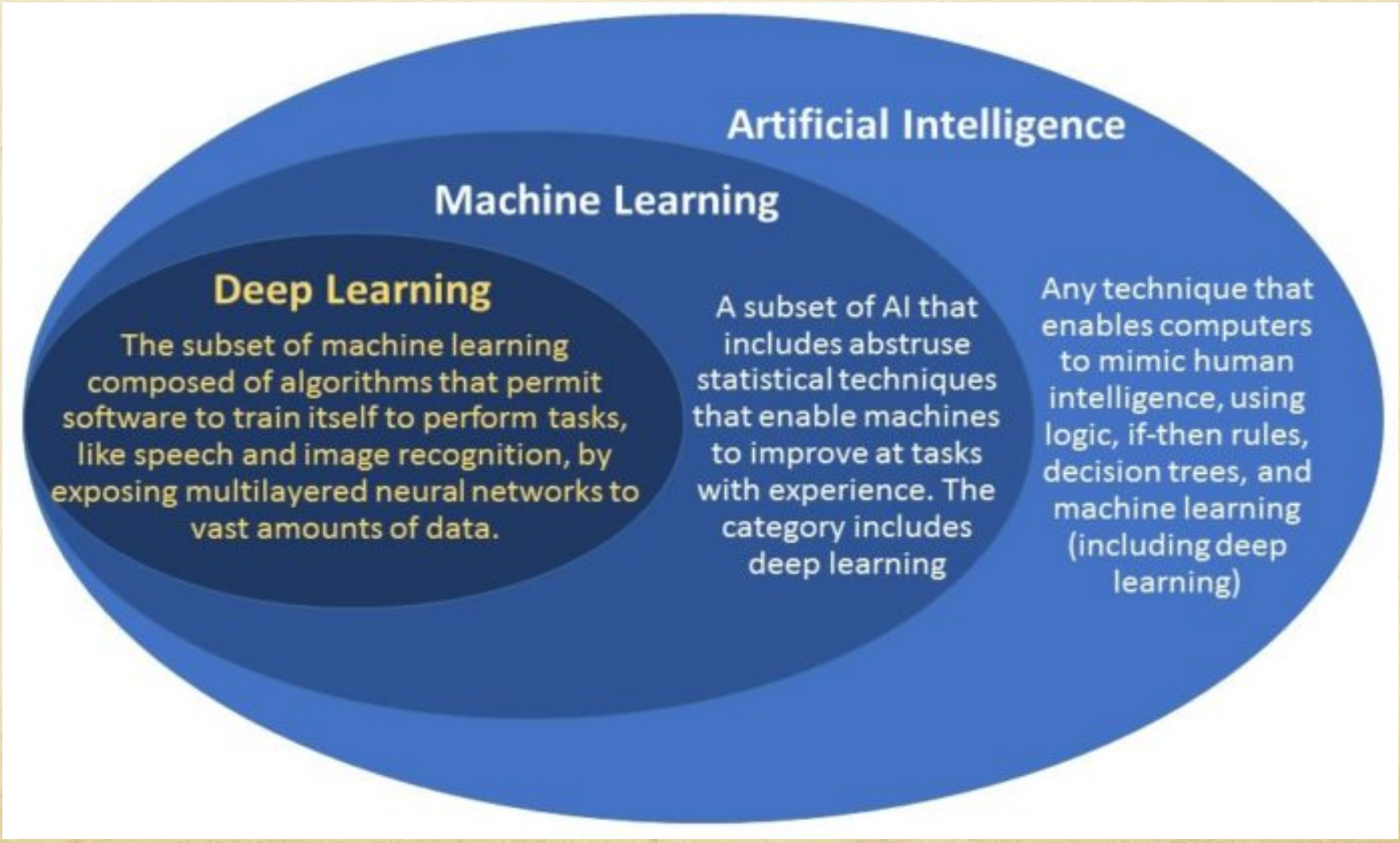
What is Deep learning ?



Deep Learning in practice

YouTube

Large-scale Video Classification with Convolutional Neural Networks, CVPR 2014

Youtube

Microsoft Deep Learning Semantic Image Segmentation

Website

Youtube

Deep Sensorimotor Learning

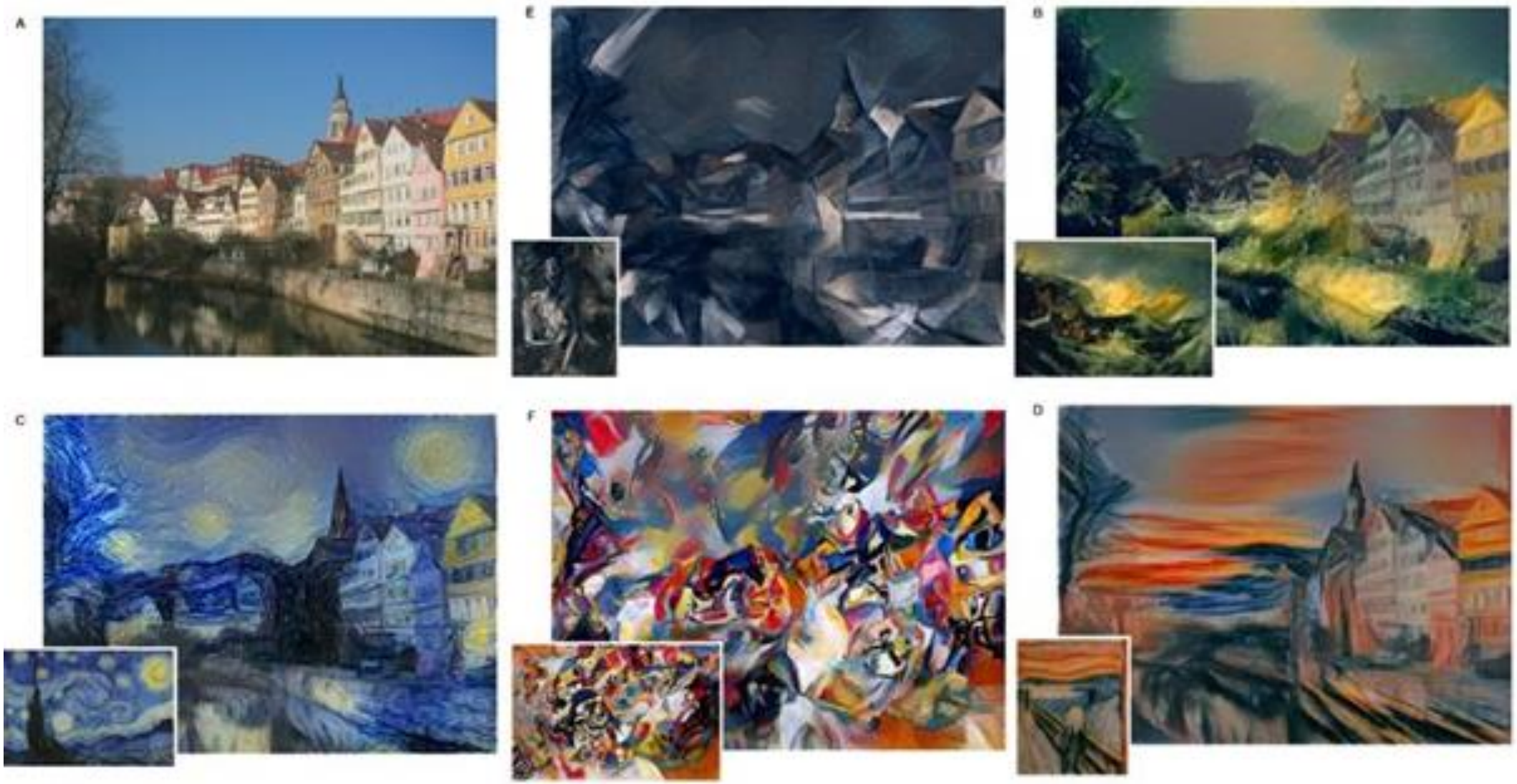
Youtube

Google DeepMind's Deep Q-learning playing Atari Breakout

Newspapers			
New York San Jose	New York Times San Jose Mercury News	Baltimore Cincinnati	Baltimore Sun Cincinnati Enquirer
NHL Teams			
Boston Phoenix	Boston Bruins Phoenix Coyotes	Montreal Nashville	Montreal Canadiens Nashville Predators
NBA Teams			
Detroit Oakland	Detroit Pistons Golden State Warriors	Toronto Memphis	Toronto Raptors Memphis Grizzlies
Airlines			
Austria Belgium	Austrian Airlines Brussels Airlines	Spain Greece	Spainair Aegean Airlines
Company executives			
Steve Ballmer Samuel J. Palmisano	Microsoft IBM	Larry Page Werner Vogels	Google Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

Deep Learning even for the arts



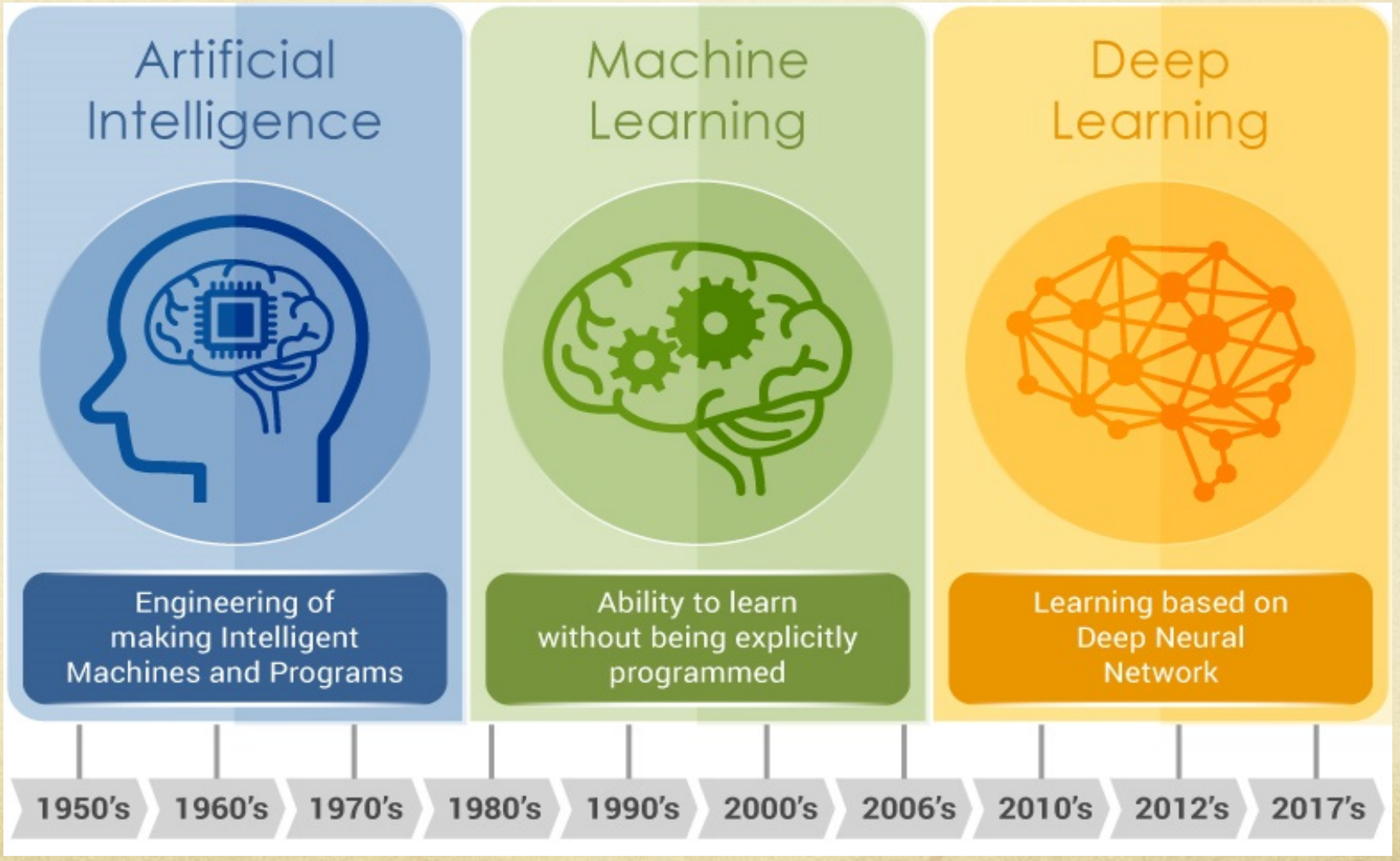
Why should we be impressed?

- **Vision** is ultra challenging!
  - For 256x256 resolution → $2^{524,288}$ of possible images (10^{24}stars in the universe)
  - Large visual object variations (viewpoints, scales, deformations, occlusions)
  - Large semantic object variations
- **Robotics** is typically considered in controlled environme
- **Game AI** involves extreme number of possible games states ($10^{10^{48}}$ possible GO games)
- **NLP** is extremely high dimensional and vague (just for English: 150K words)

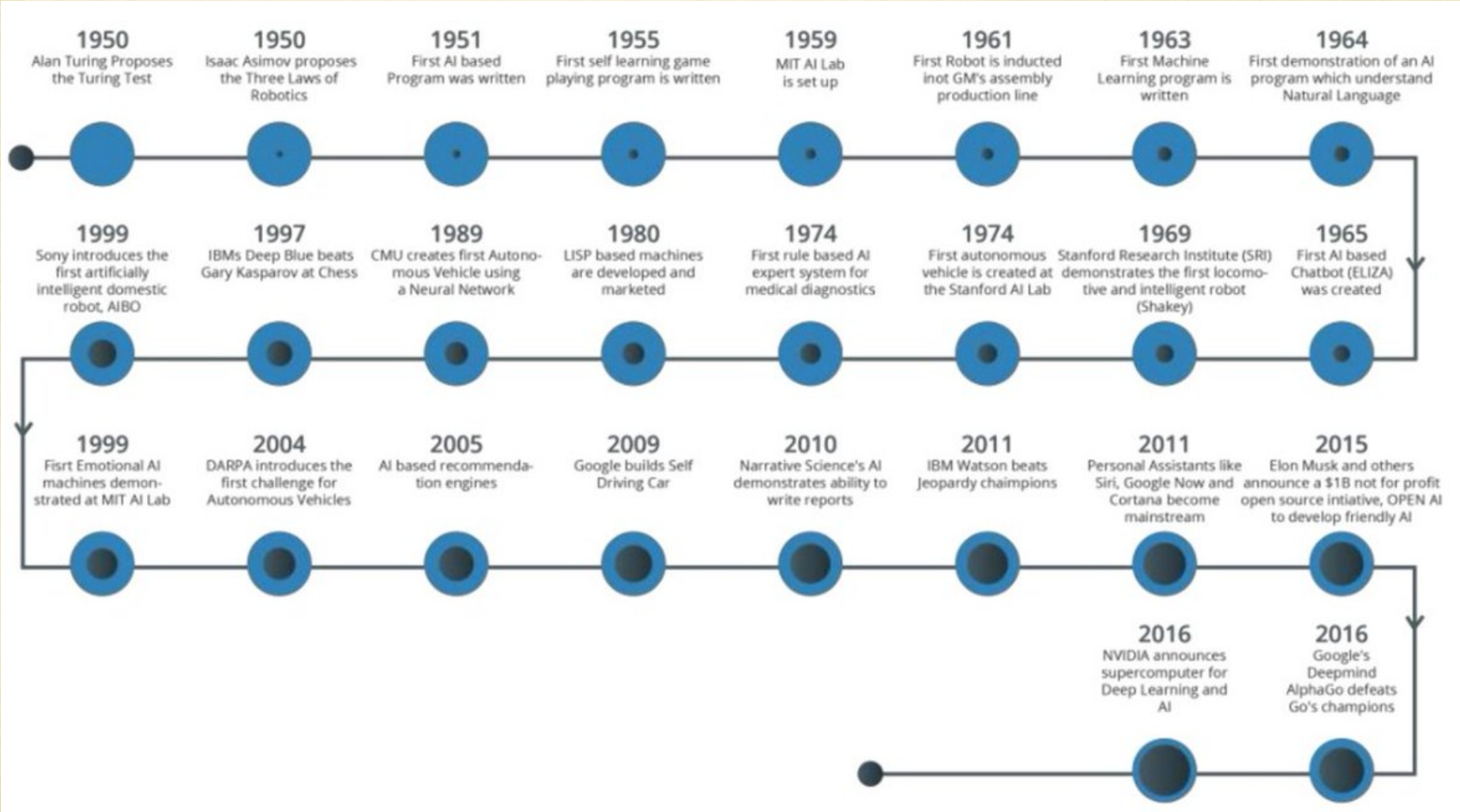
Inter-class variation

Intra-class overlap

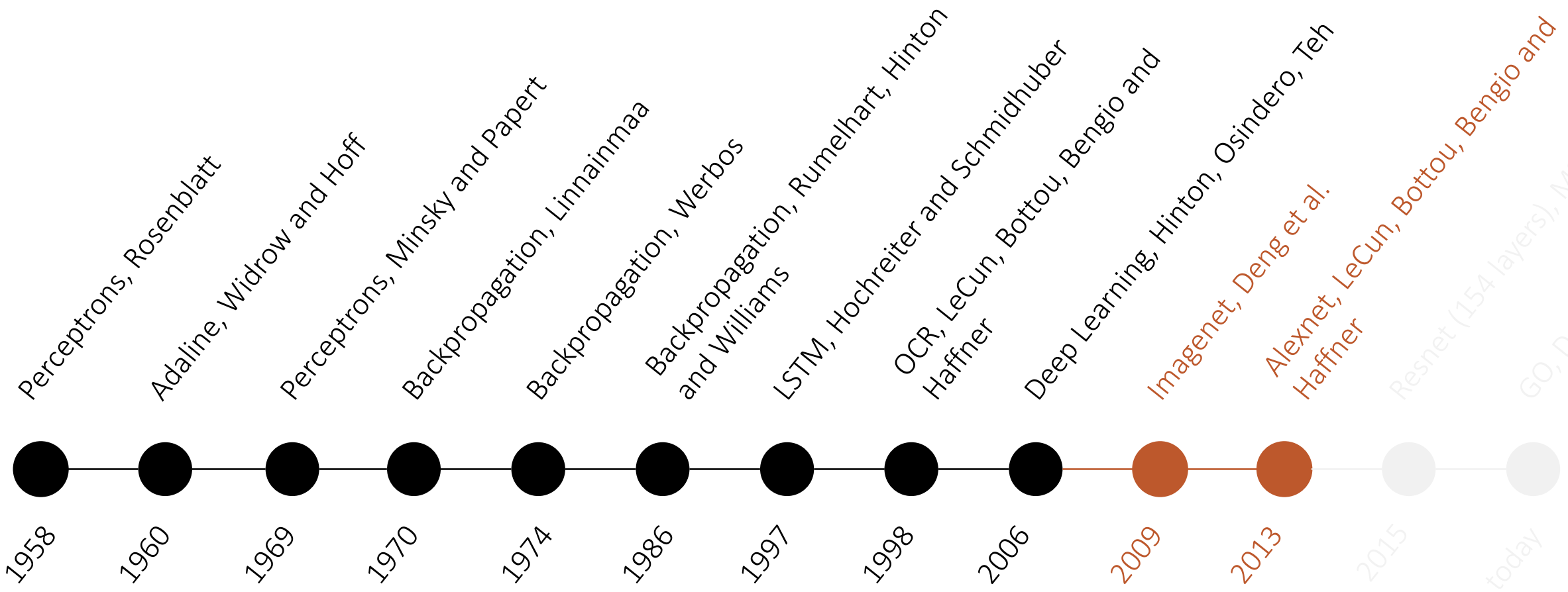
History of development



History of development



Deep Learning Renaissance



Alexnet architecture

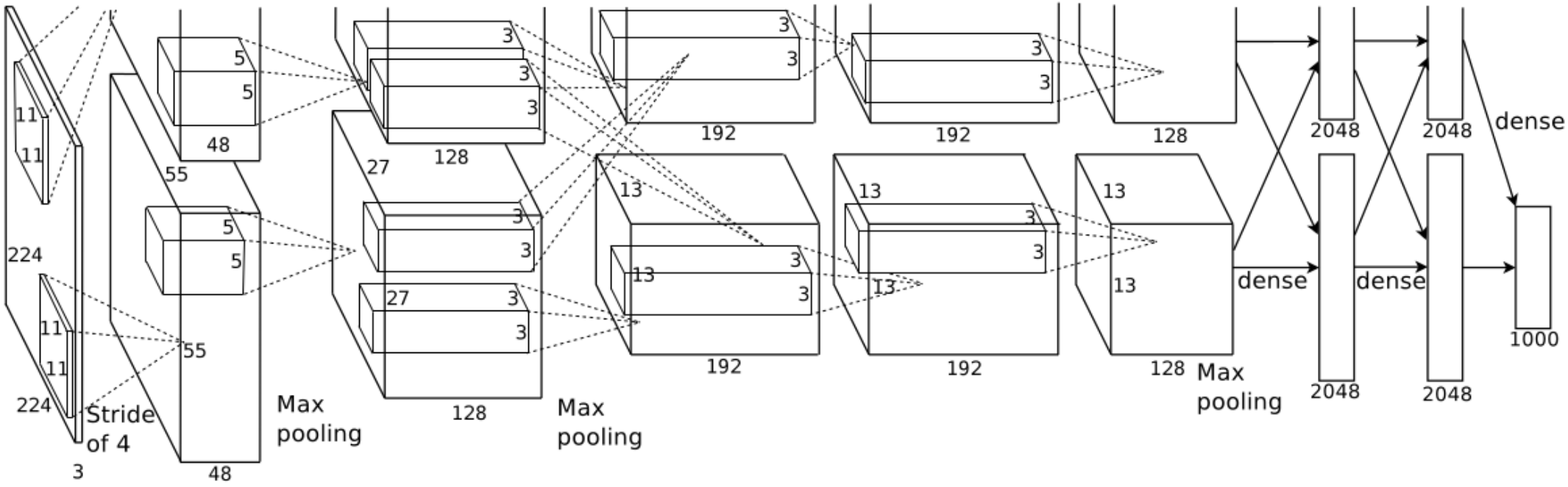


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Deep Learning is **Big Data Hungry!**

