Computer Vision with Deep Learning

Pascal Mettes

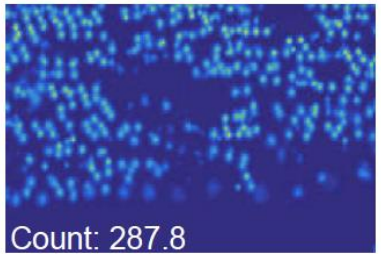
UNIVERSITY OF AMSTERDAM

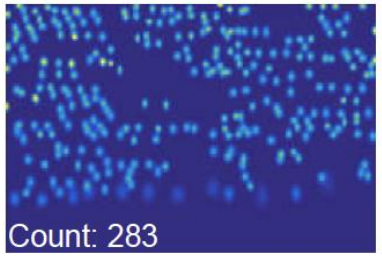
A short introduction

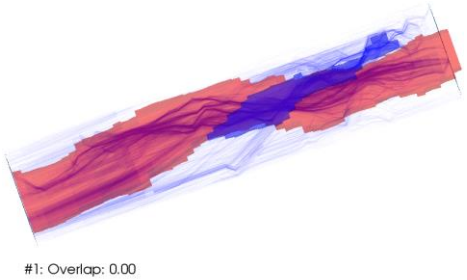
Assistant Professor at University of Amsterdam

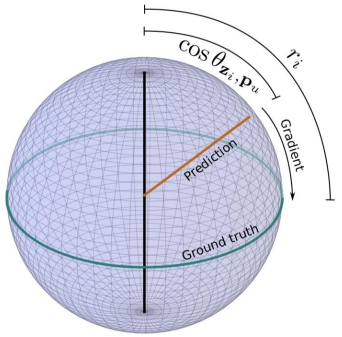
- Formerly at UvA (PhD, postdoc) and Columbia University (visiting).

A focus on action recognition / localization and deep learning for vision.









Why computer vision and deep learning match

Traditional machine learning thrives on structured data, for which features are defined.

They struggle with loosely structured data, with unknown feature representations.

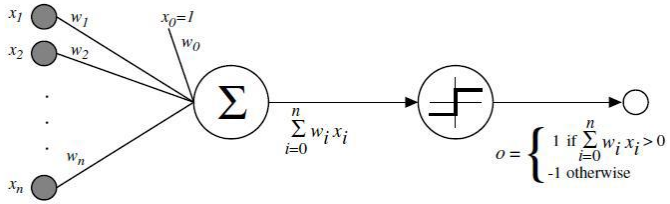
Deep learning enables joint recognition and feature representation.

Attributes of 2009EQ.csv Events

Year	Month	Day	Time_UTC	Latitude	Longitude	Magnitude	Depth	Shape *
2009	1	1	12:43:12 PM	-15.43	-173.14	4	35	Point
2009	1	1	9:36:00 PM	17.22	40.52	5	10	Point
2009	1	1	9:50:24 AM	-55.16	-29.09	4.7	35	Point
2009	1	1	1:26:24 PM	80.85	-3.03	4.8	10	Point
2009	1	1	6:28:48 AM	-6.92	155.18	4.6	82	Point
2009	1	1	10:19:12 AM	-6.83	129.99	4.7	50	Point
2009	1	1	12:00:00 PM	-33.8	-72.72	4.6	0	Point
2009	1	1	12:57:36 PM	-58.29	-21.81	4.4	10	Point
2009	1	1	3:21:36 AM	-6.86	155.93	4.7	50	Point
2009	1	1	7:55:12 PM	1.12	120.73	4.7	49	Point
2009	1	1	11:16:48 AM	-11.66	166.75	4.7	254	Point
2009	1	1	2:09:36 AM	40.62	123.02	4.1	10	Point
2009	1	1	5:16:48 AM	-34.84	-107.65	5.8	10	Point
2009	1	1	1:40:48 PM	-9.61	120.72	4.2	20	Point
2009	1	1	2:38:24 AM	-22.04	-179.6	4.5	601	Point
2009	1	1	6:43:12 AM	1.32	121.84	5.1	33	Point
2009	1	1	4:19:12 PM	14.73	-91.39	4.7	169	Point
2009	1	1	5:45:36 PM	9.43	124.15	4.5	525	Point
2009	1	1	4:48:00 PM	-34.88	-107.78	5	10	Point
2009	1	1	8:09:36 PM	44.58	148.22	4.2	59	Point
2009	1	1	2:38:24 AM	-4.33	101.3	5.5	19	Point
2009	1	1	2:52:48 AM	-4.33	101.24	5.3	26	Point

Record: 1 Show: All Selected Records (0 out of 2048 Selected)

Topics of today



The Core

Single- and multi-layer perceptrons, forward/backward propagation, optimization, convolutional networks, state-of-the-art, video recognition

Beyond The Core

Object detection, per-pixel prediction, fully convolutional networks, geometric deep learning, hyperspheres

Deep learning in one slide

A family of parametric, non-linear and hierarchical representation learning functions, which are massively optimized with stochastic gradient descent to encode domain knowledge, i.e. domain invariances, stationarity.

$$a_L\left(x; \theta_{1,\dots,\mathrm{L}}\right) = h_L\left(h_{L-1}(\dots h_1(x, \theta_1), \theta_{L-1}), \theta_L\right)$$

x:input, θ_l: parameters for layer l, $a_l = h_l(x, \theta_l)$: (non-)linear function

Given training corpus $\{X, Y\}$ find optimal parameters

$$\theta^* \leftarrow \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} \ell\left(y, a_L\left(x; \theta_{1,\dots,\mathrm{L}}\right)\right)$$

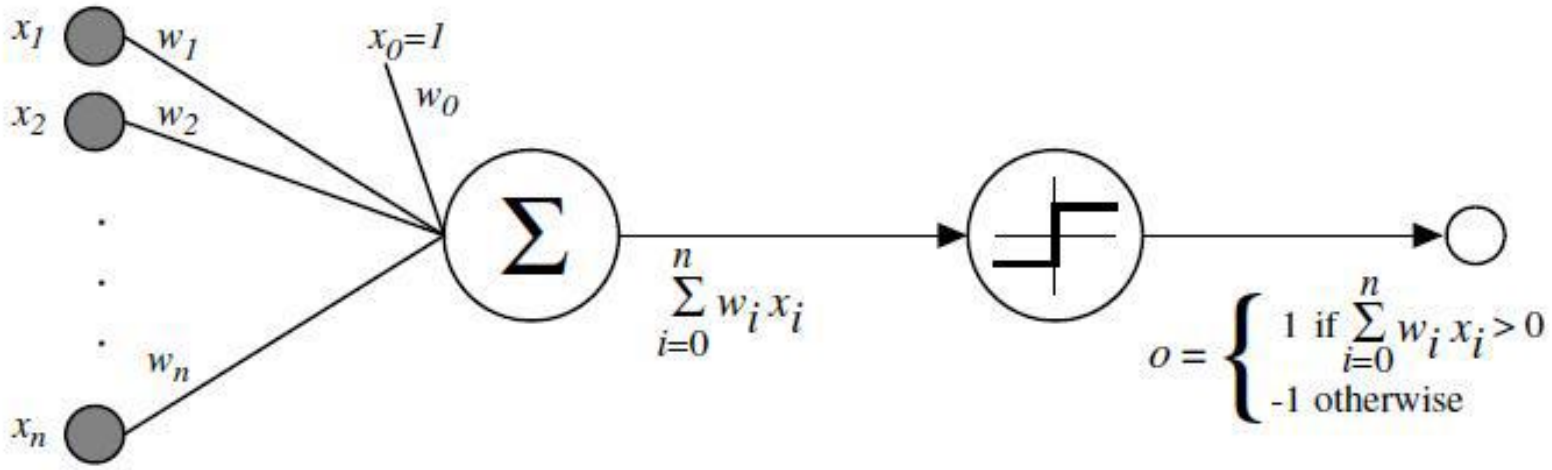
Deep learning essentials

Historical perspective on deep learning

The perceptron

Single layer perceptron for binary classification.

- One weight w_i per input x_i
- Multiple each input with its weight, sum, and add bias
- If result larger than threshold, return 1, otherwise return 0

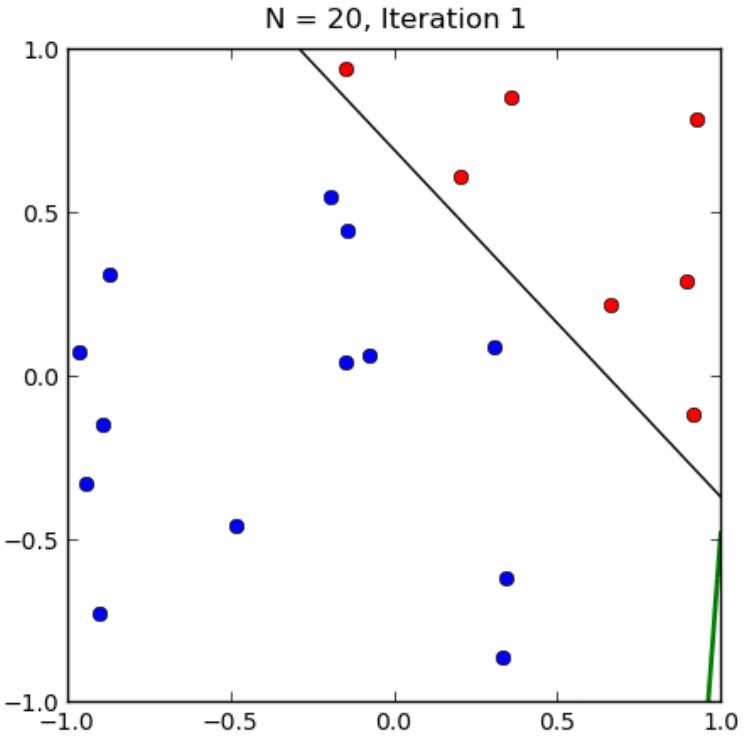


Training a perceptron

Rosenblatt's innovation was the learning algorithm for perceptrons.

Learning algorithm:

- Initialize weights randomly
- Take one sample x_i and predict y_i
- For erroneous predictions update weights
  - If prediction $\hat{y}_i = 0$ and ground truth $y_i = 1$, increase weights
  - If prediction $\hat{y}_i = 1$ and ground truth $y_i = 0$, decrease weights
- Repeat until no errors are made



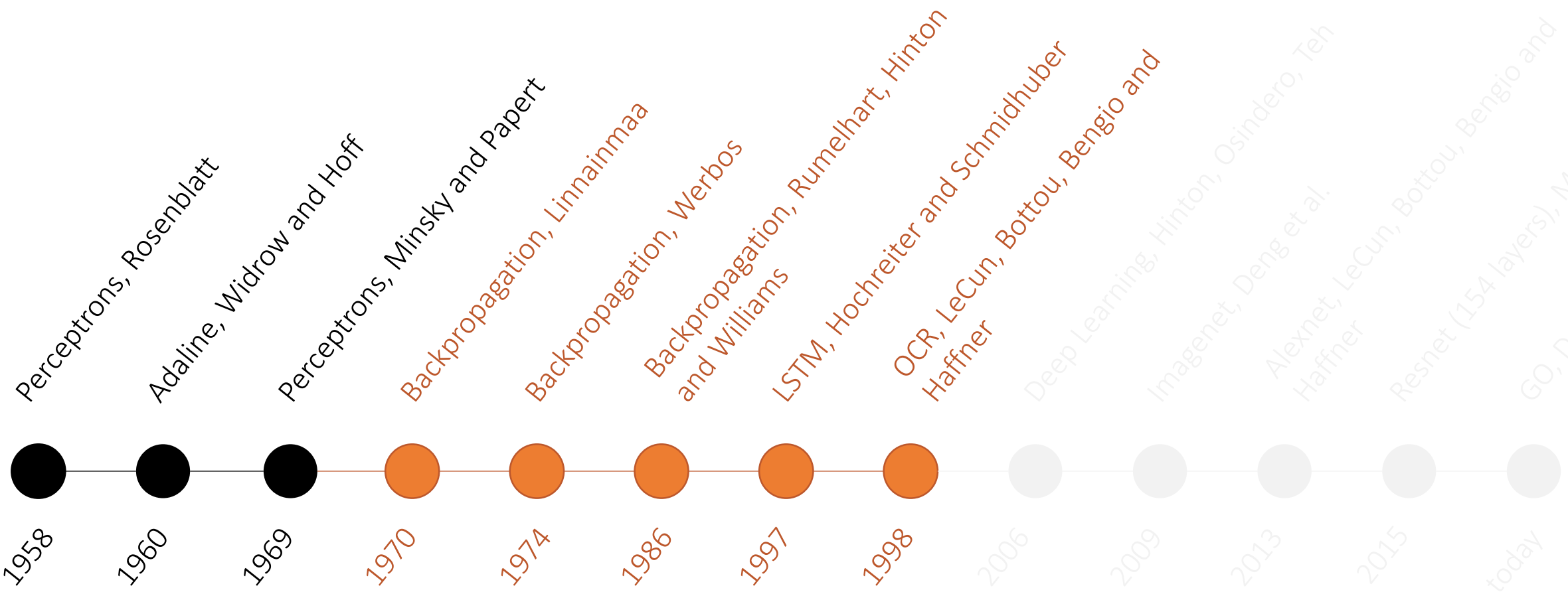
Problems with the perceptron

Rosenblatt (1958) at US Navy press conference:

> *"[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."*

Perceptrons turned out to only solve linearly separable problems.

Historical perspective on deep learning



An AI winter

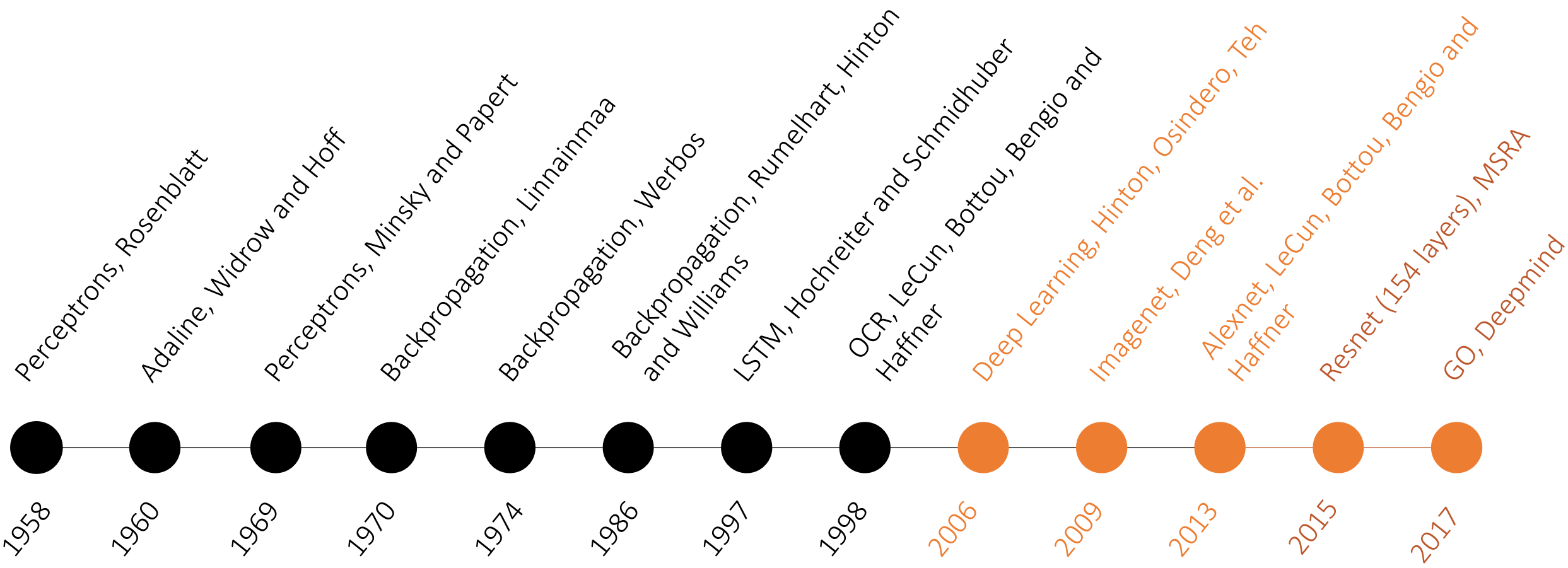
Results not as promised due to overhyping.

- Government and industry cut research funding.

Interestingly, still significant discoveries made in this period.

- Backpropagation
- Convolutional networks
- Recurrent networks

Historical perspective on deep learning



New AI hype

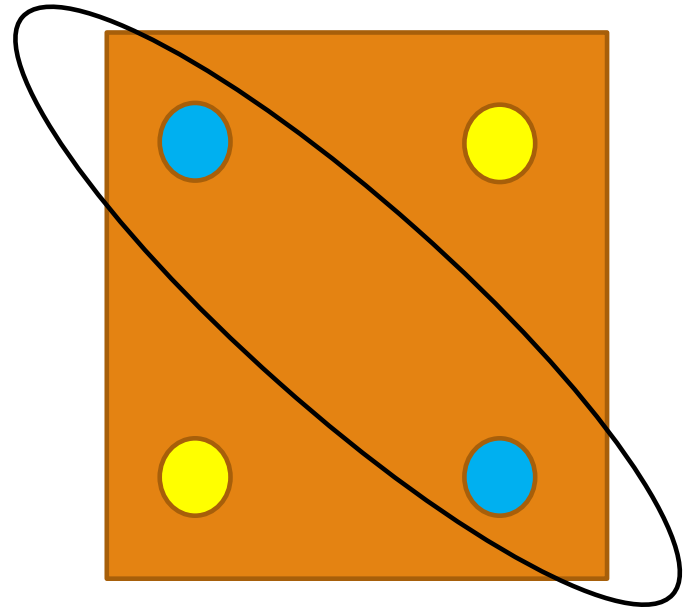
The new deep learning summer is a result of these elements:

1. Data
2. Hardware
3. Open-source software
4. Tricks
5. Deep learning beyond recognition

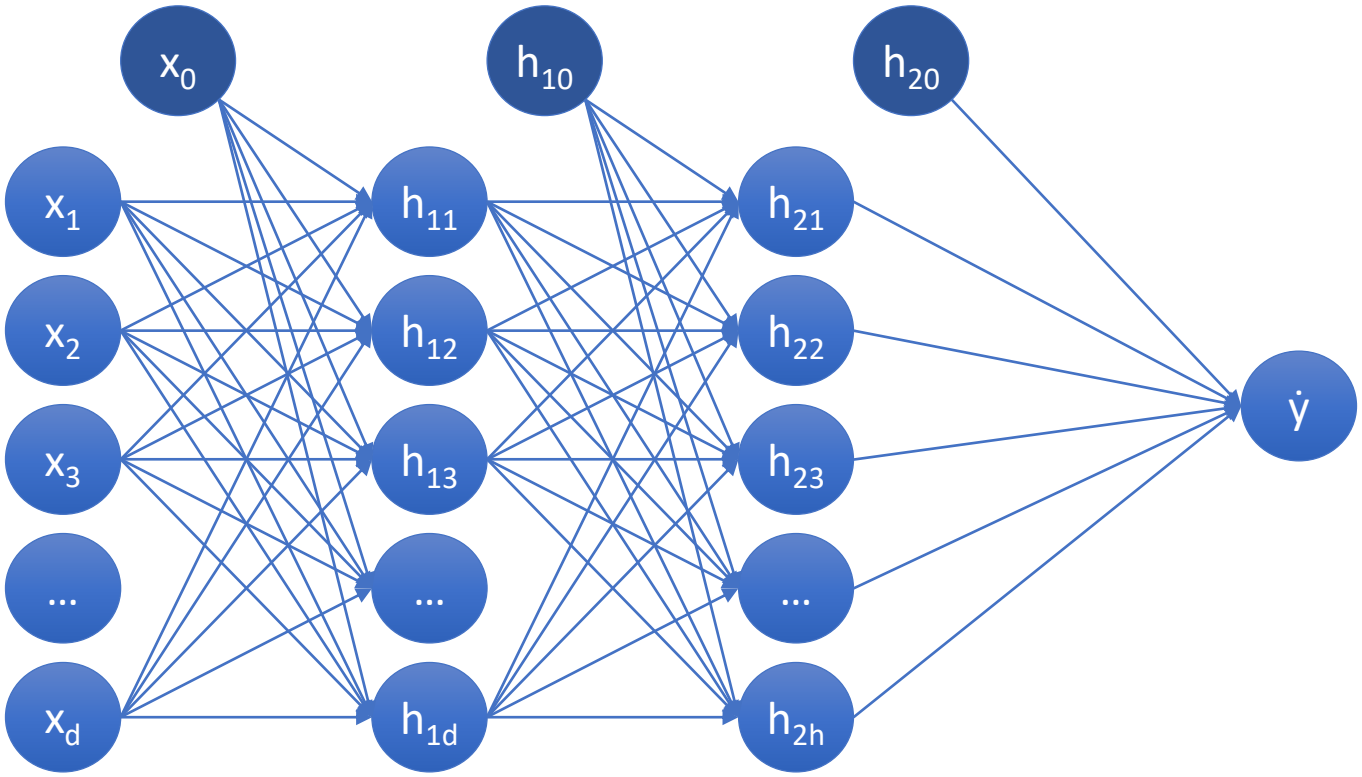
Tackling the limitations of the perceptron

Famous counterexample: the XOR function.

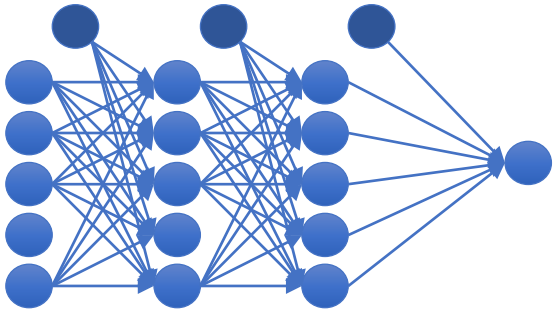
How can we separate data points beyond lines?



Multi-layer perceptrons



Training goal and overview



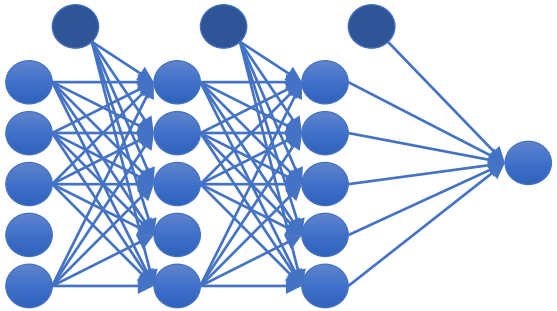
We have a dataset of inputs and outputs.

Initialize all weights and biases with random values.

Learn weights and biases through “forward-backward” propagation.

- Forward step: Map input to predicted output.
- Loss step: Compare predicted output to ground truth output.
- Backward step: Correct predictions by propagating gradients.

Forward propagation



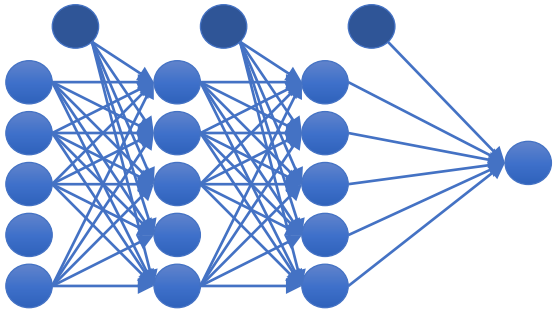
In the basics, the same as the perceptron:

- Start from the input, multiply with weights, sum, add bias.
- Repeat for all following layers until you reach the end.

There is one main new element (next to the multiple layers):

- Activation functions after each layer.

Why have activation functions?



Each hidden/output neuron is a linear sum.

A combination of linear functions is a linear function!

$v(x) = a\,x + b$

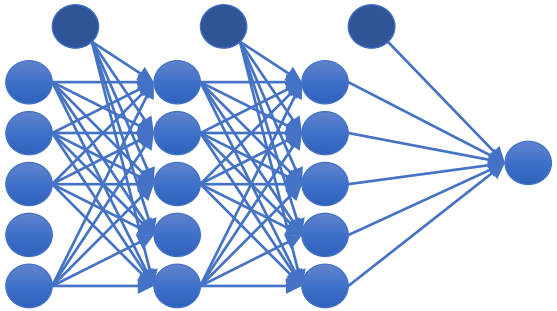
$w(z) = c\,z + d$

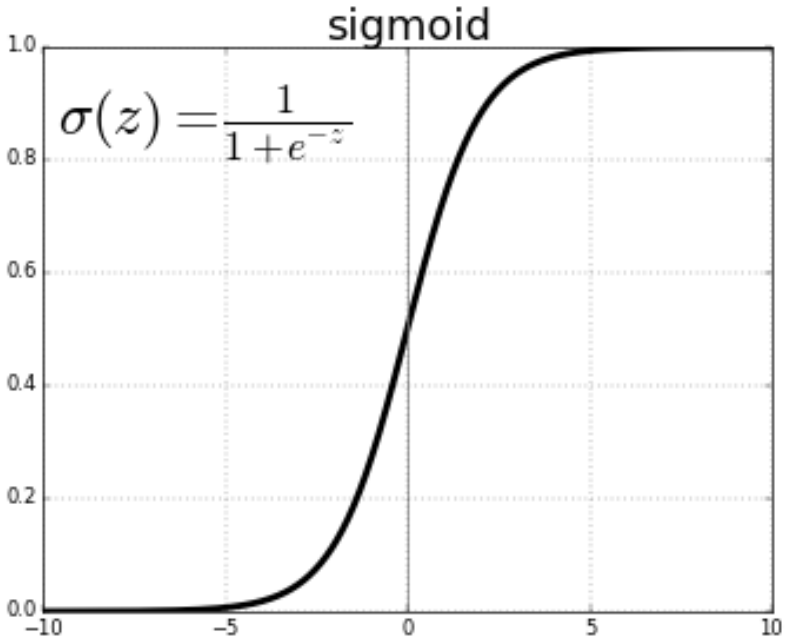
$w(v(x)) = c\,(a\,x + b) + d = (ac)\,x + (cb + d)$

Activation functions transforms the outputs of each neuron.

This results in non-linear functions.

Non-linear activations





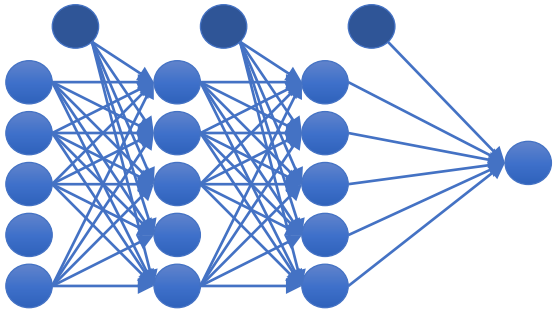
Sigmoid function

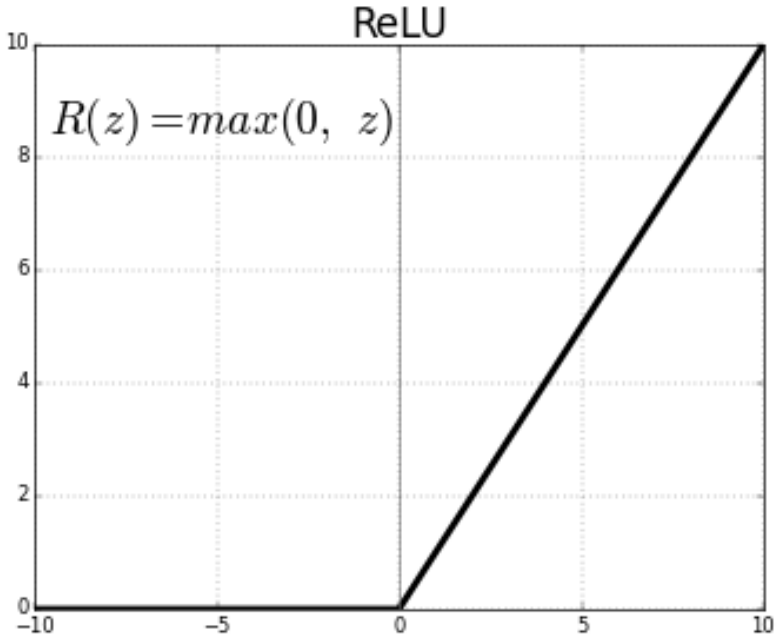
Range: (0,1)

Differentiable

d/dz σ(z) = σ(z) (1 - σ(z))

Non-linear activations





ReLU function

Range: [0,inf)

Differentiable?

d/dz R(z) = 1 if z at least 0, 0 otherwise

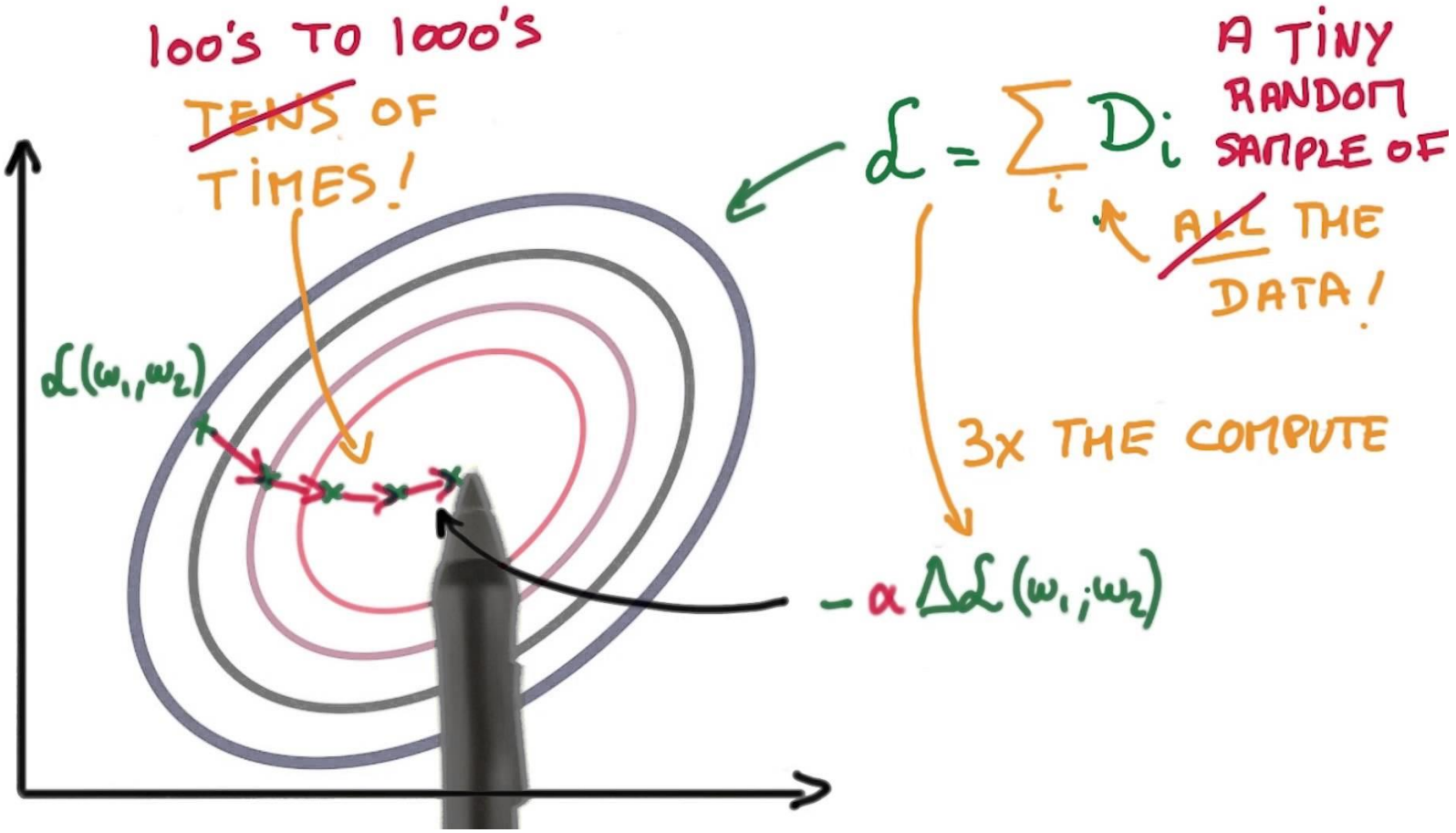
Forward pass conclusion

Going from input to output is a standard procedure.

Go from one layer to the next until you are at the output.

At each layer, weighted sum with activation function.

