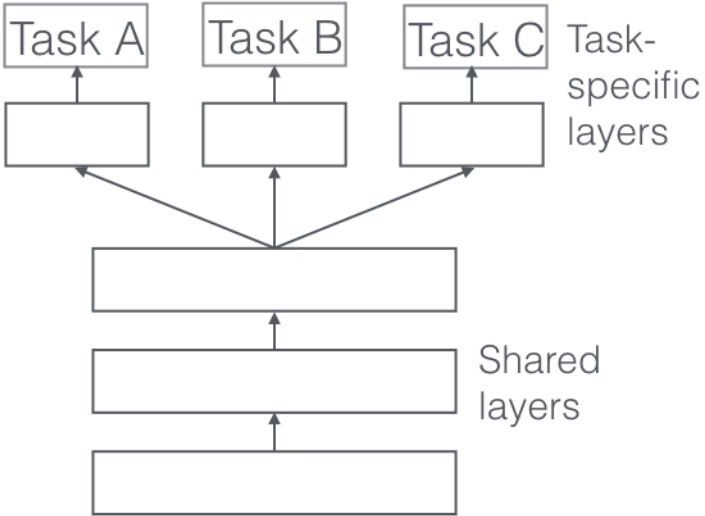


Soft parameter sharing for multi-task learning in deep neural networks

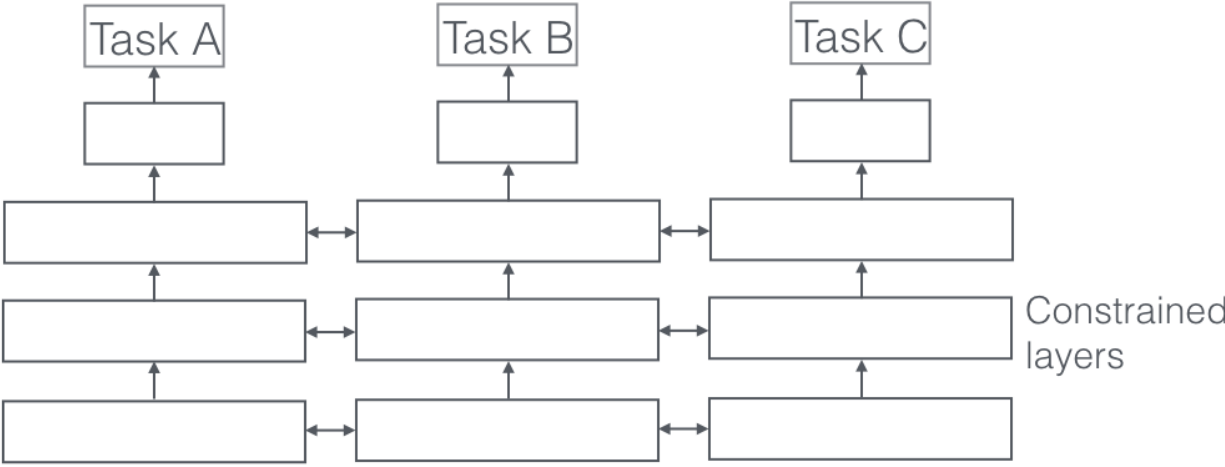
Multi-task Learning

Jointly minimize the losses of different tasks (combine loss terms)

$$L = l_a + \alpha l_b + \beta l_c + \dots$$



Hard parameter sharing for multi-task learning in deep neural networks

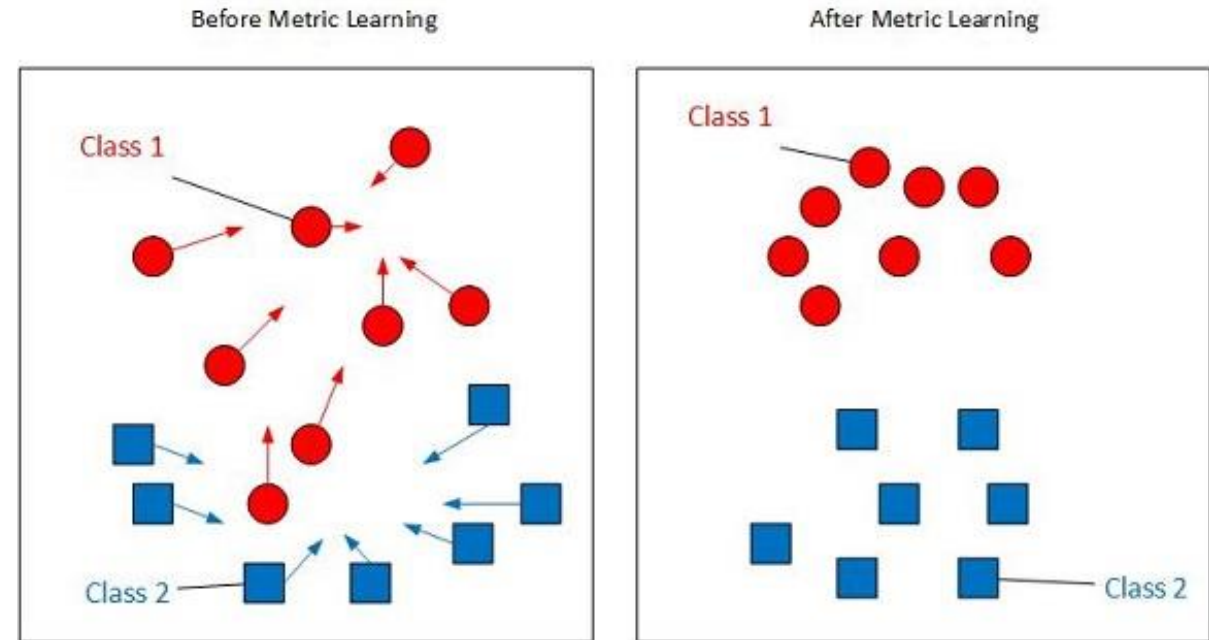


Soft parameter sharing for multi-task learning in deep neural networks

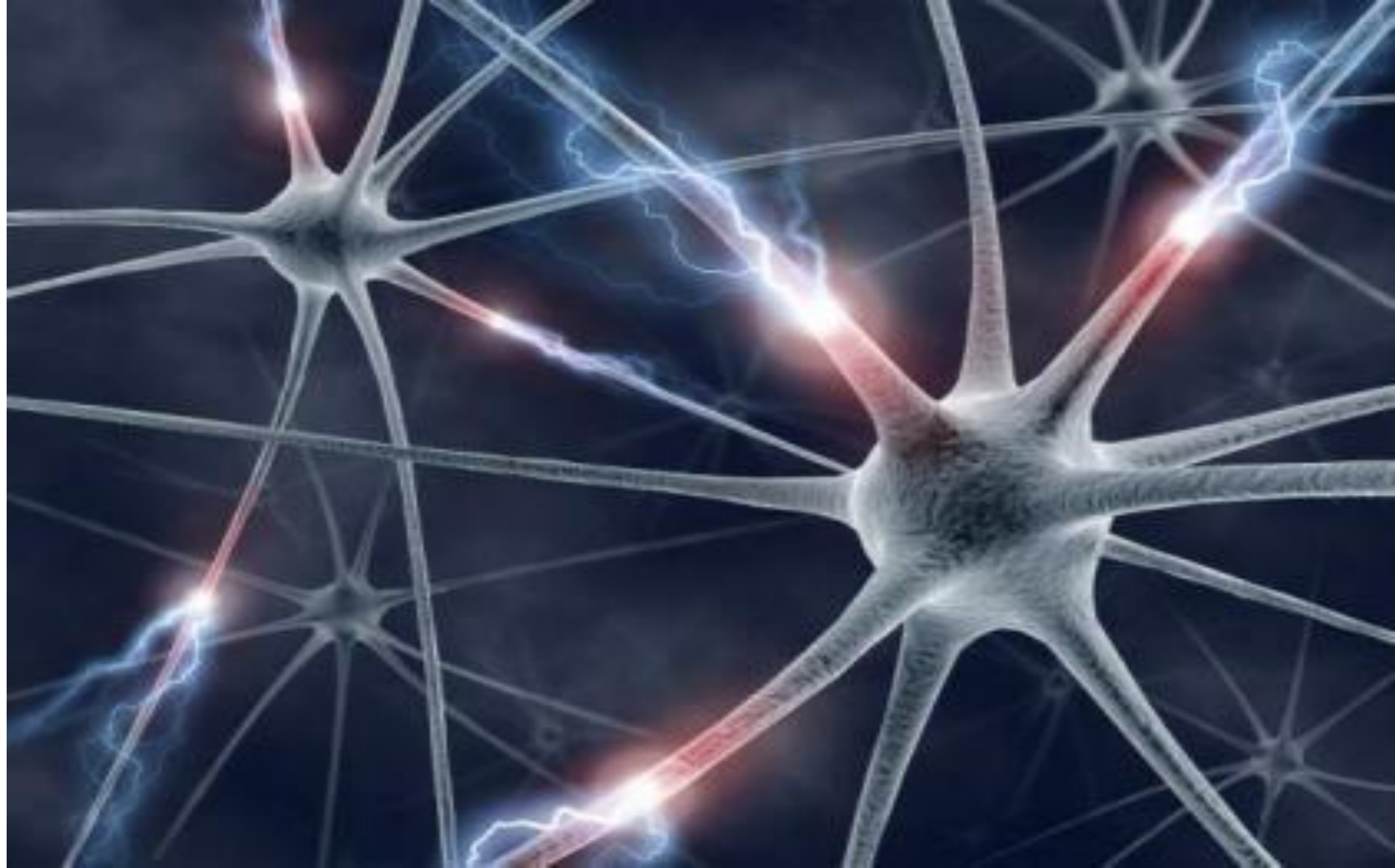
Metric/Contrastive Learning

Learn *distinctiveness*

1. A distance-based loss function (as opposed to prediction error-based loss functions like Logistic loss or Hinge loss used in Classification).
2. Like any distance-based loss, it tries to ensure that semantically similar examples are embedded close together.
3. Defined based on pairs (+/- class pairs) or groups of samples.



$$L_i = \sum_{i \neq j} \|w^T x_{i,c1} - w^T x_{j,c1}\| - \sum_k \|w^T x_{i,c1} - w^T x_{k,c2}\|$$



Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network:

$$f = W_2 \max(0, W_1 x)$$

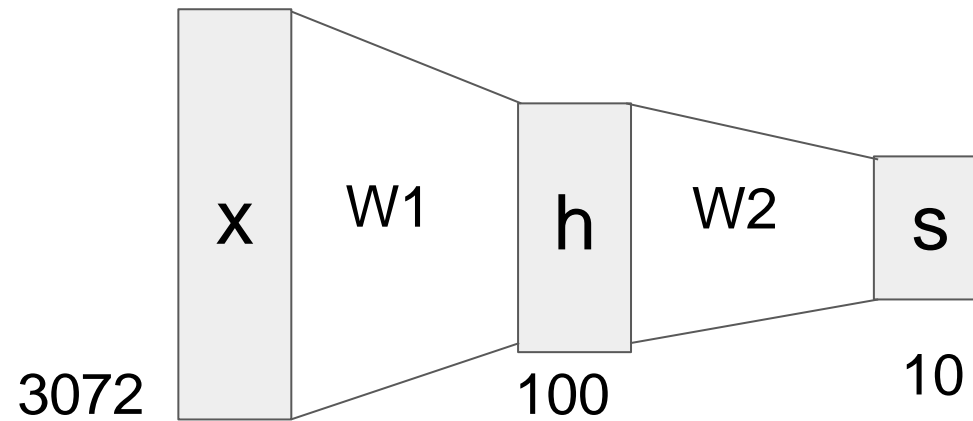
Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network:

$$f = W_2 \max(0, W_1 x)$$



Neural Networks

Linear score function:

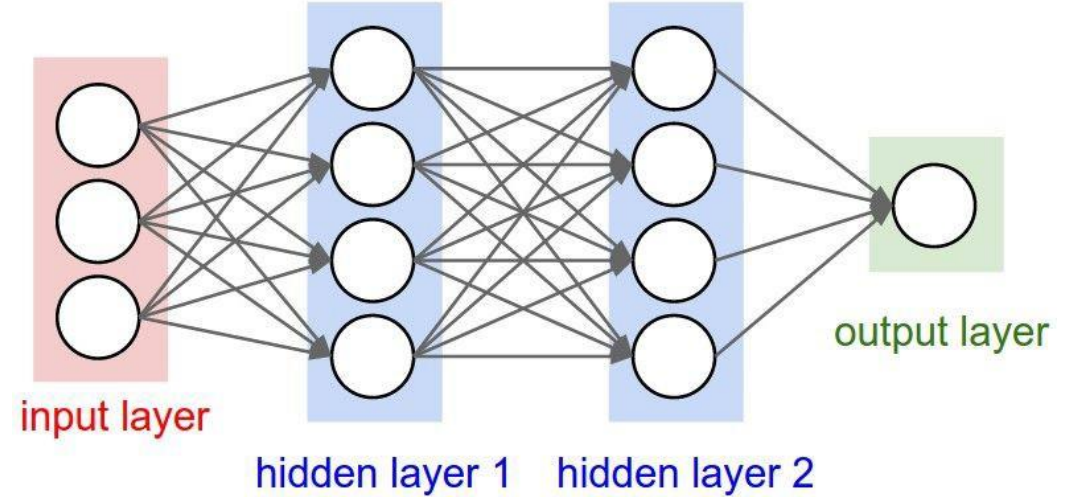
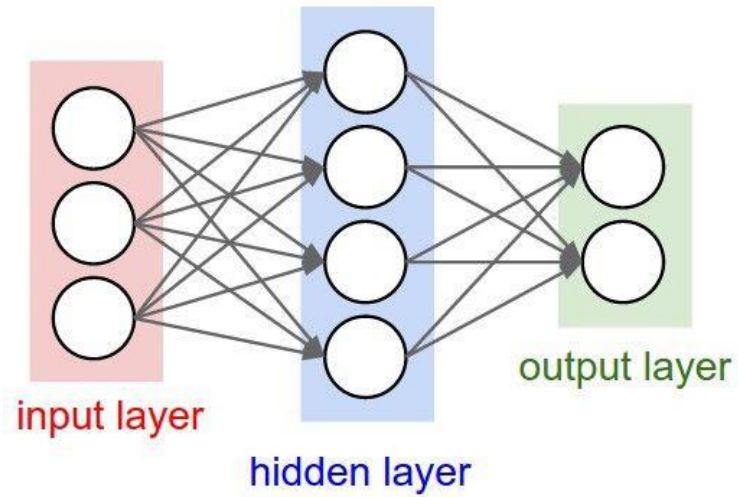
$$f = Wx$$

2-layer Neural Network
or 3-layer Neural Network

$$f = W_2 \max(0, W_1 x)$$

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Neural Networks: Architectures

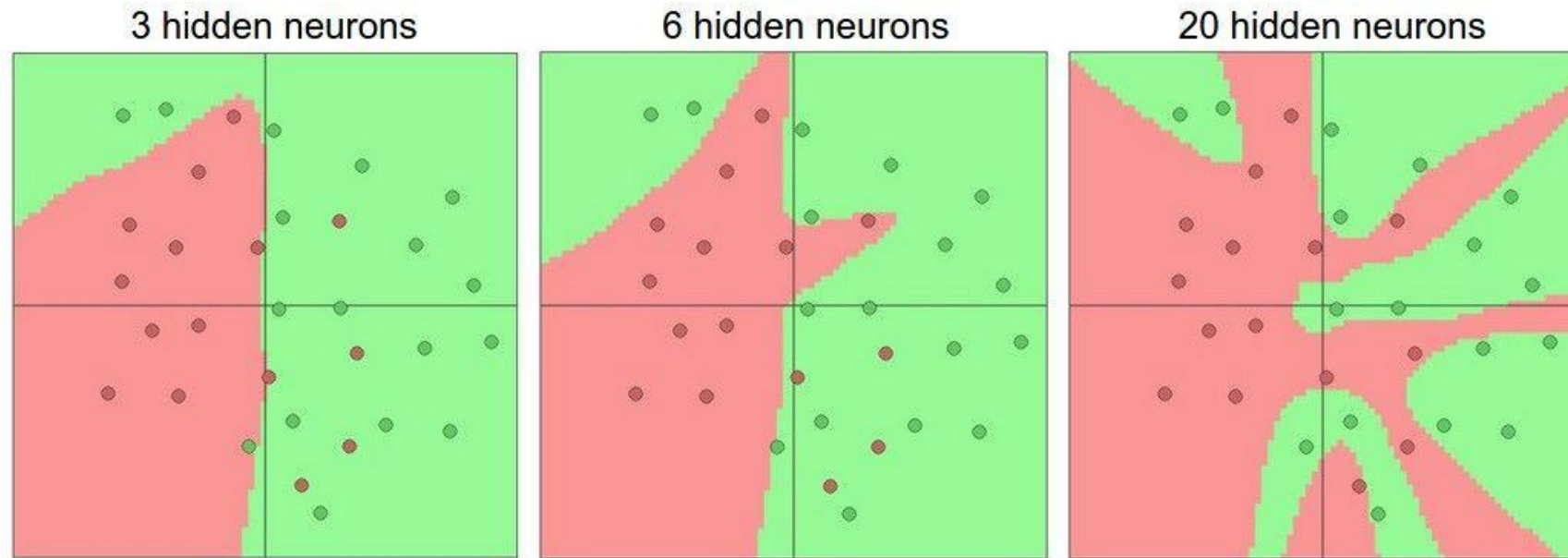


“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

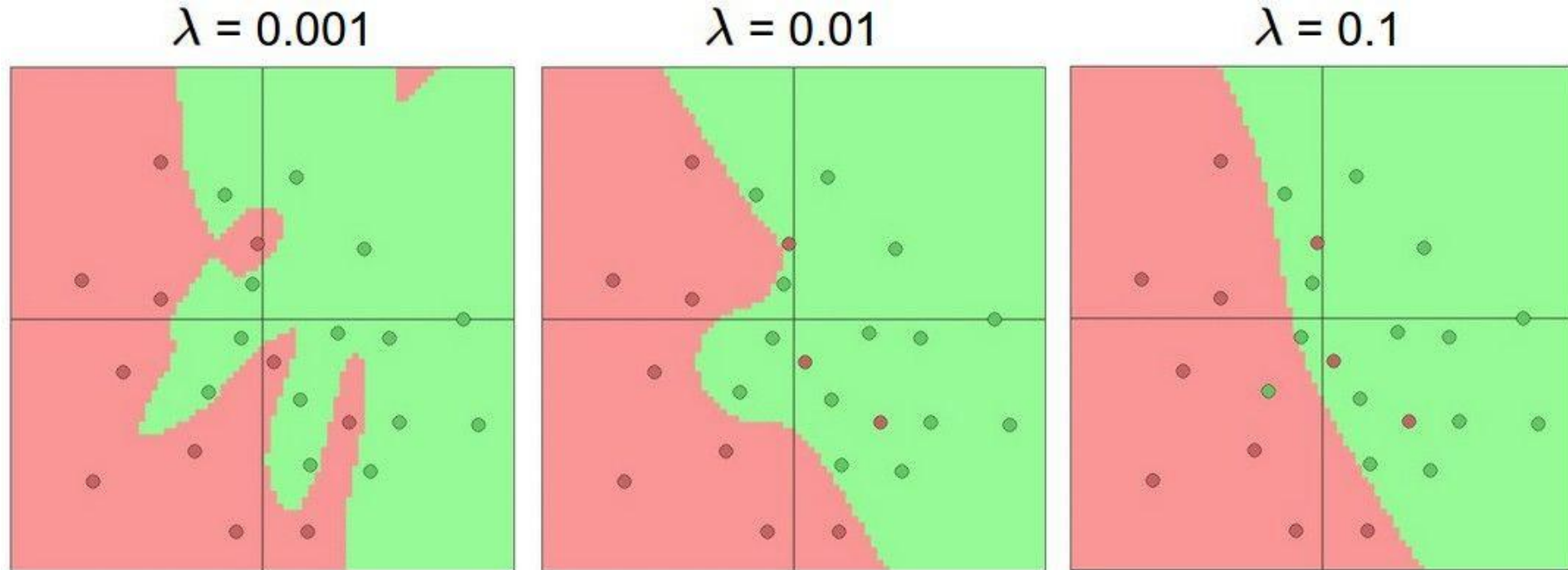
“Fully-connected” layers

Setting the number of layers and their sizes



↑
more neurons = more capacity

Do not use size of neural network as a regularizer. Use stronger regularization instead:

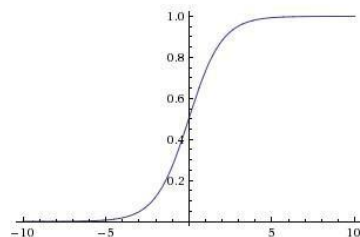


Activation Functions

Activation Functions

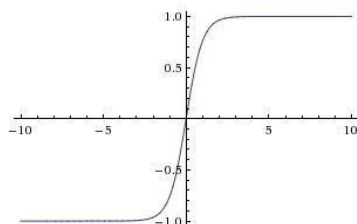
Sigmoid

$$\sigma(x) = 1 / (1 + e^{-x})$$



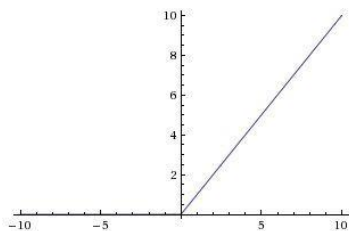
tanh

 $\tanh(x)$



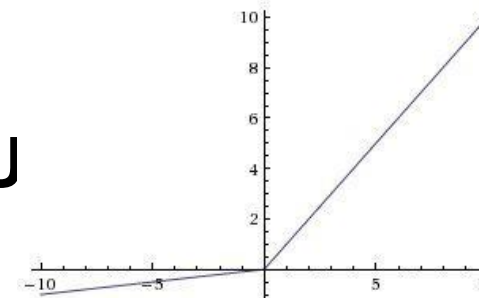
ReLU

 $\max(0, x)$



Leaky ReLU

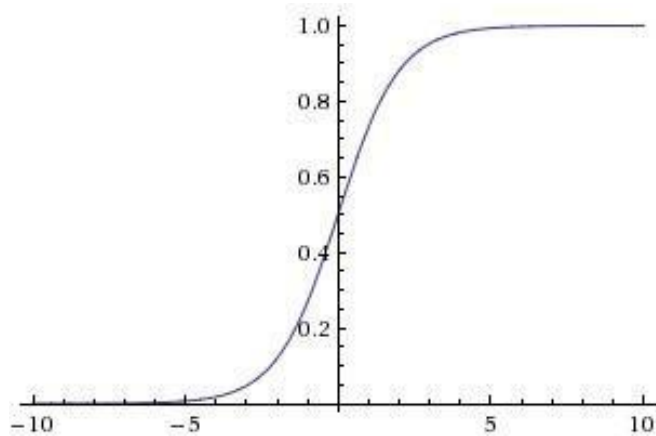
$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Activation Functions



Sigmoid

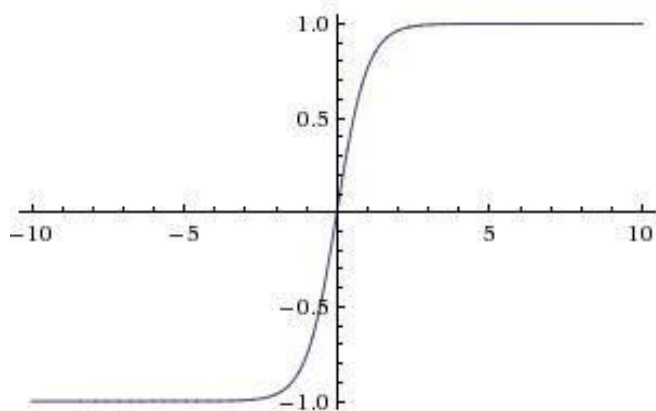
$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit computationally expensive

Activation Functions



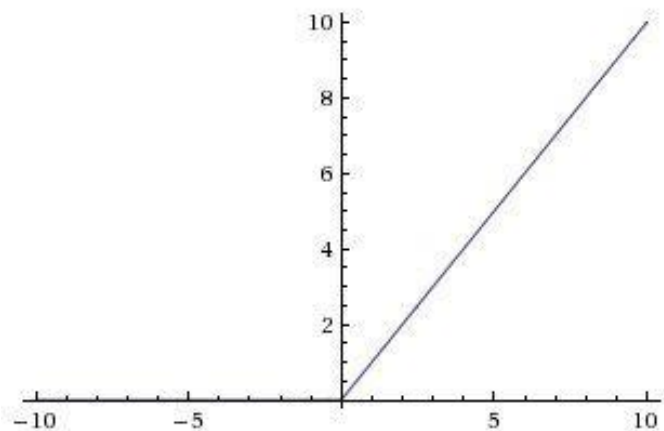
tanh(x)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes **$f(x) = \max(0, x)$**
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Not zero-centered output
- An annoyance:

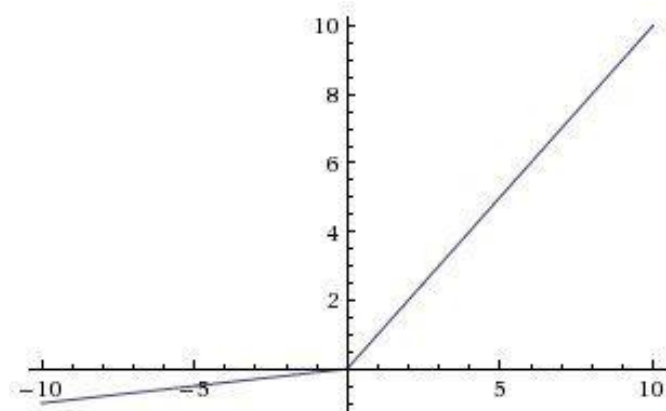
hint: what is the gradient when $x < 0$?

