GE 461 Introduction to Data Science

Spring 2021

Deep Learning

Hamdi Dibeklioğlu

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

Slide by Dhruv Batra

So, What is DEEP Machine Learning

A few different ideas:

- **(Hierarchical) Compositionality**
  - Cascade of non-linear transformations
  - Multiple layers of representations

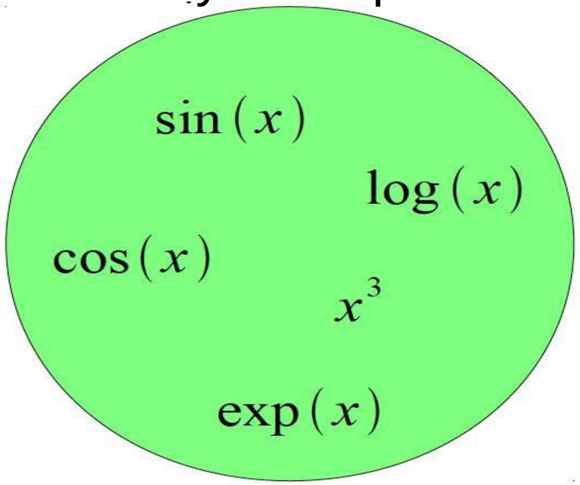
- **End-to-End Learning**
  - Learning (goal-driven) representations
  - Learning feature extraction

- **Distributed Representations**
  - No single neuron “encodes” everything
  - Groups of neurons work together

Slide by Dhruv Batra

Building A Complicated Function

Given a library of simple functions



Compose a

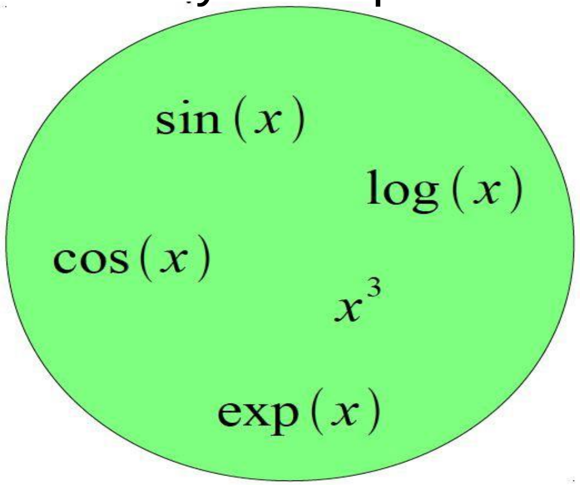
complicated function

Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



Compose a

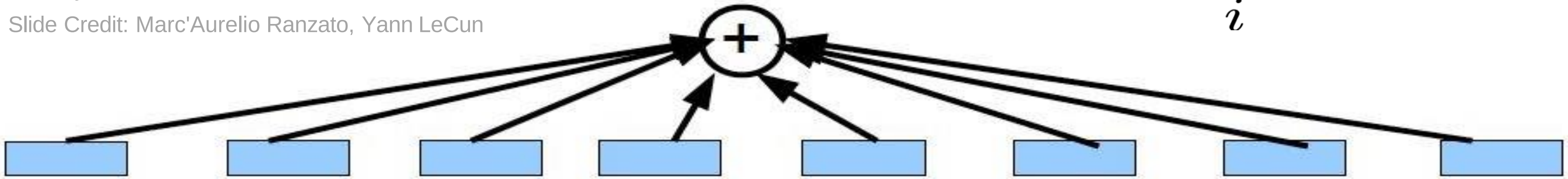
complicated function

Idea 1: Linear Combinations

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

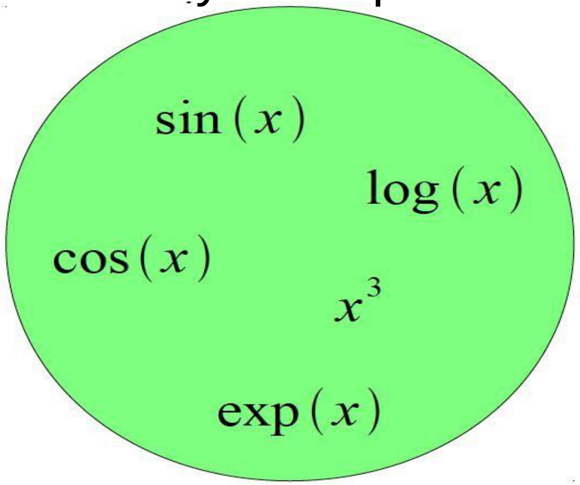
Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Building A Complicated Function

Given a library of simple functions



Compose a

complicated function

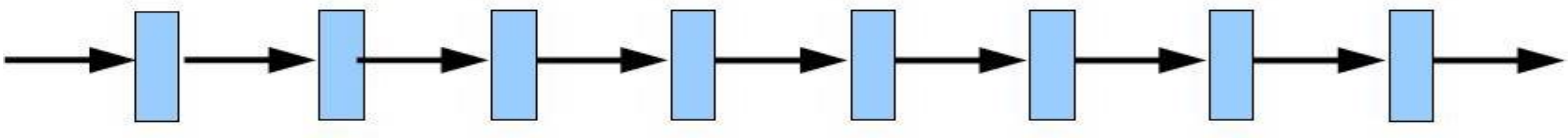
Idea 2: Compositions

- Deep Learning

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

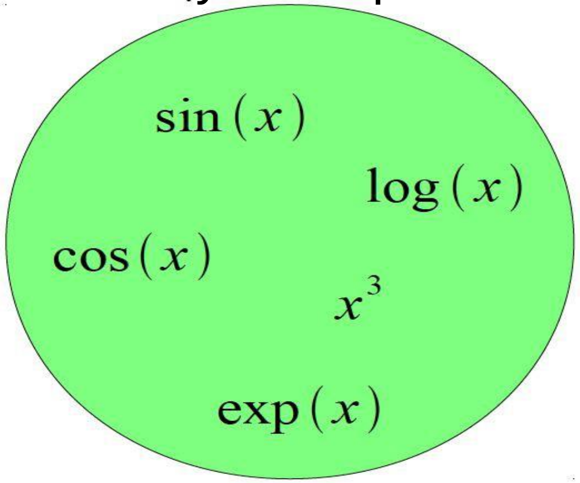
Slide by Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun



Building A Complicated Function

Given a library of simple functions



Compose a

complicated function

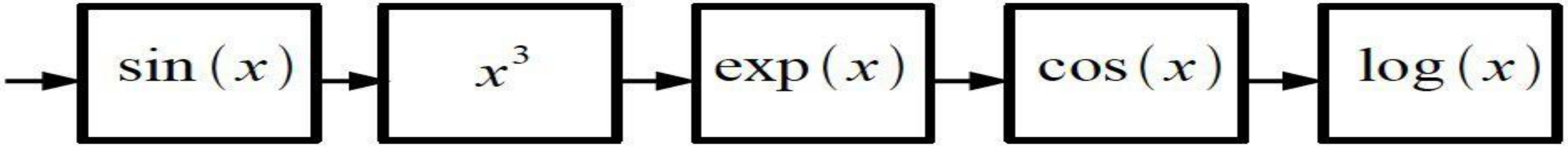
Idea 2: Compositions

- Deep Learning

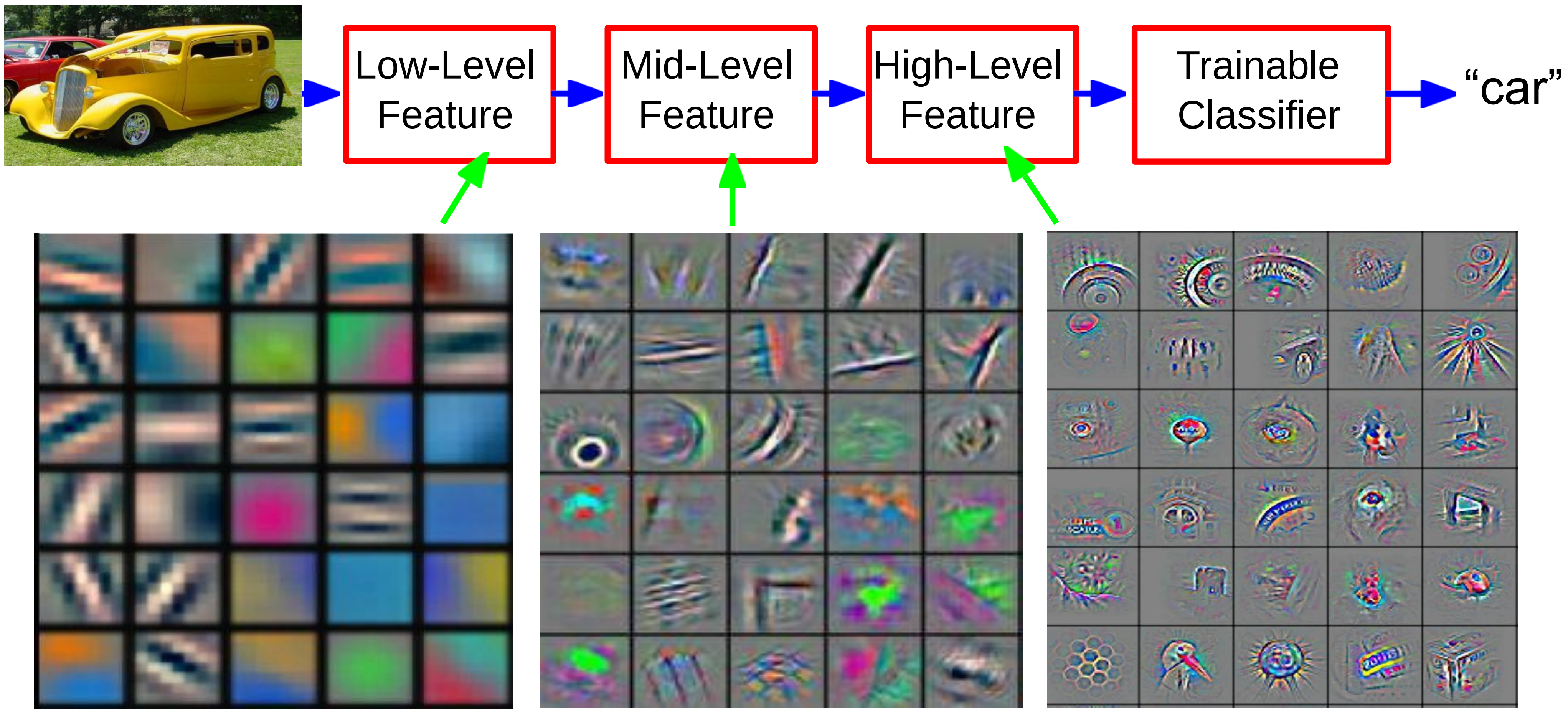
$$f(x) = \log(\cos(\exp(\sin^3(x))))$$

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

Slide by Dhruv Batra

"Shallow" vs Deep Learning

- "Shallow" models

hand-crafted Feature Extractor
fixed

"Simple" Trainable Classifier
learned

- Deep models (especially supervised deep learning)

Trainable Feature-Transform / Classifier

Trainable Feature-Transform / Classifier

Trainable Feature-Transform / Classifier

Learned Internal Representations

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

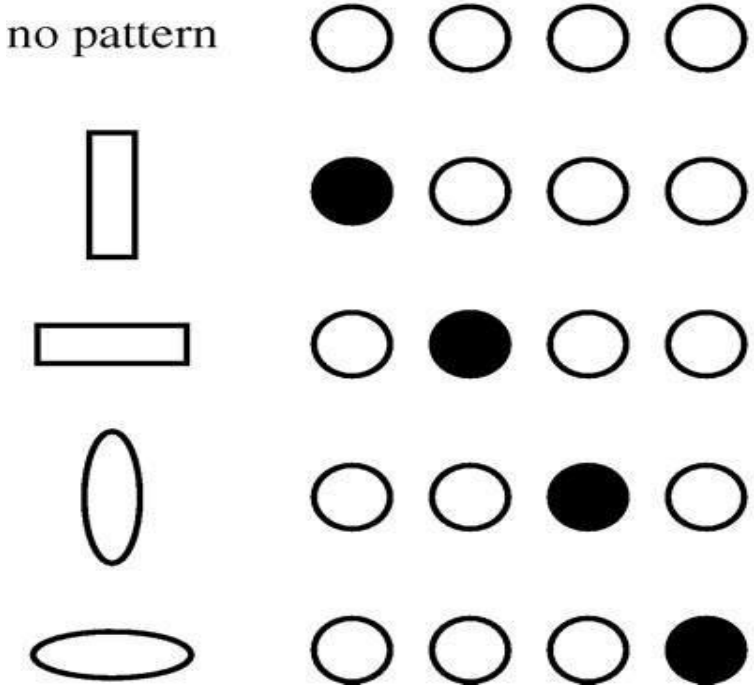
- **End-to-End Learning**
  - Learning (goal-driven) representations
  - Learning feature extraction

- **Distributed Representations**
  - No single neuron “encodes” everything
  - Groups of neurons work together

Slide by Dhruv Batra

Distributed Representations Toy Example

- Local vs Distributed

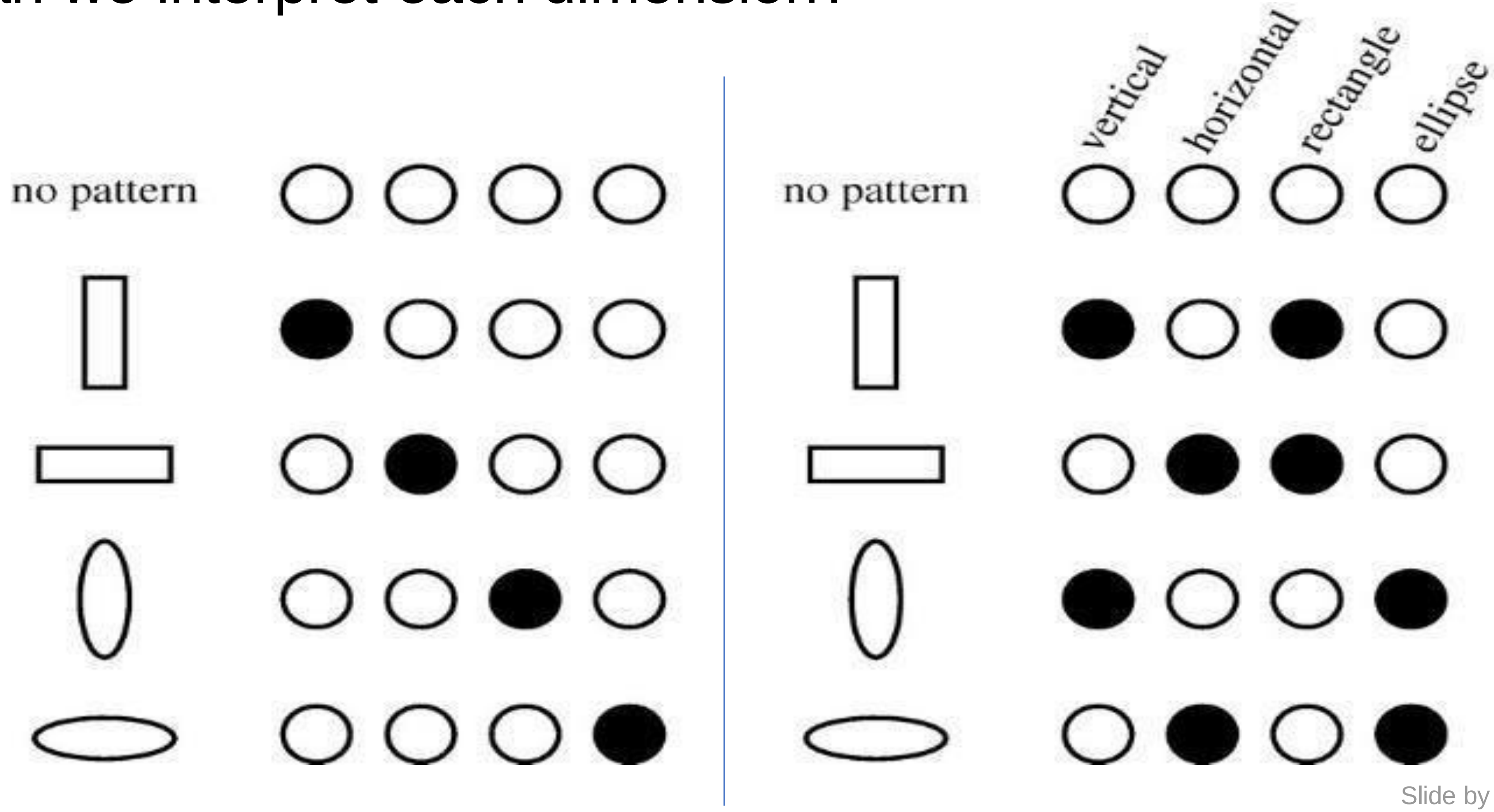


Slide by Dhruv Batra

Slide Credit: Moontae Lee

Distributed Representations Toy Example

- Can we interpret each dimension?



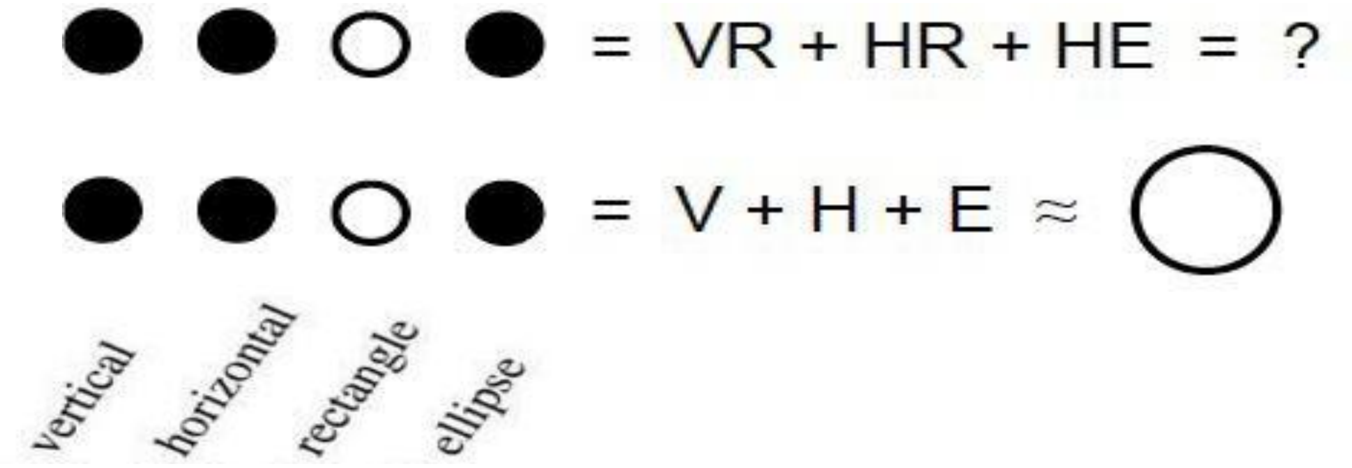
Slide by Dhruv Batra

Slide Credit: Moontae Lee

Power of distributed representations!

Local

Distributed



Slide by Dhruv Batra

Slide Credit: Moontae Lee

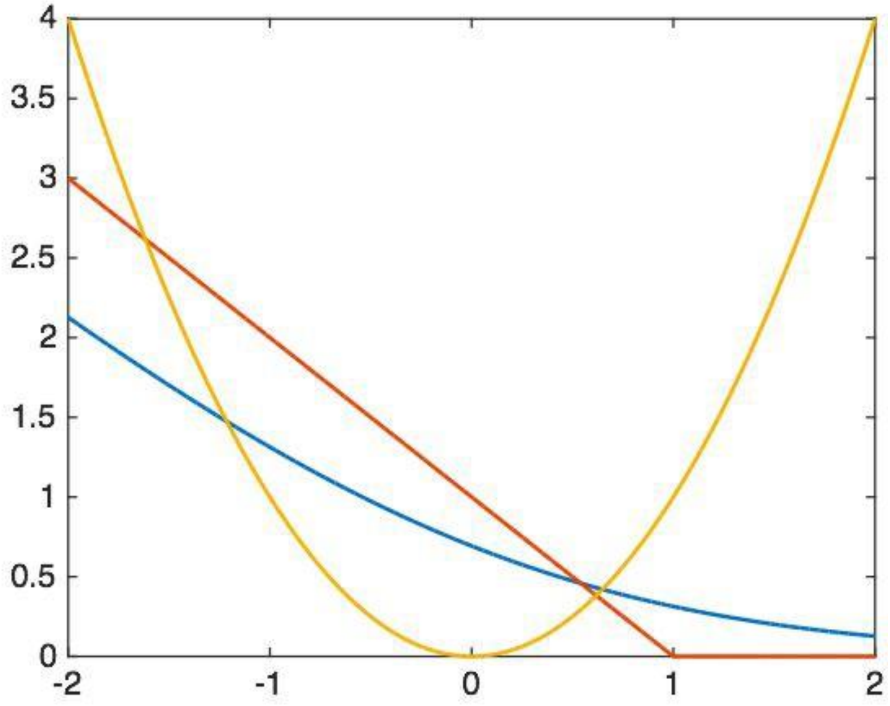
$$f(x, W) = Wx$$

- 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

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

- 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

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

- 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 - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith training example.

Regression Losses

Mean Absolute Error/L1 Loss

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

- n - Number of training examples.
- i - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith 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 - ith training example in a data set.
- y(i) - Ground truth label for ith training example.
- y_hat(i) - Prediction for ith training example.

Weight Regularization

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

\lambda = regularization strength (hyperparameter)

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

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

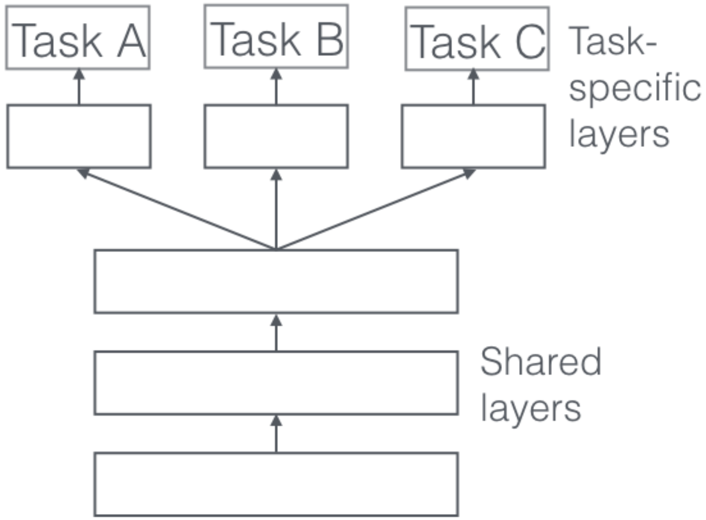
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Why does it make sense?