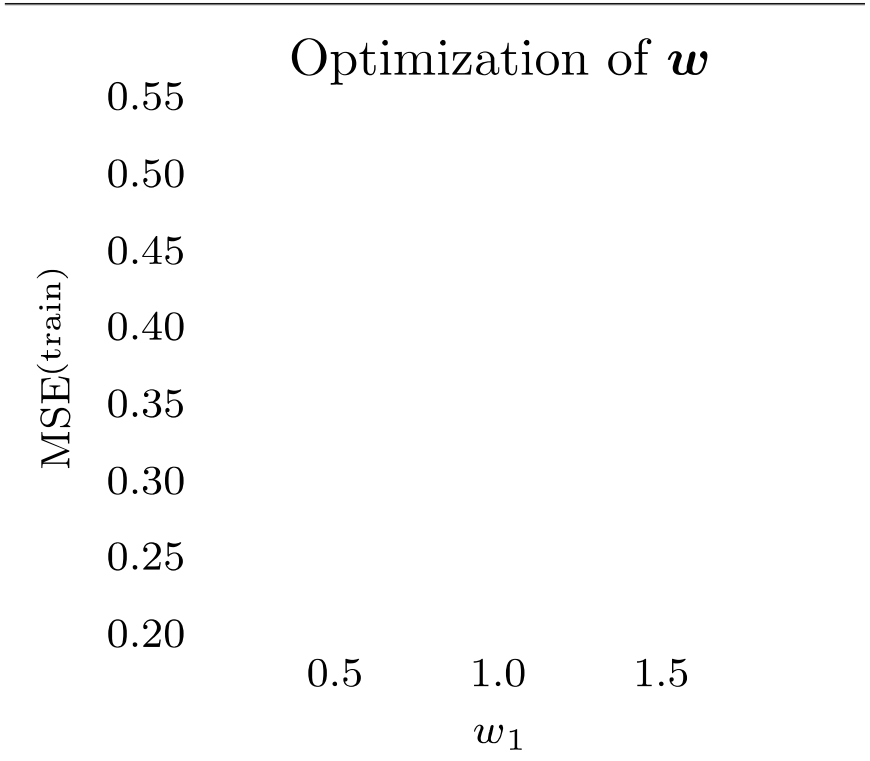
Going back: gradient descent



Back to basics: finding minima

We can find the best value of *w* by examining the gradient of *J(w)*.

Minimal loss when gradient is 0!



Back to basics: finding *w*

To find the best w, we first need the gradient of *J(w).*

$$J(w) = 1/2 \sum_n (t_n - x_n^T w)^2$$

The derivative of *J(W)* w.r.t. *w* by chain rule:

$$\nabla w\, J(w) = - \sum_n (t_n - x_n^T w)\, x_n$$

Gradient descent

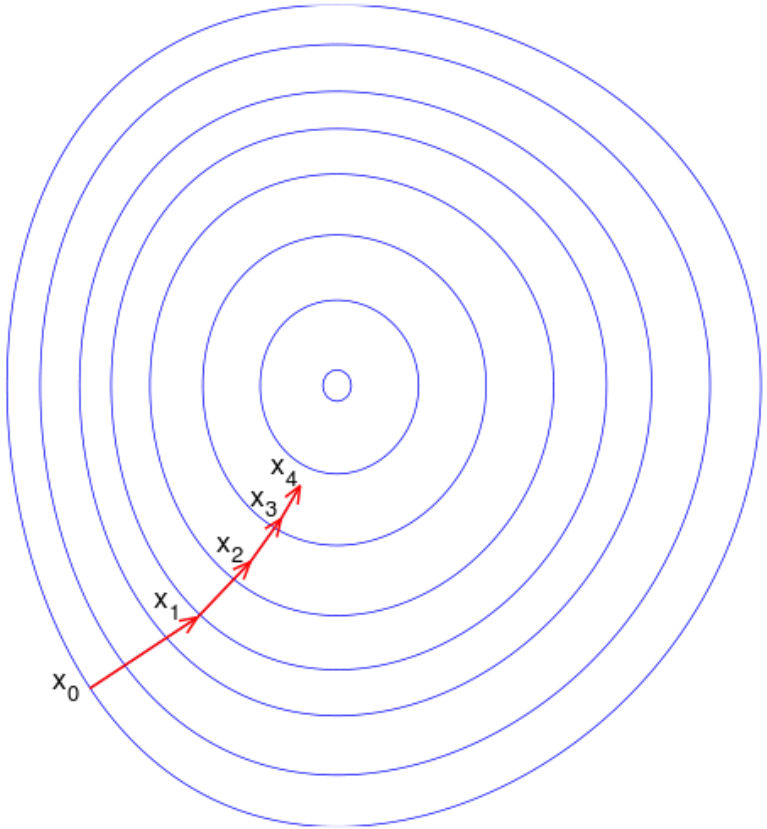
The go-to method for optimization in deep networks, we will find out soon why this is so.

Start with w_0

For t=1,..,T

$$w_{t+1} = w_t - \boldsymbol{\gamma}\, d/dw_t\, f(w_t)$$

with $\boldsymbol{\gamma}$ a small value

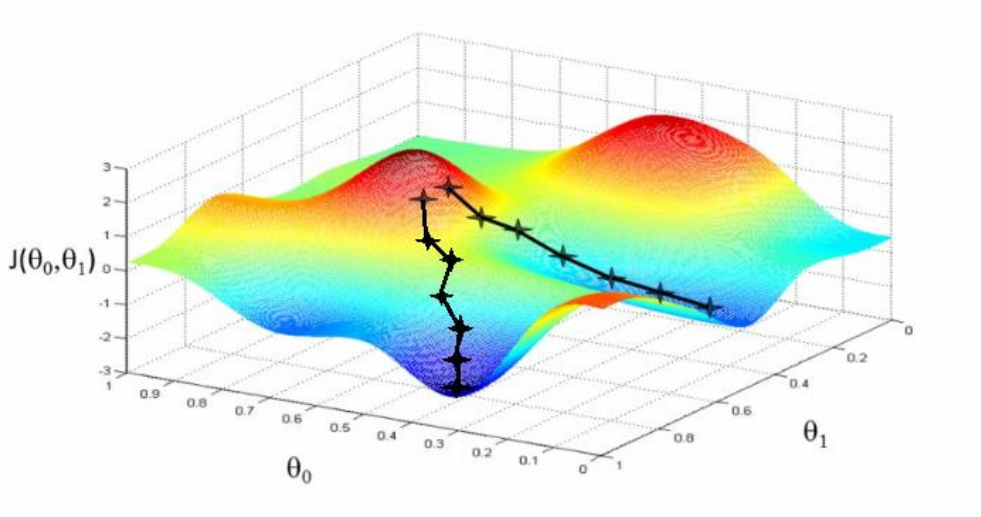


Pros and cos of gradient descent

Pros:

Simple and effective for many ML tasks

Highly scalable



Cons:

Applies to smooth functions

Finds a local minimum, not directly global minimum

Stochastic gradient descent

Start with w_0

For t=1,..,T

Select a few examples $\{x\}_t$

$w_{t+1} = w_t - \boldsymbol{\gamma}\, \partial_{loss}\, f(w_{t,}, \{x\}_t)$

where $\boldsymbol{\gamma}$ is small

Stochastic gradient descent

Advantages

- Fast to compute, memory efficient
- Provable convergence for small $\boldsymbol{\gamma}$
- Implicit regularization

Disadvantages

- Convergence requires tuning of learning rate ($\boldsymbol{\gamma}$)
- Less advanced optimization techniques
    - However: recently new optimizers (eg ADAM, momentum, …)

The backward propagation algorithm

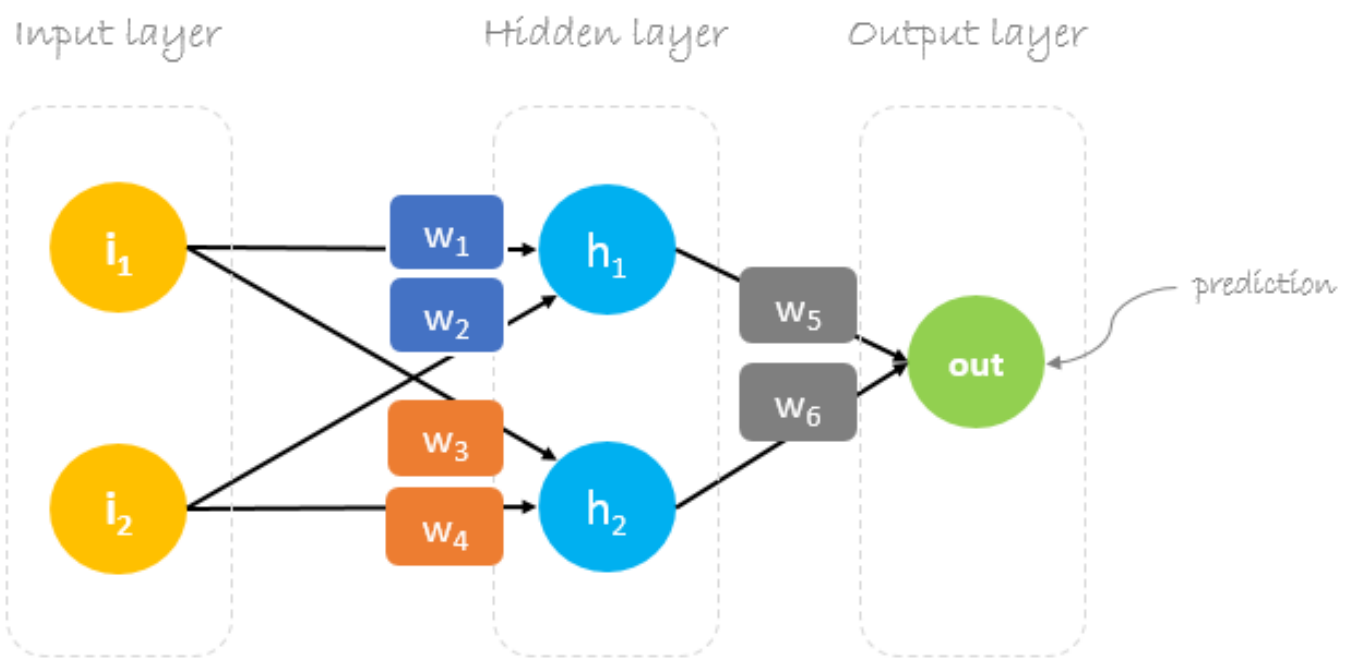
Backpropagation = gradient propagation over the whole network.

Propagate gradient obtained at output back to all neurons.

Propagation over layers is done with the chain rule.

Credit to hmkcode.github.io for example

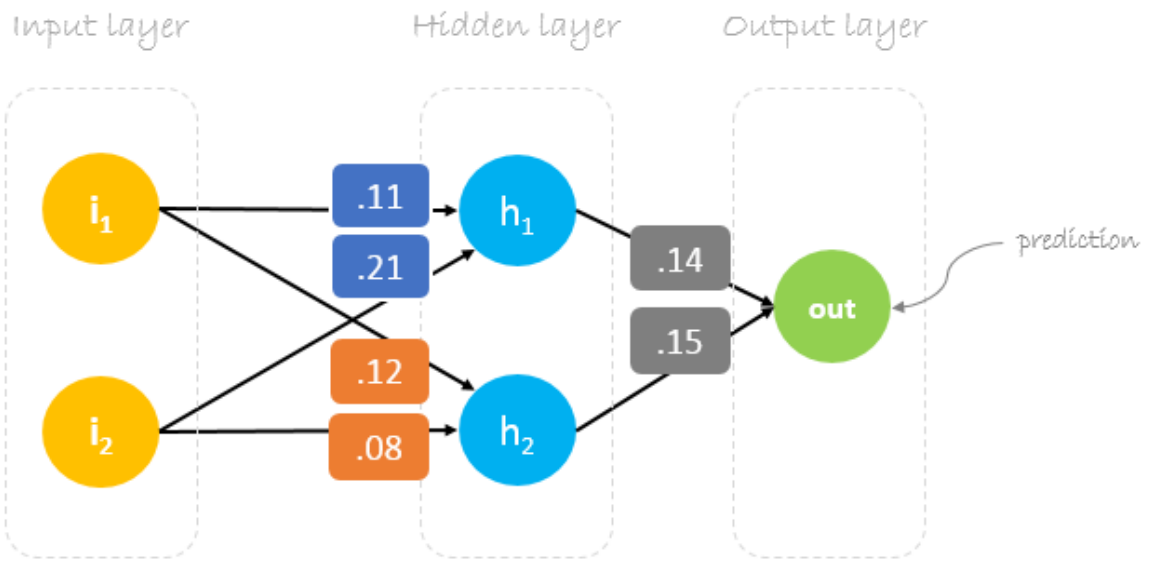
Forward-backward by example



Credit to hmkcode.github.io for example

Forward-backward by example

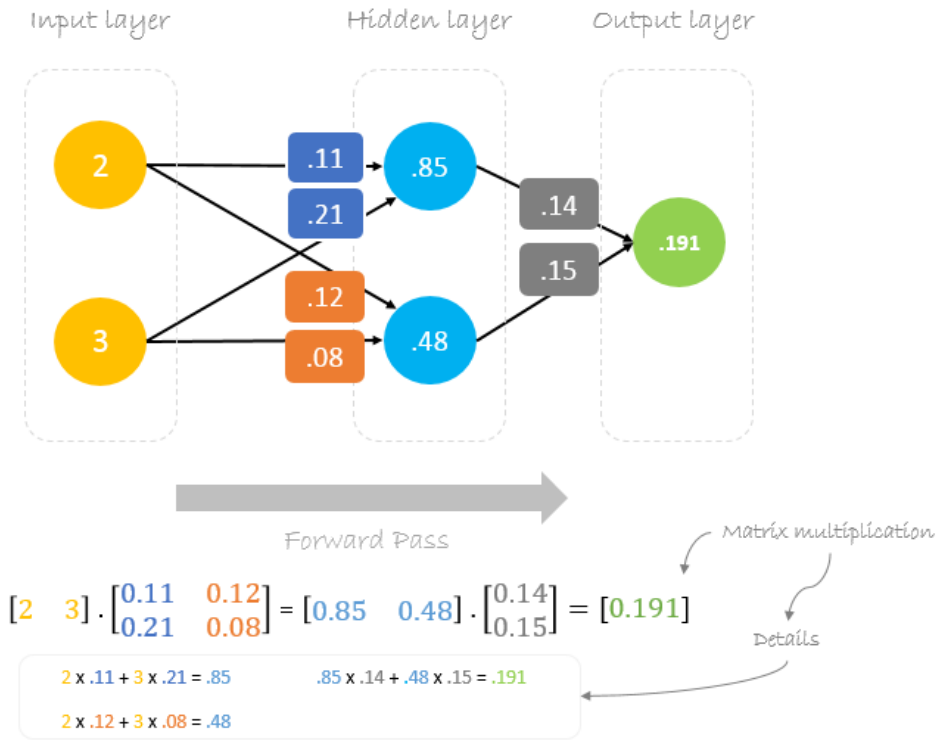
Step 1: Initialize parameters with random values.



Credit to hmkcode.github.io for example

Forward-backward by example

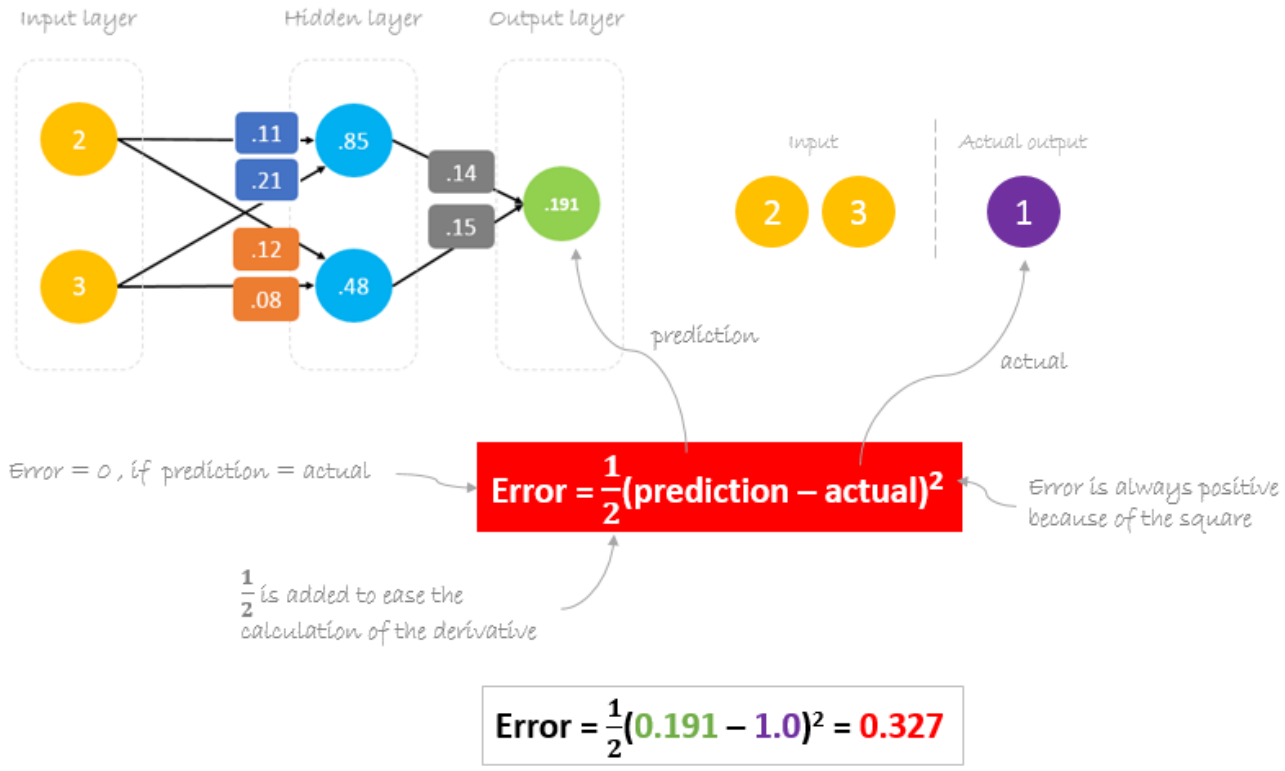
Step 2: Forward propagation given training example.



Credit to hmkcode.github.io for example

Forward-backward by example

Step 3: Calculate error at the output.



Credit to hmkcode.github.io for example

Forward-backward by example

Step 4: Backpropagate error.

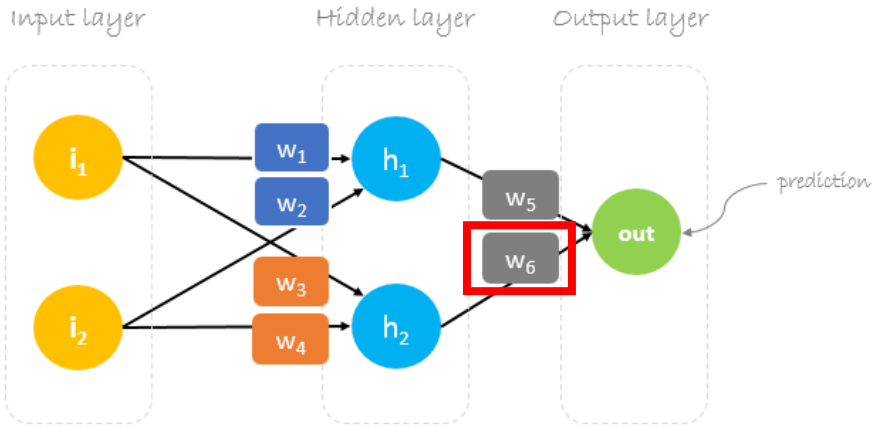
$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

Old weight

Derivative of Error with respect to weight

New weight

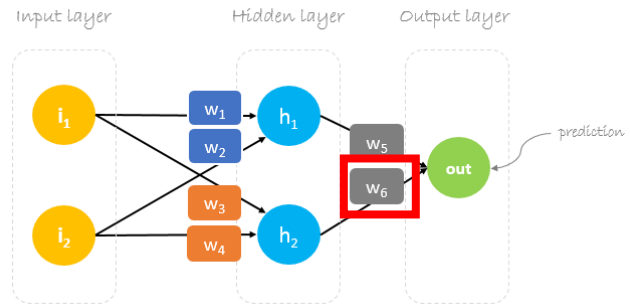
Learning rate



$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

Credit to hmkcode.github.io for example

Forward-backward by example



Step 4: Backpropagate error.

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ ← chain rule

Error $= \frac{1}{2}(prediction - actual)^2$

prediction $= (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(prediction - actual)^2}{\partial prediction} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(prediction - actual) \frac{\partial (prediction - actual)}{\partial prediction} * (i_1 w_3 + i_2 w_4)$$ ← $h_2 = i_1 w_3 + i_2 w_4$

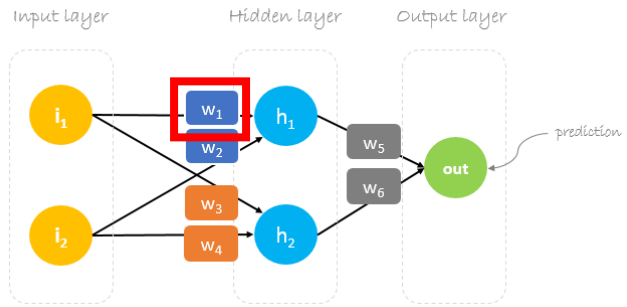
$$\frac{\partial Error}{\partial W_6} = (prediction - actual) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Credit to hmkcode.github.io for example

Forward-backward by example



Step 4: Backpropagate error.

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(prediction - actual)^2}{\partial prediction} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(prediction - actual) \frac{\partial (prediction - actual)}{\partial prediction} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (prediction - actual) * (w_5 i_1)$$

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Error = $\frac{1}{2}$(prediction – actual)²

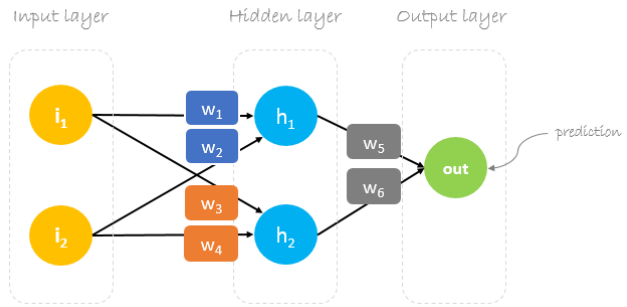
prediction = $(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

Δ = prediction – actual ← delta

Credit to hmkcode.github.io for example

Forward-backward by example



Step 5: Update weights.

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

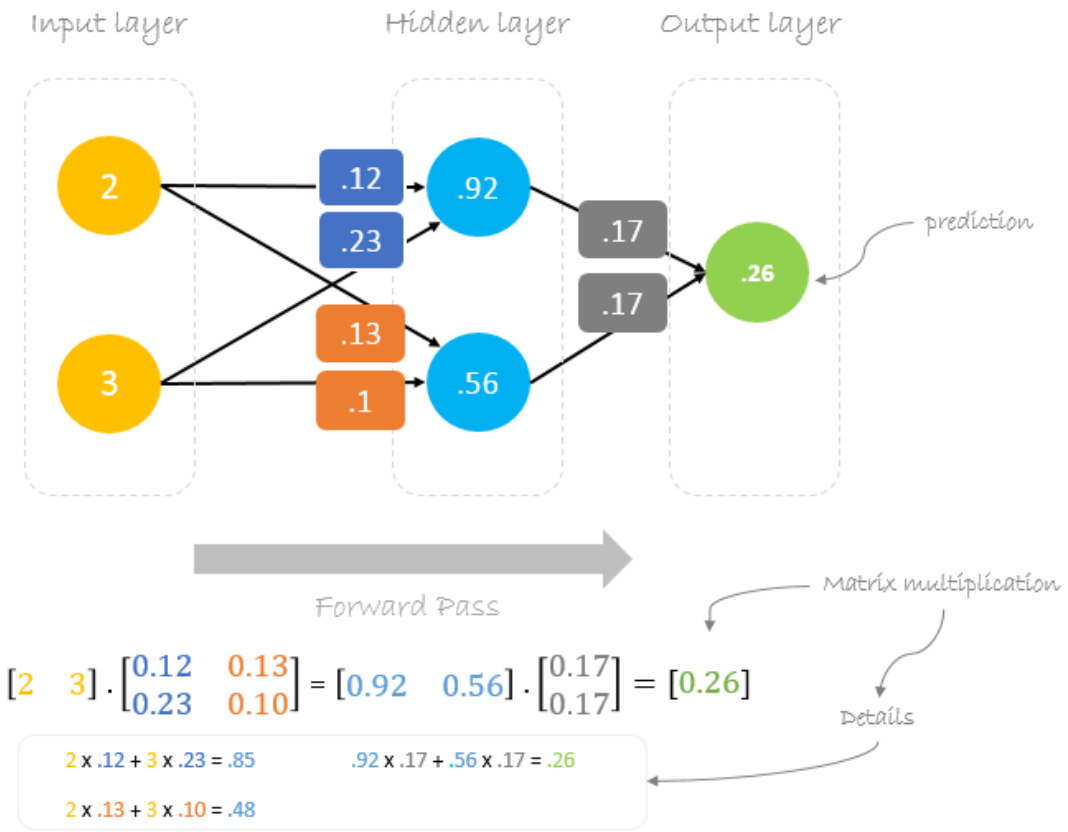
$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.[0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Credit to hmkcode.github.io for example

Forward-backward by example

Step 6: Repeat.



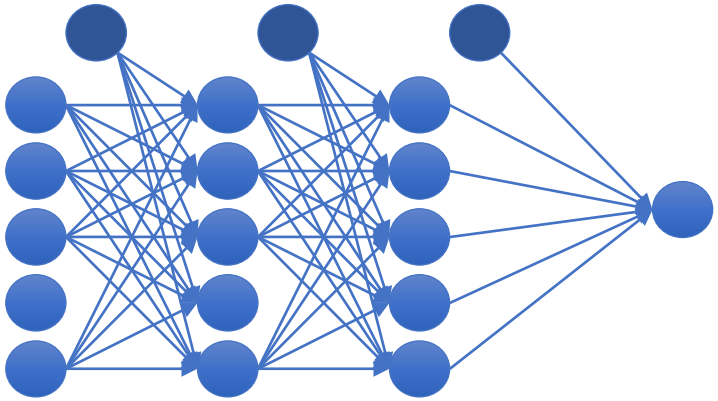
Binary classification

Akin to regression, one output value.

Apply sigmoid over output to obtain ”probability” output.

Binary cross-entropy loss for optimization:

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1-y) * log(1-\hat{y}_i))$$

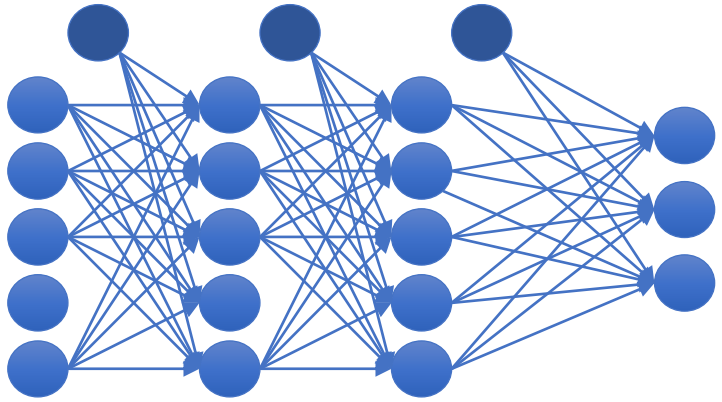


Multi-class classification

Multiple outputs: one per class.

Now apply softmax function over all outputs:

```
def softmax(X):
    exps = np.exp(X)
    return exps / np.sum(exps)
```

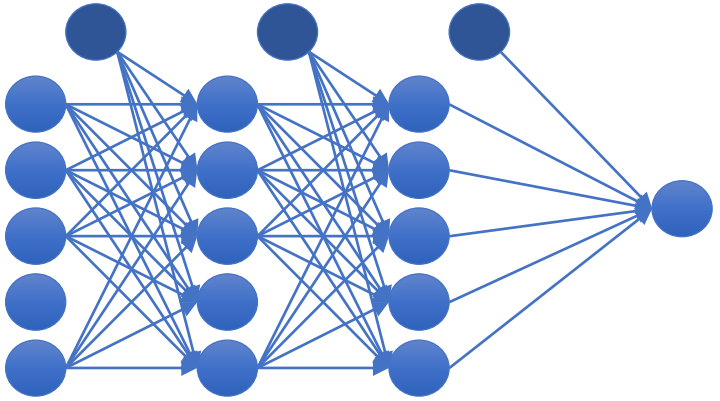


Cross-entropy loss:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Deep learning for images

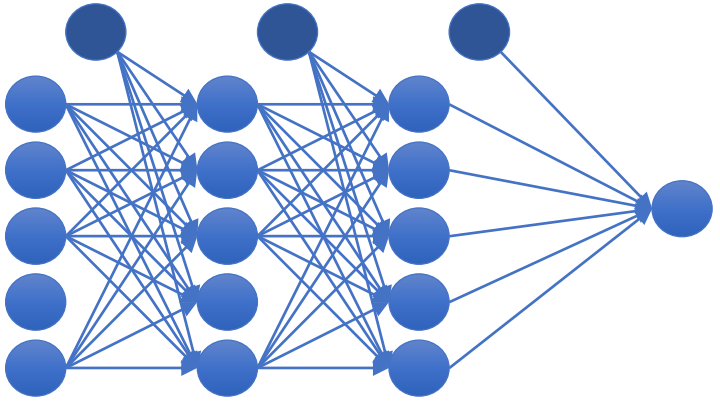
MLPs for images



Two reasons why directly applying images to MLPs is suboptimal:

- Many, many parameters needed.
- All the spatial structures are lost.

Towards a network for vision



We need a layer with two properties:

- Features with local spatial structure.
- Shared features across spatial regions of the image.

The convolutional operation

Linear operation: $f(x, w) = x^T w$

- Global operation, separate weight per feature
- Dimensionality of w = dimensionality of x
- 1 output value

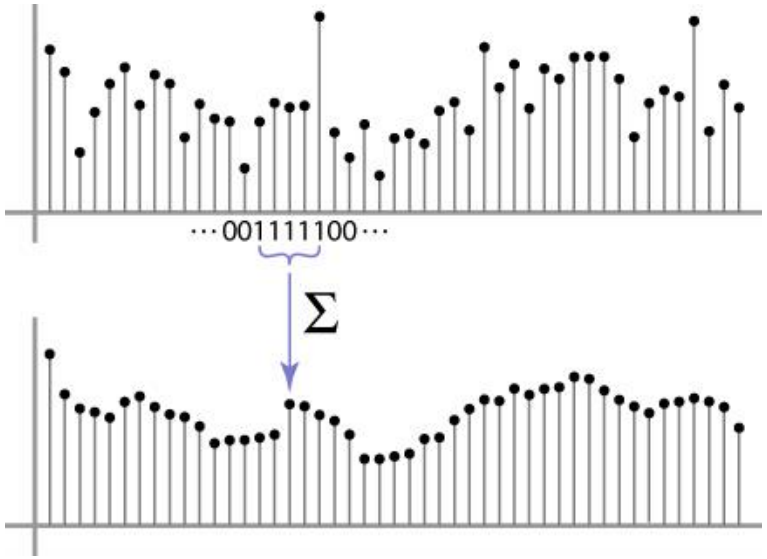
Convolutional operation: $f(x, w) = x * w$

- Local operation, shared weights over local regions
- Dimensionality of w << dimensionality of x
- Output same size as input

1D convolutions

Transform data points using neighbors on the line.

Transformation determined by the weights of the convolutional filter.



Weights: [1 1 1 1 1] / 5

Q1: Why is there a division by 5 in the weights?
Q2: Why is the filter of uneven size?
Q3: What happens when we use a smaller/larger filter size?