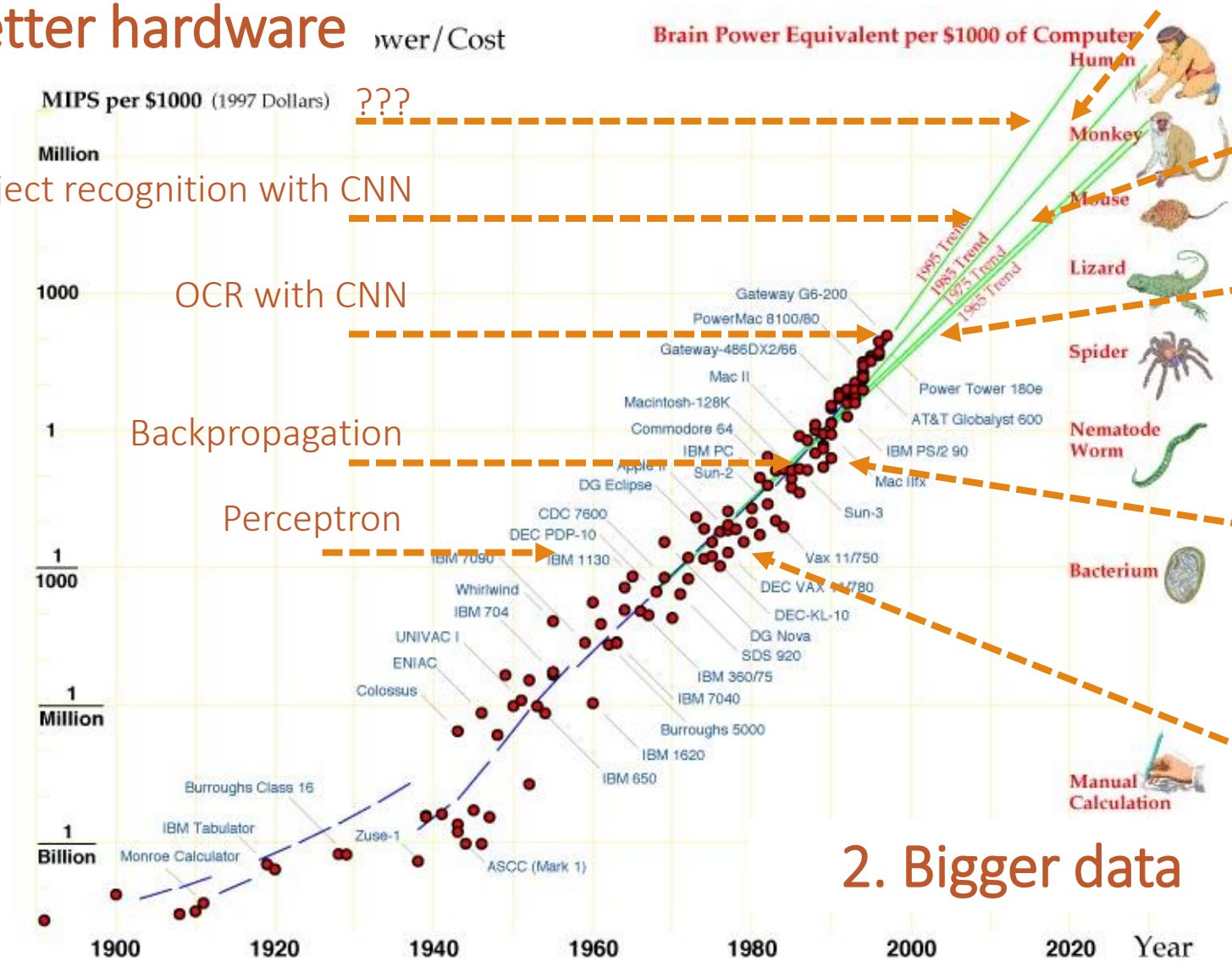
- In 2009 the Imagenet dataset was published [Deng et al., 2009]
  - Collected images for each of the 100K terms in Wordnet (16M images in total)
  - Terms organized hierarchically: “Vehicle”→“Ambulance”

- Imagenet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1 million images
  - 1,000 classes
  - Top-5 and top-1 error measured

Why now?

1. Better hardware

Datasets of everything (captions, question-answering, …), reinforcement learning, ???



???

Object recognition with CNN

OCR with CNN

Backpropagation

Perceptron

2. Bigger data

Imagenet: 1,000 classes from real images, 1,000,000 images

Results:

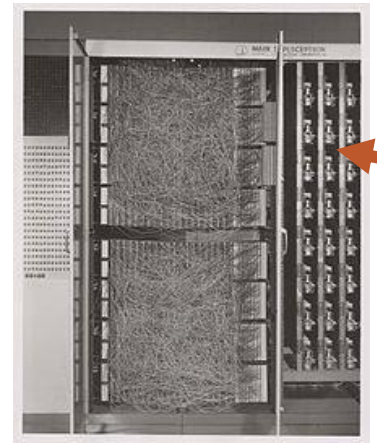
- Persian cat: 0.35211
- Egyptian cat: 0.23635
- hamster: 0.20282
- tiger cat: 0.05896
- lynx: 0.05759

Bank cheques

Parity, negation problems

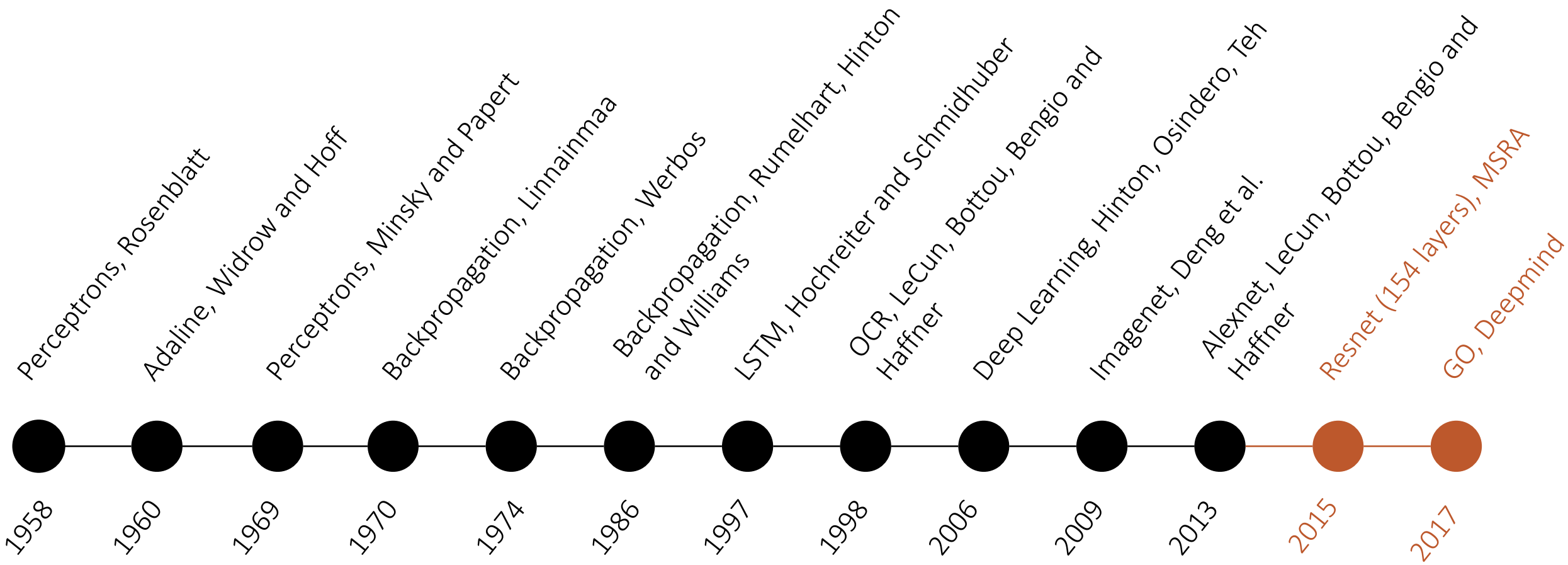
D1	D2	D3	Even-Parity
0	0	0	True
0	0	1	False
0	1	0	False
0	1	1	True
1	0	0	False
1	0	1	True
1	1	0	True
1	1	1	False

Mark I Perceptron

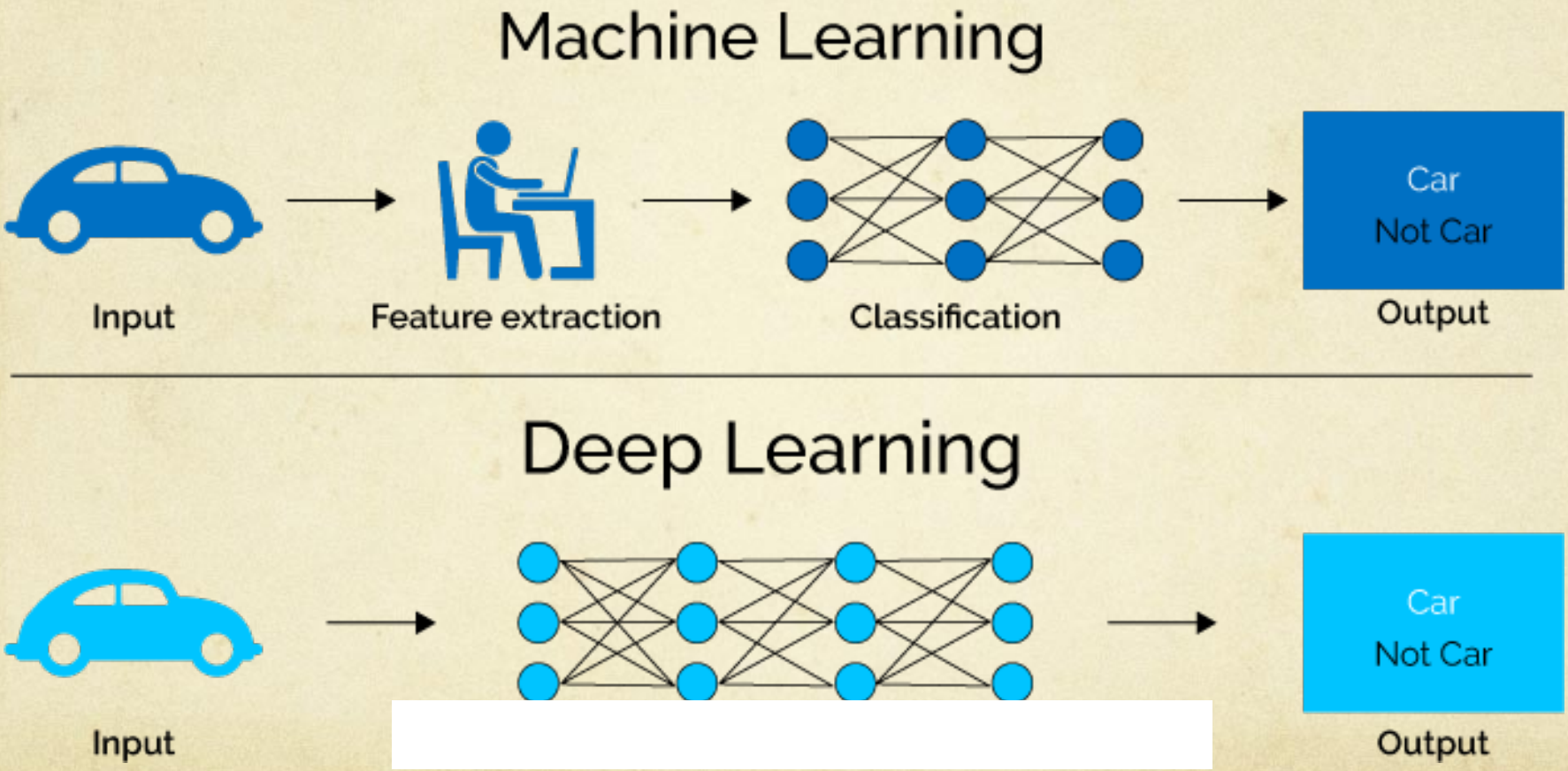


Potentiometers implement perceptron weights

Deep Learning Golden Era

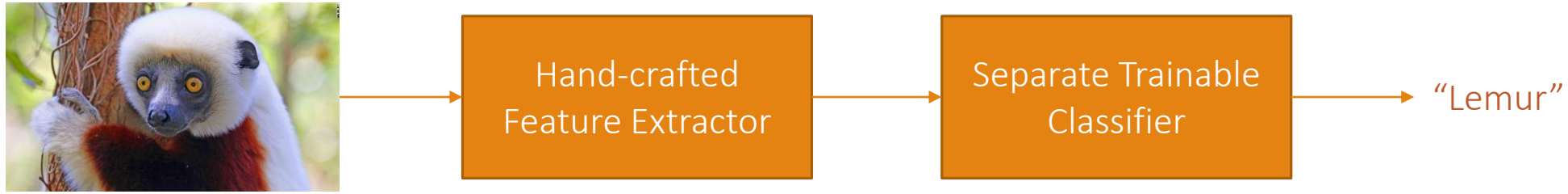


Deep Learning vs Machine Learning

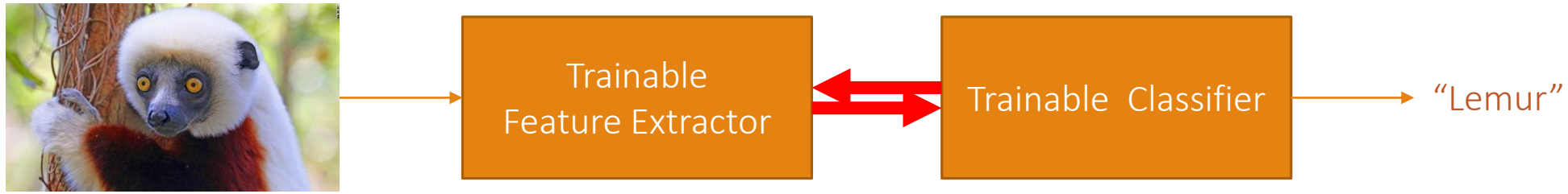


Learning Representations & Features

- Traditional pattern recognition



- End-to-end learning → Features are also learned from data

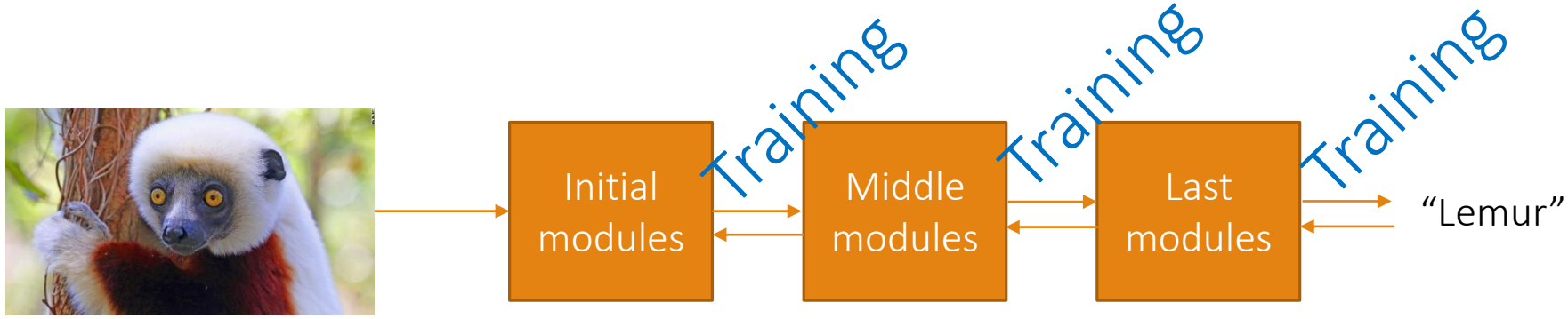


Why learn the features?

- Manually designed features
  - Expensive to research & validate
- Learned features
  - If data is enough, easy to learn, compact and specific
- Time spent for designing features now spent for designing architectures

End-to-end learning of feature hierarchies

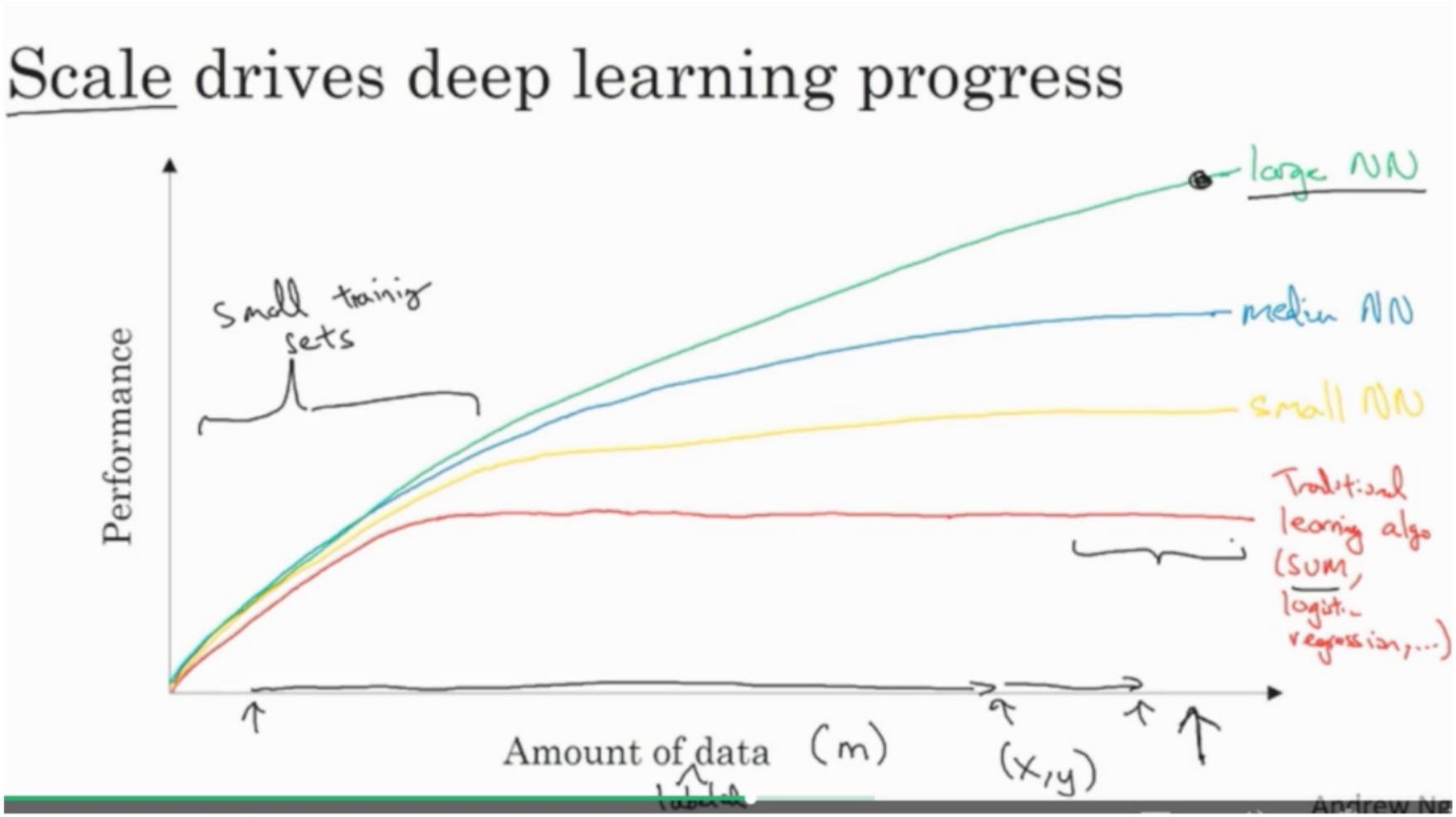
- A pipeline of successive, differentiable modules
  - Each module's output is the input for the next module
- Each subsequent module produce higher abstraction features
- Preferably, input as raw as possible



Compare to traditional ML

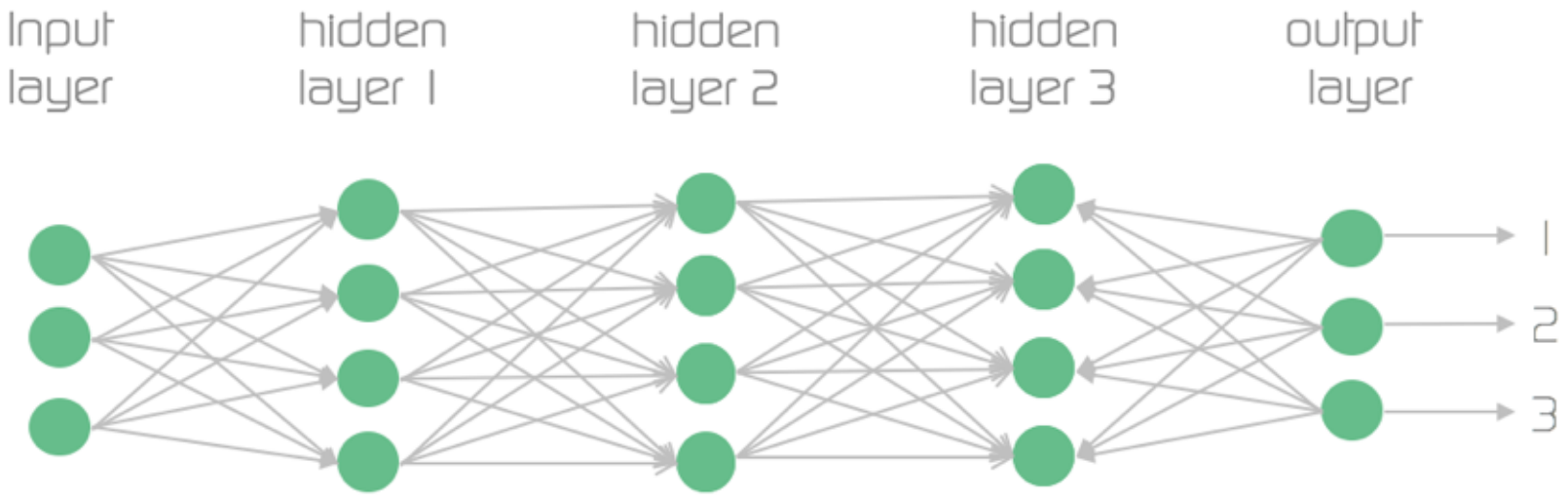
- No **expert** requirement
- No features selection
- No features extraction computation
- No optimal **feature combination**

Compare to traditional ML



Why deep learniyng is called **Convolutional Neural Network**???

