# Deep Learning

Image Classification
Object Detection
Natural Language Modeling

## Agenda I

- Introduction Machine Learning
- Neural Networks (NN)
- Overview Deep Learning frameworks
- Introduction PyTorch
- Introduction Convolutional Neural Networks (CNN)
- LeNet
- Introduction ImageNet Challenge
- Various CNN architectures

## Agenda II

- Object detection
- Faster R-CNN
- Yolo
- Natural Language Modeling
- Recurrent Neural Networks (RNN)
- Long short-term memory (LSTM)

# Introduction Machine Learning I

| Supervised Learning | Unsupervised Learning | Reinforcement Learning |
|---|---|---|
| Linear Regression<br>Support Vector Machine<br>Decision Tree<br>Neural Network | K-means clustering<br>... | Q-learning<br>... |

**Supervised Learning**

1. Goal: find mapping/function from input X to output Y
2. Examples of X/Y pairs are given to train your model

# Introduction Machine Learning II

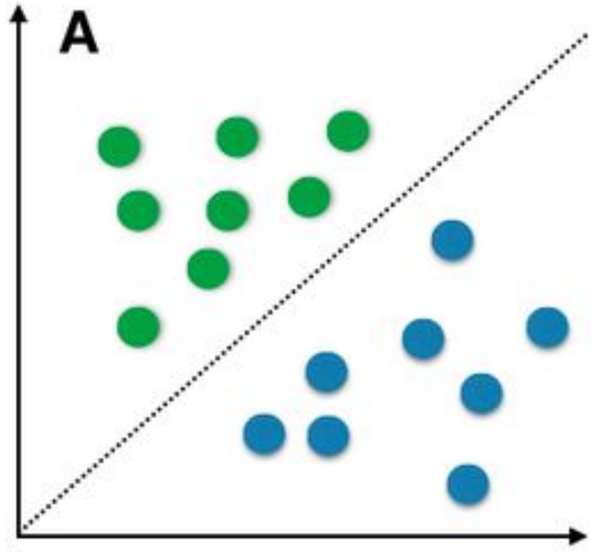
$$\begin{tikzcd} \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \end{pmatrix} = X \arrow[r] & \boxed{\text{Unknown computation}} \arrow[r] & \begin{pmatrix} y1 \\ y2 \end{pmatrix} = Y \end{tikzcd}$$

**Unknown computation**

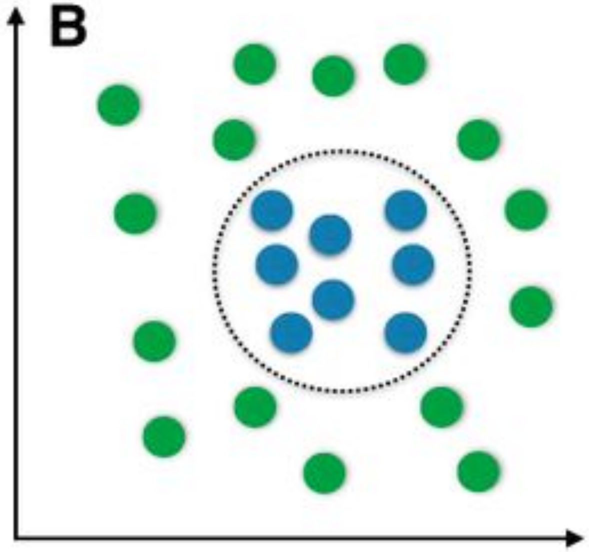
1. Linear: Linear Regression, Linear Classification
2. Non-linear: Decision Tree, Random Forest, Neural Network