2D convolutions

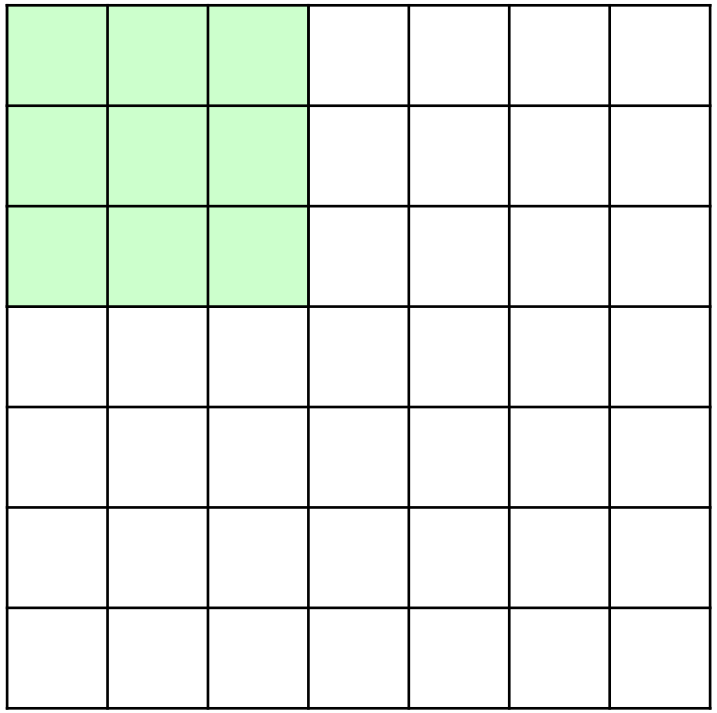
Input: Image F of size W x H

Filter: “Image” H of size K x K (K = set by hand)

Output: Image G of size W x H

$$G[x,y] = \sum_{i=-(K-1)/2}^{(K-1)/2} \sum_{j=-(K-1)/2}^{(K-1)/2} H[x+i, y+i] * F[x+i, y+i]$$

2D convolutions

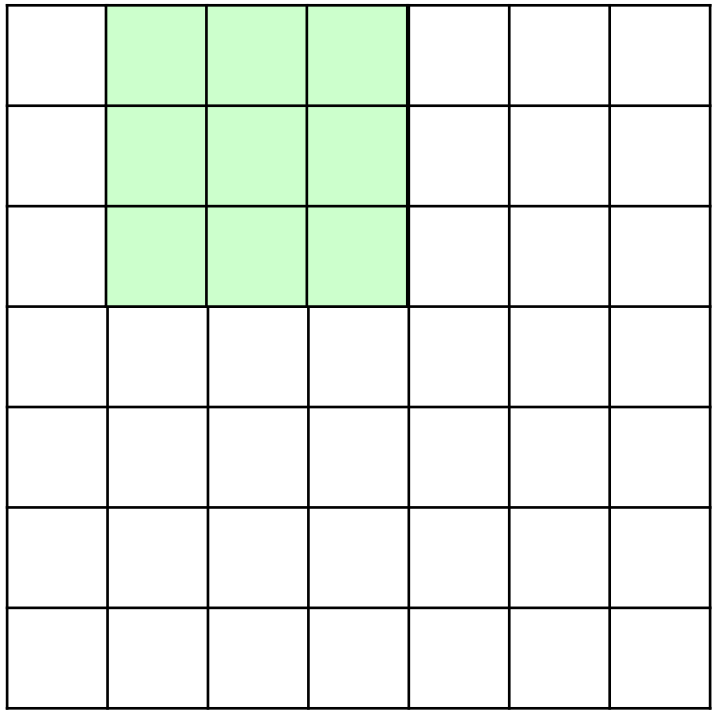


Input image: 7x7

Filter size: 3x3

Move filter over image and repeat operation for each location

2D convolutions

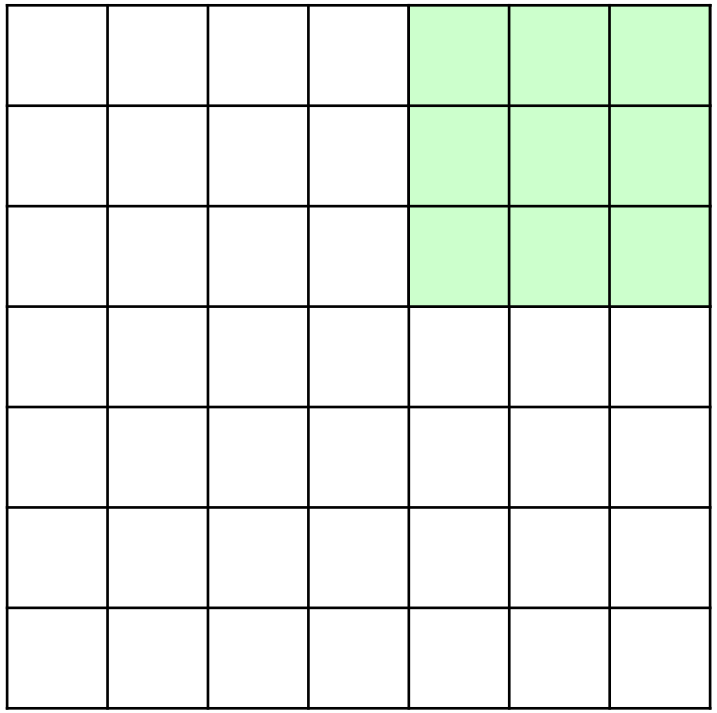


Input image: 7x7

Filter size: 3x3

Move filter over image and repeat operation for each location

2D convolutions



Input image: 7x7

Filter size: 3x3

Move filter over image and repeat operation for each location

What is the size of the output?

Dealing with the border

Now, we shrink the image size with each convolution.

Possible solutions to keep same dimensionality:

Zero padding | Wrap around | Copy | Reflect

Source: S. Marschner

Short break

Practice examples

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice examples

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice examples

Original

0	0	0
0	0	1
0	0	0

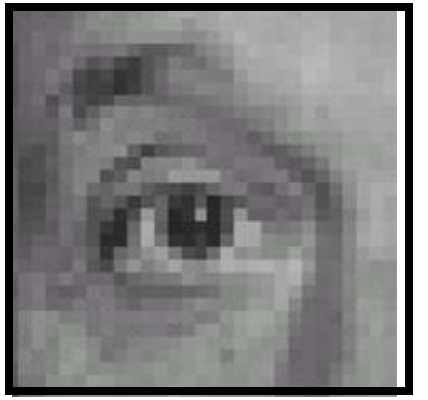
?

Source: D. Lowe

Practice examples

Original

0	0	0
0	0	1
0	0	0



Shifted left
by 1 pixel

Source: D. Lowe

Practice examples

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

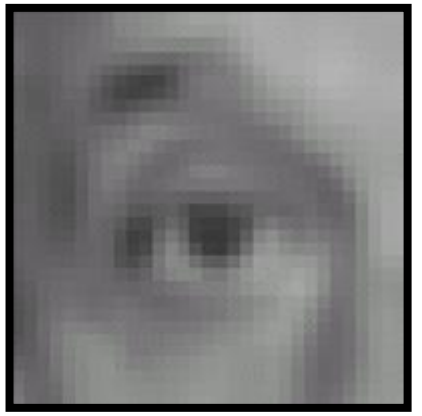
?

Source: D. Lowe

Practice examples

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Source: D. Lowe

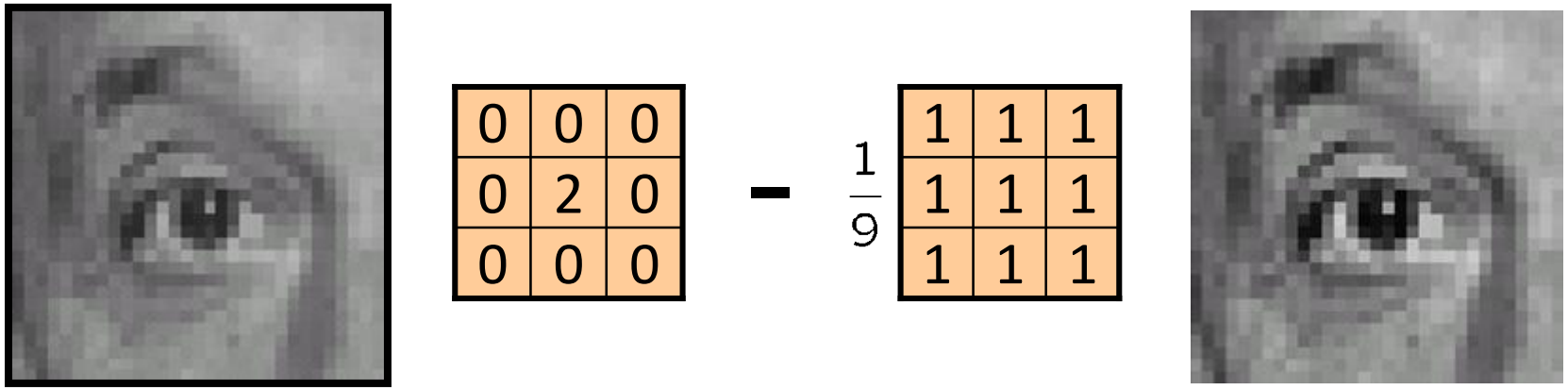
Practice examples

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad ?$$

Source: D. Lowe

Practice examples



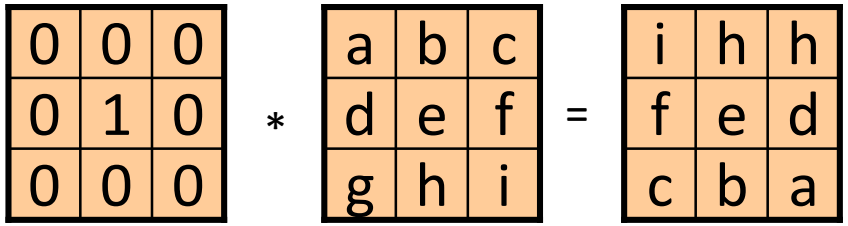
Original

Sharpening

Source: D. Lowe

Convolution errata

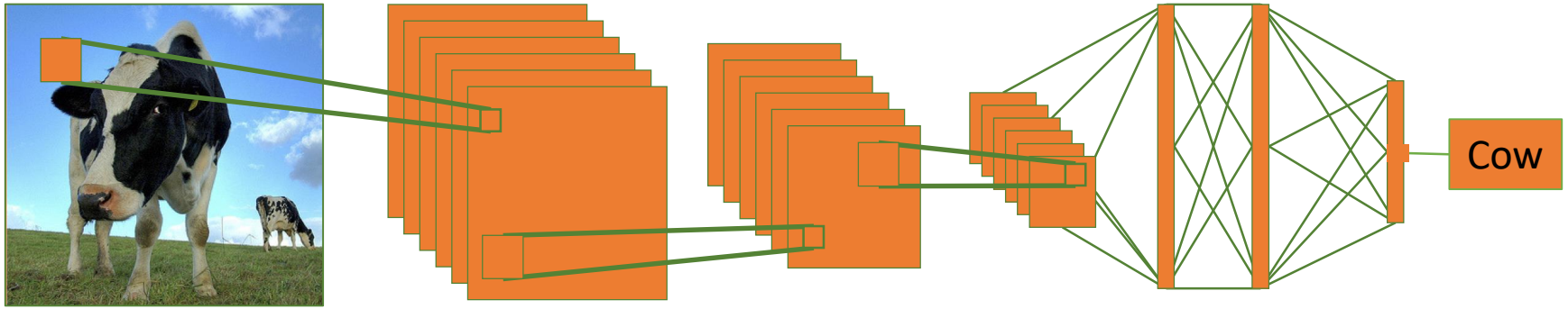
Technically, we have been doing cross-correlation.



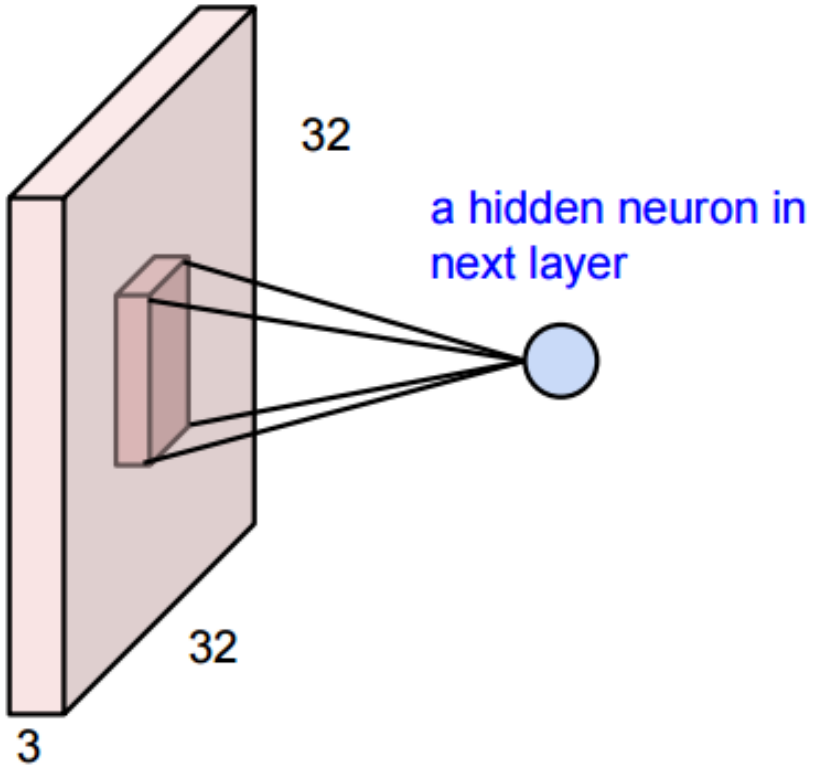
For a convolutional operation, we need to flip the filter horizontally and vertically first, to preserve identity.

Only matters for asymmetrical filters, and is irrelevant in convolutional networks since we learn the filters.

Convolutional networks



Local connectivity



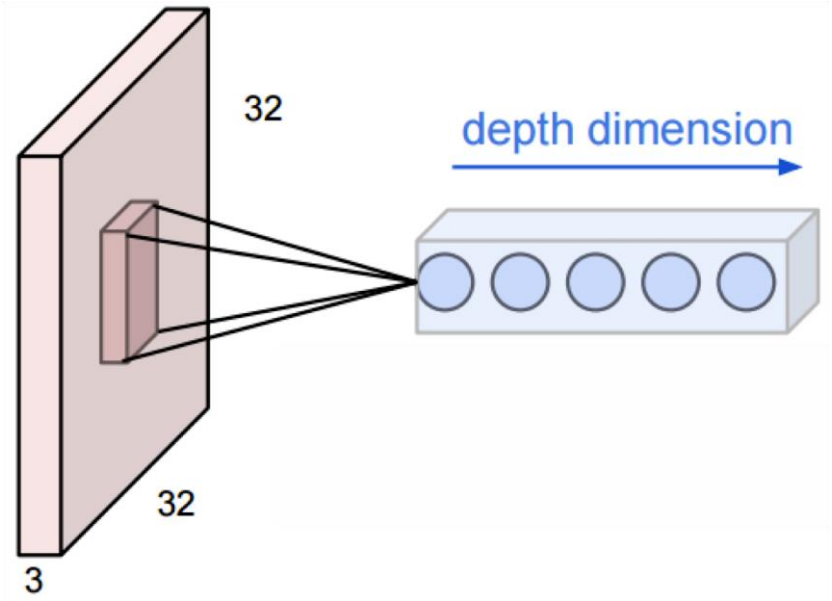
One output neuron is a combination of a local input patch

Connectivity is:

- Local in space
- Full in depth

Convolutional networks on images always have a third dimension!

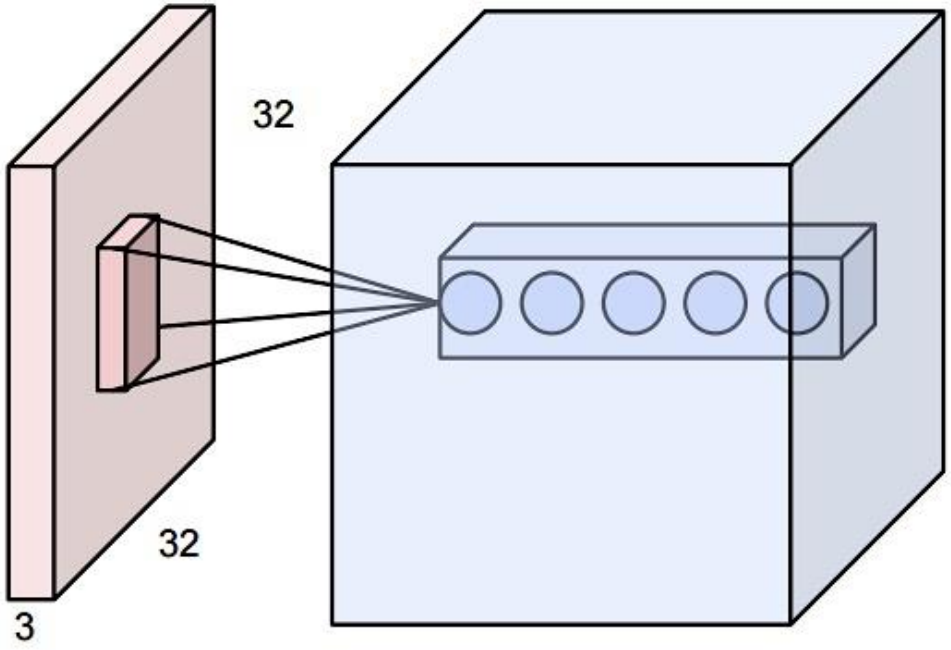
Multiple neurons



Each filter yields a different output value for the same location.

We stack the outputs in the depth dimension of the new layer.

Multiple neurons

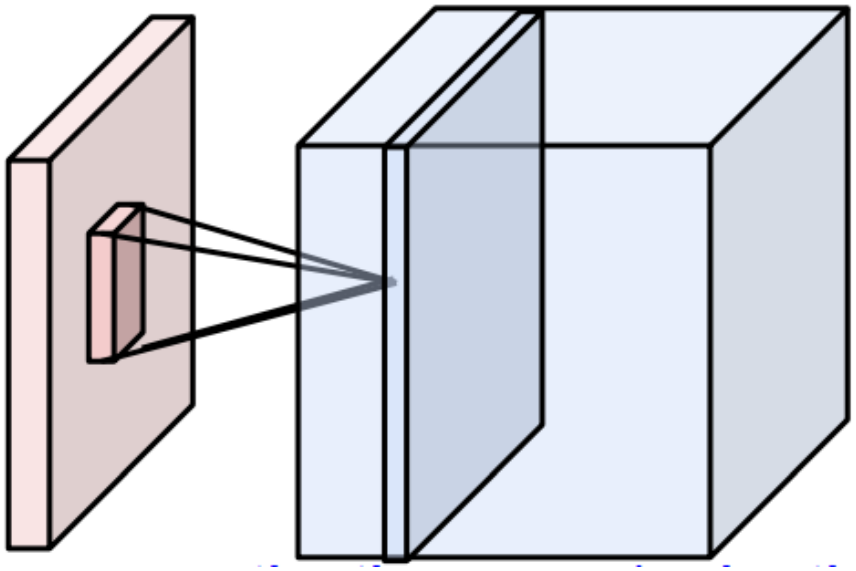


Given input size 32x32x3

What is the output size when using D filters?

Does the output size depend on the filter size?

Filter maps



The outputs of a single filter over all spatial locations is called a **filter map**

Filter and layer dimensionality

First layer:

Input 32x32x3, D_1 filters of size 5x5x3

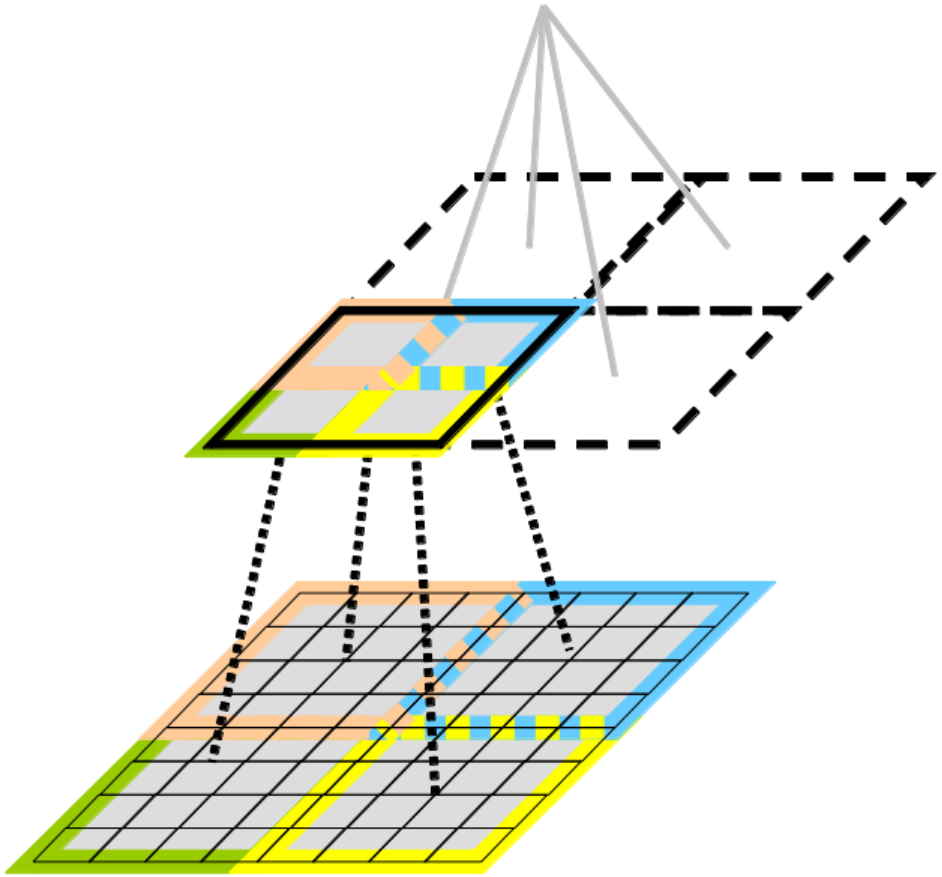
Output: 32x32xD_1

Second layer:

Input: 32x32xD_1, D_2 filters of size 5x5x[ ]

Output size: [ ]

Pooling



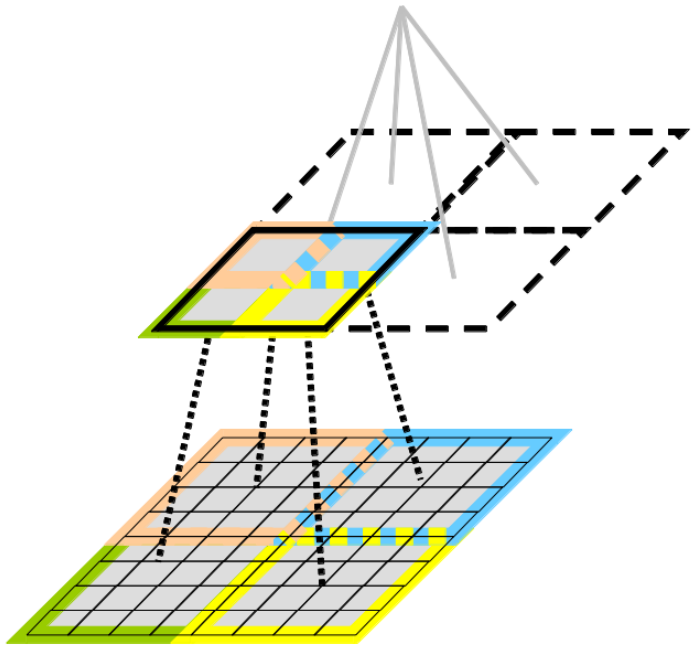
Pooling image regions

Two reasons for pooling:

- Invariance to small deformations
- Faster computation

Two ways of pooling:

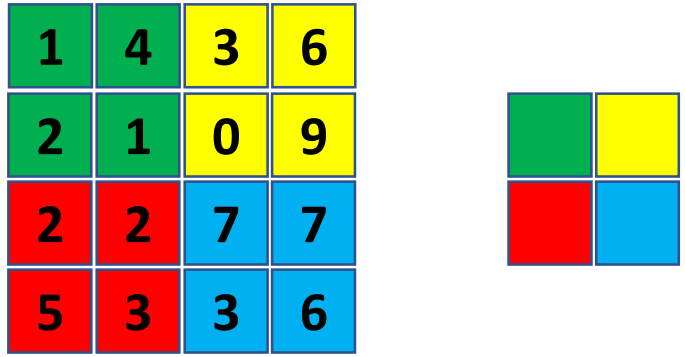
- Mean pooling
- Max pooling



Max pooling

Activation function:

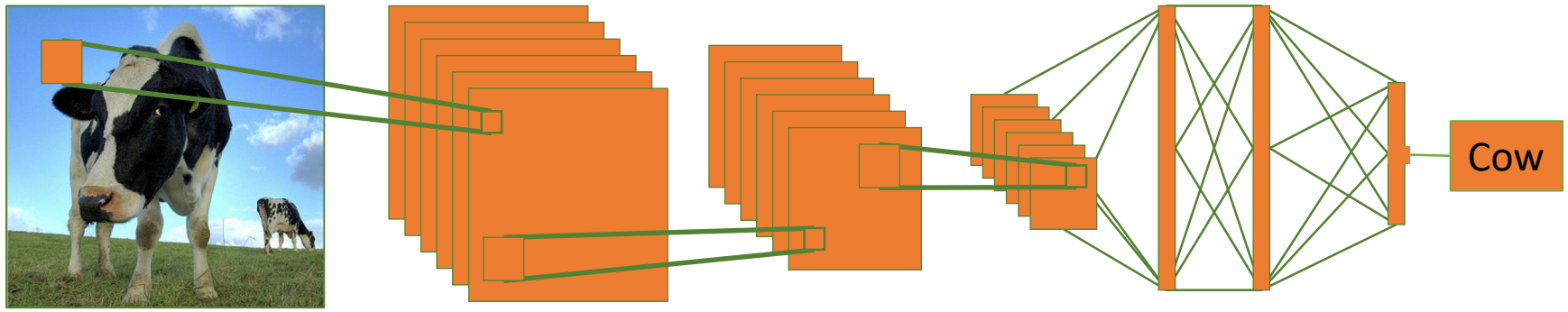
$$\mathrm{i}_{\max}, \mathrm{j}_{\max} = \underset{i,j \in \Omega(r,c)}{\arg\max}\, x_{ij} \rightarrow a_{rc}$$



Gradient w.r.t. input

$$\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$$

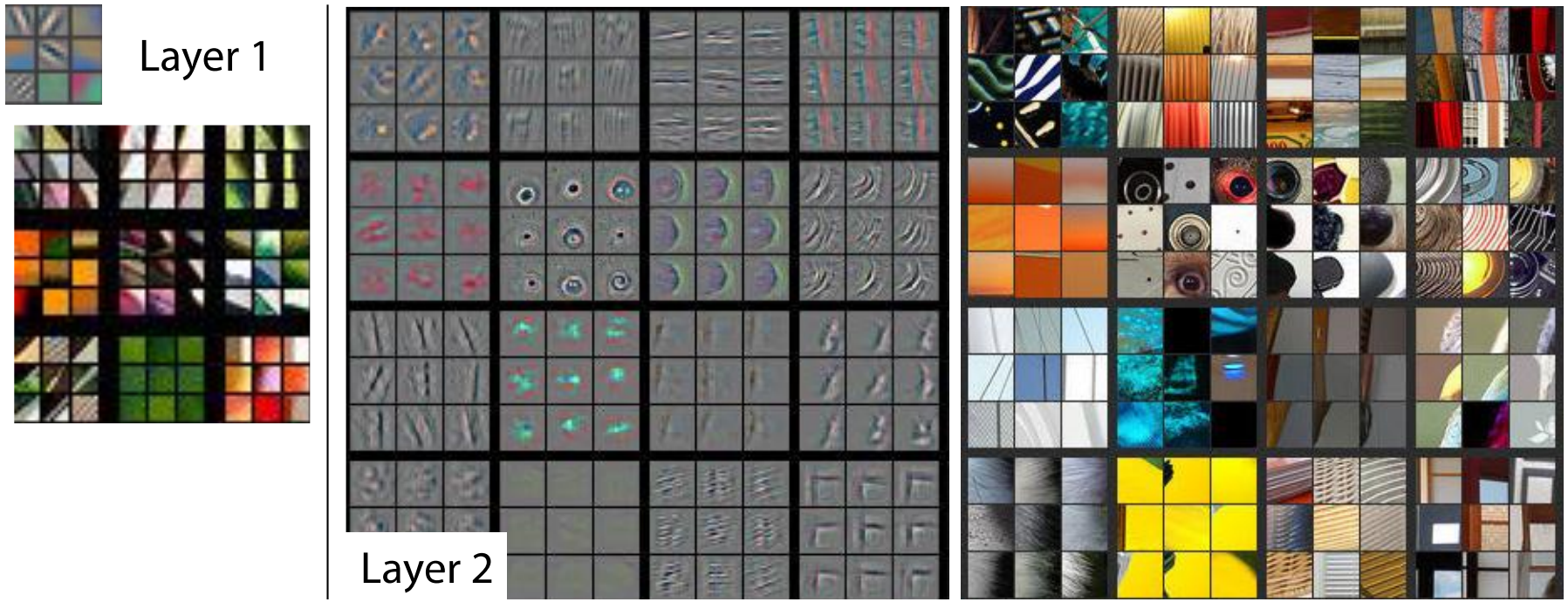
The Convolutional Network



- Learn local filters at each layer.
- Aggregate local filter with a spatial neighborhood.
- Same filters for each spatial area.
- Downsize output maps over layers with pooling.
- Final layers are standard network layers.

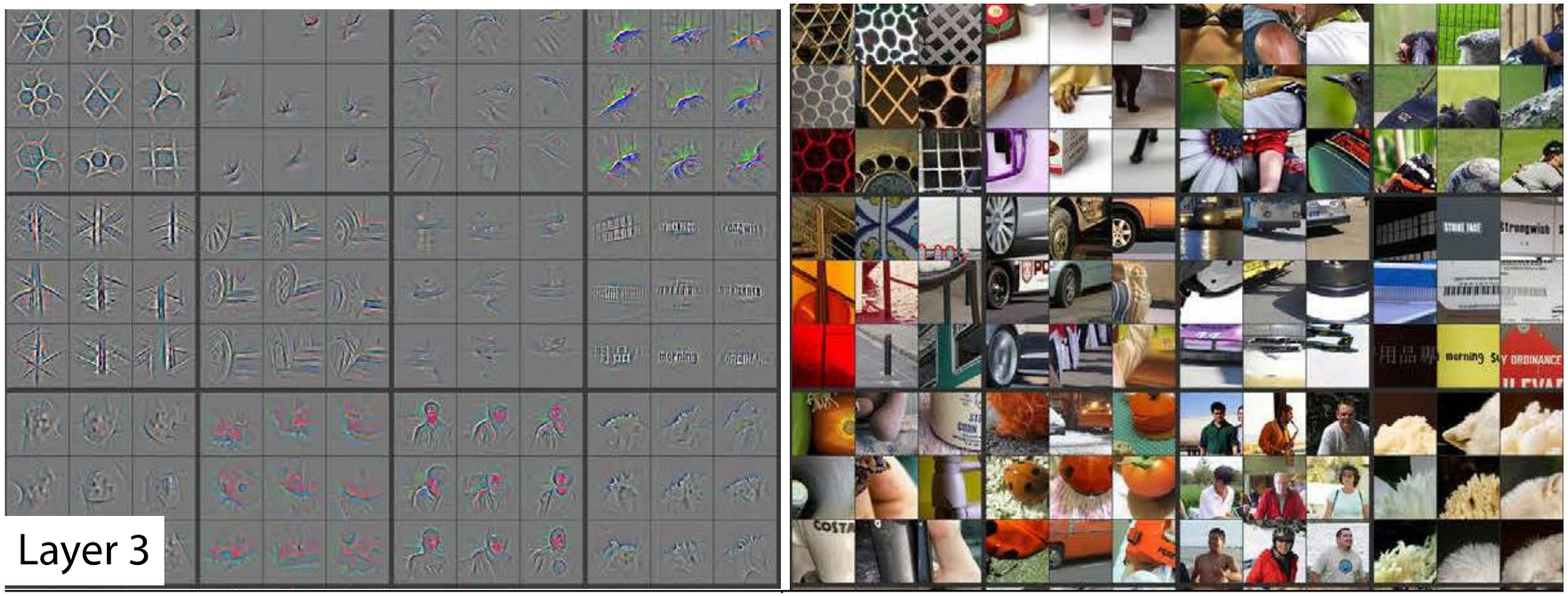
[Zeiler and Fergus. ECCV, 2014]

What does each layer learn?



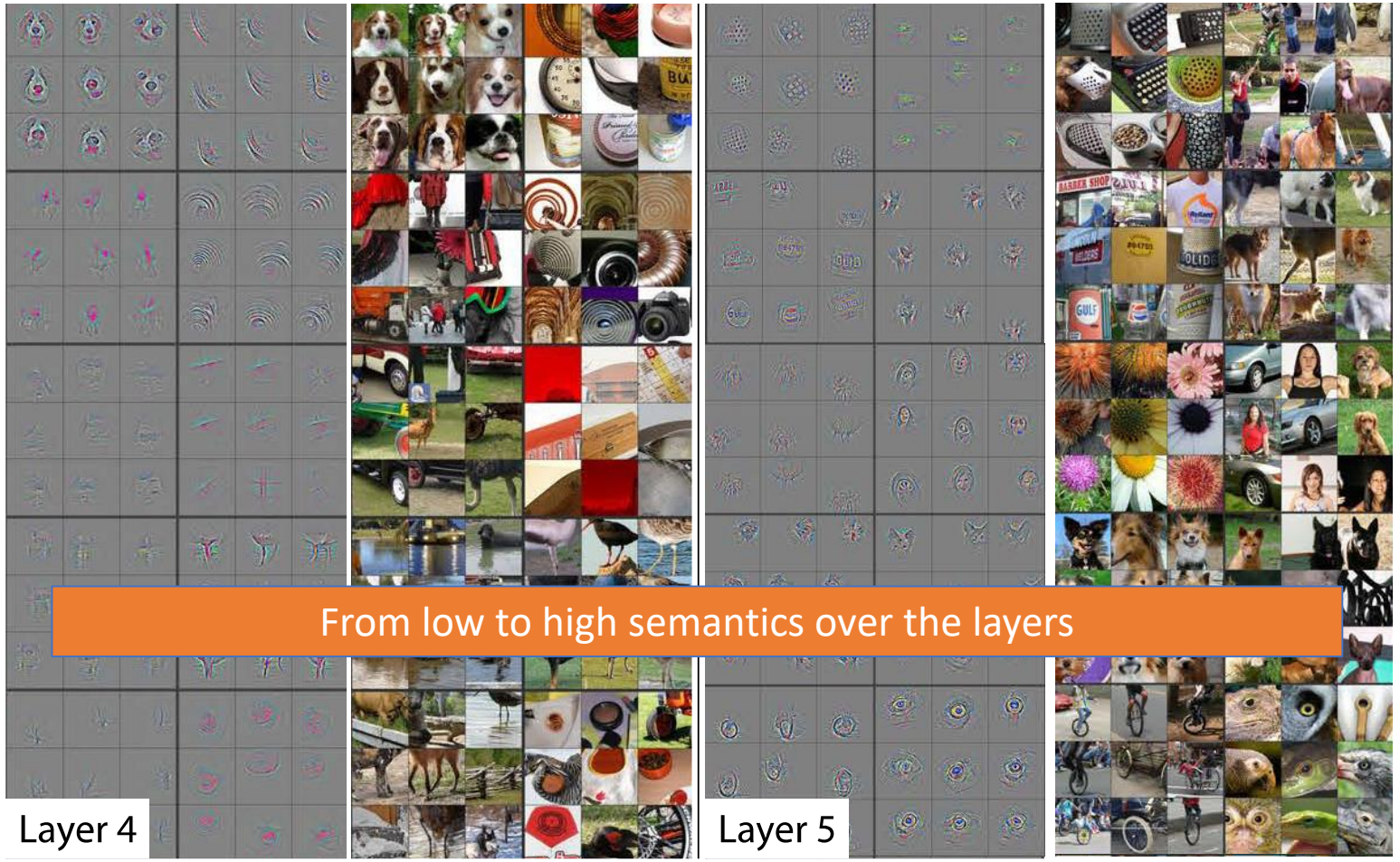
[Zeiler and Fergus. ECCV, 2014]

What does each layer learn?