Basic structure of neural network



Basic definition:

-Neural

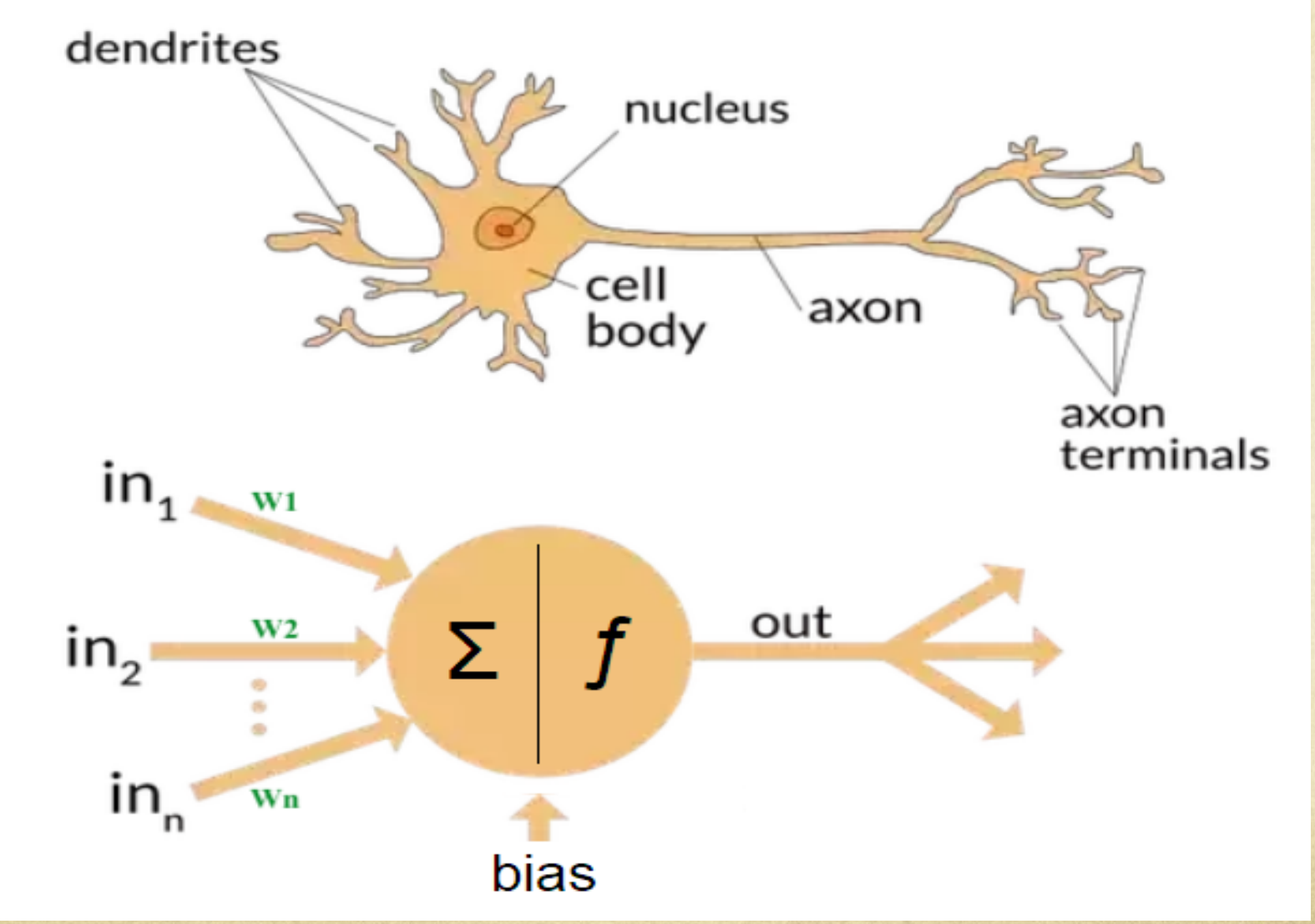
-Link → **"Dense Network"**

-Layers → **"Deep" Neural Network**

-Input layer and output layer

-Hidden layers

A SINGLE NEURAL

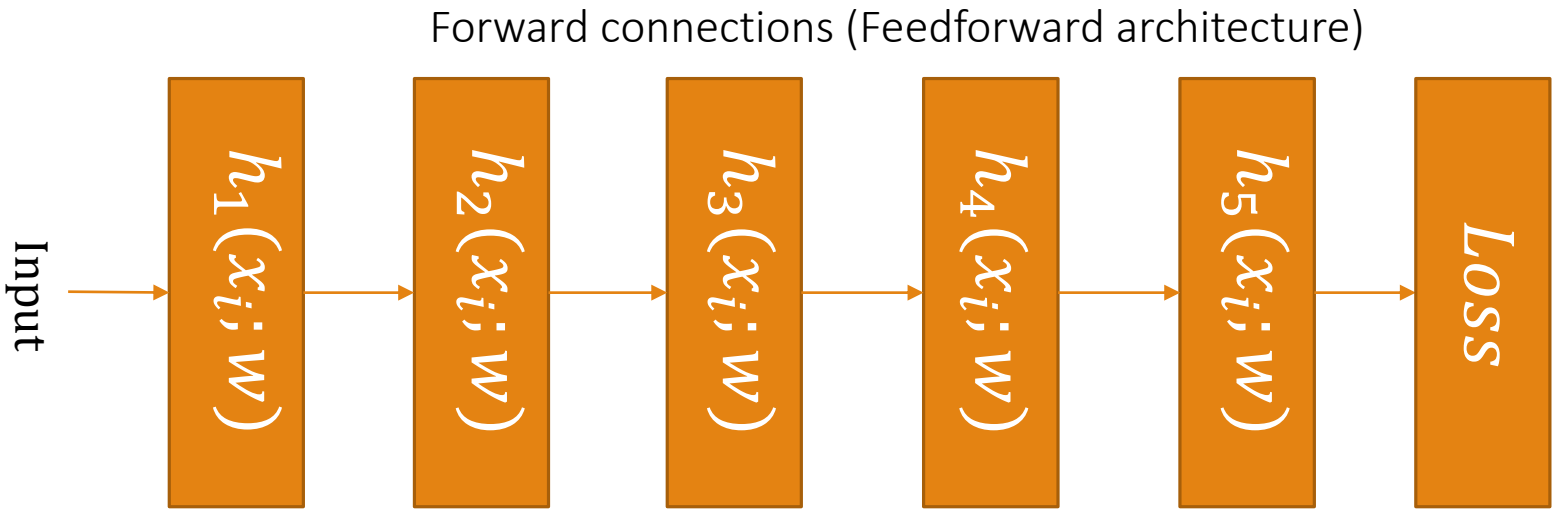


How the neural work?

- -Take information from inputs
- -Combine all information it has
- -Put on a decision functions
- -Make a decision!!!

Neural network models

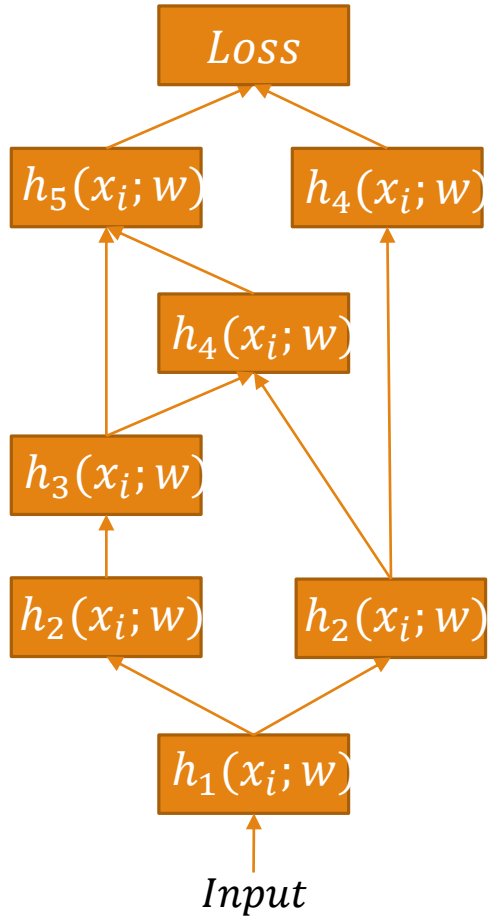
- A neural network model is a series of hierarchically connected functions
- This hierarchies can be very, very complex



Neural network models

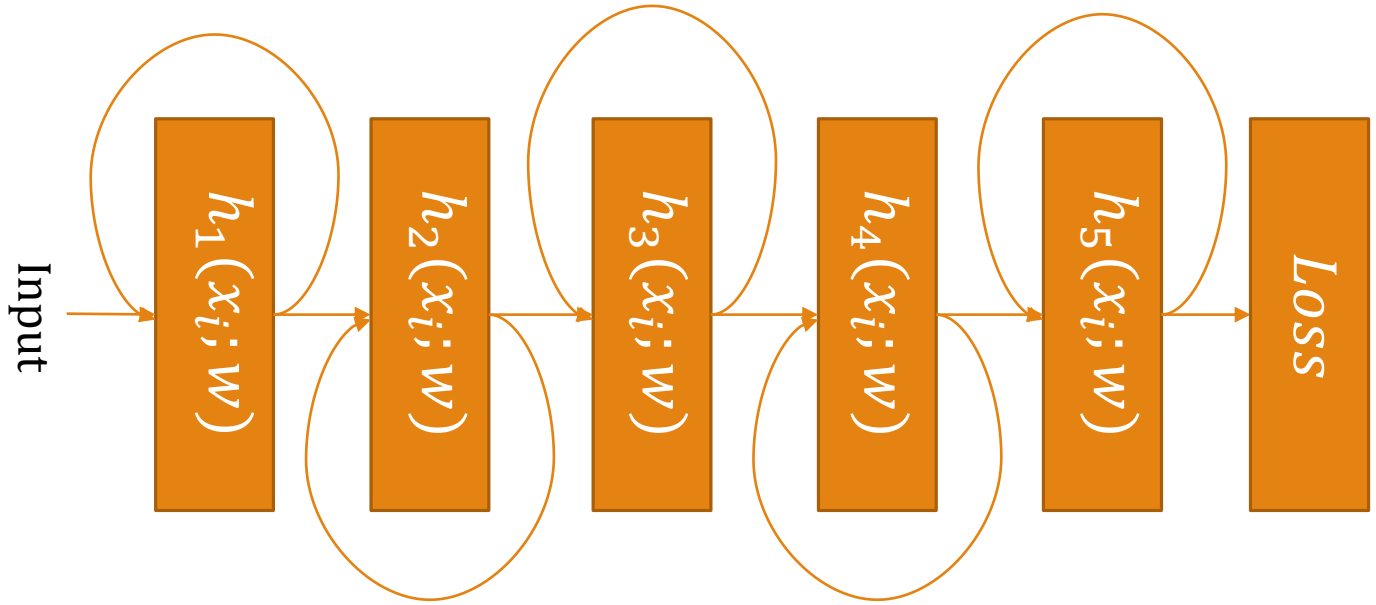
- A neural network model is a series of hierarchically connected functions
- This hierarchies can be very, very complex

Interweaved connections (Directed Acyclic Graphs- DAGNN)



Neural network models

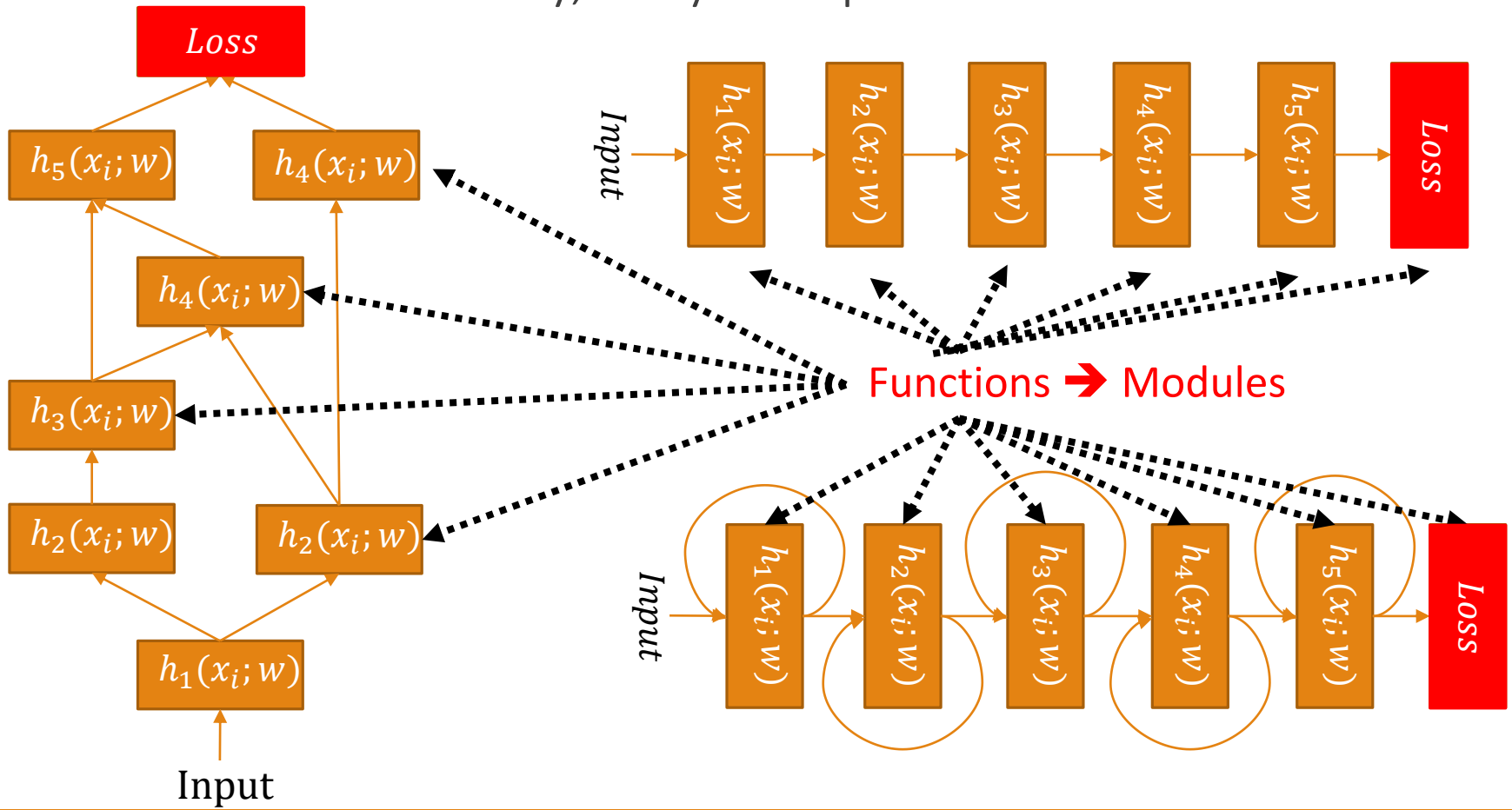
- A neural network model is a series of hierarchically connected functions
- This hierarchies can be very, very complex



Loopy connections (Recurrent architecture, special care needed)

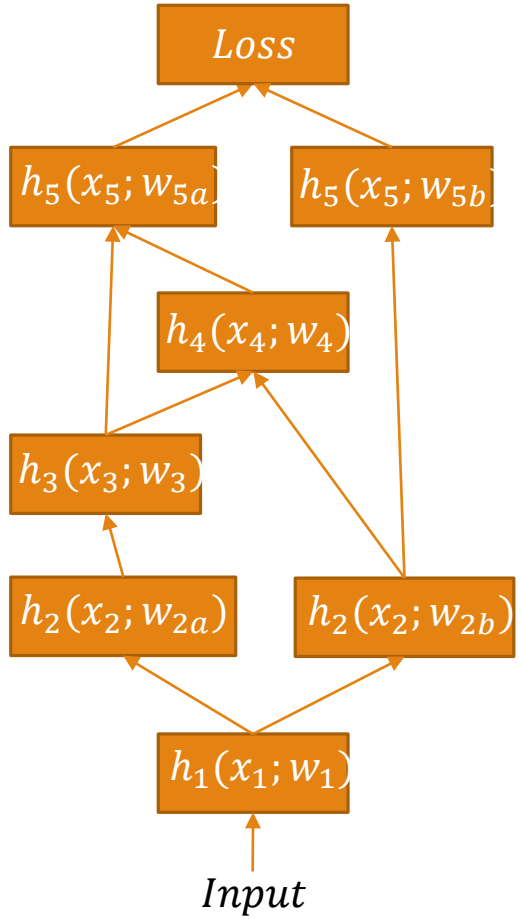
Neural network models

- A neural network model is a series of hierarchically connected functions
- This hierarchies can be very, very complex



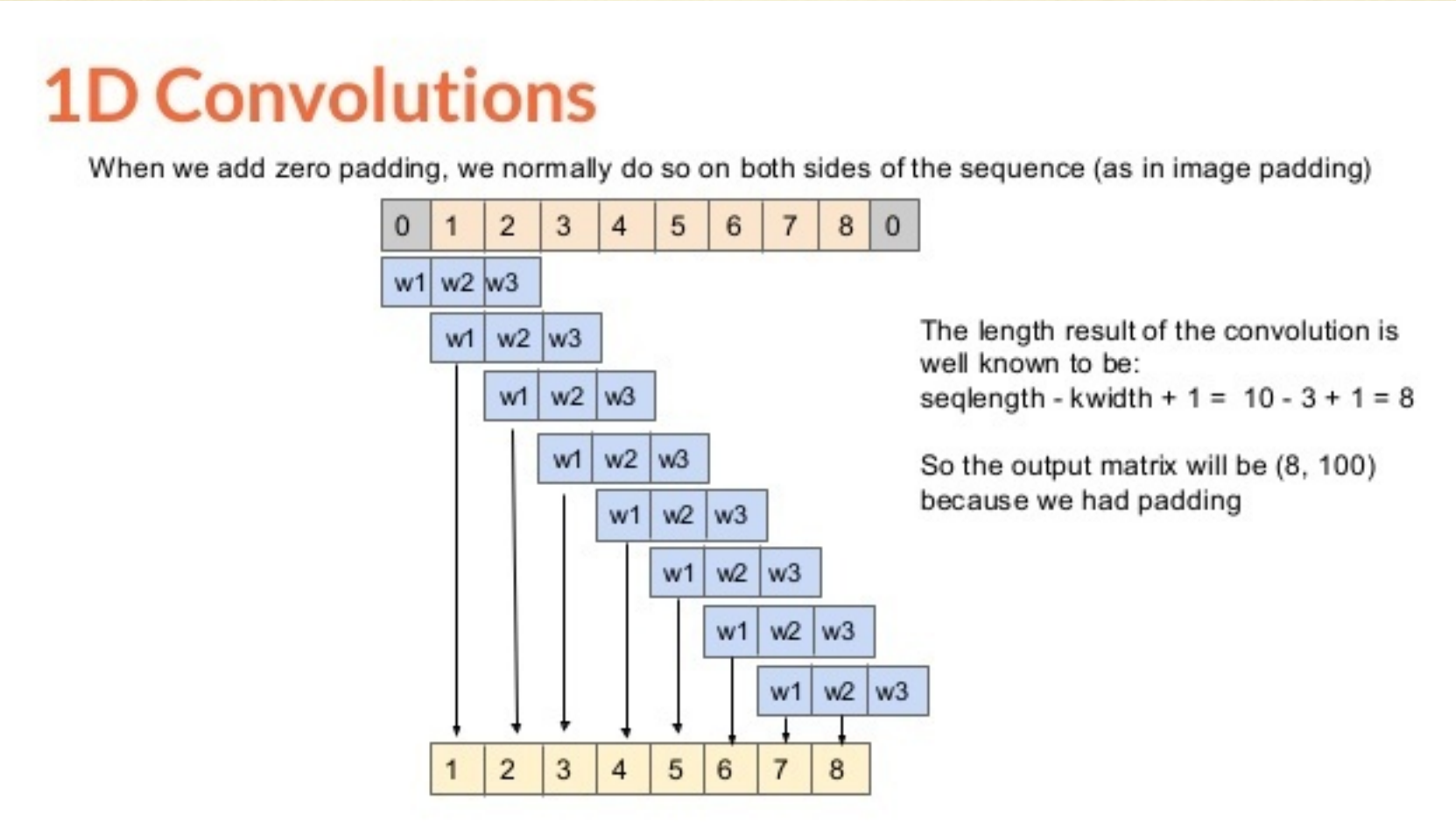
What is a module?