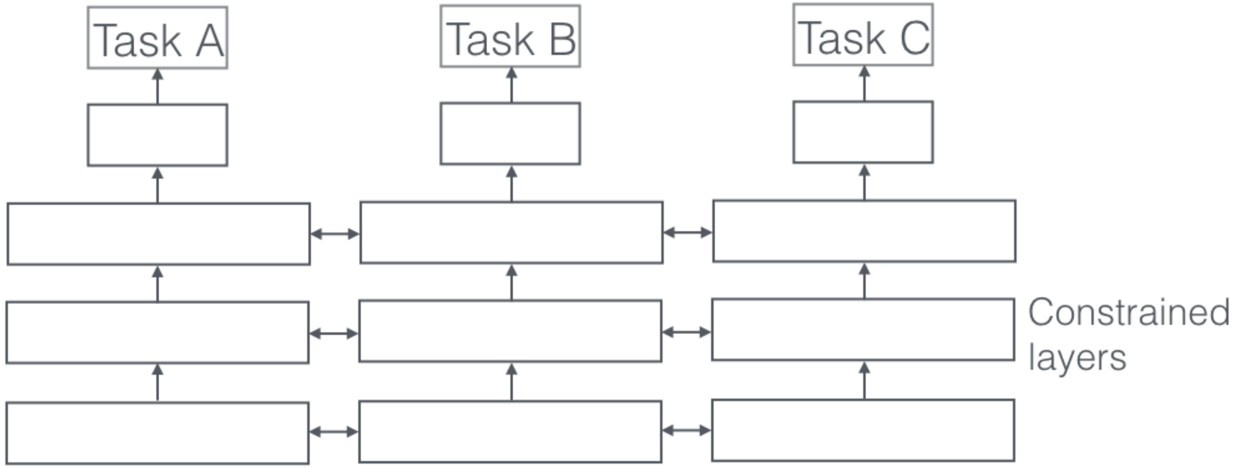
$$w_1^T x = w_2^T x = 1$$

Multi-task Learning

Jointly minimize the losses of different tasks



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

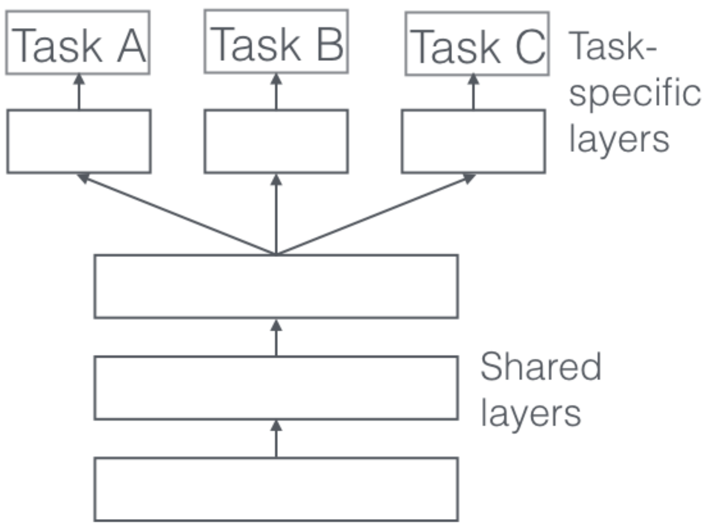


Soft parameter sharing for multi-task learning in deep neural networks

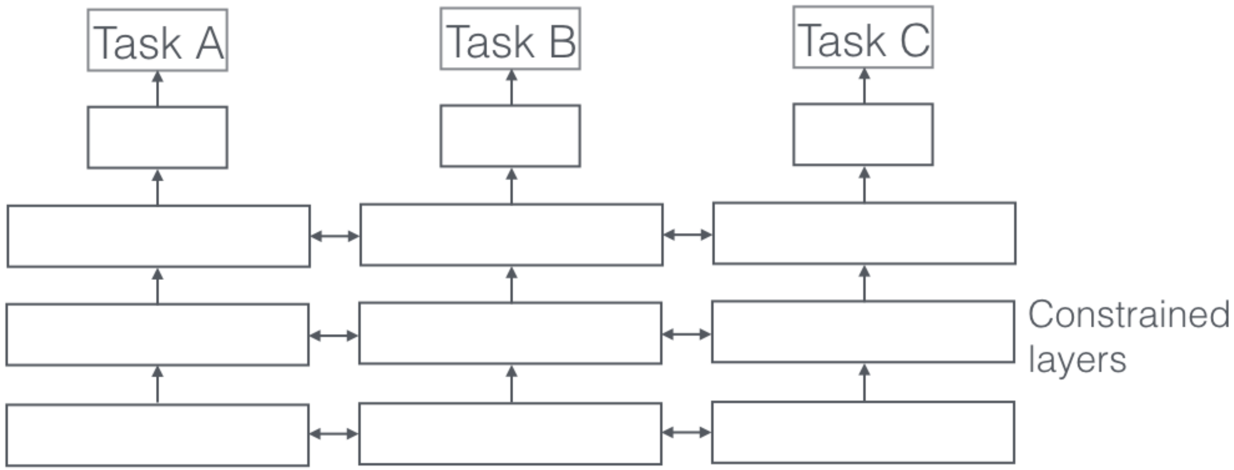
Multi-task Learning

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

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



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



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

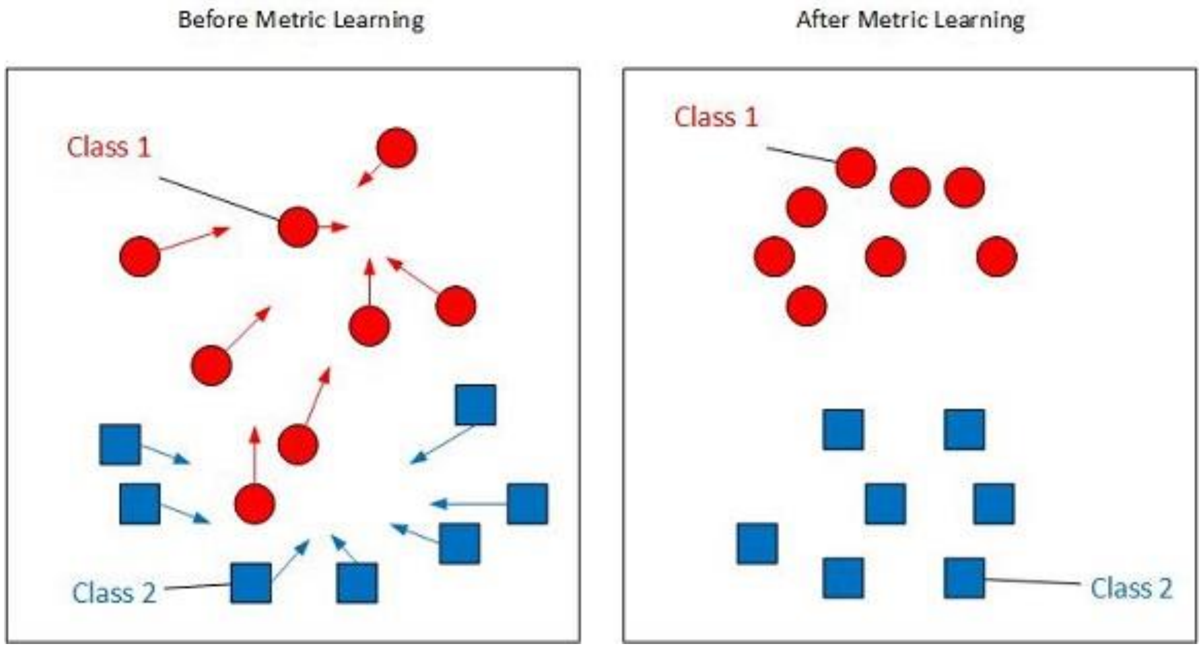
Metric/Contrastive Learning

Learn *distinctiveness*

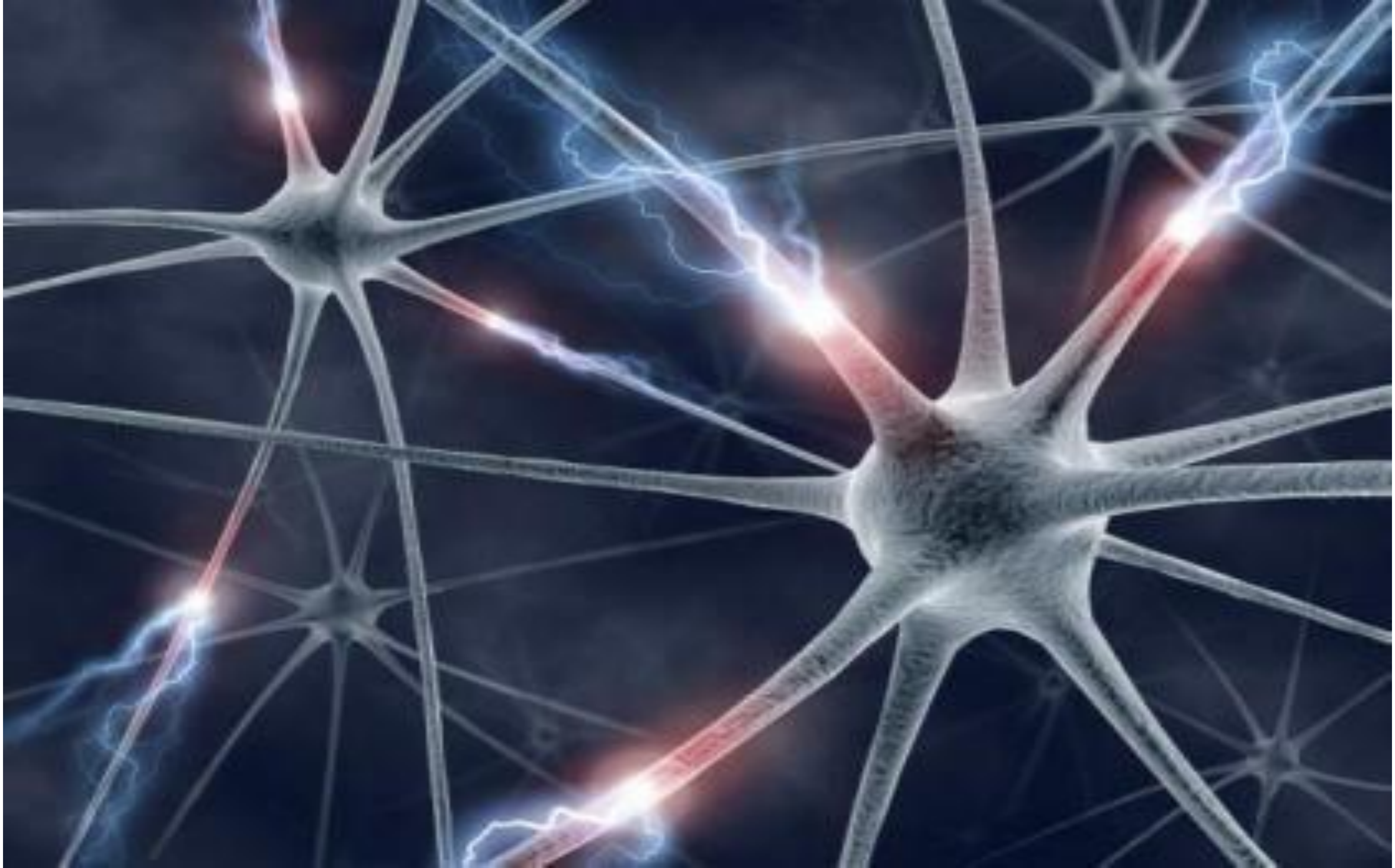
1. A distance-based loss function (as opposed to prediction error-based loss functions like Logistic loss or Hinge loss used in Classification).

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.



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



Neural Networks

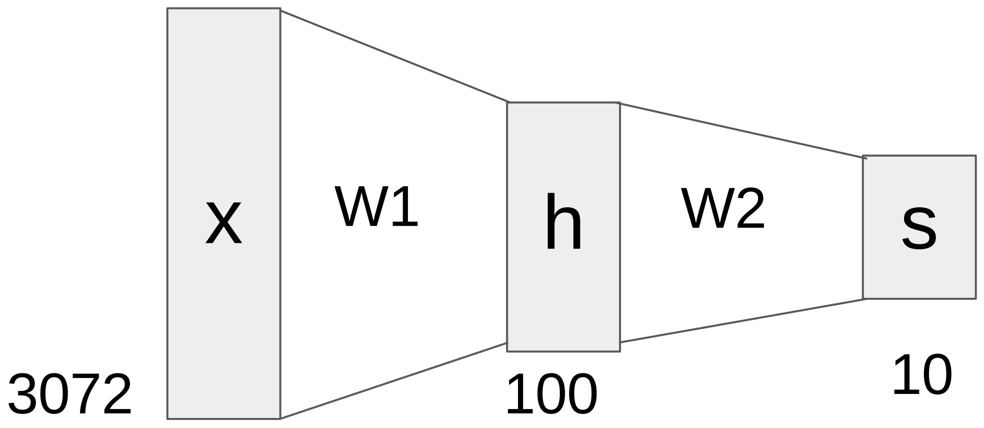
Linear score function: $f = Wx$

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

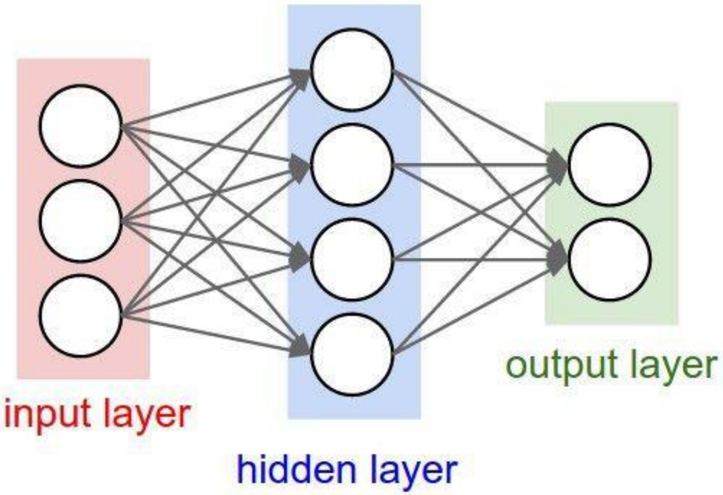
Linear score function: $f = Wx$

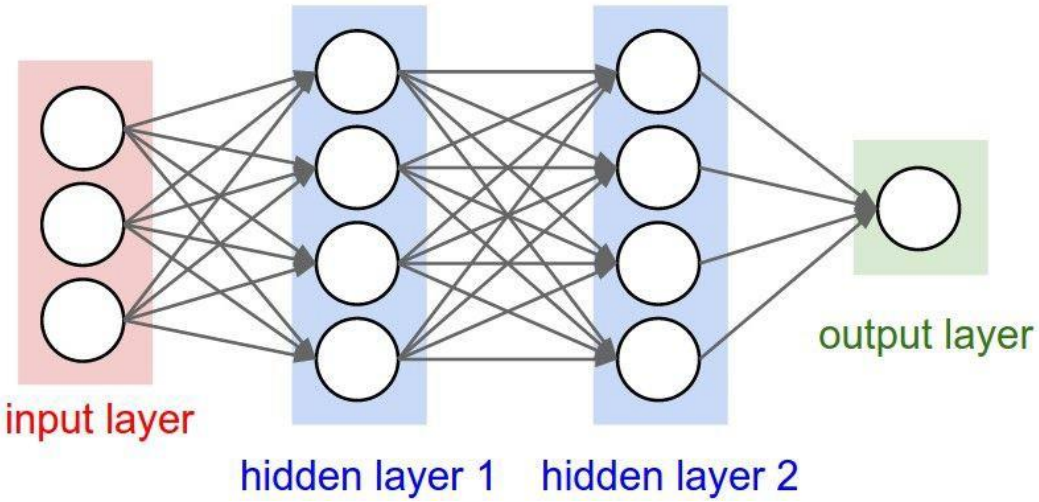
2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

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

Neural Networks: Architectures



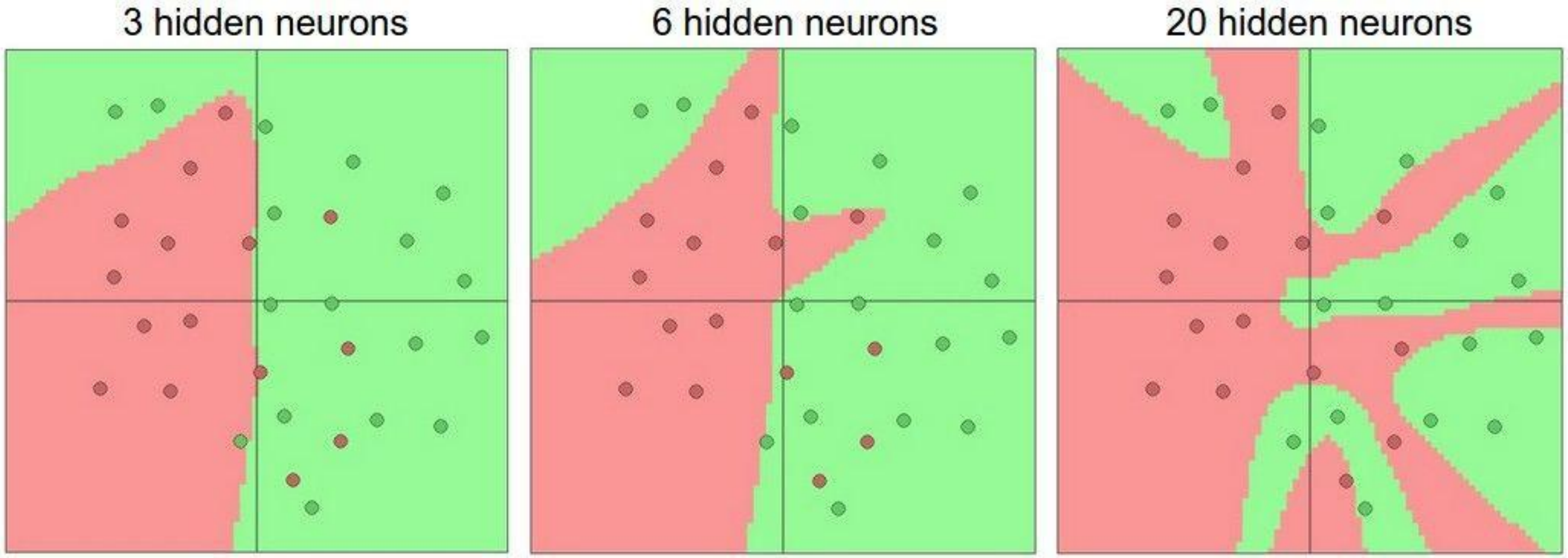


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

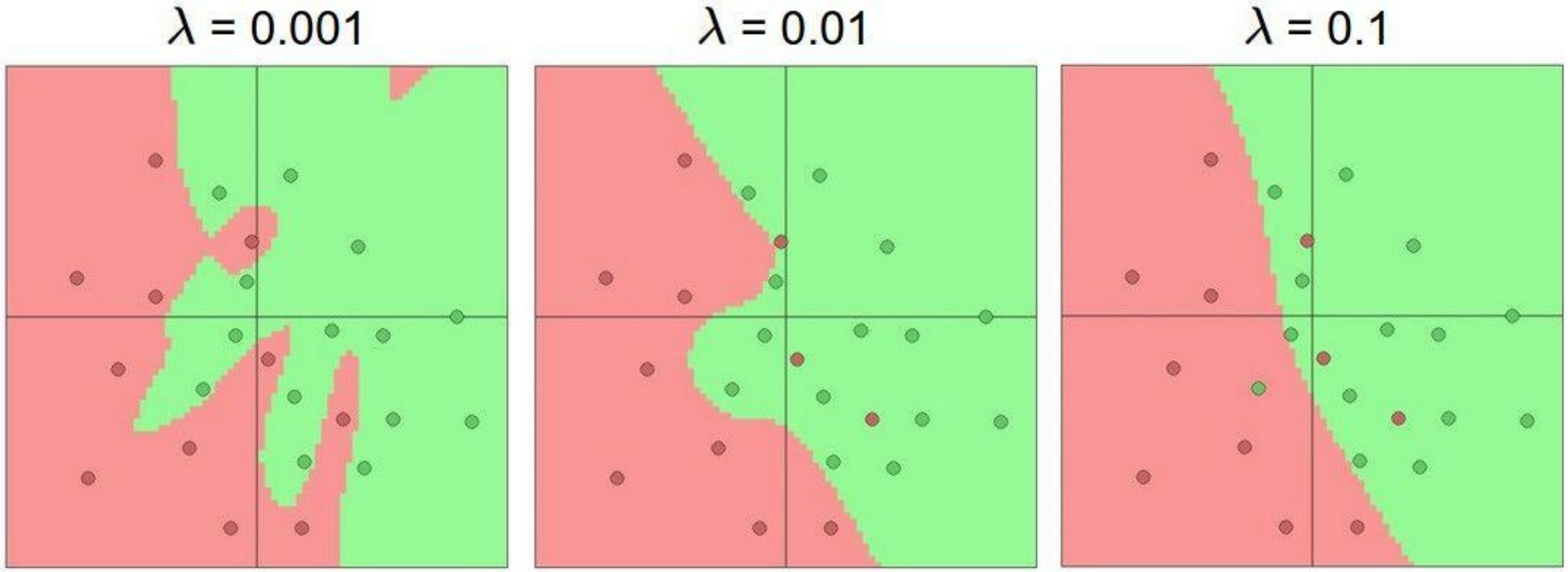
"Fully-connected" layers

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:

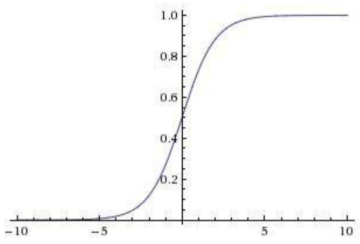


Activation Functions

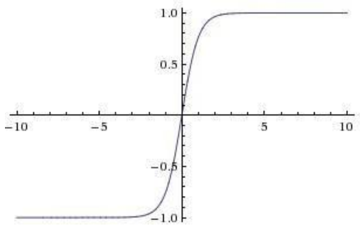
Activation Functions

Sigmoid

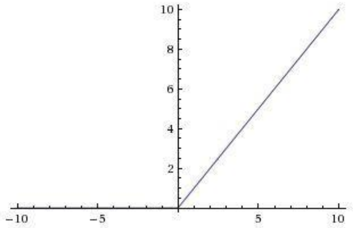
$$\sigma(x) = 1/(1+e^{-x})$$



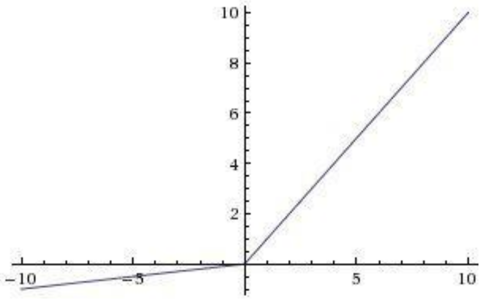
tanh tanh(x)



ReLU max(0,x)



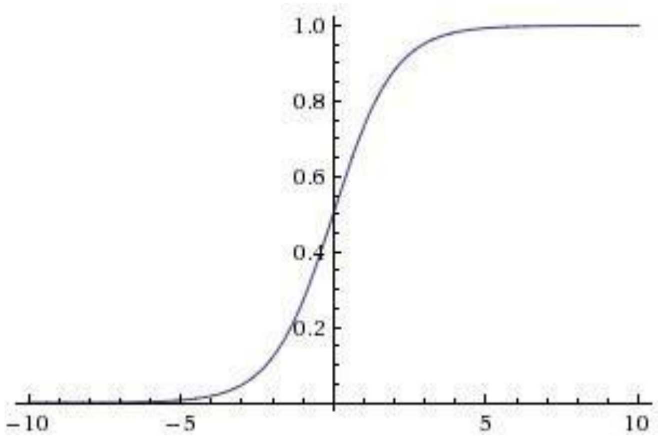
Leaky ReLU
max(0.1x, x)



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

Activation Functions

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



Sigmoid

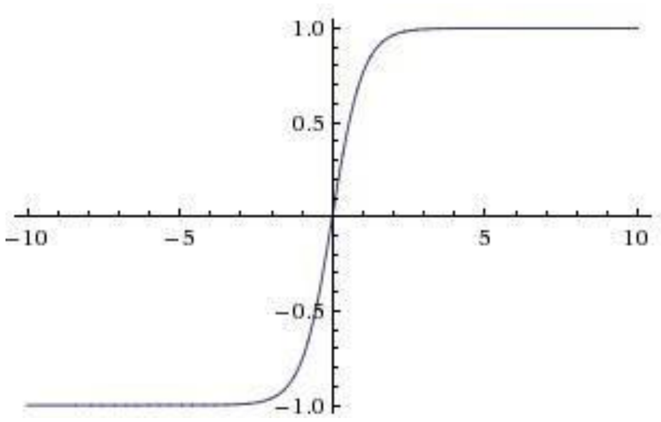
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

3 problems:

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

Activation Functions

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

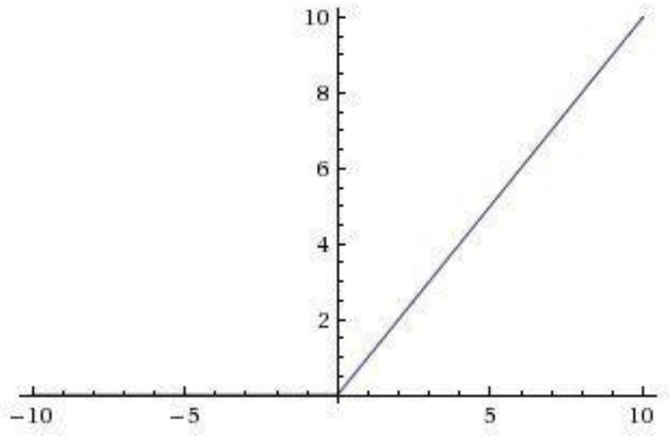


tanh(x)

- 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)

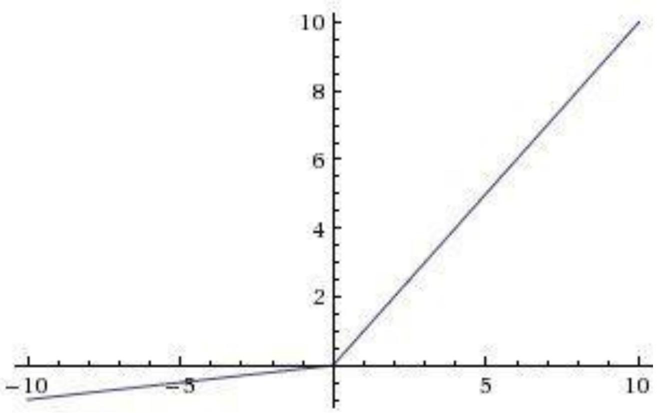
- Not zero-centered output
- An annoyance:

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

[Krizhevsky et al., 2012]

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



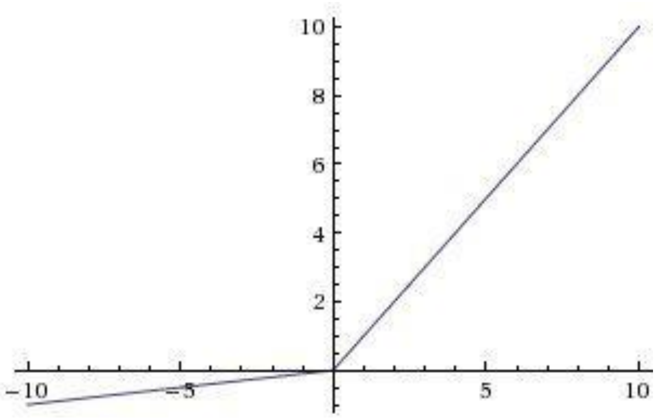
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

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

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



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

Leaky ReLU

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

Parametric Rectifier (PReLU)

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

backprop into \alpha
(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!

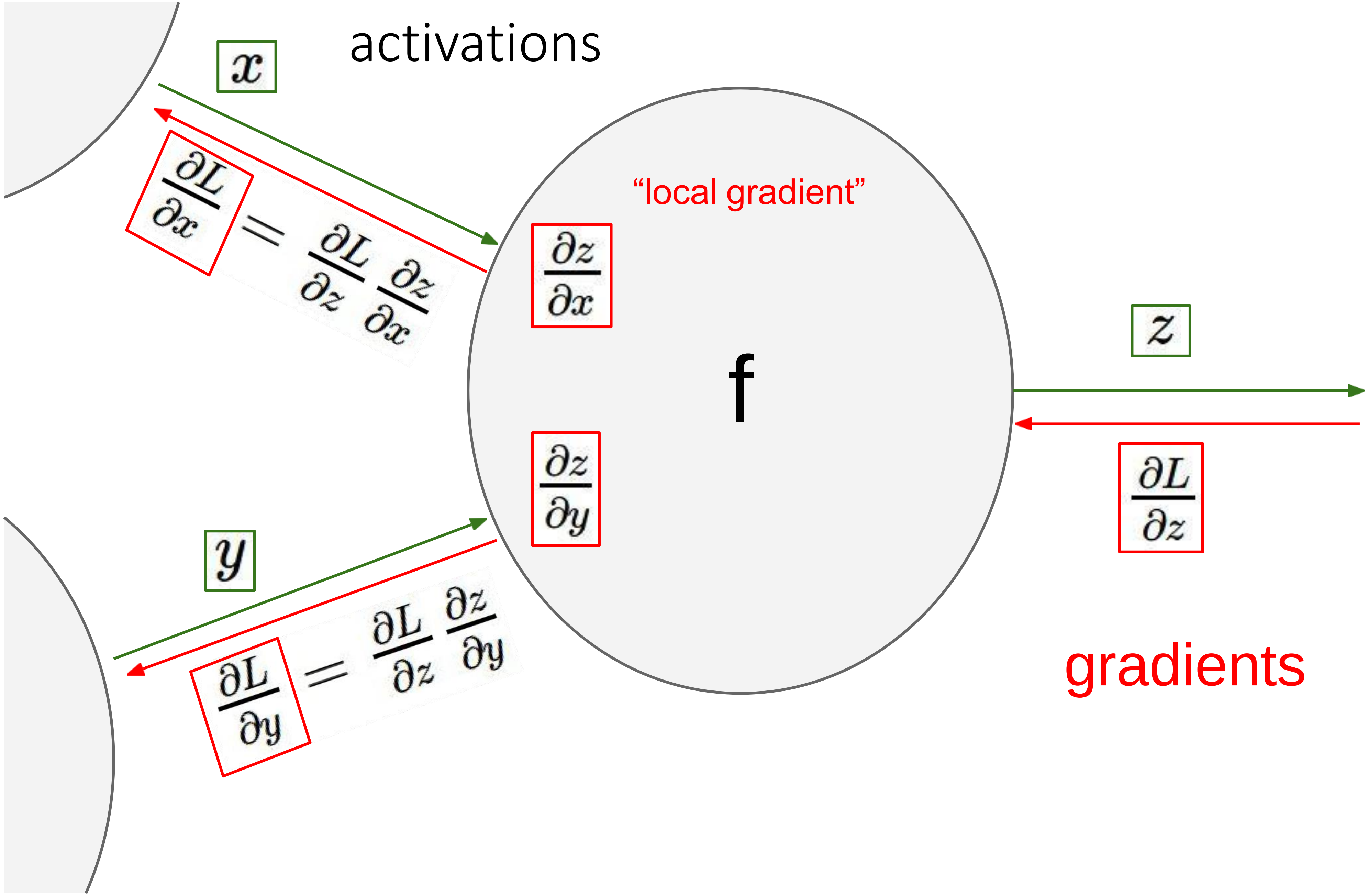
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