[Zeiler and Fergus. ECCV, 2014]

What does each layer learn?



[He et al. CVPR, 2016]

Improving ConvNets: Residual connections

What looks weird about the lines in the figures below?

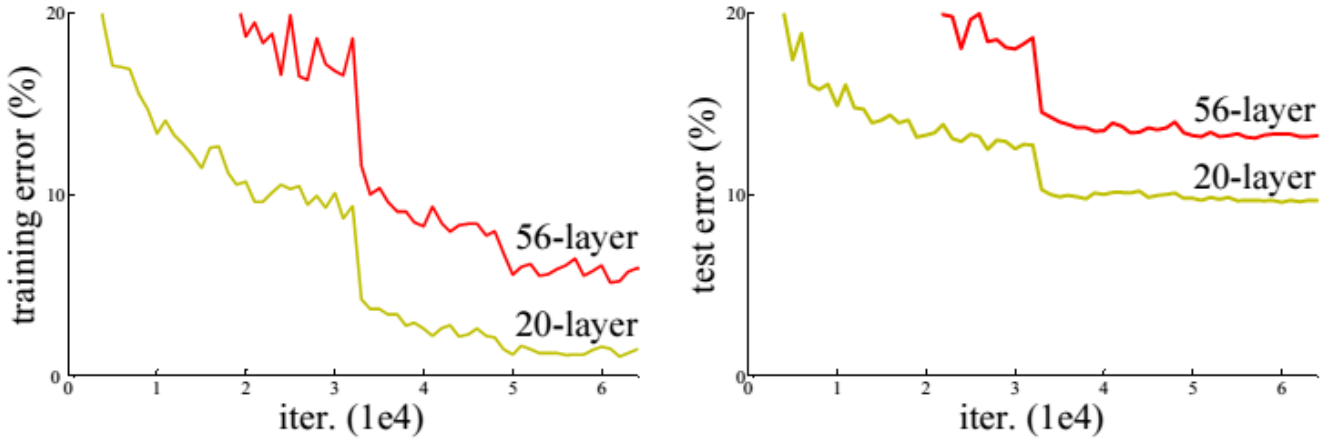


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

[He et al. CVPR, 2016]

ResNet: Main idea

Layer models residual $\mathrm{F}(\mathrm{x}) = H(x) - x$ instead of $H(x)$

If anything, the optimizer can simply set the weights to 0

Adding identity layers should lead to larger networks that have at least lower training error

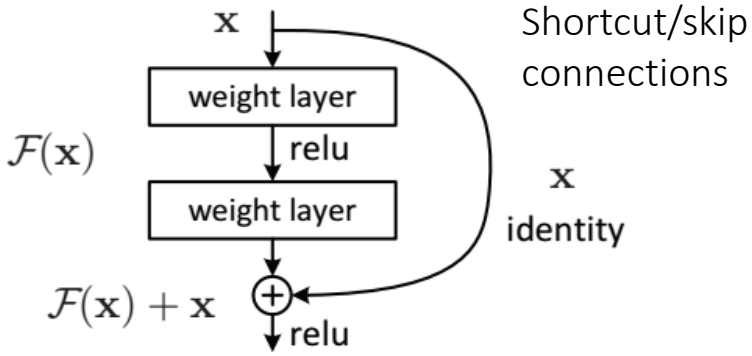
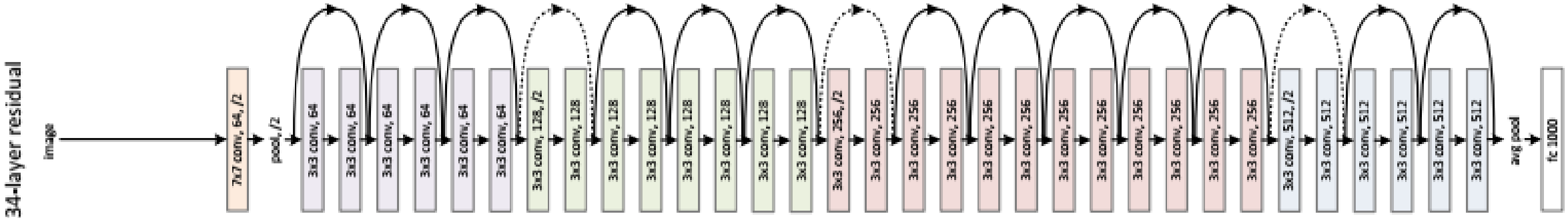


Figure 2. Residual learning: a building block.

[He et al. CVPR, 2016]

ResNet



$$x_{l+1} = x_l + \mathrm{F}(\mathrm{x_l}) \qquad x_{l+2} = x_{l+1} + \mathrm{F}(\mathrm{x_{l+1}}) \qquad \ldots \; x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

[He et al. CVPR, 2016]

No degradation anymore

With residual connections, deeper networks are trainable.

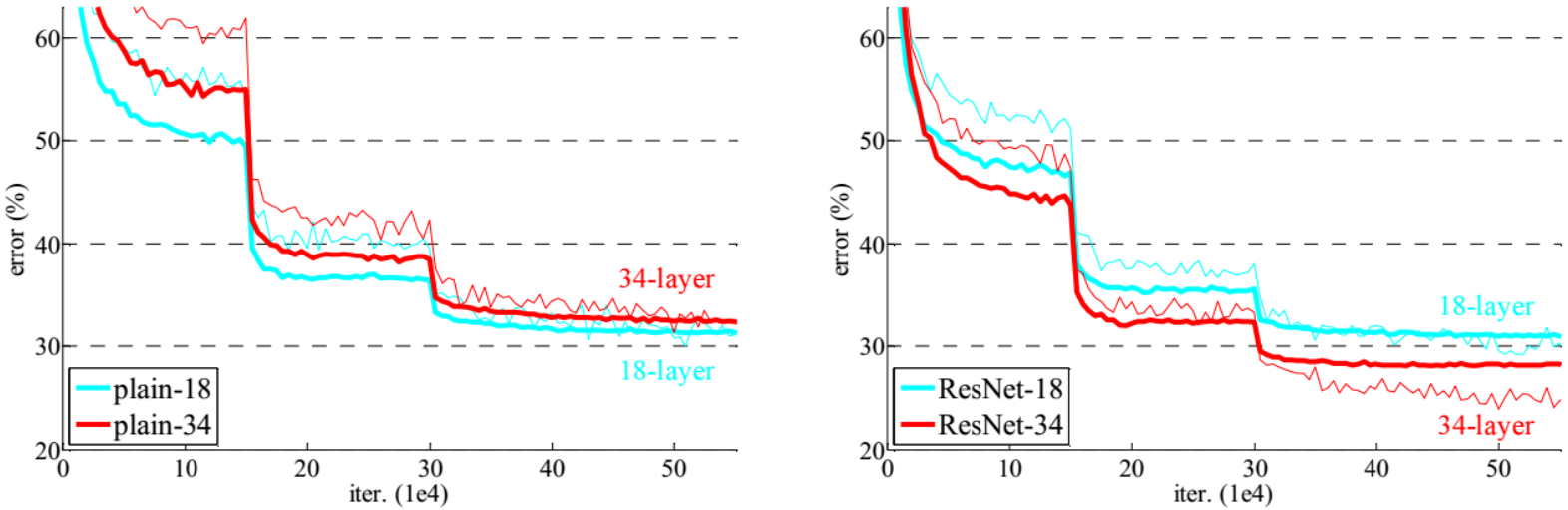
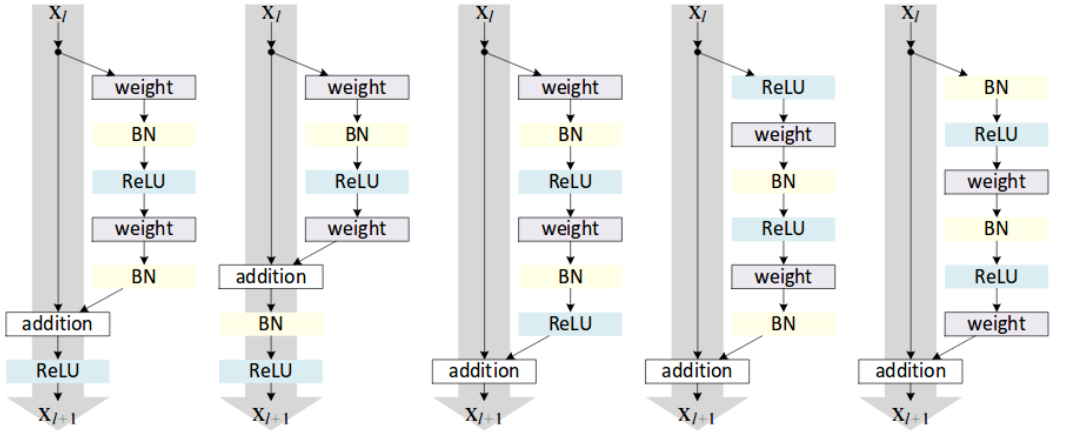


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

[Xie et al. CVPR, 2017]

ResNext



case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

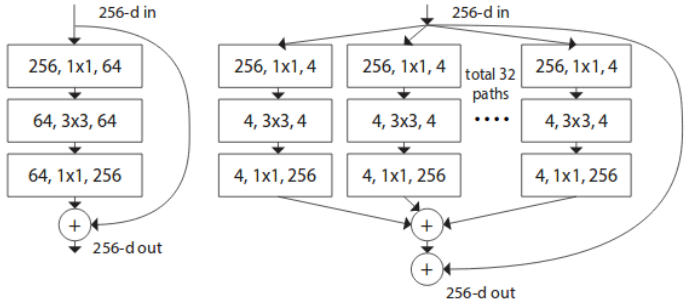


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

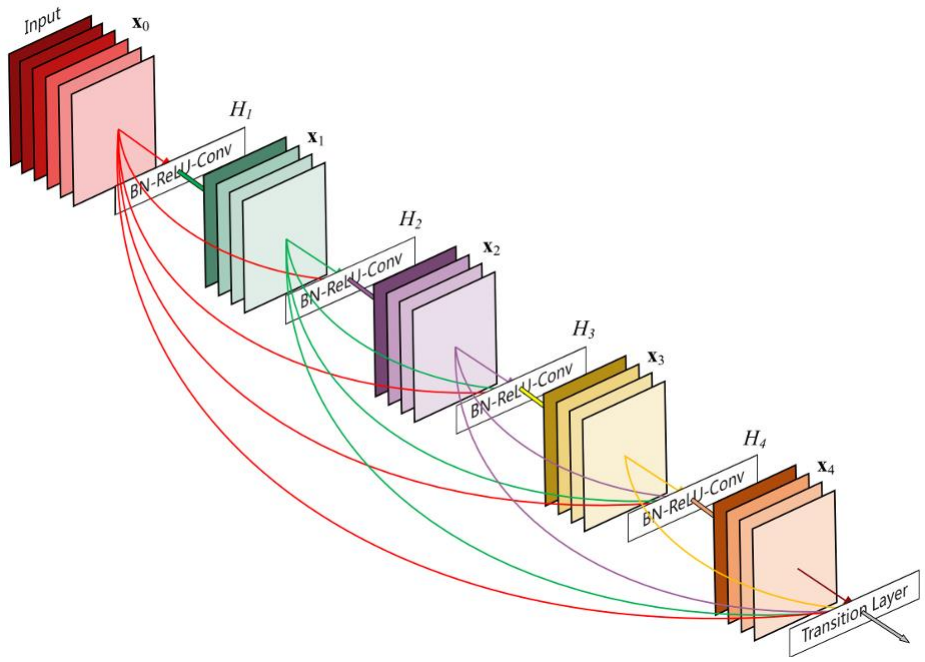
Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

[Huang et al. CVPR, 2017]

Improving ConvNets: Dense connections

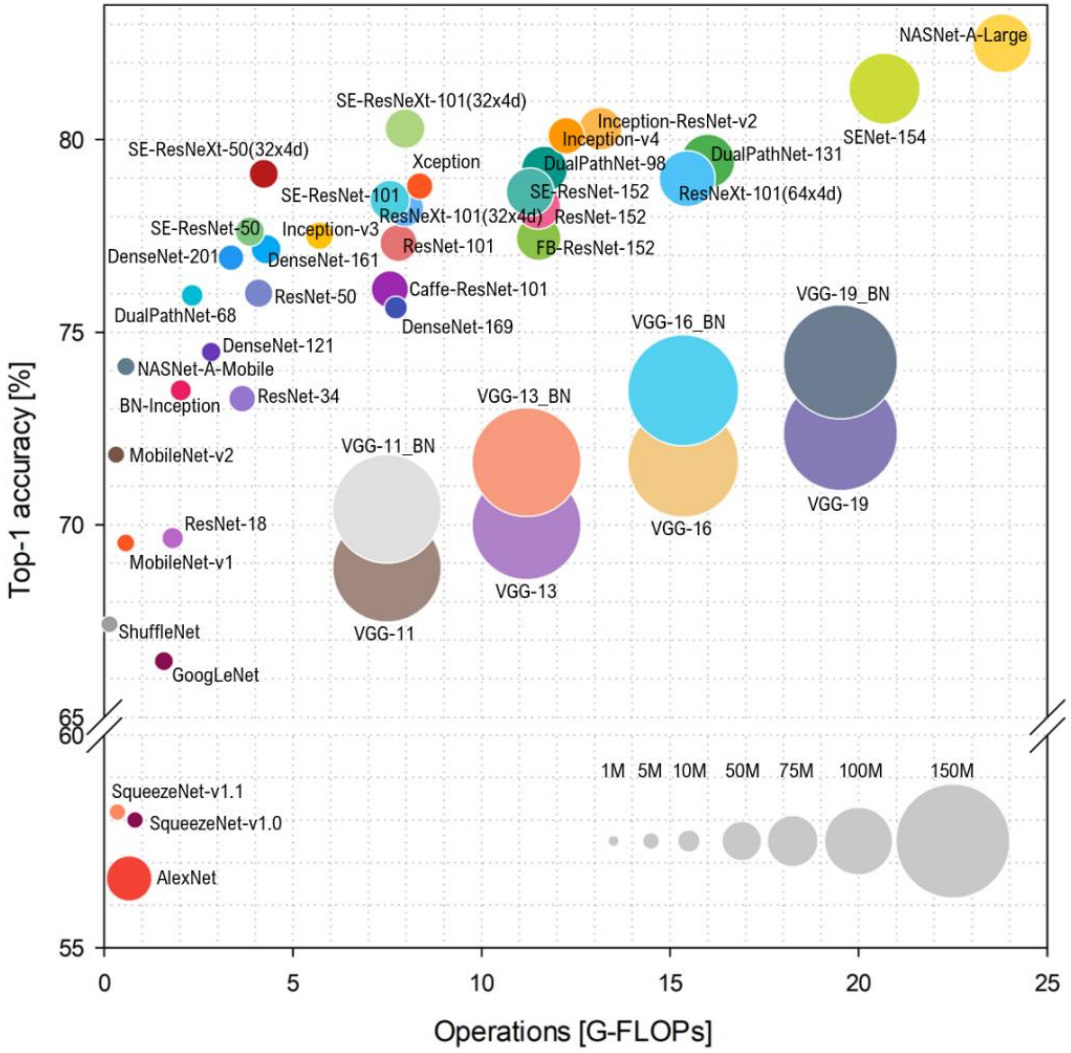
Rather than having single skip connections, connect all pairs of convolutional layers.

Further improvements, at a loss of simplicity and speed.



[Canziani et al. arXiv, 2016]

Recent overview of ConvNets



ConvNets in practice

Many additional tricks required to make deep learning effective:

Hyperparameter tuning.

Batch normalization.

Hyperparameter tuning.

Regularization (weight decay, DropOut, ...).

Hyperparameter tuning.

Deep learning for videos

From images to videos

Input no longer 1 frame, but a collection of ordered frames.

Many new problems:

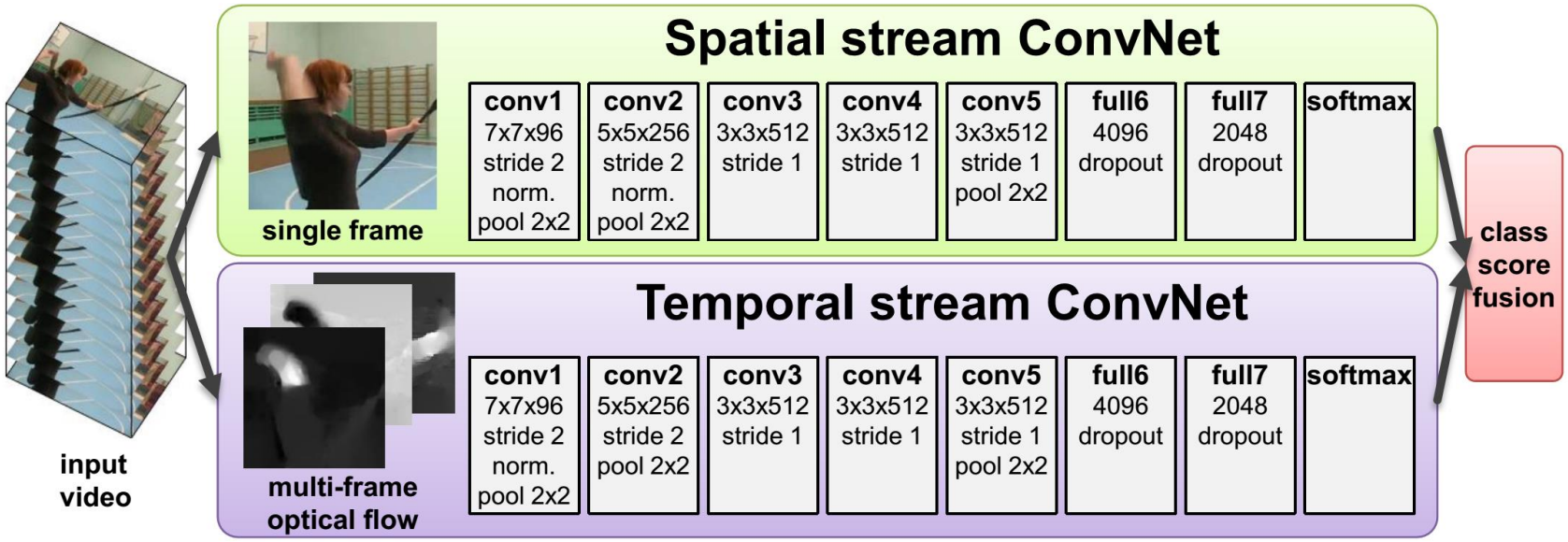
- Order of magnitude more input dimensions.
- Order of magnitude less examples.
- How to deal with the temporal dimension?

[Simonyan and Zisserman. NeurIPS, 2014]

Two-stream networks

Main idea: One network classifies from single frames one from flow.

For the flow stream: stack multiple flow images before prediction.



[Pigou et al. ECCVw, 2014]

3D convolutional networks

Temporal axis becomes extra input dimension.

Local convolutions in width, height, and time dimensions.

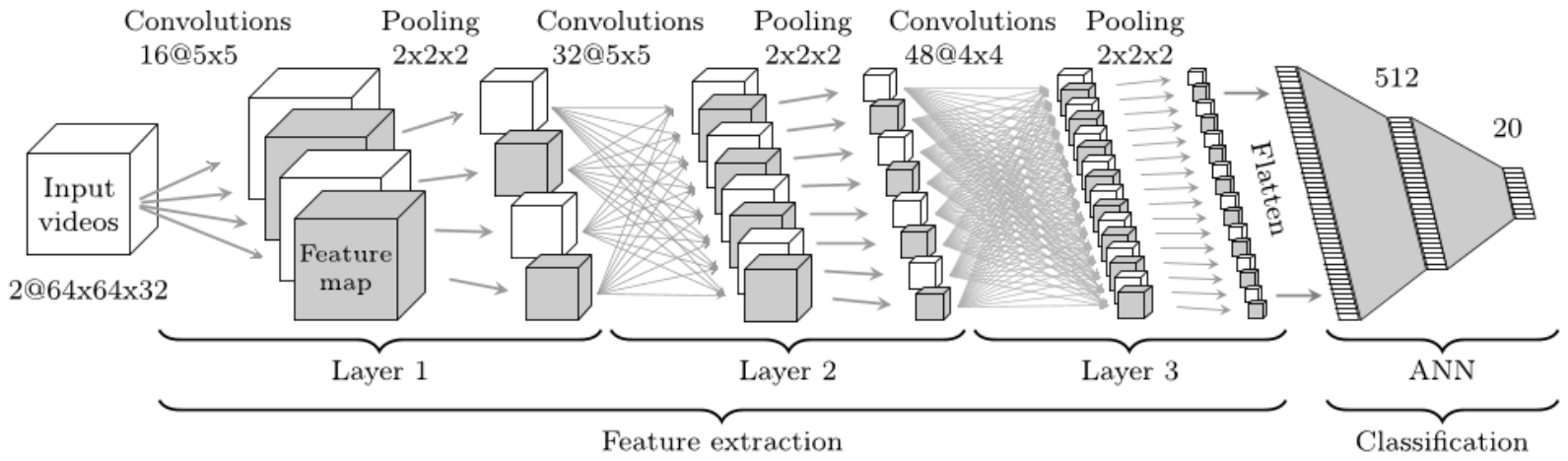


Image courtesy of: https://imatge-upc.github.io/activitynet-2016-cvprw/

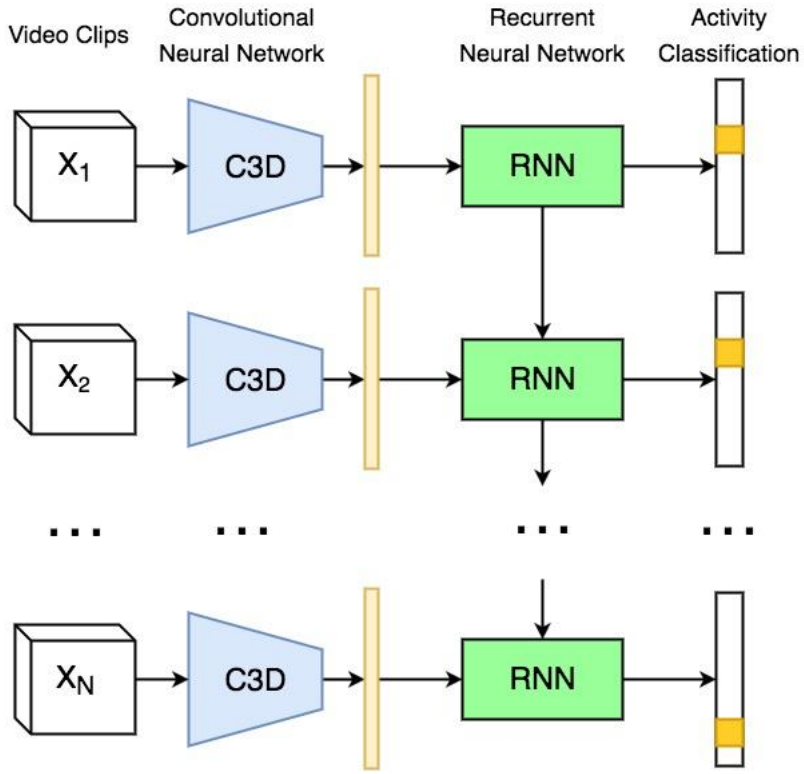
Recurrent networks

Exploit link between frames.

Recurrent networks have 2 inputs:

Current features and output of previous step.

Useful for many task, but hard to optimize.



A problem with video recognition

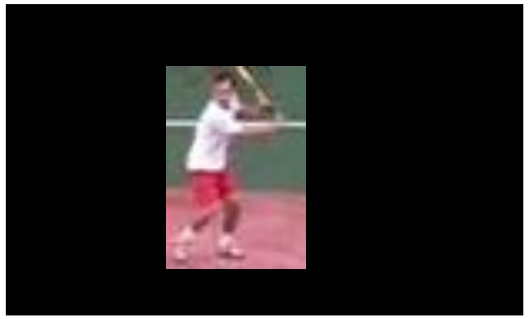
We train a deep video network on whole videos of UCF-101.

We test on 4 settings: what is the order from best to worst accuracy?

Normal videos

1

No background



4

No actor

2

Other actor

3

Longer break

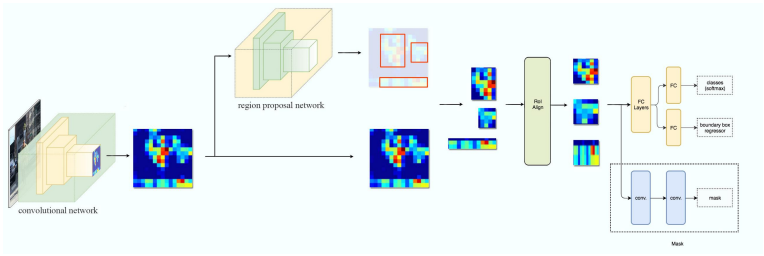
Topics of today

The Core

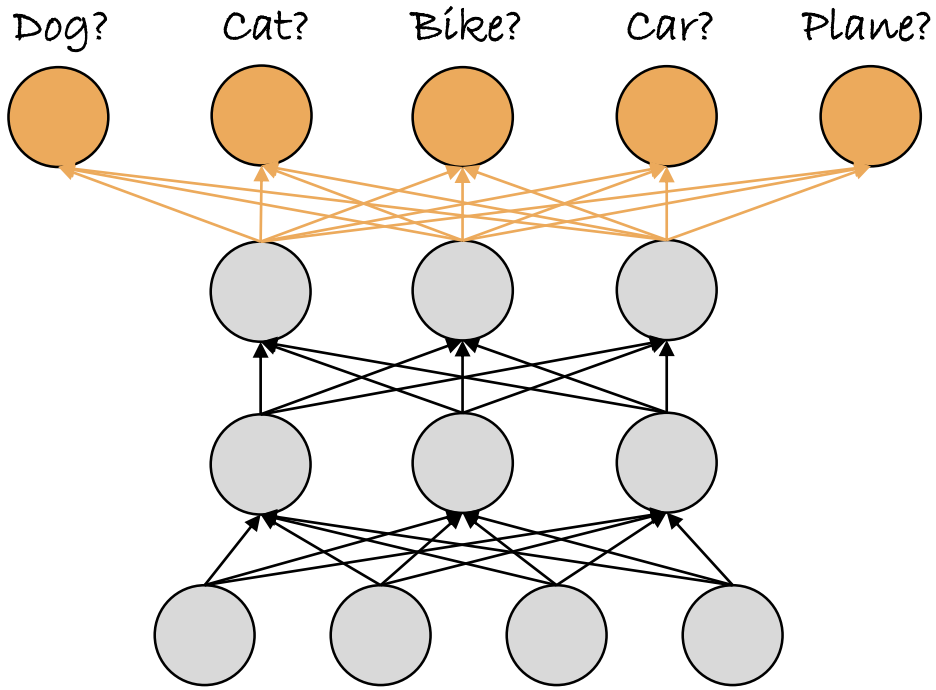
Single- and multi-layer perceptrons, forward/backward propagation, optimization, convolutional networks, state-of-the-art, video recognition

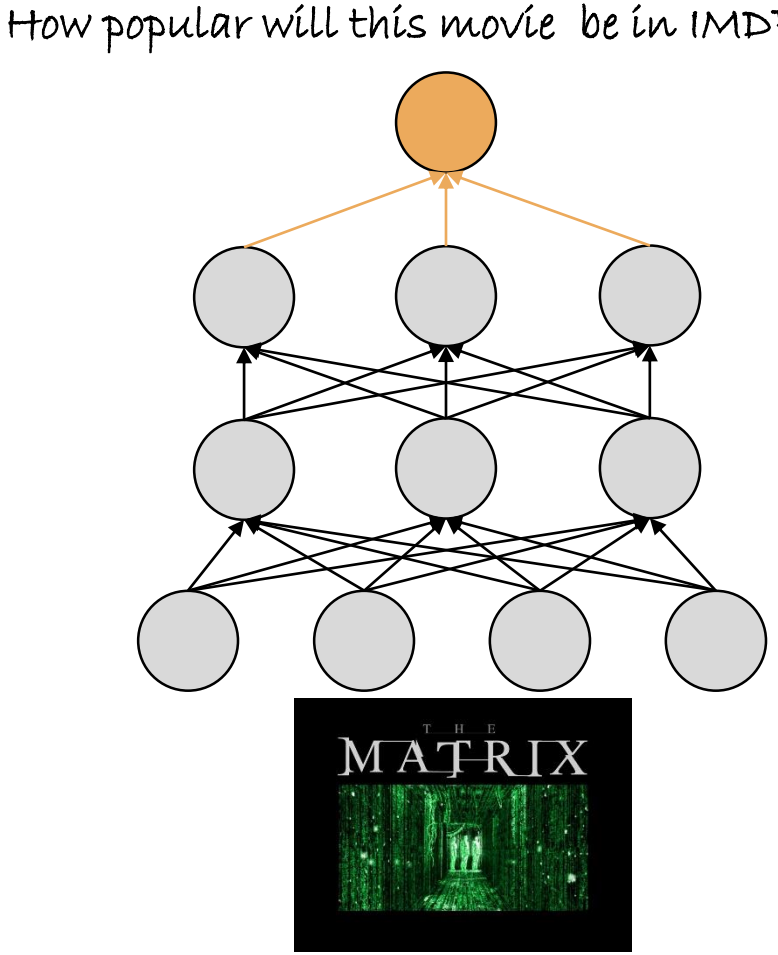
Beyond The Core

Object detection, per-pixel prediction, fully convolutional networks, geometric deep learning, hyperspheres



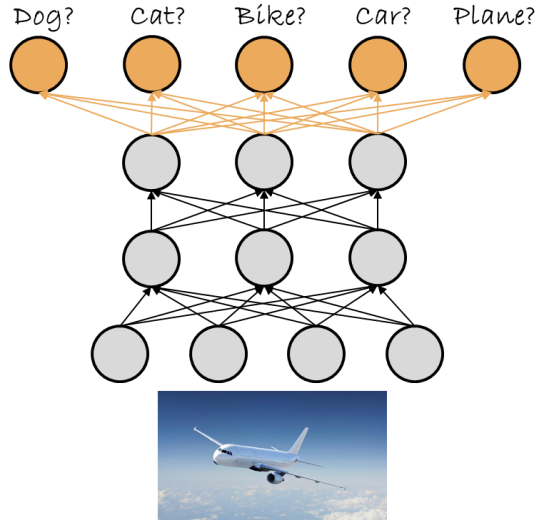
Story so far


Dog?
Cat?
Bike?
Car?
Plane?


How popular will this movie be in IMDB?
THE
MATRIX

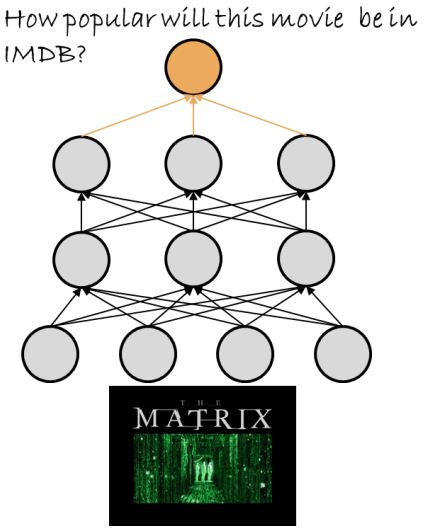
Beyond single value prediction

Not all tasks only need a single output.



What about:

- Localization?
- Per pixel outputs?
  - Segmentation, depth estimation, etc.
- Sequential predictions?

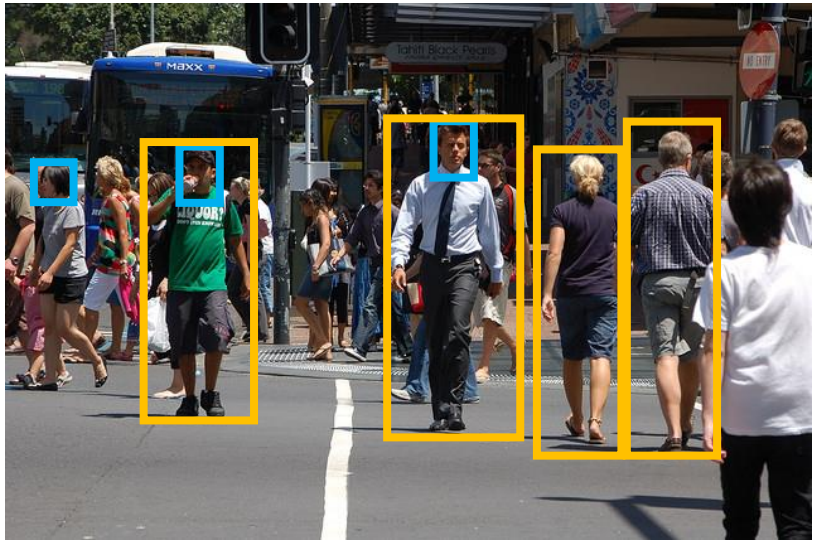


Deep learning for object detection

Object detection

Predict bounding box around each object of interest in a dataset.