[Krizhevsky et al., 2012]

Activation Functions

[Mass et al., 2013]

[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

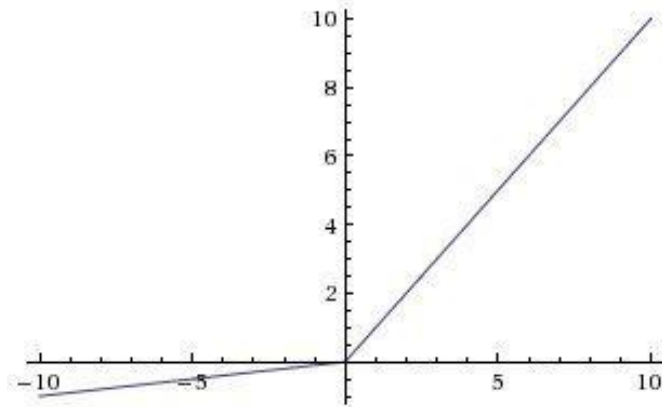
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Activation Functions

[Mass et al., 2013]

[He et al., 2015]



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into α
(parameter)

Maxout “Neuron”

[Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

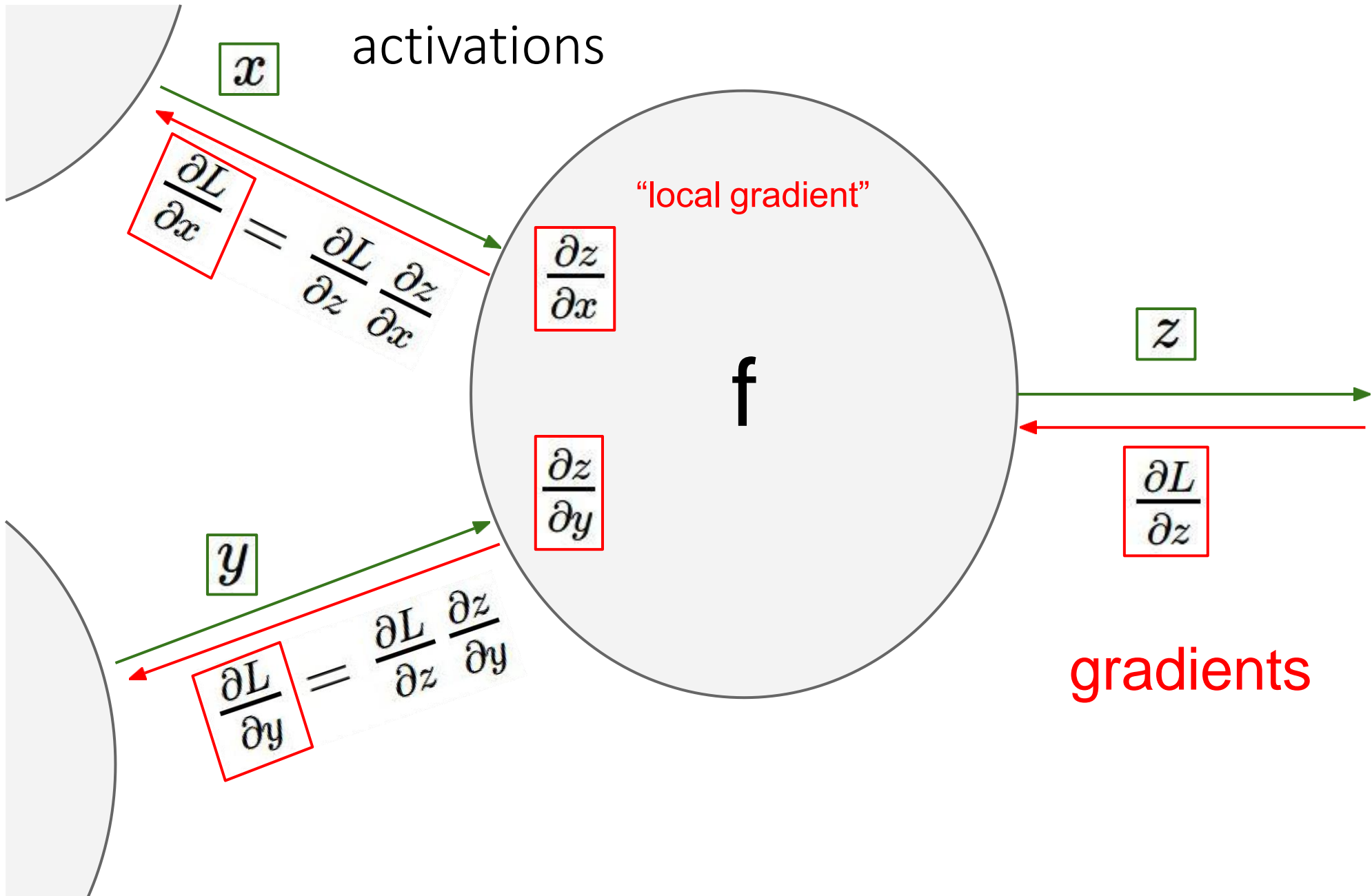
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters/neuron :(

In practice

- Use **ReLU**. Be careful with your learning rates
- Try out **Leaky ReLU / Maxout**
- Try out **tanh** but don't expect much
- **Don't use sigmoid**

Parameter Updates



Training a neural network, main loop:

```
while True:  
    data_batch = dataset.sample_data_batch()  
    loss = network.forward(data_batch)  
    dx = network.backward()  
    x += - learning_rate * dx
```

simple gradient descent update



Optimize the parameters using one of the SGD variants

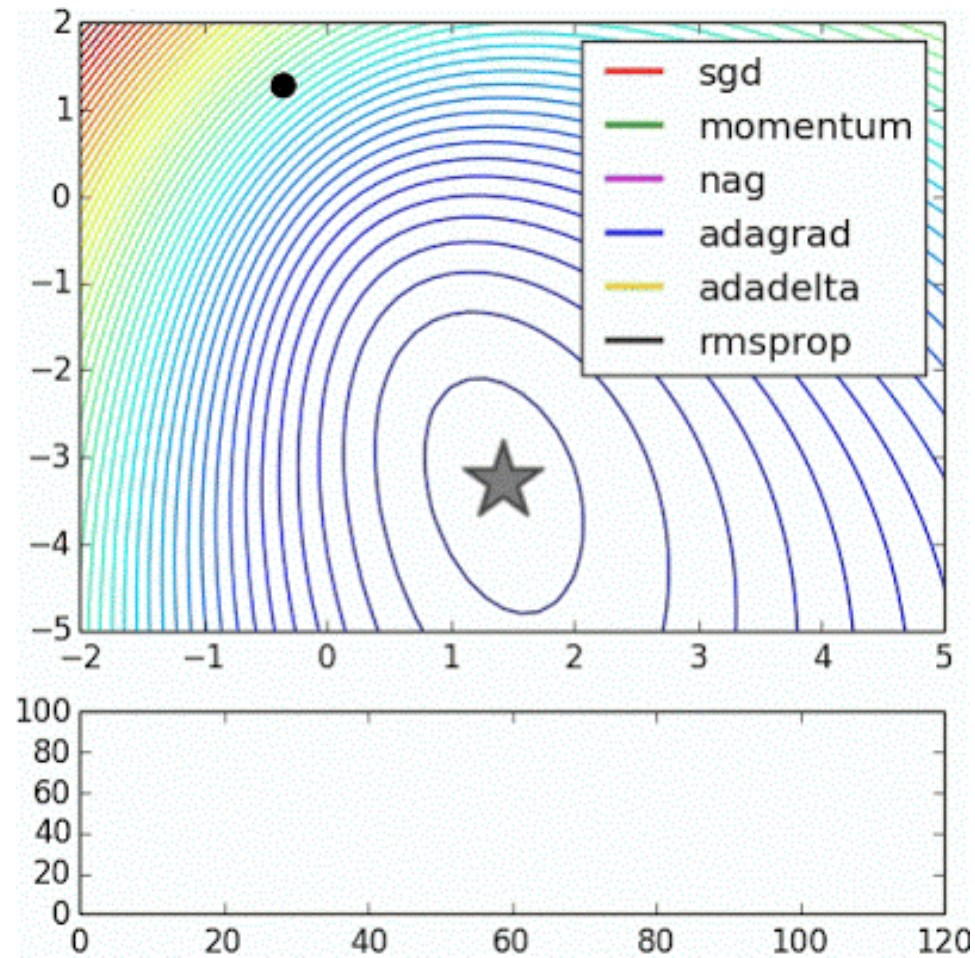
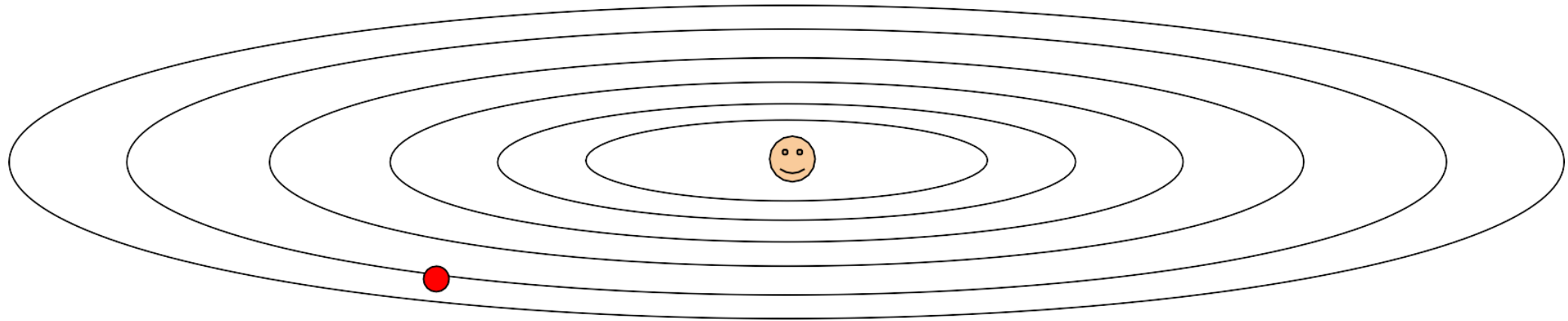


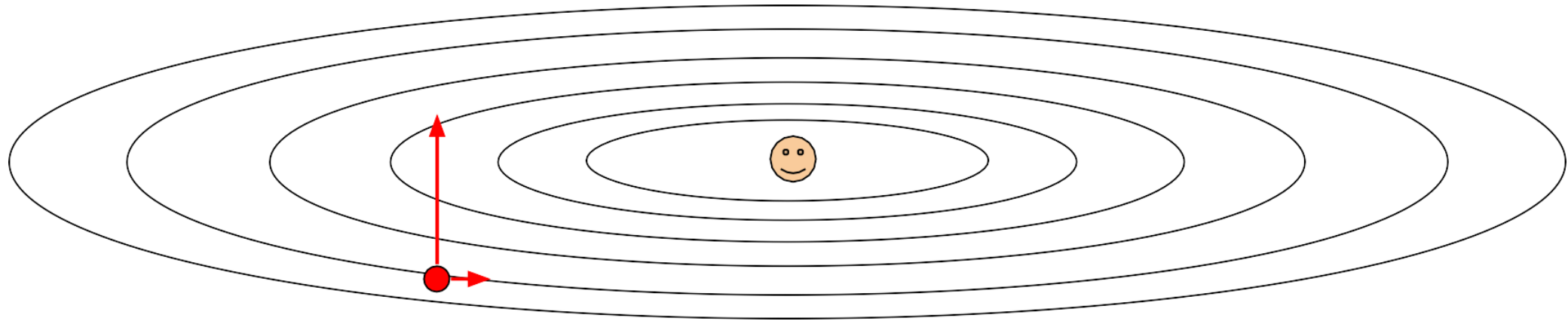
Image credits: Alec Radford

Suppose loss function is steep vertically but shallow horizontally:



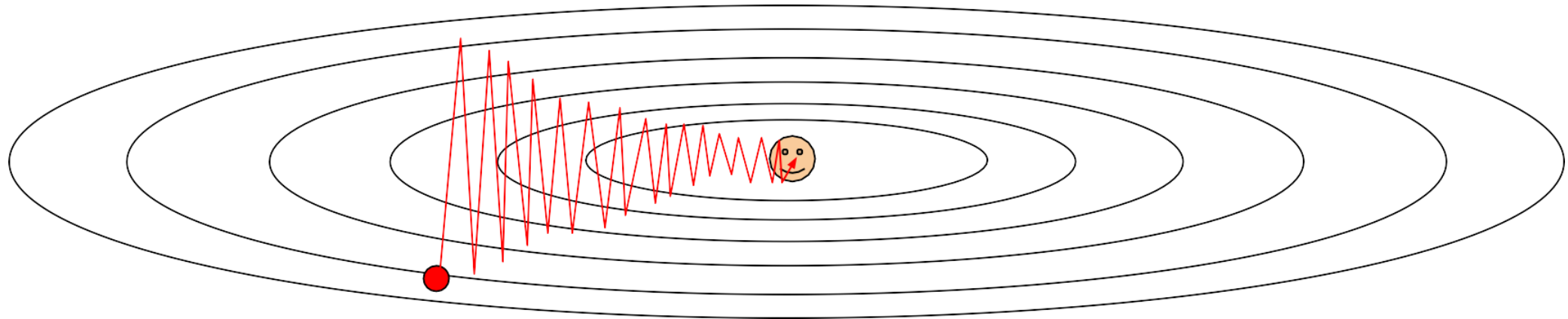
Q: What is the trajectory along which we converge towards the minimum with SGD?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with SGD?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with SGD? **very slow progress along flat direction, jitter along steep one**

Momentum Update

```
# Gradient descent update  
x += - learning_rate * dx
```



```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

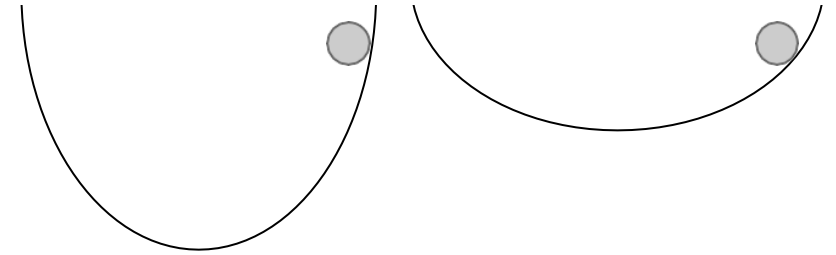
- Physical interpretation as ball rolling down the loss function + friction (mu coefficient).
- mu = usually ~0.5, 0.9, or 0.99 (Sometimes annealed over time, e.g. from 0.5 -> 0.99)

Momentum Update

```
# Gradient descent update  
x += - learning_rate * dx
```



```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

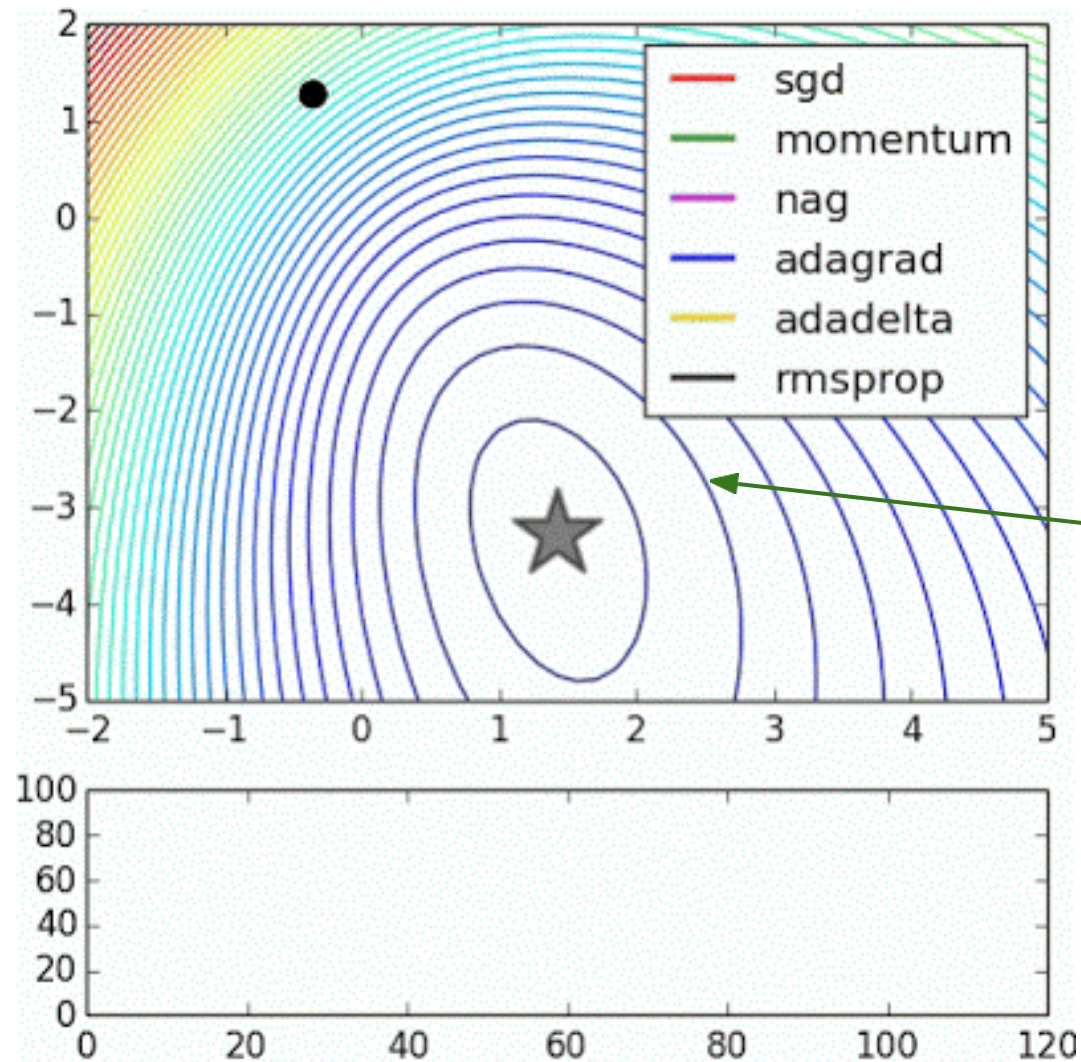


- Allows a velocity to “build up” along shallow (yet consistent) directions
- Velocity becomes damped in steep (inconsistent) direction due to quickly changing sign

SGD

VS

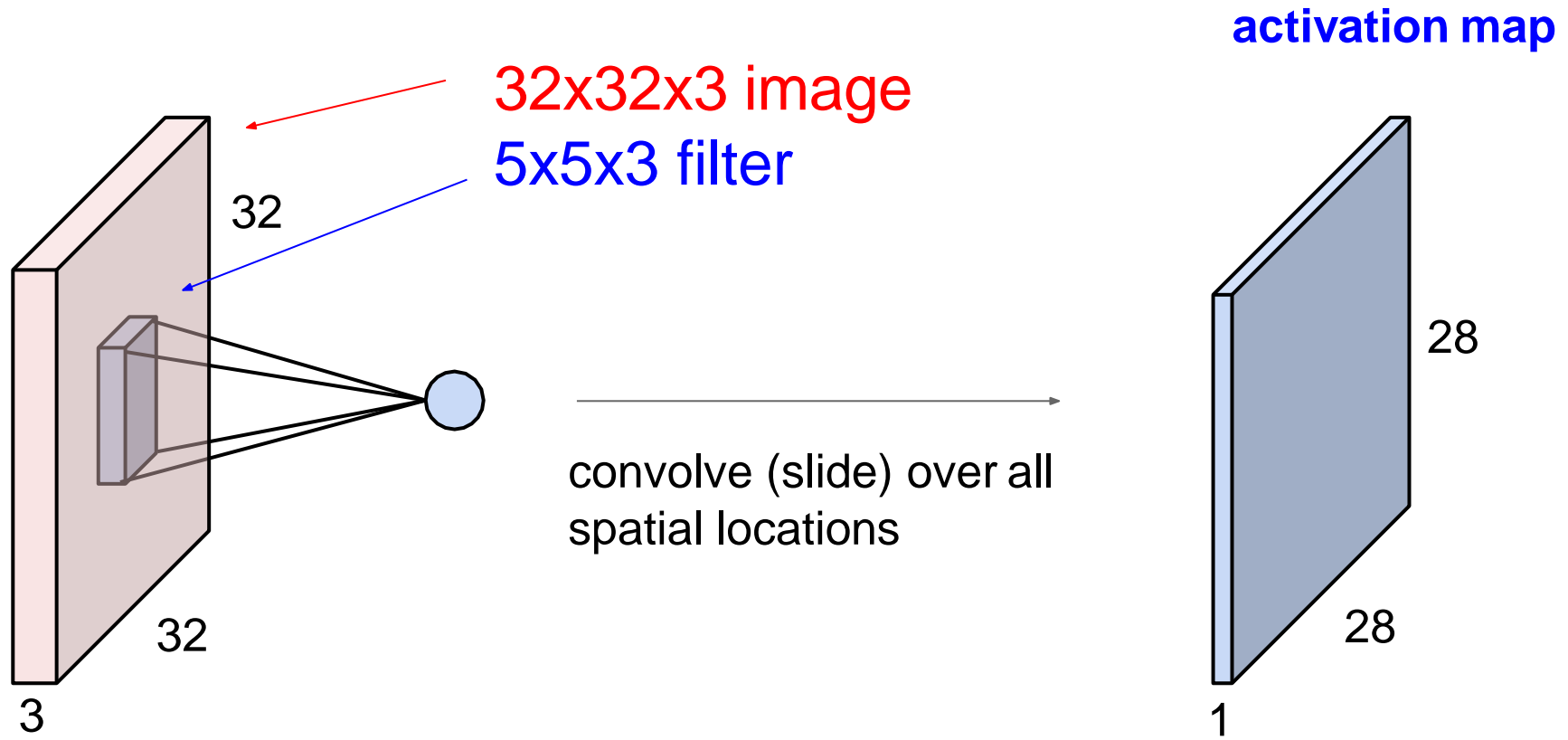
Momentum



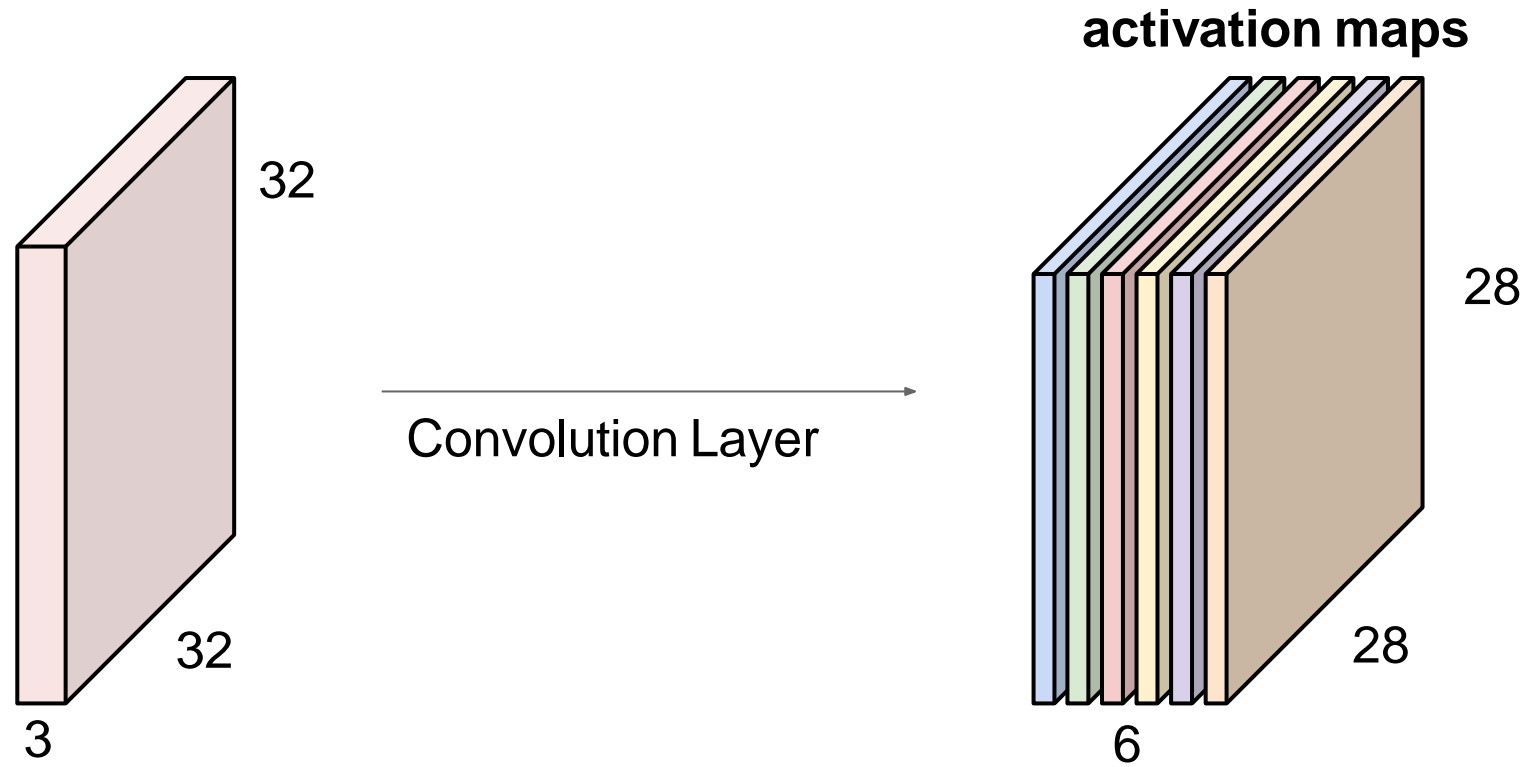
notice momentum overshooting the target, but overall getting to the minimum much faster.