more powerful: map complex X/Y pairs

# Introduction Machine Learning III



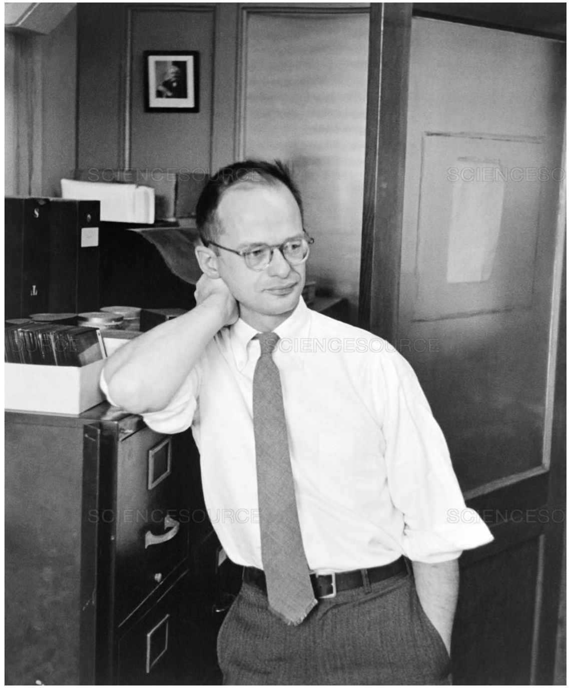
Linear function



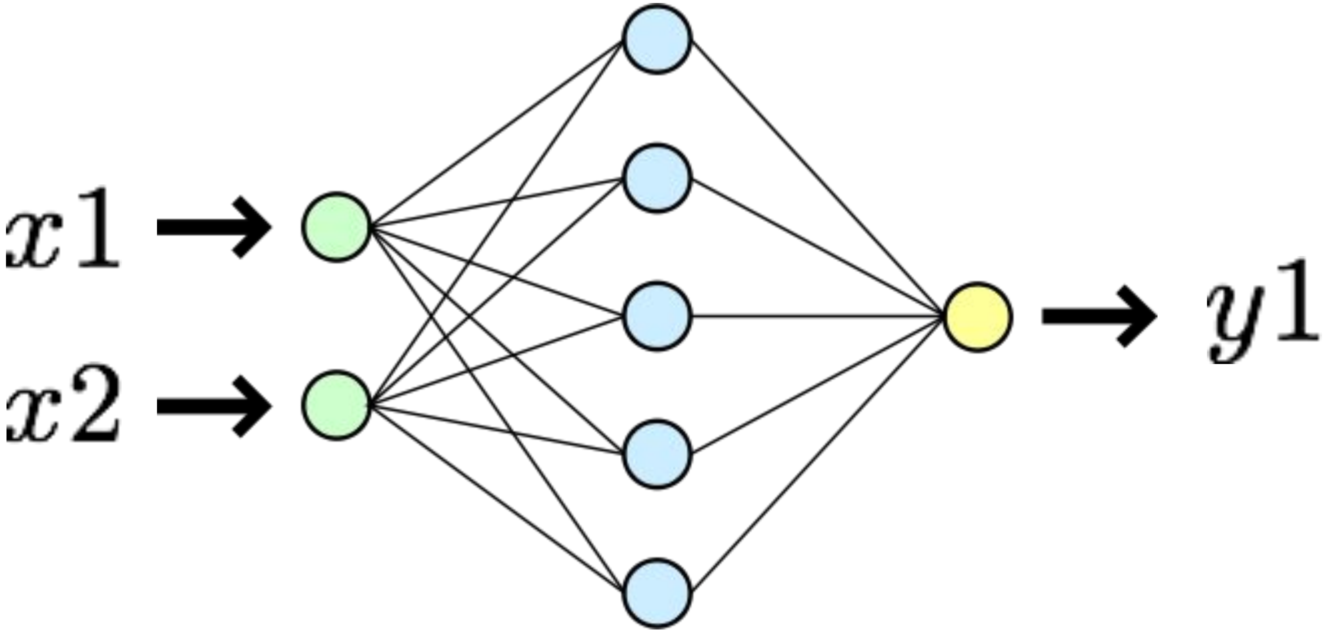
Non-Linear function