Each image can have any number of objects, which can be small or even (partially) occluded.



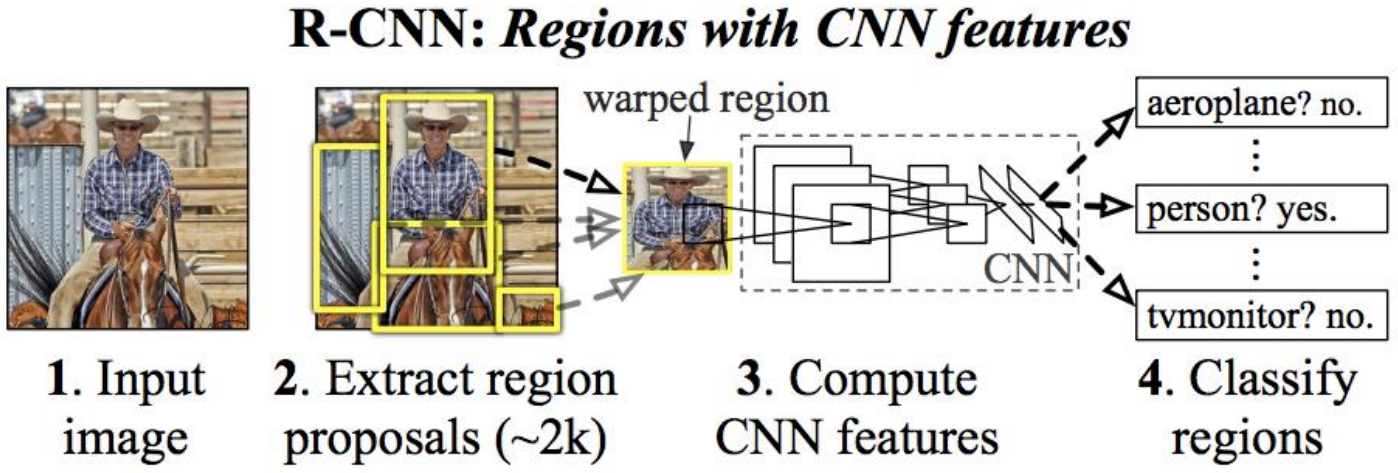
[Girshick et al. CVPR, 2014]

Deep Learning for Detection: R-CNN

Basic idea follows pipeline of “shallow” object detectors of that time.

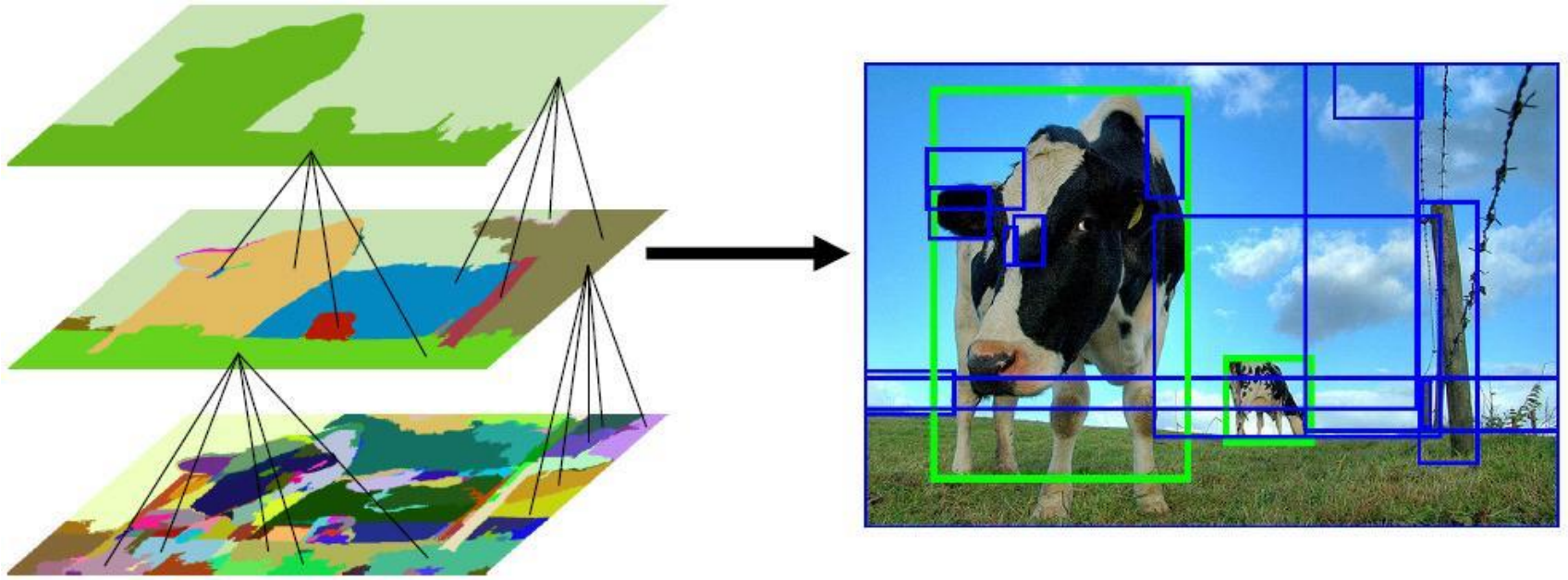
For each image, precompute object proposals.

Feed proposals to pre-trained deep network, rather than fixed features.



[Uijlings et al. IJCV, 2013]

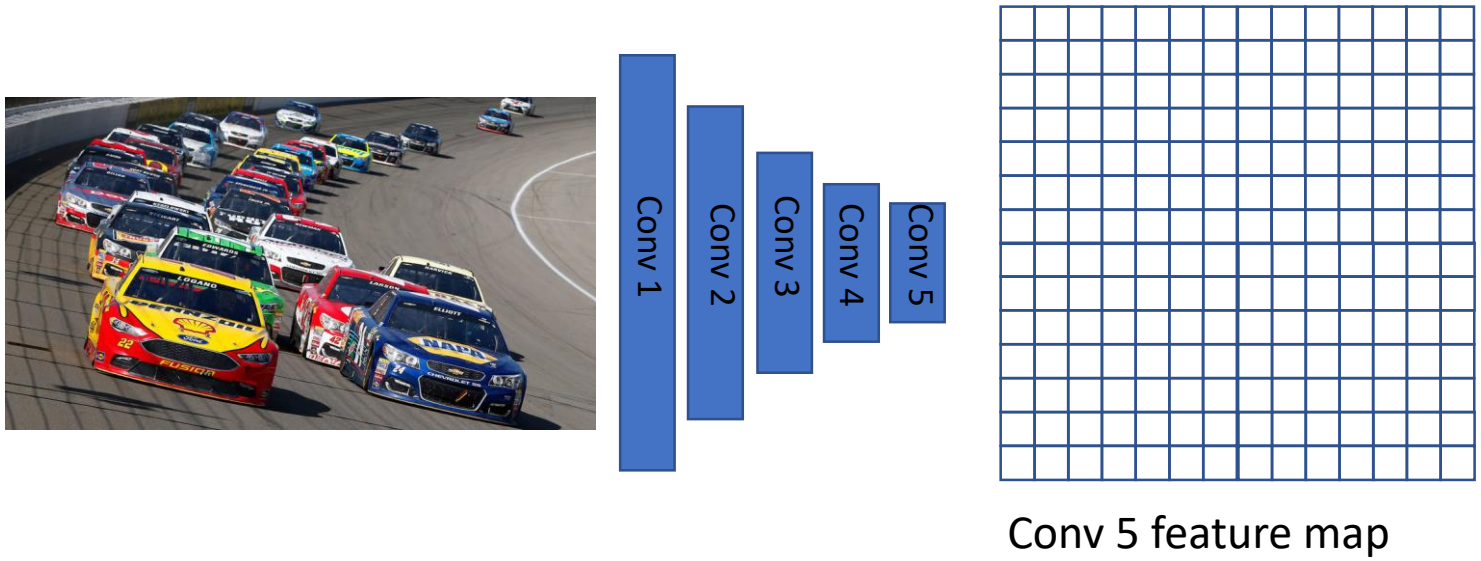
Backbone of R-CNN: Selective search



[Girshick. ICCV, 2015]

Deep Learning for Detection: Fast R-CNN

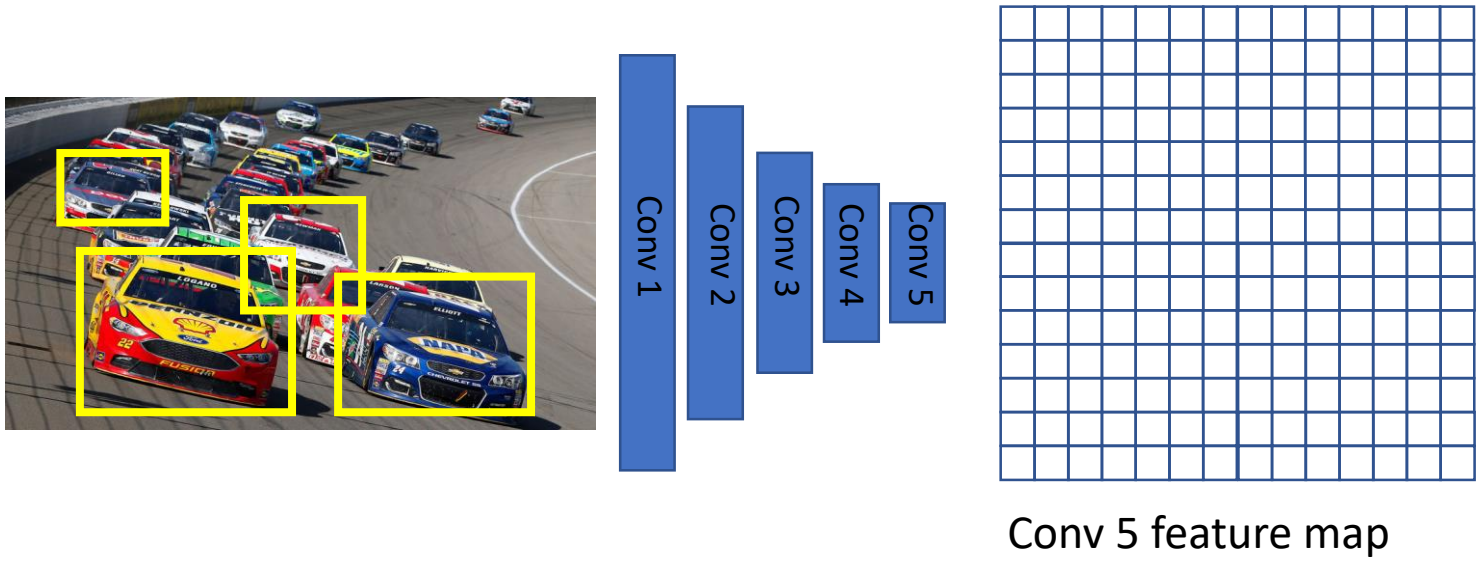
Making object proposal features part of the network.



Fast R-CNN: Steps

Process the whole image up to conv5.

Compute possible locations for objects.

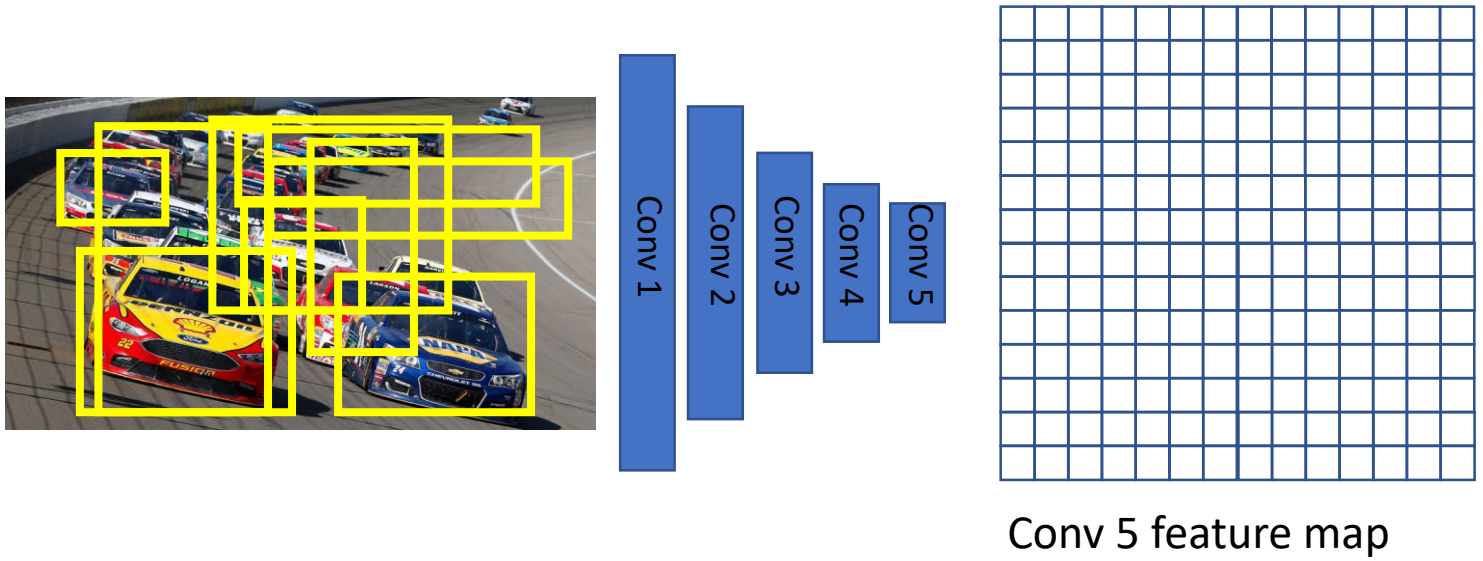


Fast R-CNN: Steps

Process the whole image up to conv5.

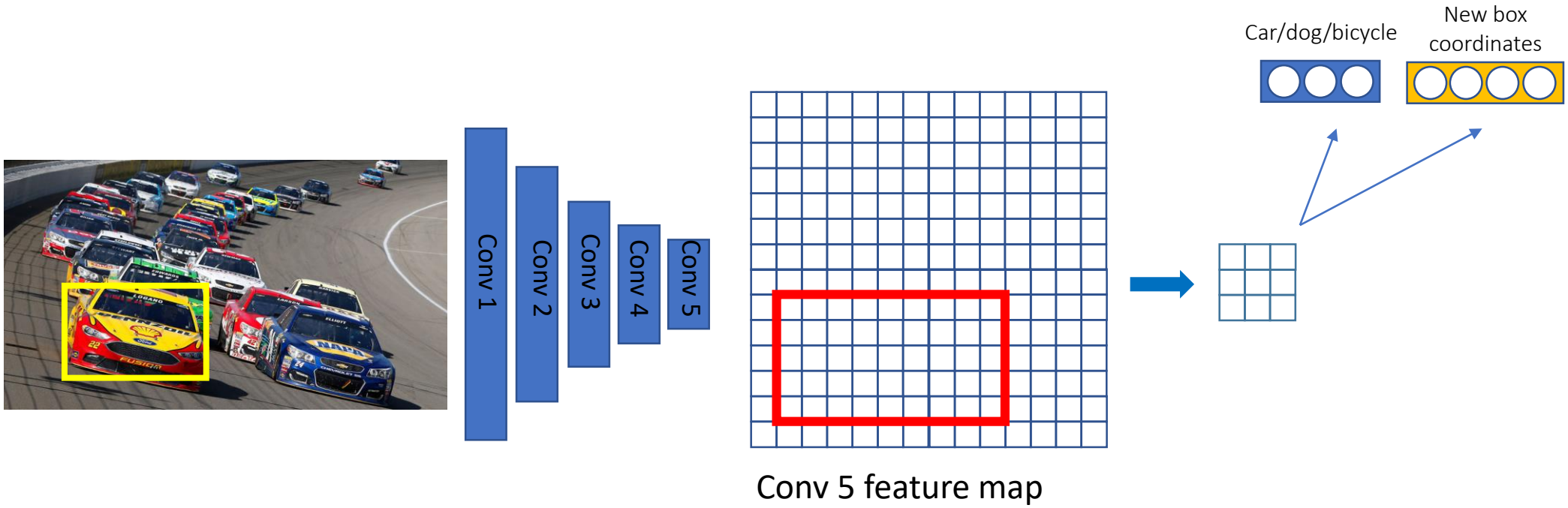
Compute possible locations for objects.

- some correct, most wrong.



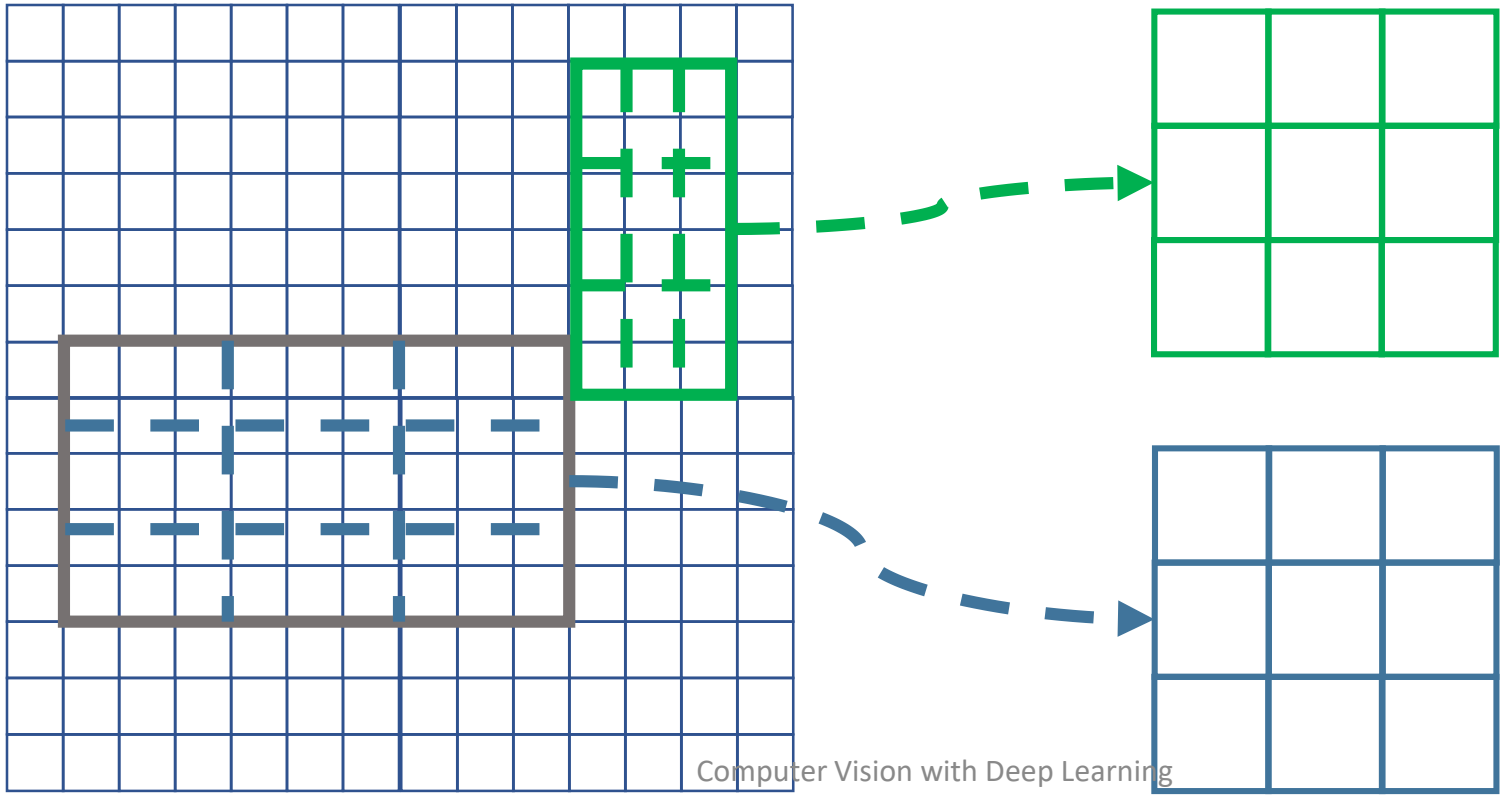
Fast R-CNN: Steps

Given single location: ROI pooling module extracts fixed length feature.



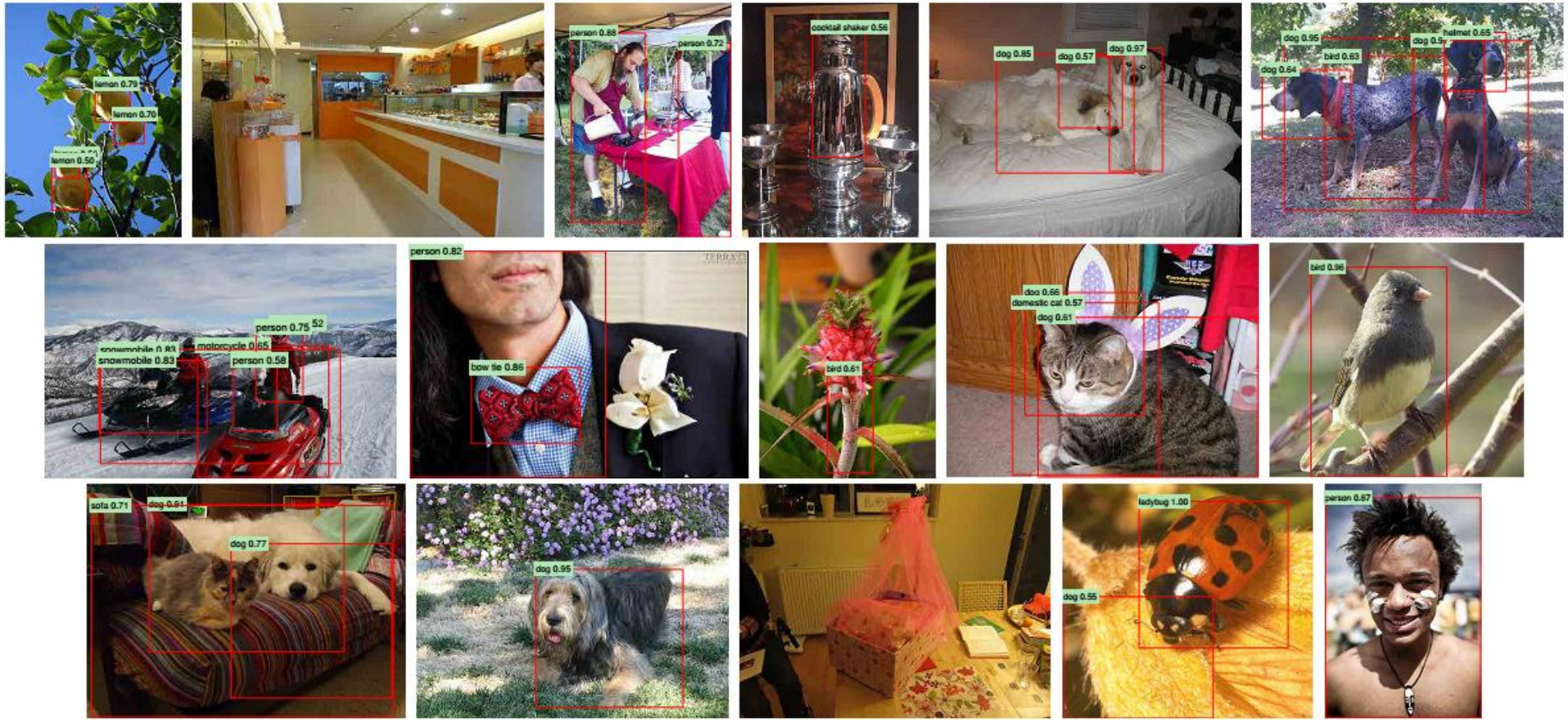
ROI pooling

Divide cell area into fixed sized bins.



Always 3x3 no matter the size of candidate location

Some results



Fast R-CNN

Reuse convolutions for different candidate boxes

- Compute feature maps only once

Region-of-Interest pooling

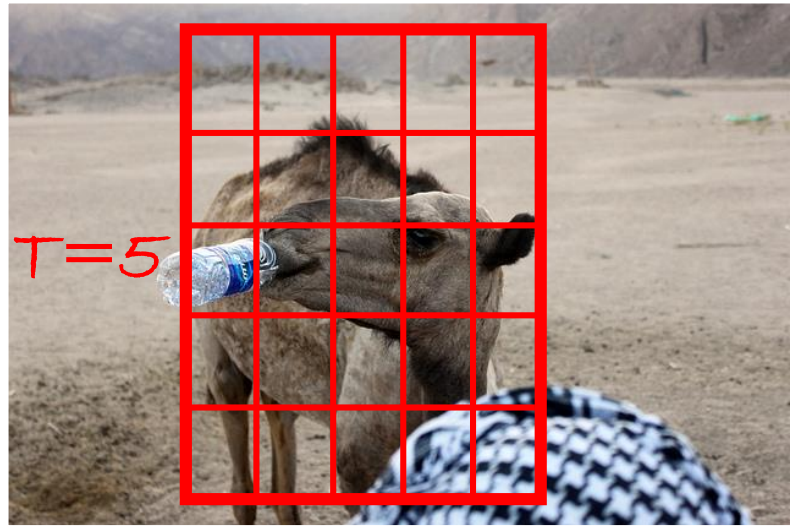
- From variable box area to fixed areas and fixed representations.

End-to-end training!

(Very) Accurate object detection

(Very) Faster

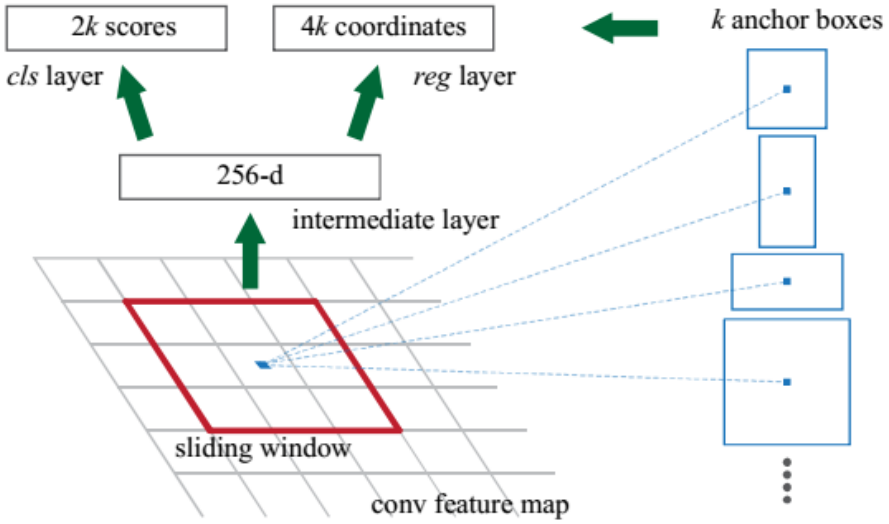
External box proposals needed



[Ren et al. NeurIPS, 2015]

Deep Learning for Detection: Faster R-CNN

Rather than using pre-defined object proposals, use sliding window over feature maps.



Region Proposal Network

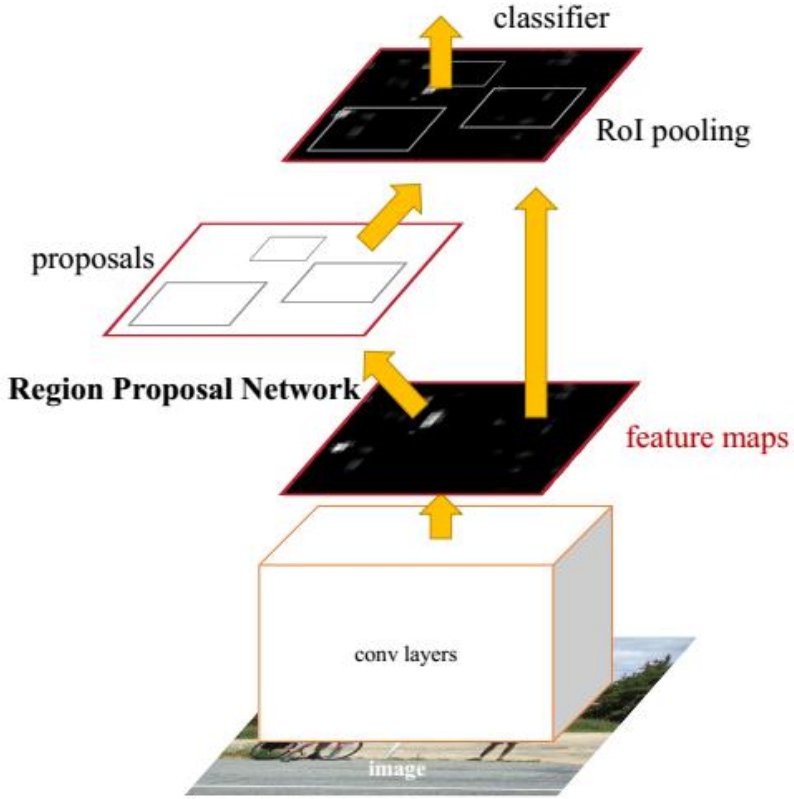
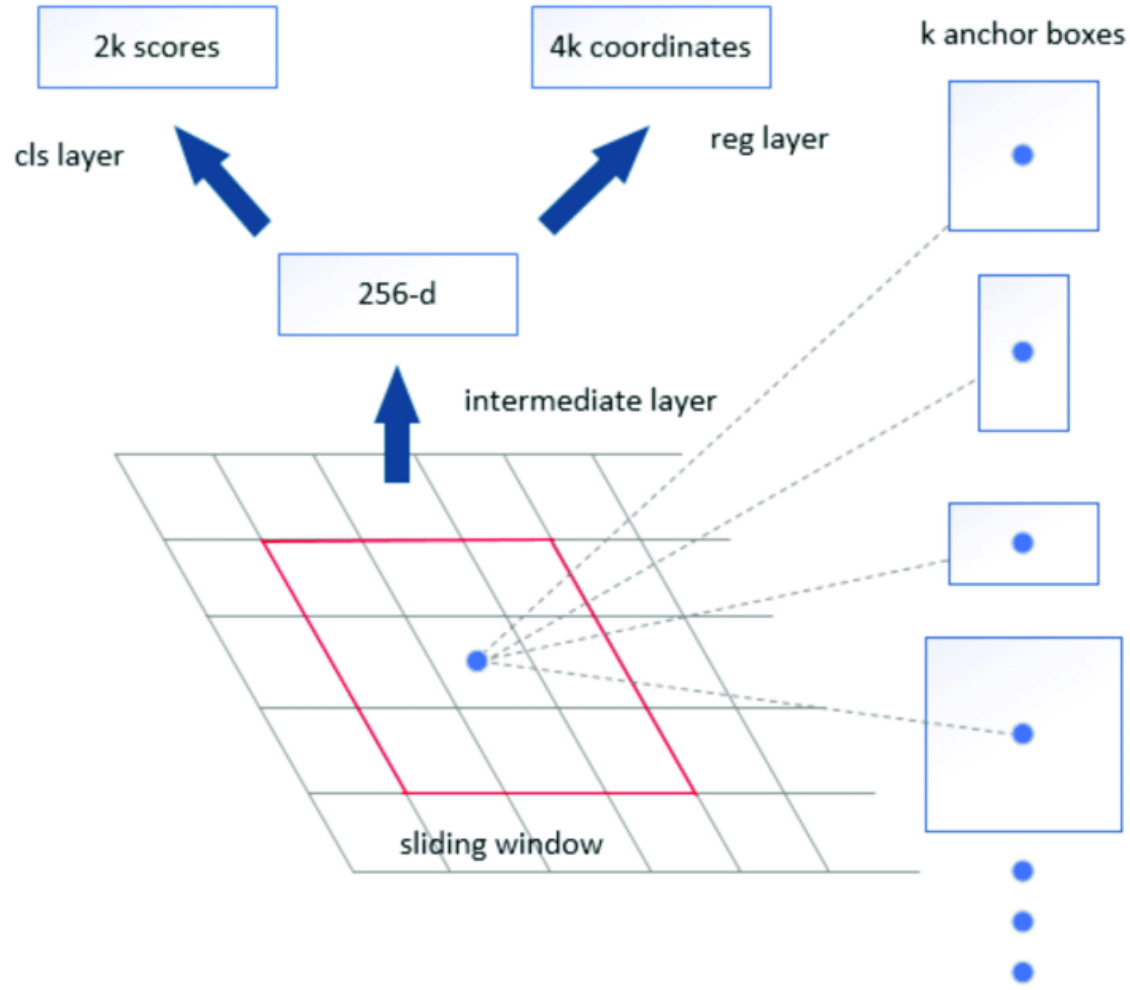


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Region Proposal Network



Deep learning for per-pixel outputs

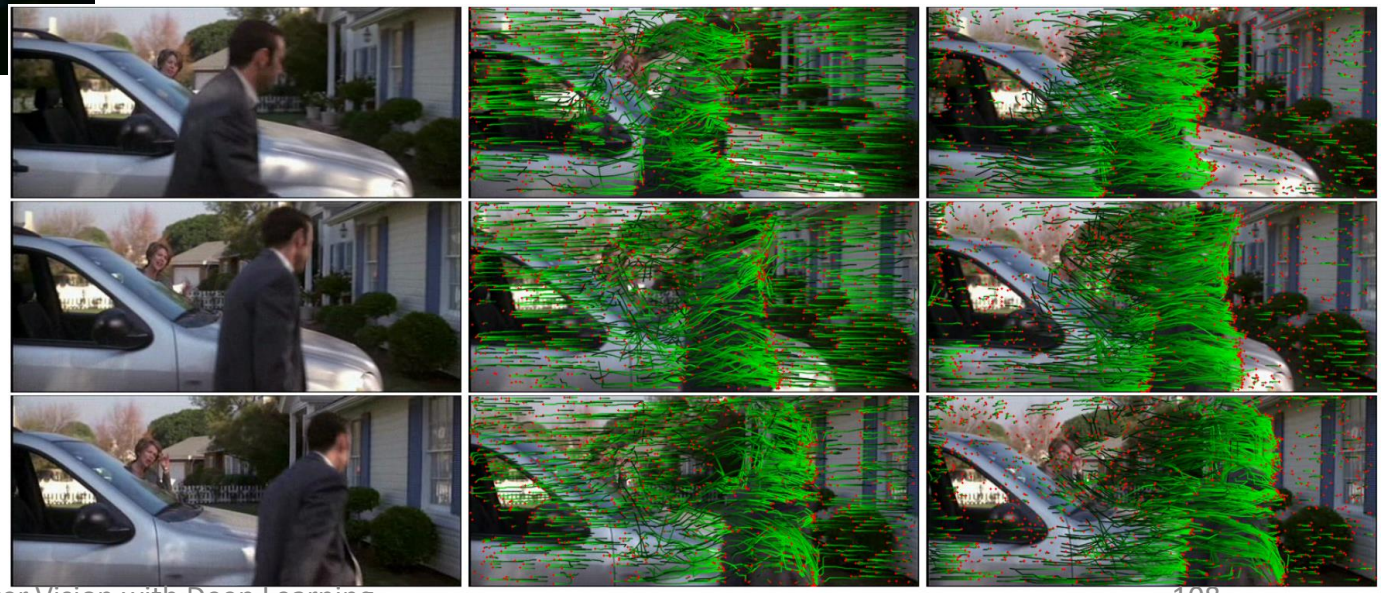
Per-pixel predictions

Many tasks require detailed prediction all the way at the pixel level.