Convolutional Neural Networks

Convolution Layer

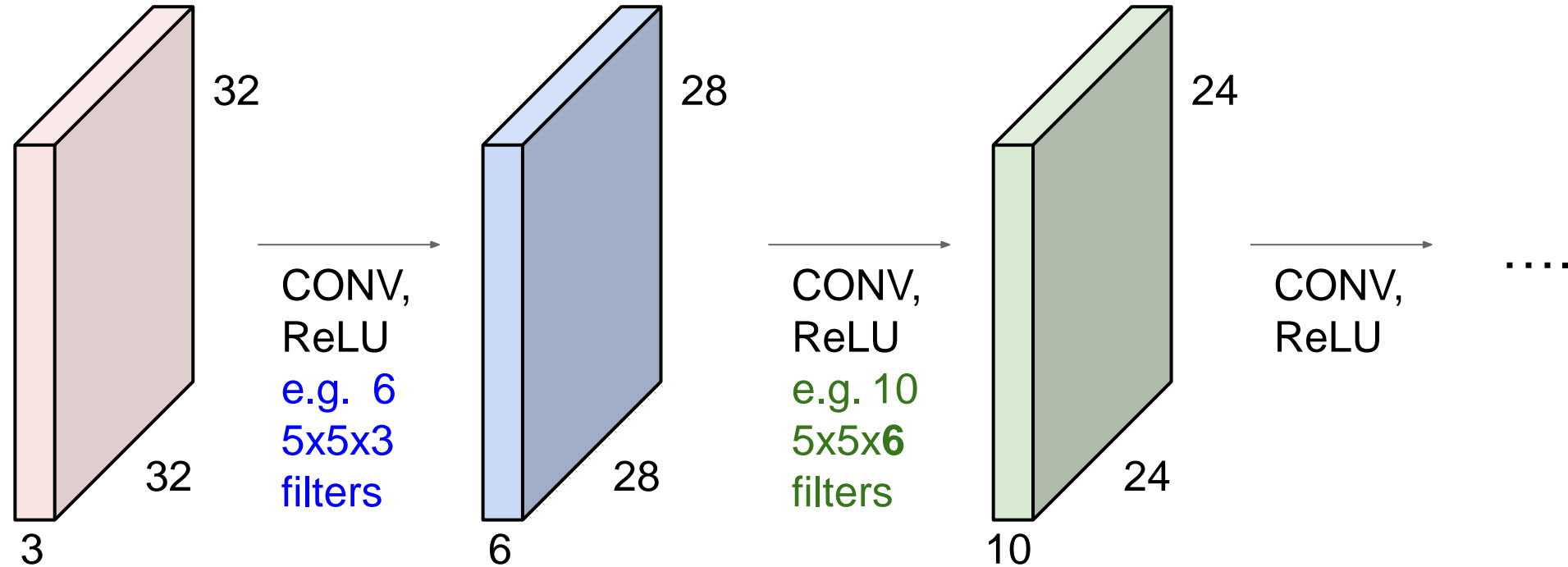


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

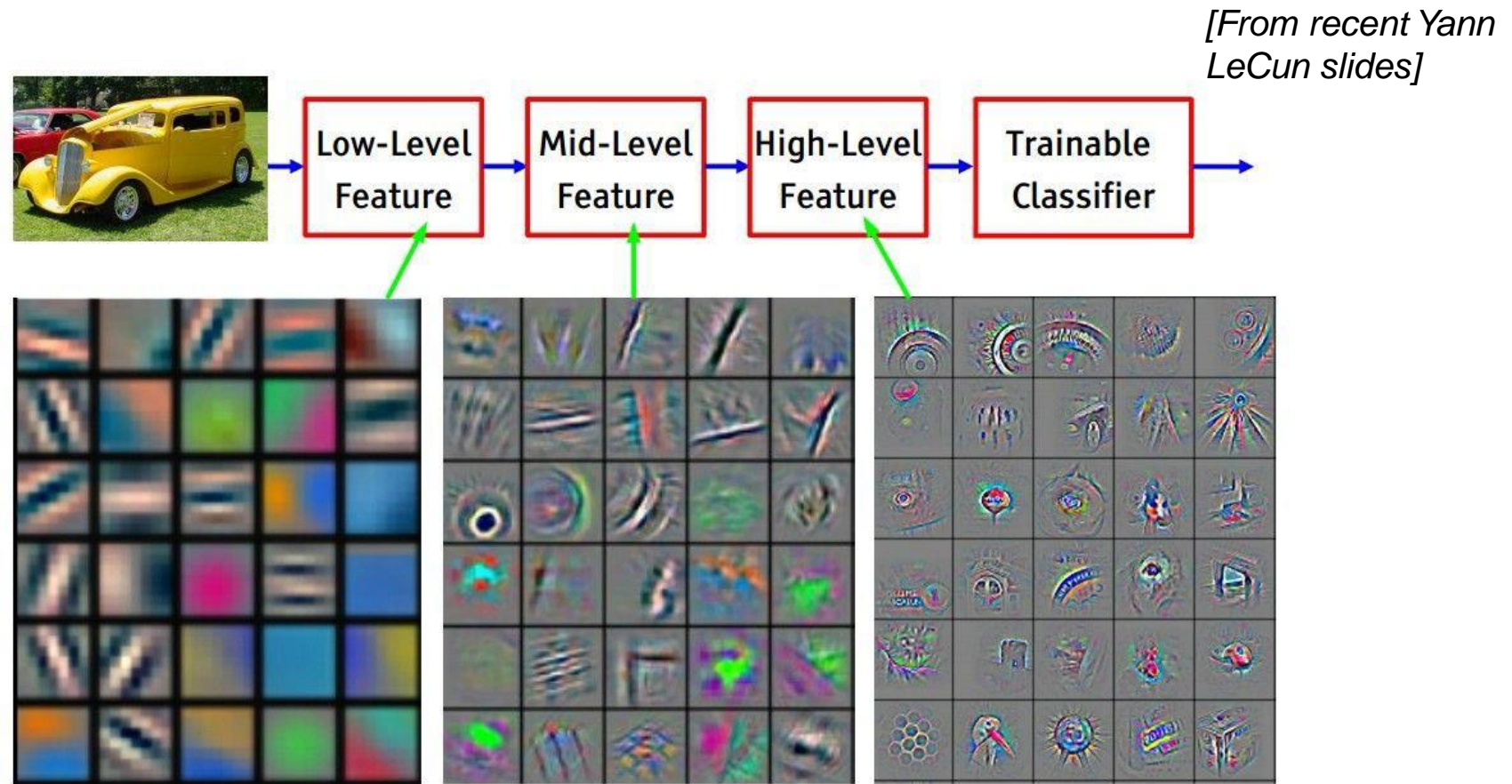


We stack these up to get a “new image” of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



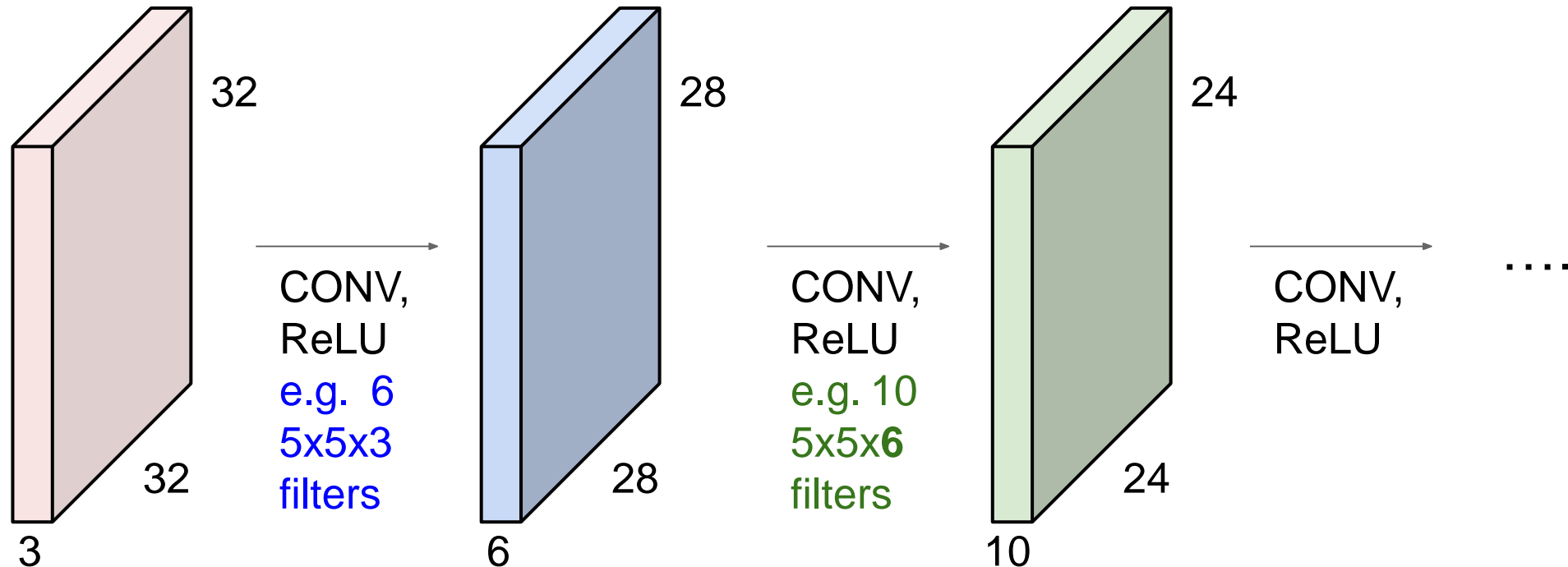
Preview:



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

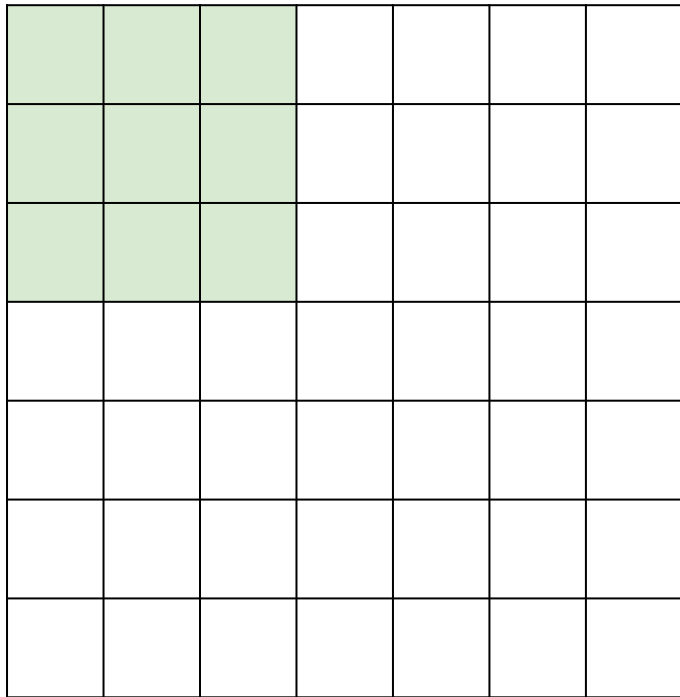
Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 → 28 → 24 ...). Shrinking too fast is not good, doesn't work well.



A closer look at spatial dimensions:

7

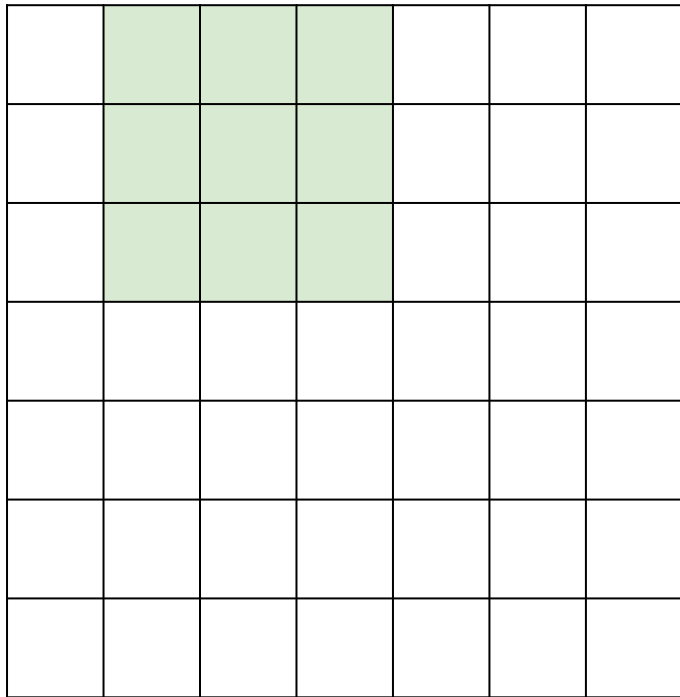


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:

7

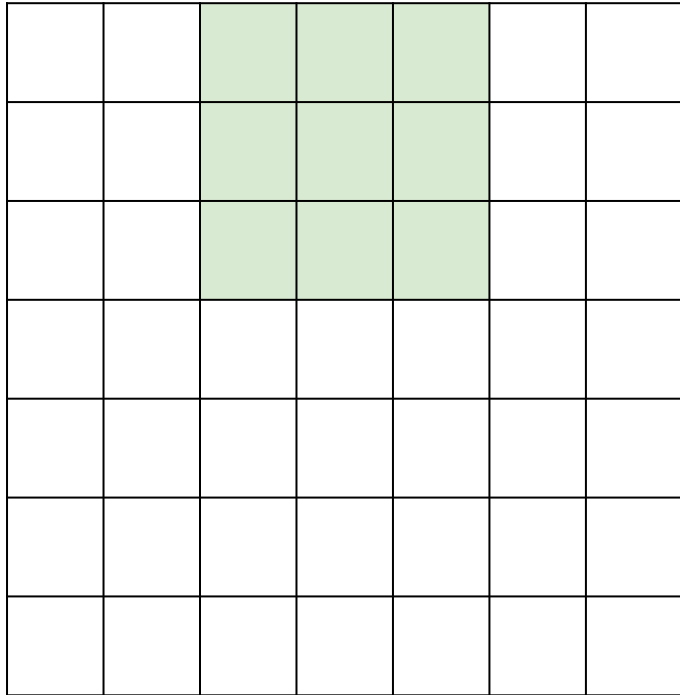


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:

7

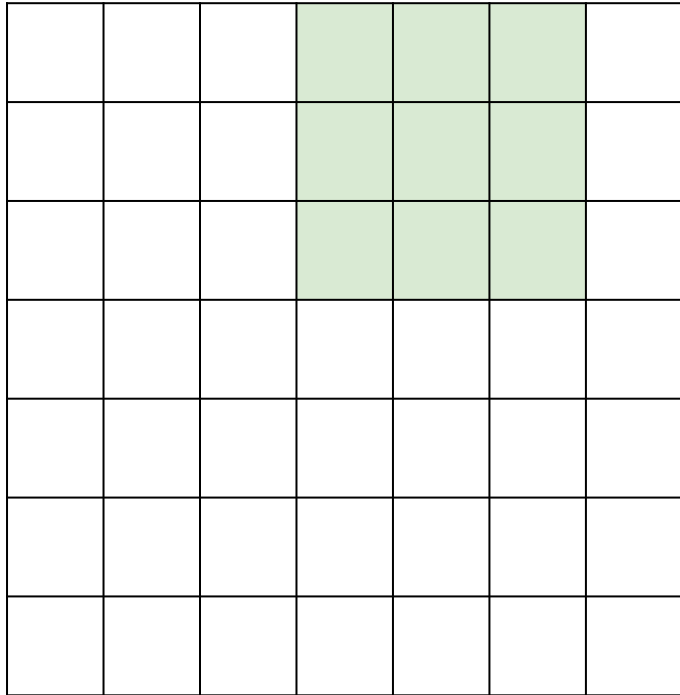


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:

7

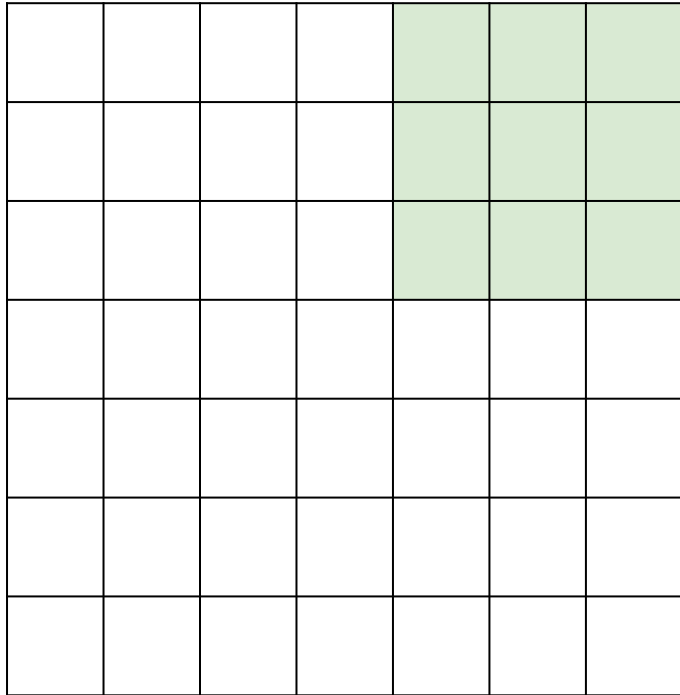


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:

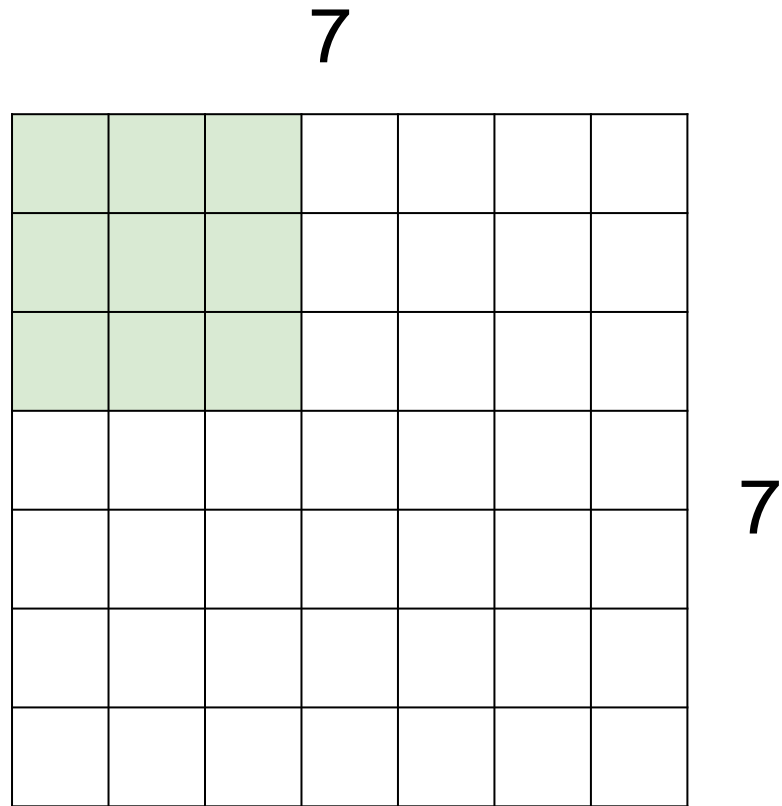
7



7

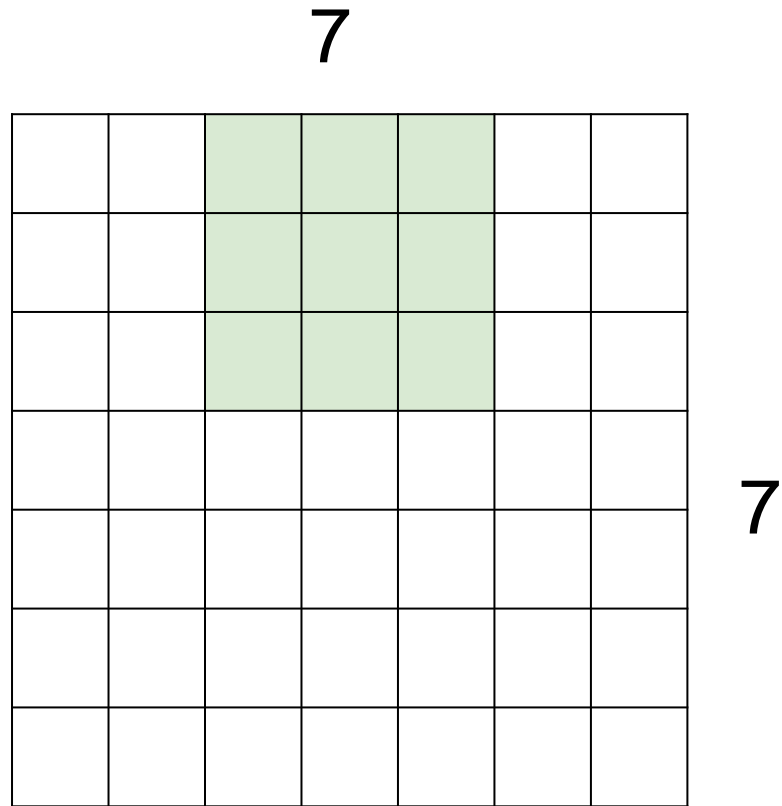
7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**
→ **5x5 output**

A closer look at spatial dimensions:



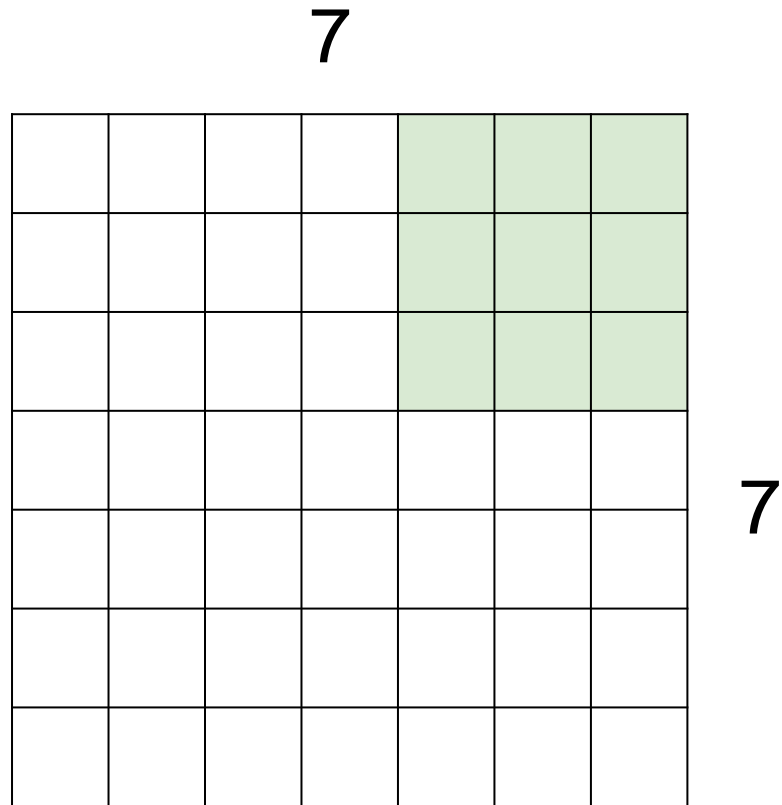
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



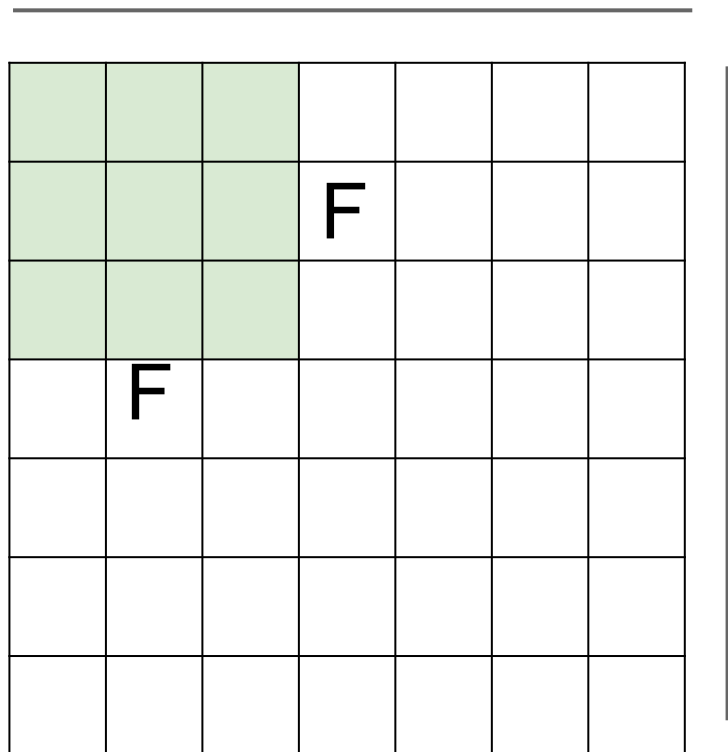
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
→ **3x3 output!**

N



Output size:

$$(N - F) / \text{stride} + 1$$

e.g. $N = 7, F = 3$:

$$\text{stride } 1 \Rightarrow (7 - 3) / 1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3) / 2 + 1 = 3$$

...