- A module is a building block for our network
- Each module is an object/function $a = h(x; w)$ that
  - Contains trainable parameters w
  - Receives as an argument an input x
  - And returns an output a based on the activation function $h(\dots)$
- The activation function should be (at least) **first order differentiable (almost) everywhere**
- For easier/more efficient backpropagation → store module input
  - easy to get module output fast
  - easy to compute derivatives

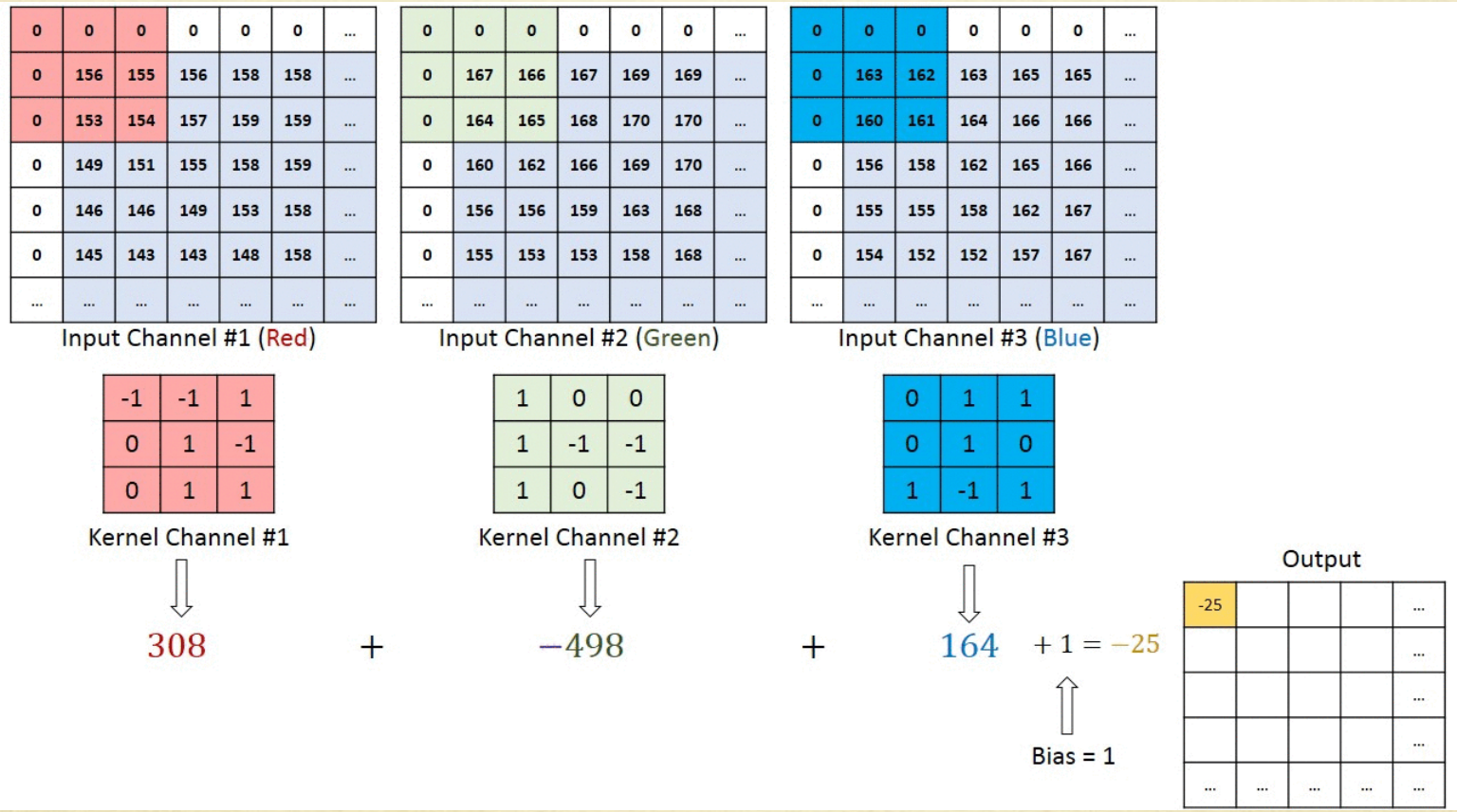


Convolution operation

Convolution= Shift and Sum:



Convolution operation

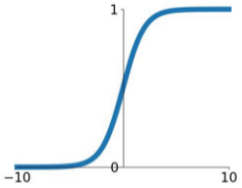


Active functions

Activation Functions

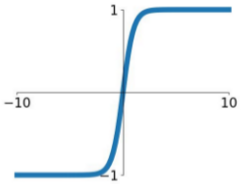
Sigmoid

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



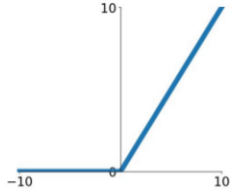
tanh

$$\tanh(x)$$



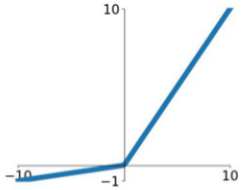
ReLU

$$\max(0, x)$$



Leaky ReLU

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

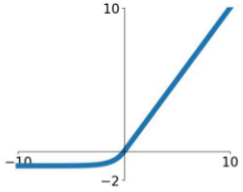


Maxout

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

ELU

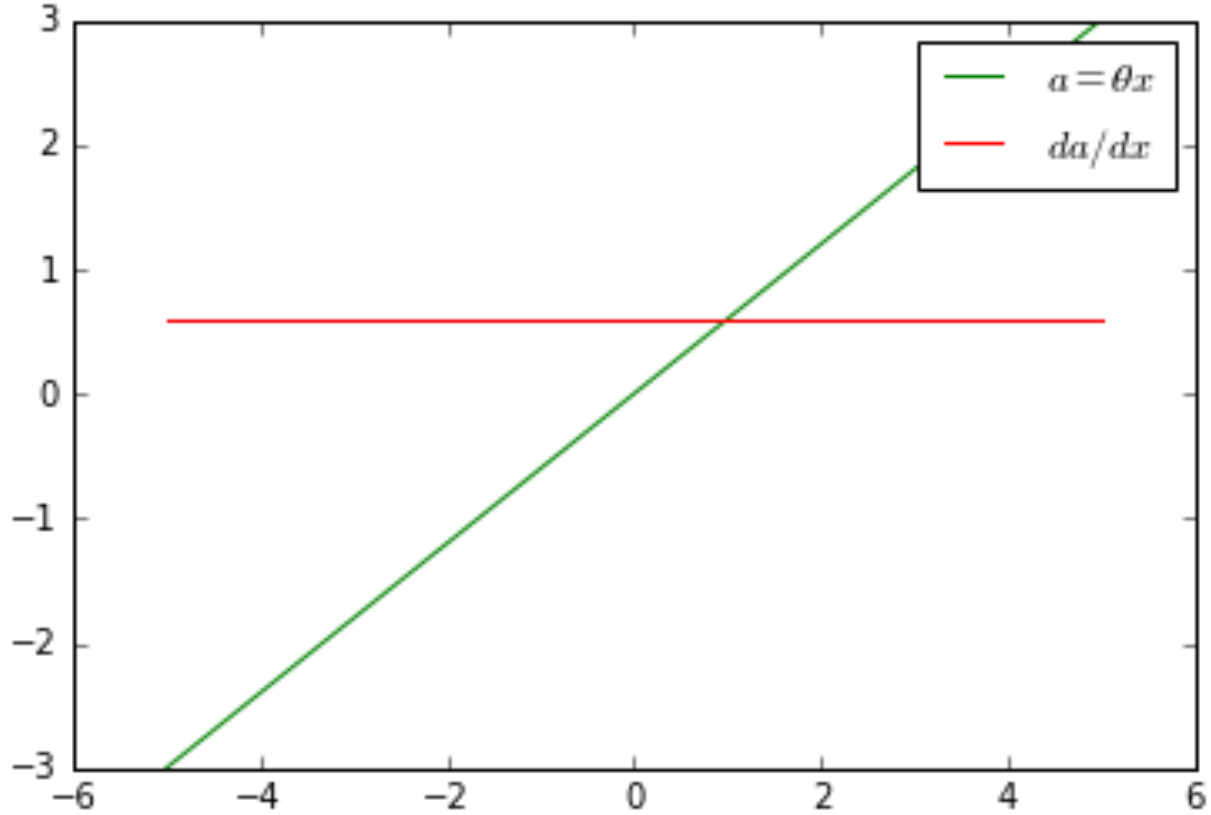
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Most deep networks use ReLU for hidden layers because of faster and to prevent gradient vanishing problem

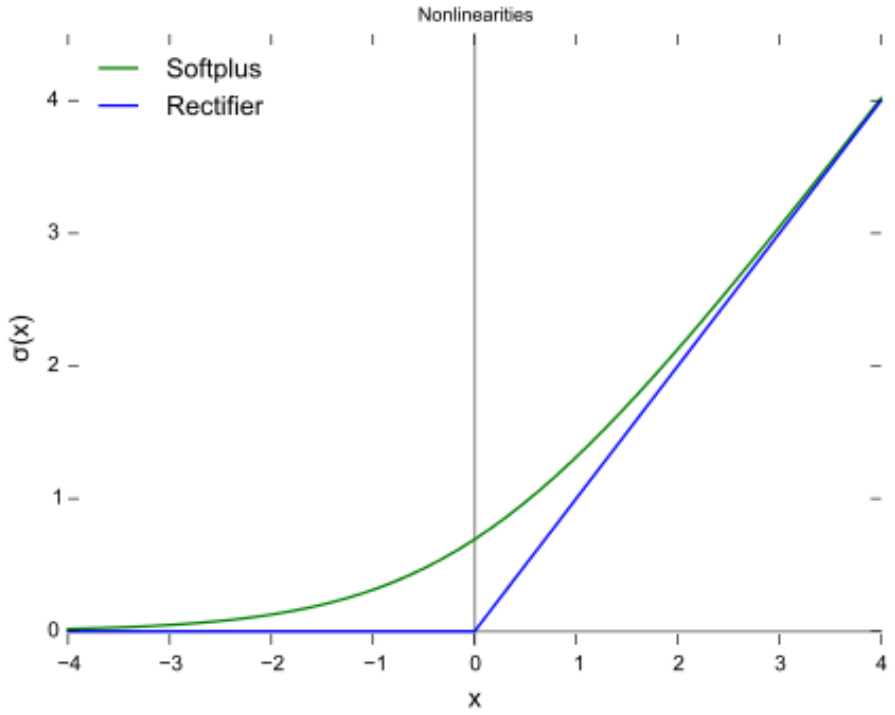
Linear module

- Activation: $a = wx$
- Gradient: $\frac{\partial a}{\partial w} = x$
- No activation saturation
- Hence, strong & stable gradients
  - Reliable learning with linear modules



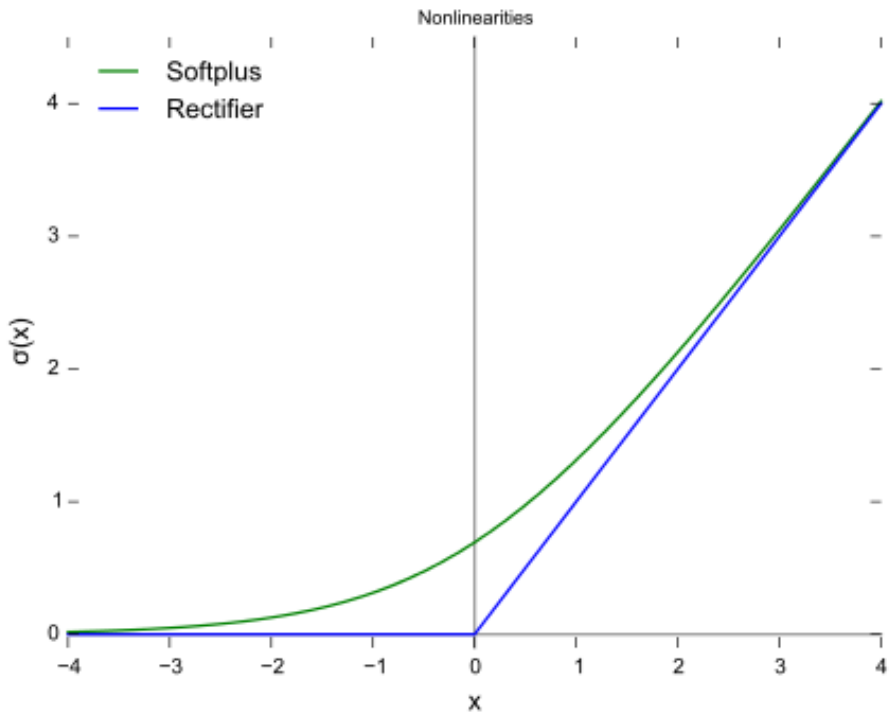
Rectified Linear Unit (ReLU) module

- Activation: $a = h(x) = \max(0, x)$
- Gradient: $\frac{\partial a}{\partial x} = \begin{cases} 0, if\ x \leq 0 \\ 1, if x > 0 \end{cases}$

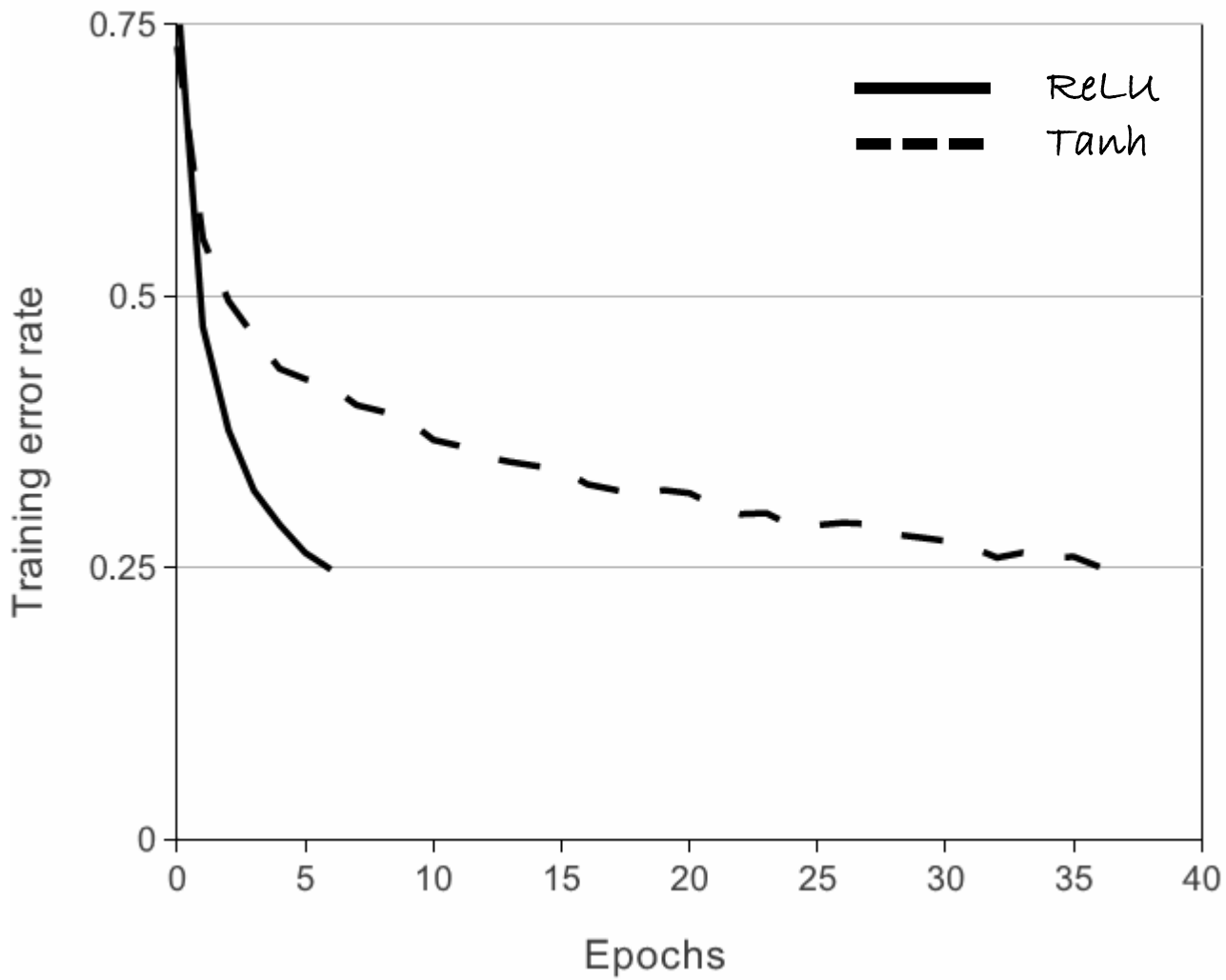


Rectified Linear Unit (ReLU) module

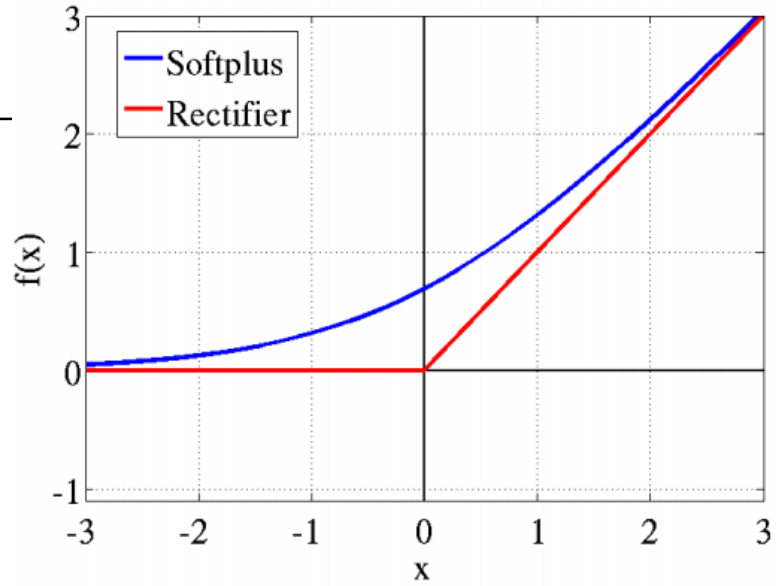
- Activation: $a = h(x) = \max(0, x)$
- Gradient: $\frac{\partial a}{\partial x} = \begin{cases} 0, if\ x \leq 0 \\ 1, if x > 0 \end{cases}$
- Strong gradients: either 0 or 1 ☺
- Fast gradients: just a binary comparison ☺
- It is not differentiable at 0, but not a big problem ☺
  - An activation of precisely 0 rarely happens with non-zero weights, and if it happens we choose a convention
- Dead neurons is an issue
  - Large gradients might cause a neuron to die. Higher learning rates might be beneficial
  - Assuming a linear layer before ReLU $h(x) = \max(0, wx + b)$, make sure the bias term b is initialized with a small initial value, $e.g. 0.1$ → more likely the ReLU is positive and therefore there is non zero gradient
- Nowadays ReLU is the default non-linearity