Can you name a number of such tasks?

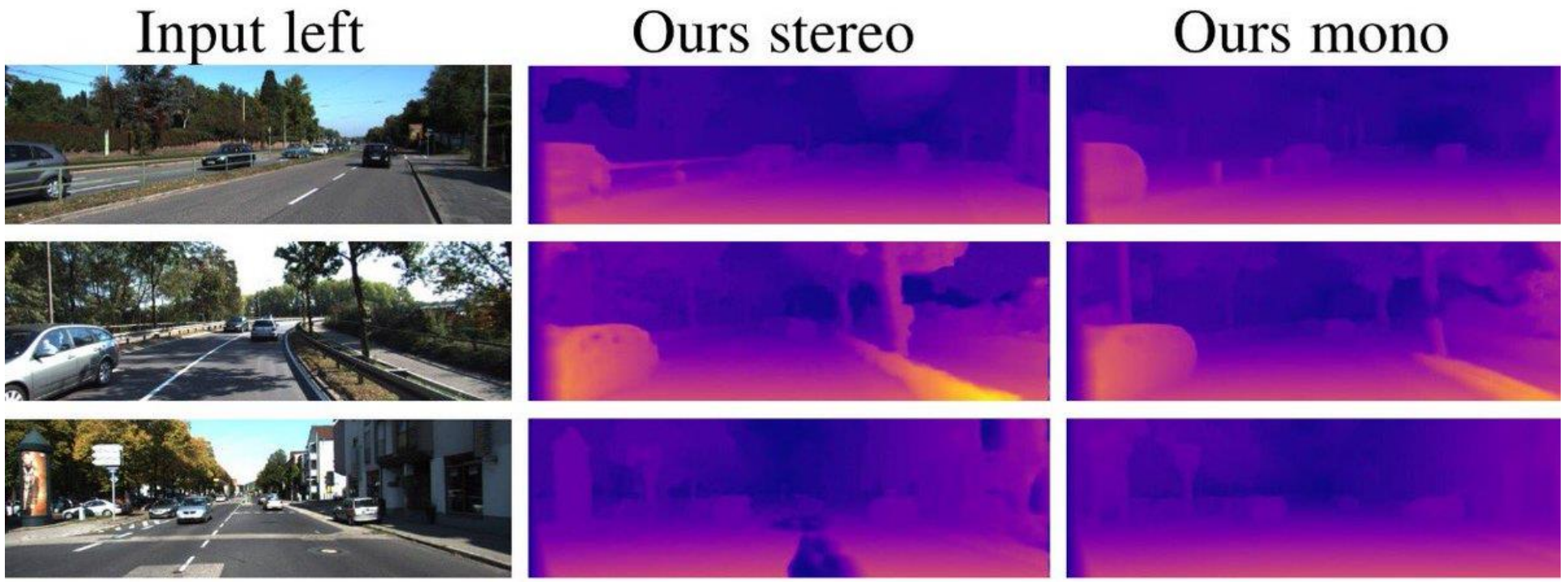
Object segmentation

Optical flow & motion estimation



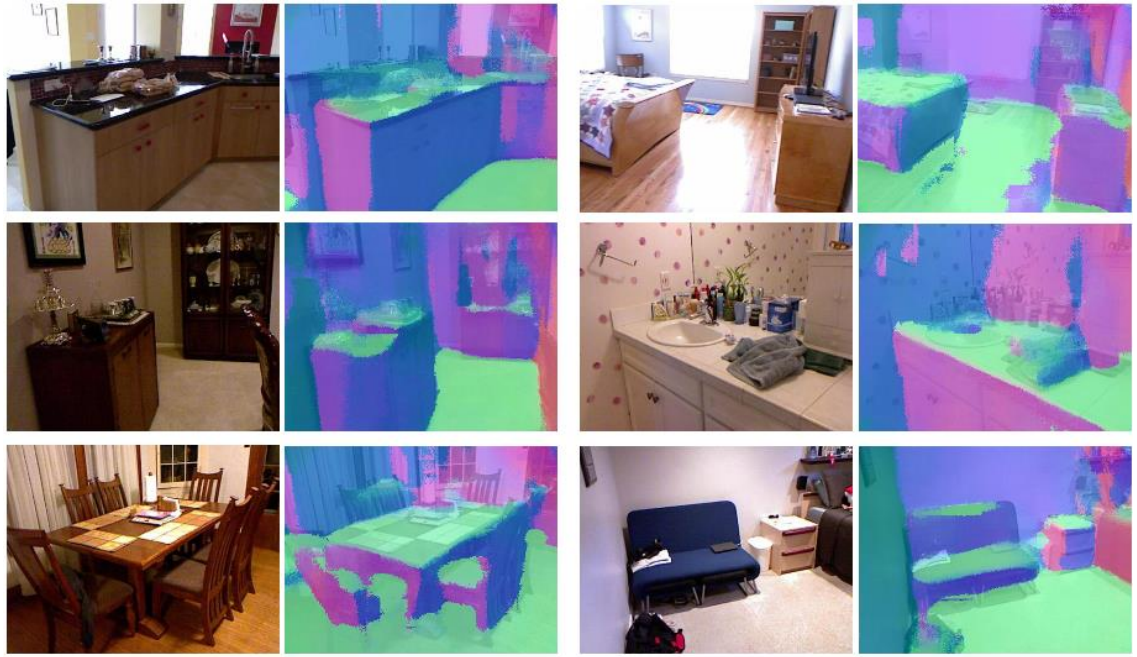
(a) Consecutive frames (b) Trajectories from Optical Flow (c) ω-trajectories

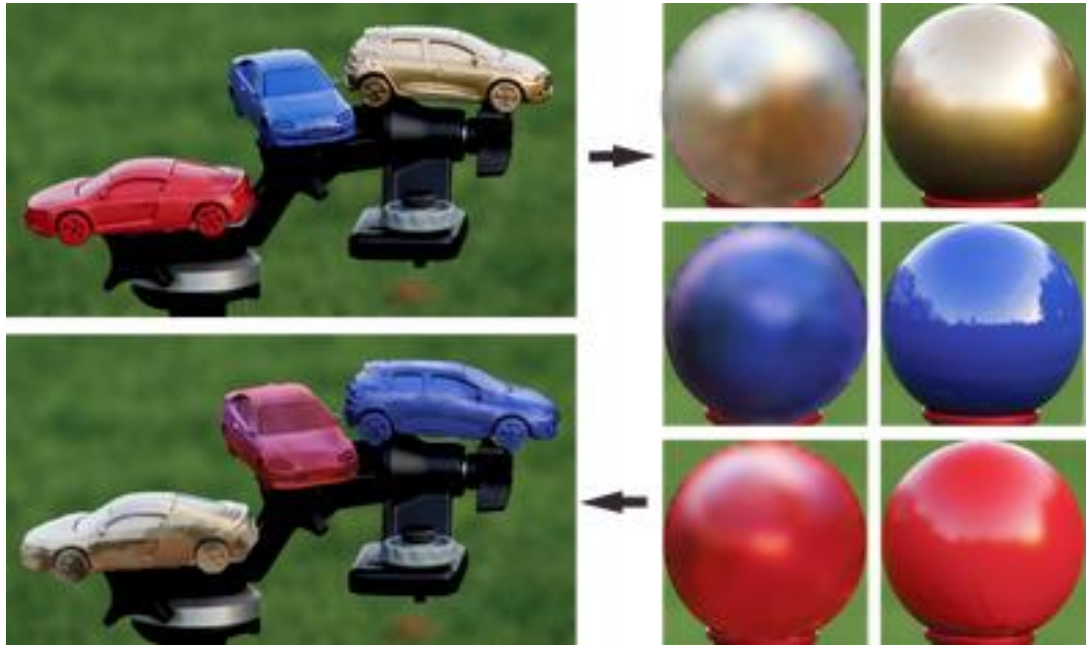
Depth estimation



Godard et al., Unsupervised Monocular Depth Estimation with Left-Right Consistency, 2016

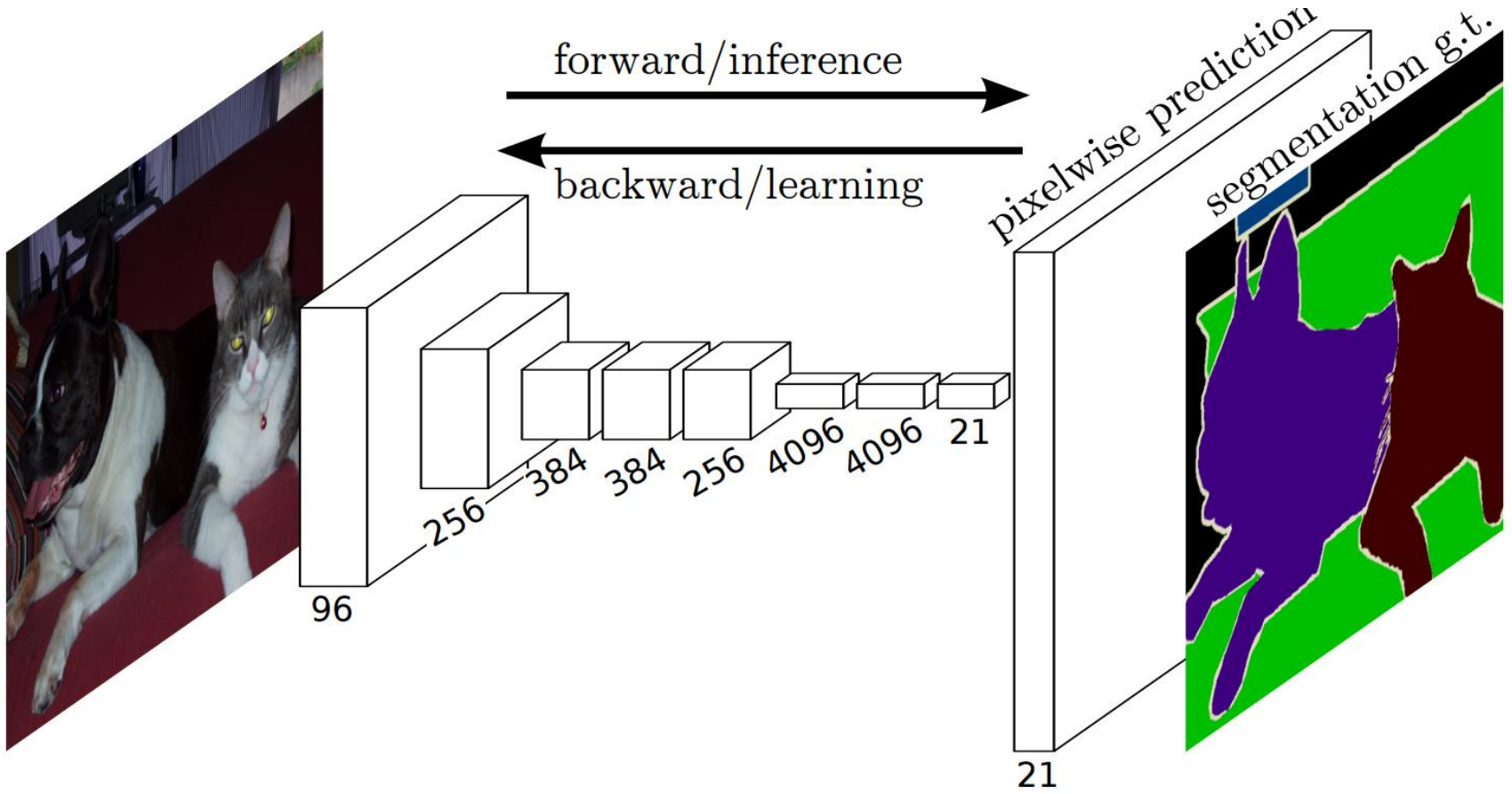
Normal and reflectance estimation



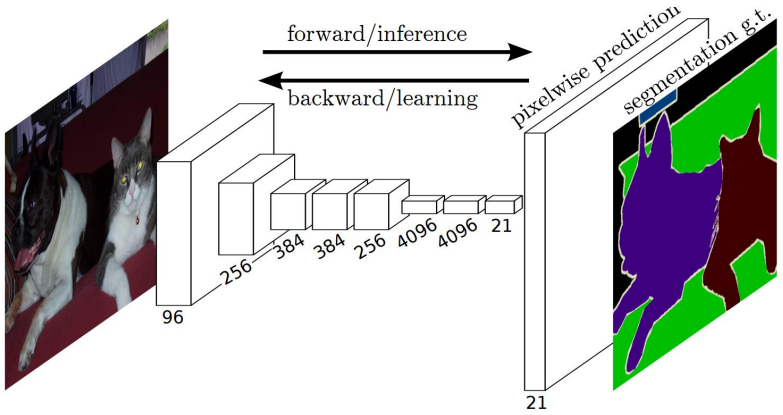


[Long et al. CVPR, 2015]

Fully-convolutional networks



Fully-convolutional networks

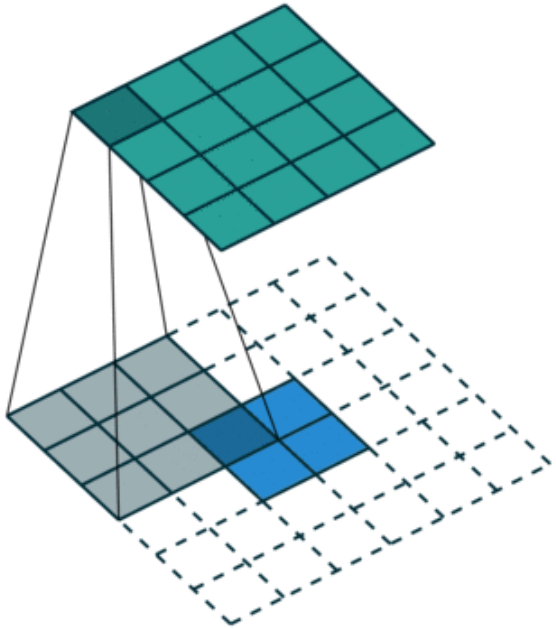


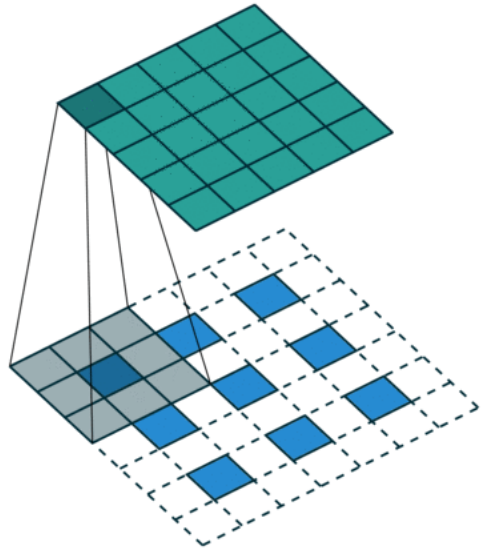
Two main differences:

- Deconvolution to obtain original image size.
- Loss at pixel-level, instead of a global loss.

Per-pixel loss means, compute e.g. squared loss per pixel and backpropagate from all output pixels back to the network.

Deconvolution





Object segmentation: Mask R-CNN

State-of-the-art in semantic segmentation.

Heavily relies on Fast R-CNN.

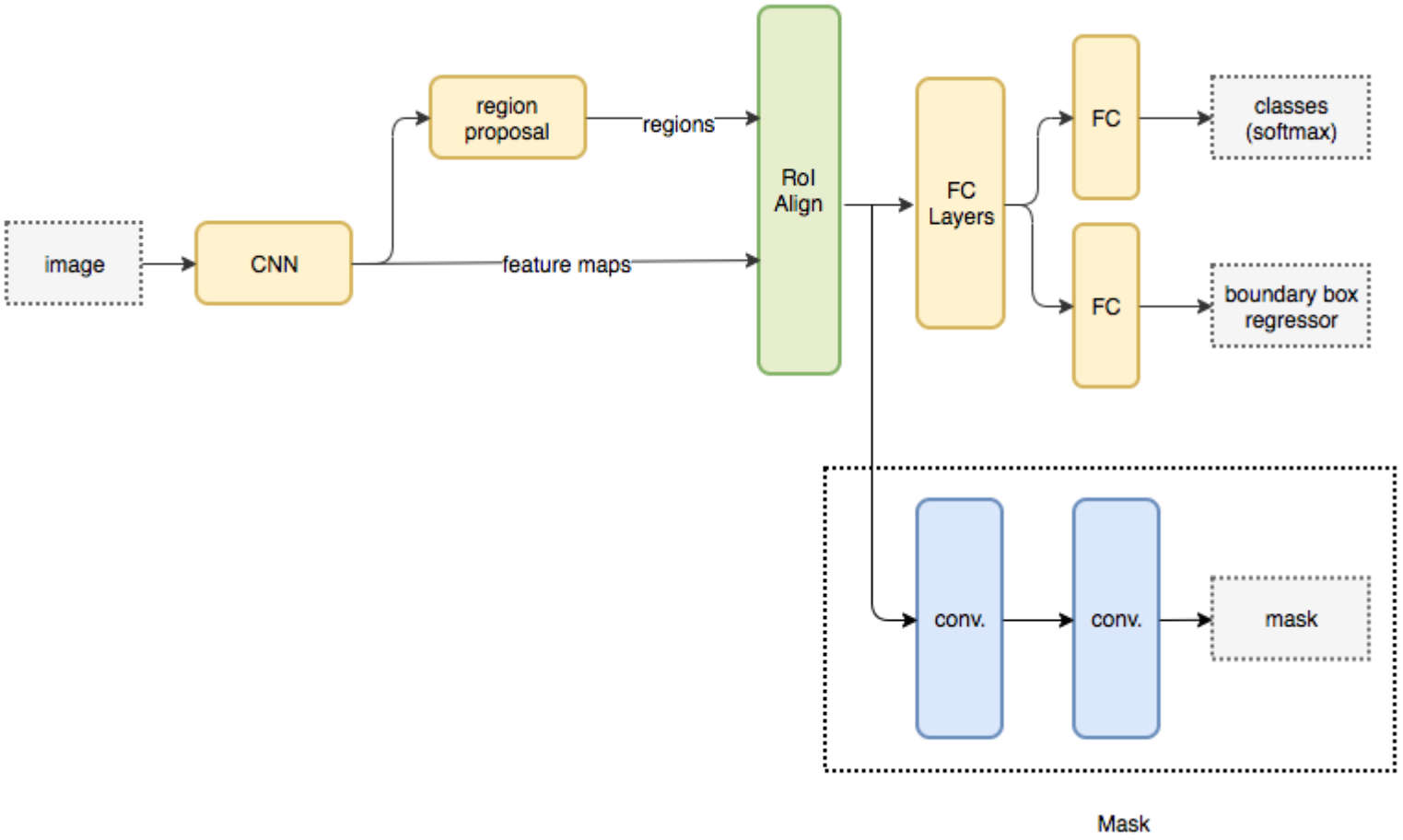
Can work with different architectures, also ResNet.

Runs at 195ms per image on an Nvidia Tesla M40 GPU.

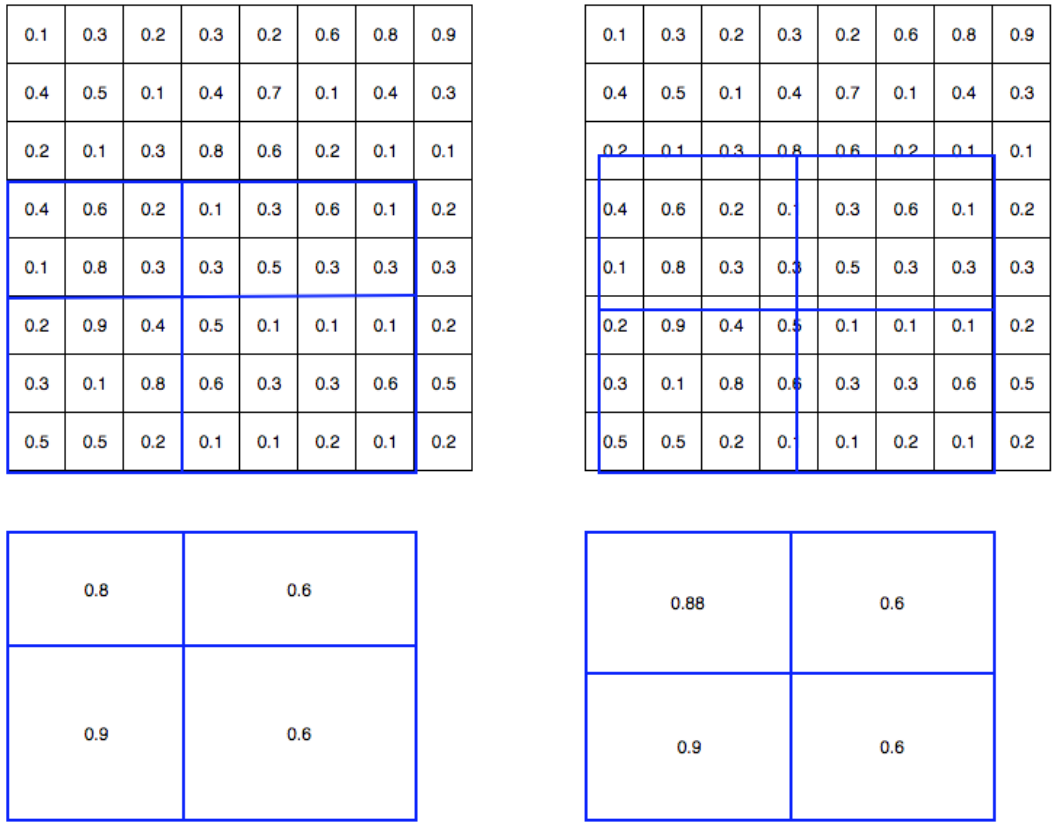
Can also be used for Human Pose Estimation.

[He et al. ICCV, 2017]

Mask R-CNN: Faster R-CNN + 2 layers

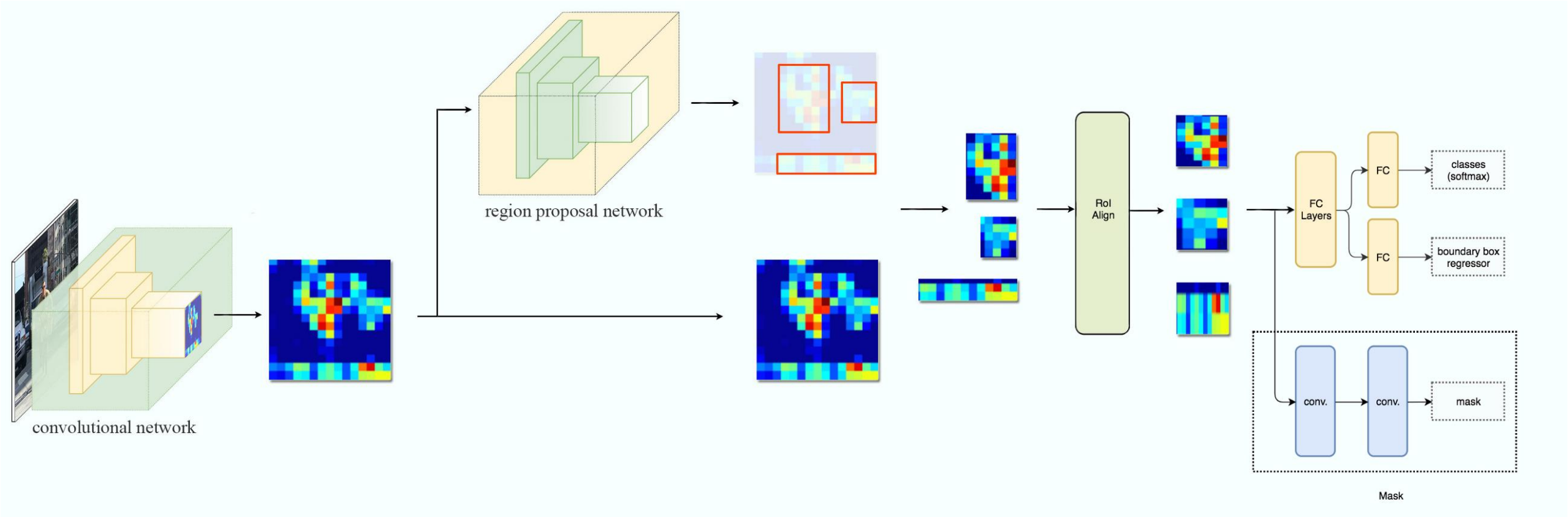


Mask R-CNN: ROI Align

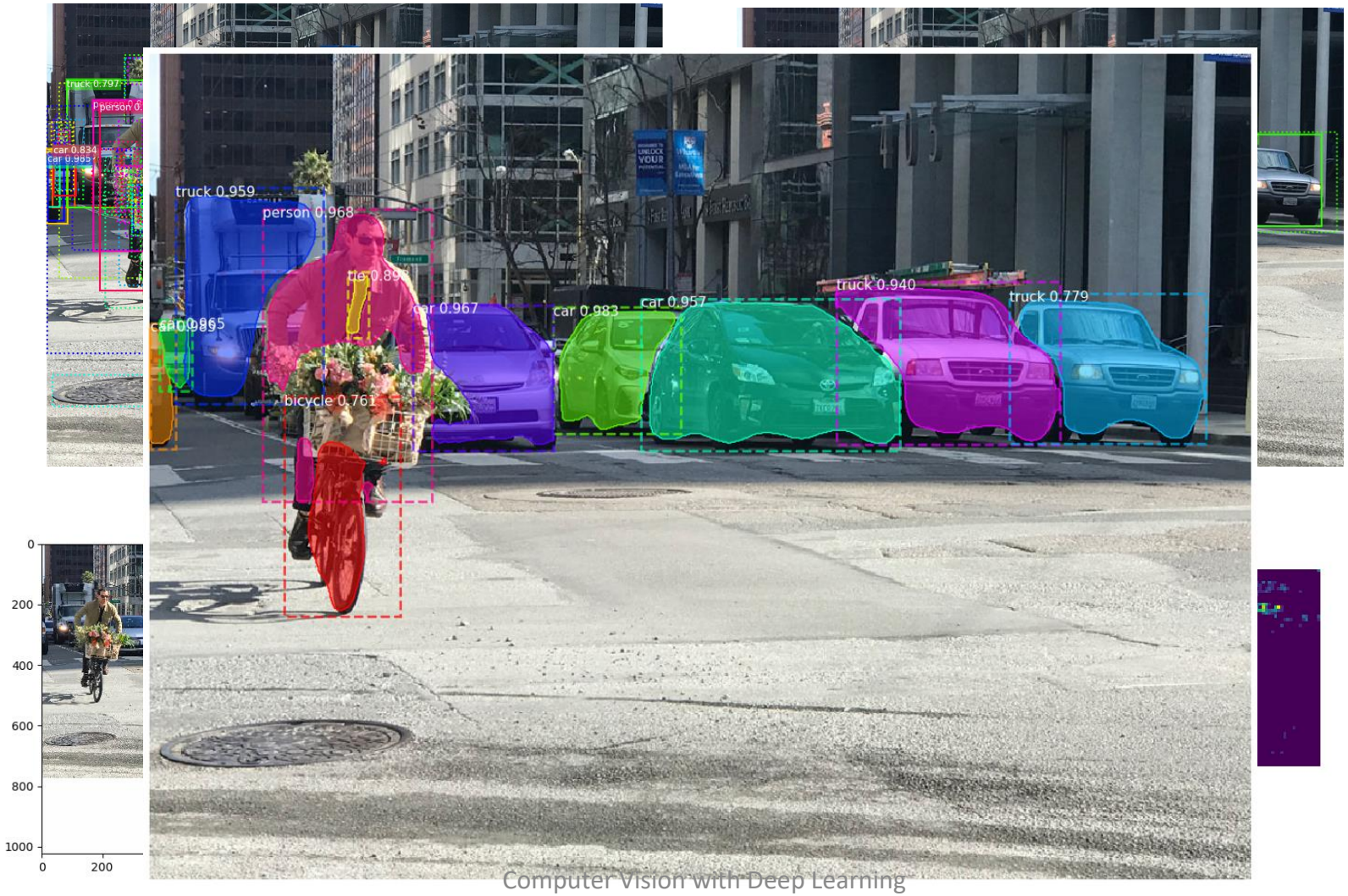


	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

Mask R-CNN


region proposal network
convolutional network
RoI Align
FC Layers
FC
classes (softmax)
FC
boundary box regressor
conv.
conv.
mask
Mask

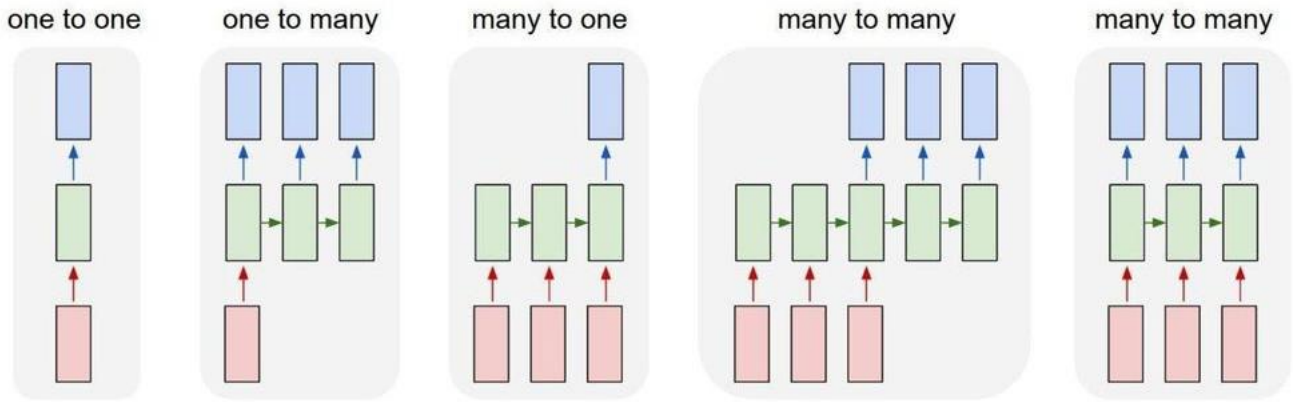
Mask R-CNN

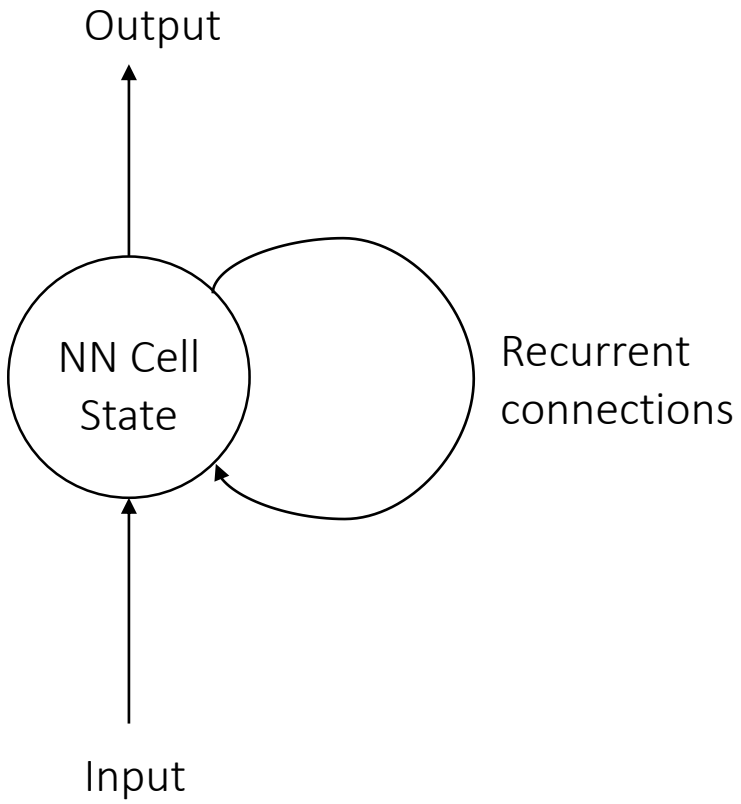

truck 0.959
person 0.968
car 0.967
car 0.983
car 0.957
truck 0.940
truck 0.779
bicycle 0.761
0
200
400
600
800
1000
0
200

Deep learning for sequences

Sequence prediction in vision

Many non-standard tasks are enabled with the use of RNNs.