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



activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

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

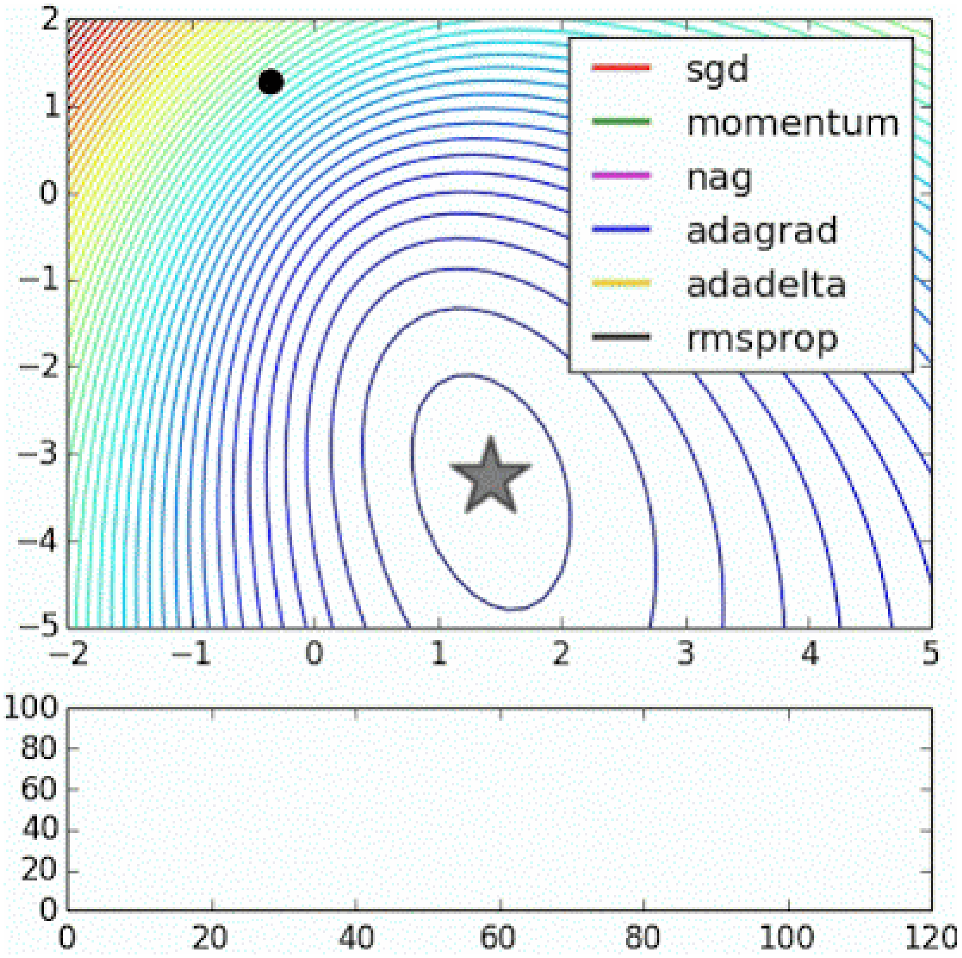
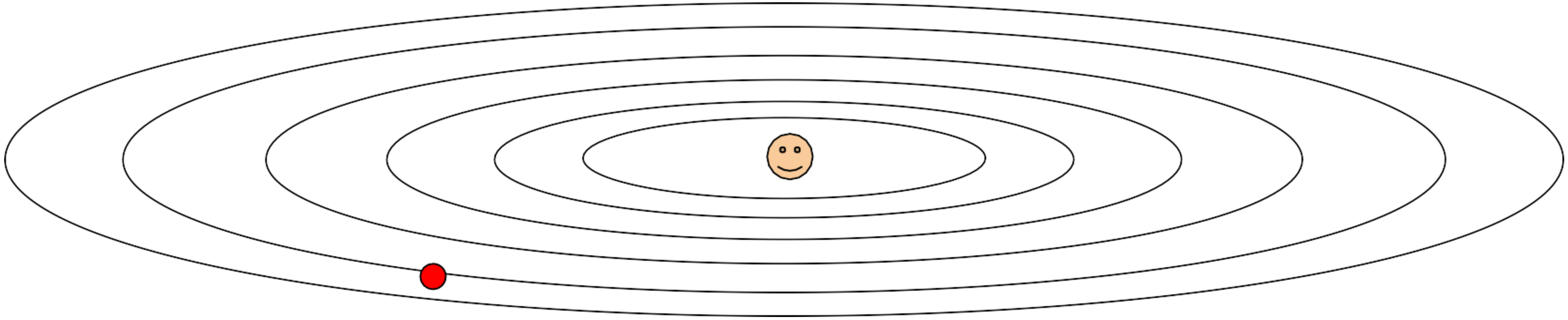


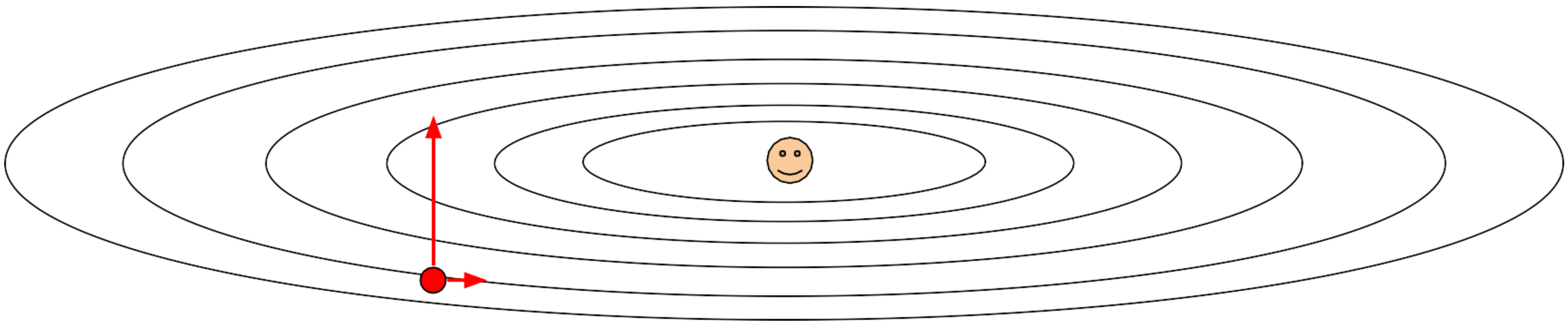
Image credits: Alec Radford

Suppose loss function is steep vertically but shallow horizontally:



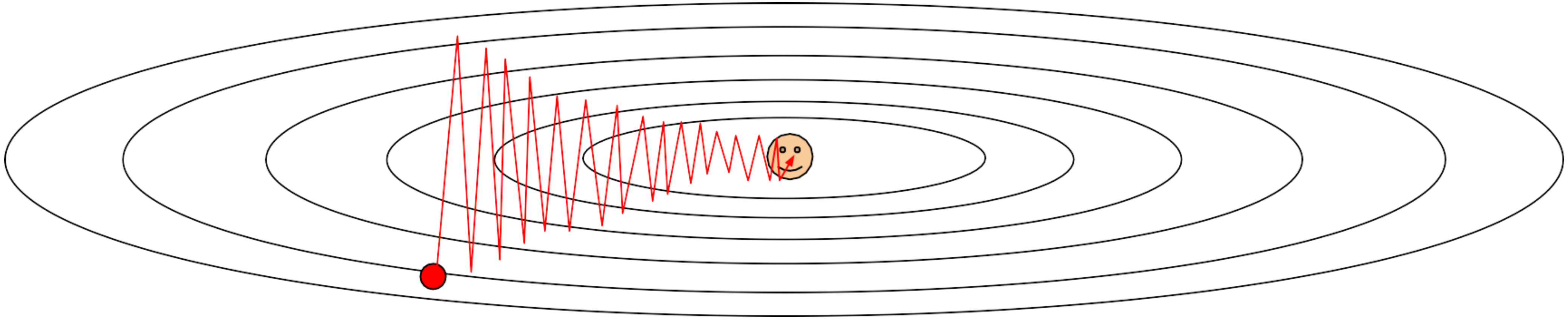
Q: What is the trajectory along which we converge towards the minimum with SGD?

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

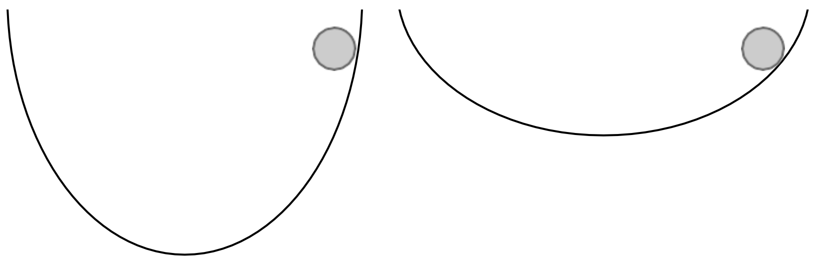
```
# Gradient descent update
x += - learning_rate * dx
```

```
# 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

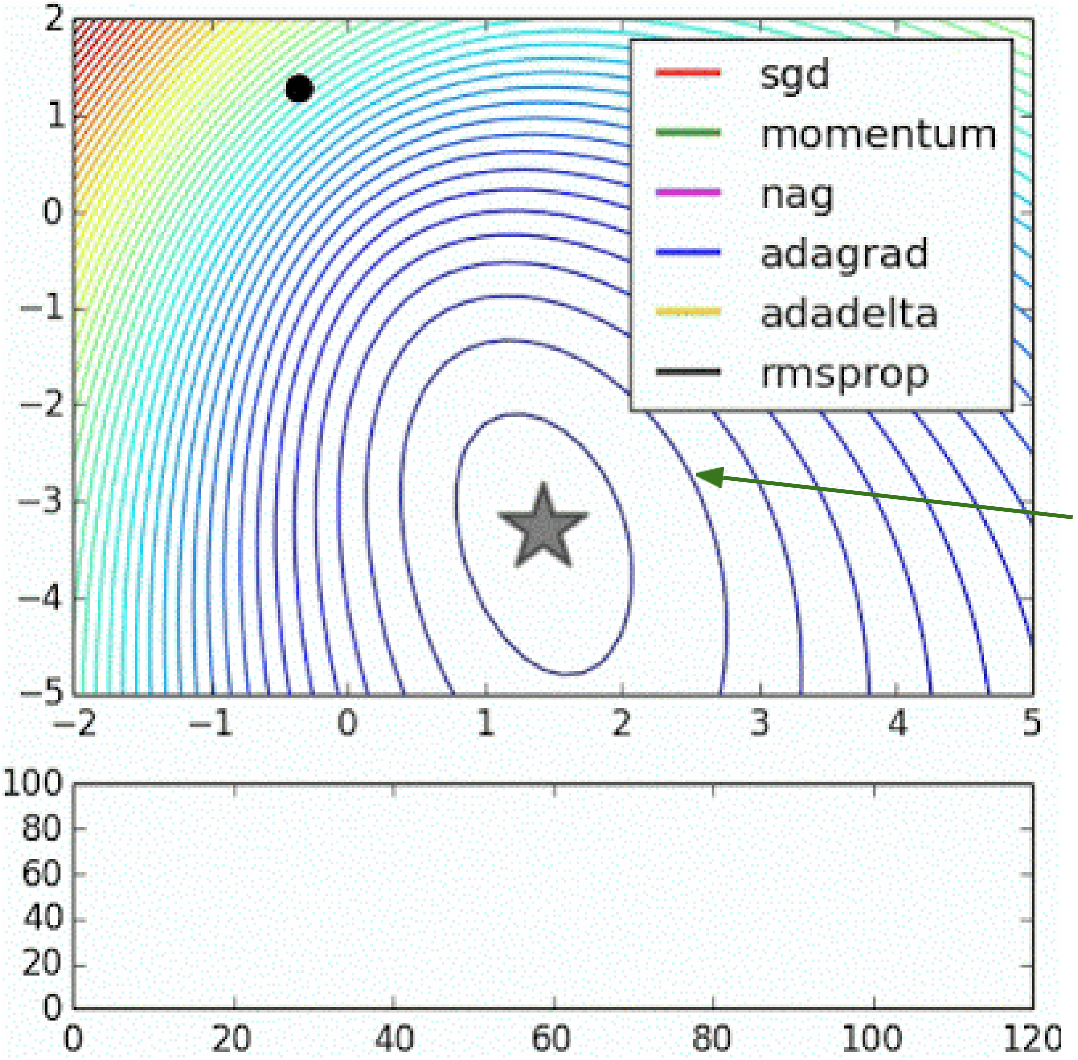
```
# Gradient descent update
x += - learning_rate * dx
```



```
# 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

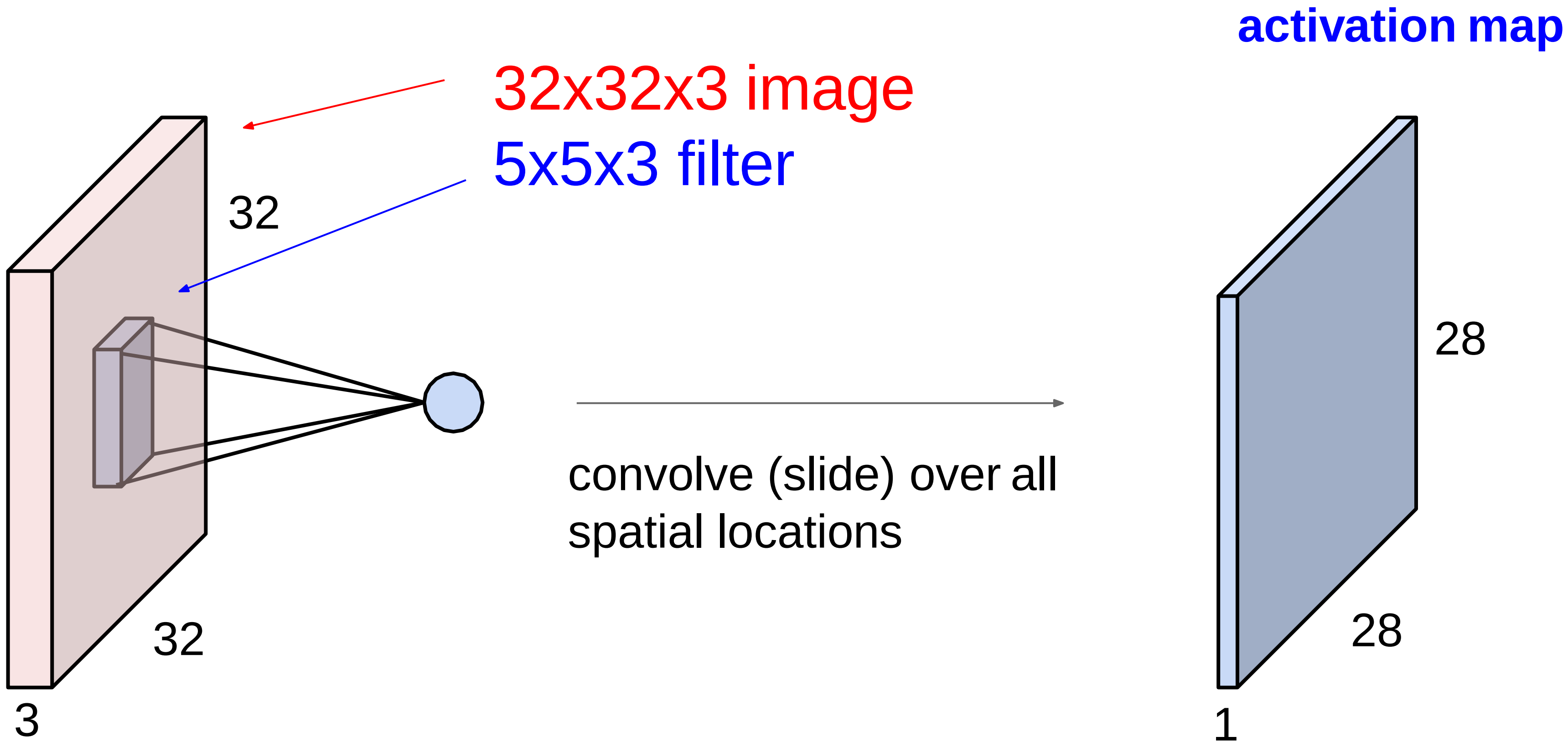
SGD
vs
Momentum



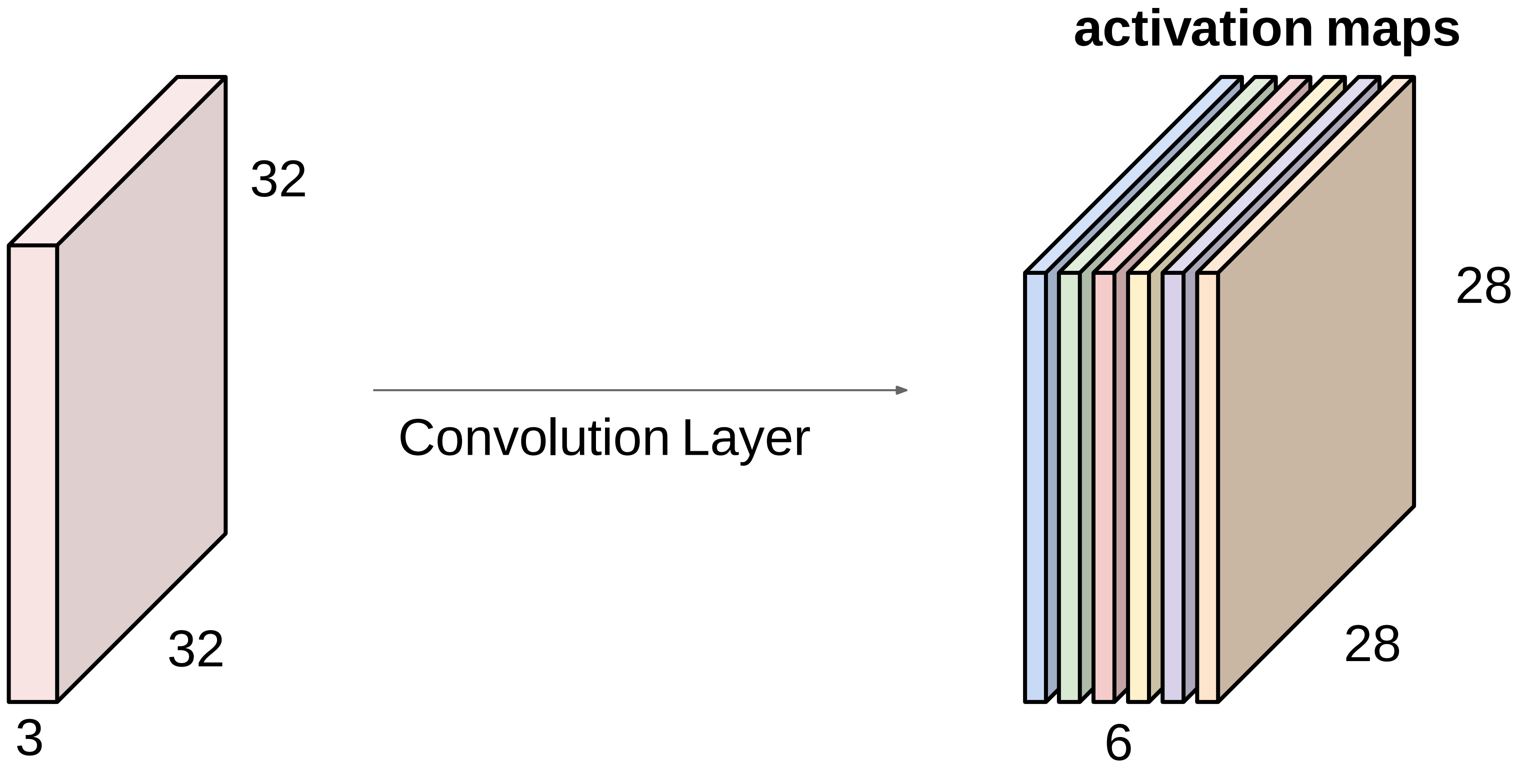
notice momentum overshooting the target, but overall getting to the minimum much faster.

Convolutional Neural Networks

Convolution Layer

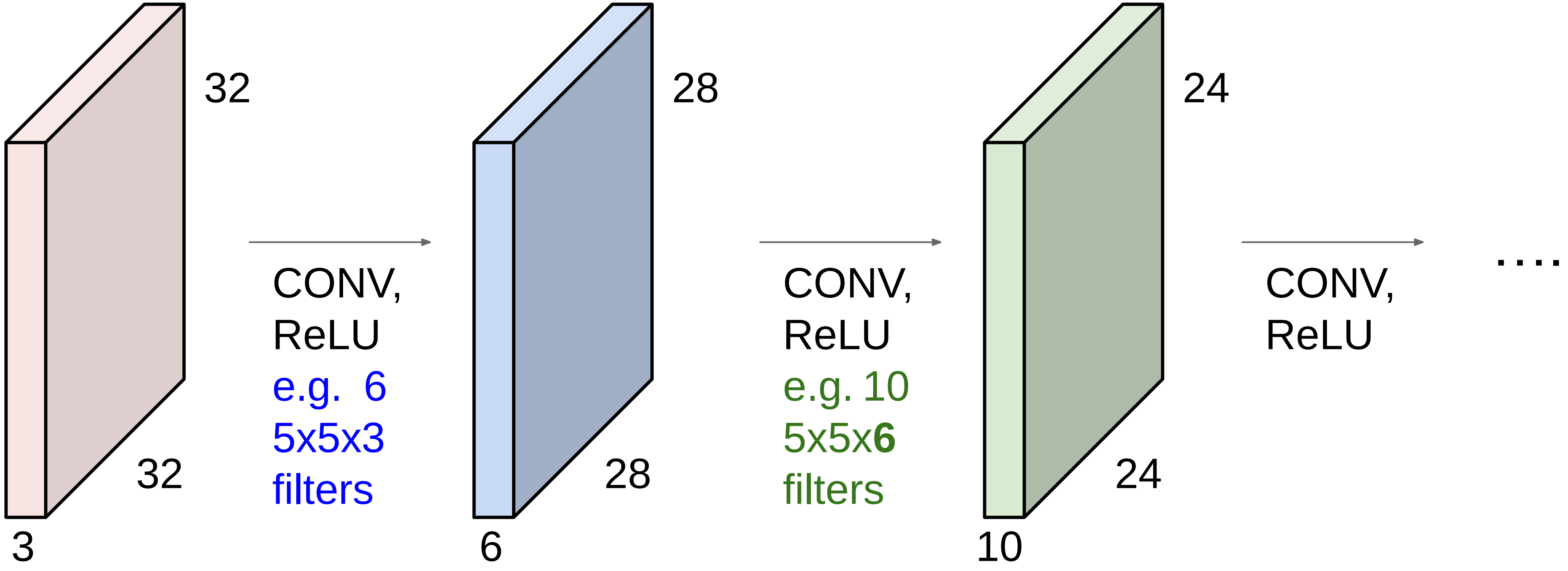


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



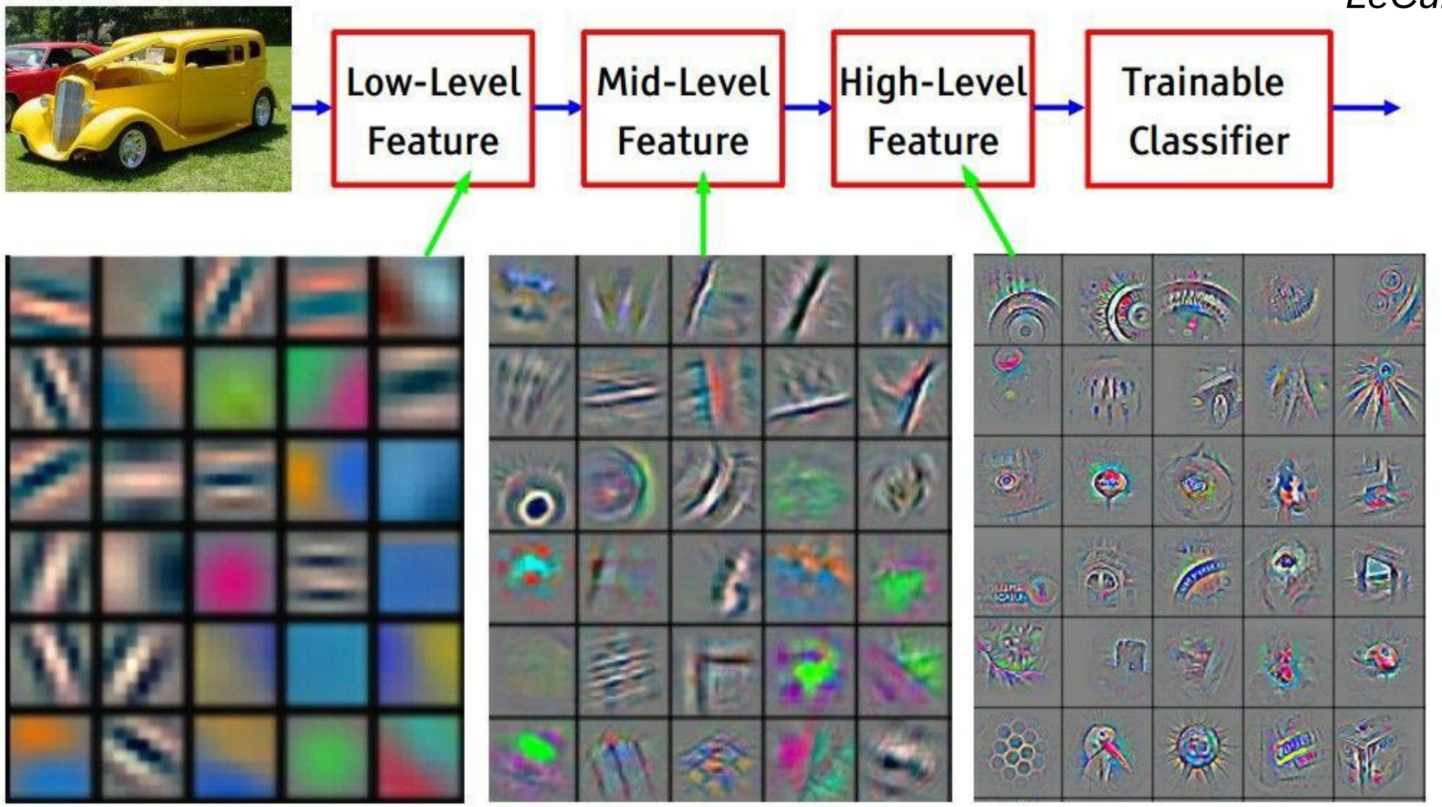
We stack these up to get a "new image" of size 28x28x6!

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



Preview:

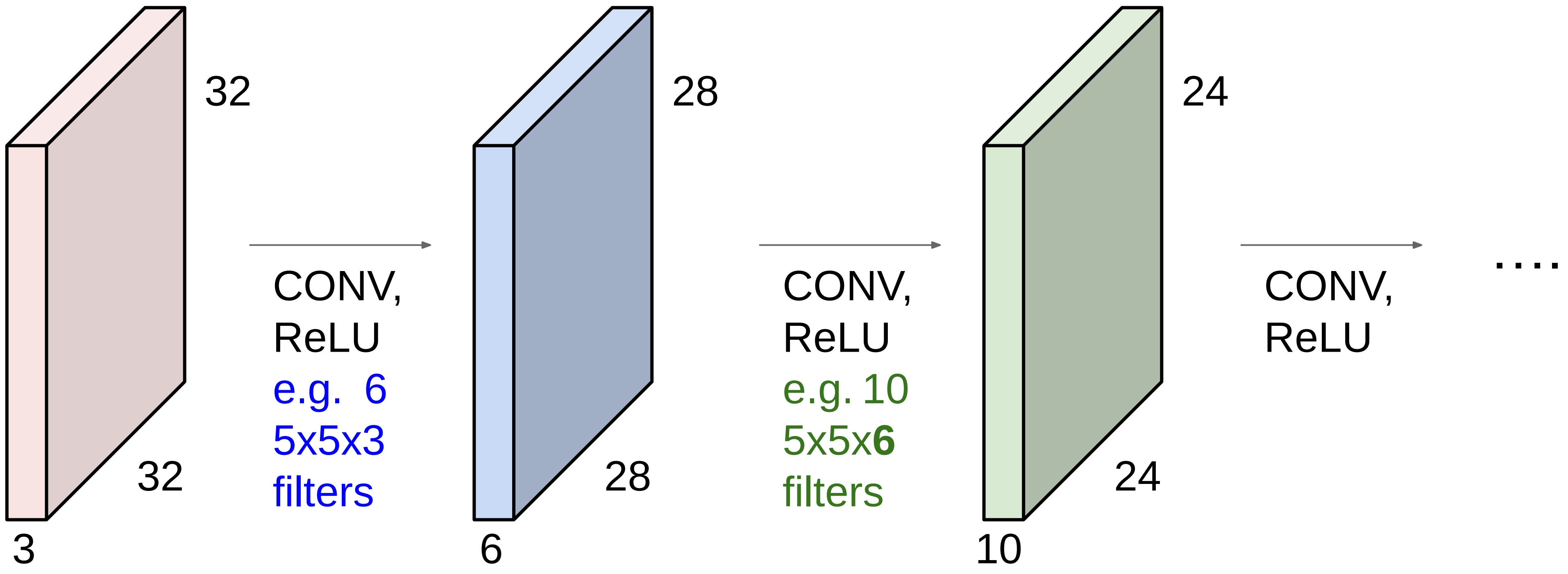
[From recent Yann LeCun slides]



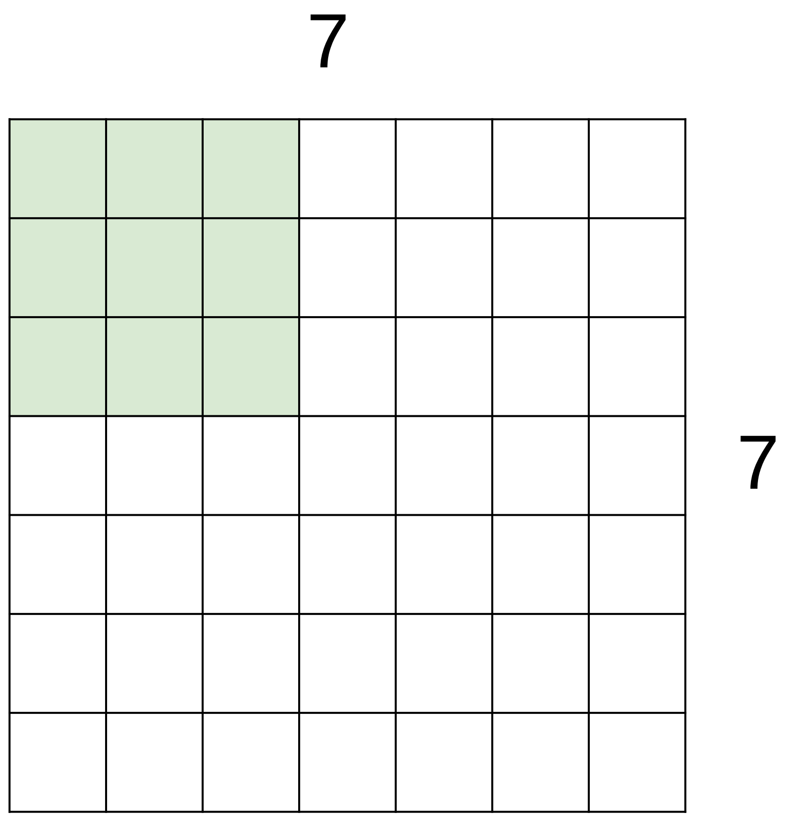
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

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.