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

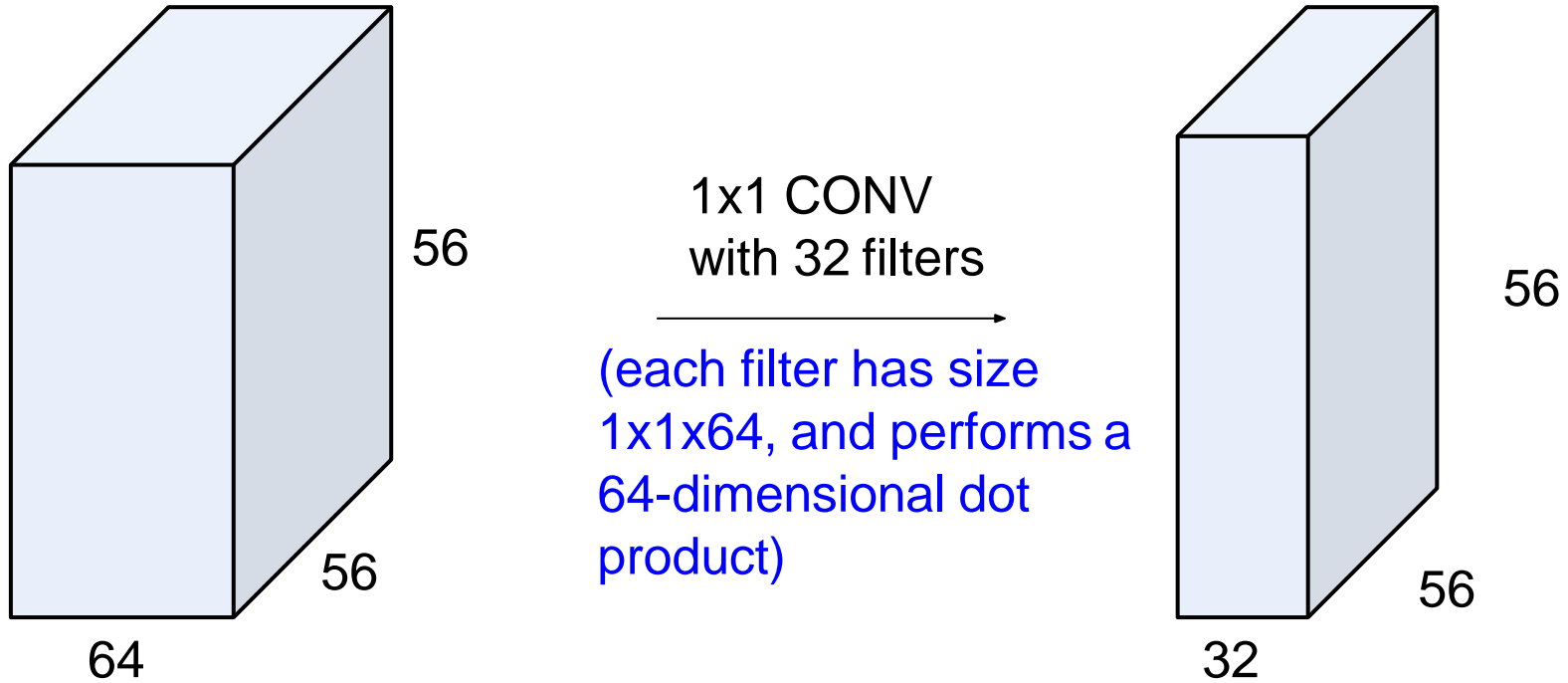
In general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

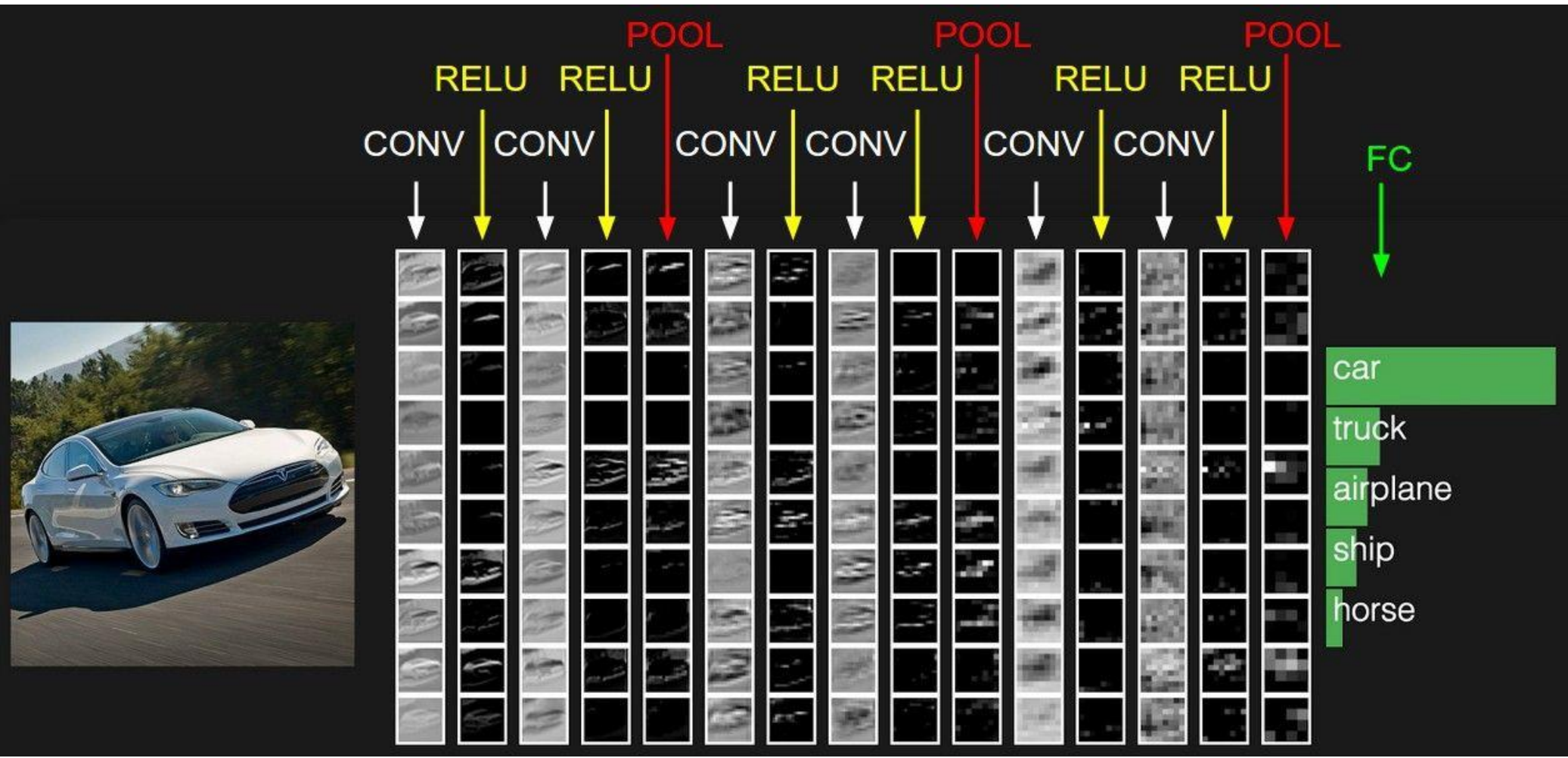
e.g. $F = 3$ => zero pad with 1

$F = 5$ => zero pad with 2

$F = 7$ => zero pad with 3

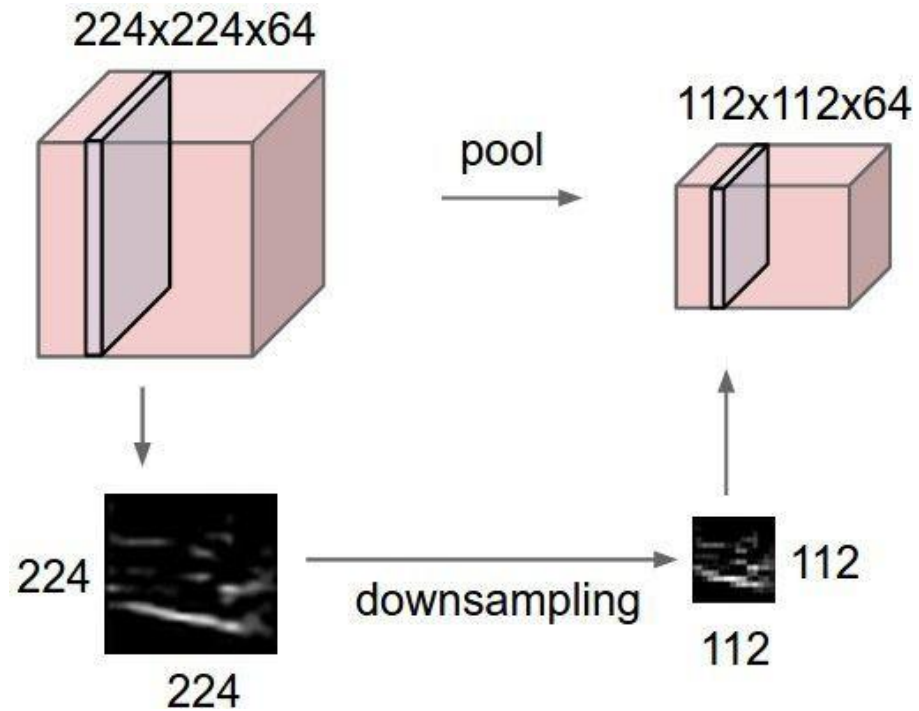
1x1 convolution layers



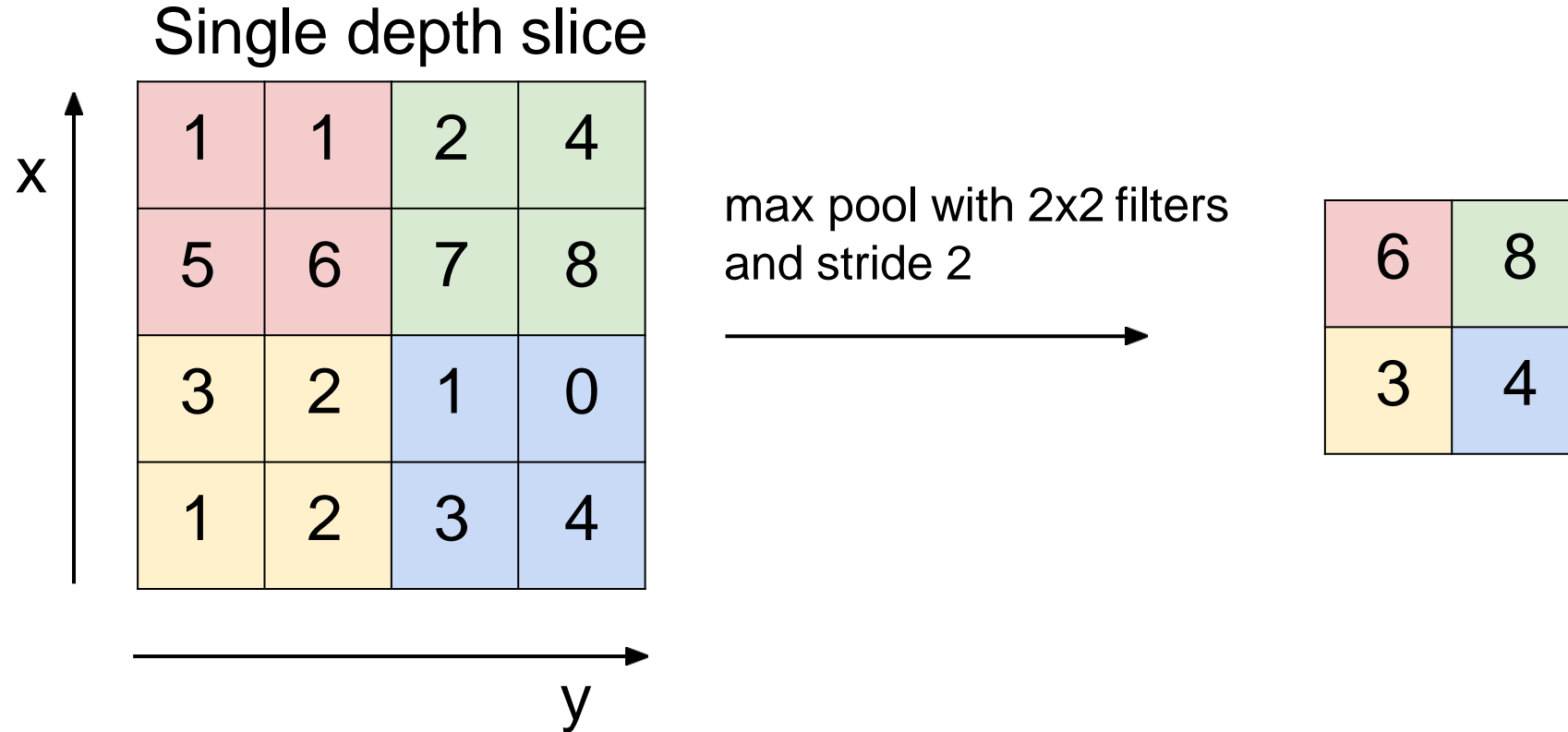


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

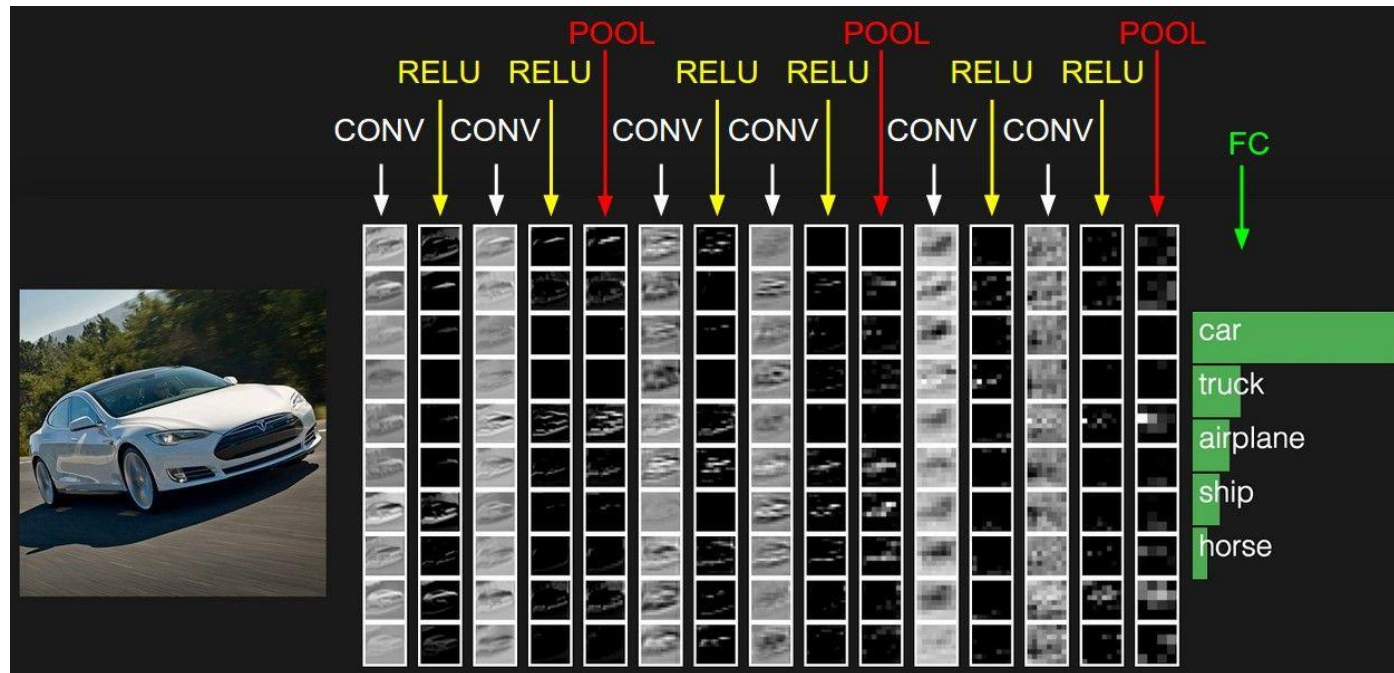


MAX Pooling

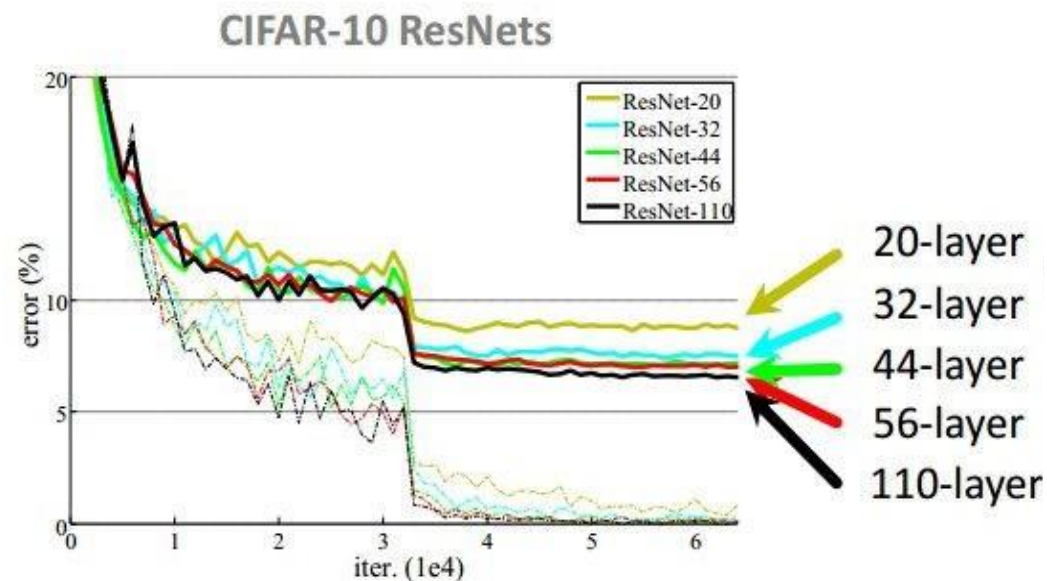
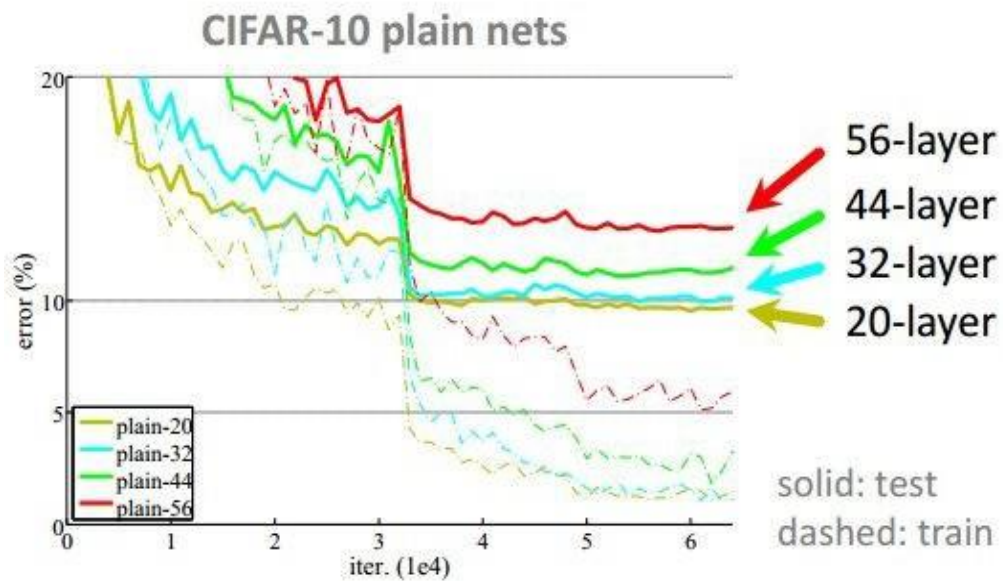


Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

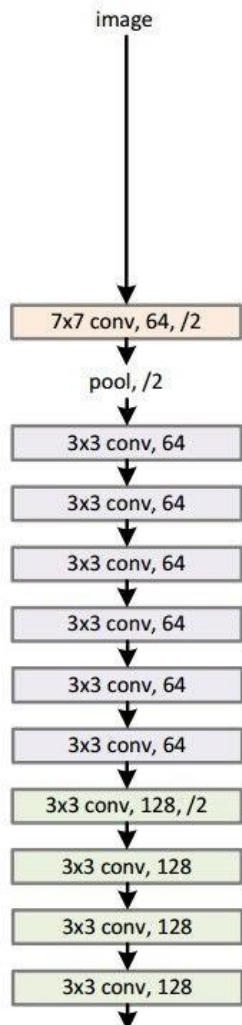


Residual Networks

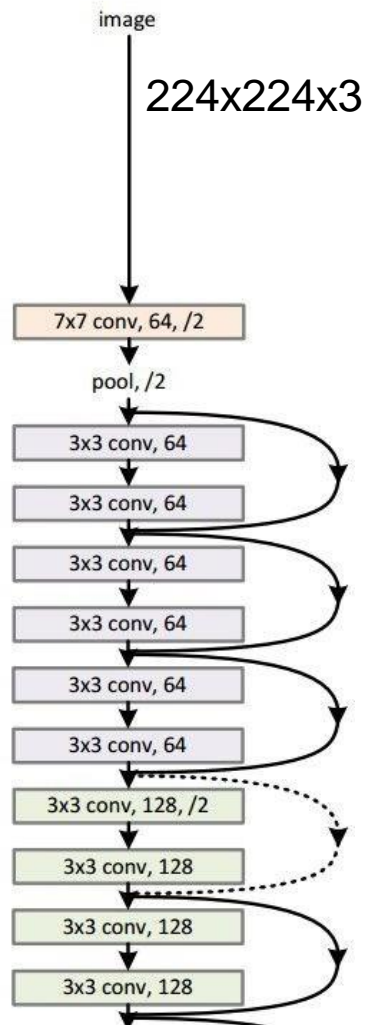


Residual Networks

34-layer plain

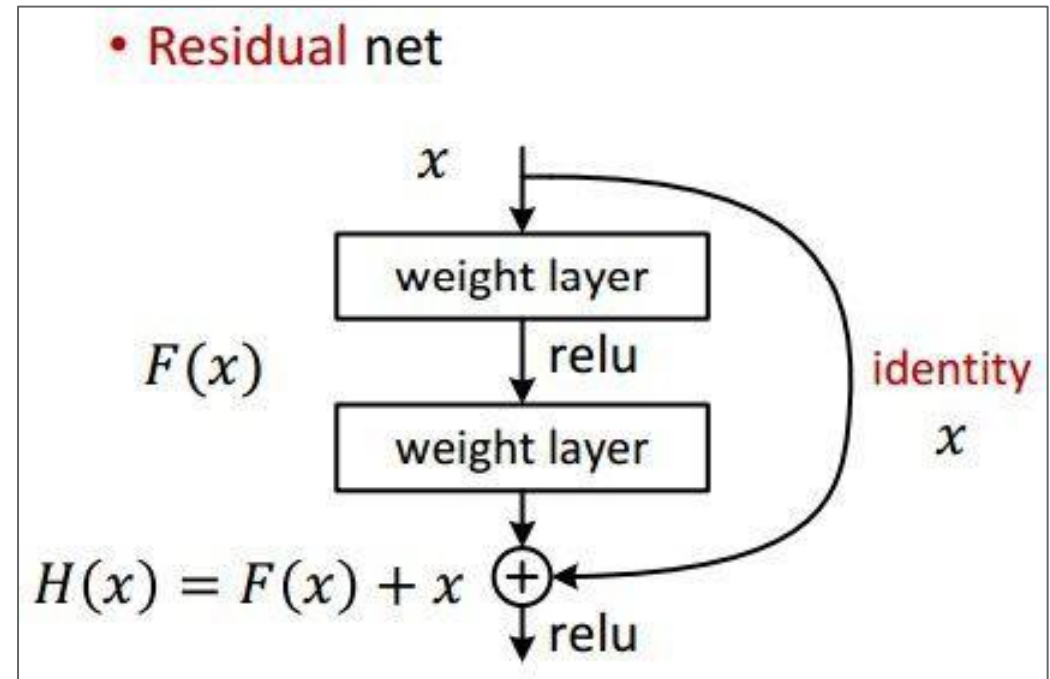
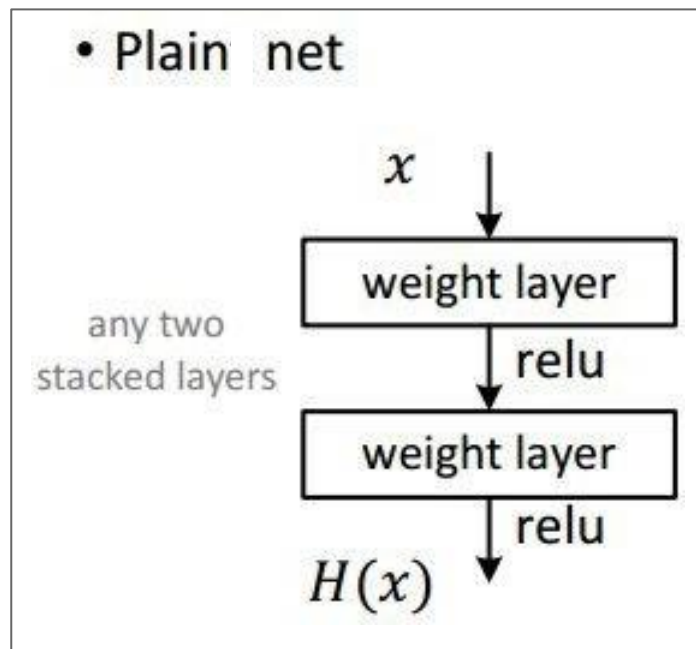