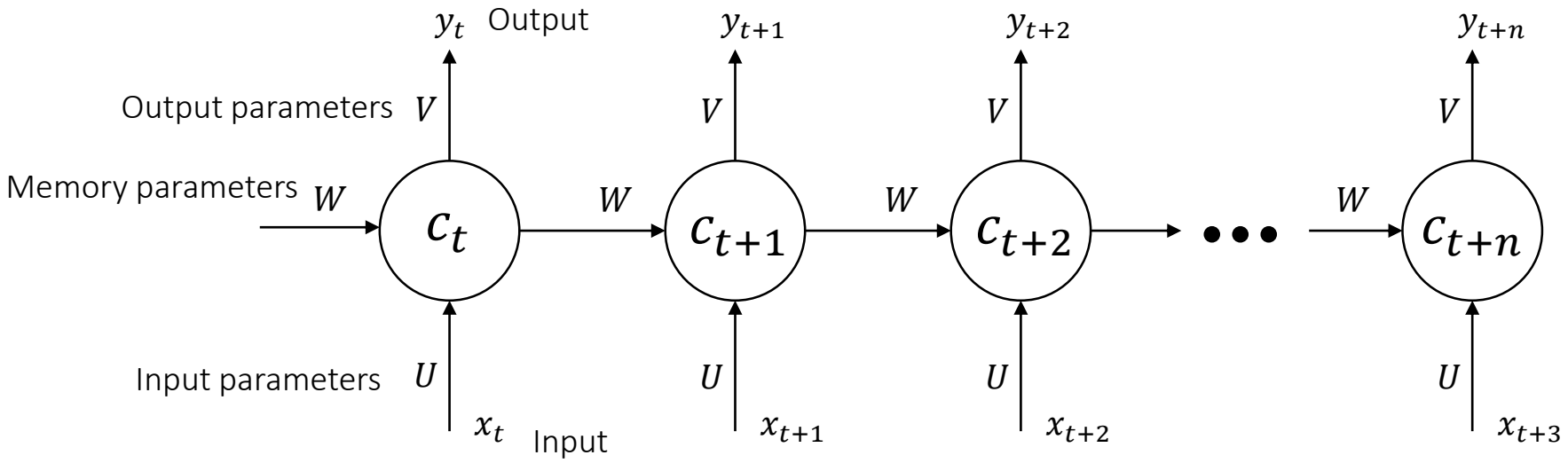


Layout of Recurrent Networks

Sequence inputs: we model them with parameters U

Sequence outputs: we model them with parameters V

Memory I/O: we model it with parameters W

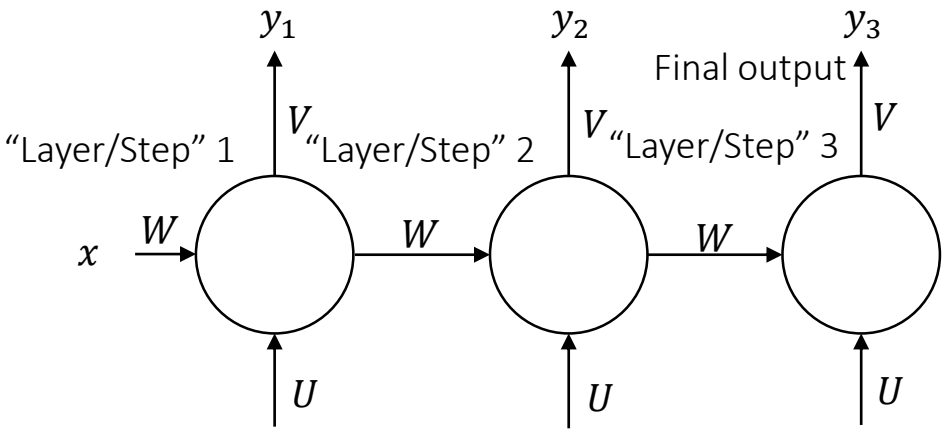


RNNs vs MLPs

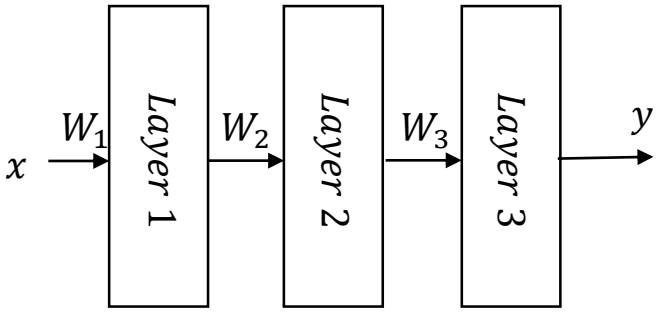
Is there a big difference?

Outputs at every step (MLP outputs in every layer also possible).

Instead of layer-specific parameters, we have layer-shared parameters.



3-gram Unrolled Recurrent Network



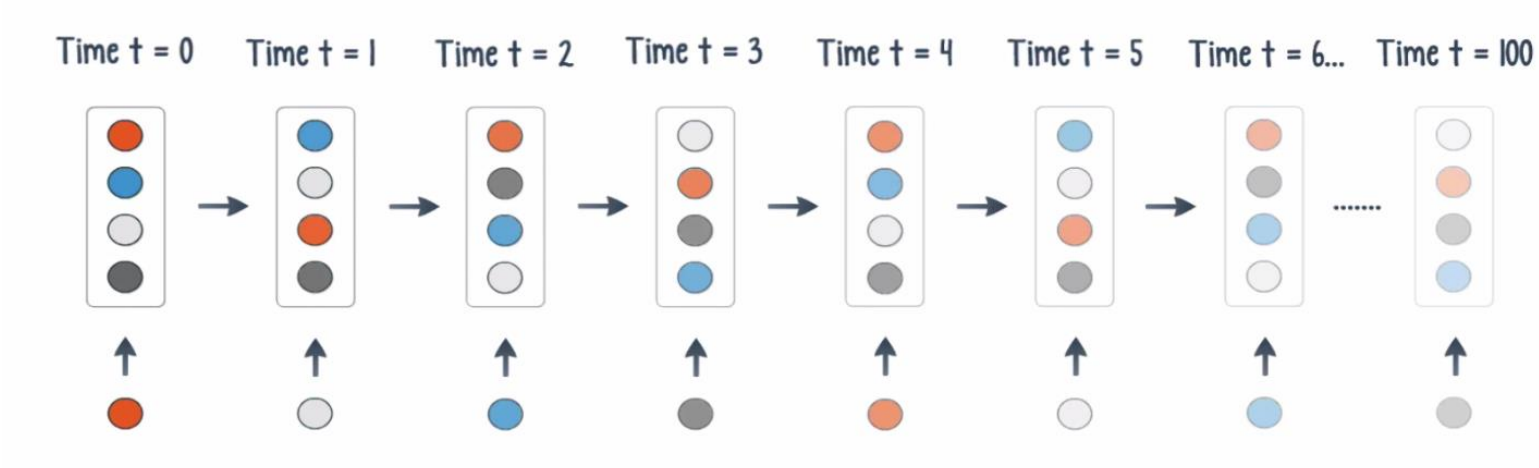
3-layer Neural Network

The problem with RNNs

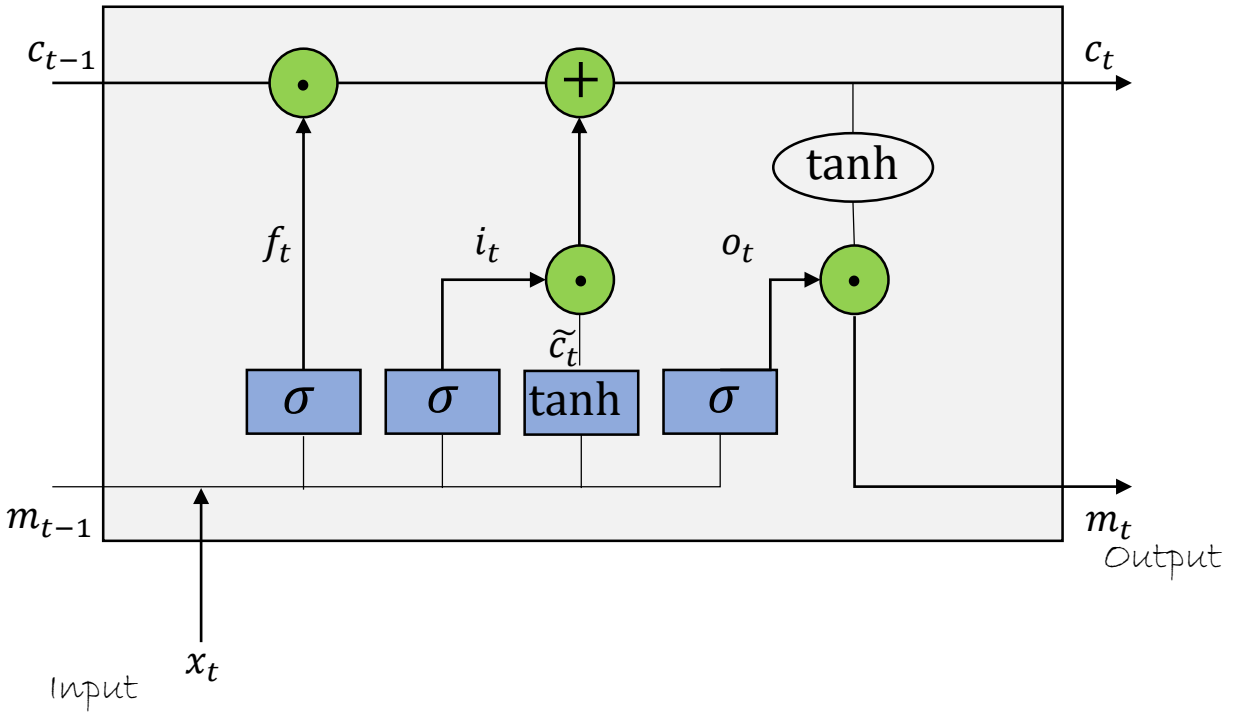
As RNNs get deeper, they suffer from exploding/vanishing gradients.

Especially vanishing gradients are a problem.

- RNNs tend to focus on short-term transitions, ignoring long-term ones.



Long-Short Term Networks



LSTMs

LSTMs tackle multiple issues in RNNs.

- At each RNN step the hidden state is overwritten.
- Not all inputs are important enough to be written down.
- Not all outputs are important enough to be written.
- Not all information is important enough to be remembered.

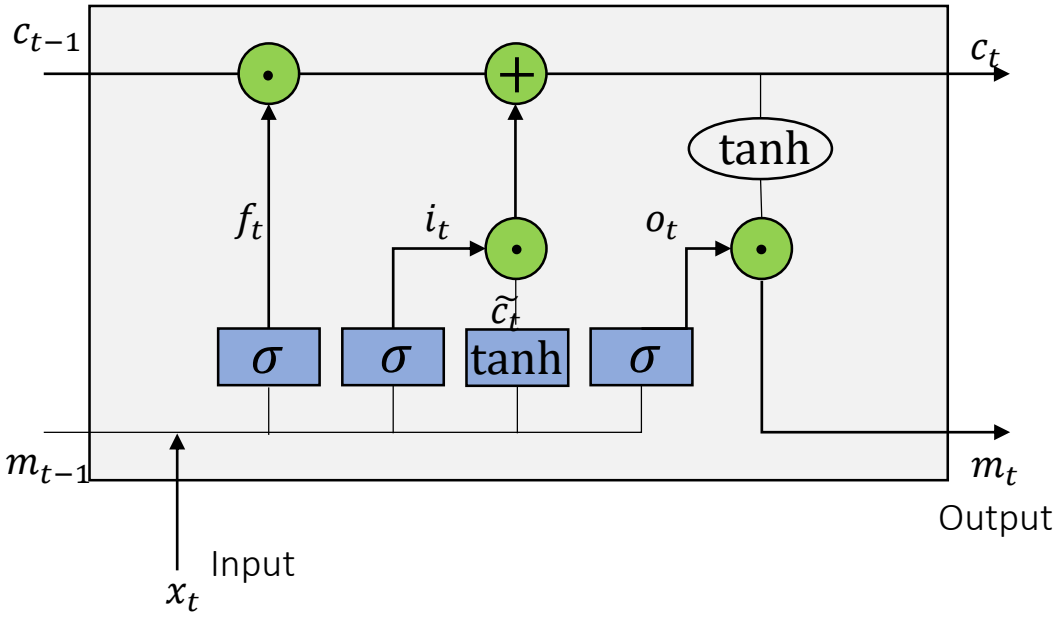
LSTMs vs RNNs

RNNs

$$c_t = W \cdot \tanh(c_{t-1}) + U \cdot x_t + b$$

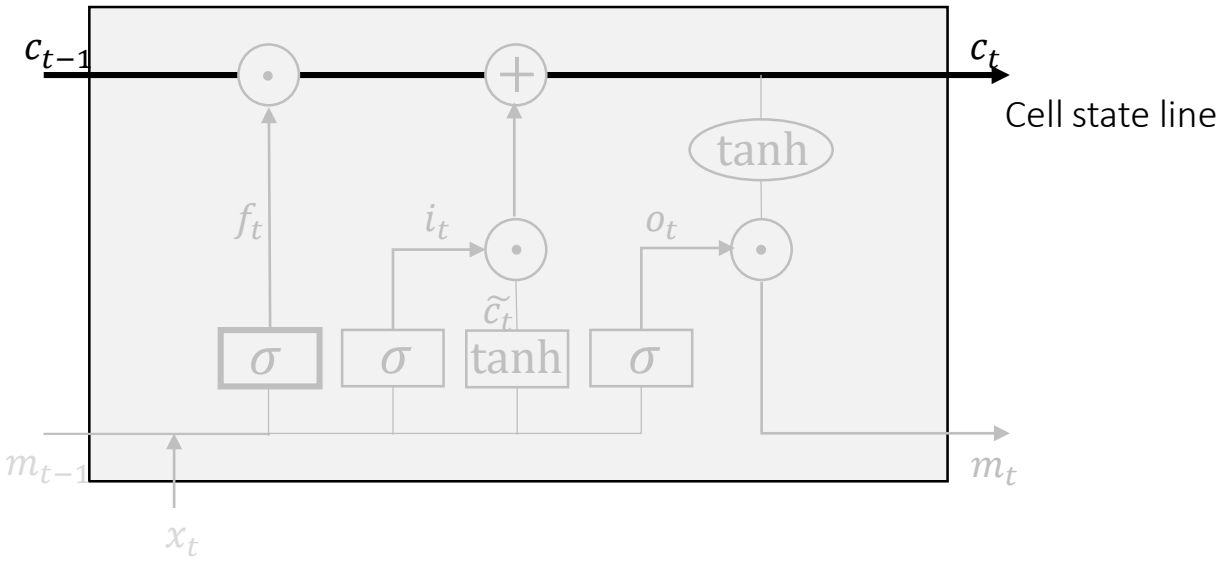
LSTMs

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\tilde{c}_t = \tanh(x_t U^{(g)} + m_{t-1} W^{(g)})$$
$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$
$$m_t = \tanh(c_t) \odot o$$



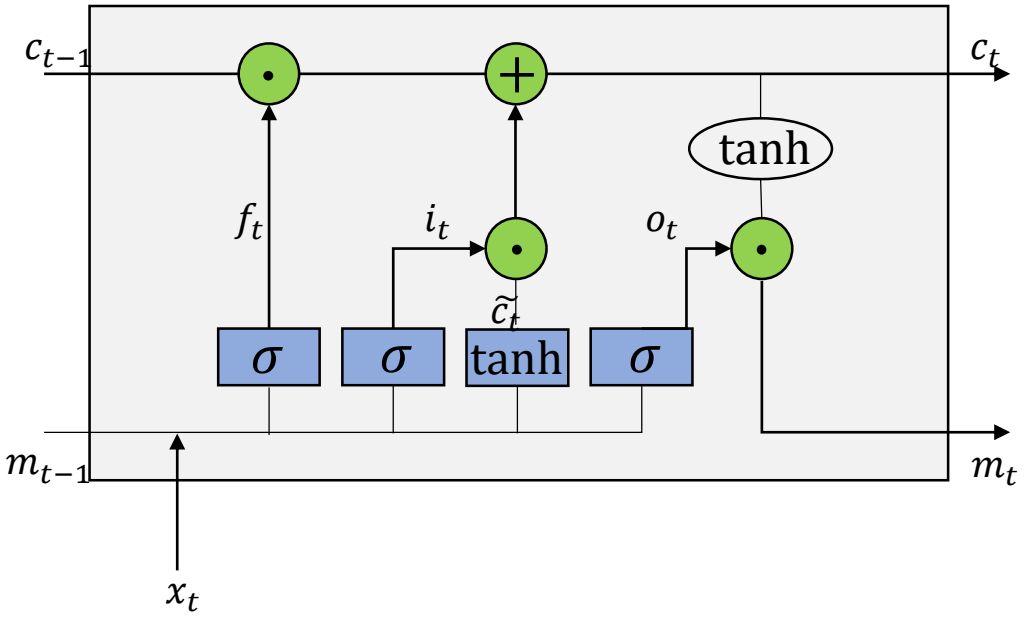
LSTM: step-by-step

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\widetilde{c_t} = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$
$$c_t = c_{t-1} \odot f + \widetilde{c_t} \odot i$$
$$m_t = \tanh(c_t) \odot o$$



LSTM: step-by-step

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\tilde{c}_t = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$
$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$
$$m_t = \tanh(c_t) \odot o$$



$\sigma \in (0, 1)$: control gate – something like a switch

$\tanh \in (-1, 1)$: recurrent nonlinearity

LSTM: step-by-step #1

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

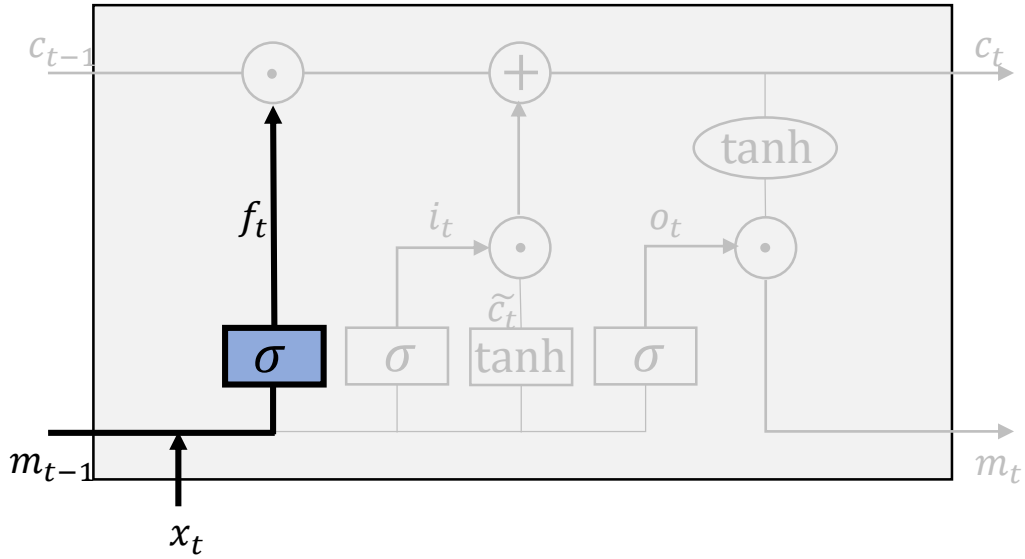
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\widetilde{c_t} = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$

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

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



LSTM: step-by-step #2

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$

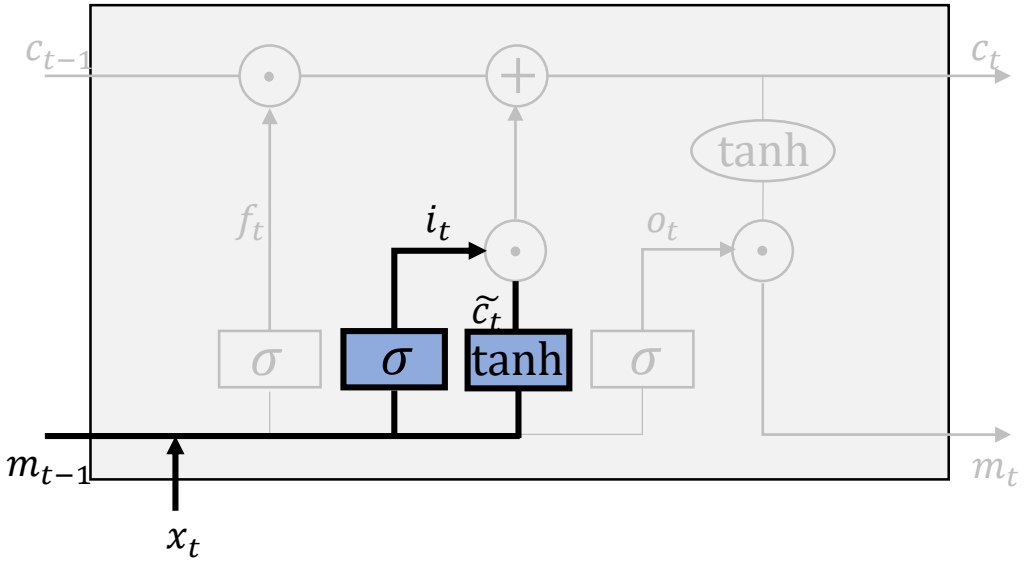
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$

$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$

$$\tilde{c}_t = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$

$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$

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

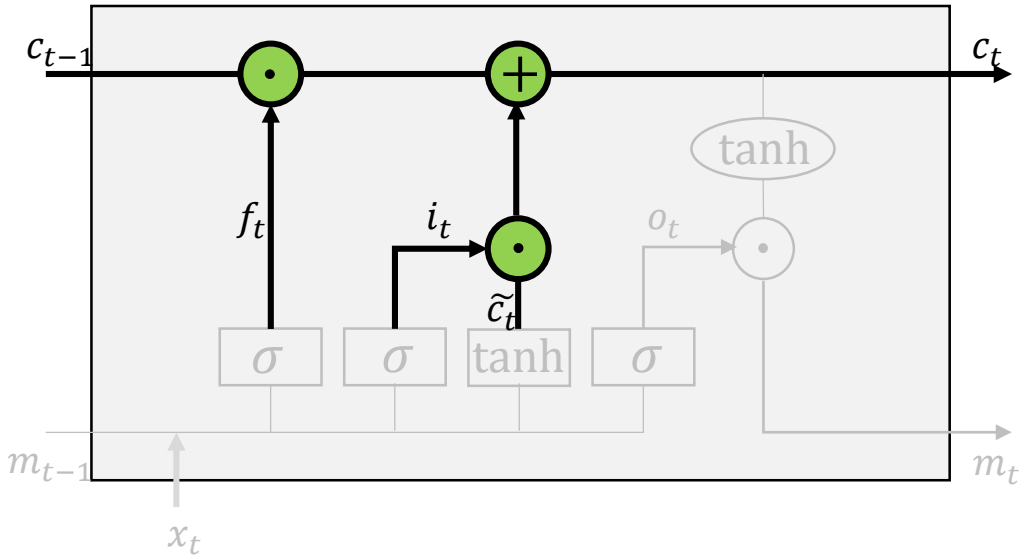


Decide what new information is relevant from the new input and should be added to the new memory

- Modulate the input i_t
- Generate candidate memories $\tilde{c}_t$

LSTM: step-by-step #3

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\tilde{c}_t = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$
$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$
$$m_t = \tanh(c_t) \odot o$$

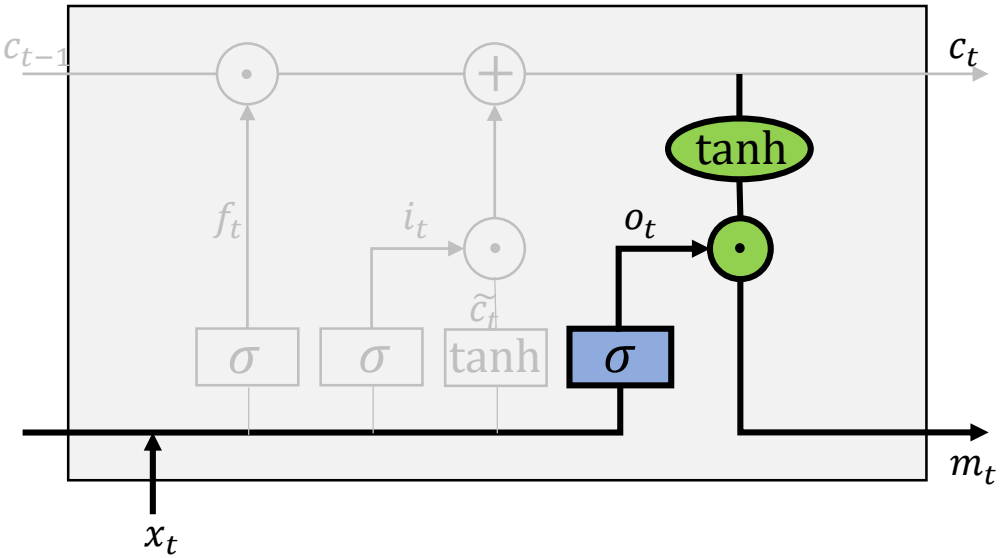


Compute and update the current cell state c_t

- Depends on the previous cell state
- What we decide to forget
- What inputs we allow
- The candidate memories

LSTM: step-by-step #4

$$i = \sigma\left(x_t U^{(i)} + m_{t-1} W^{(i)}\right)$$
$$f = \sigma\left(x_t U^{(f)} + m_{t-1} W^{(f)}\right)$$
$$o = \sigma\left(x_t U^{(o)} + m_{t-1} W^{(o)}\right)$$
$$\tilde{c}_t = \tanh\left(x_t U^{(g)} + m_{t-1} W^{(g)}\right)$$
$$c_t = c_{t-1} \odot f + \tilde{c}_t \odot i$$
$$m_t = \tanh(c_t) \odot o$$

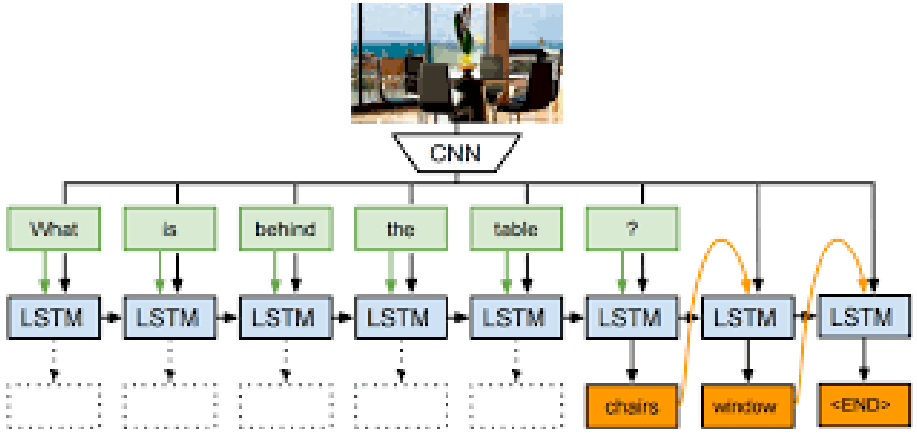


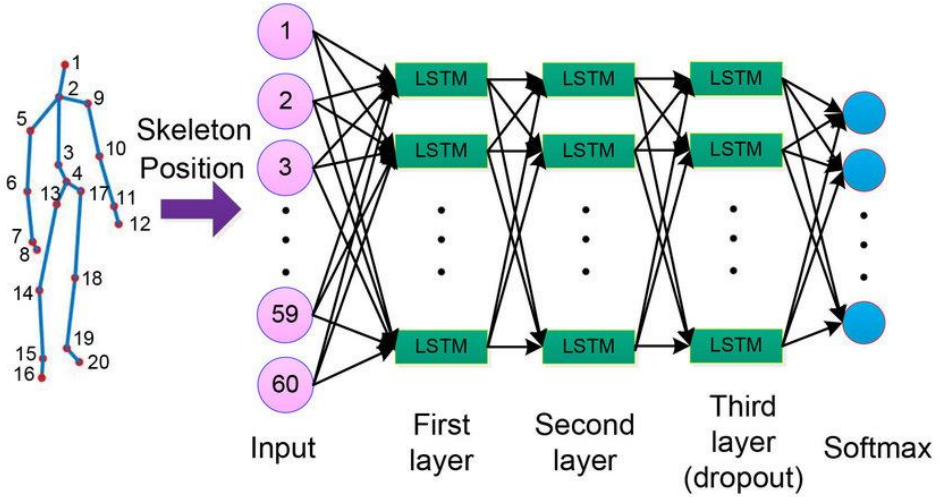
Modulate the output

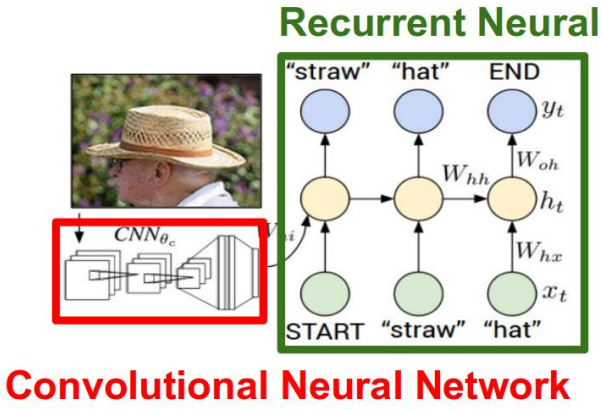
- Does the new cell state relevant? → Sigmoid 1
- If not → Sigmoid 0

Generate the new memory

LSTMs in computer vision



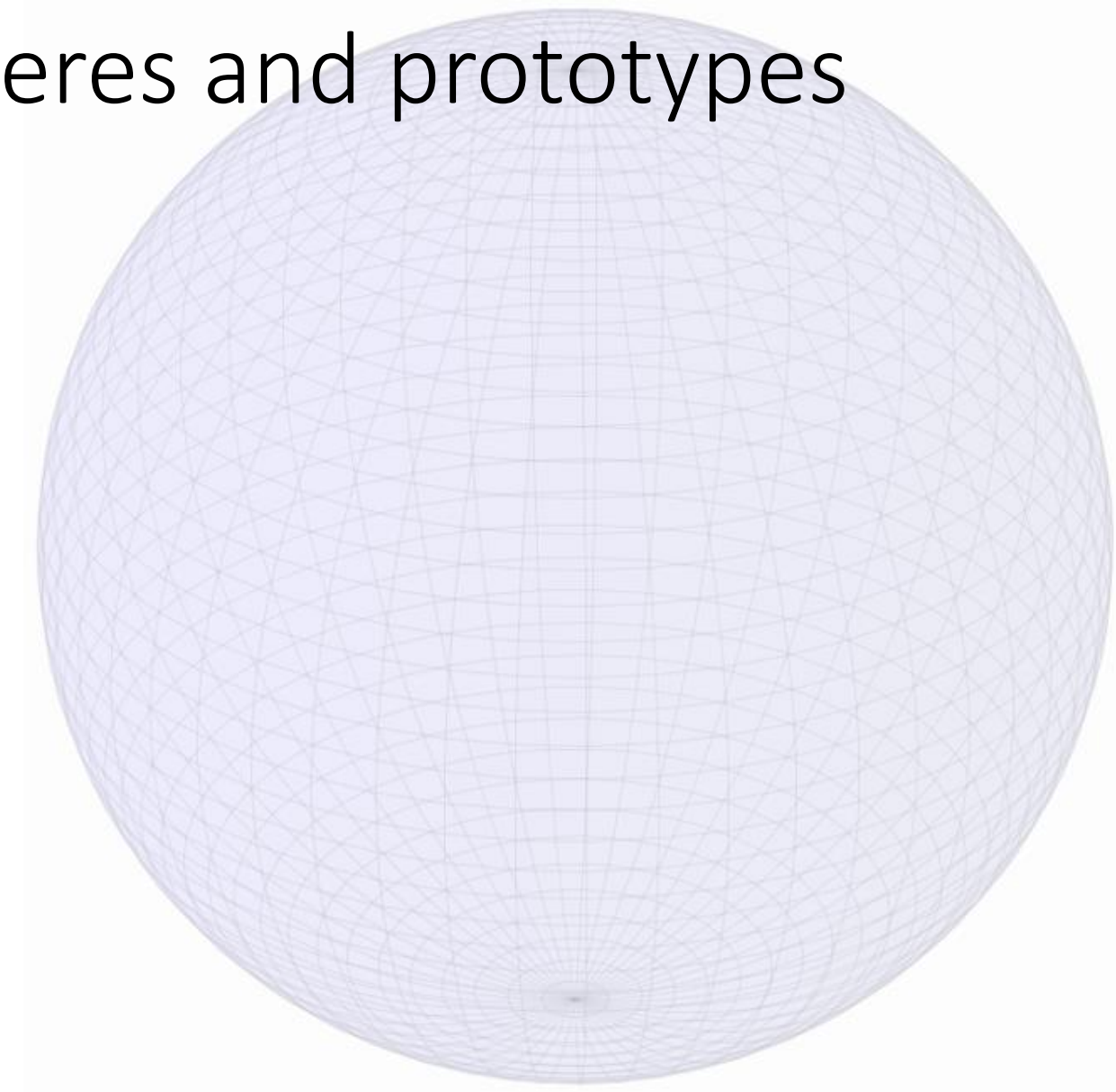




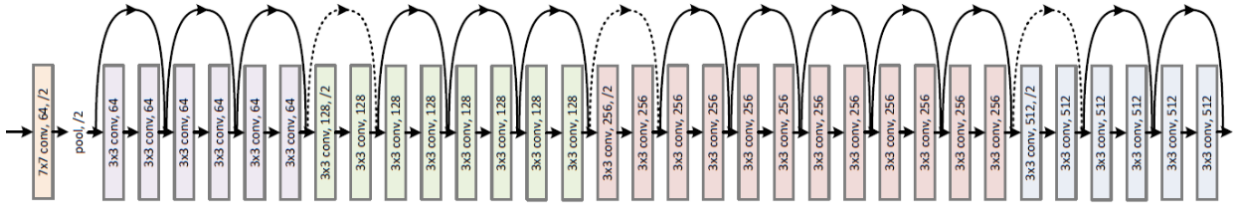
Short break

Deep learning on hyperspheres

Hyperspheres and prototypes



Status-quo in deep classification and regression



[He et al. CVPR 2016]

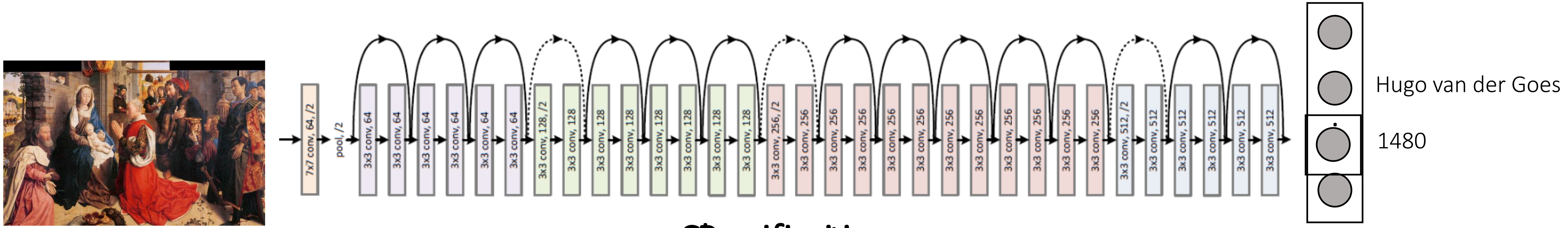
softmax cross-entropy

Hugo van der Goes

1480

squared error

Limits of the status-quo



Classification
Regression

One dimension for each class.
One dimension in total.