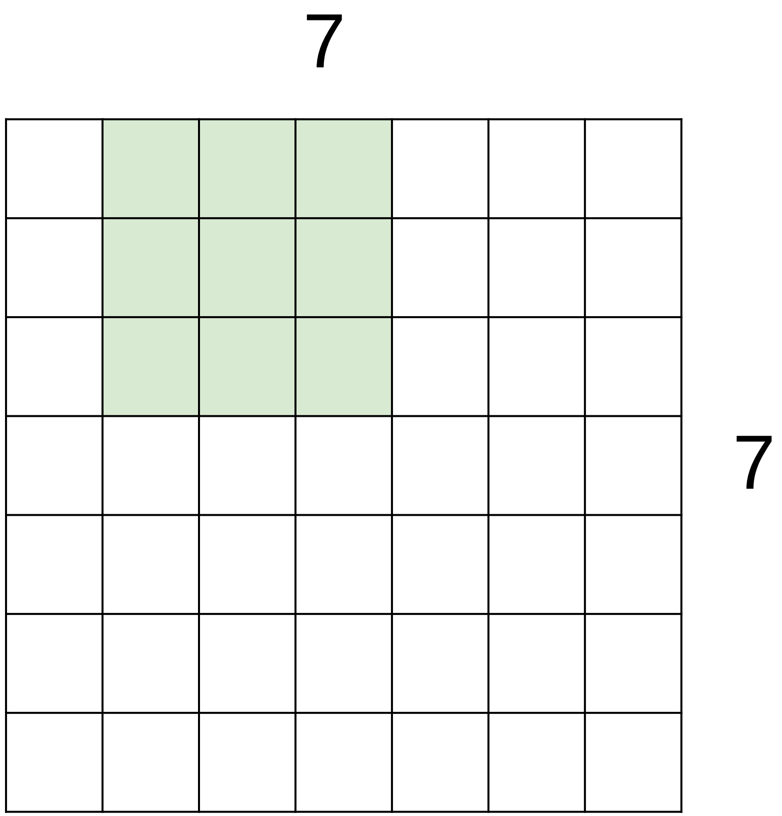


A closer look at spatial dimensions:



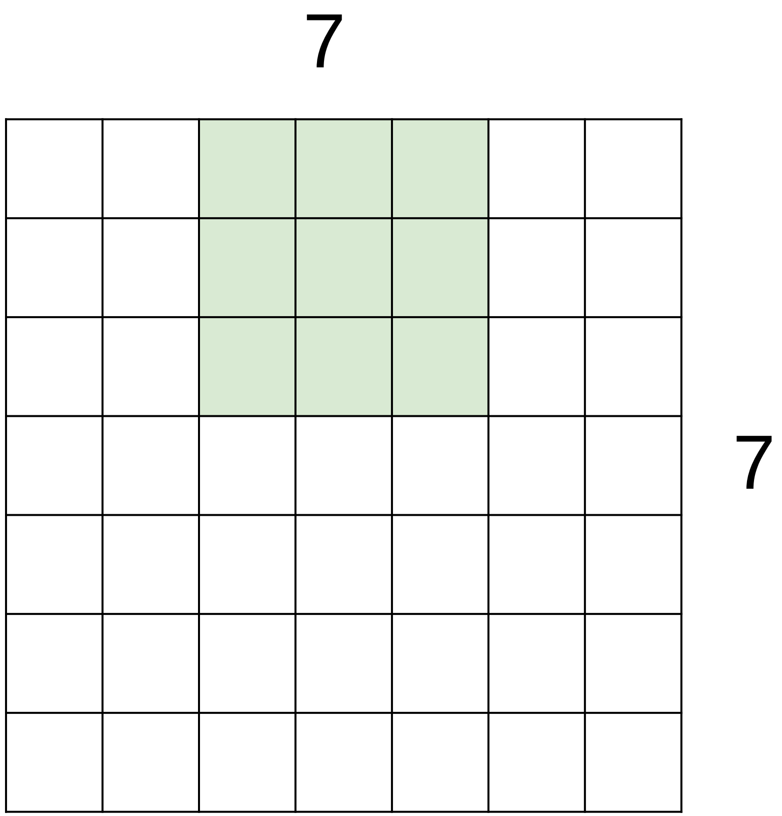
7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:



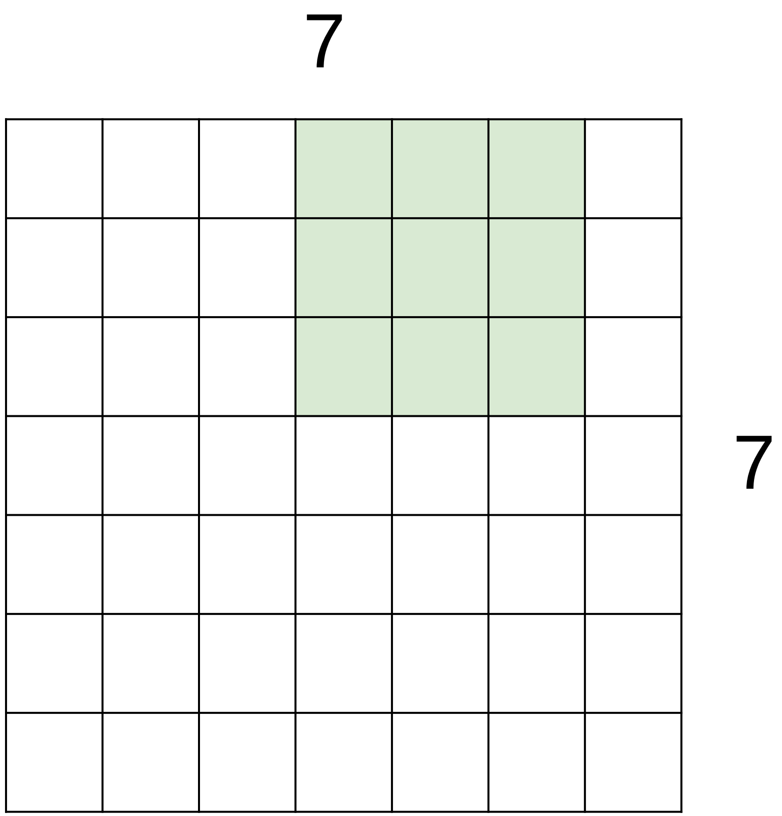
7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:



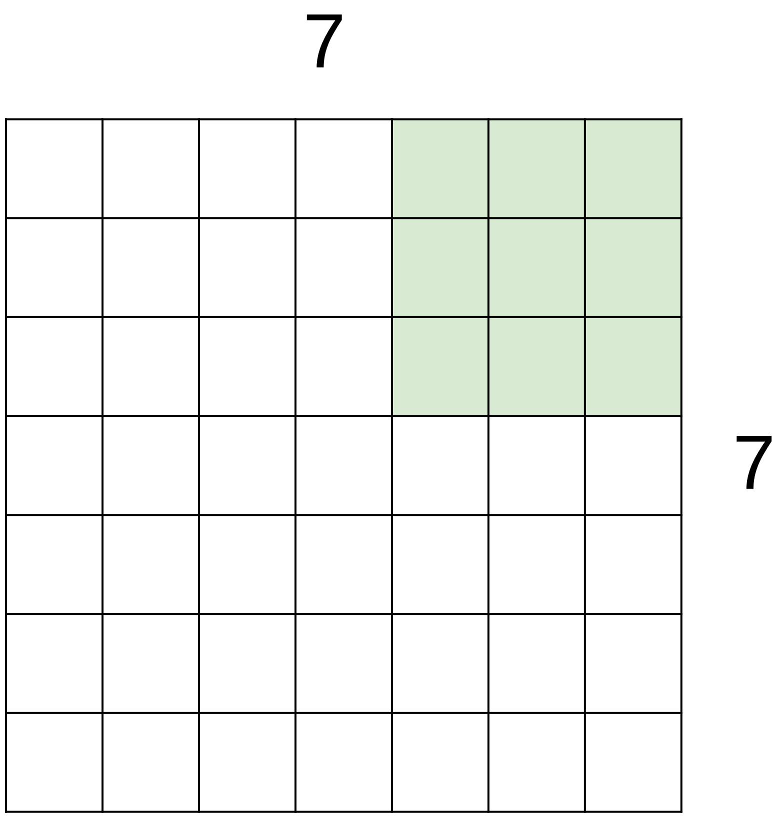
7x7 input (spatially)
assume 3x3 filter
applied **with stride 1**

A closer look at spatial dimensions:



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

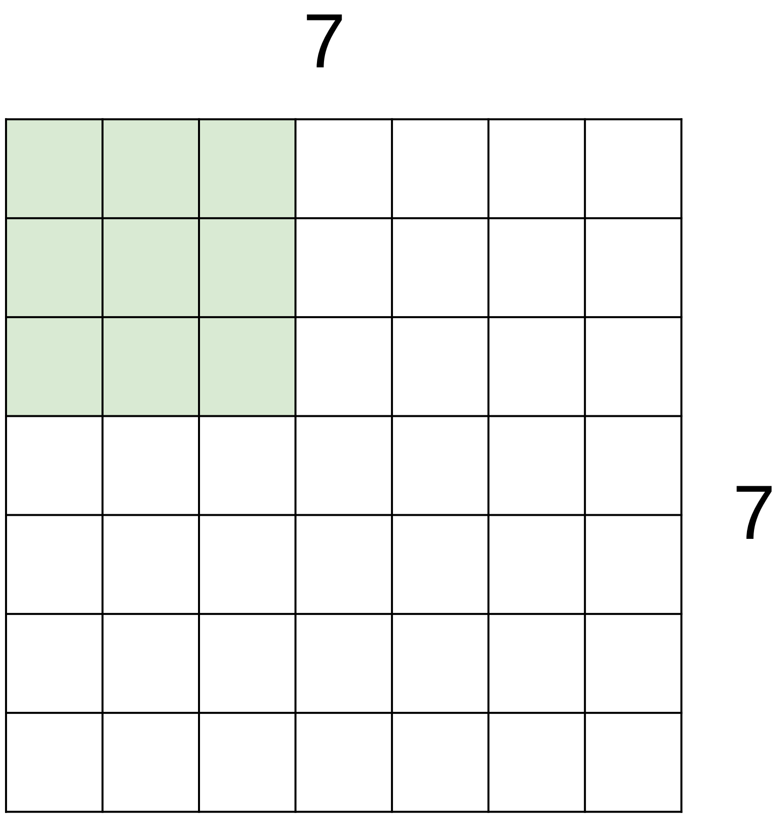
A closer look at spatial dimensions:



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

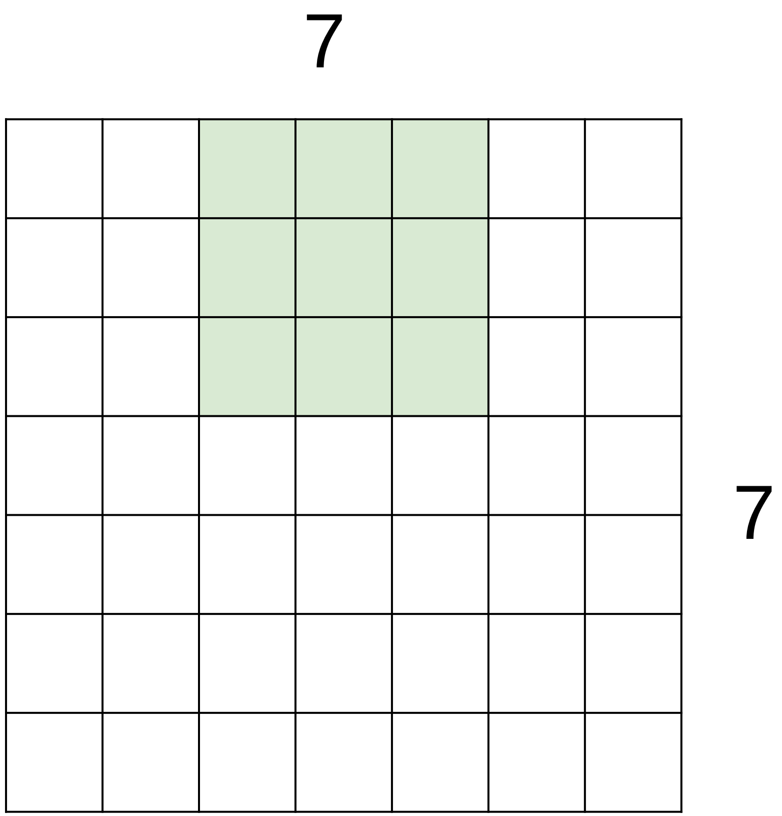
→ **5x5 output**

A closer look at spatial dimensions:



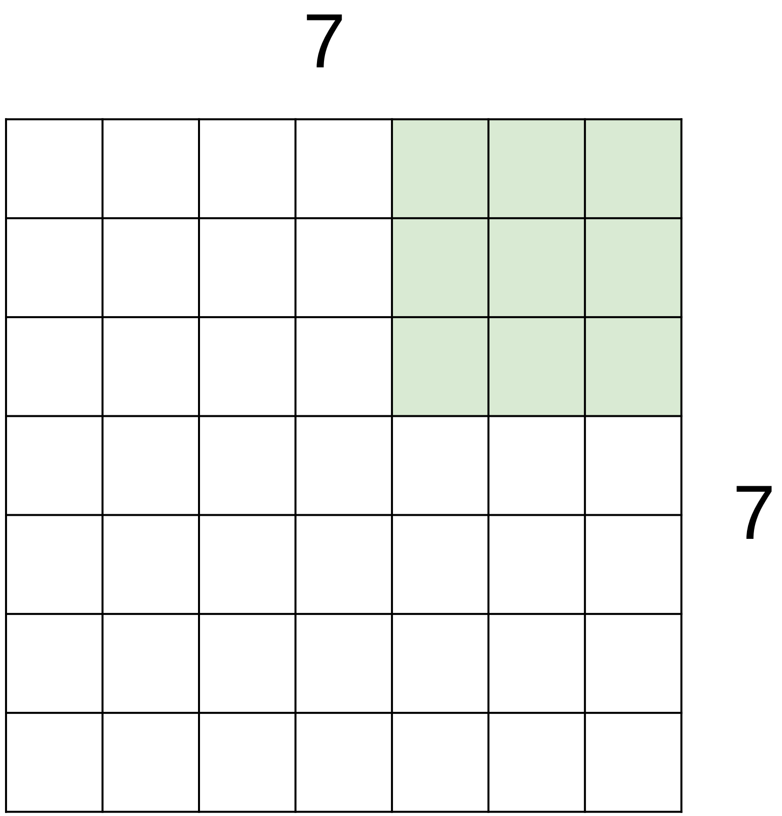
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:

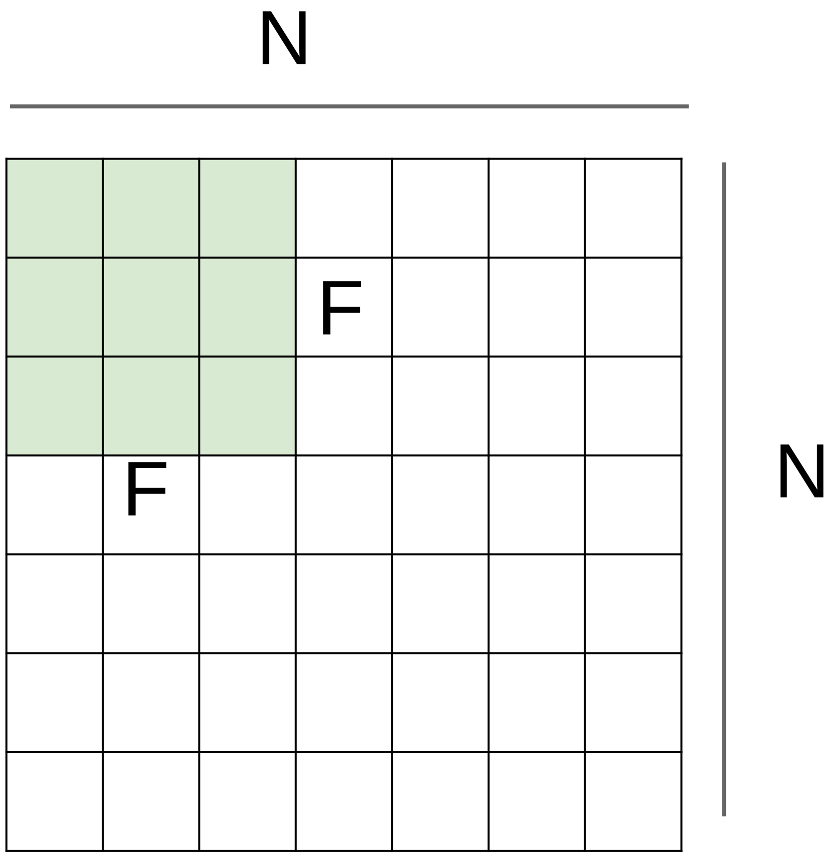


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



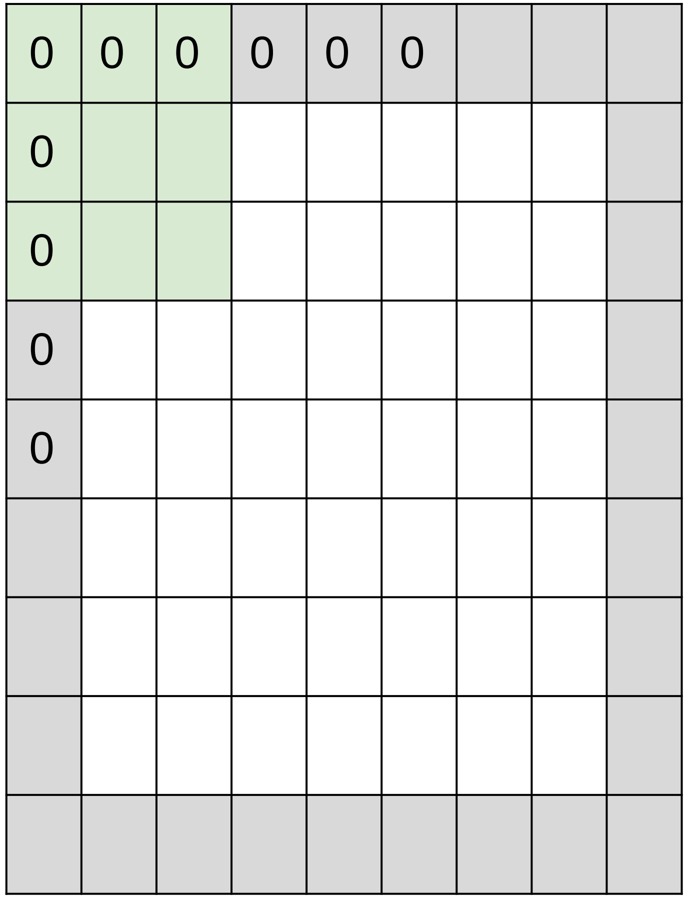
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
→ 3x3 output!



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
…

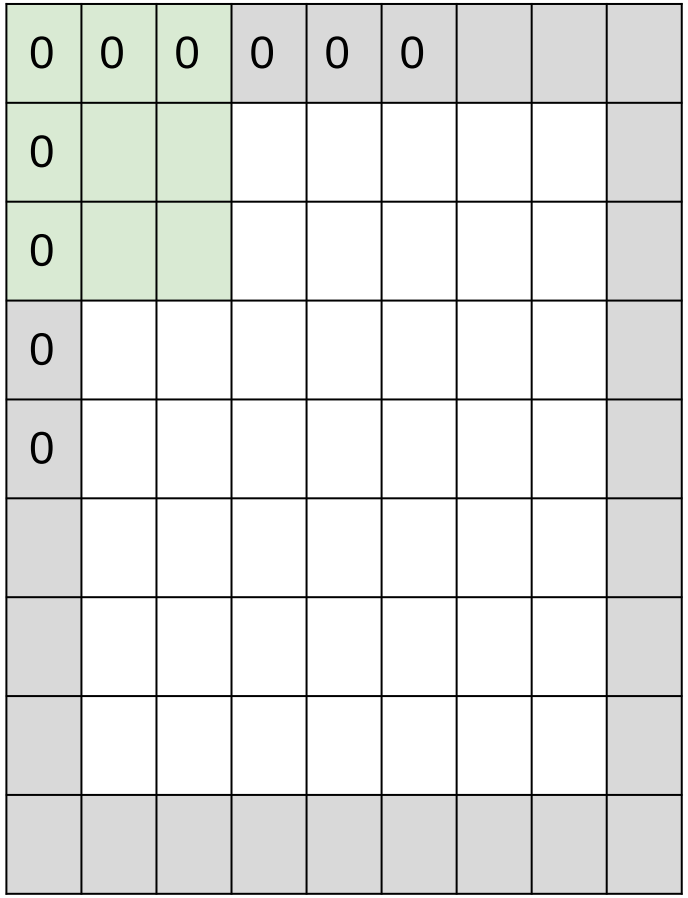
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

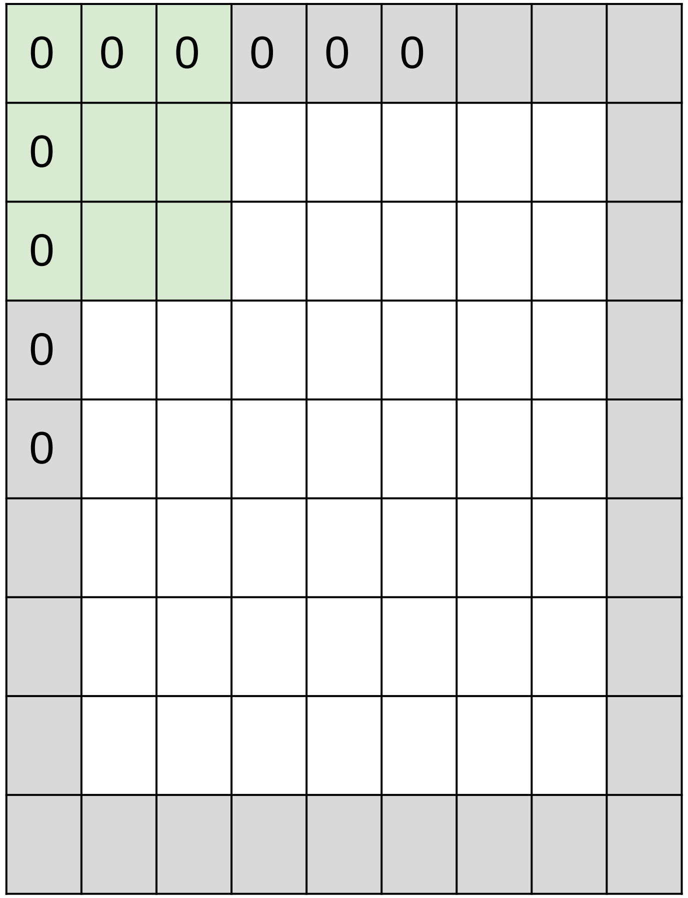
In practice: Common to zero pad the border



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

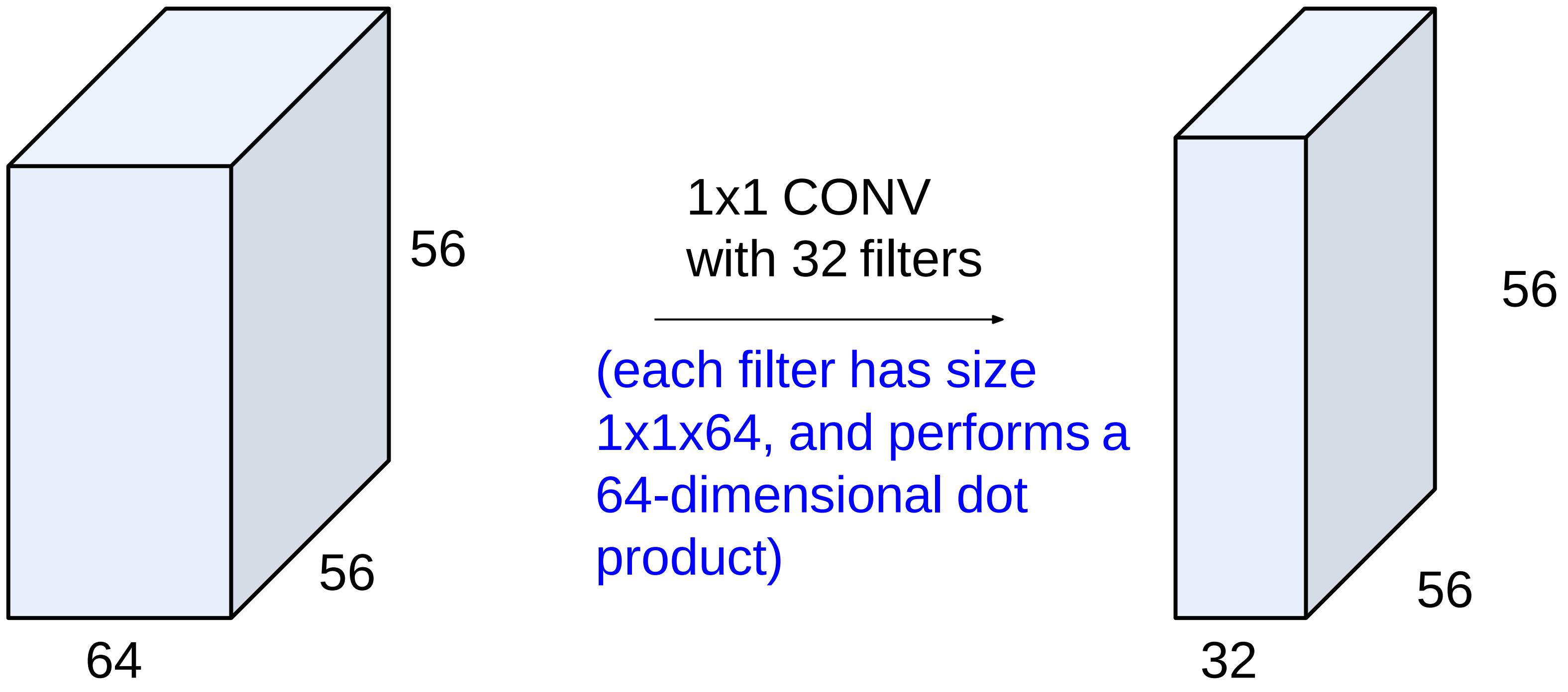


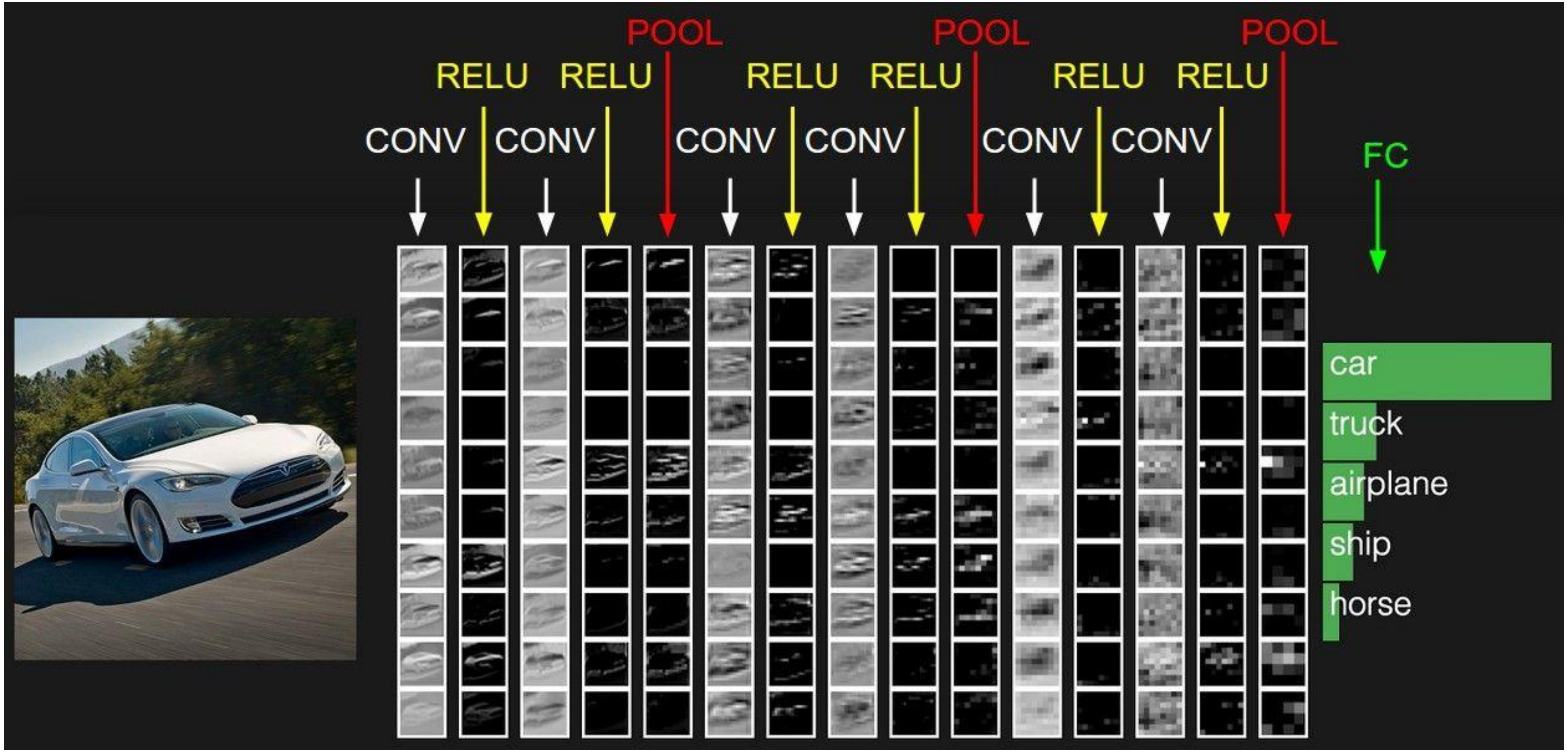
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