## Neural Network: Introduction

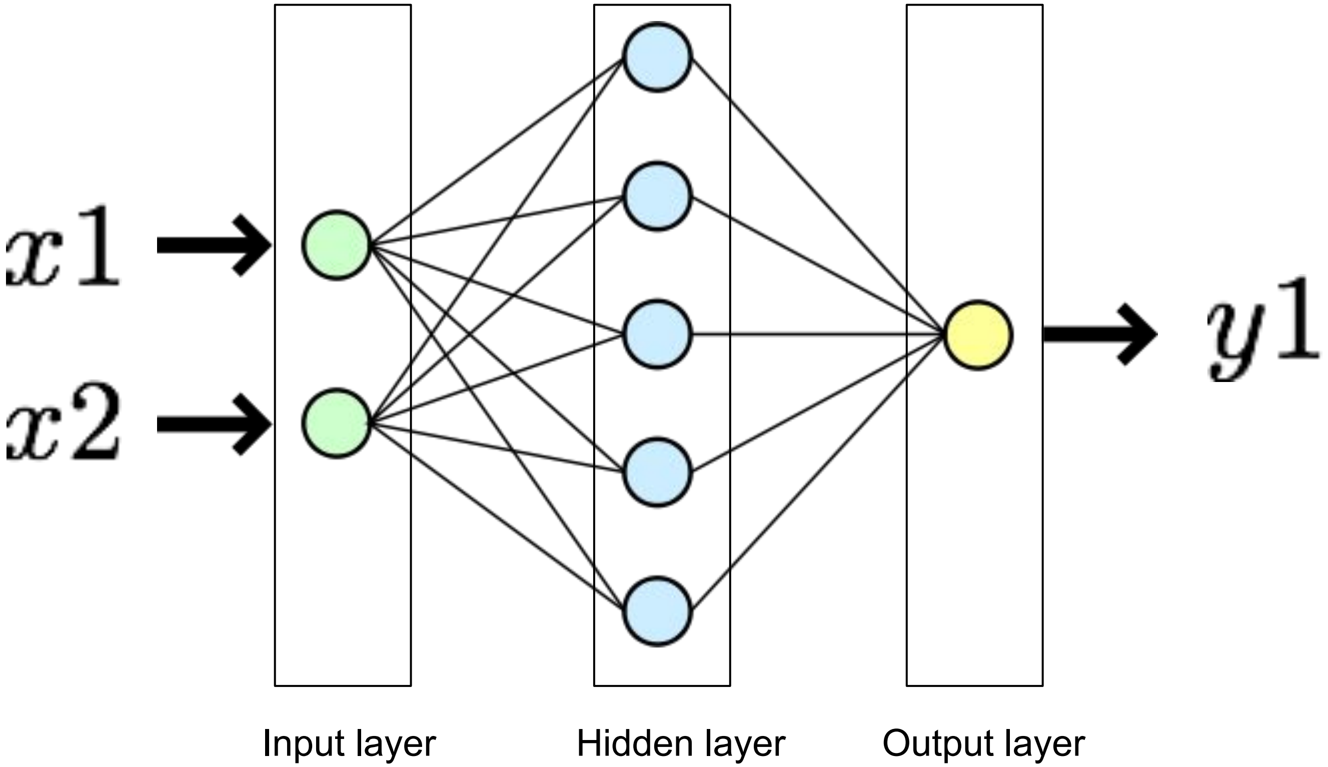
- Non-linear function approximator
- Can approximate any function arbitrarily close
- Classification and regression