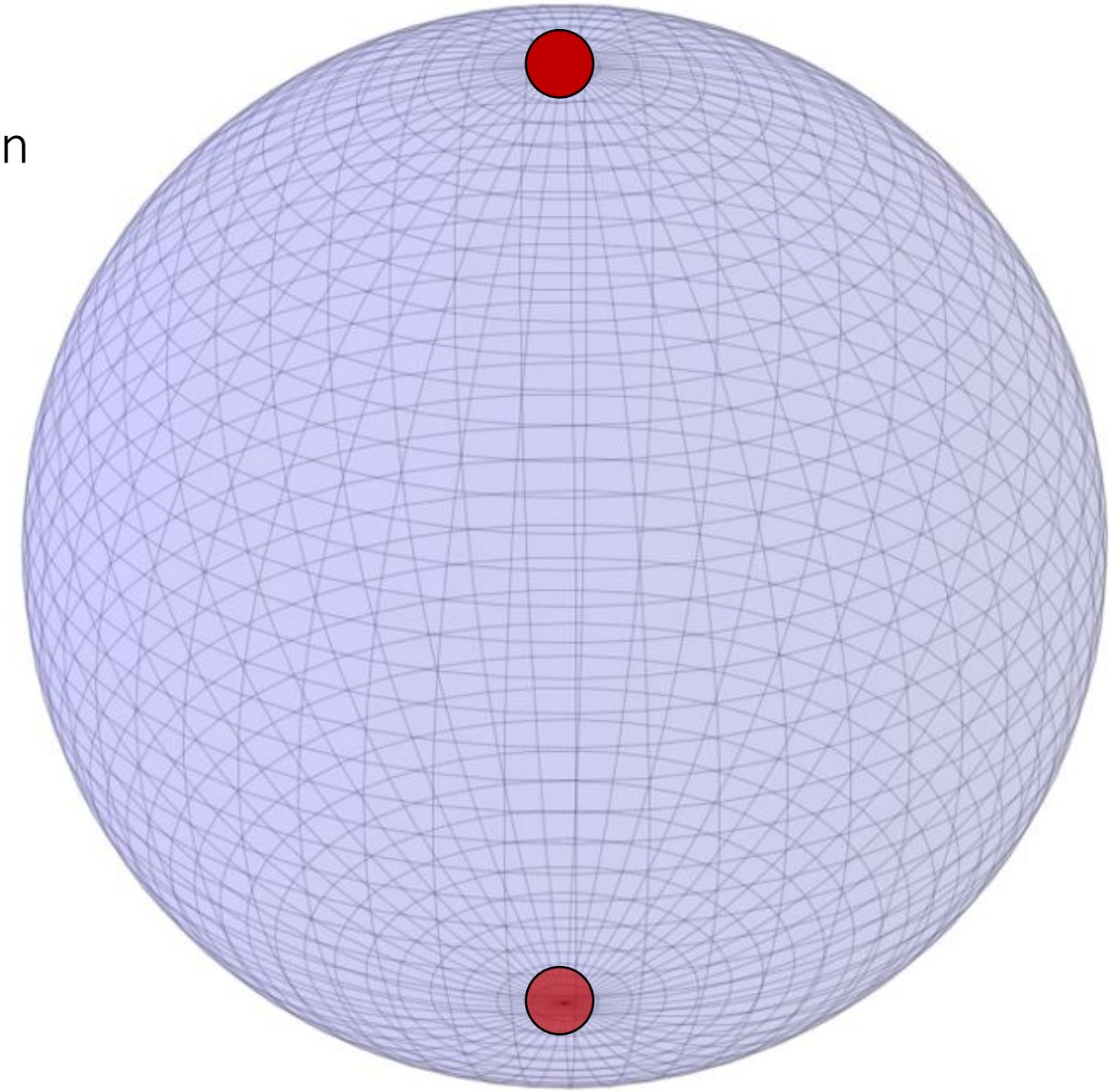
A 1,000,000 dimensions for 1,000,000 classes.
All information compressed into a single number.

Completely different loss and optimization for classification.
Fully parametric, many inductive biases and priors ignored.

Difficult to even learn these tasks at the same time, gradient scales differ order of magnitude.
Knowledge about classes priors to learning gets you nowhere.

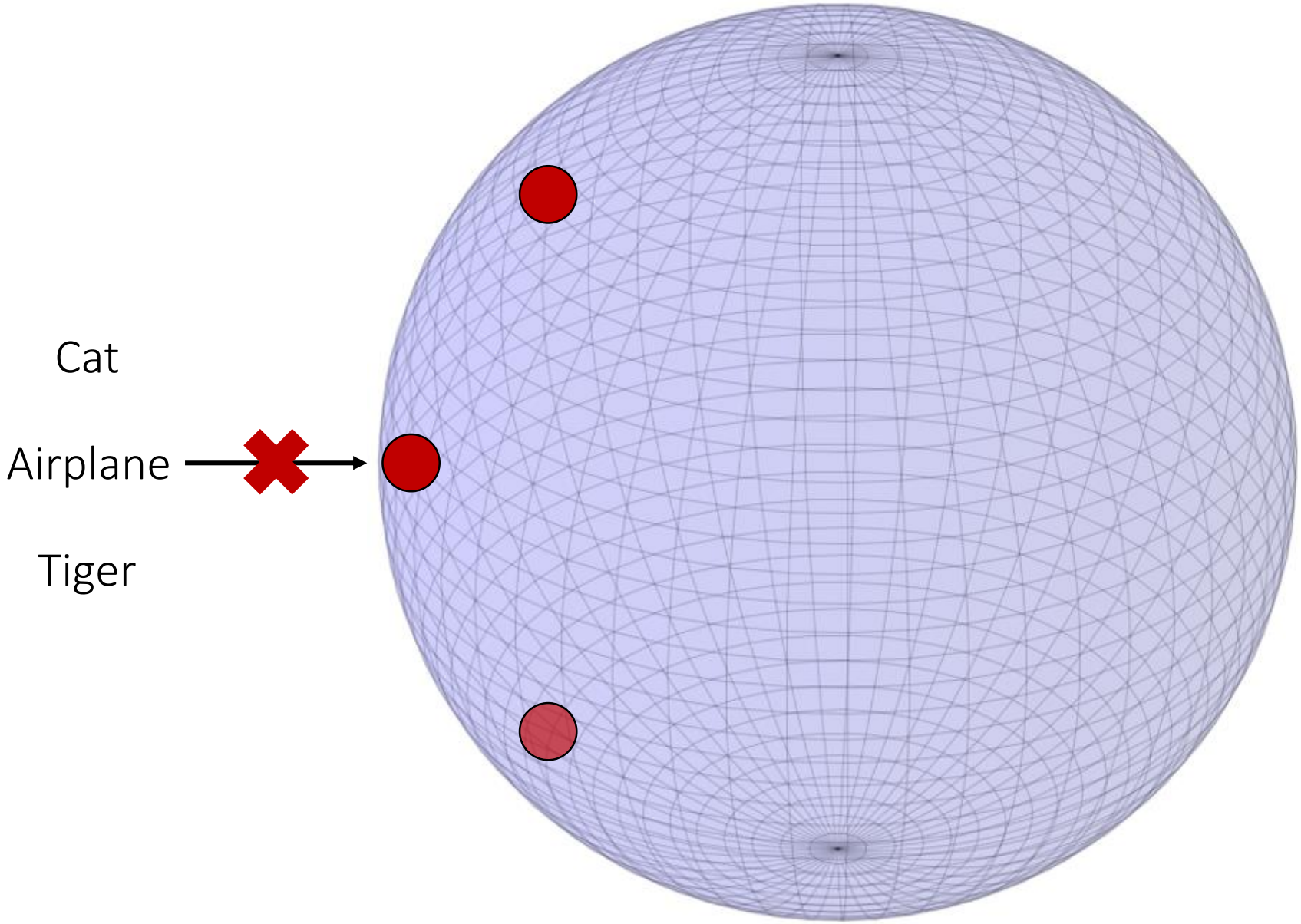
Inductive bias

Large margin separation



Inductive bias

Occam’s Razor



Inductive bias

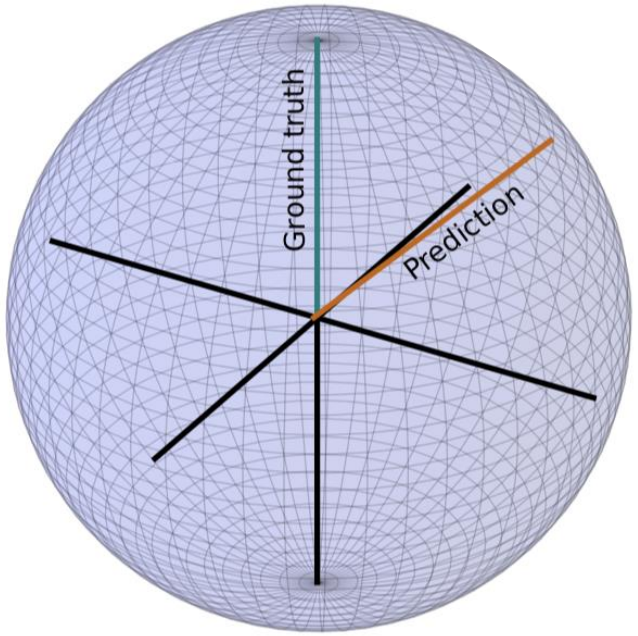
Prior knowledge

Hyperspherical Prototype classification: main idea

Each class is a prototype (point) on the output hypersphere.

Prototypes are placed with maximal separation on hypersphere *prior* to training.

For training, update examples angularly towards prototypes.



Hyperspherical Prototype classification

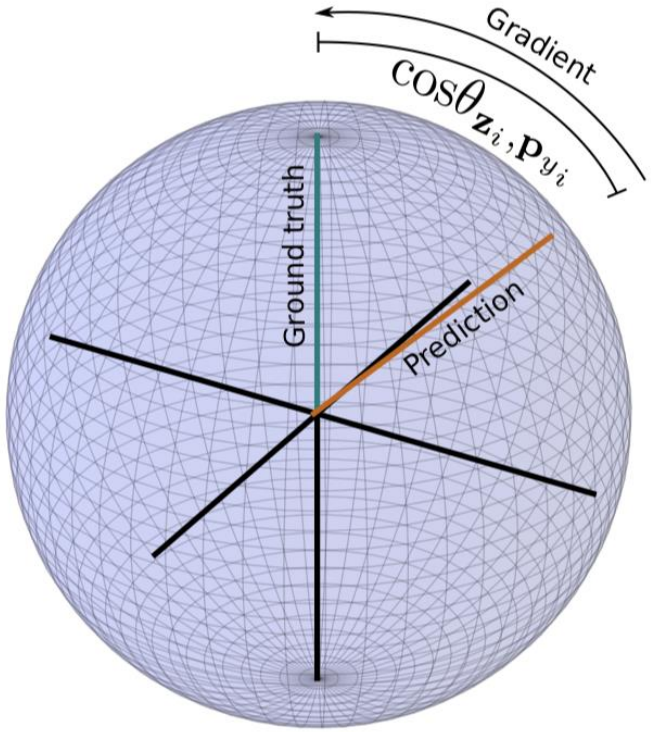
Given pre-computed prototypes, we define the following loss:

$$\mathcal{L}_{\mathrm{c}} = \sum_{i=1}^{N}(1 - \cos\theta_{\mathbf{z}_i,\mathbf{p}_{y_i}})^2 \quad = \sum_{i=1}^{N}(1 - \frac{|\mathbf{z}_i \cdot \mathbf{p}_{y_i}|}{||\mathbf{z}_i|| \ ||\mathbf{p}_{y_i}||})^2.$$

Number of training examples

Network output

Class prototype



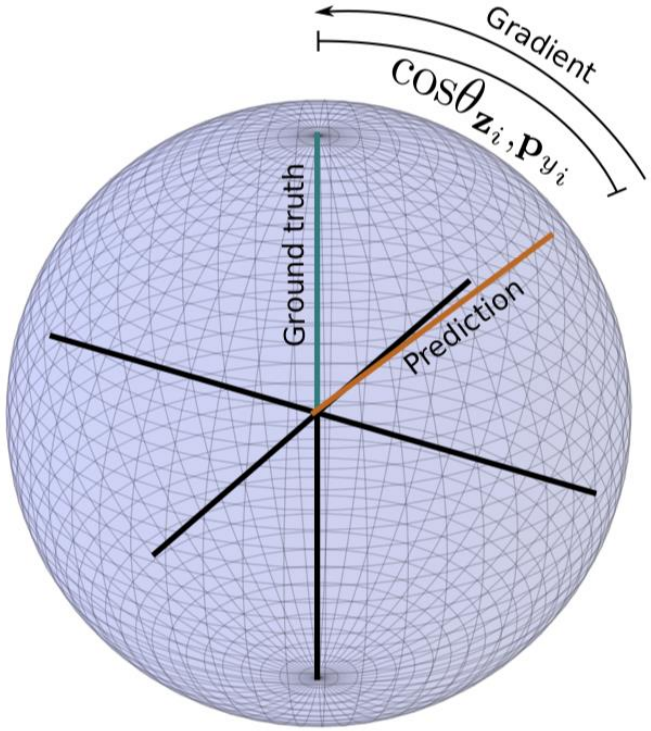
Hyperspherical Prototype classification

Backpropagate partial derivative w.r.t. training examples:

$$\frac{d}{\mathbf{z}_i}(1-\cos\theta_i)^2 = 2\cdot(1-\cos\theta_i)\cdot\frac{d}{\mathbf{z}_i}(1-\cos\theta_i),$$

$\theta_{\mathbf{z}_i,\mathbf{p}_{y_i}}$

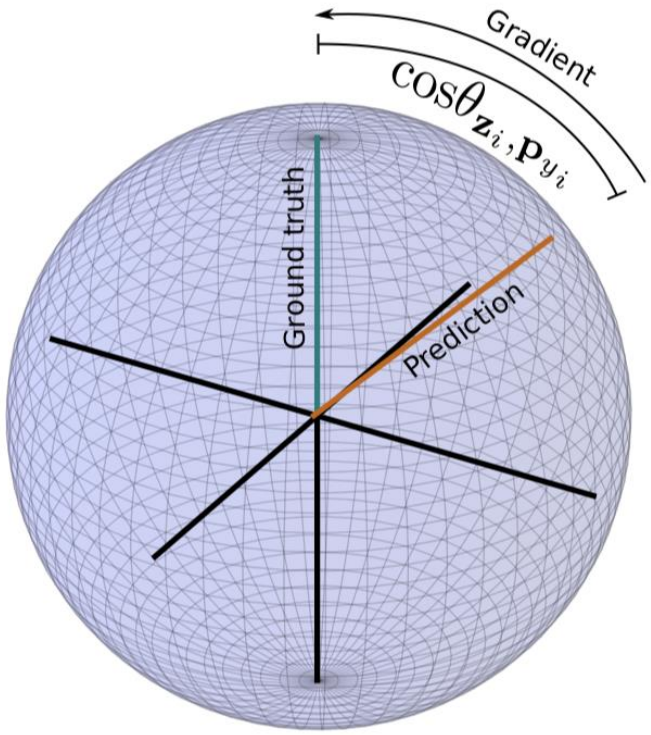
$$\frac{d}{\mathbf{z}_i}(1-\cos\theta_i) = \frac{\cos\theta_i\cdot\mathbf{z}_i}{||\mathbf{z}_i||^2} - \frac{\mathbf{p}_{y_i}}{||\mathbf{z}_i||\cdot||\mathbf{p}_{y_i}||}.$$



Hyperspherical Prototype classification

During inference, assign label to class with highest similarity:

$$c^* = \arg\max_{c \in C} \left(\cos \theta_{f_\phi(\tilde{\mathbf{x}}), \mathbf{p}_c} \right).$$



Hyperspherical Prototype classification - PyTorch

```python
def main_train(model, device, trainloader, optimizer, f_loss, epoch):
    # Set mode to training.
    model.train()
    avgloss, avglosscount = 0., 0.

    # Go over all batches.
    for bidx, (data, target) in enumerate(trainloader):
        # Labels to prototypes.
        target = model.polars[target]

        # Data and labels to device.
        data   = torch.autograd.Variable(data).cuda()
        target = torch.autograd.Variable(target).cuda()

        # Compute outputs and losses.
        output = model(data)
        loss   = (1 - f_loss(output, target)).pow(2).sum()

        # Backpropagation.
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

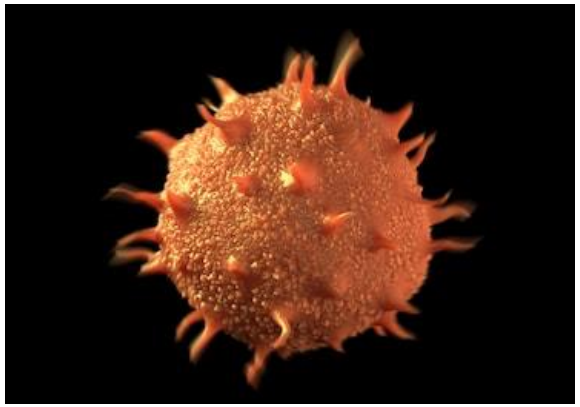
```python
def main_train(model, device, trainloader, optimizer, f_loss, epoch):
    # Set mode to training.
    model.train()
    avgloss, avglosscount = 0., 0.

    # Go over all batches.
    for bidx, (data, target) in enumerate(trainloader):
        # Data to device.
        data   = torch.autograd.Variable(data).cuda()
        target = torch.autograd.Variable(target).cuda()

        # Compute outputs and losses.
        output = model(data)
        loss = f_loss(output, target)

        # Backpropagation.
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Use Cosine Similarity() for loss and class prototypes.

Assign the id as the prototypes.

Obtaining hyperspherical prototypes

P.M.L. Tammes (1930): Pores on pollen grains seem to be uniformly distributed, regardless of the size of the grains or the number of pores. *How do they do it?*



Distributing points on a (hyper)sphere is an open mathematical problem to this day.

Obtaining hyperspherical prototypes

We approximate separation by optimizing the following objective:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}' \in \mathbb{P}} \left(\max_{(k,l,k \neq l) \in C} \cos \theta_{(\mathbf{p}'_k, \mathbf{p}'_l)} \right)$$

We extend the objective with a loss that preserves class semantics:

$$\mathcal{L}_{\text{PI}} = \frac{1}{|T|} \sum_{(i,j,k) \in T} -\bar{S}_{ijk} \log S_{ijk} - (1 - \bar{S}_{ijk}) \log(1 - S_{ijk})$$

Binary label: 1 if *j* closer *i* than *k* (semantically).

Sigmoid of cosine difference of prototypes.

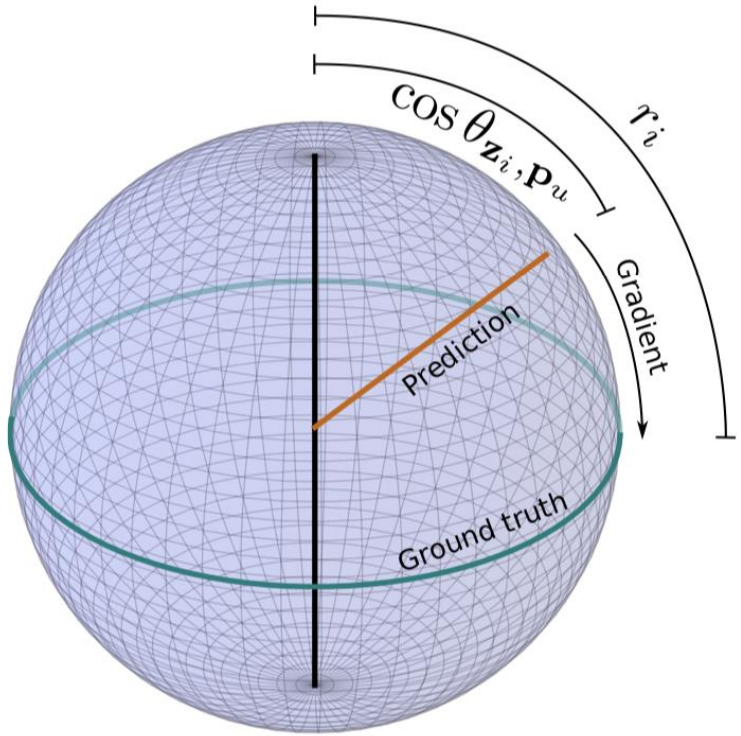
Hyperspherical Prototype Regression

We perform continuous hyperspherical learning as:

$$\mathcal{L}_{\mathrm{r}} = \sum_{i=1}^{N} (r_i - \cos\theta_{\mathbf{z}_i, \mathbf{p}_u})^2,$$

$$r_i = 2 \cdot \frac{y_i - v_l}{v_u - v_l} - 1.$$

Normalized position between upper and lower bound.

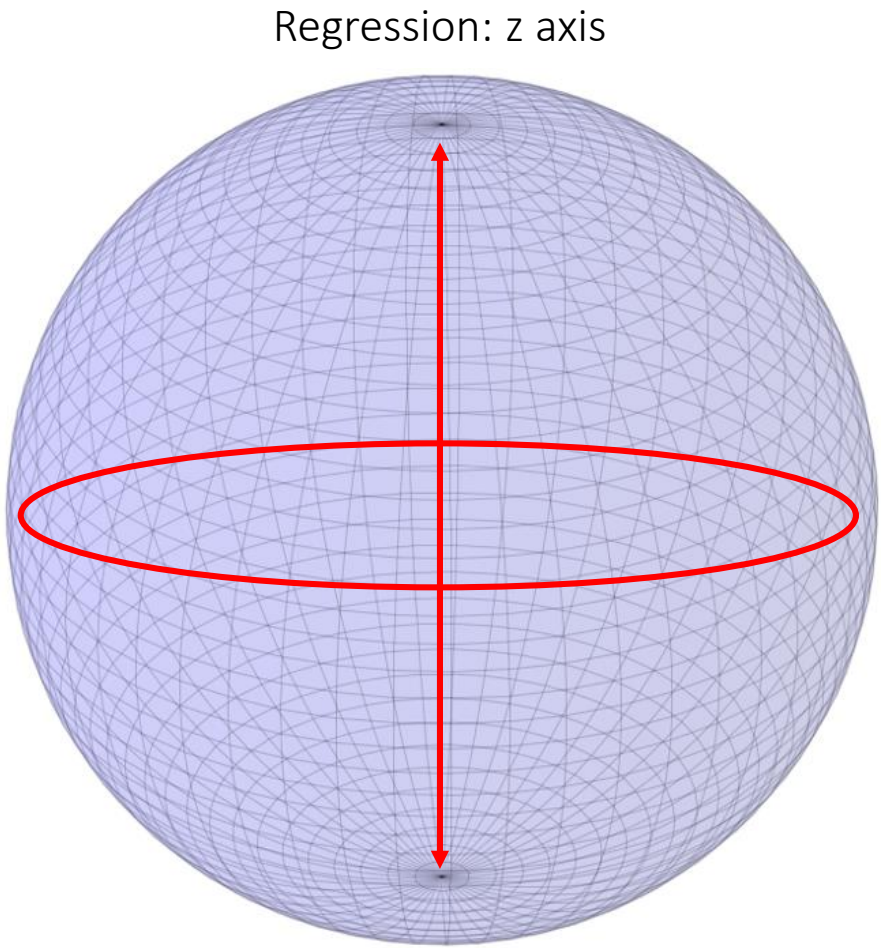


Optimization same as for classification.

Joint classification and regression

We can now jointly optimize both tasks simultaneously:

1. in the same space,
2. without the need to tune the scaling for the gradient backpropagation.



Embedding in related work

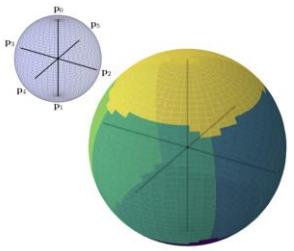
Prototype networks

- ✓ Clearly structured output space
- ⛔ Chicken-egg problem
- ⛔ Constant prototype updating

Classification with angular separation

- ✓ Improved margins for classification
- ⛔ Still in softmax cross-entropy shackle
- ⛔ No application for regression

Evaluating hyperspherical prototypes



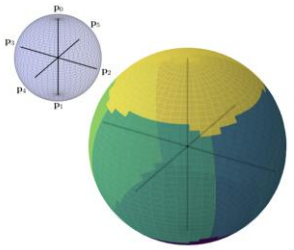
	CIFAR-100			
Dimensions	10	25	50	100
One-hot	-	-	-	62.1 ± 0.1
Word2vec	29.0 ± 0.0	44.5 ± 0.5	54.3 ± 0.1	57.6 ± 0.6
This paper	**51.1** ± 0.7	**63.0** ± 0.1	**64.7** ± 0.2	**65.0** ± 0.3

Akin to softmax cross-entropy

Prototypes from word embeddings

Our prototypes outperform baselines, especially with small outputs

Evaluating prototypes with privileged informatic



	CIFAR-100			
Dimensions	3	5	10	25
Hyperspherical prototypes	5.5 $\pm$ 0.3	28.7 $\pm$ 0.4	51.1 $\pm$ 0.7	63.0 $\pm$ 0.1
w/ privileged info	**11.5** $\pm$ 0.4	**37.0** $\pm$ 0.8	**57.0** $\pm$ 0.6	**64.0** $\pm$ 0.2

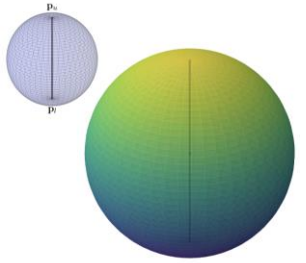
The smaller the output, the bigger the benefit of privileged information

Comparative evaluation

	CIFAR-100		CUB-200
ex / class	500	2 to 200	∼30
Softmax CE	64.4 ± 0.4	44.2 ± 0.0	43.1 ± 0.6
This paper	65.0 ± 0.3	**46.4** ± 0.0	**47.3** ± 0.1

Better, more stable, and lower entropy than state-of-the-art prototype network

Evaluating hyperspherical regression



1480

1754

1190

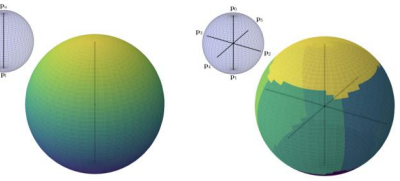
1890

[Strezoski et al. TMM 2019]

	Omniart			
Output space	2D		3D	
Learning rate	1e-2	1e-3	1e-2	1e-3
MSE	210.7 ± 140.1	110.3 ± 0.8	339.9 ± 0.0	109.9 ± 0.5
This paper	**84.4** ± 10.7	**76.3** ± 5.6	**82.9** ± 1.9	**73.2** ± 0.6

More effective regression than standard mean squared error

Evaluating joint classification and regress



Proof of concept

Jointly predict MNIST digit (classification) and rotation (regression).

Quantitative evaluation

Multi-task learning of creation year (regression) and art style (classification).

	Creation year (MAE ↓)	Art style (Accuracy ↑)
MTL baseline	354.7 ± 7.8	47.2 ± 0.4
This paper	**68.3** ± 1.1	**52.6** ± 0.1

Future of deep learning

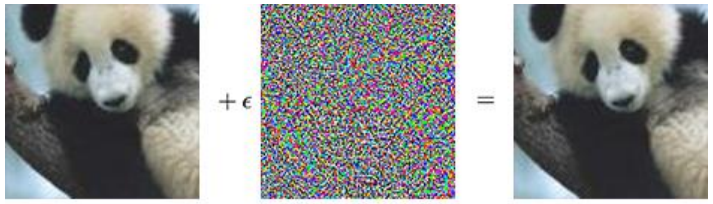
Open research problems - 1

Correlation vs. Understanding

You: What is an elephant?

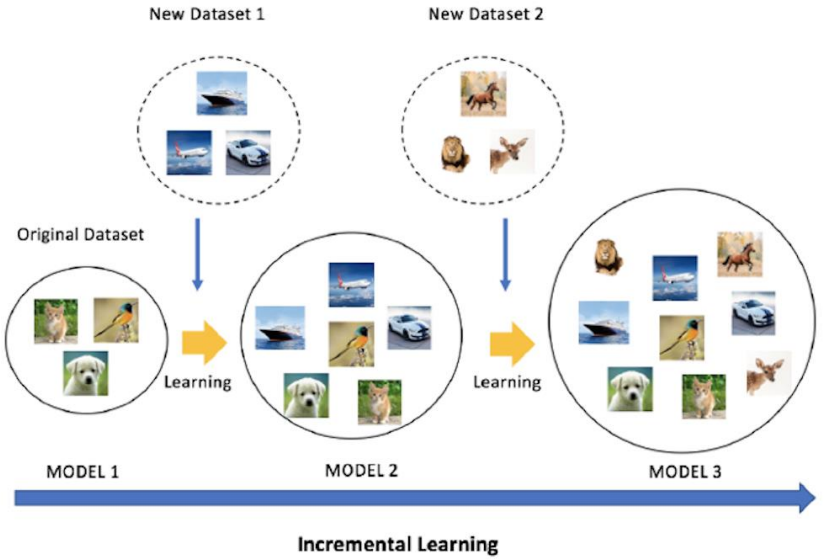
Deep network: Show me an example and
I can tell you if it has an elephant.

Adversarial learning



[Goodfellow et al. ICLR, 2015]

Life-long learning



[Kuo et al. JVCIR, 2018]

Take-away messages

Deep learning boils down to learning with forward backward propagation.

Many vision tasks can be tackled with deep learning.

- Classification, regression, detection, segmentation, etc.

Hardly any problem is solved, research needed everywhere.

- We can hardly solve the current set of problems.
- Many problems not tackled, especially on learning to learn and consciousness.

Afternoon session

Part I: MLPs and CNNs in PyTorch.

- Learn essentials for training and evaluating deep networks.

Part II: RNNs.

- Build your own RNN and LSTM step-by-step.

Slide credits

Slides and code of this course taken from my course “Applied Machine Learning” and the “Deep Learning” course by Efstratios Gavves.

Link: https://uvadlc.github.io