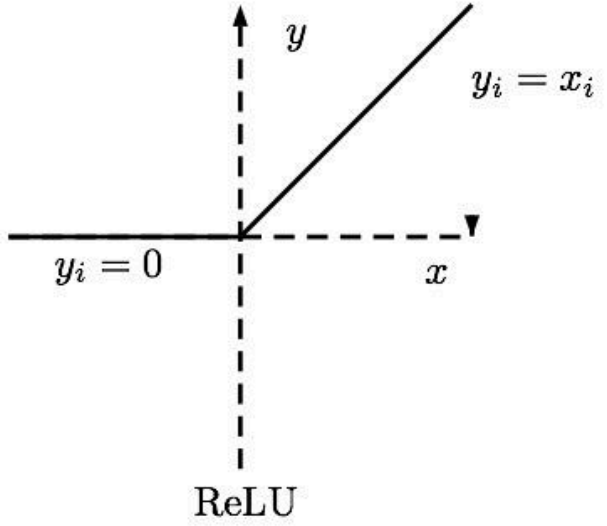
ReLU convergence rate

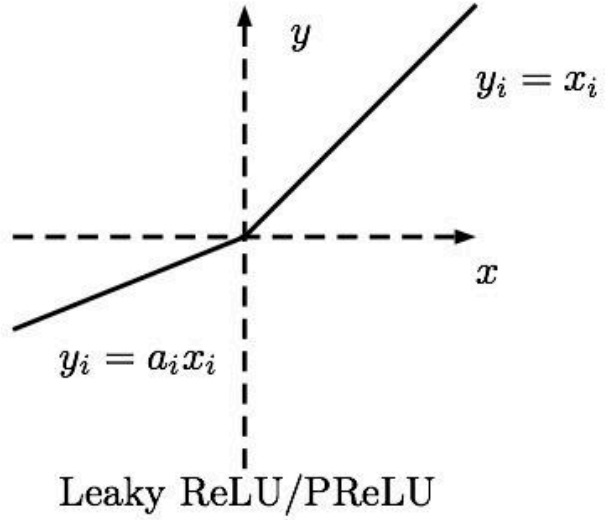


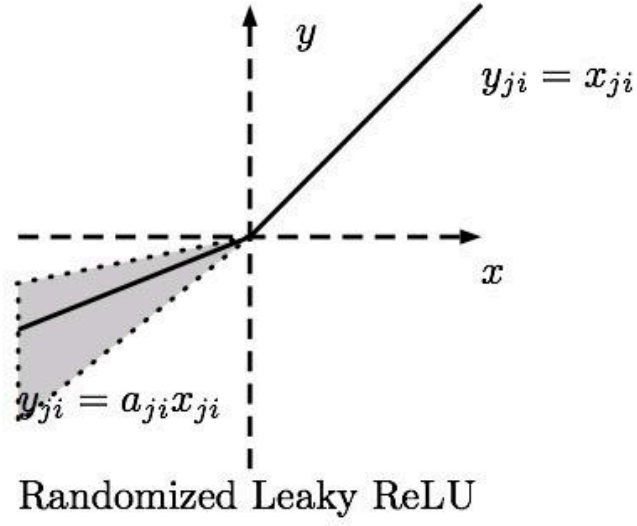
Other ReLUs



- Soft approximation (softplus): $a = h(x) = \ln(1 + e^x)$
- Noisy ReLU: $a = h(x) = \max(0, \mathrm{x} + \varepsilon), \varepsilon \sim N(0, \sigma(\mathrm{x}))$
- Leaky ReLU: $a = h(x) = \begin{cases} x, if\ x > 0 \\ 0.01x\ otherwise \end{cases}$
- Parametric ReLu: $a = h(x) = \begin{cases} x, if\ x > 0 \\ \beta x\ otherwise \end{cases}$ (parameter β is trainable)

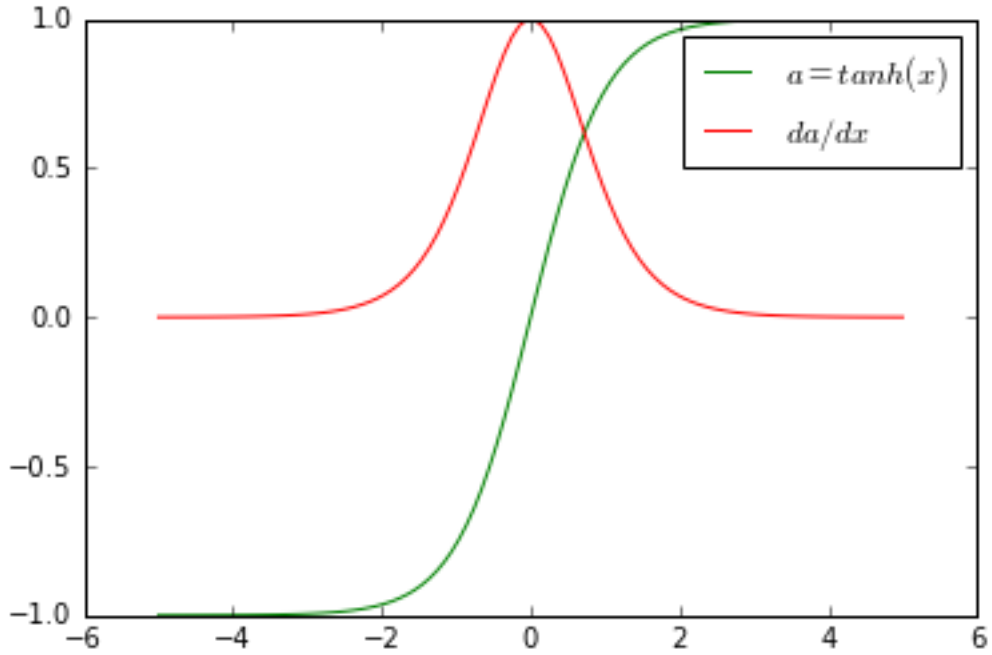






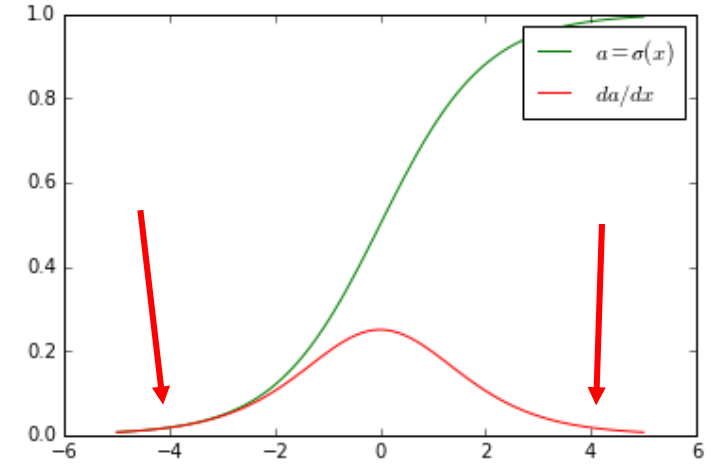
Tanh module

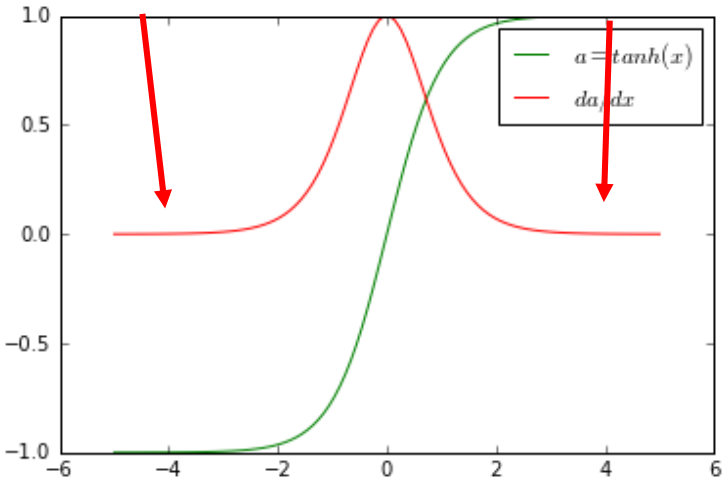
- Activation: $a = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Gradient: $\frac{\partial a}{\partial x} = 1 - tanh^2(x)$



Tanh vs Sigmoids

- Functional form is very similar: $tanh(x) = 2\sigma(2x) - 1$
- $tanh(x)$ has better output $[-1, +1]$ range
  - Stronger gradients, because data is centered around 0 (not 0.5)
  - Less "positive" bias to hidden layer neurons as now outputs can be both positive and negative (more likely to have zero mean in the end)
- Both saturate at the extreme → 0 gradients
  - "Overconfident", without necessarily being correct
  - Especially bad when in the middle layers: why should a neuron be overconfident, when it represents a latent variable
- The gradients are < 1, so in deep layers the chain rule returns very small total gradient
- From the two, $tanh(x)$ enables better learning
  - But still, not a great choice





Sigmoid: An exception

- An exception for sigmoids is when used as the final output layer
- Sigmoid outputs can return very small or very large values (saturate)
  - Output units are not latent variables (have access to ground truth labels)
  - Still “overconfident”, but at least towards true values

Softmax module

- Activation: $a^{(k)} = softmax(x^{(k)}) = \frac{e^{x^{(k)}}}{\sum_j e^{x^{(j)}}}$
  - Outputs probability distribution, $\sum_{k=1}^{K} a^{(k)} = 1$ for K classes
- Avoid exponentianting too large/small numbers → better stability

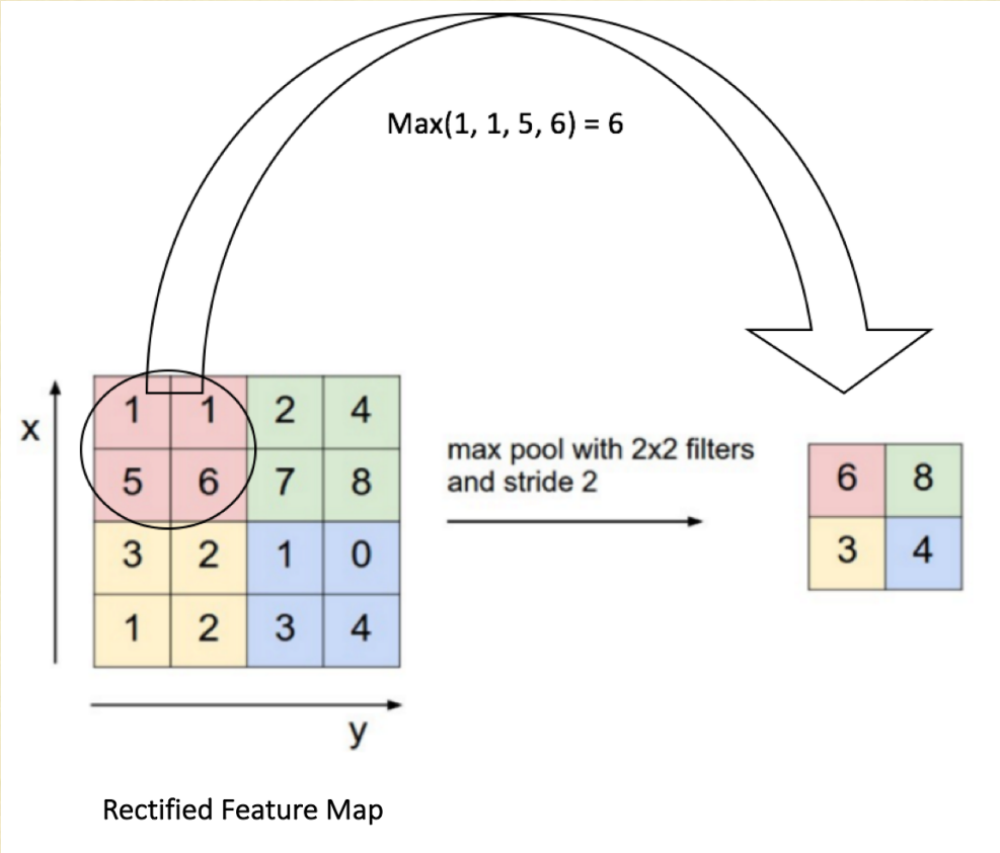
$$a^{(k)} = \frac{e^{x^{(k)}-\mu}}{\sum_j e^{x^{(j)}-\mu}}, \mu = \max_k x^{(k)} \text{ because}$$

$$\frac{e^{x^{(k)}-\mu}}{\sum_j e^{x^{(j)}-\mu}} = \frac{e^{\mu} e^{x^{(k)}}}{e^{\mu} \sum_j e^{x^{(j)}}} = \frac{e^{x^{(k)}}}{\sum_j e^{x^{(j)}}}$$

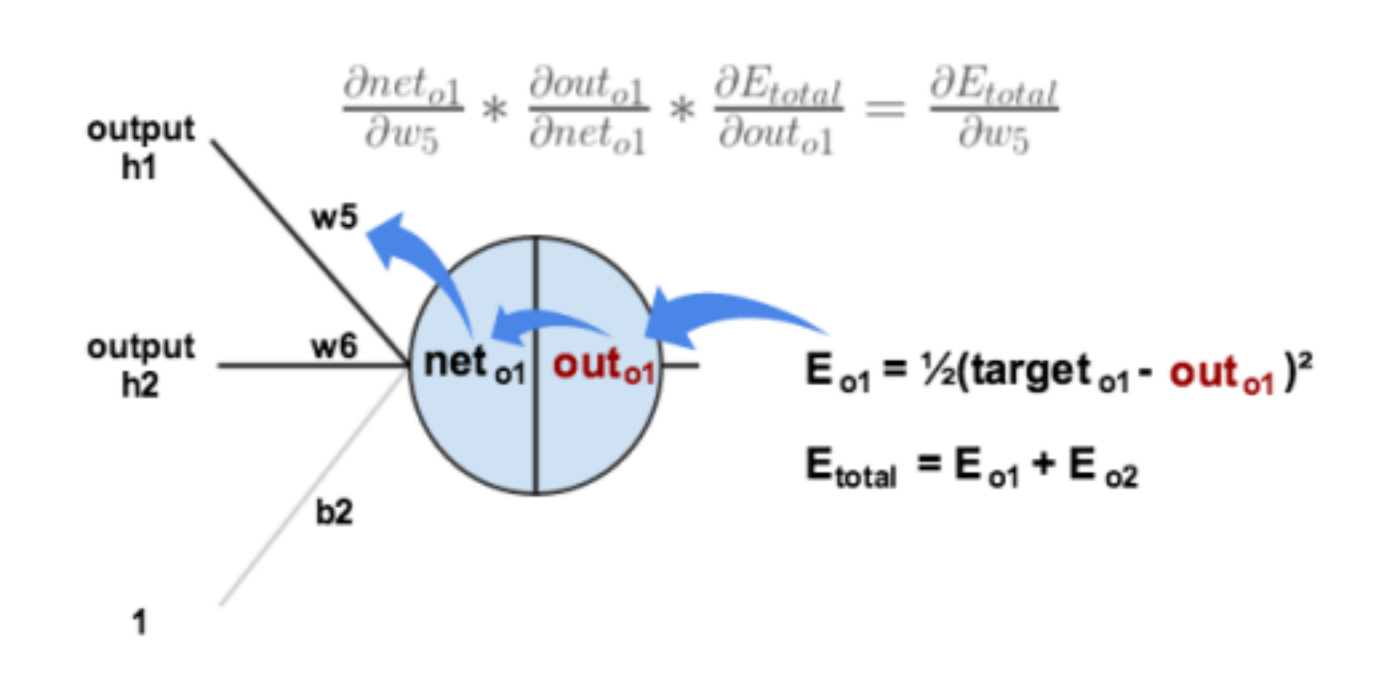
Many, many more modules out there …

- Many will work comparably to existing ones
  - Not interesting, unless they work consistently better and there is a reason
- Regularization modules
  - Dropout
- Normalization modules
  - ℓ_2-normalization, ℓ_1-normalization
- Loss modules
  - Hinge loss
- Most of concepts discussed in the course can be casted as modules

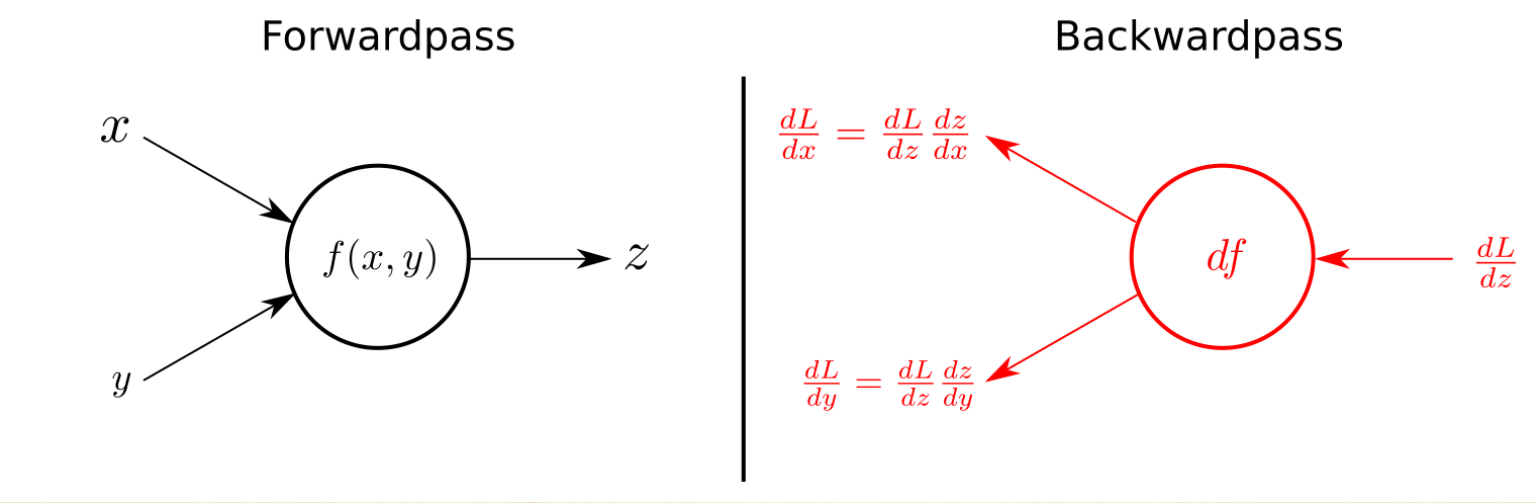
Pooling


Max(1, 1, 5, 6) = 6
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
Rectified Feature Map