34-layer residual



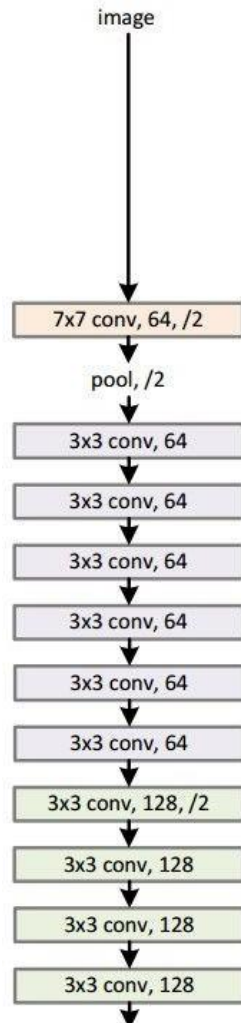
[He et al., 2015]

Residual Networks

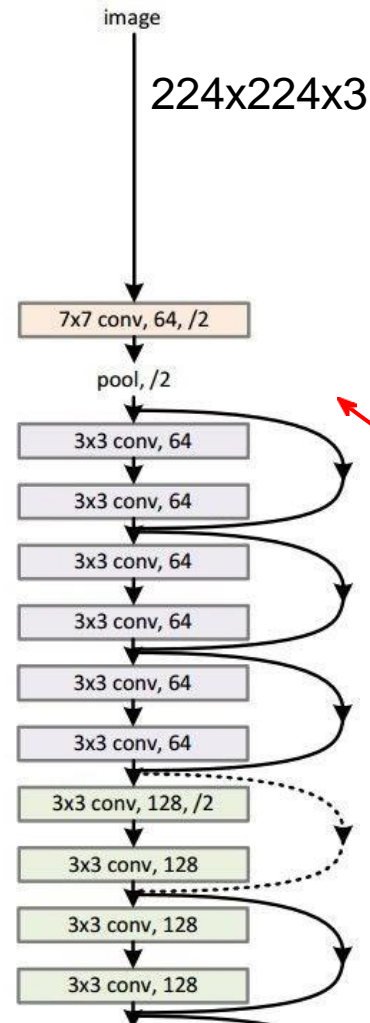


Residual Networks

34-layer plain



34-layer residual

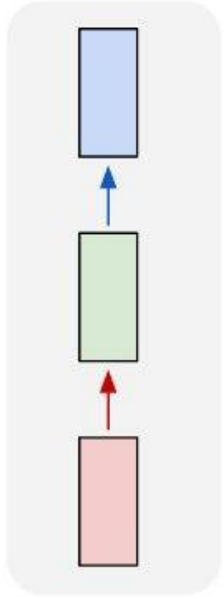


During back-prop, gradient is flows through layers without vanishing

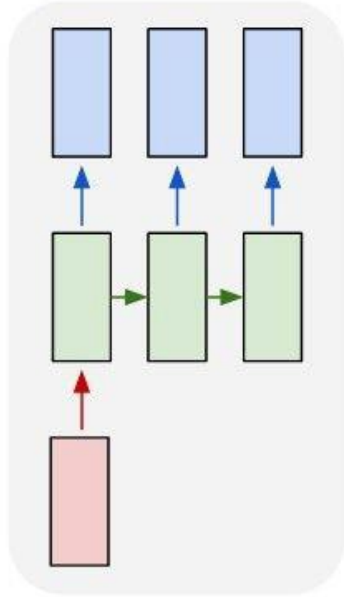
Recurrent Neural Networks

Recurrent Networks offer a lot of flexibility:

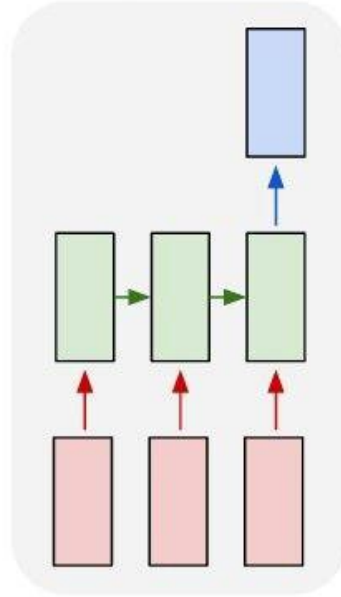
one to one



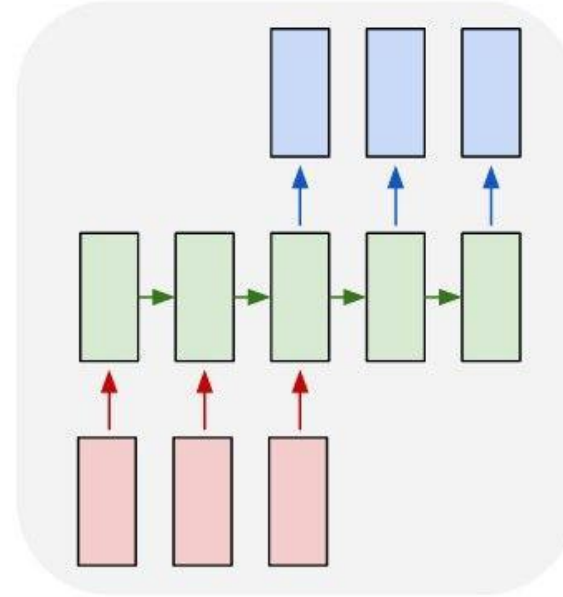
one to many



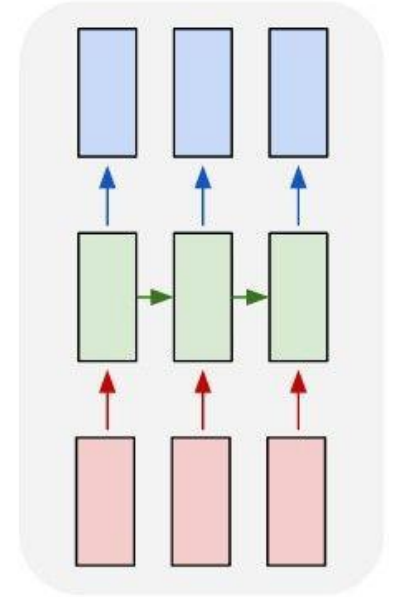
many to one



many to many



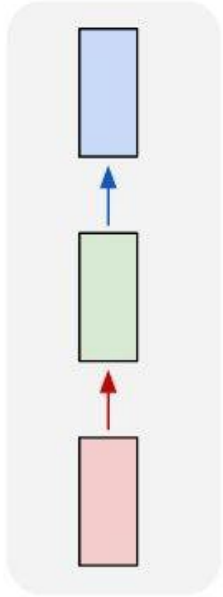
many to many



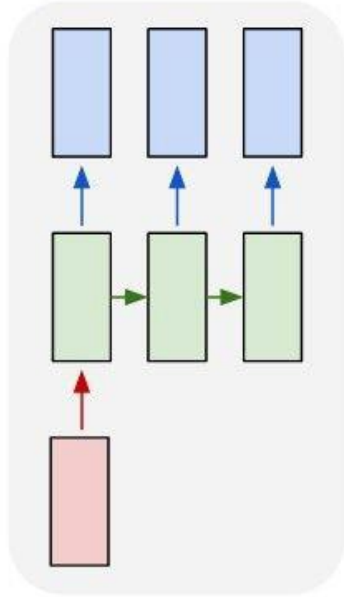
↙ **Vanilla Neural Networks**

Recurrent Networks offer a lot of flexibility:

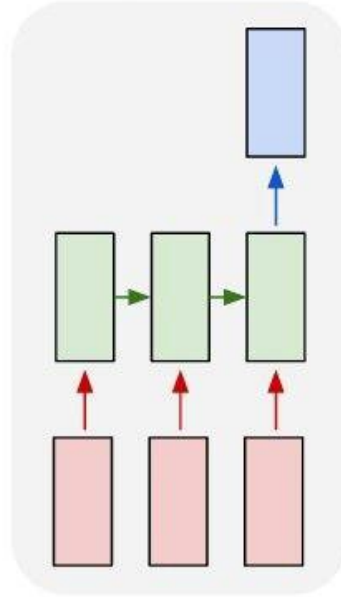
one to one



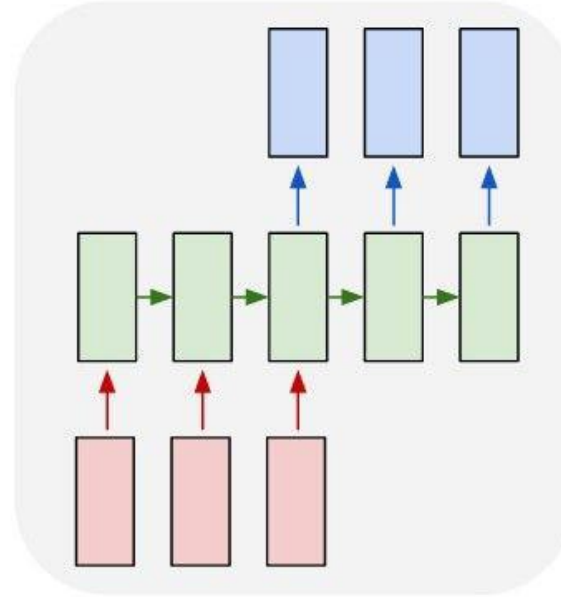
one to many



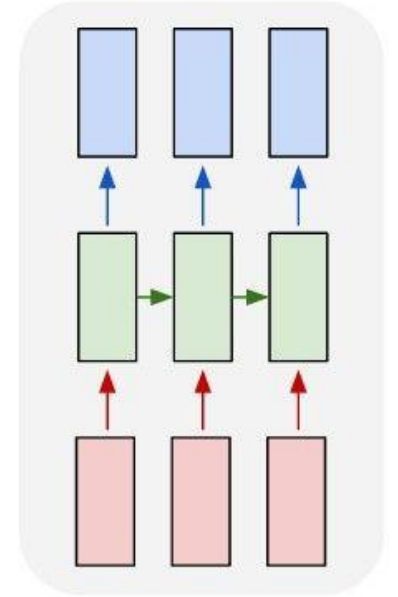
many to one



many to many



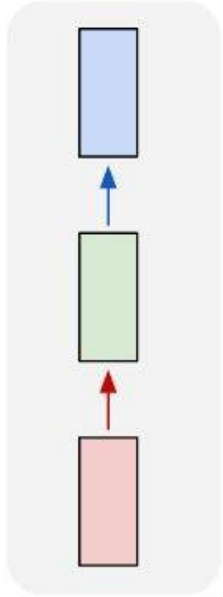
many to many



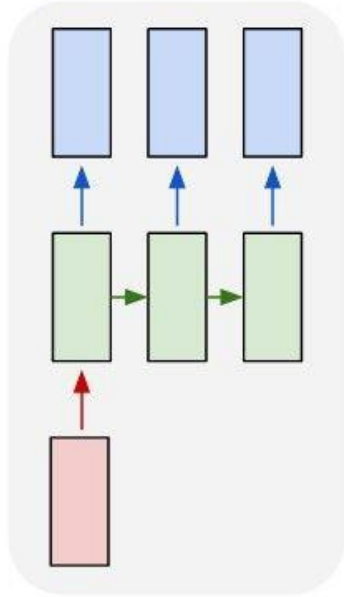
↖ e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:

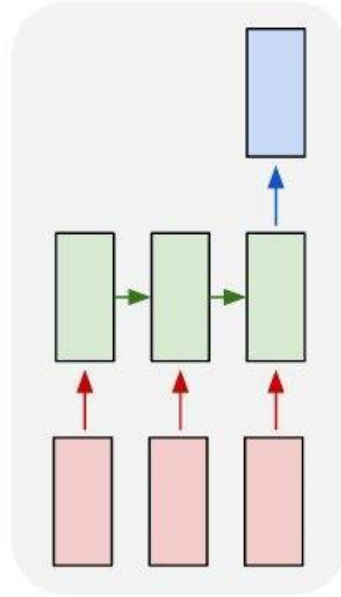
one to one



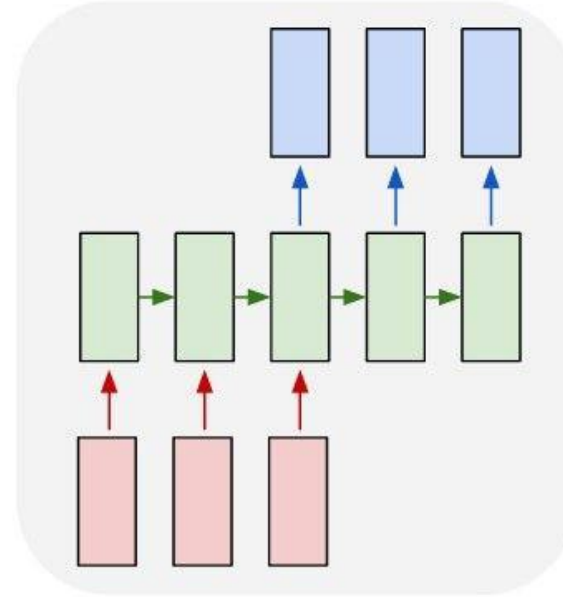
one to many



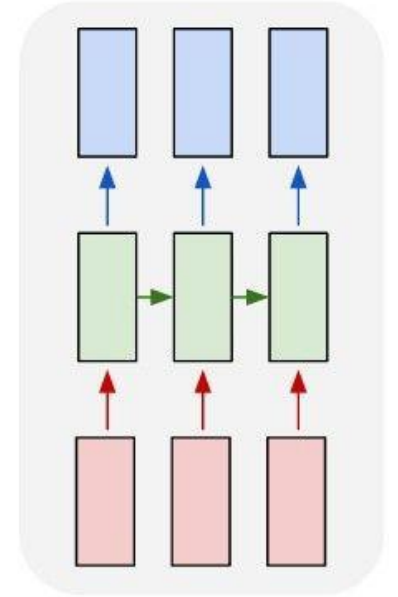
many to one



many to many



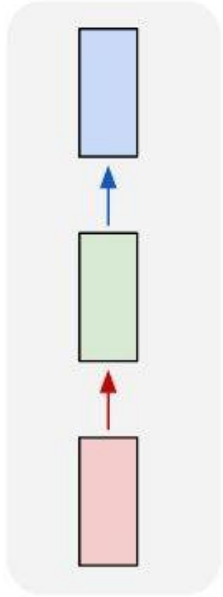
many to many



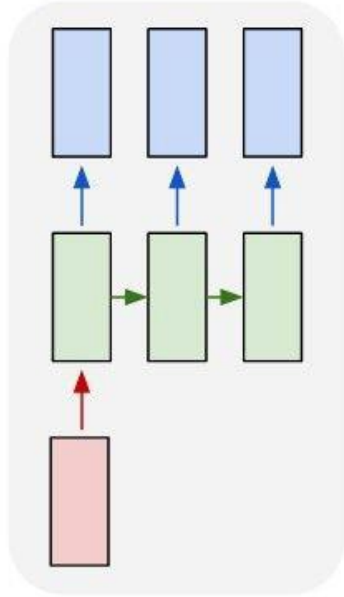
↖ e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:

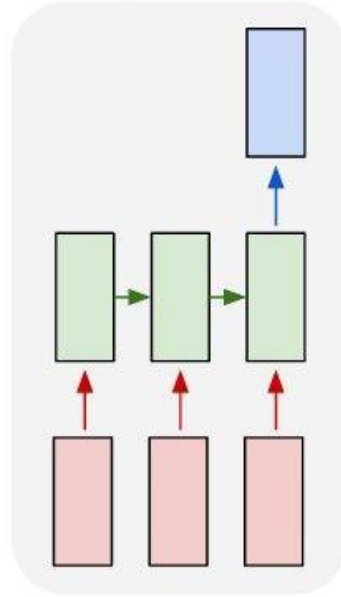
one to one



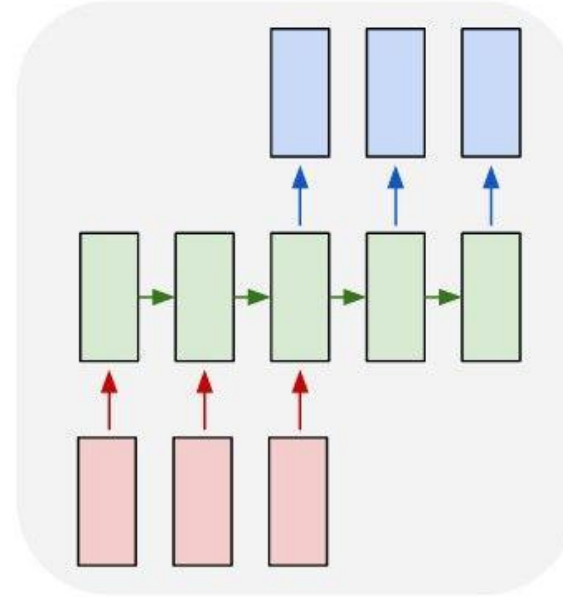
one to many



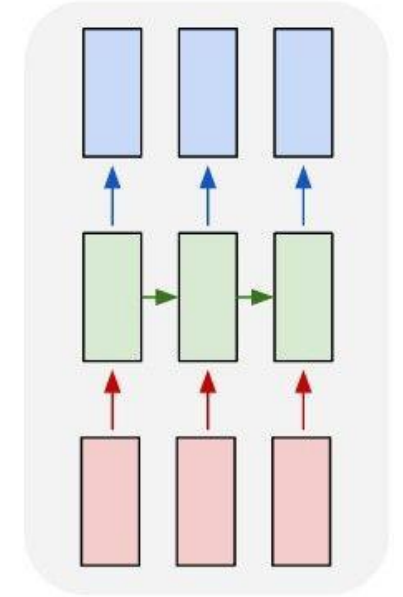
many to one



many to many



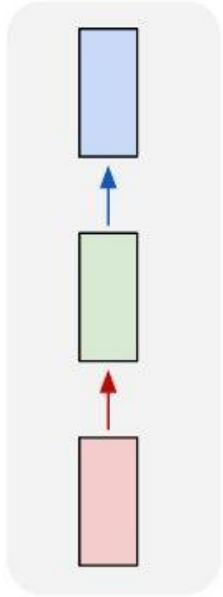
many to many



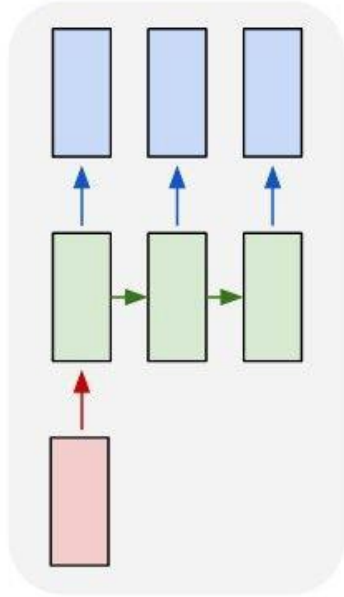
↖ e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

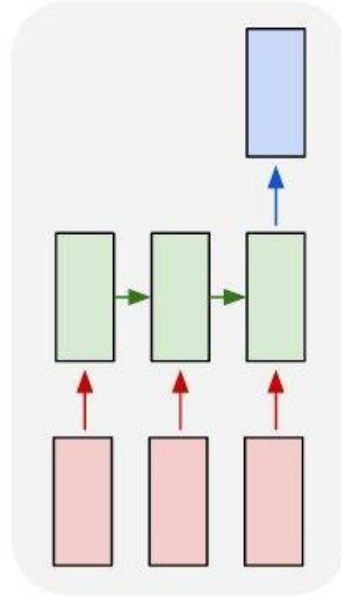
one to one



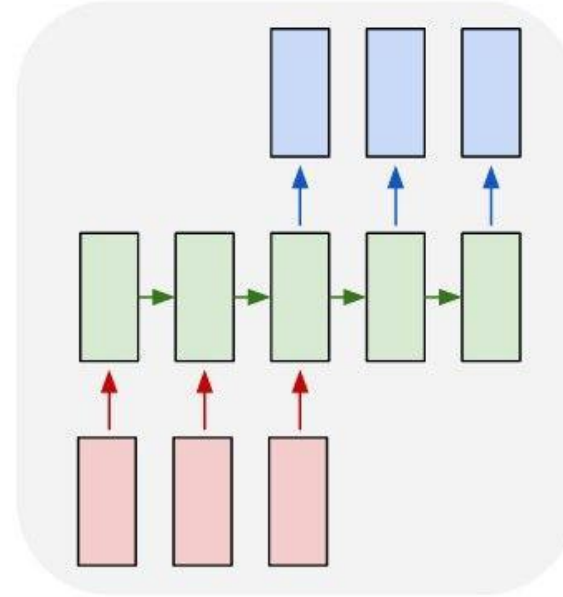
one to many



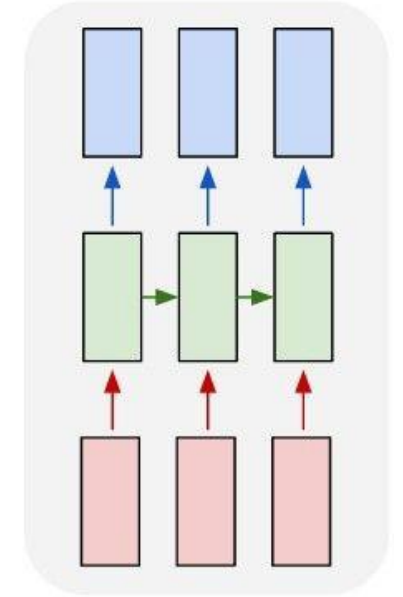
many to one



many to many



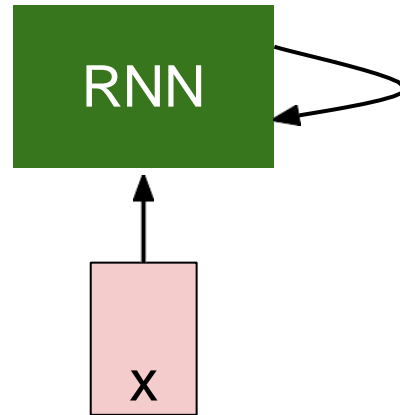
many to many



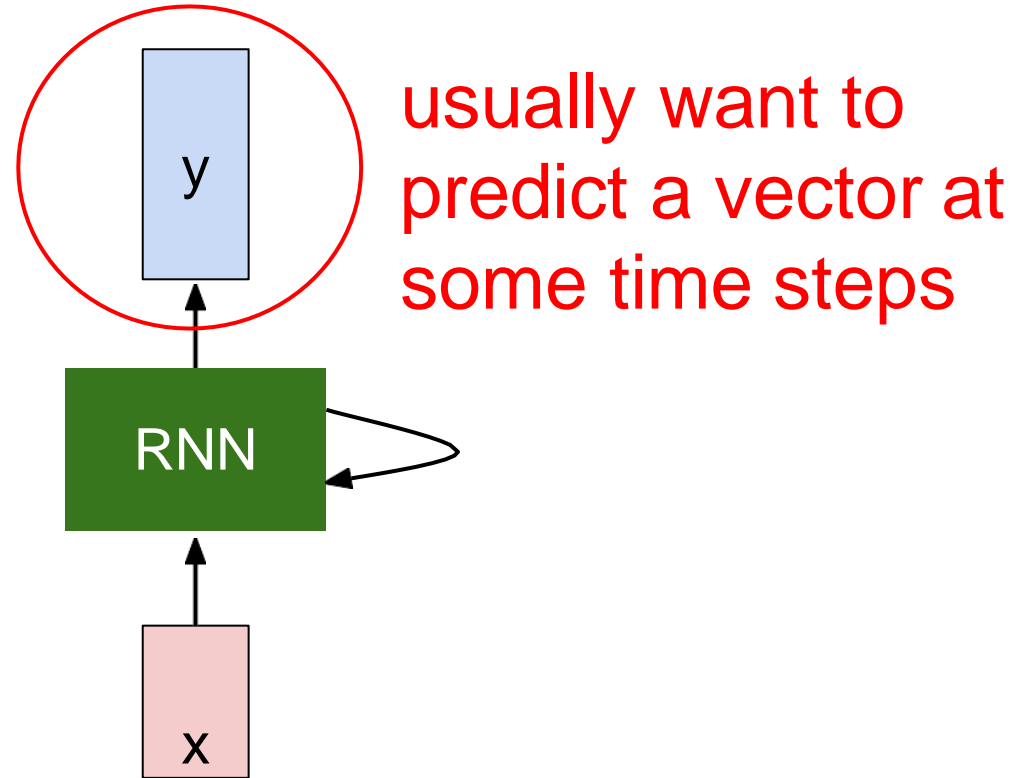
e.g. **Video classification on frame level**



Recurrent Neural Network



Recurrent Neural Network



Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

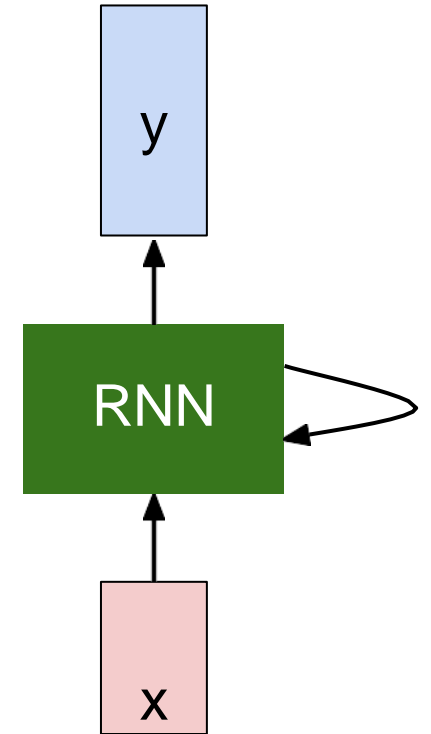
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step