In general, common to see CONV layers with stride 1, filters of size FxF, 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

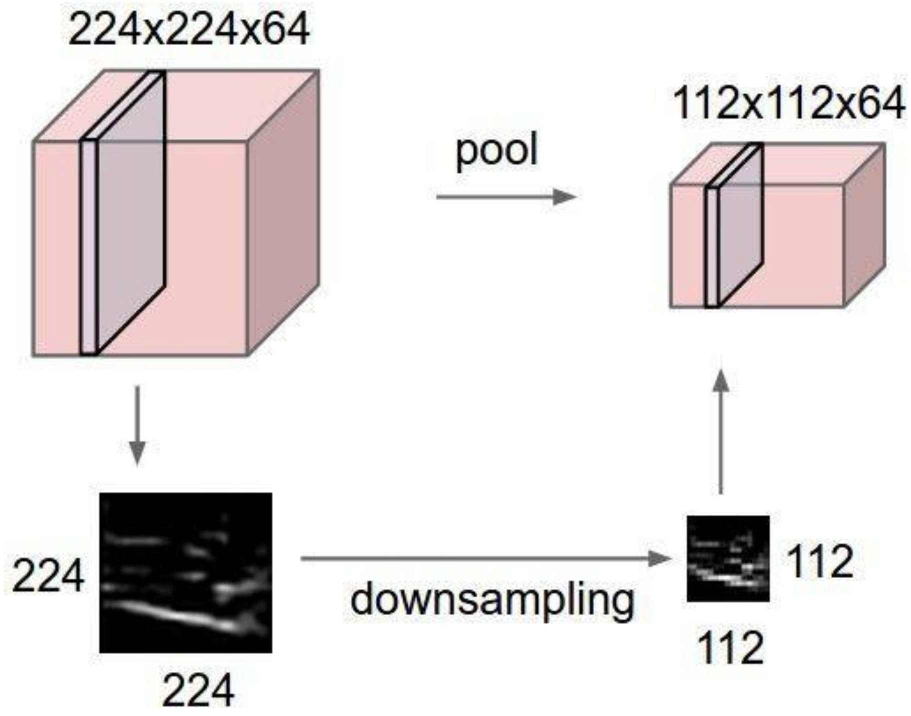
1x1 convolution layers



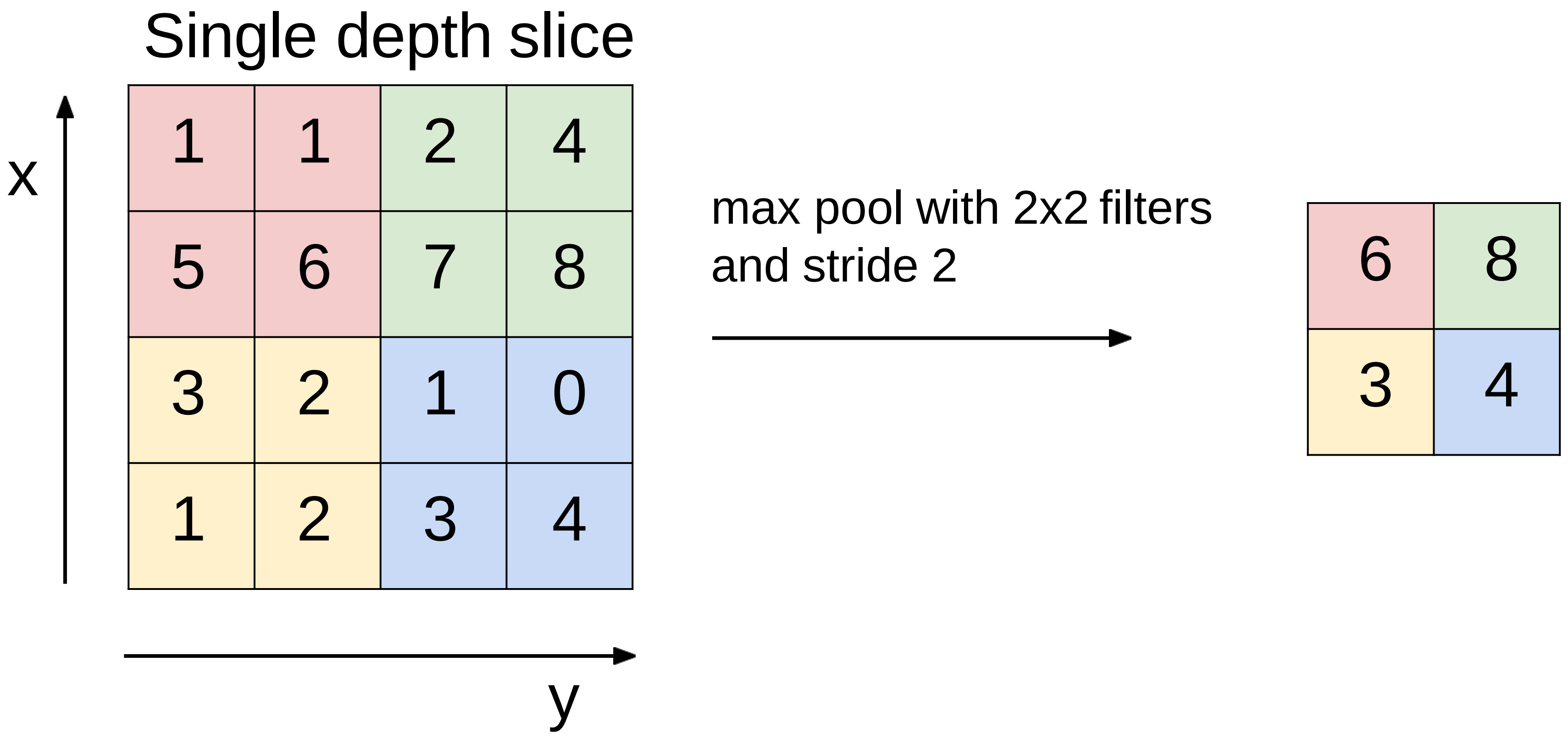

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

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

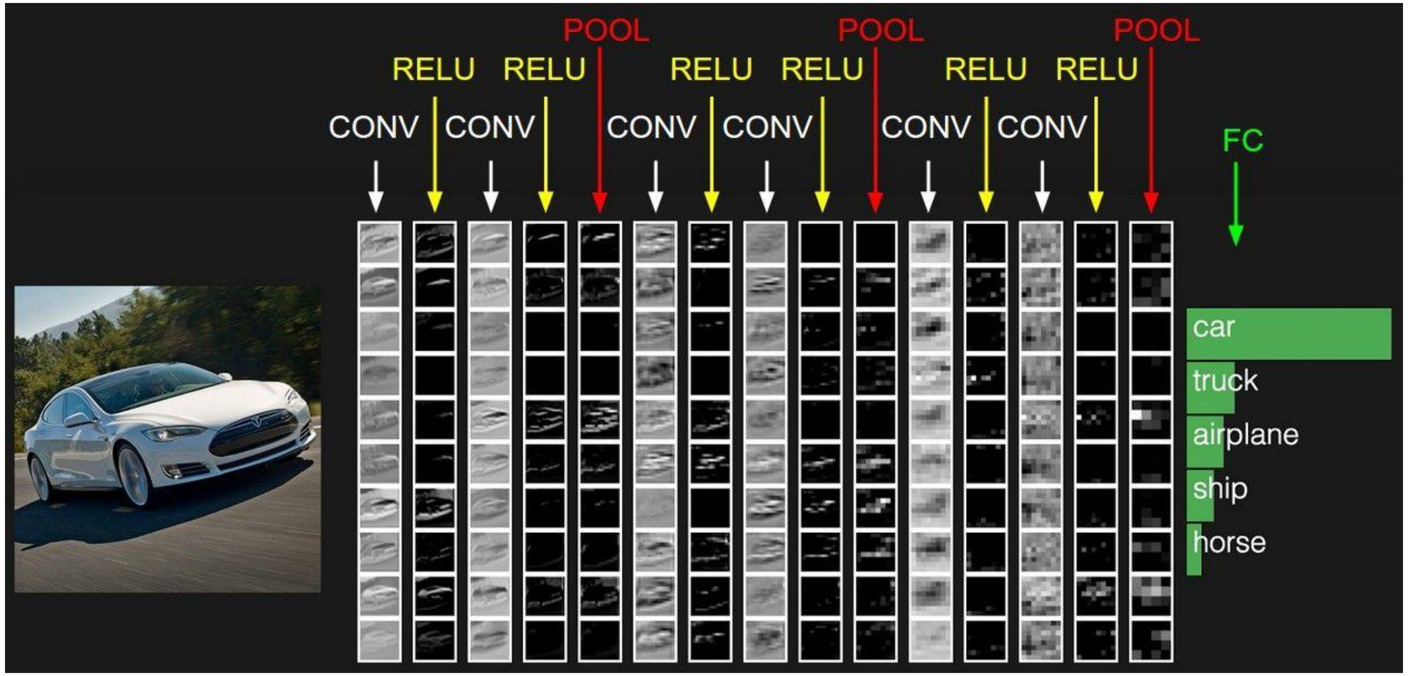


MAX Pooling

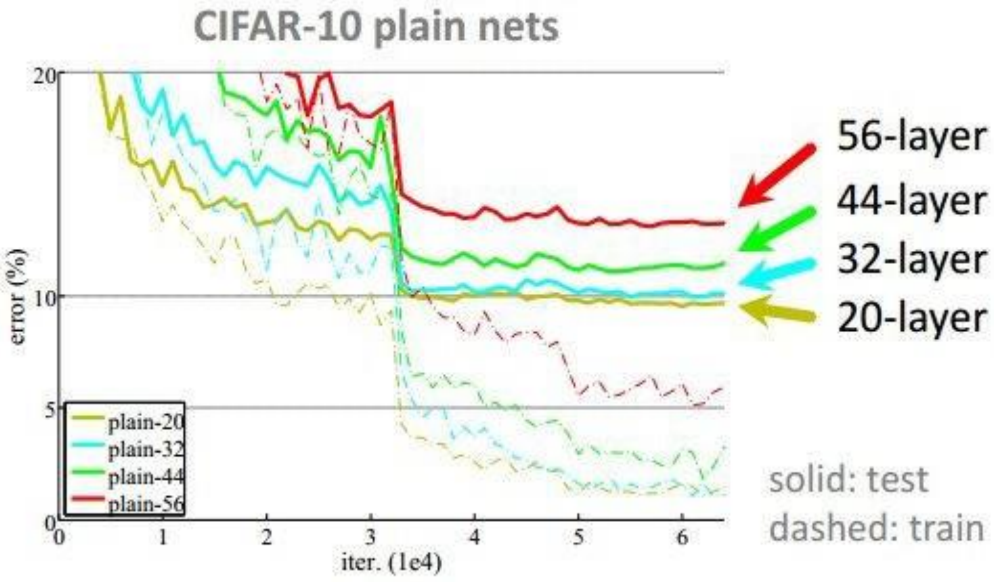

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

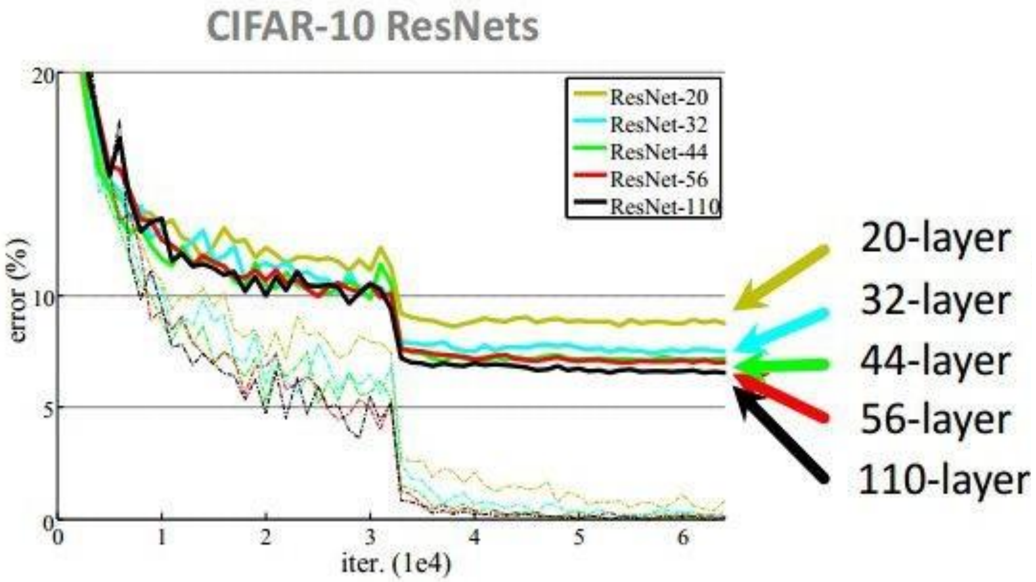
Fully Connected Layer (FC layer)

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



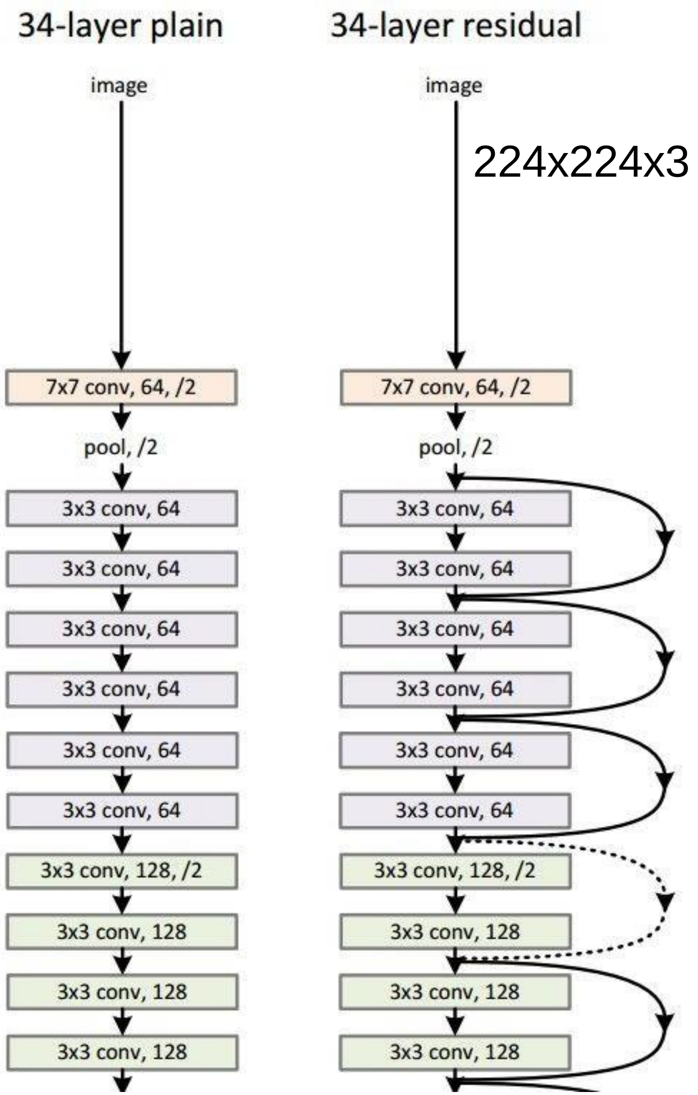
Residual Networks





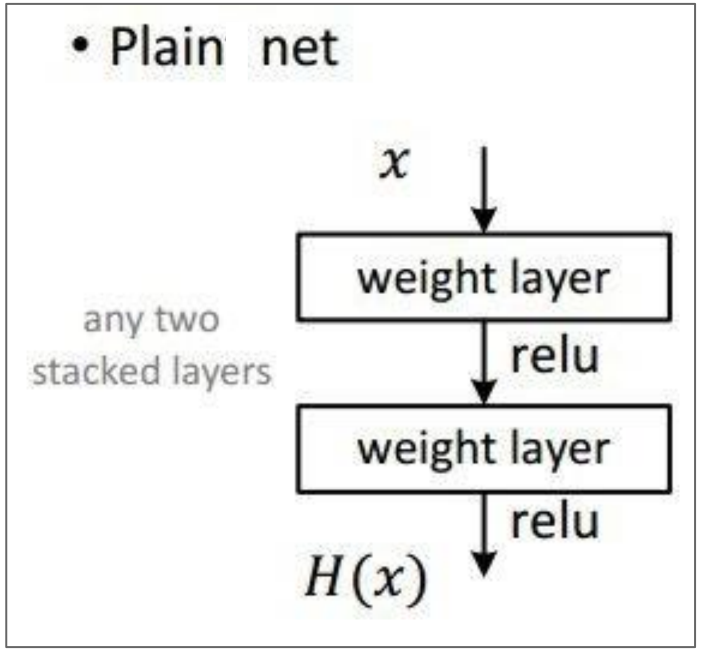
[He at al., 2015]

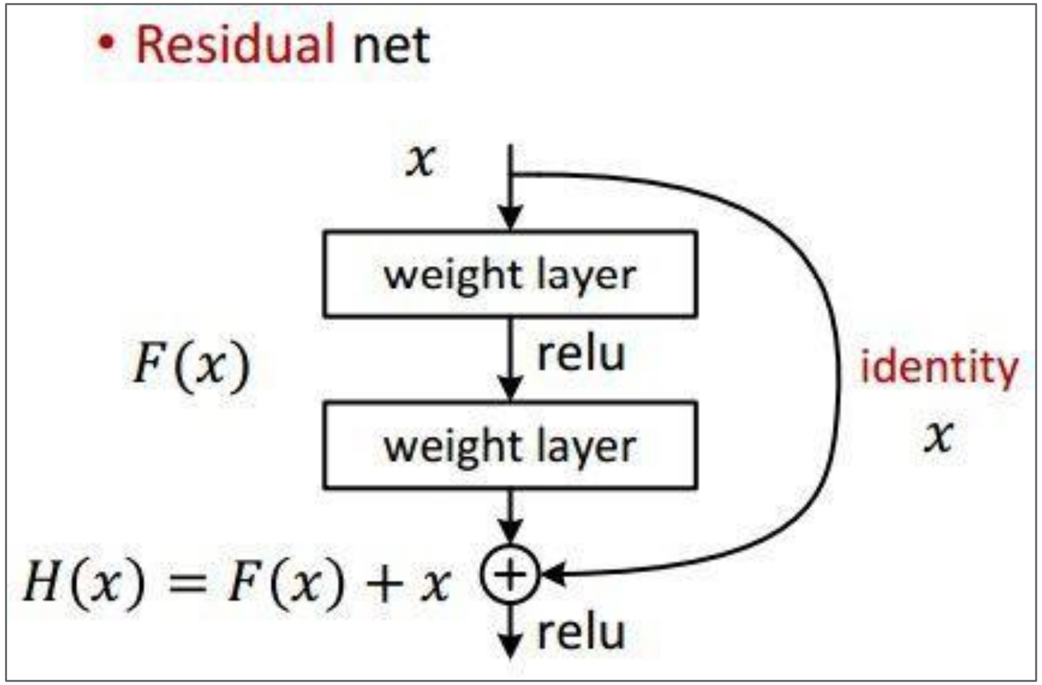
Residual Networks



[He at al., 2015]

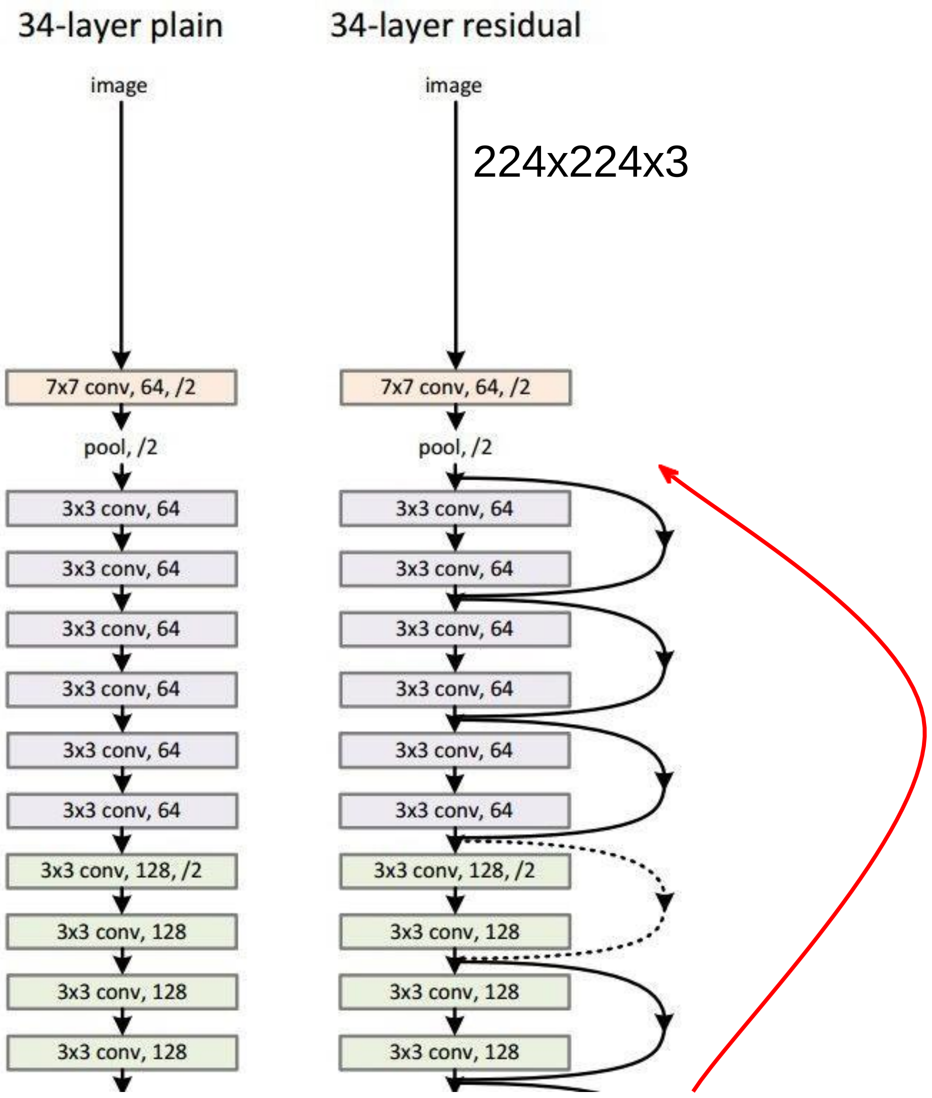
Residual Networks





[He at al., 2015]

Residual Networks

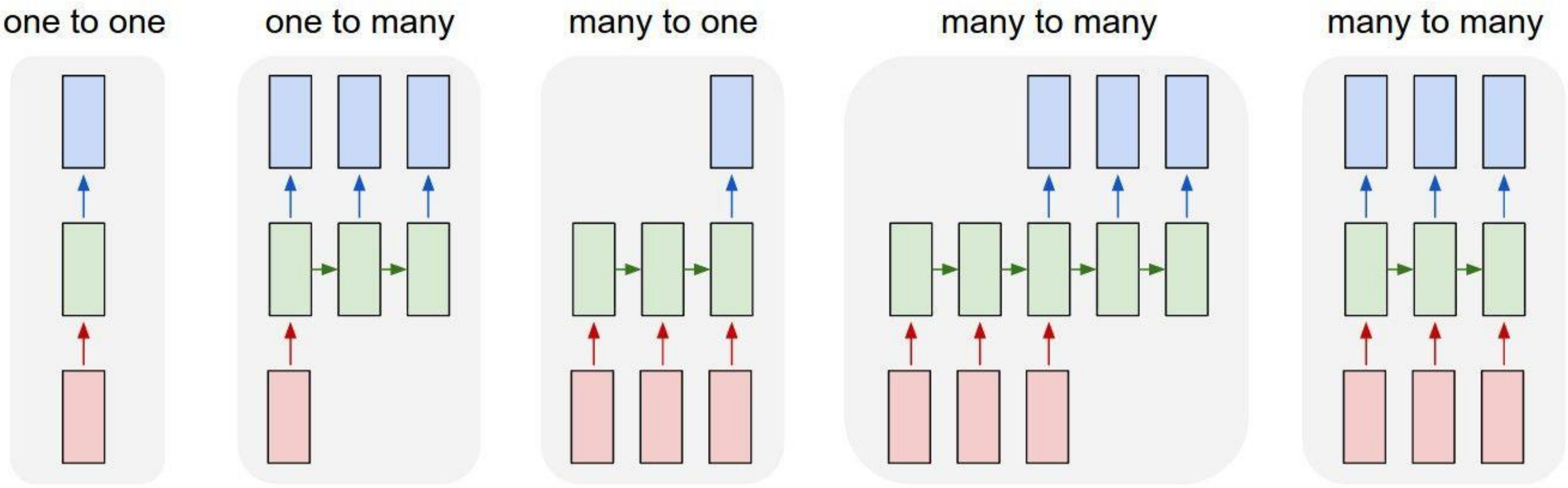


During back-prop, gradient is flows through layers without vanishing

[He at al., 2015]