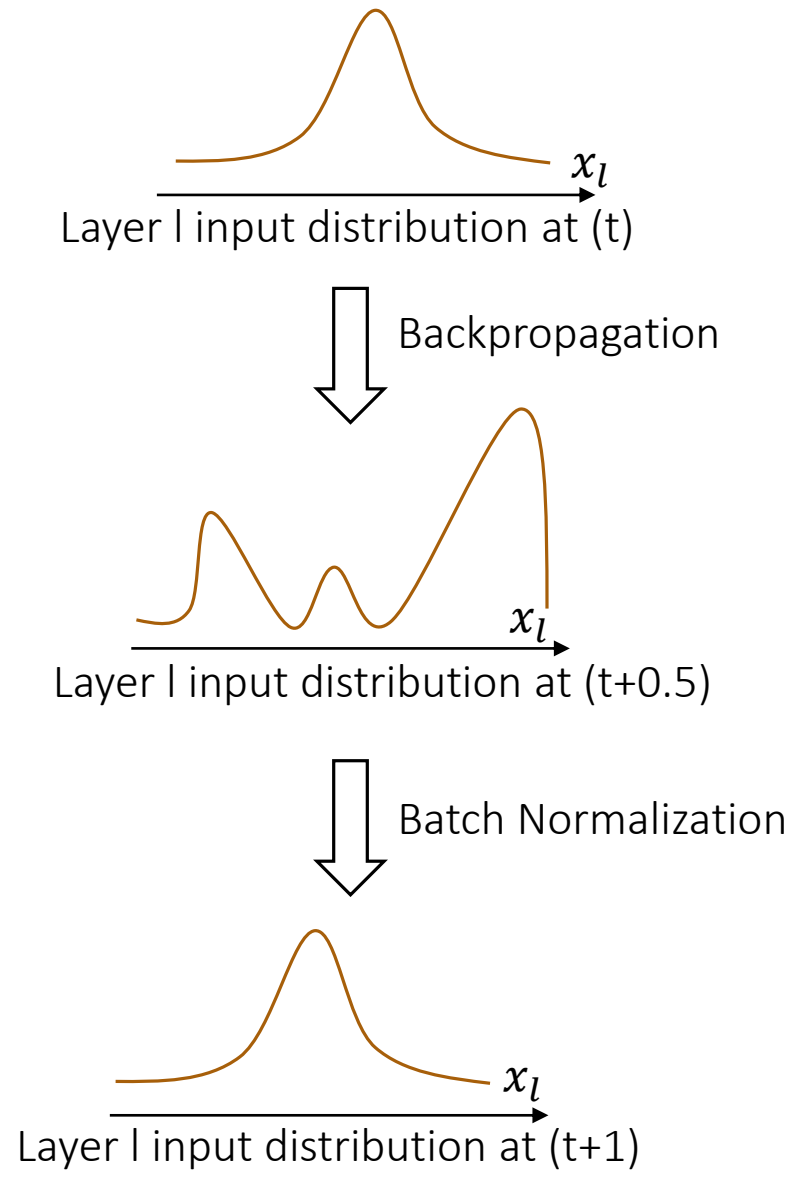
Training or learning phase

- Activation function **f** and **bias** (threshold) are fixed
- Training on both Positive examples and Negative examples to obtain **all weigh** parameters
- 

Training or learning phase



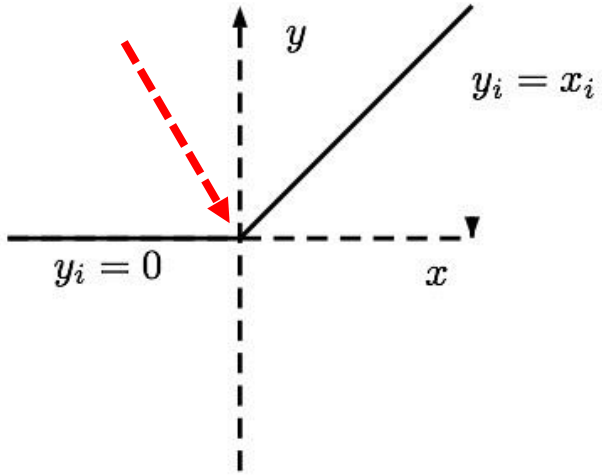
Input normalization



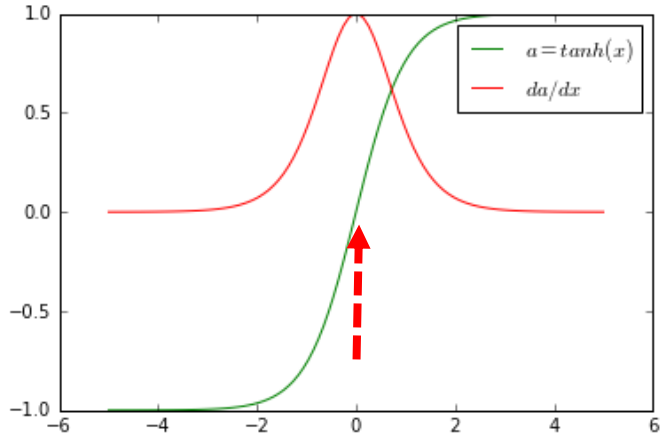
Data pre-processing

- Center data to be roughly 0
  - Activation functions usually “centered” around 0
  - Convergence usually faster
  - Otherwise **maybe** bias on gradient direction → might slow down learning

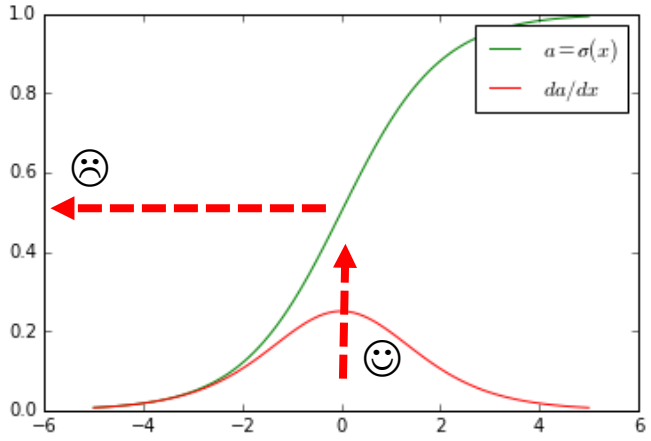
ReLU ☺



$\tanh(x)$ ☺



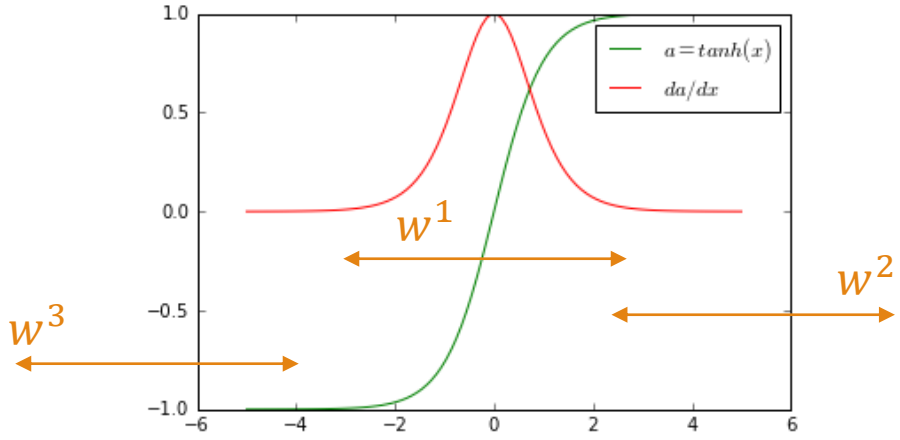
$\sigma(x)$ ☹



Data pre-processing

- Scale input variables to have similar diagonal covariances $c_i = \sum_j (x_i^{(j)})^2$
  - Similar covariances → more balanced rate of learning for different weights
  - Rescaling to 1 is a good choice, unless some dimensions are less important

$x = [x_1, x_2, x_3]^T, w = [w_1, w_2, w_3]^T, a = \tanh(w^{\mathrm{T}} x)$



x_1, x_2, x_3 → much different covariances

↓

Generated gradients $\left.\frac{d\mathcal{L}}{d\theta}\right|_{x_1,x_2,x_3}$: much different

↓

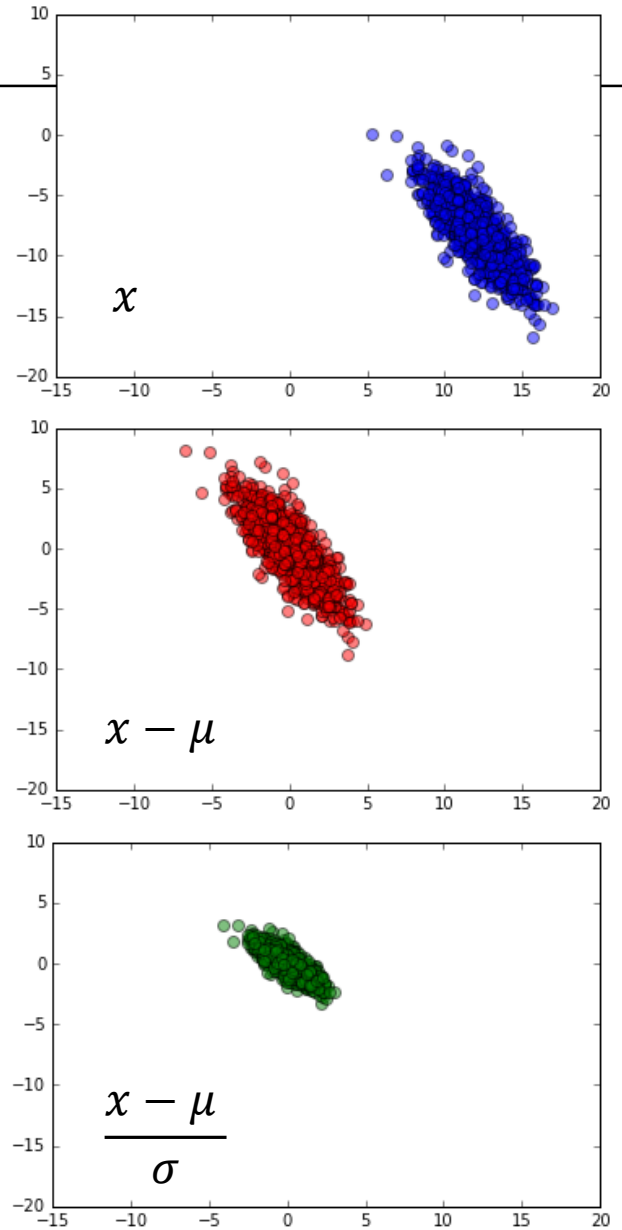
Gradient update harder: $w_{t+1} = w_t - \eta_t \begin{bmatrix} d\mathcal{L}/dw_1 \\ d\mathcal{L}/dw_2 \\ d\mathcal{L}/dw_3 \end{bmatrix}$

Data pre-processing

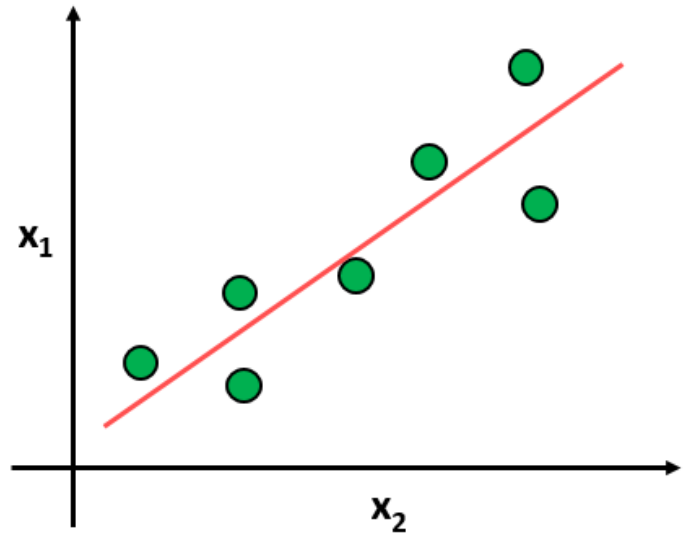
- Input variables should be as decorrelated as possible
  - Input variables are “more independent”
  - Model is forced to find non-trivial correlations between inputs
  - Decorrelated inputs → Better optimization
- Extreme case
  - extreme correlation (linear dependency) might cause problems [CAUTION]
- Obviously decorrelating inputs is not good when inputs are by definition correlated, like when in sequences

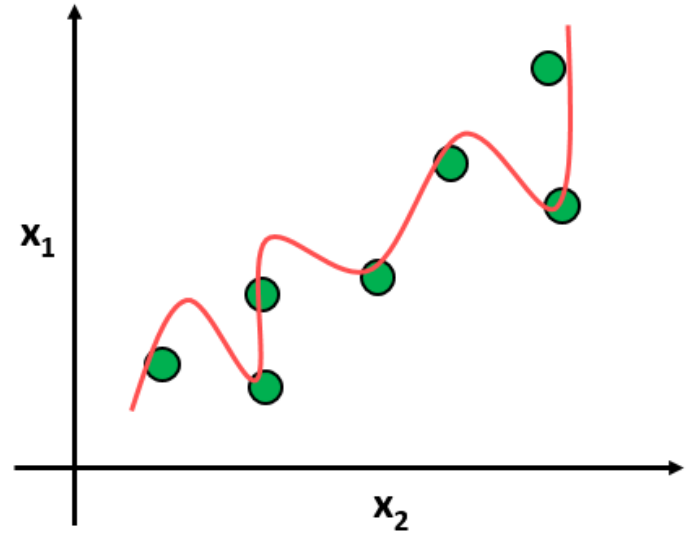
Unit Normalization: $N(\mu, \sigma^2) = N(0, 1)$

- Input variables follow a Gaussian distribution (roughly)
- In practice:
  - from training set compute mean and standard deviation
  - Then subtract the mean from training samples
  - Then divide the result by the standard deviation



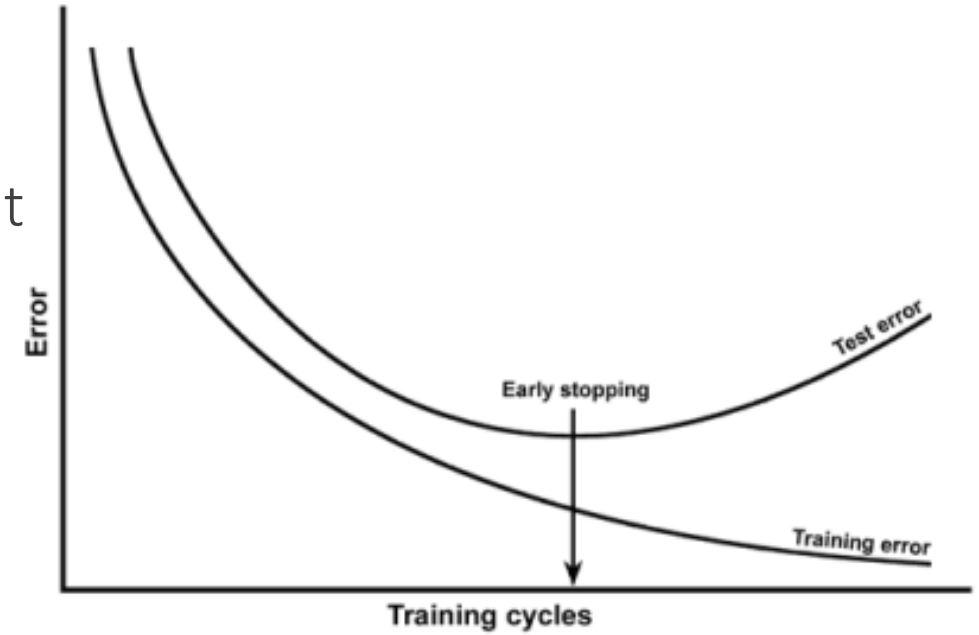
Regularization





Early stopping

- To tackle overfitting another popular technique is early stopping
- Monitor performance on a separate validation set
- Training the network will decrease training error, as well validation error (although with a slower rate usually)
- Stop when validation error starts increasing
  - This quite likely means the network starts to overfit

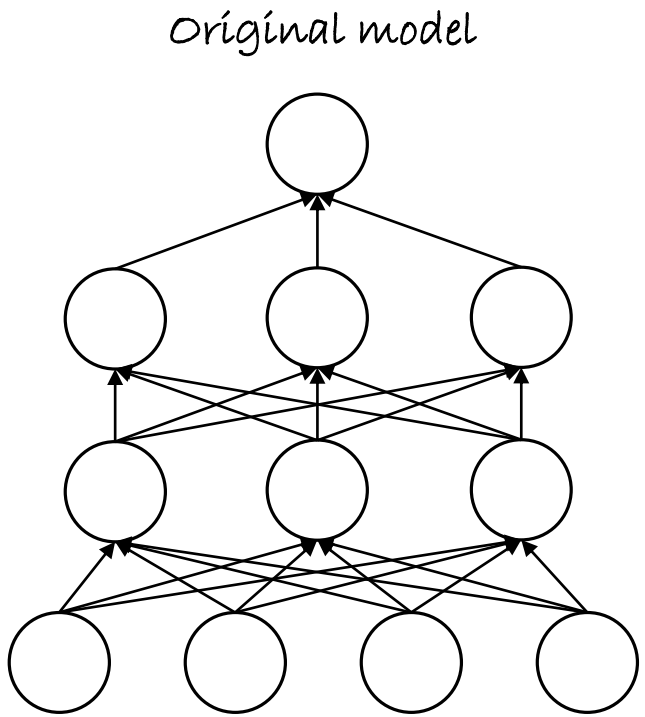


Dropout [Srivastava2014]

- During training setting activations randomly to 0
  - Neurons sampled at random from a Bernoulli distribution with $p = 0.5$
- At test time all neurons are used
  - Neuron activations reweighted by p
- Benefits
  - Reduces complex co-adaptations or co-dependencies between neurons
  - No “free-rider” neurons that rely on others
  - Every neuron becomes more robust
  - Decreases significantly overfitting
  - Improves significantly training speed

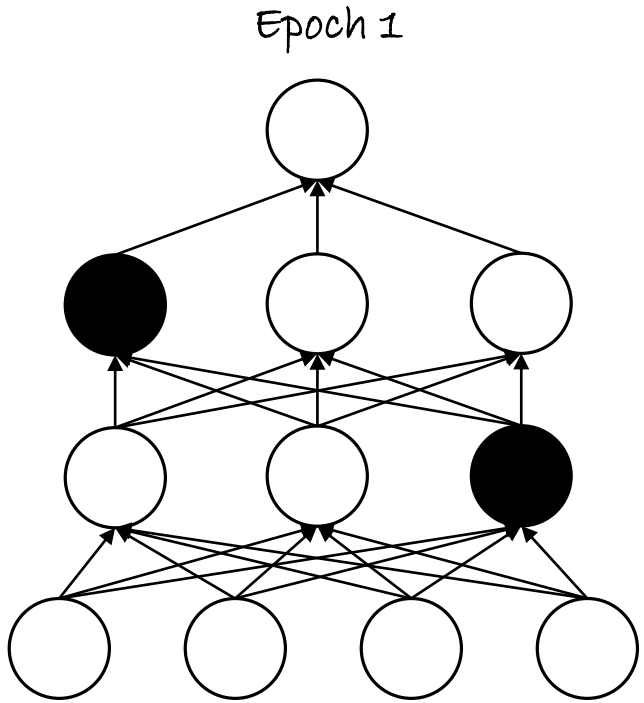
Dropout

- Effectively, a different architecture at every training epoch
  - Similar to model ensembles



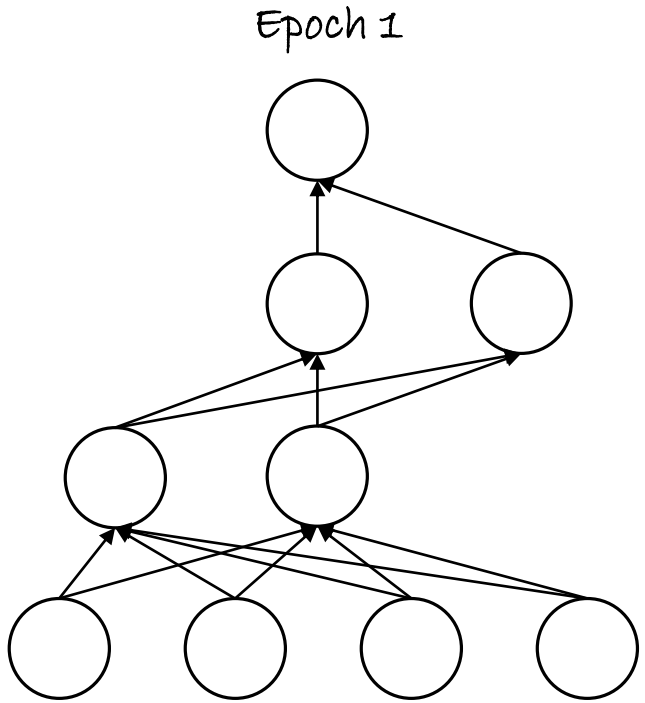
Dropout

- Effectively, a different architecture at every training epoch
  - Similar to model ensembles