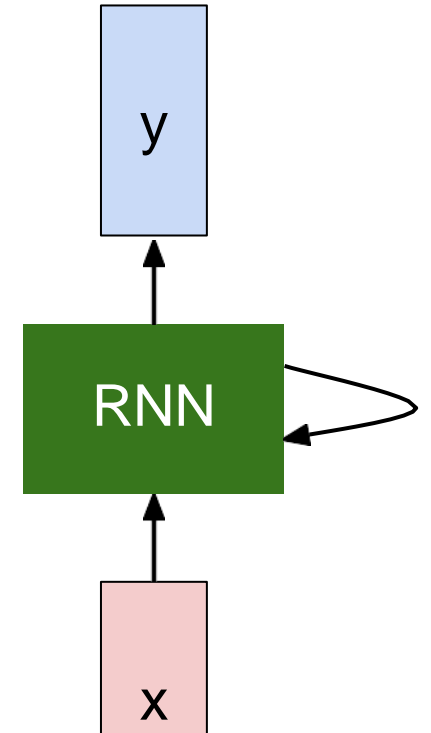


Recurrent Neural Network

We can process a sequence of vectors \mathbf{x} by applying a recurrence formula at every time step:

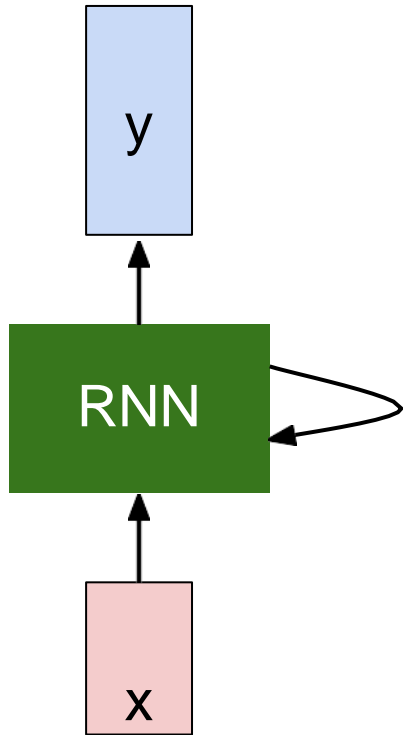
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

The state consists of a single “*hidden*” vector \mathbf{h} :



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



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

$$y_t = W_{hy}h_t$$

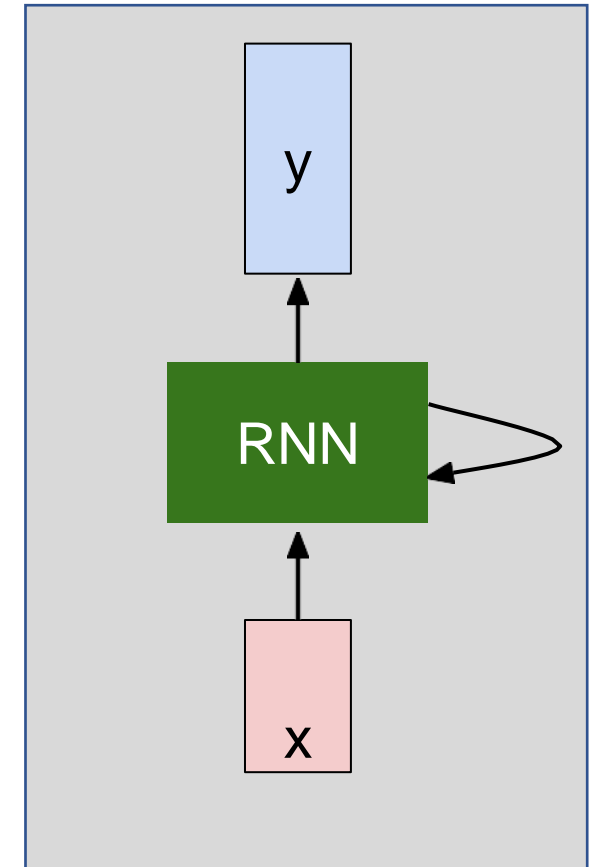
Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

Objective:

Predict the next character given the previous characters



One-hot (one-of-n) encoding

Example: letters. $|V| = 30$

$$\text{'a'} : \mathbf{x}^T = [1, 0, 0, \dots, 0]$$

$$\text{'b'} : \mathbf{x}^T = [0, 1, 0, \dots, 0]$$

$$\text{'c'} : \mathbf{x}^T = [0, 0, 1, \dots, 0]$$

.

.

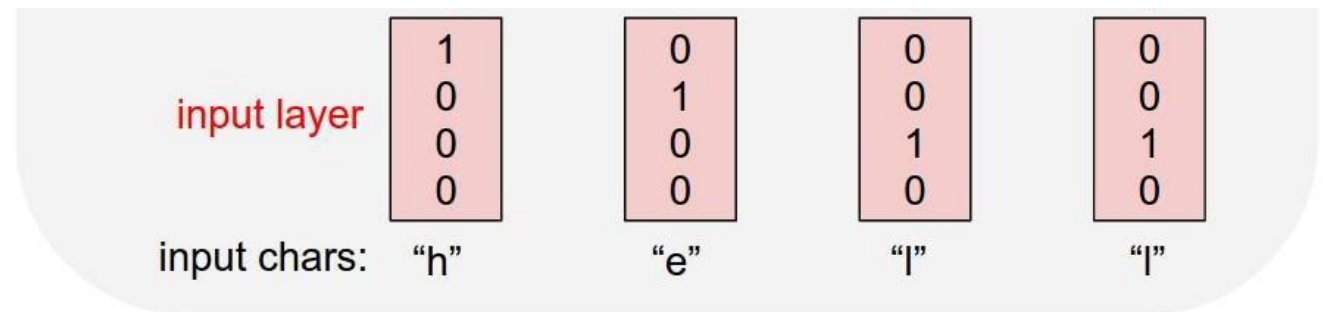
.

$$\text{'.'} : \mathbf{x}^T = [0, 0, 0, \dots, 1]$$

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Objective:

Predict the next character given the previous characters

Character-level language model example

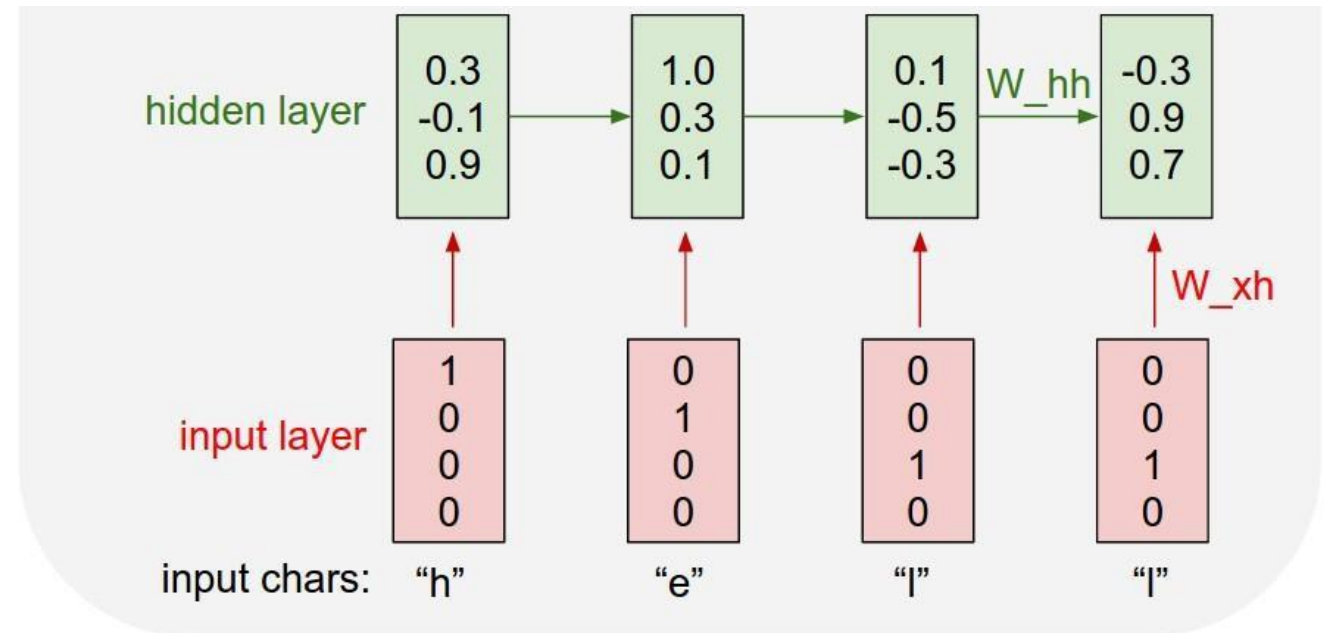
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

Objective:

Predict the next character given the previous characters

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



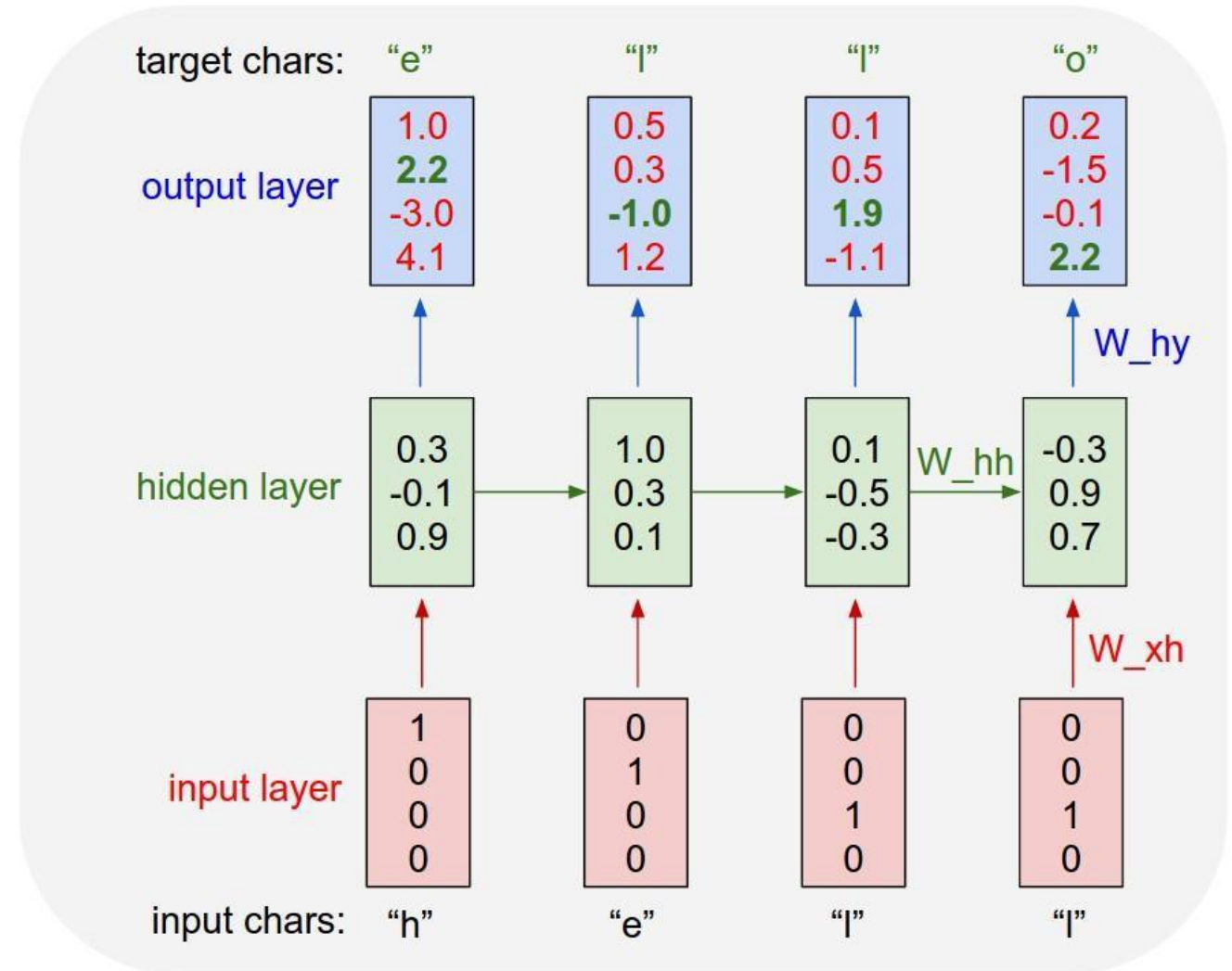
Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

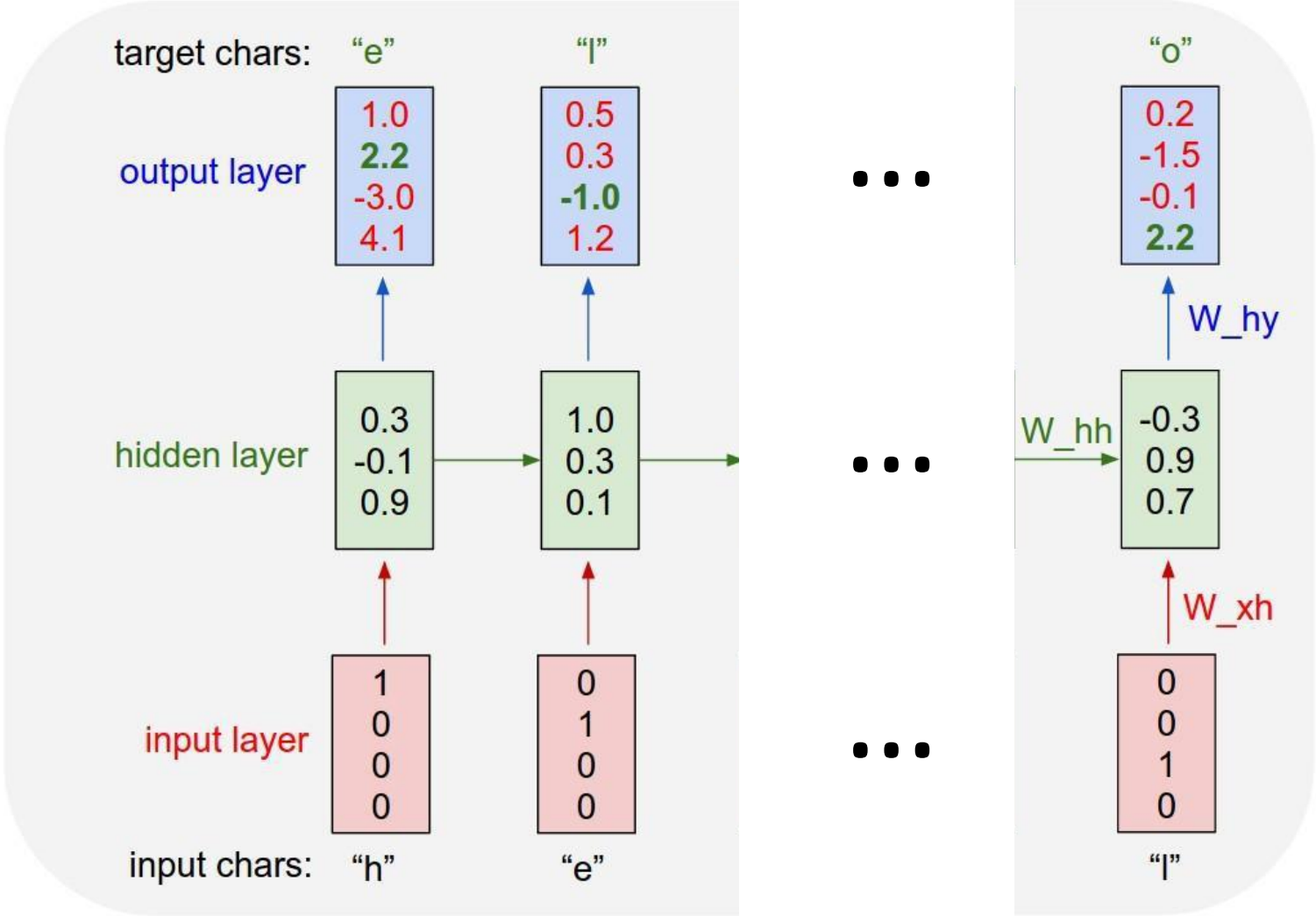
Objective:

Predict the next character given the previous characters

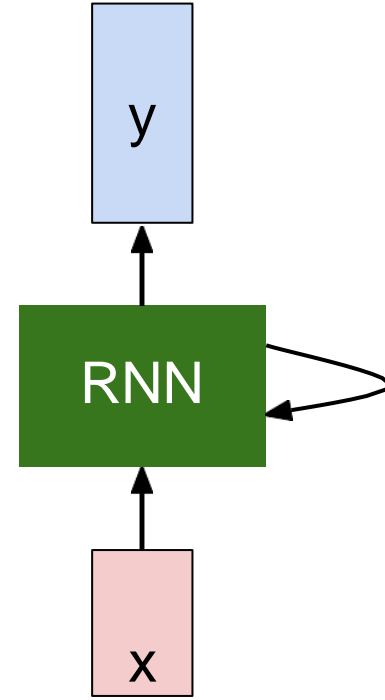


Varying length input

Forward and backward passes are conducted on consequent subsequences iteratively



"p class="clear" Products: Laser Printers. The fundamental everyday requirement for mono and colour laser printing throughout today's offices is perfectly met with the extensive Epson laser printer range. The latest AcuLaser printer range offers users exceptionally Epson AcuLaser C1900. Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. more Where to Buy Support Epson AcuLaser C3000. The fastest colour laser printer in its class. The perfect printer for small businesses and work groups, the Epson AcuLaser C3000 prints high volumes in black and white and vibrant colour, at high speed and with low running costs. more Where to Buy High quality resolution: 2400dpi RIT* Large paper capacity: 600 sheets, expandable up to 1,600 sheets. Compatible Windows and Mac. High speed USB and EpsonNet 10/100 Base Tx Ethernet interfaces as standard** * Epson AcuLaser Resolution Improvement Technology **EpsonNet 10/100 Base Tx Ethernet standard with Epson AcuLaser C3000 model only. AcuLaser C3000: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing as standard, AcuLaser C3000i: 64MB Memory, 100 sheet MP Tray, 500 sheet cassette, Duplex printing, 10/100BaseTX Ethernet Interface. Networked compact colour laser printer for professional enterprises. Businesses have been denied simple and affordable colour laser printing for far too long. The traditionally high costs and poor speeds of colour lasers has left many offices looking a bit, well, grey. But not any more: with the Epson AcuLaser C1900, Epson brings both colour and monochrome laser printing together at a black and white price. Key Features cost effective mono printing for day to day business needs and vivid versatile colour when required. search Search Epson UK Epson AcuLaser C500. Outstanding professional colour printing for business. Add colour to your business with the Epson AcuLaser C500 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high performance black and white production. more Where to Buy Support As cost efficient to run as a mono-only laser printer. Paper capacity of 700 sheets from two media sources. Easy to operate with advanced printer driver. Memory expandable from 32MB to 1024MB. Pre-configured models available with Wireless 802.11b, Adobe® PostScript® Level 3™ and two-sided printing. The AcuLaser C1900 is available in 5 configurations: - AcuLaser C1900S: with 32MB, 200 Sheet MP Tray, 10/100BaseTX Networking - AcuLaser C1900: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking Support Epson AcuLaser C4100. High performance colour lasers for all your business printing needs. The Epson AcuLaser C4100 provides businesses with a high performance colour and monochrome printing solution. It adds crucial colour to your business, while producing high quality monochrome output at lower costs than many monochrome-only printers, and it's just as easy to operate. So now there's no reason to buy two printers, because perfect monochrome and colour solutions are available in one. more Where to Buy Support Epson AcuLaser C2500. Professional high performance A3W colour laser printer. Epson AcuLaser C8600 is the perfect professional printing solution for users who require exceptional quality colour and mono output on a range of media formats from C5 up to A3W in size. The Epson AcuLaser C8600 is able to achieve superb print quality by utilising a combination of Epson's exclusive AcuLaser Colour Laser Technologies. more Where to Buy Support - AcuLaser C1800PS: with Adobe® PostScript® 3™, 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1800: with Duplex unit (two sided printing) 96MB, 200 Sheet MP Tray, 500 Sheet Cassette, 10/100BaseTX Networking - AcuLaser C1800 WiFi: with 32MB, 200 Sheet MP Tray, 500 Sheet Cassette, Wireless Networking facility. Add colour to your business with the Epson AcuLaser C500 from Epson. Its perfect for the smaller workgroup, being a compact and cost effective laser printing workhorse that offers amazing colour output as well as high Support Epson AcuLaser C4000. High performance colour laser. The Epson AcuLaser C4000 provides businesses with high performance colour and monochrome printing solutions. more Where to Buy Epson AcuLaser C9100. High speed A3 colour laser printer. Why have separate black and white and colour printers when you can have the Epson AcuLaser C9100? Epson has taken the lead in laser technology to deliver a complete high-performance solution for all your colour and mono printing needs. Support EPL-6200L. High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more Where to Buy Support EPL-6200. High performance A4 mono laser professional printers. The Epson EPL-6200 and EPL-6200L are the ideal printing solutions for small to medium workgroups and personal users. They deliver professional performance quickly, easily, reliably and cost-effectively, and are perfect for users who need high levels of laser quality and productivity at a low investment. more performance black and white production. For the first time, you can now bring the power of high quality colour to your documents without suffering the high costs or low speeds traditionally associated with colour



Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
 Admit impediments. Love is not love
Which alters when it alteration finds,
 Or bends with the remover to remove:
O no! it is an ever-fixed mark
 That looks on tempests and is never shaken;
It is the star to every wandering bark,
 Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
 Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
 But bears it out even to the edge of doom.
If this be error and upon me proved,
 I never writ, nor no man ever loved.

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhtnee e
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtkie,aoaenns lng

↓
train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓
train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and offer.

↓
train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftended him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nunes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

RNN:

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

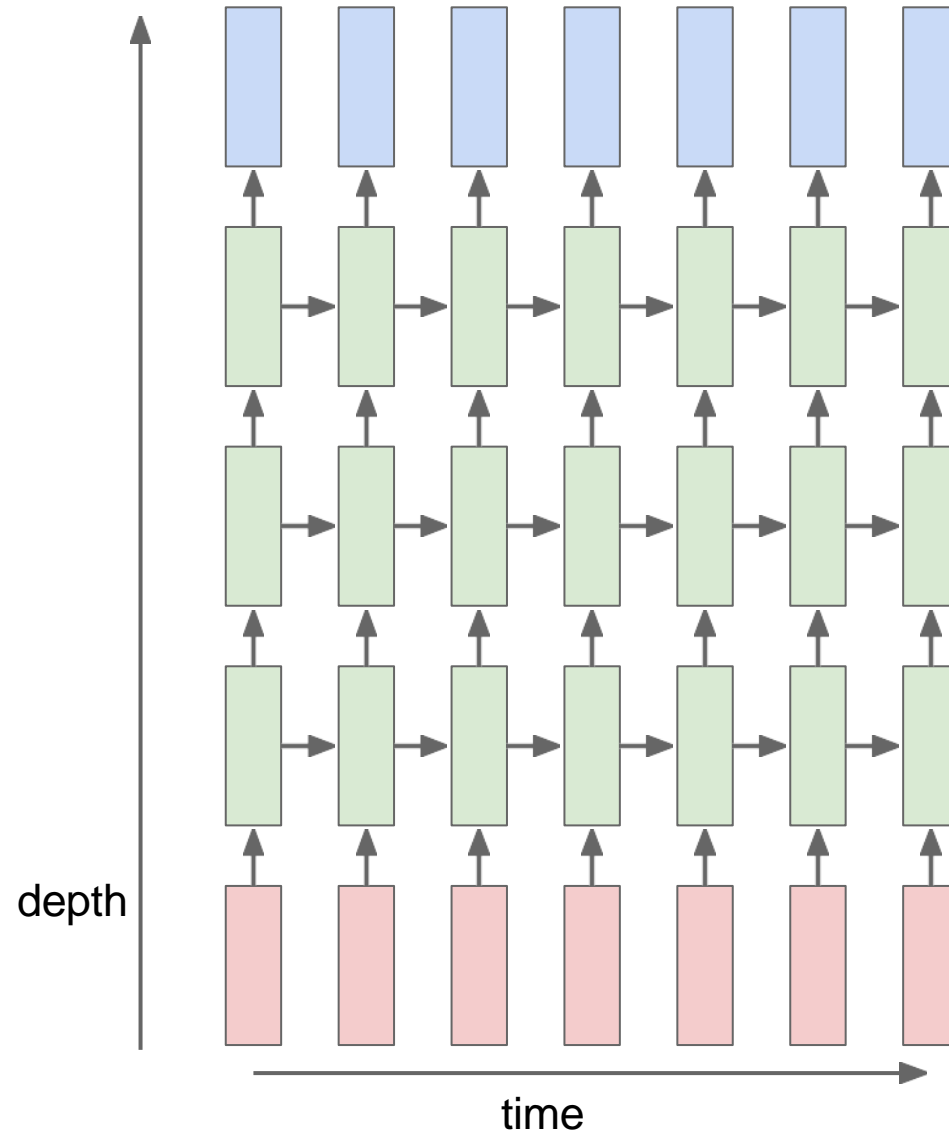
$$h \in \mathbb{R}^n \quad W^l [n \times 2n]$$

A generalization of RNN. At $l=1$:

- $h_t^{l-1} = x_t$
- $W^l = [W_{xh} \ W_{hh}]$

It is equivalent to:

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



RNN:

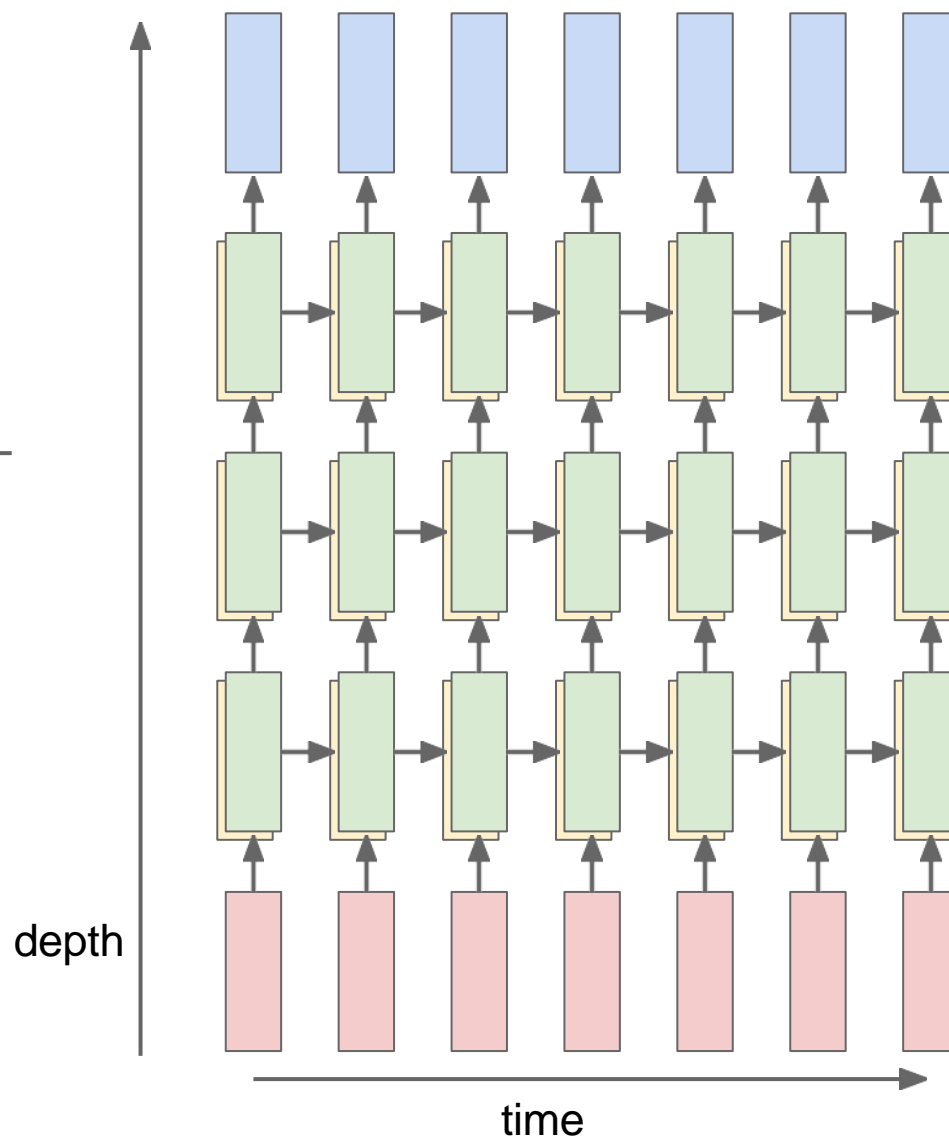
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

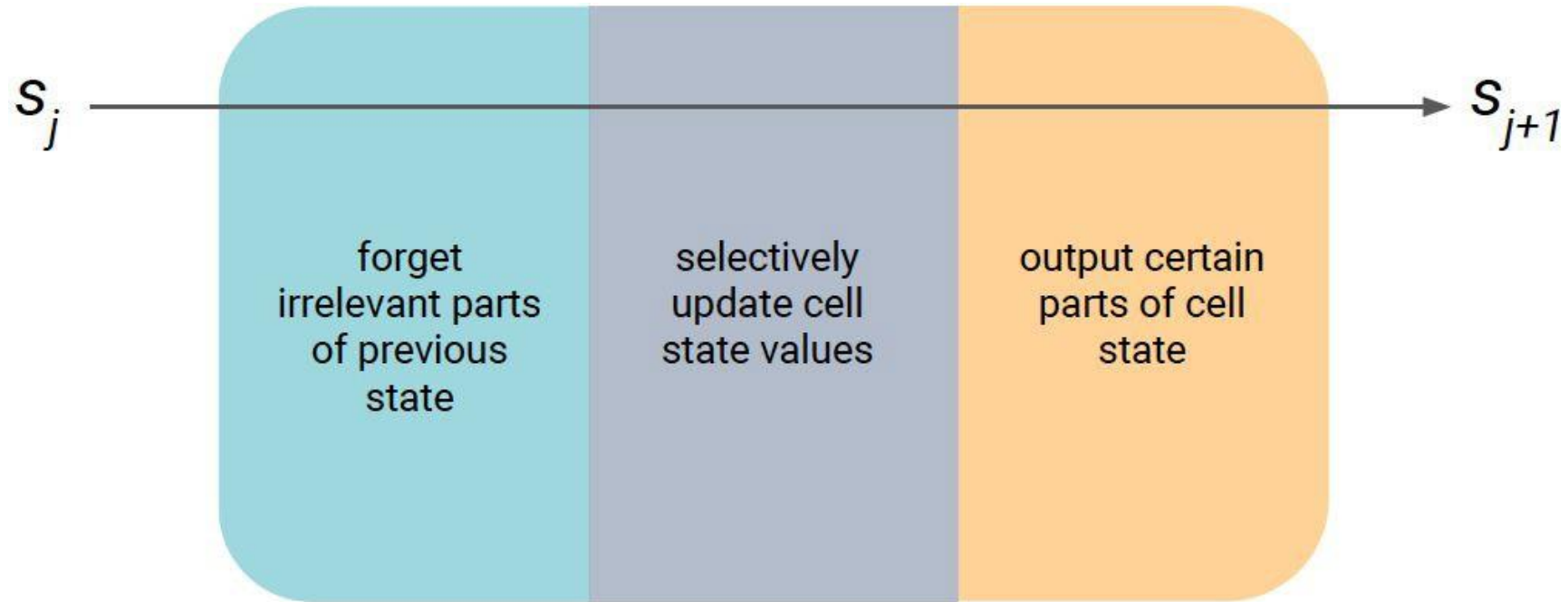
LSTM:

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

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

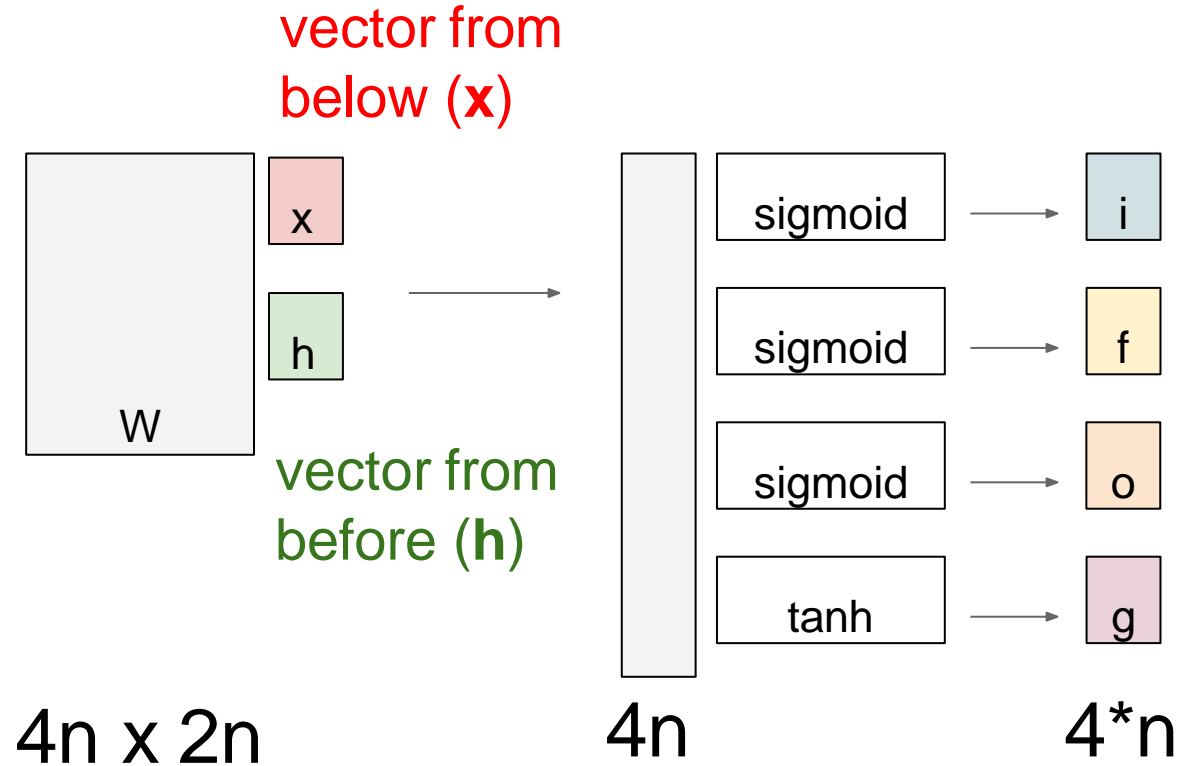


LSTM - main idea



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



- *c*: cell state
- *h*: hidden state (cell output)
- *i*: input gate, weight of acquiring new information
- *f*: forget gate, weight of remembering old information
- *g*: transformed input ($[-1, +1]$)
- *o*: output gate, decides values to be activated based on current memory

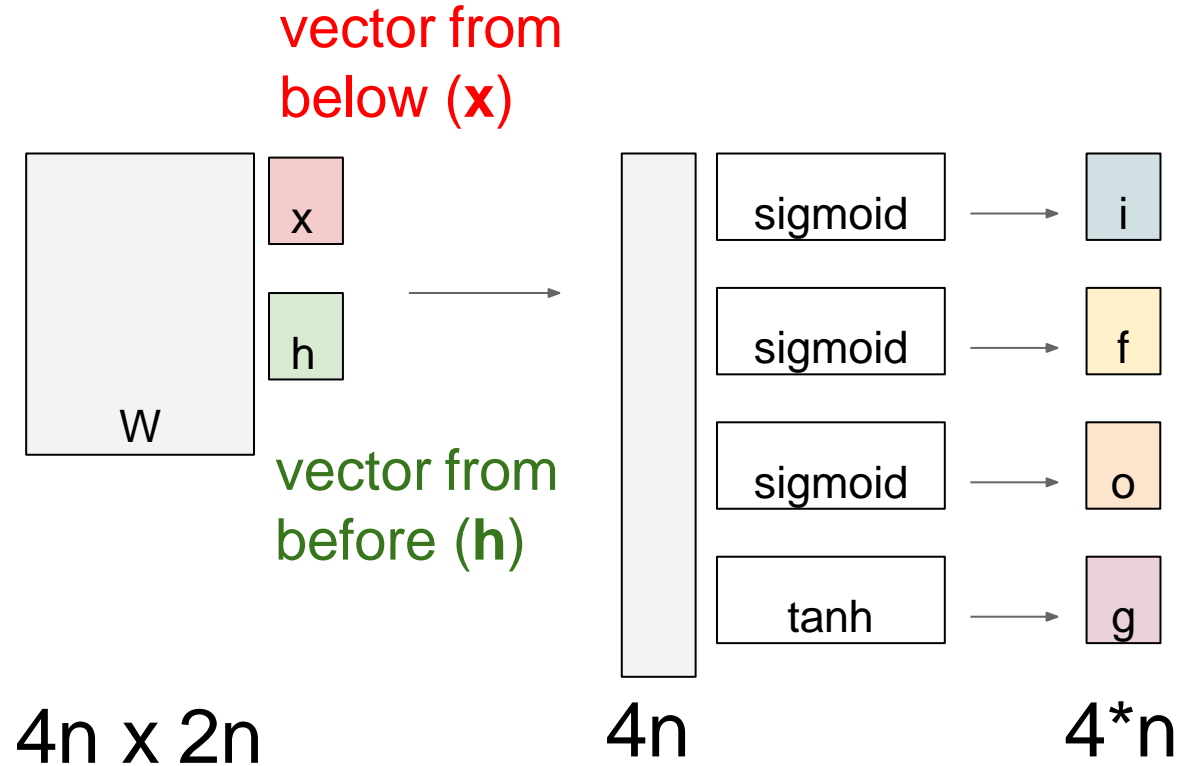
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



f decides the degree of preservation for cell state, by scaling it with a number in $[0, 1]$

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

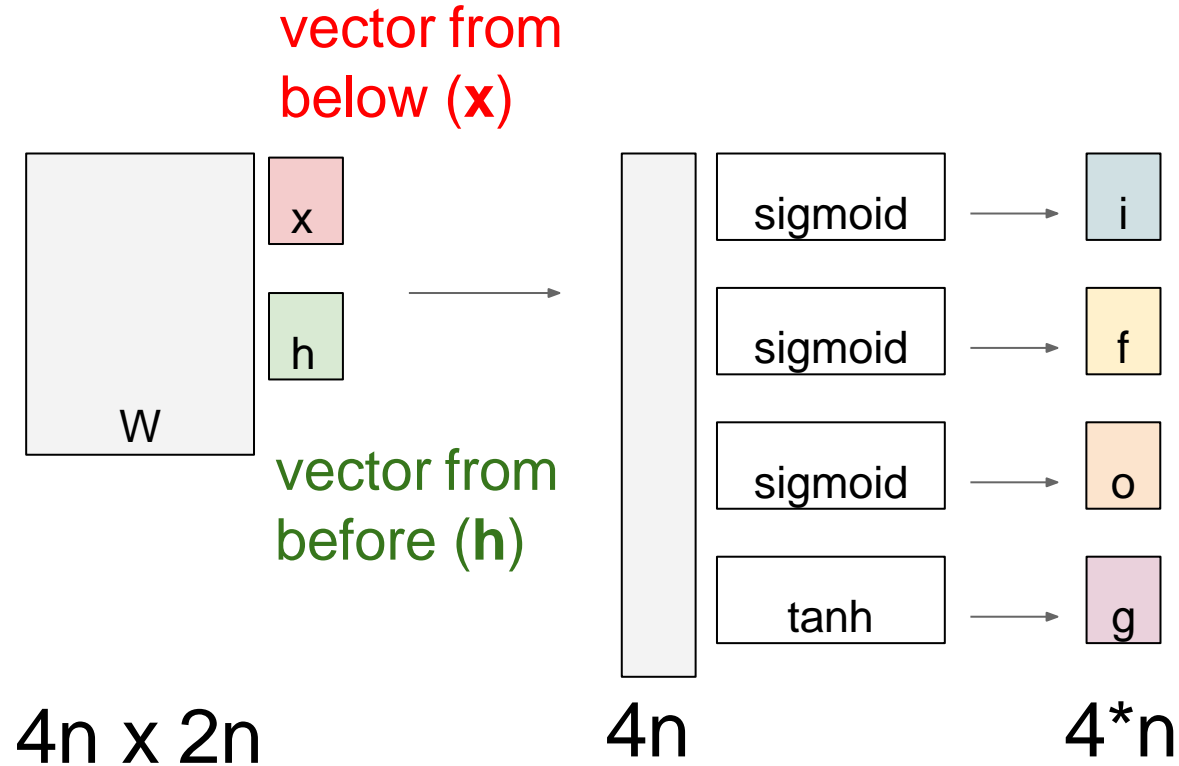
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

g is a transformation
of input / hidden
state



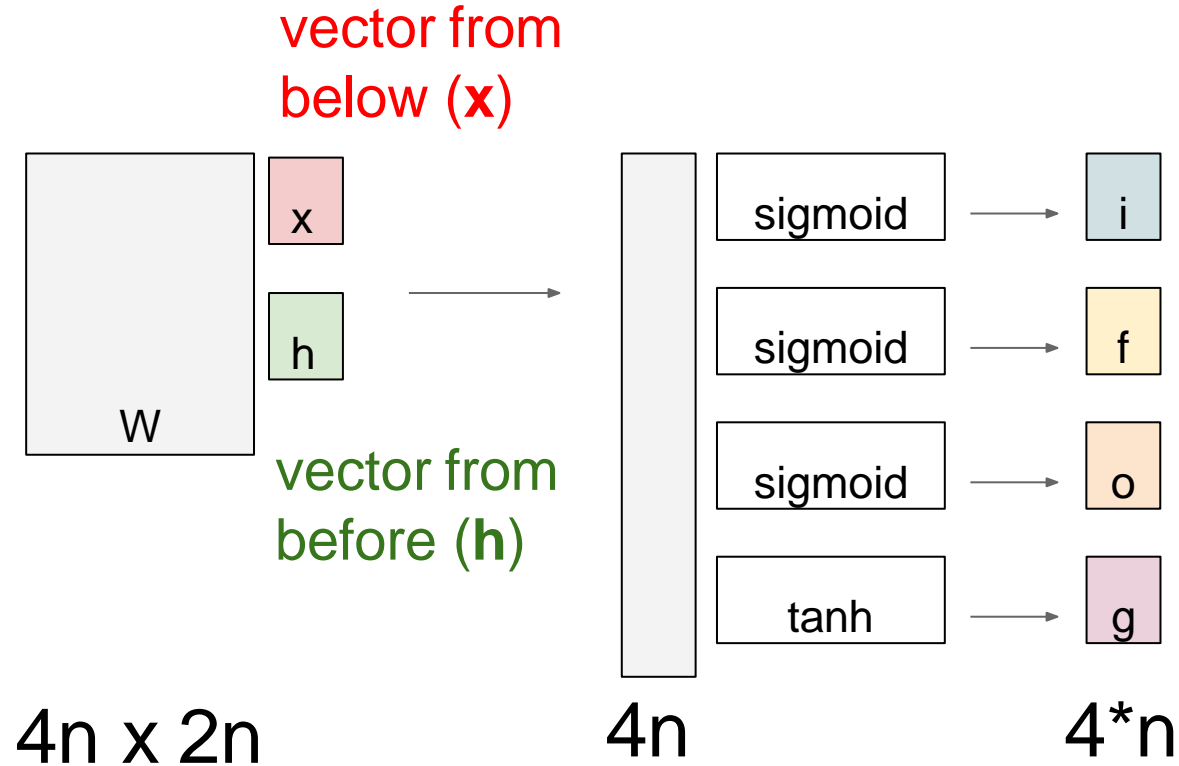
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Add g into the *cell state*,
weighted by i
(weight of acquiring new
information)

Alternative interpretation:
 $i * g$ decouples the "influence
of g " and " g itself".

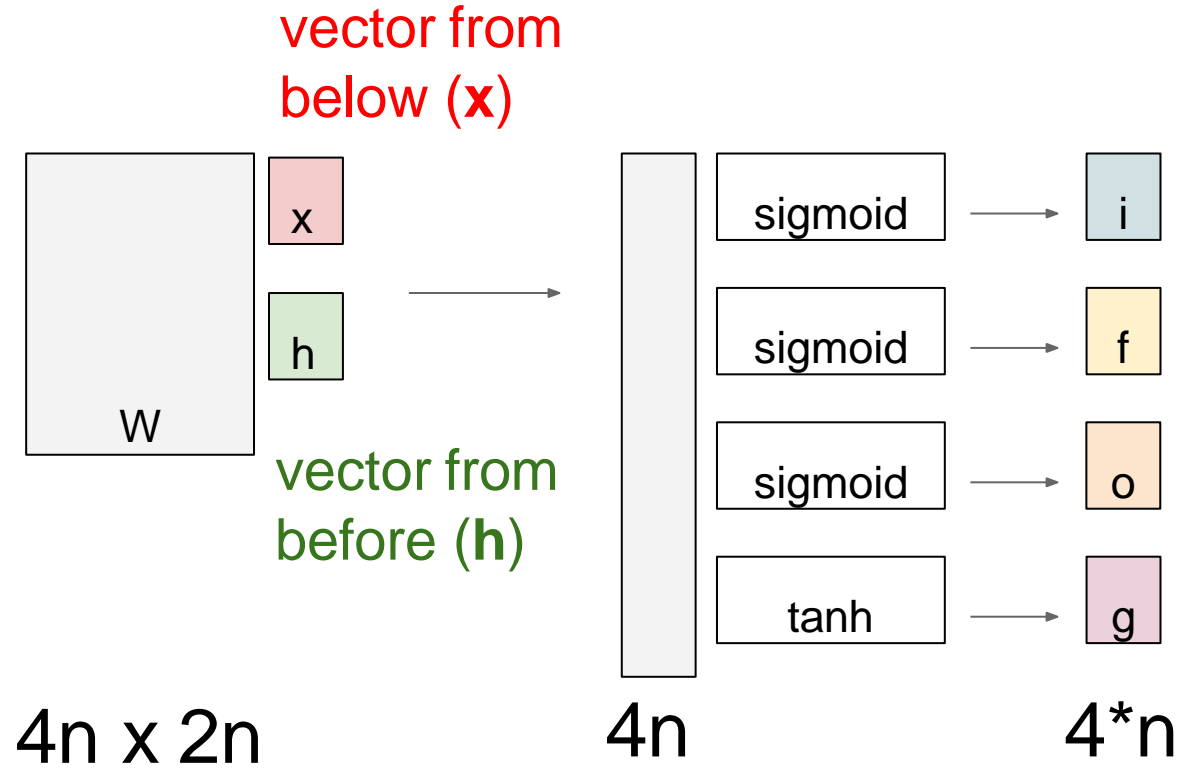
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



New hidden state is a scaled version of $\tanh(\text{cell state})$.

o: output gate, decides values to be activated based on current memory

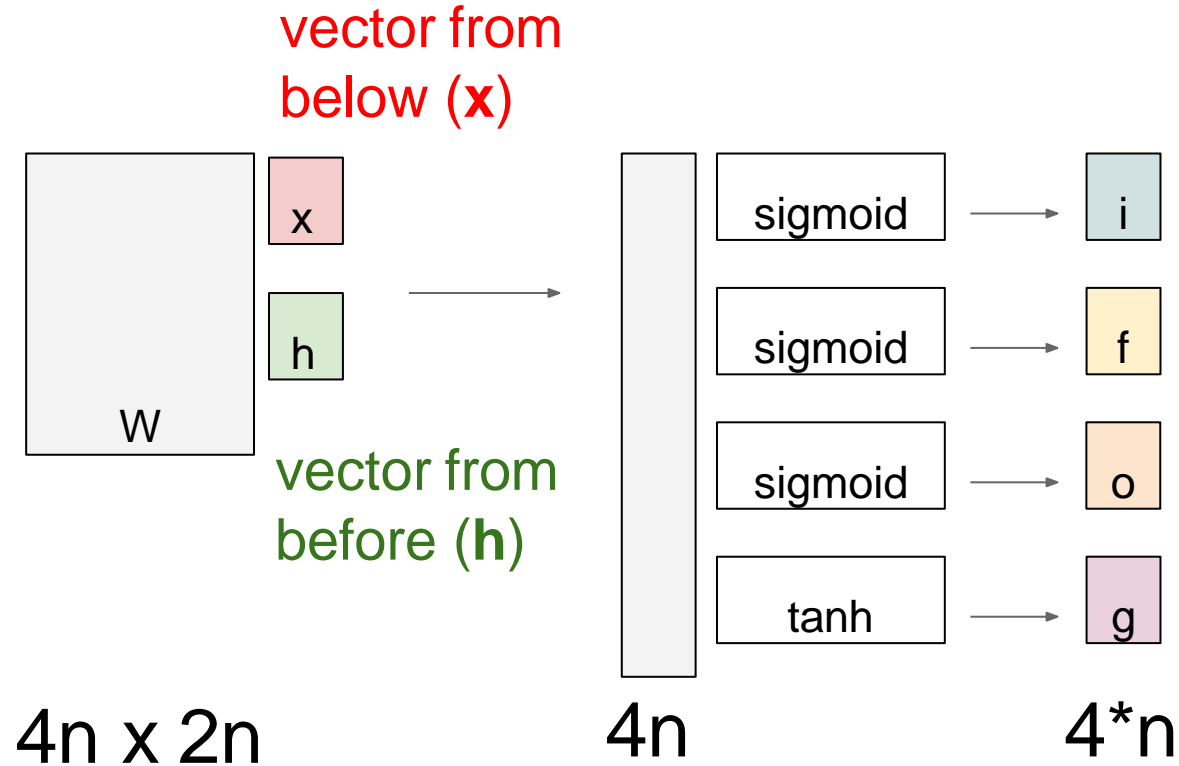
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Q: Why tanh?