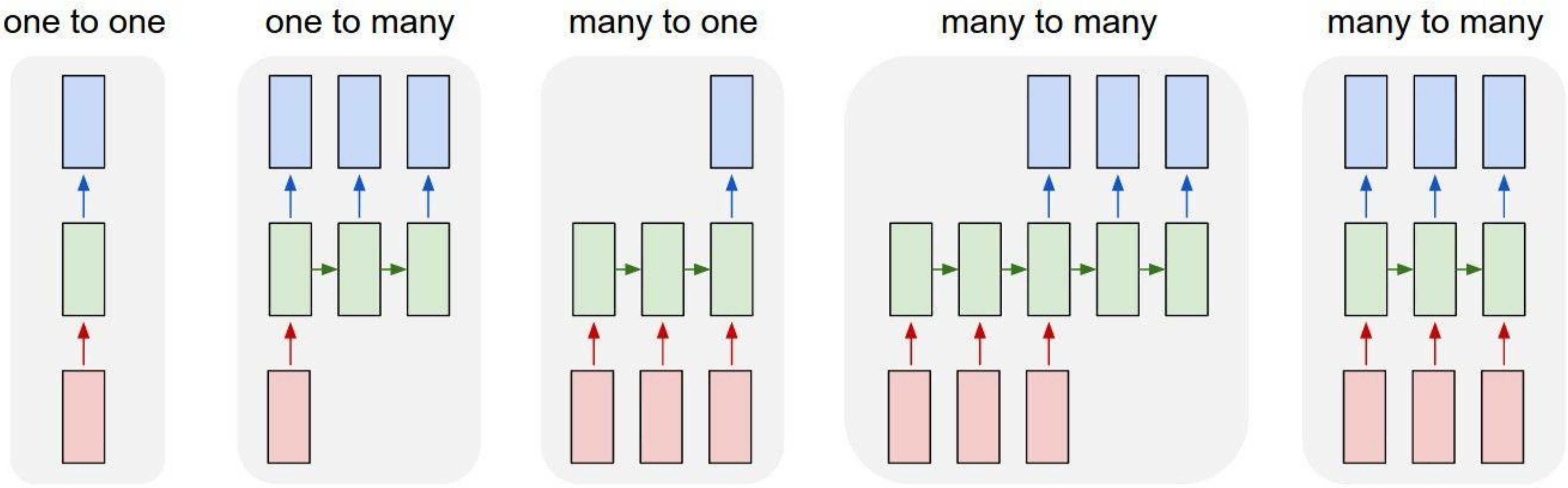
Recurrent Neural Networks

Recurrent Networks offer a lot of flexibility:



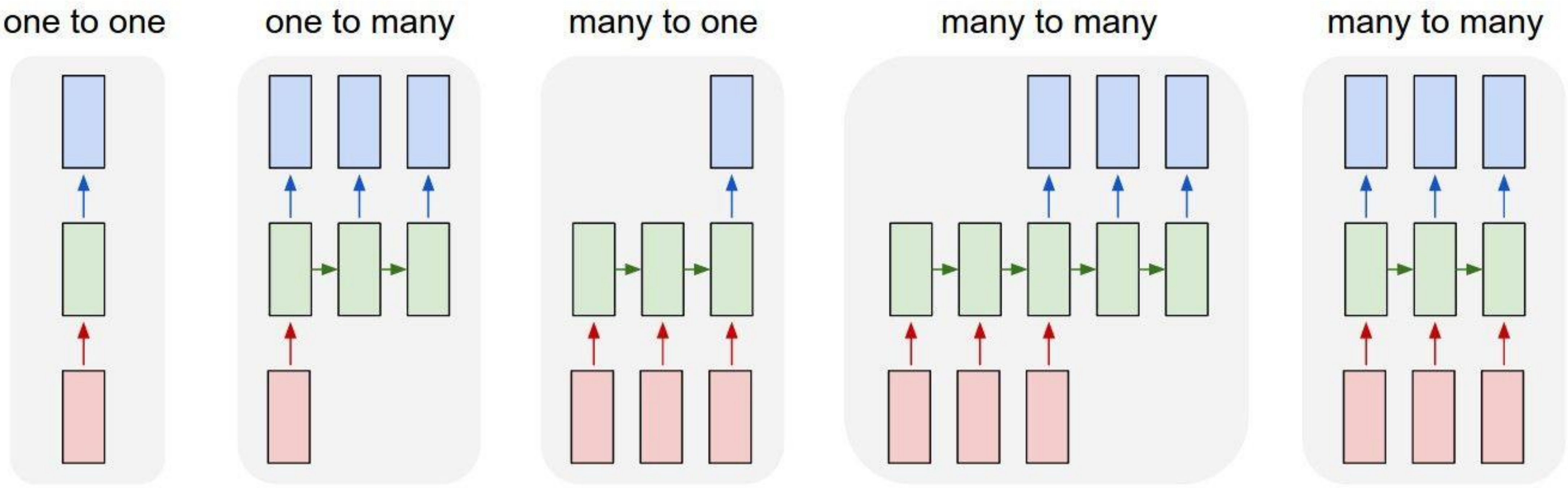
Vanilla Neural Networks

Recurrent Networks offer a lot of flexibility:



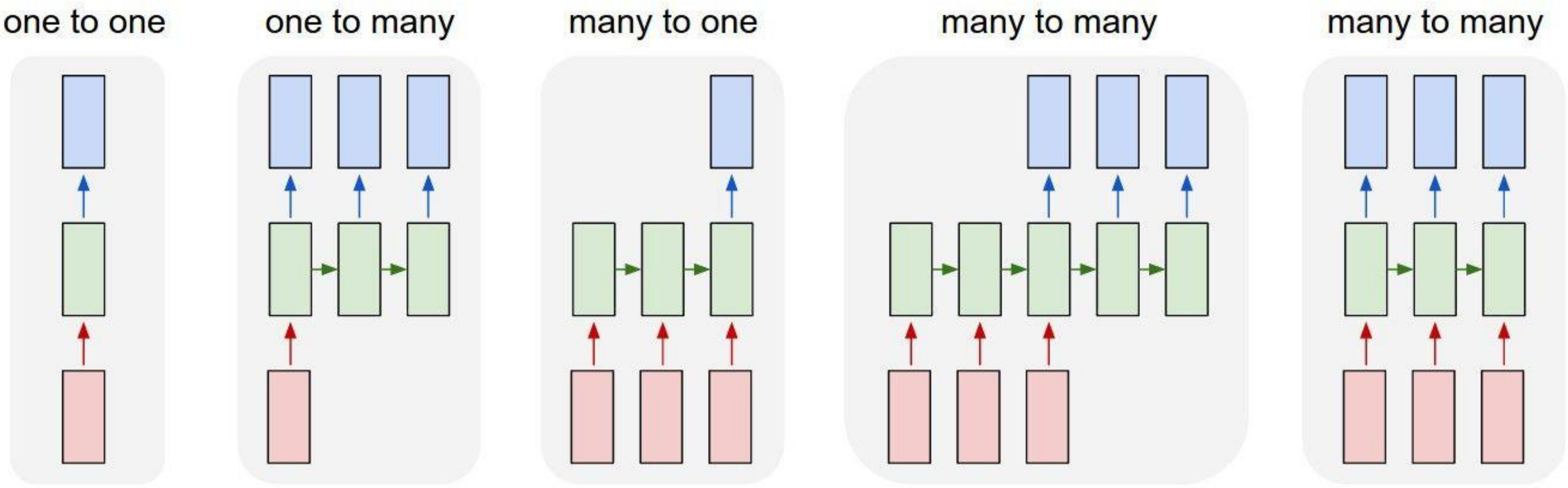
e.g. **Image Captioning**
image -> sequence of words

Recurrent Networks offer a lot of flexibility:



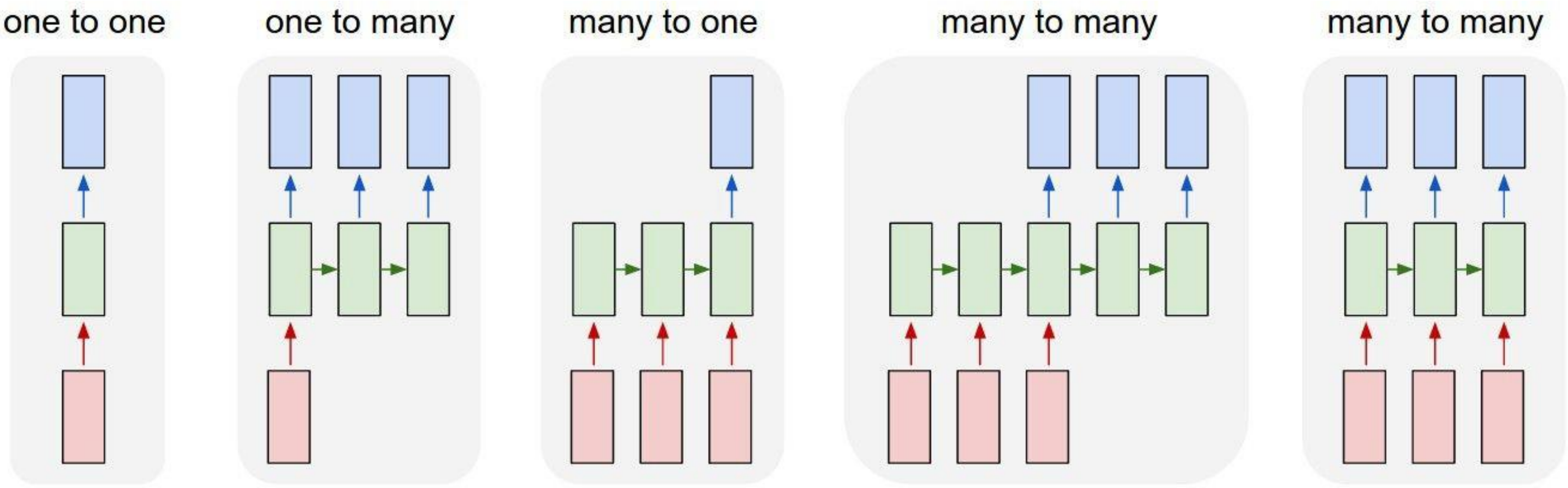
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility:



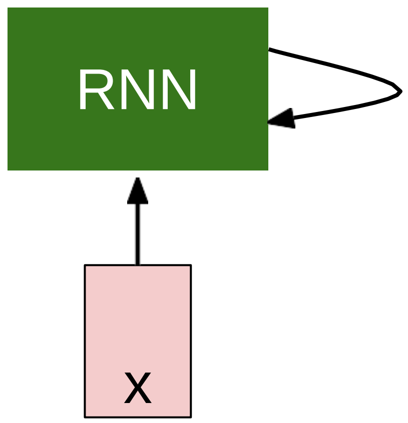
e.g. **Machine Translation**
seq of words -> seq of words

Recurrent Networks offer a lot of flexibility:

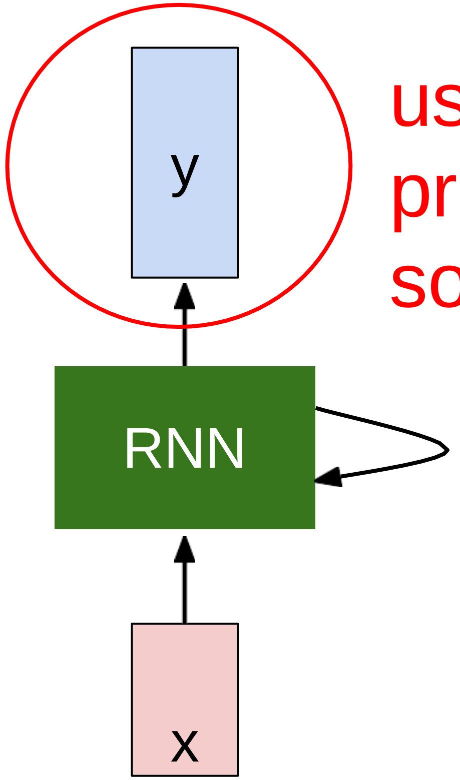


e.g. **Video classification on frame level**

Recurrent Neural Network



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:

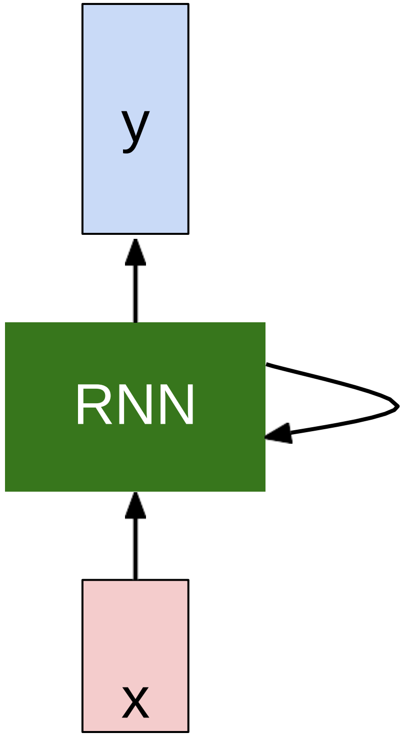
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

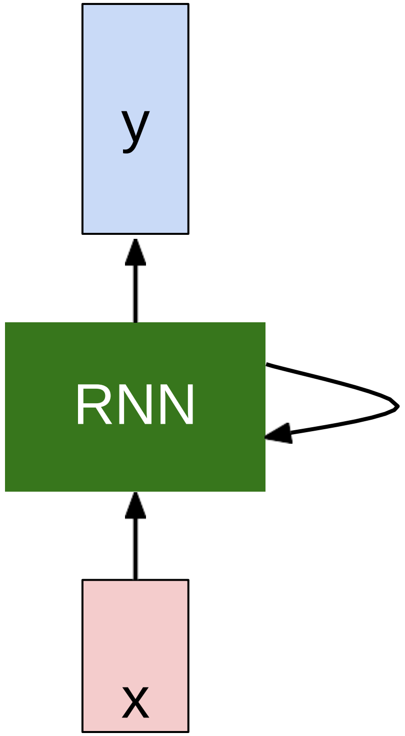
some function with parameters W



Recurrent Neural Network

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

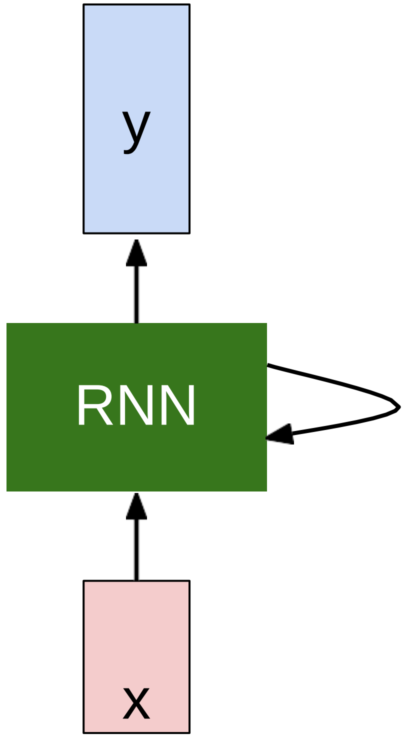
$$h_t = f_W(h_{t-1}, x_t)$$



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

(Vanilla) Recurrent Neural Network

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



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

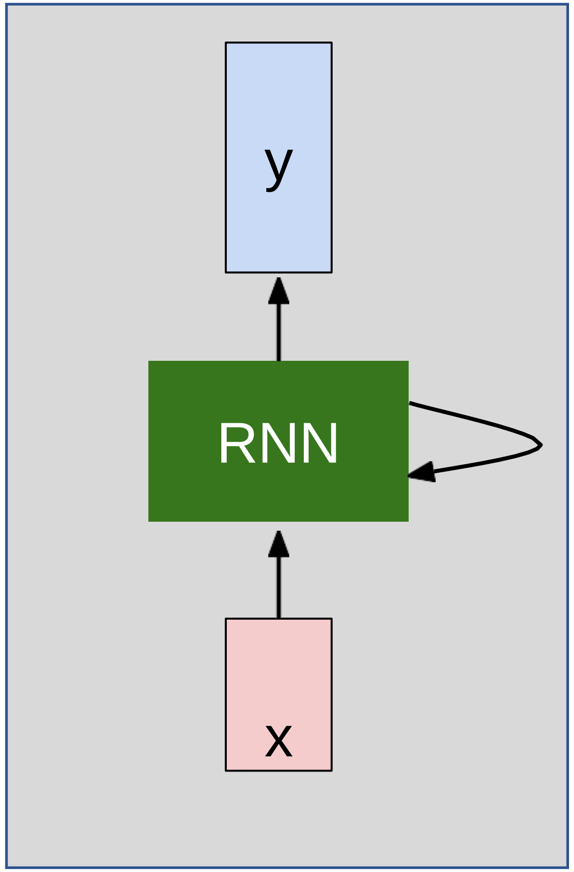
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

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. $|V| = 30$

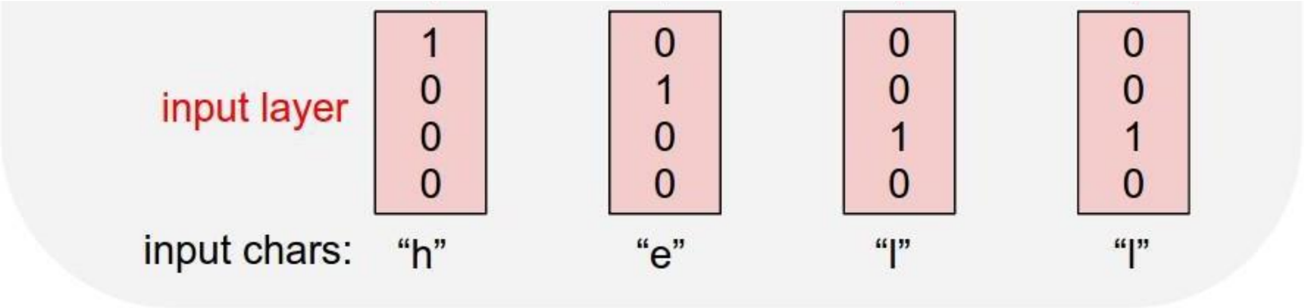
'a': x^T = [1,0,0, ..., 0]
'b': x^T = [0,1,0, ..., 0]
'c': x^T = [0,0,1, ..., 0]
.
.
.
'.': x^T = [0,0,0, ..., 1]

Slide adapted from Antonio Bonafonte

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



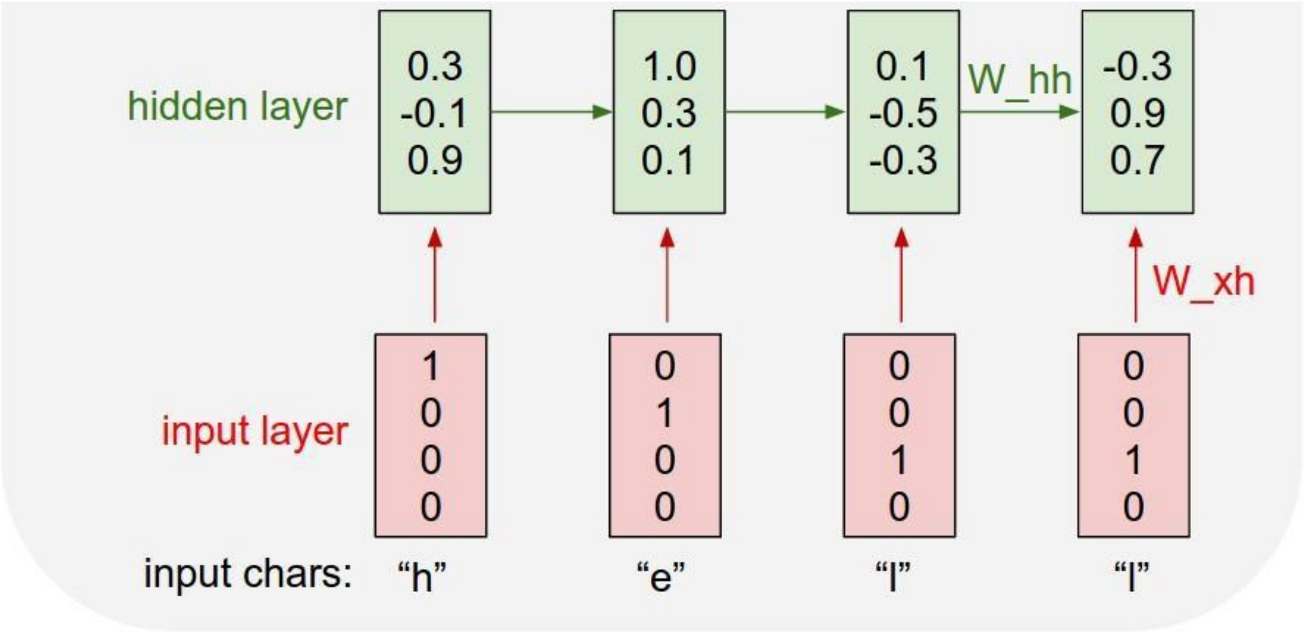
Objective:
Predict the next character given the previous characters

Character-level language model example

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

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**



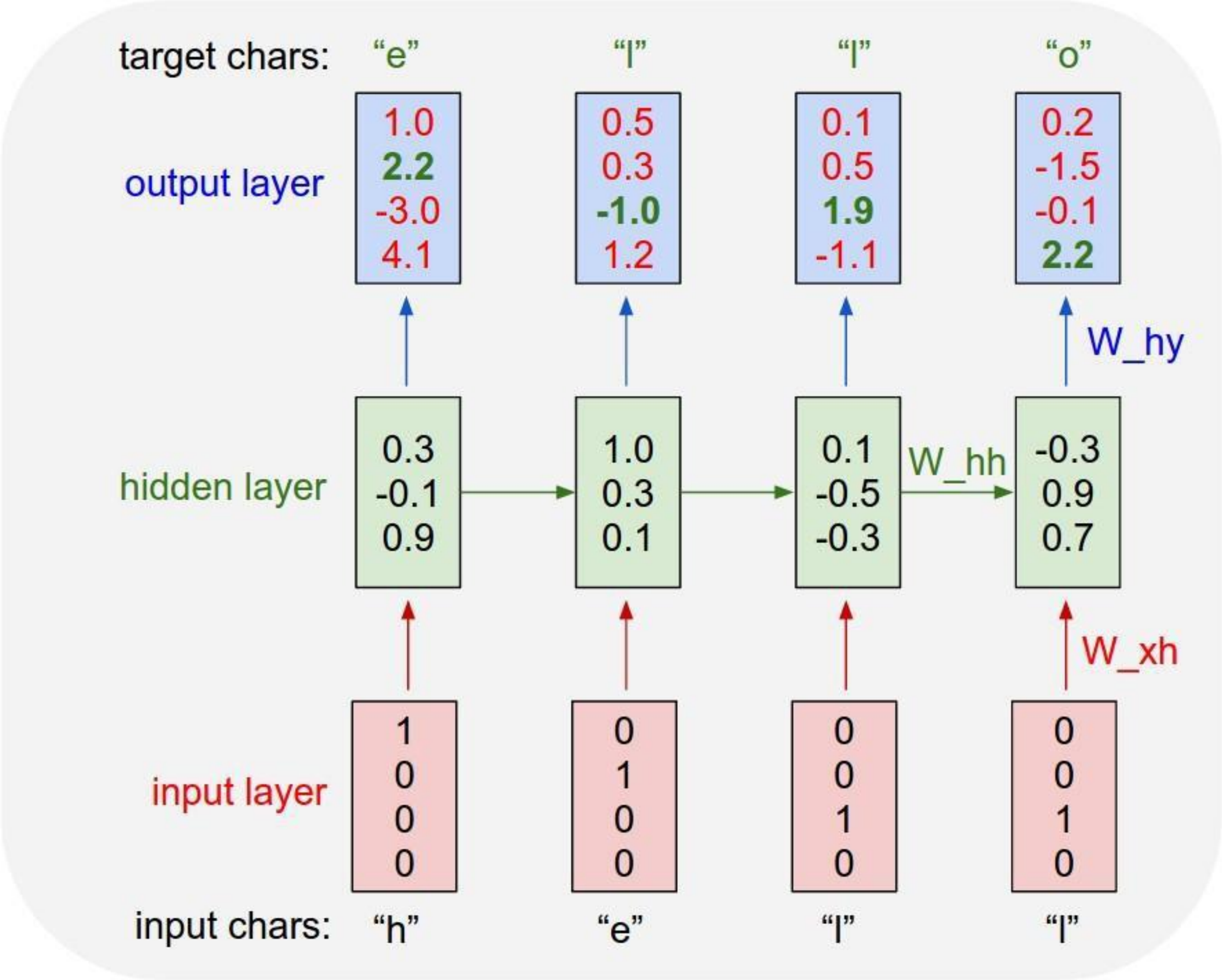
Objective:

Predict the next character given the previous characters

Character-level language model example

Vocabulary:
[h,e,l,o]

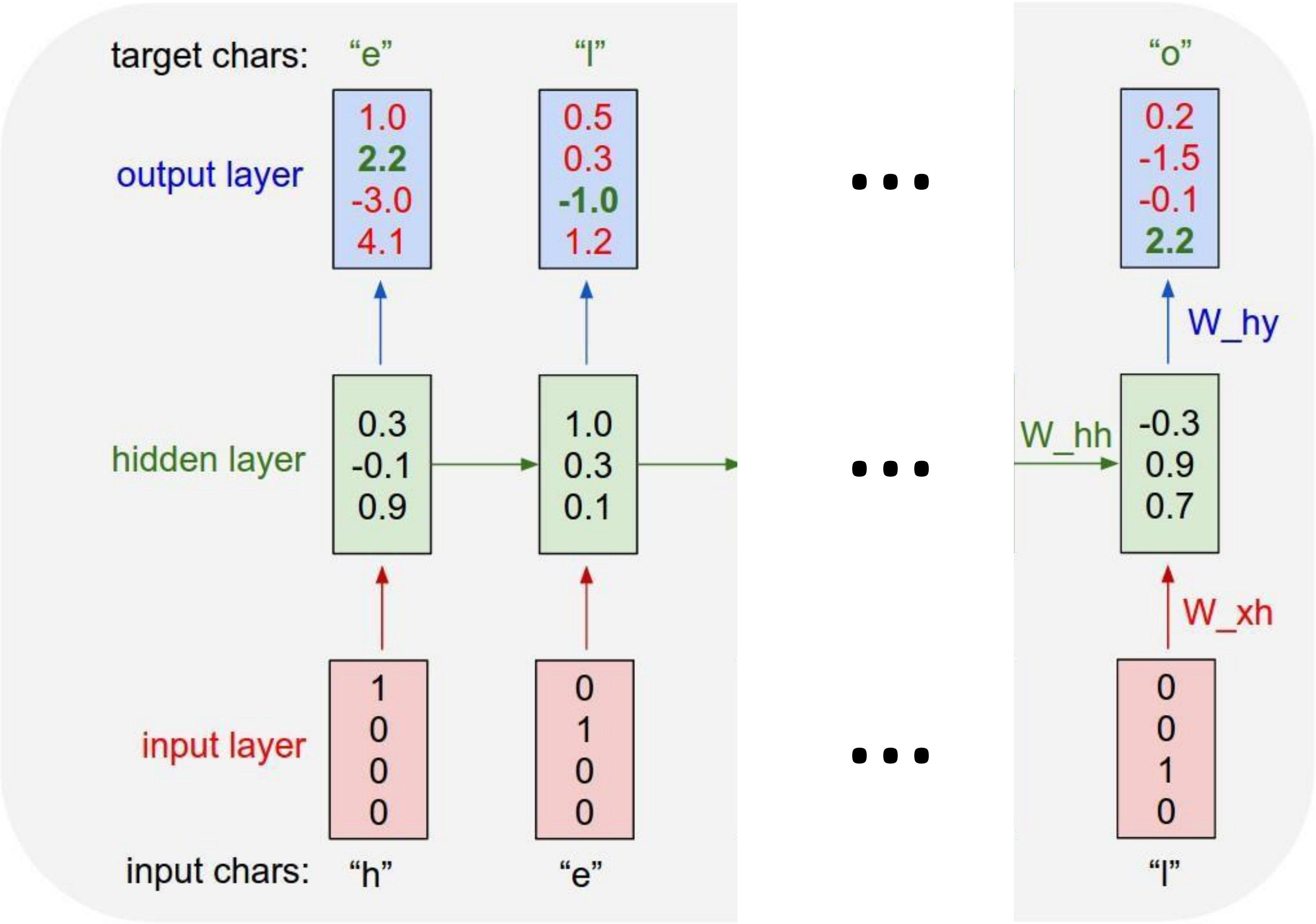
Example training sequence:
“hello”