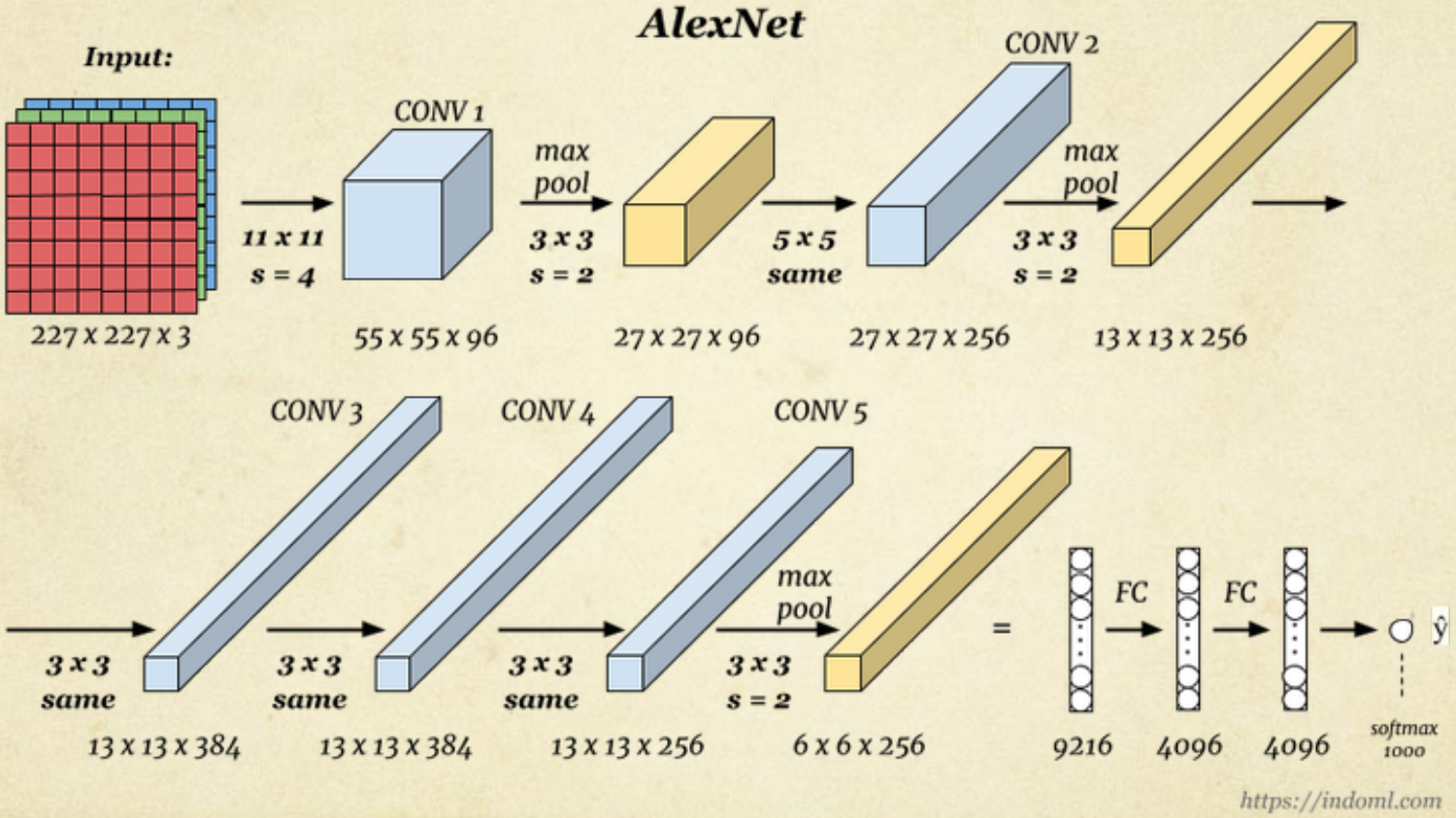


Dropout

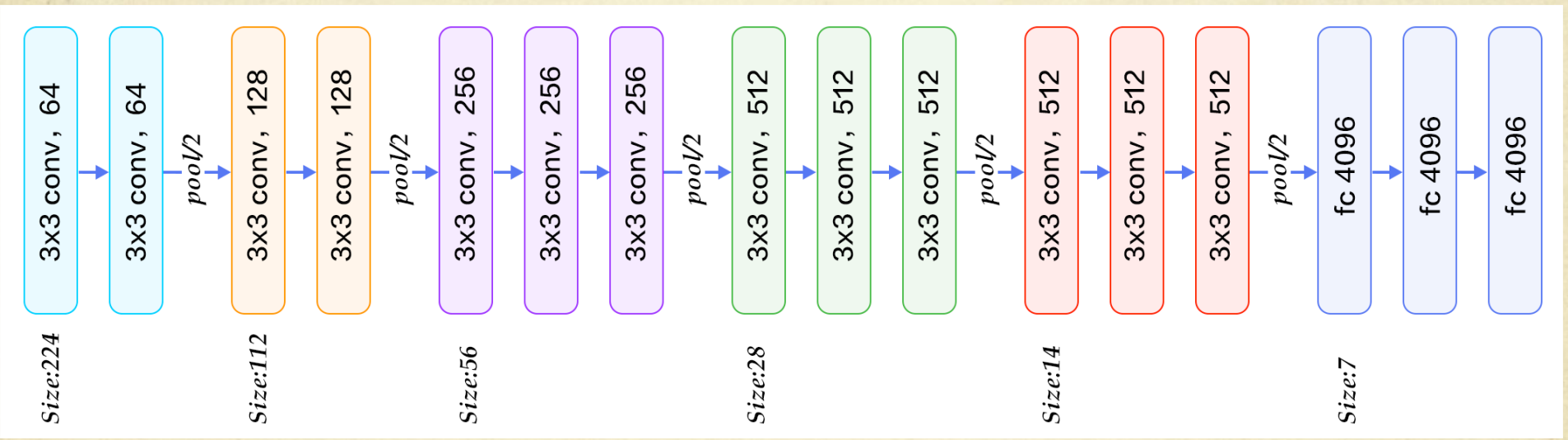
- Effectively, a different architecture at every training epoch
  - Similar to model ensembles



A CNN structure: AlexNet

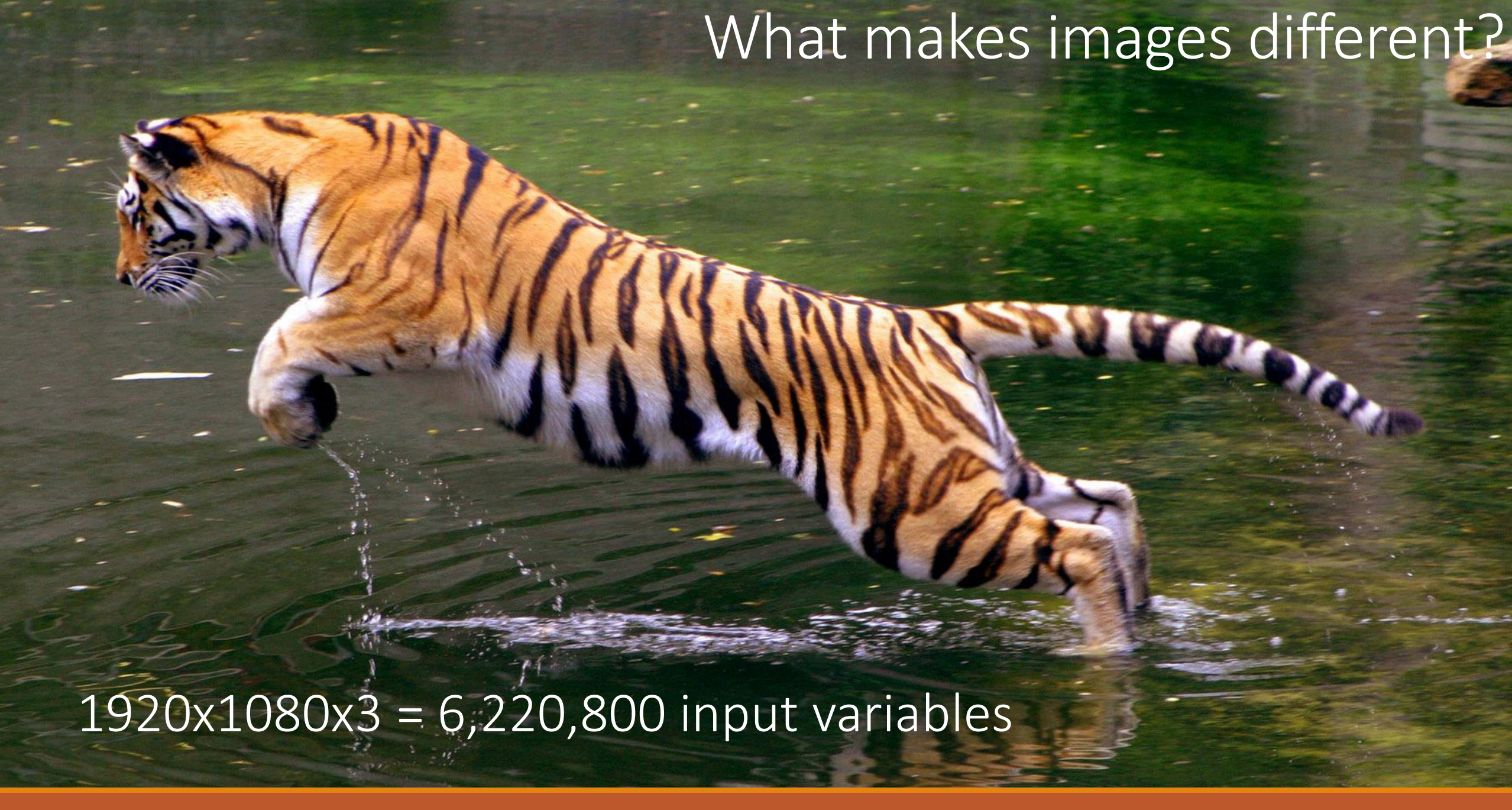


A CNN structure:VGG19



Understanding convnets

What makes images different?



1920x1080x3 = 6,220,800 input variables


What makes images different?
Image has shifted a bit to the up and the left!

What makes images different?

- An image has spatial structure
- Huge dimensionality
  - A 256x256 RGB image amounts to ~200K input variables
  - 1-layered NN with 1,000 neurons → 200 million parameters
- Images are stationary signals → they share features
  - After variances images are still meaningful
  - Small visual changes (often invisible to naked eye) → big changes to input vector
  - Still, semantics remain
  - Basic natural image statistics are the same

Input dimensions are correlated

Traditional task: Predict my salary!

Shift 1 dimension

Level of education	Age	Years of experience	Previous job	Nationality
"Higher"	28	6	Researcher	Spain

Level of education	Age	Years of experience	Previous job	Nationality
Spain	"Higher"	28	6	Researcher

Vision task: Predict the picture!

First 5x5 values

```
array([[51, 49, 51, 56, 55],
       [53, 53, 57, 61, 62],
       [67, 68, 71, 74, 75],
       [76, 77, 79, 82, 80],
       [71, 73, 76, 75, 75]], dtype=uint8)
```

First 5x5 values

```
array([[58, 57, 57, 59, 59],
       [58, 57, 57, 58, 59],
       [59, 58, 58, 58, 58],
       [61, 61, 60, 60, 59],
       [64, 63, 62, 61, 60]], dtype=uint8)
```

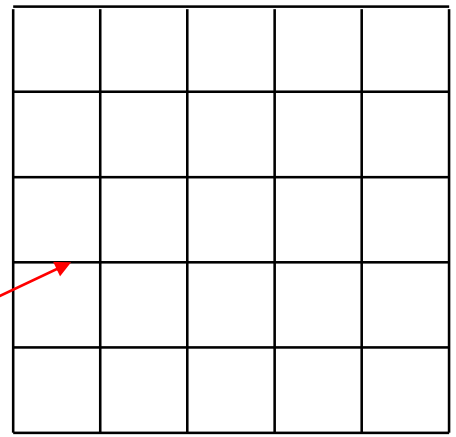
Interesting questions

- What do the image activations in different layers look like?
- What types of images create the strongest activations?
- What are the activations for the class “ostrich”?
- Do the activations occur in meaningful positions?

How large filters?

- Traditionally, medium sized filters (smaller than 11×11)
- Modern architectures prefer small filter sizes (e.g. 3×3)
- We lose frequency resolution
- Fewer parameters to train
- Deeper networks of cascade filters
  - Still, the same output dimensionalities

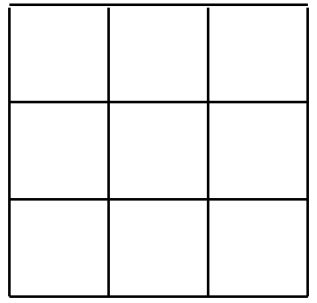
$(2d+1)^2$



For stride 1 the first feature map has dimensionality $\frac{H-2d-1}{1}+1 = H-2d$

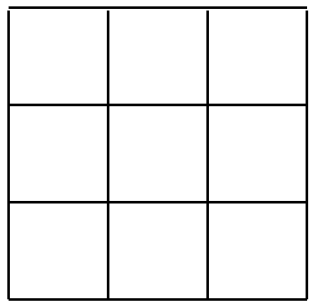
$vs.$

$(d+1)^2$



$*$

$(d+1)^2$

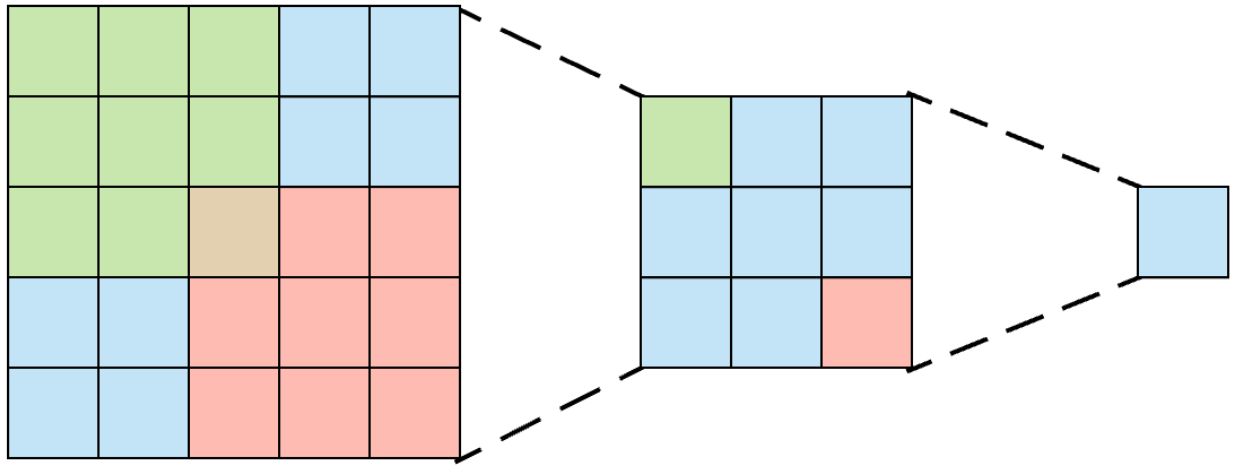


For stride 1 the first feature map has dimensionality $H-d$, the second image $\frac{H-d-d-1}{1}+1 = H-2d$

$(Layer\ l) * (Layer\ l+1)$

Why 3×3 filters?

- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of …



Picture credit: Arden Dertat

Why 3×3 filters?

- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?

Feature maps

- Convolution activations == feature maps
- A deep network has several hierarchical layers
  - hence several hierarchical feature maps going from less to more abstract

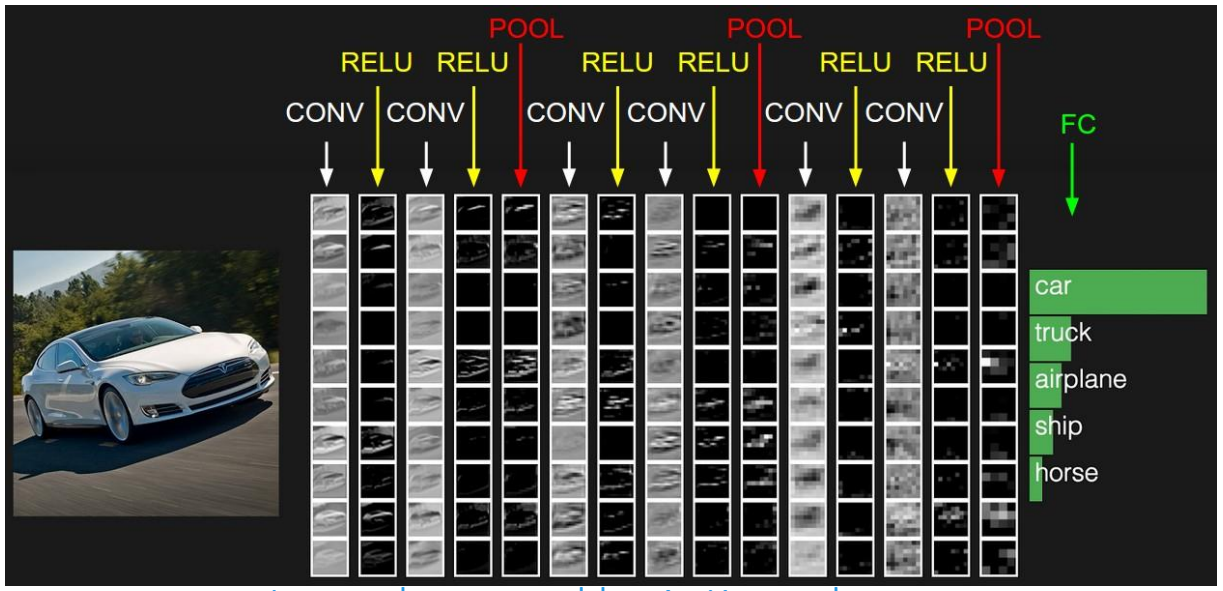


Image borrowed by A. Karpathy

Why 3×3 filters?

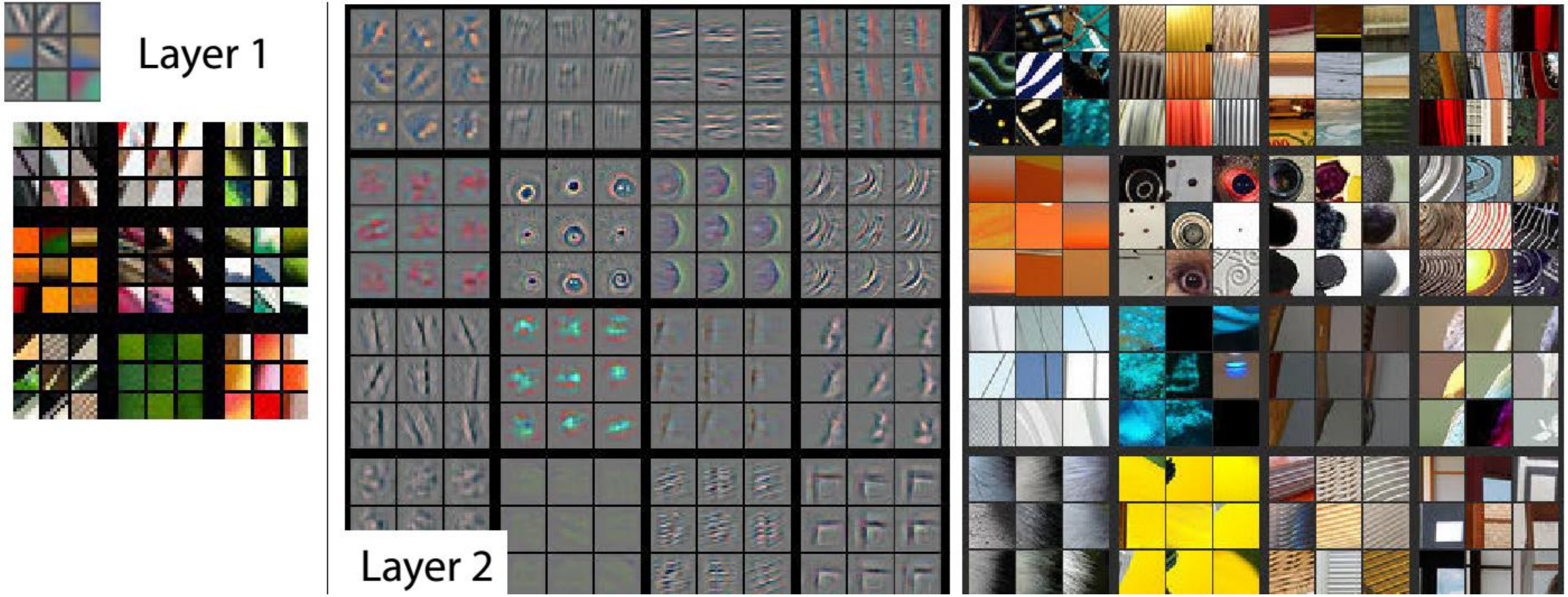
- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?
- Three more nonlinearities for the same “size” of pattern learning
- Also fewer parameters and regularization

$$(3 \times 3 \times C) \times 3 = 27 \cdot C, 7 \times 7 \times C \times 1 = 49 \cdot C$$

- **Conclusion: 1 large filter can be replaced by a deeper, potentially more powerful, stack of successive smaller filters**

What excites feature maps?