Walter Pitts (1943);
Neural Network pioneer

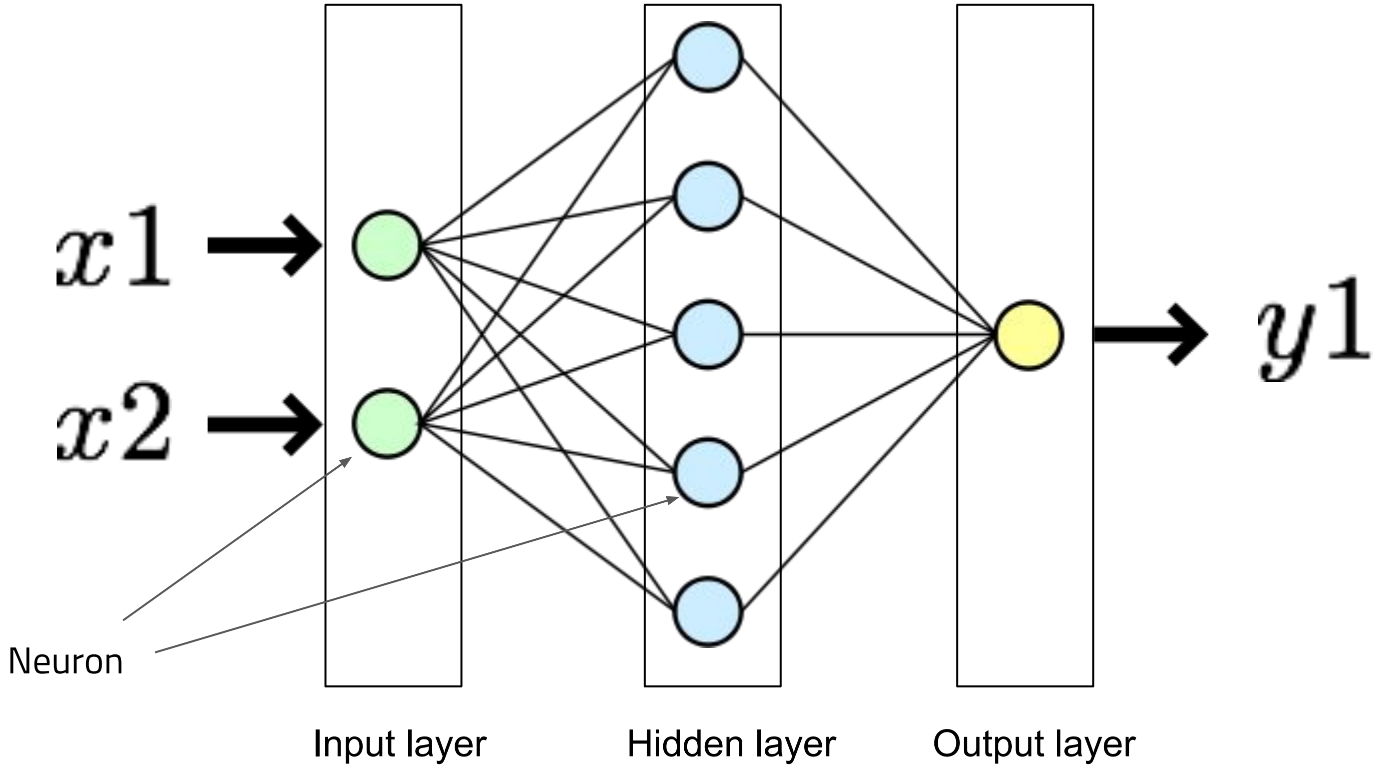
## Neural Network: Building Blocks