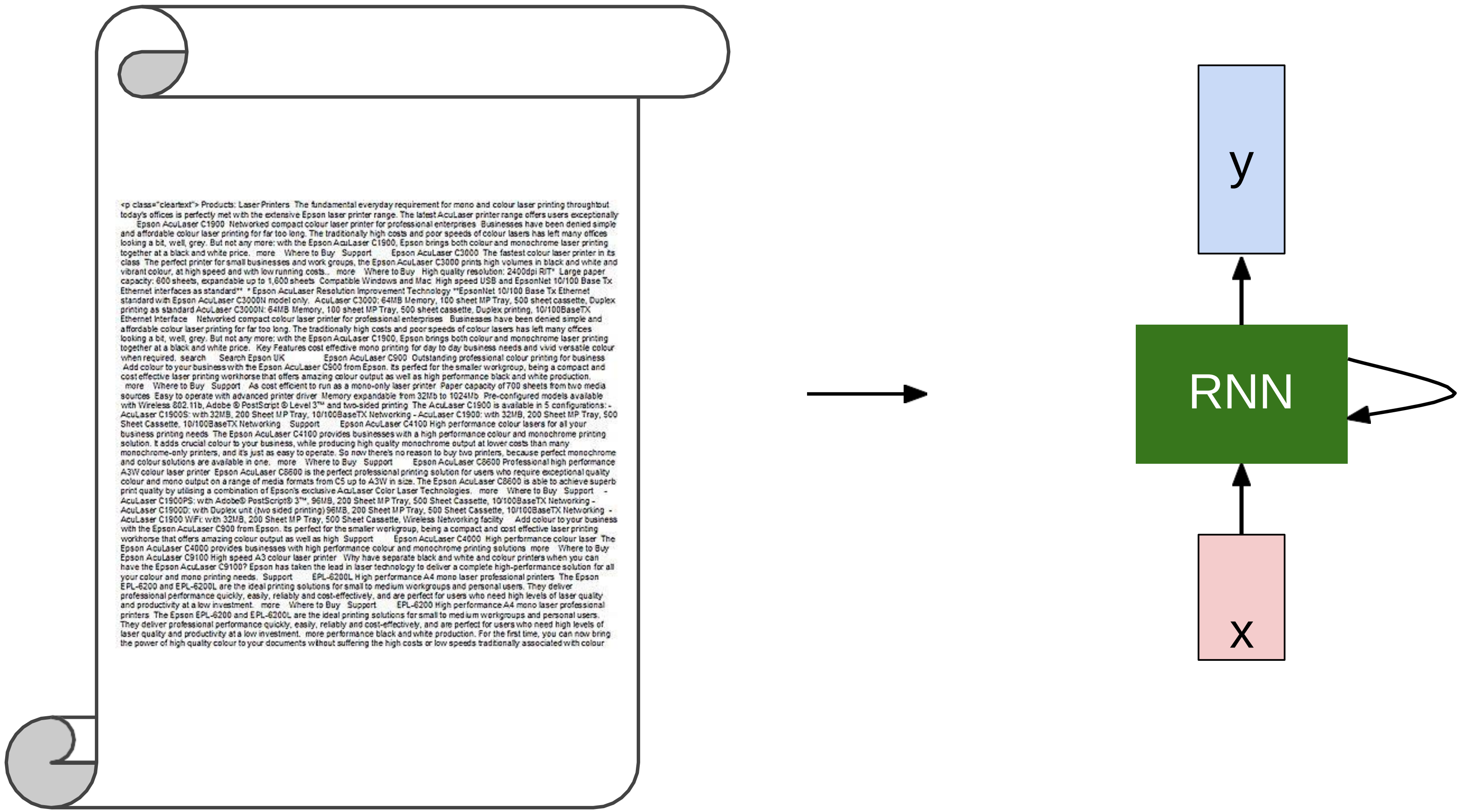


Objective:
Predict the next character given the previous characters

Varying length input

Forward and backward passes are conducted on consequent subsequences iteratively




y
RNN
x

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 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy 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 ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened 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 srain 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 nues 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 aps, 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:

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

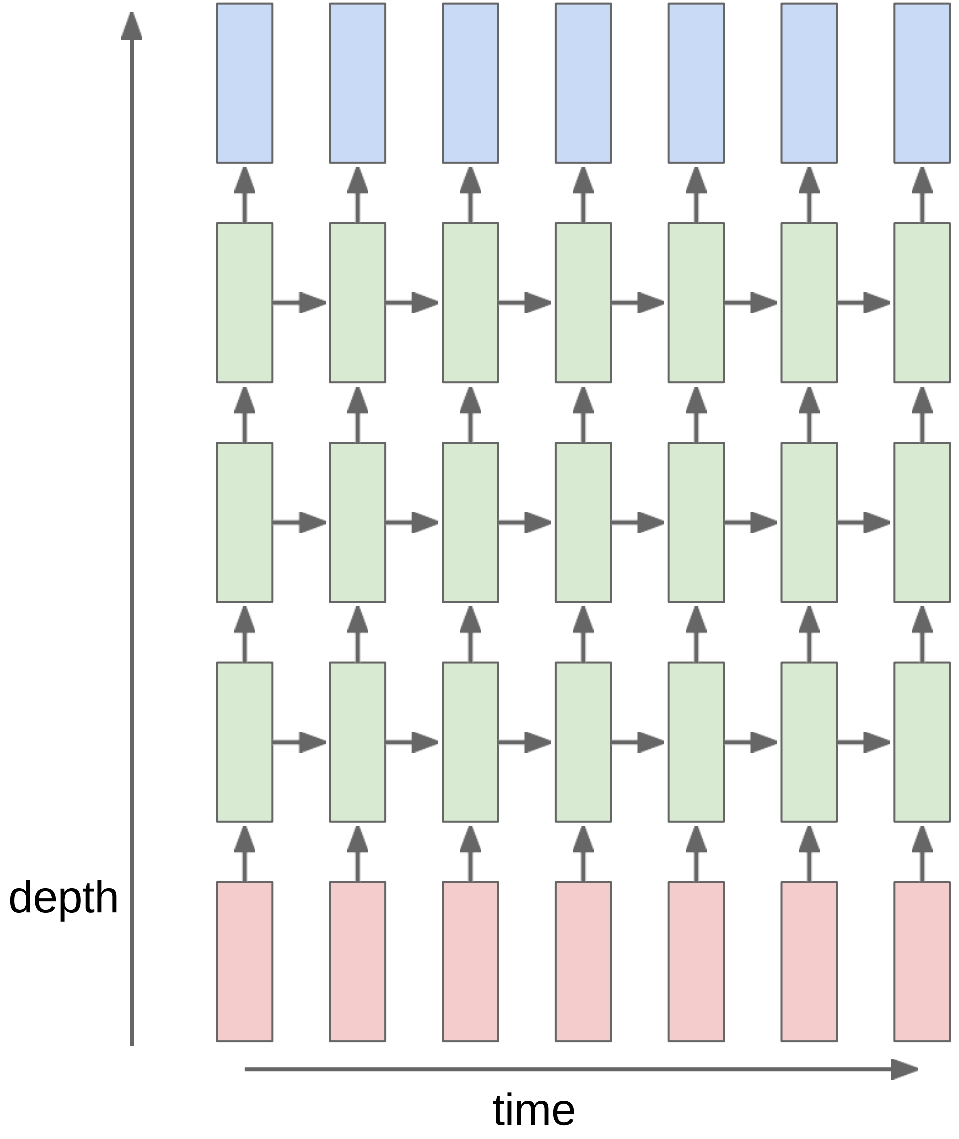
$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

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:

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

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

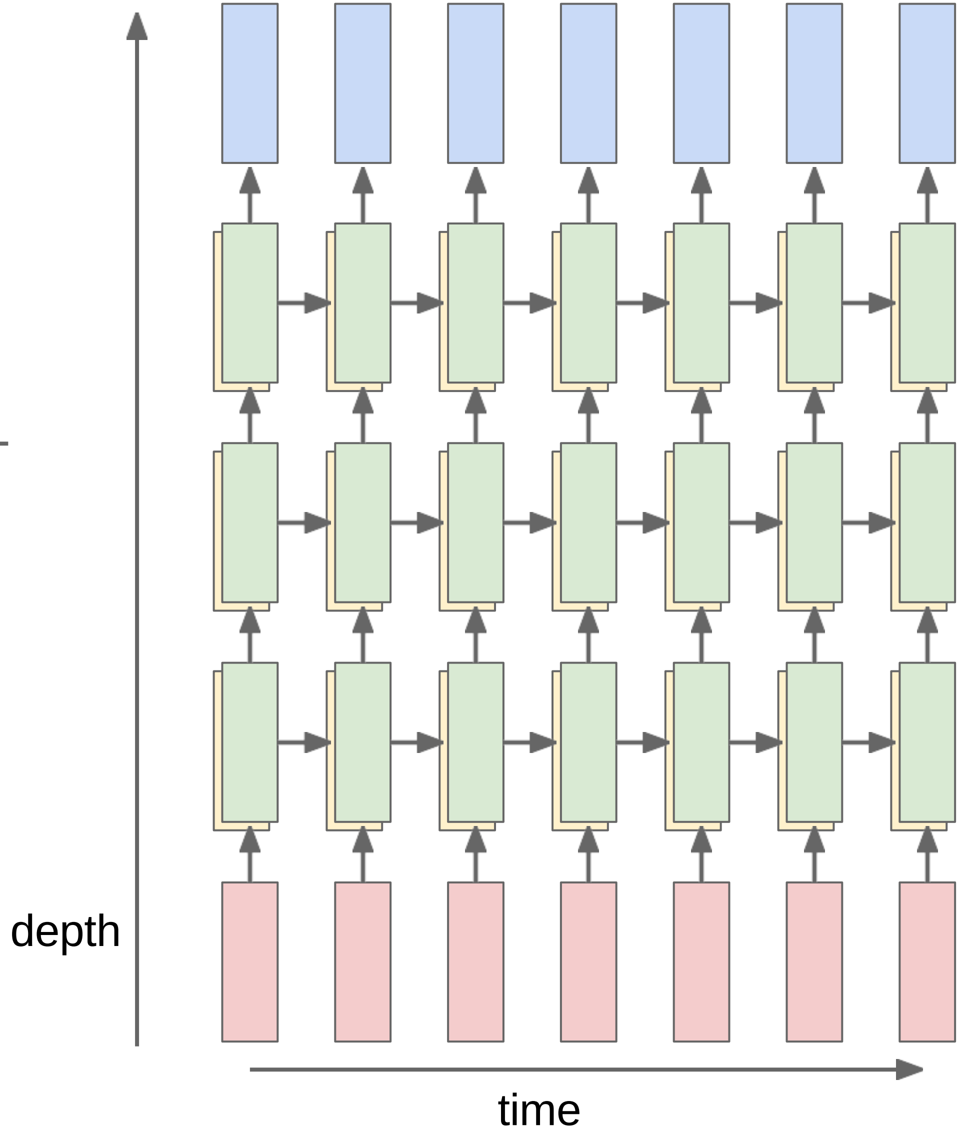
LSTM:

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

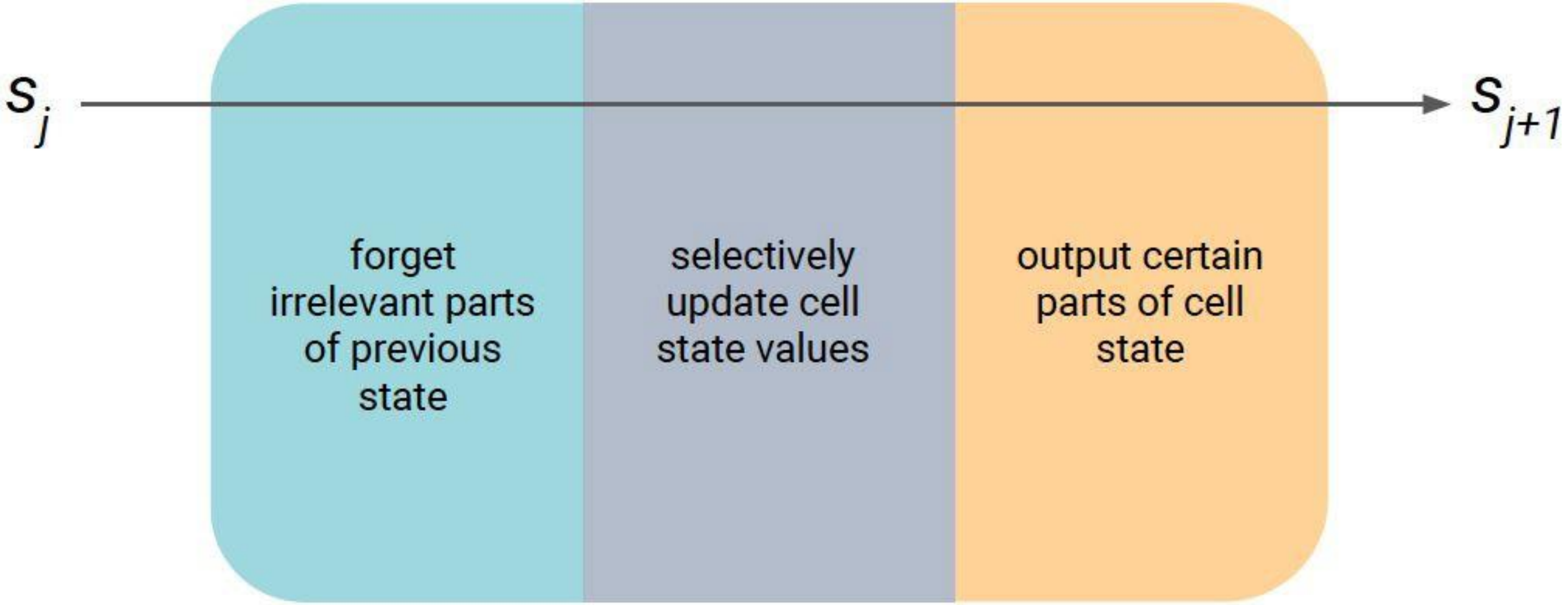
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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



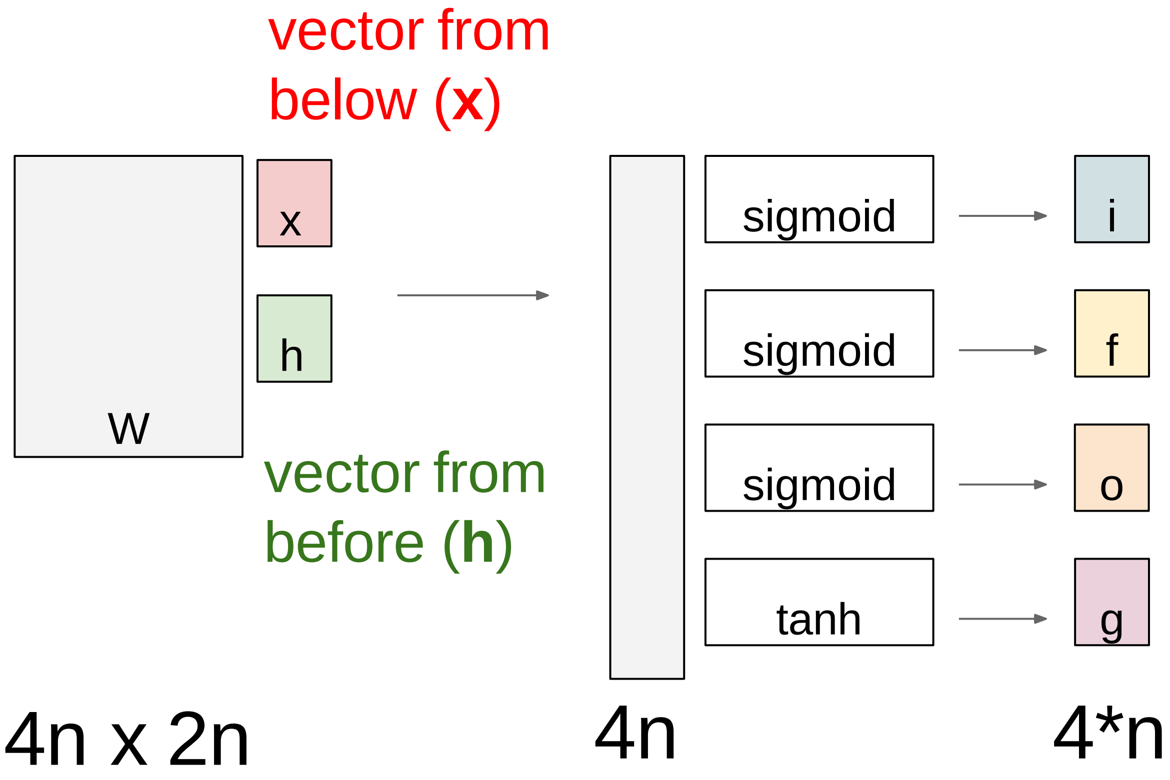
LSTM - main idea



Slide adapted from MIT 6.S191 (IAP 2017), by Harini Suresh

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*

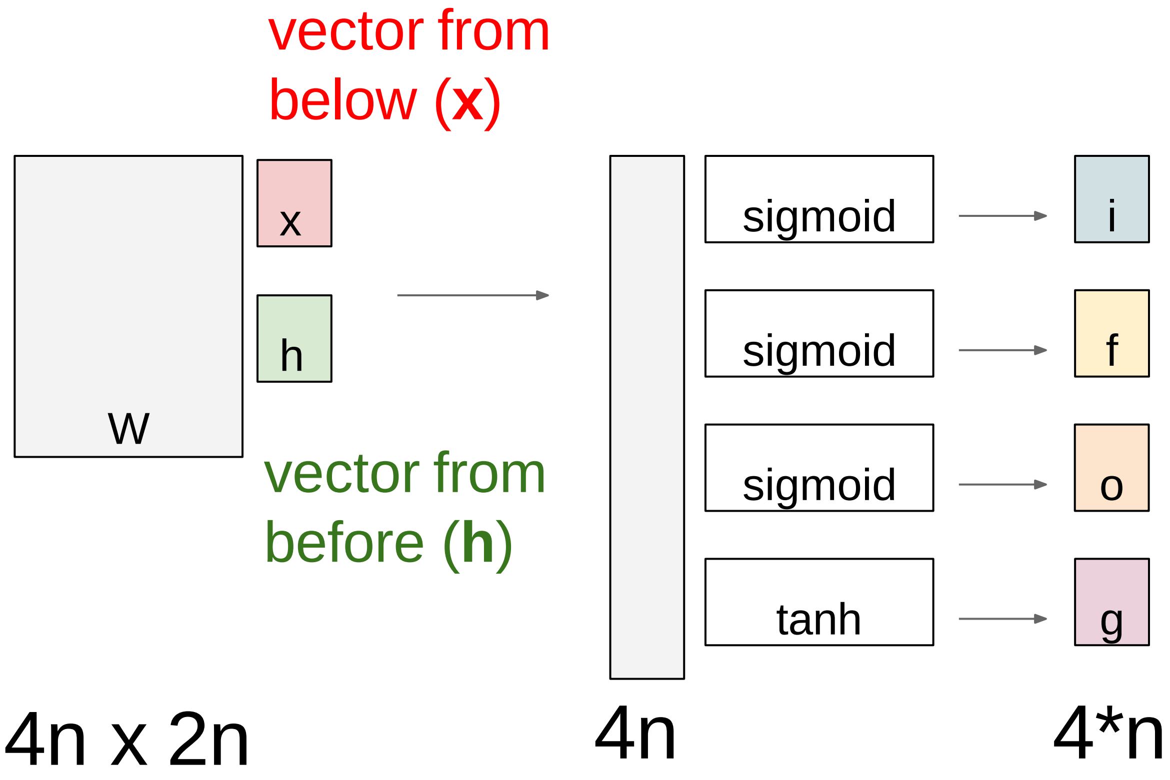
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

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]

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

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

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

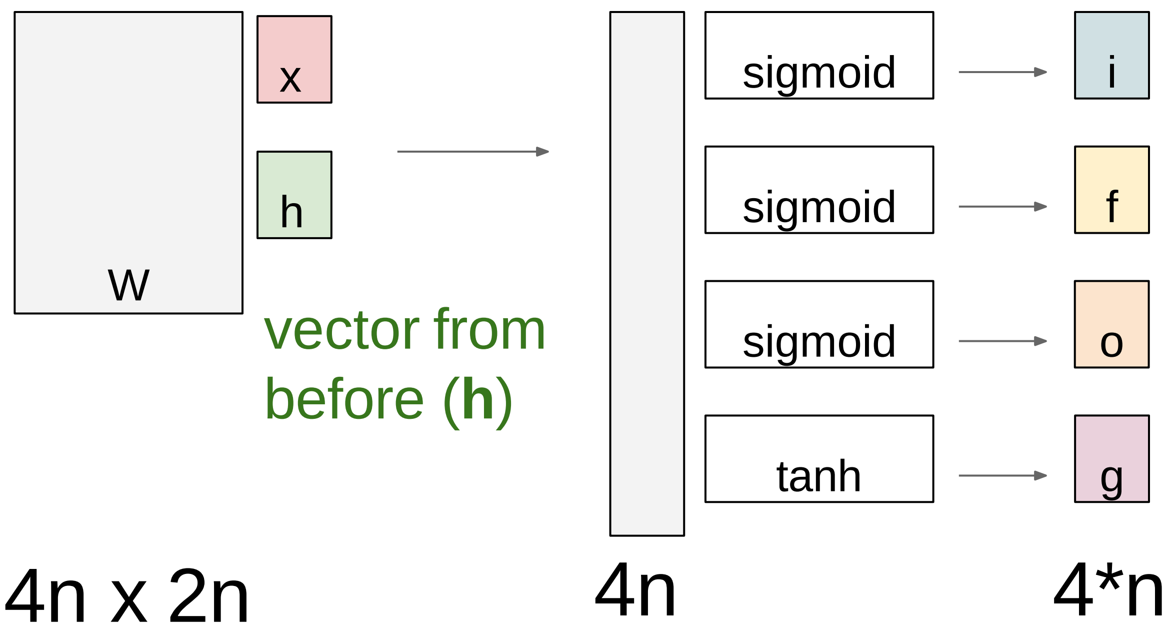
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

g is a transformation of input / hidden state

vector from below (**x**)

vector from before (**h**)

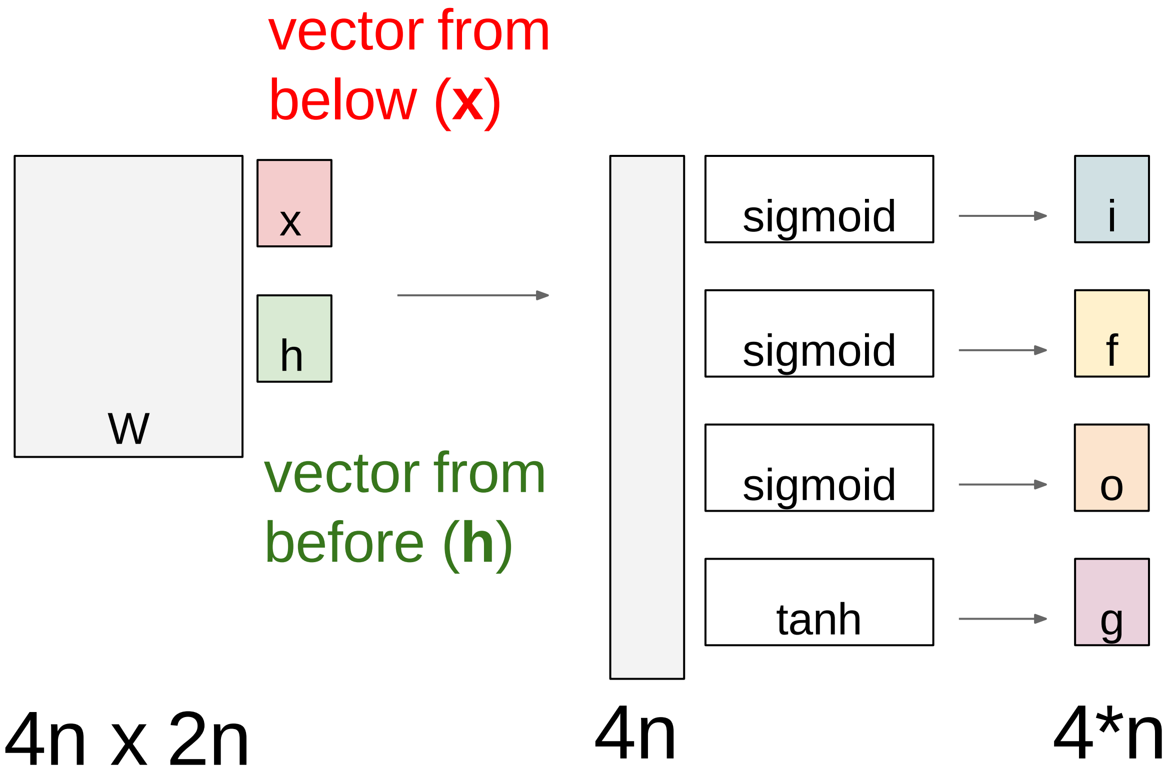


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

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".*

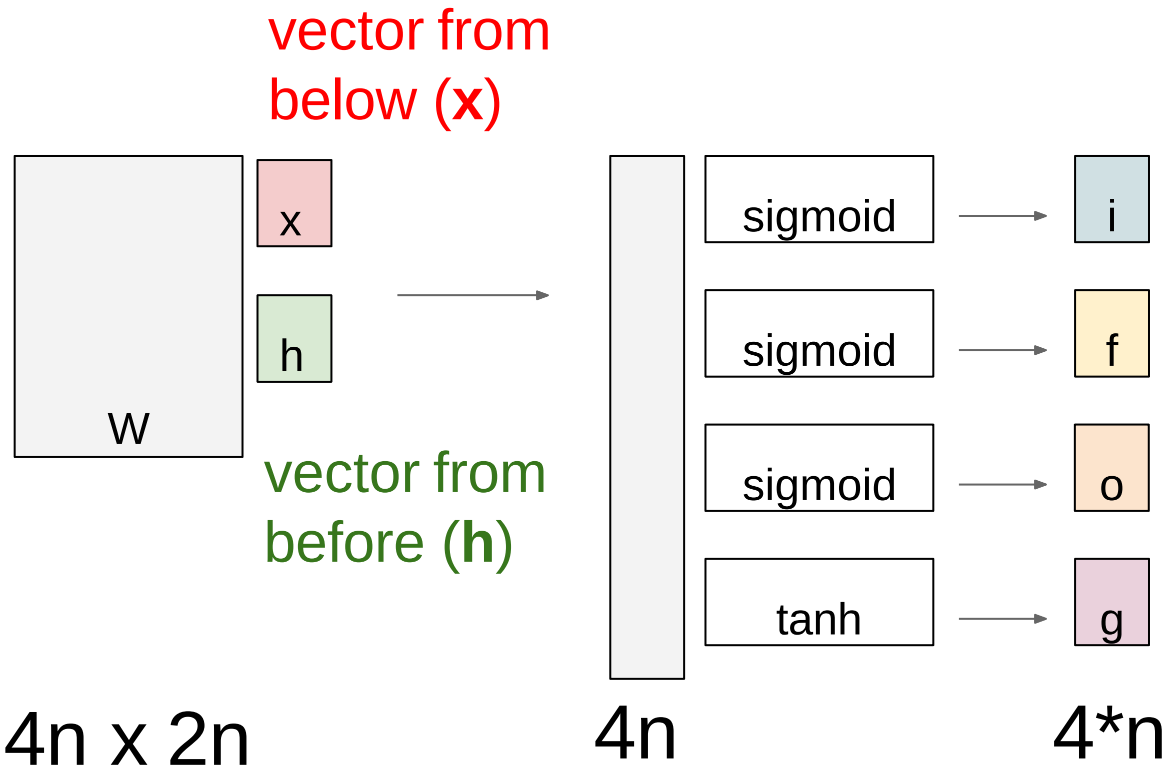
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



New hidden state is a scaled version of tanh(cell state).