A: Not very crucial, sometimes not used

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

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

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

Long Short Term Memory (LSTM)

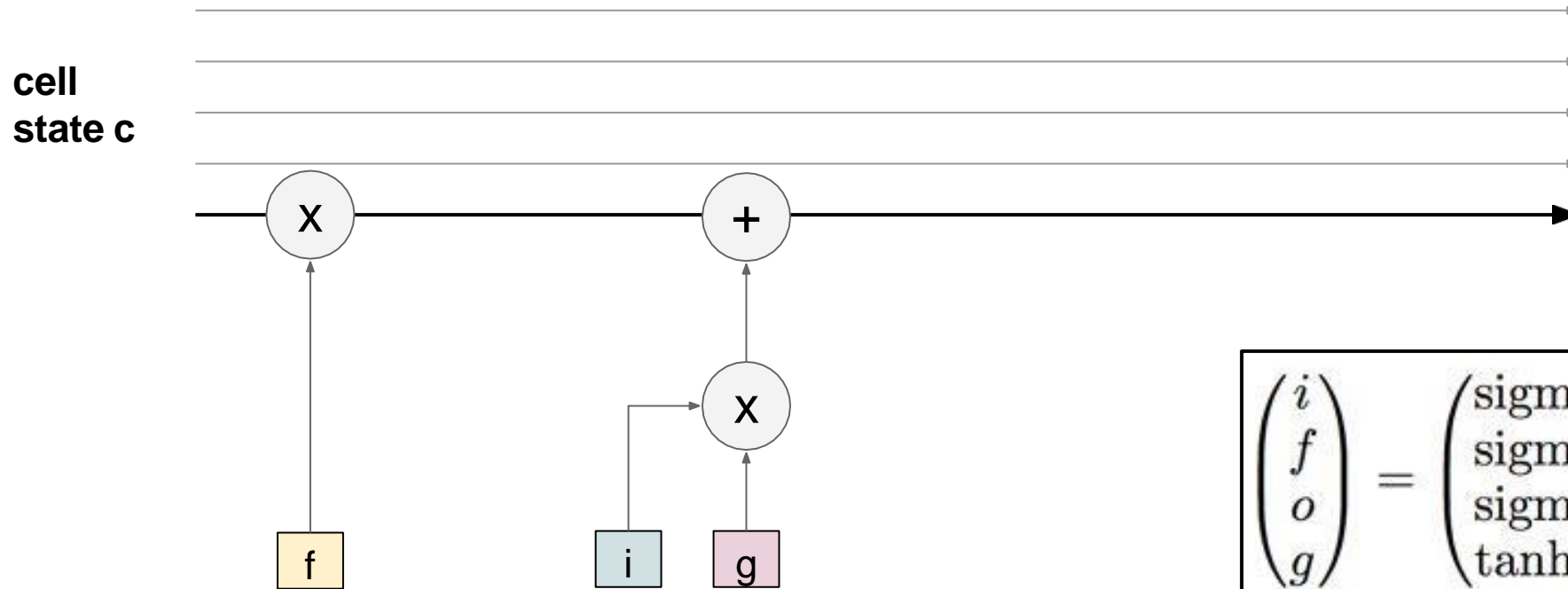
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

Long Short Term Memory (LSTM)

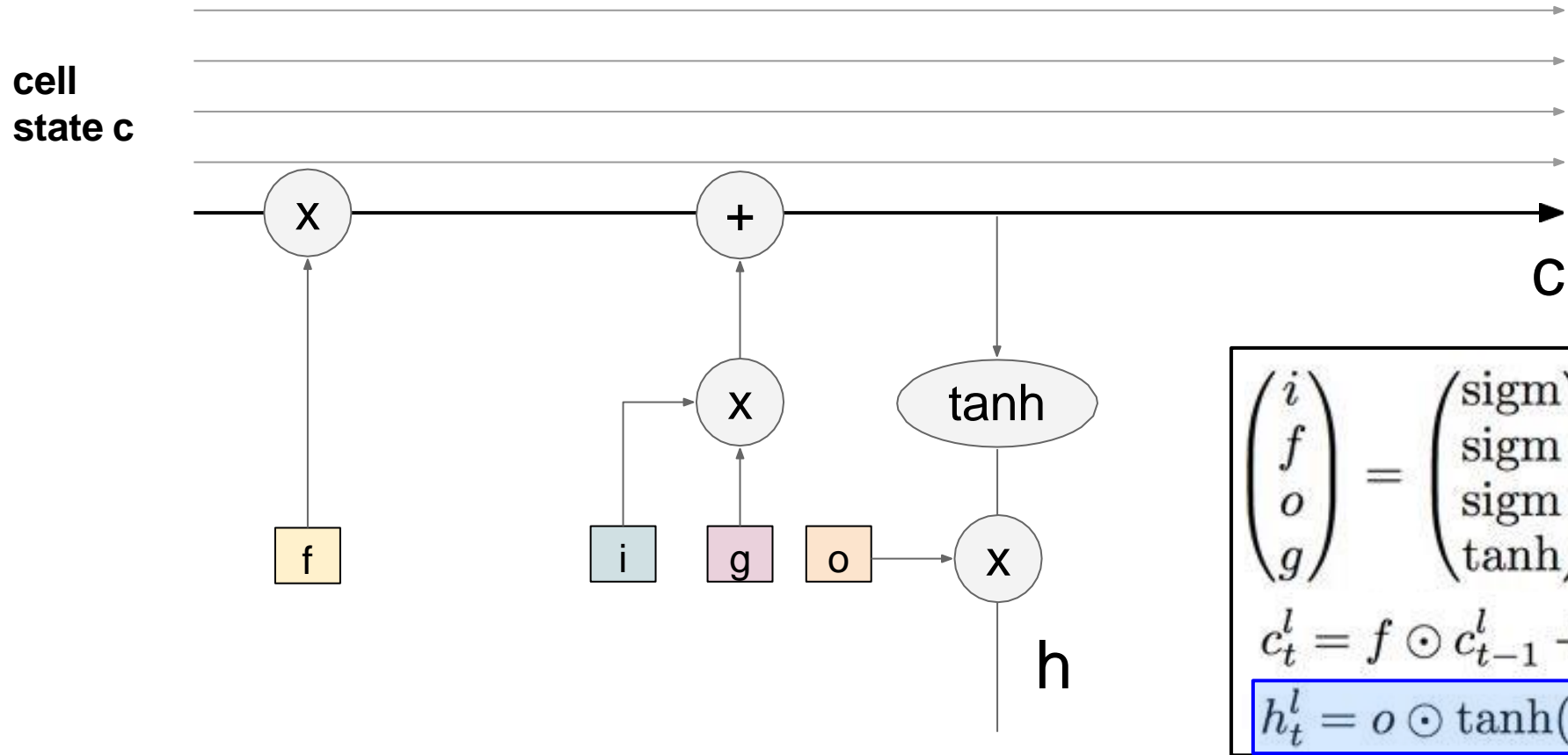
[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$

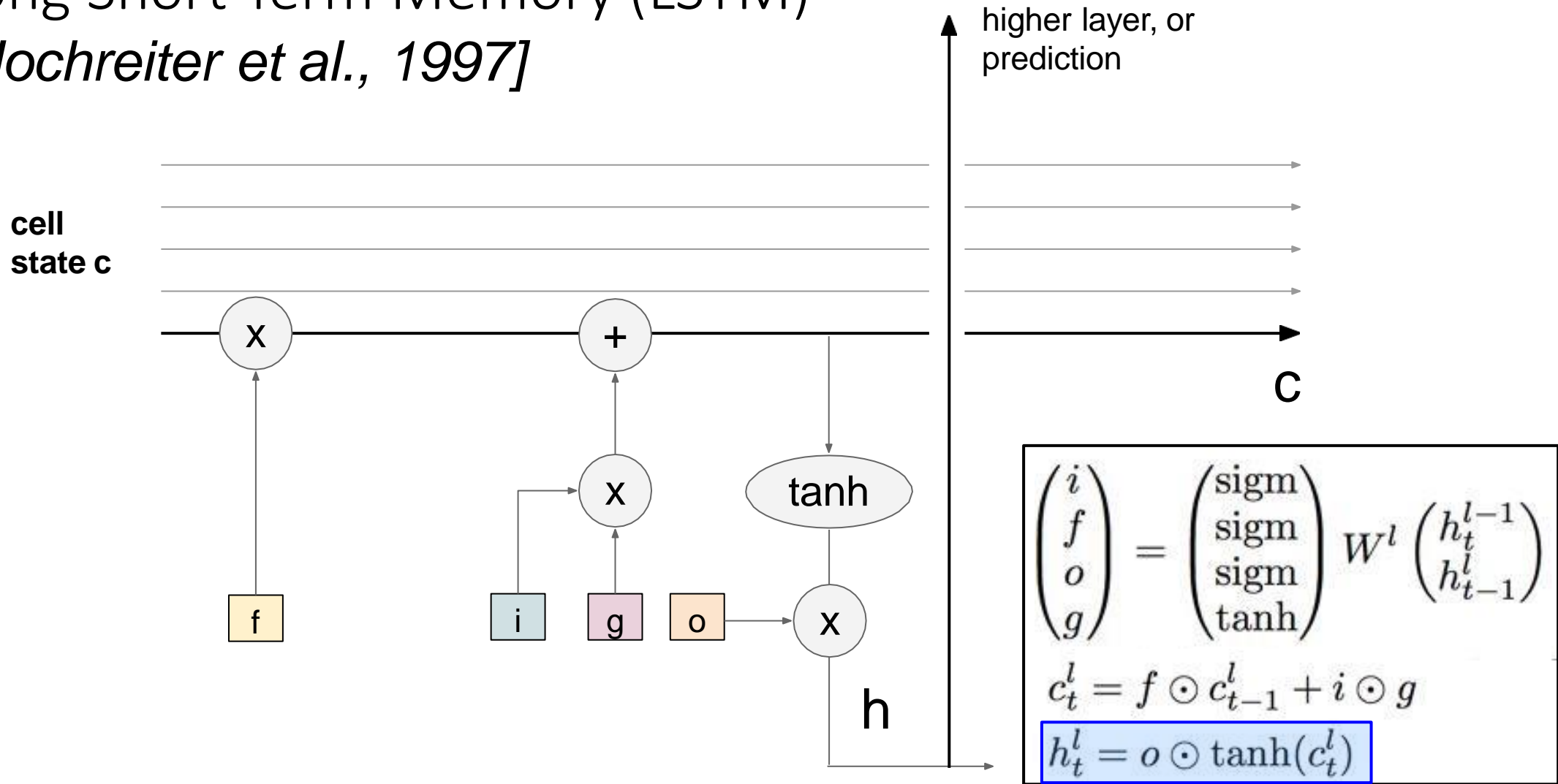
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

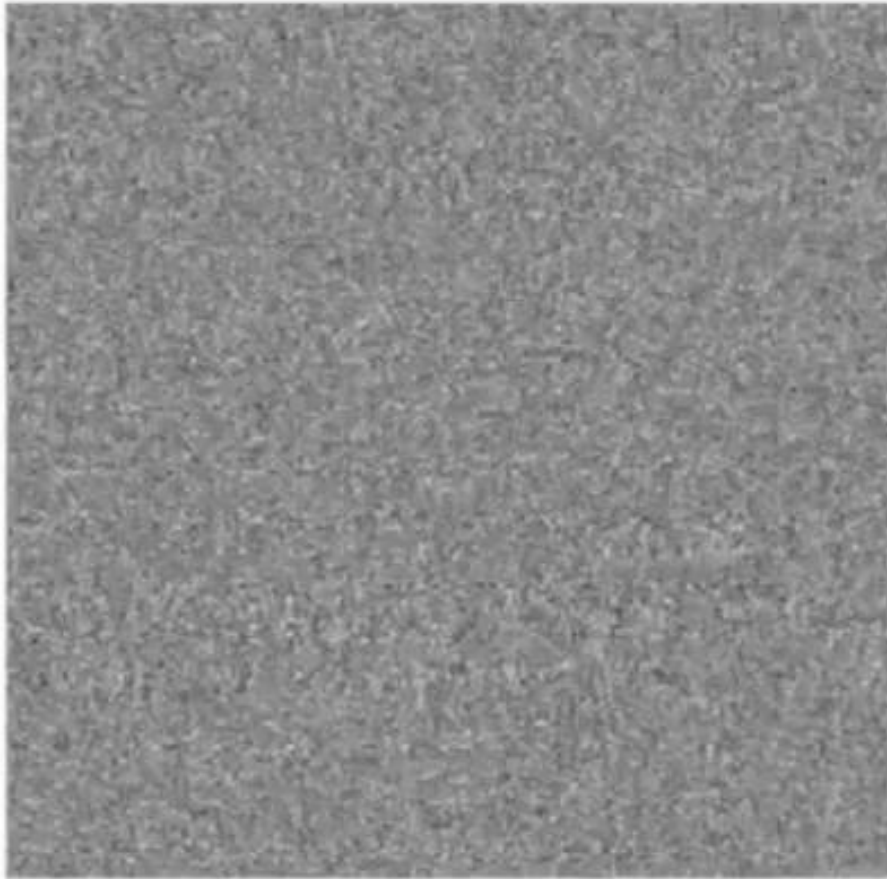
[Hochreiter et al., 1997]



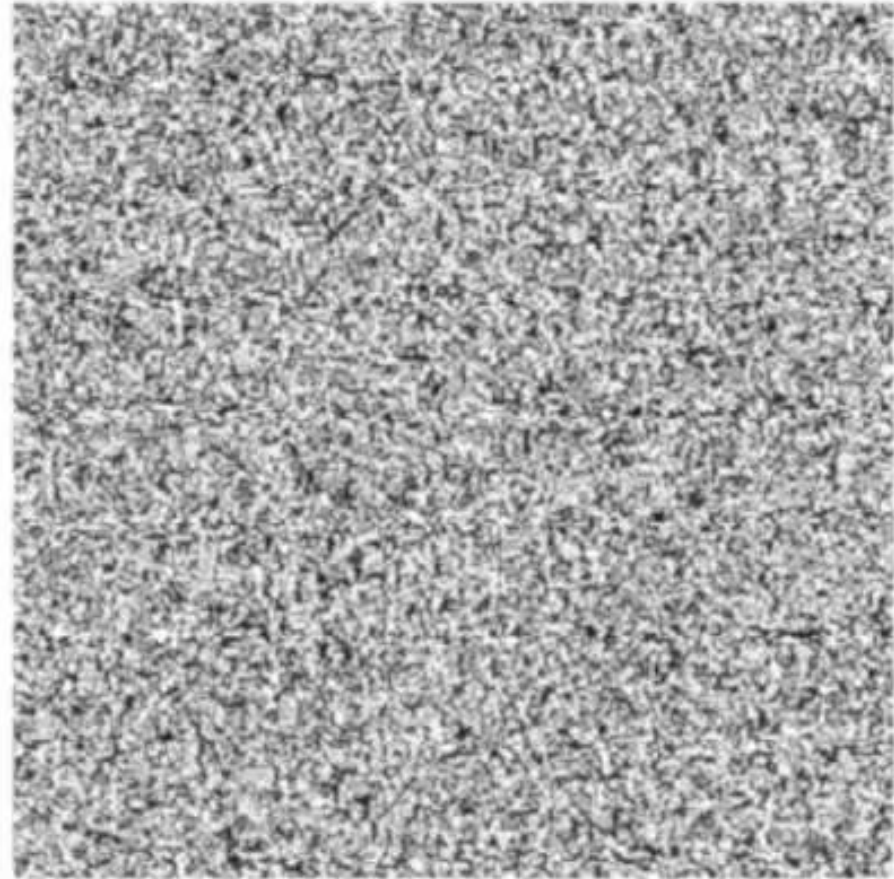
Understanding gradient flow dynamics

Backprop signal

127



127



Understanding gradient flow dynamics

Backprop signal video: <http://imgur.com/gallery/vaNahKE>

In RNN, the gradient vanishes much more quickly as we backprop from the last time step towards the first one

Therefore, RNN here cannot learn long time dependencies

Understanding gradient flow dynamics

RNN without any inputs

```
H = 5 # dimensionality of hidden state
T = 50 # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

Understanding gradient flow dynamics

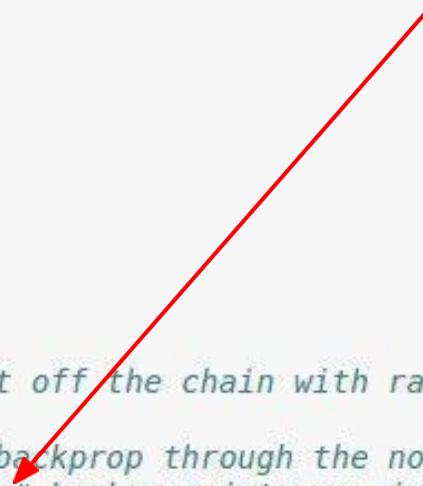
RNN without any inputs

```
H = 5 # dimensionality of hidden state
T = 50 # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

Back-propagation signal is repeatedly multiplied by Whh.



[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Understanding gradient flow dynamics RNN without any inputs

```
H = 5 # dimensionality of hidden state  
T = 50 # number of time steps
```

```
Whh = np.random.randn(H,H)
```

```
# forward pass of an RNN (ignoring inputs x)
```

```
hs = {}
```

```
ss = {}
```

```
hs[-1] = np.random.randn(H)
```

```
for t in xrange(T):
```

```
    ss[t] = np.dot(Whh, hs[t-1])
```

```
    hs[t] = np.maximum(0, ss[t])
```

```
# backward pass of the RNN
```

```
dhs = {}
```

```
dss = {}
```

```
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
```

```
for t in reversed(xrange(T)):
```

```
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
```

```
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

if the largest eigenvalue is < 1 , gradient will vanish
if the largest eigenvalue is > 1 , gradient will explode

can control vanishing with LSTM
can control exploding with gradient clipping

[On the difficulty of training Recurrent Neural Networks, Pascanu et al., 2013]

Vanishing gradient problem

An example how vanishing gradient problem can affect RNNs:

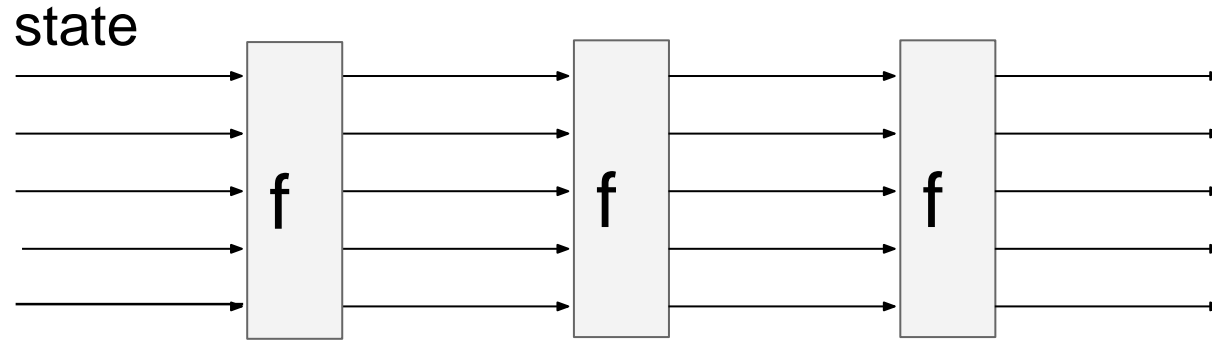
“In France, I had a great time and I learnt some of the _____ language.”



our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

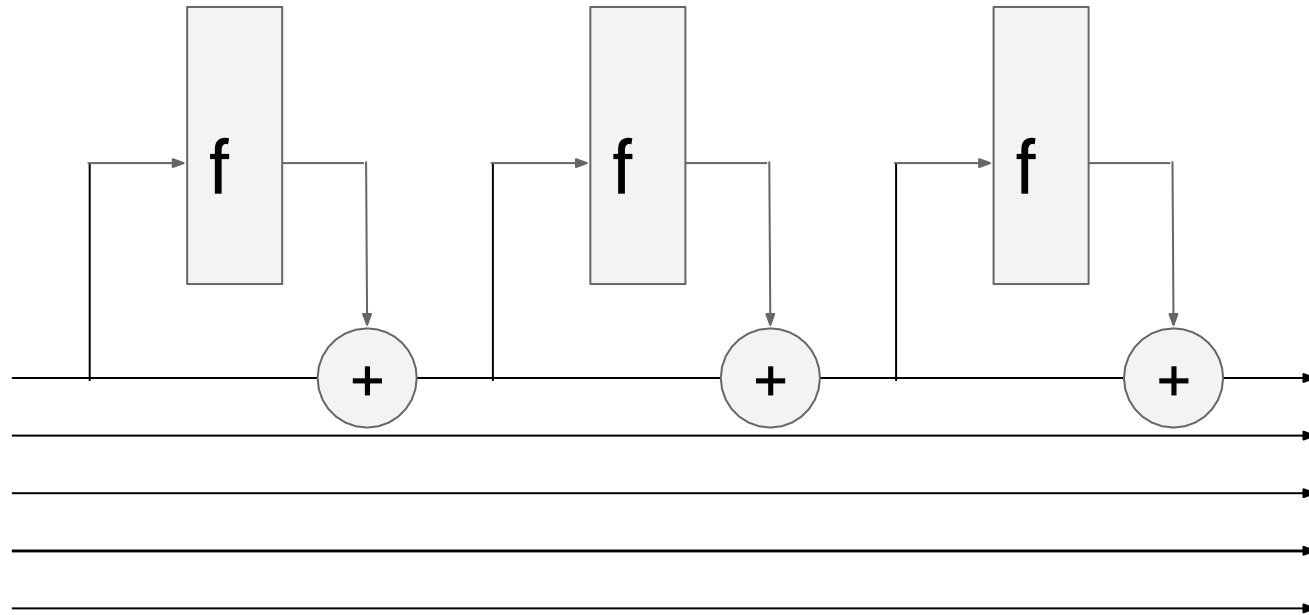
RNN

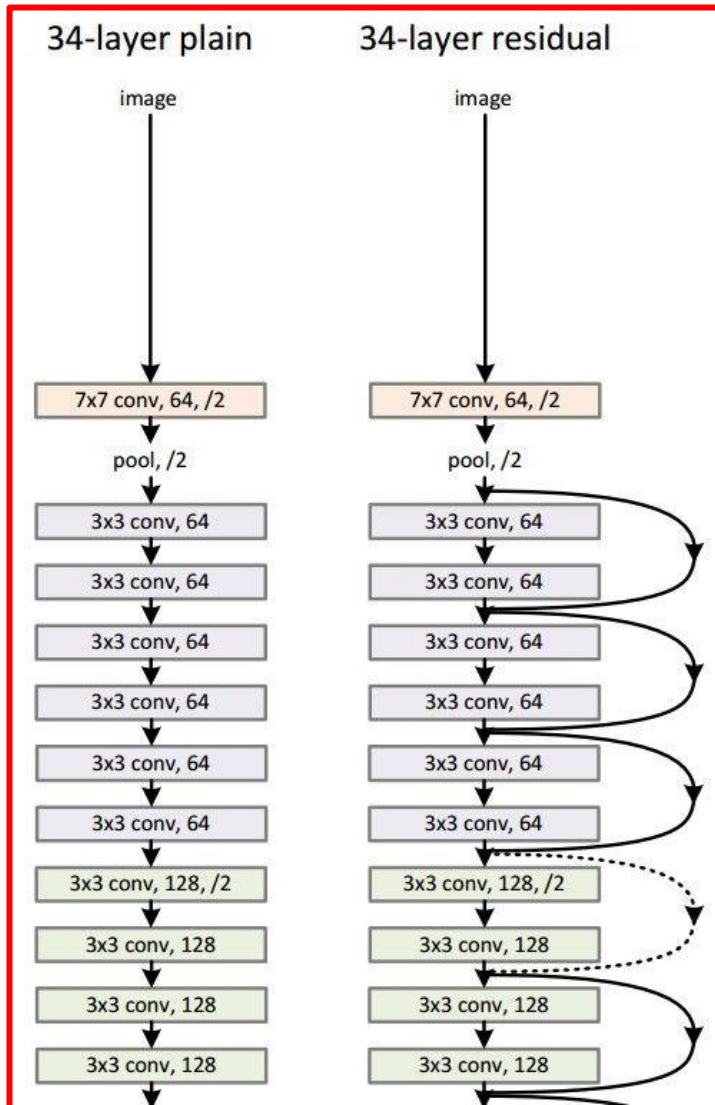
More prone to the vanishing gradient problem



LSTM

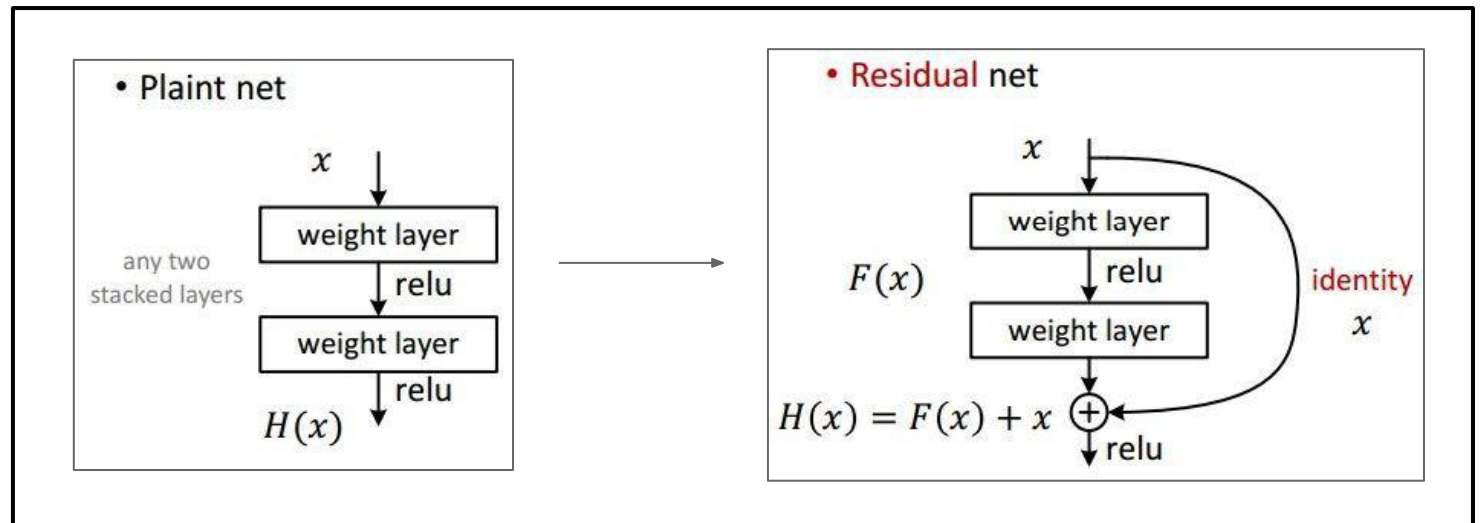
(ignoring forget gates)





Recall: “PlainNets” vs. ResNets

ResNet is to PlainNet what LSTM is to RNN, kind of.



Vanishing gradient problem summary

To address this problem, use

- better activation function (eg, ReLU)
- proper initialization ($W=Identity$, bias=zeros) to prevent W from shrinking the gradients
- replace RNN cells with LSTM or other gated cells (LSTM variants) to control what information is passed through