- *"Given a random feature map what are the top 9 activations?"*



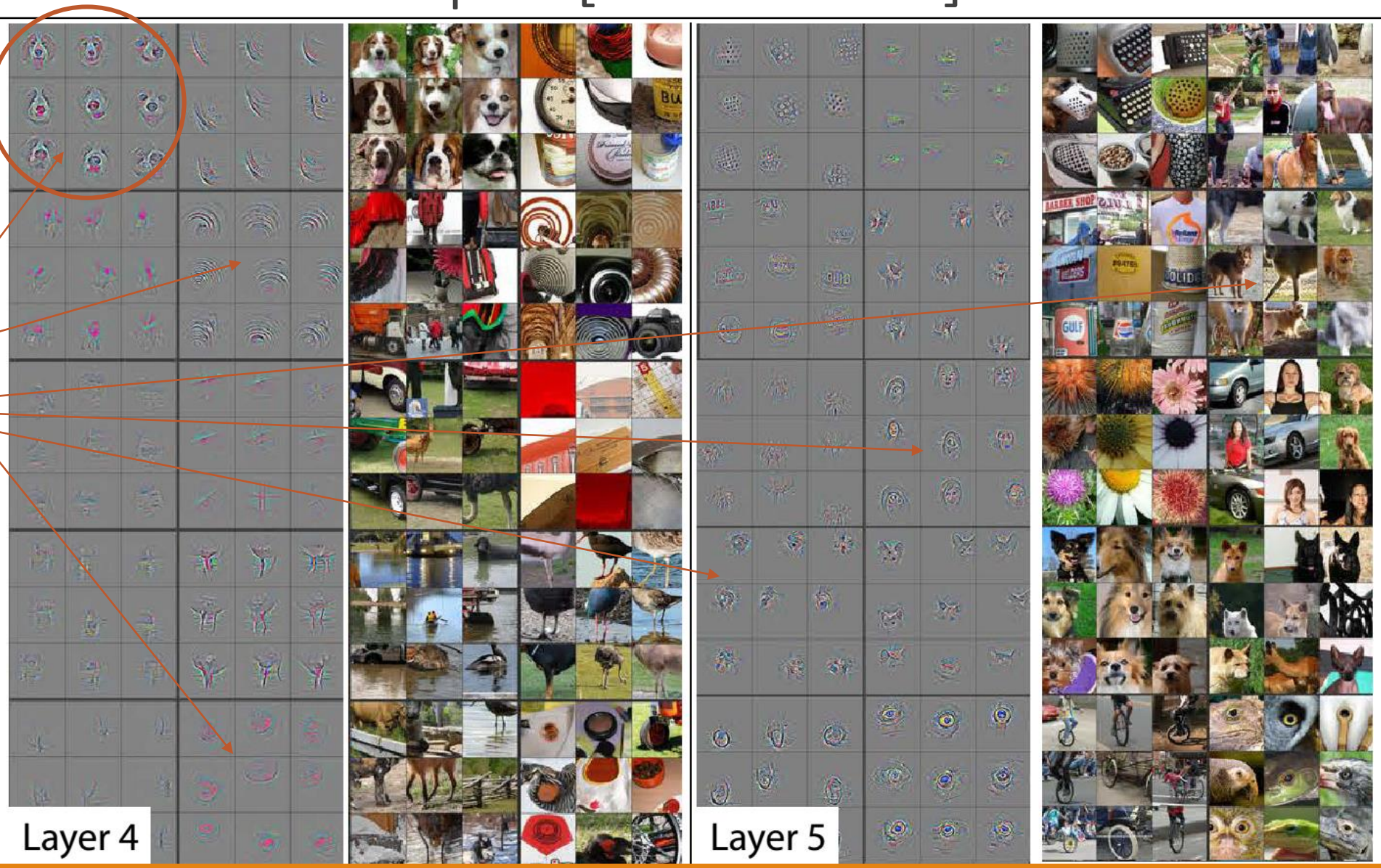
What excites feature maps?

Similar activations from lower level visual patterns

What excites feature maps? [Zeiler2014]

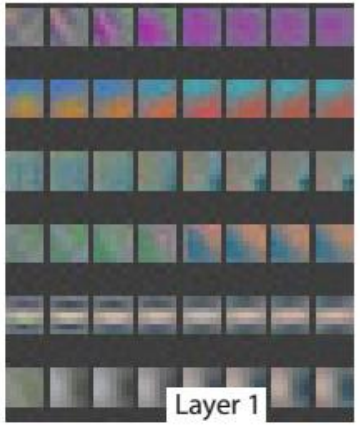
Similar activations from semantically similar pictures

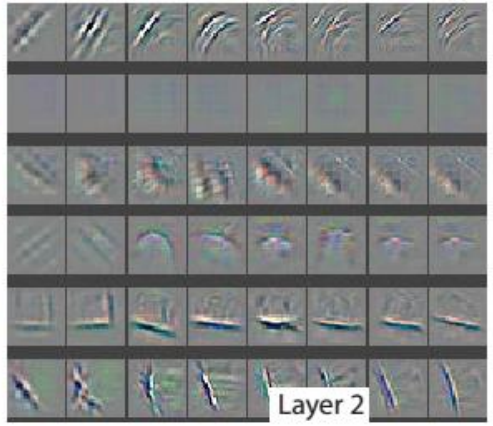
Visual patterns become more and more intricate and specific (greater invariance)

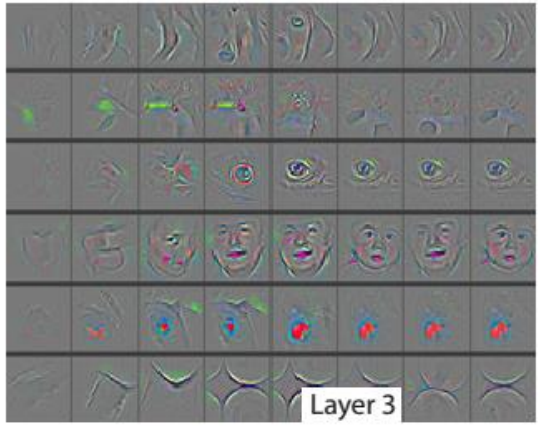


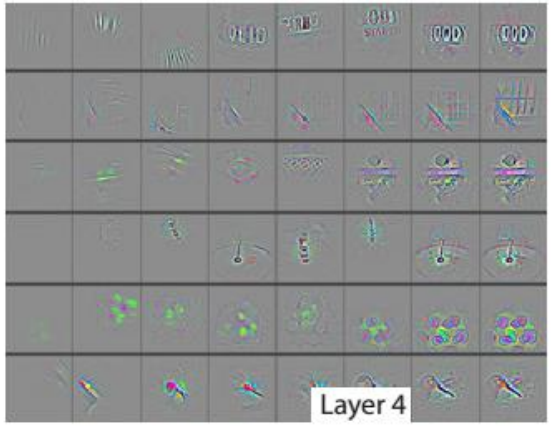
Feature evolution over training

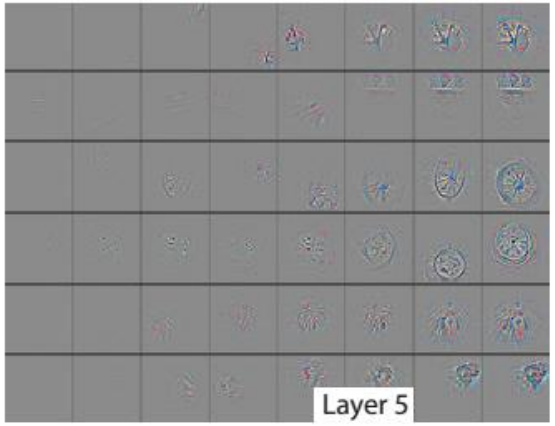
- Given a neuron (outputs a single feature map)
  - Strongest activation during training for epochs 1, 2, 5, 10, 20, 30, 40, 64



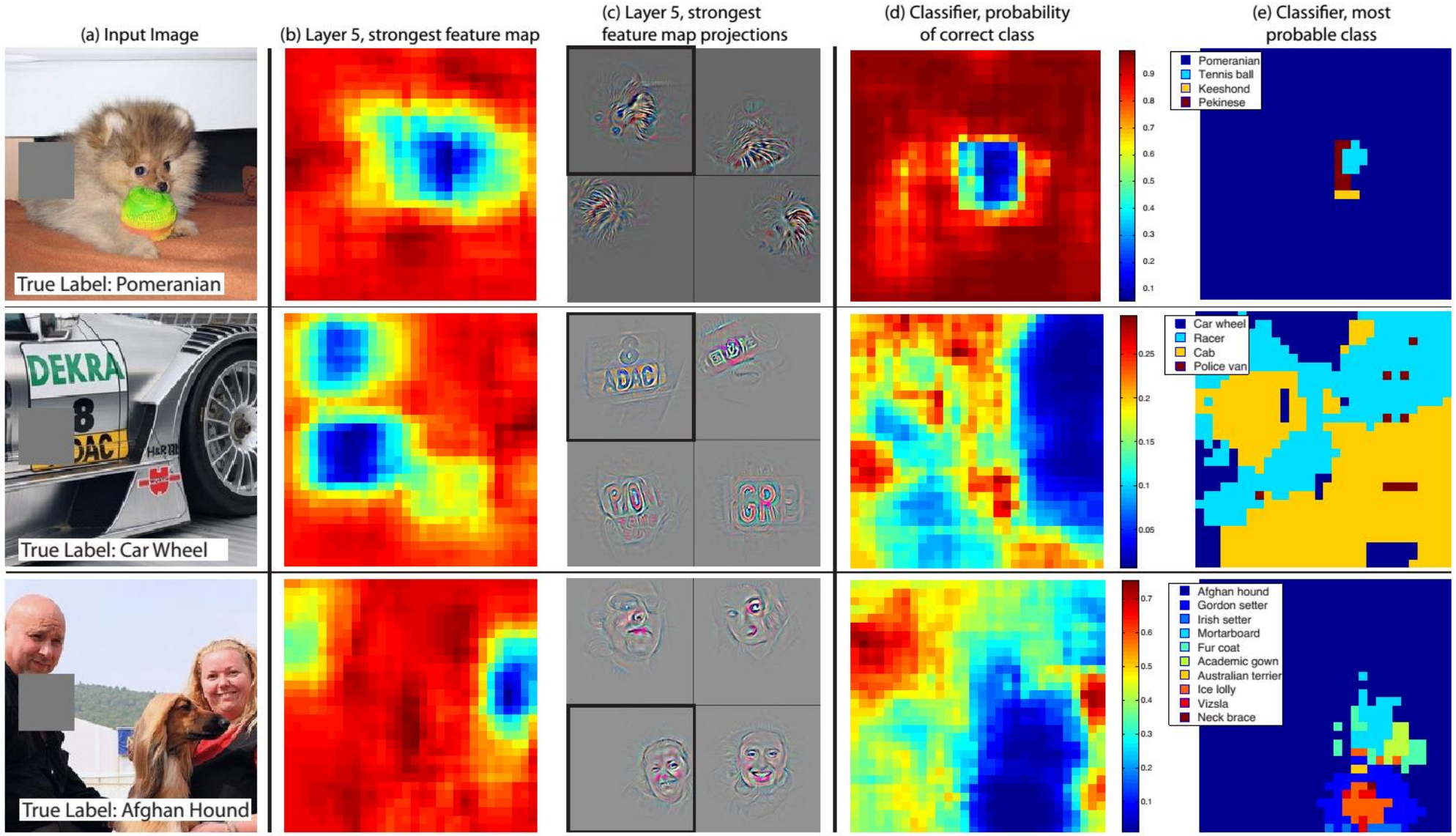




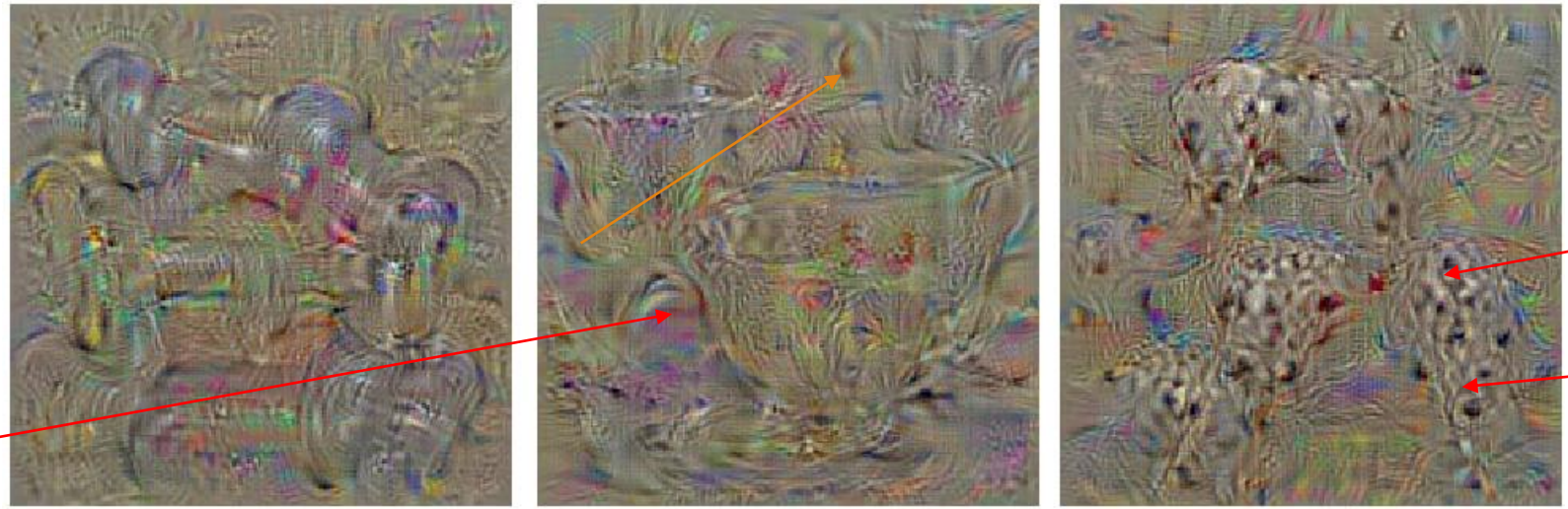


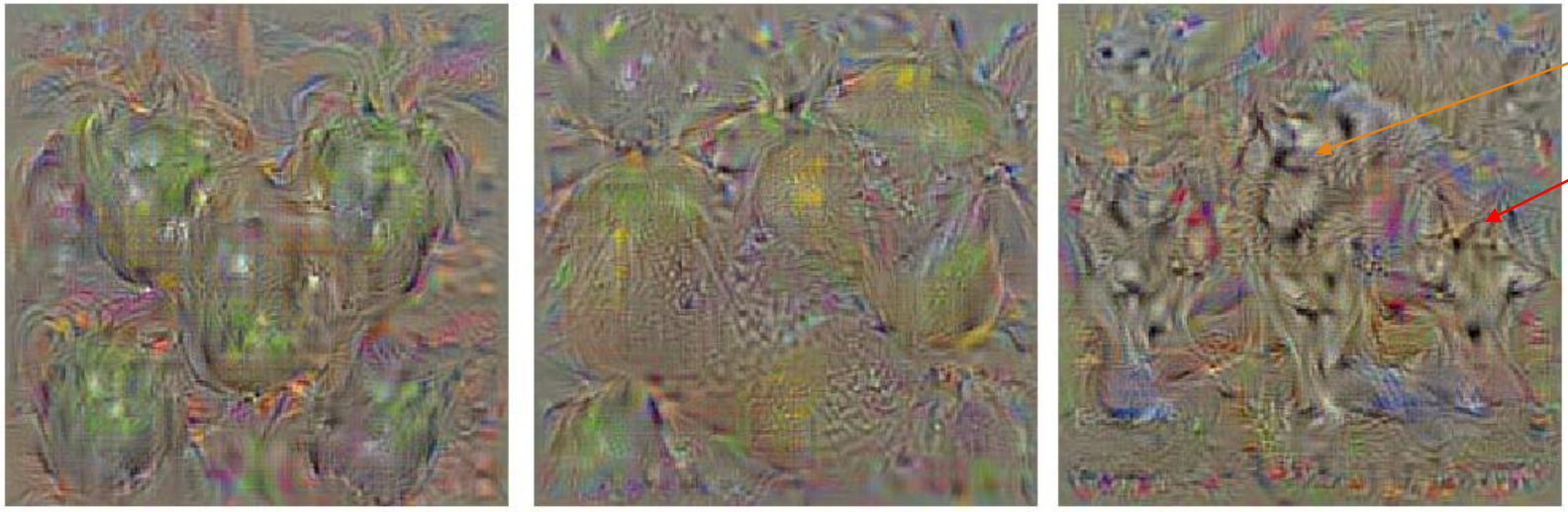


But does a Convnet really learn the object?



What is a “Convnet dog”, however? [Simonyan2014]





Transfer learning

- Assume two datasets, T and S
- Dataset S is
  - fully annotated, plenty of images
  - We can build a model h_S
- Dataset T is
  - Not as much annotated, or much fewer images
  - The annotations of T do not need to overlap with S
- We can use the model h_S to learn a better h_T
- This is called transfer learning

Imagenet: 1million

h_A

My dataset: 1,000

h_B

Why use Transfer Learning?

- A CNN can have millions of parameters
- But our datasets are not always as large
- Could we still train a CNN without overfitting problems?

Loss function

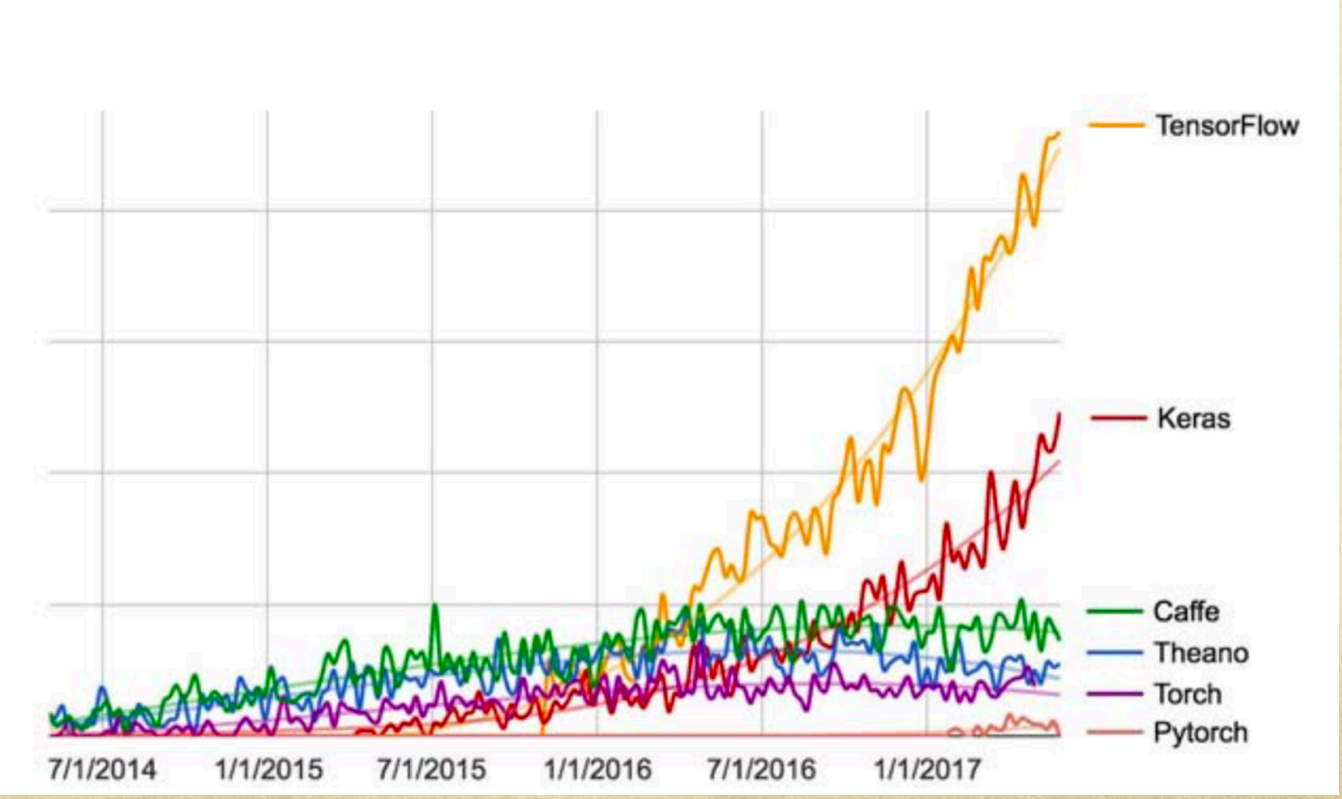
- Logistic function
- Soft-max function
- Triplet-loss function

Working with Deep Network

1. Build a model
2. Pre-pairing dataset
3. Training phase/Testing phase
4. Save model
5. Use the model with new data

Build a model

- There are several libraries to build model
- In course: **Keras**



Build a model

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
>>> model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D) (None, 13, 13, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D) (None, 5, 5, 64)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```