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

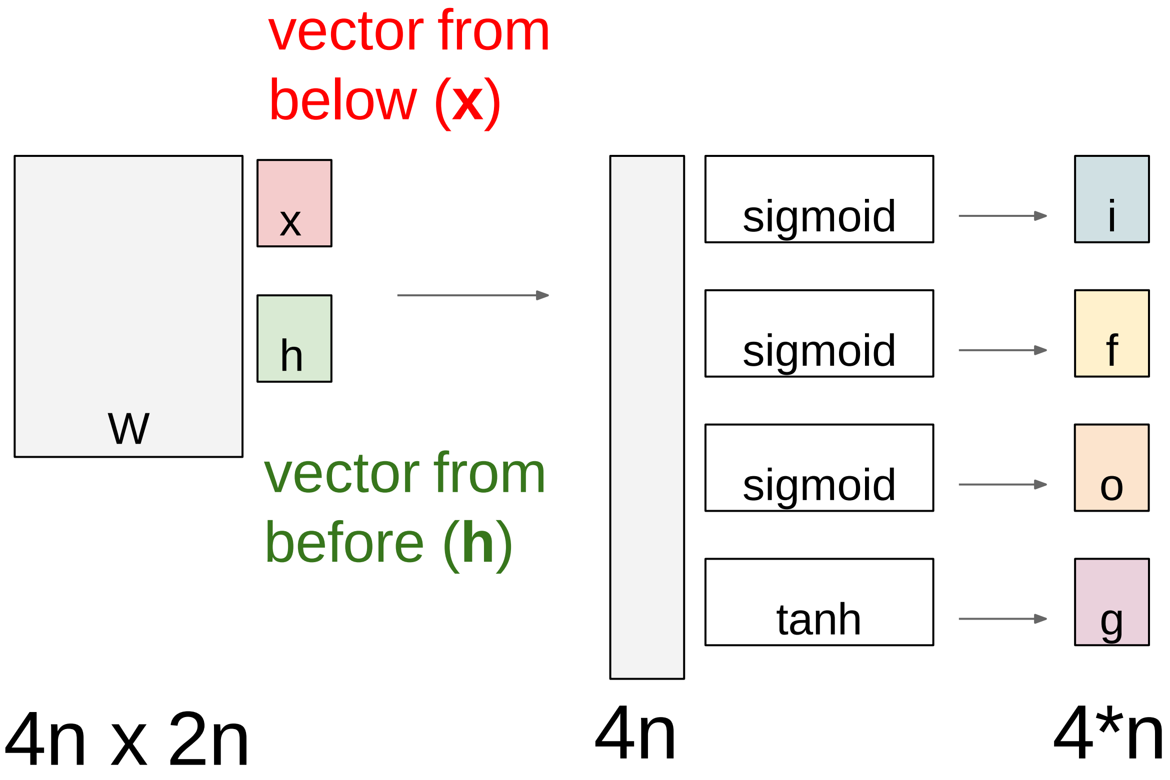
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Q: Why tanh?

A: Not very crucial, sometimes not used

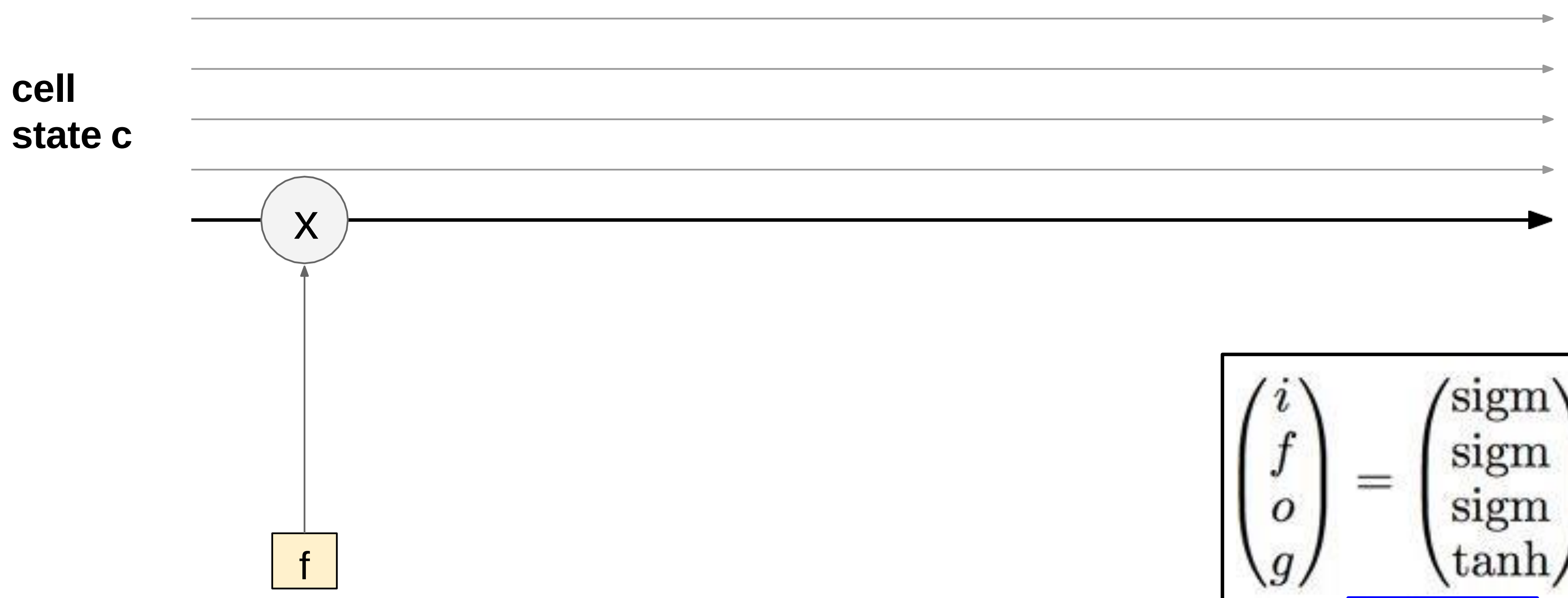
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



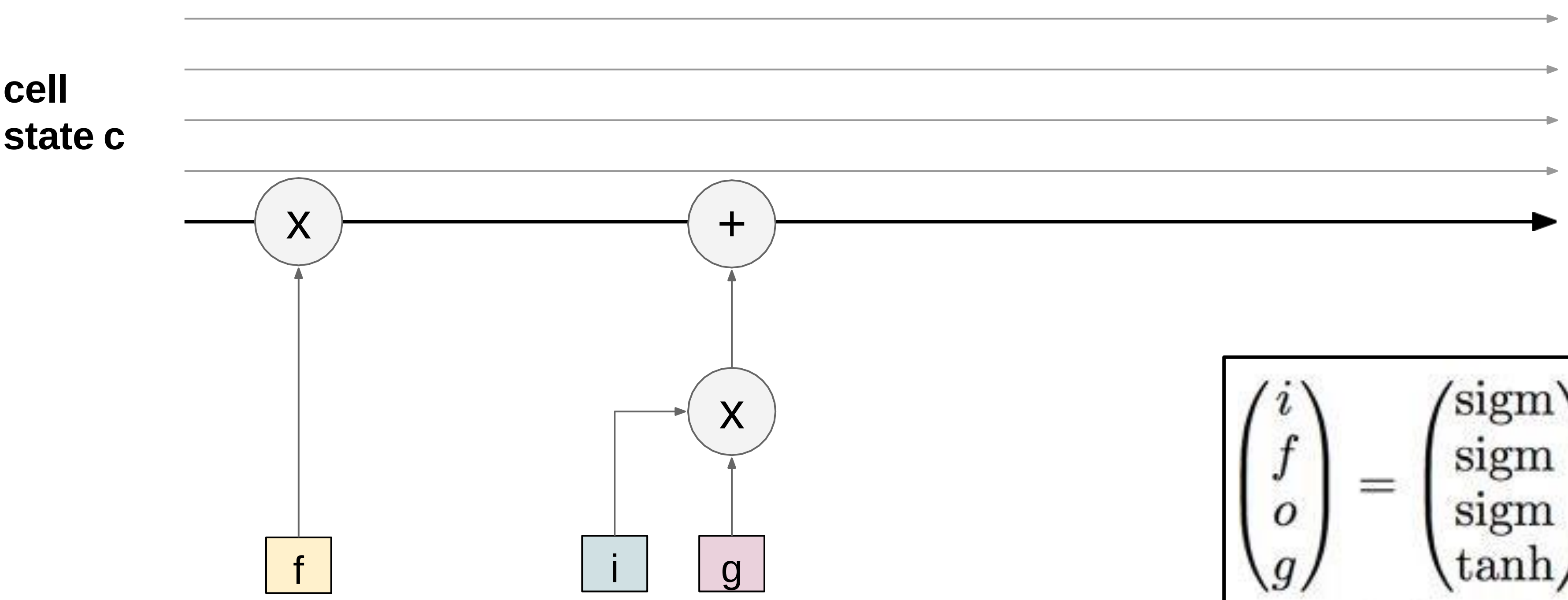
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

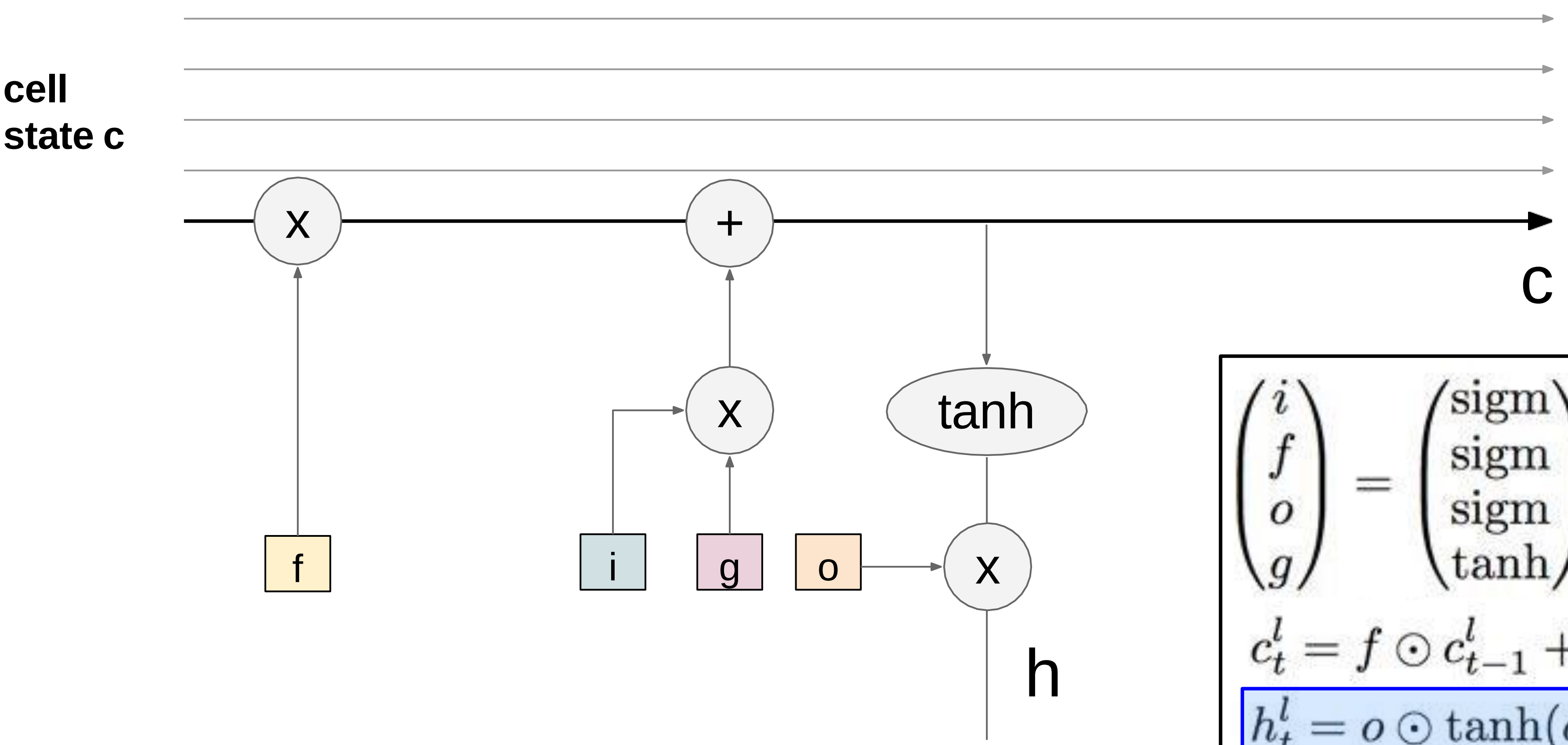


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

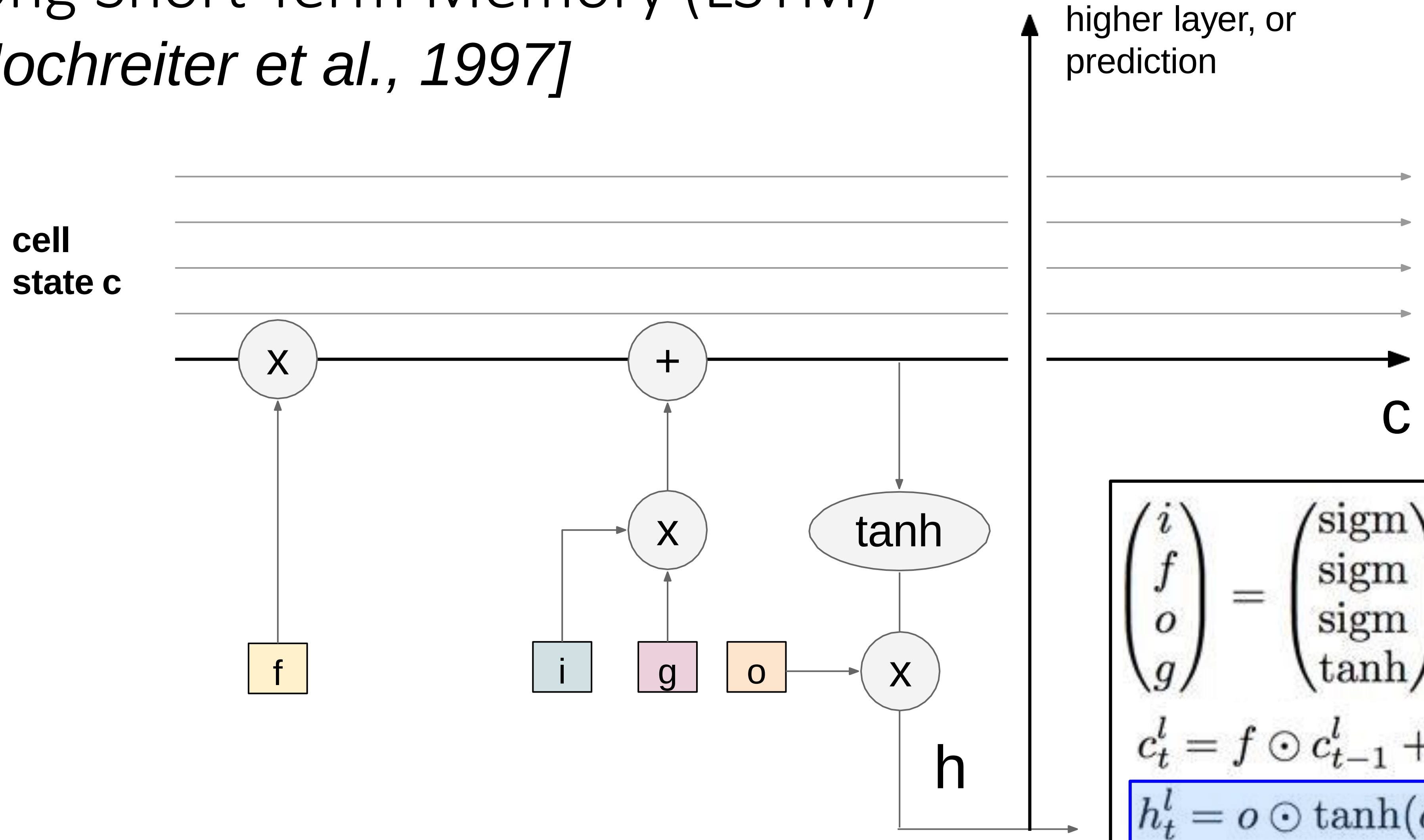


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]



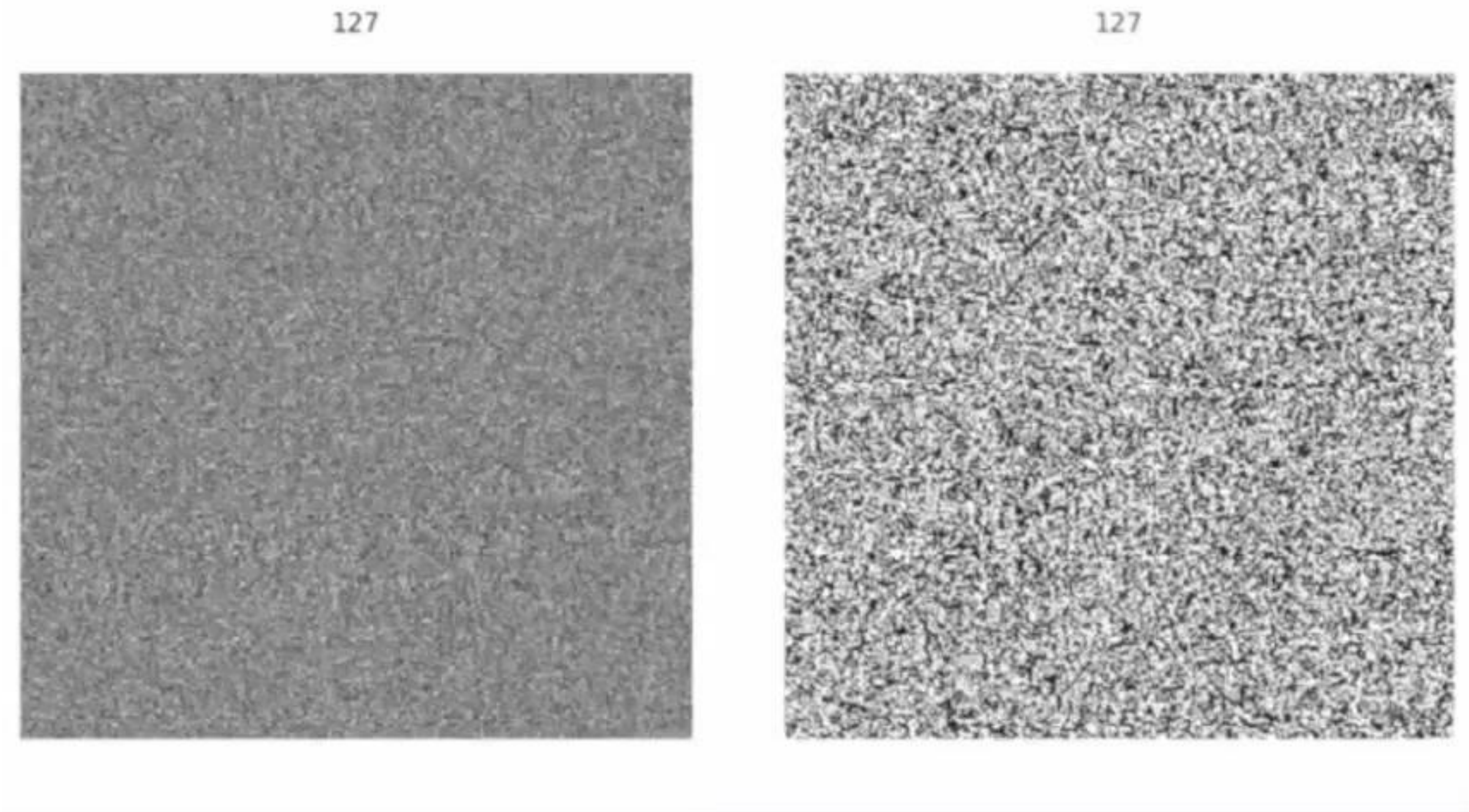
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

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

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

Understanding gradient flow dynamics

Backprop signal



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

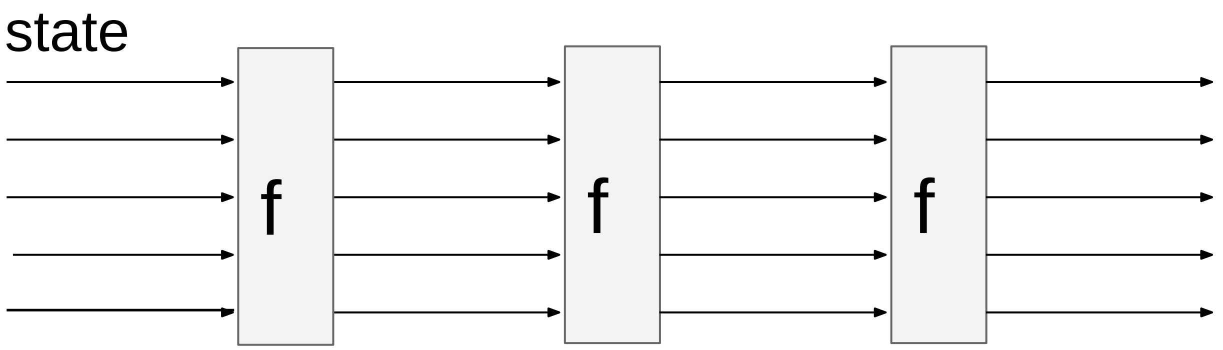
An example how vanishing gradient problem can affect RNNs:

"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

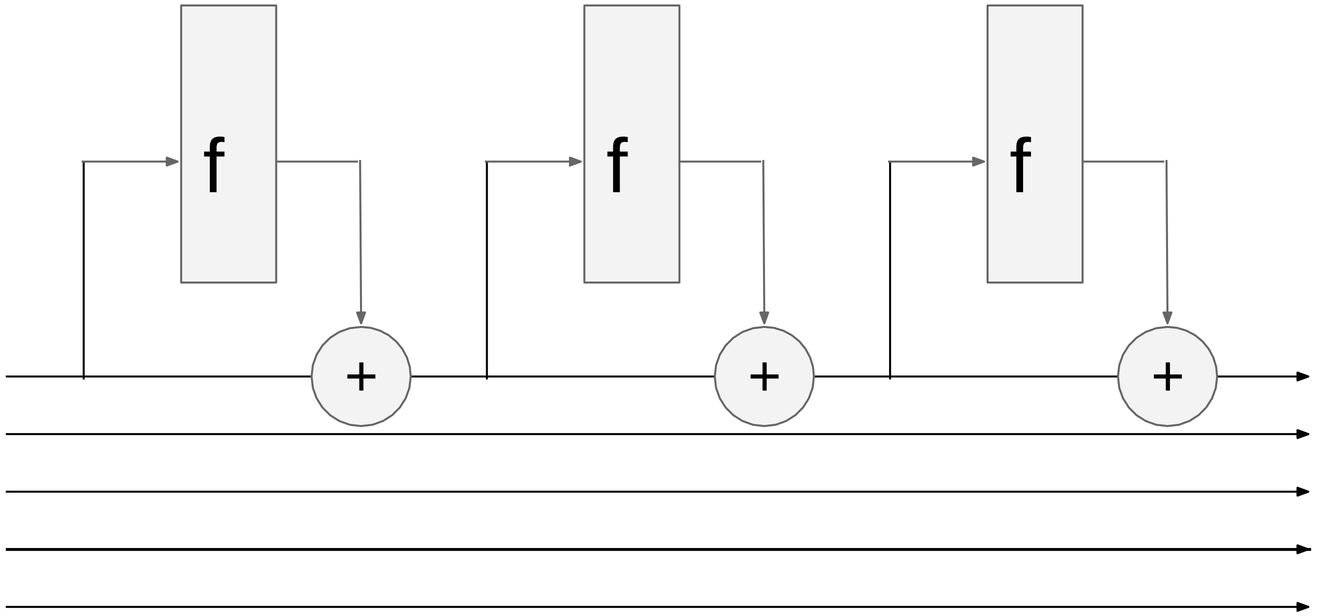
RNN

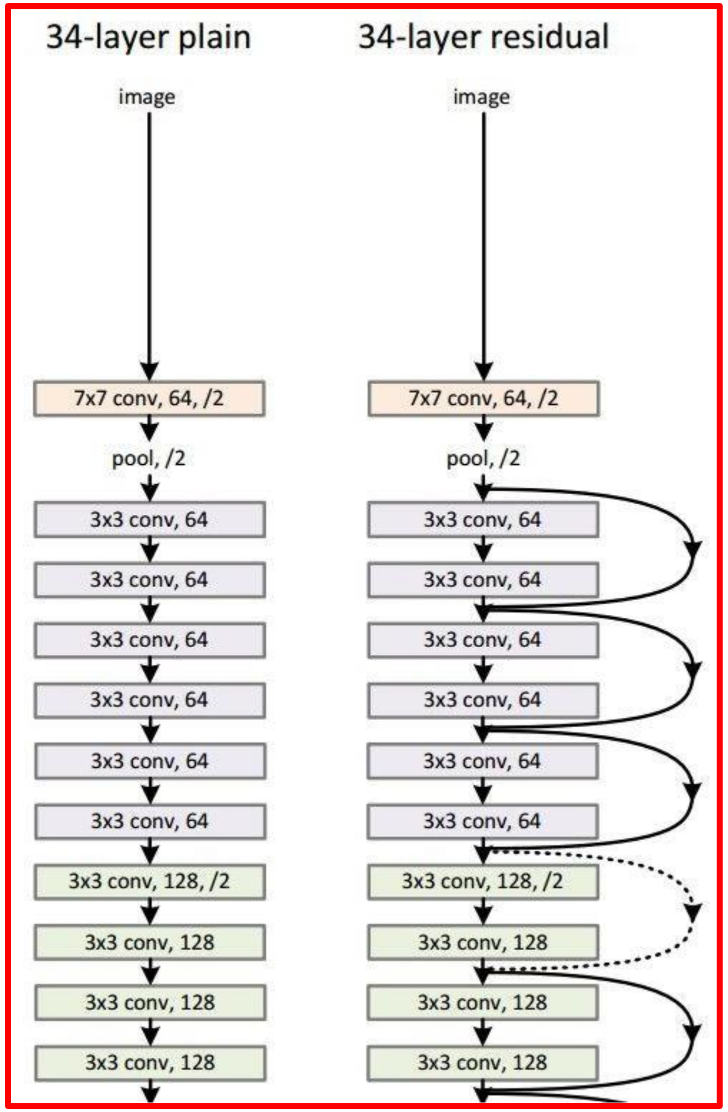
More prone to the vanishing gradient problem



LSTM

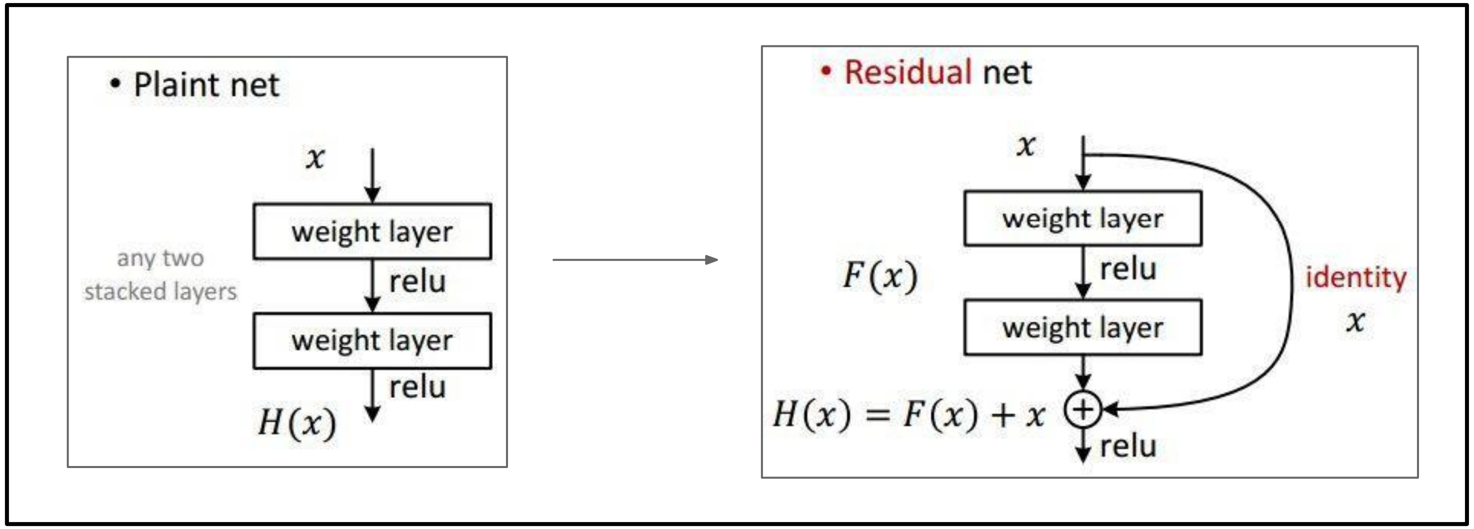
(ignoring forget gates)





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

Slide adapted from MIT 6.S191 (IAP 2017), by Harini Suresh