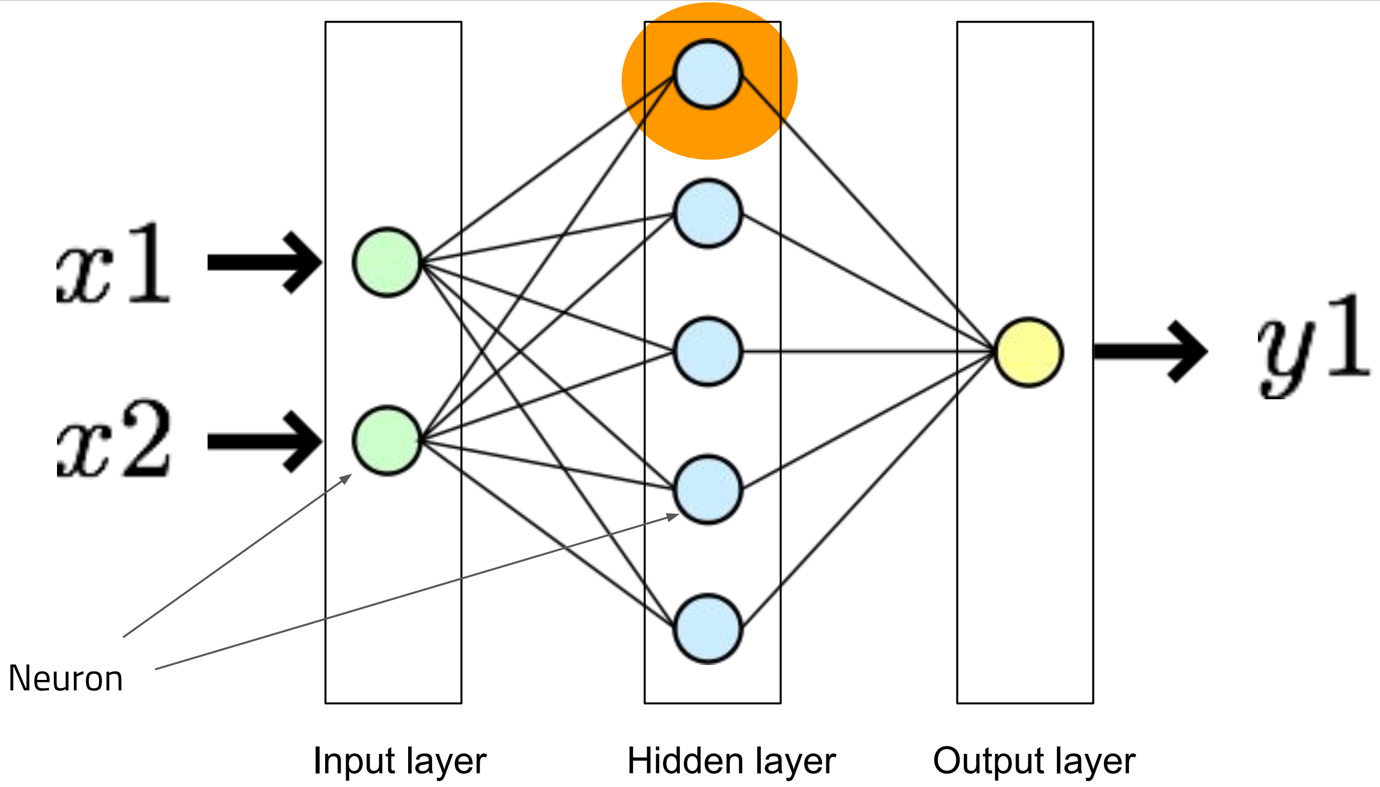
## Neural Network: Building Blocks



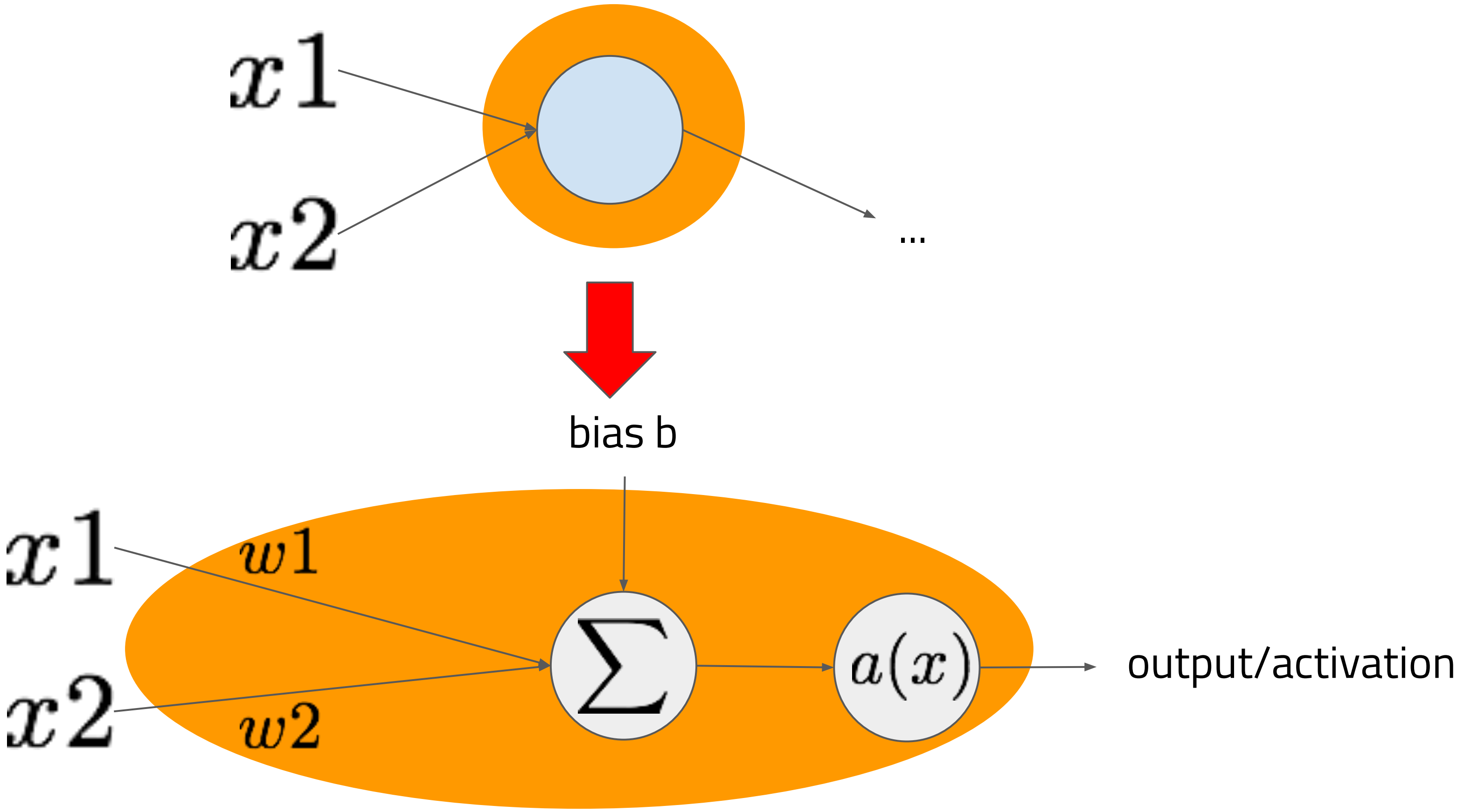
# Neural Network: Building Blocks



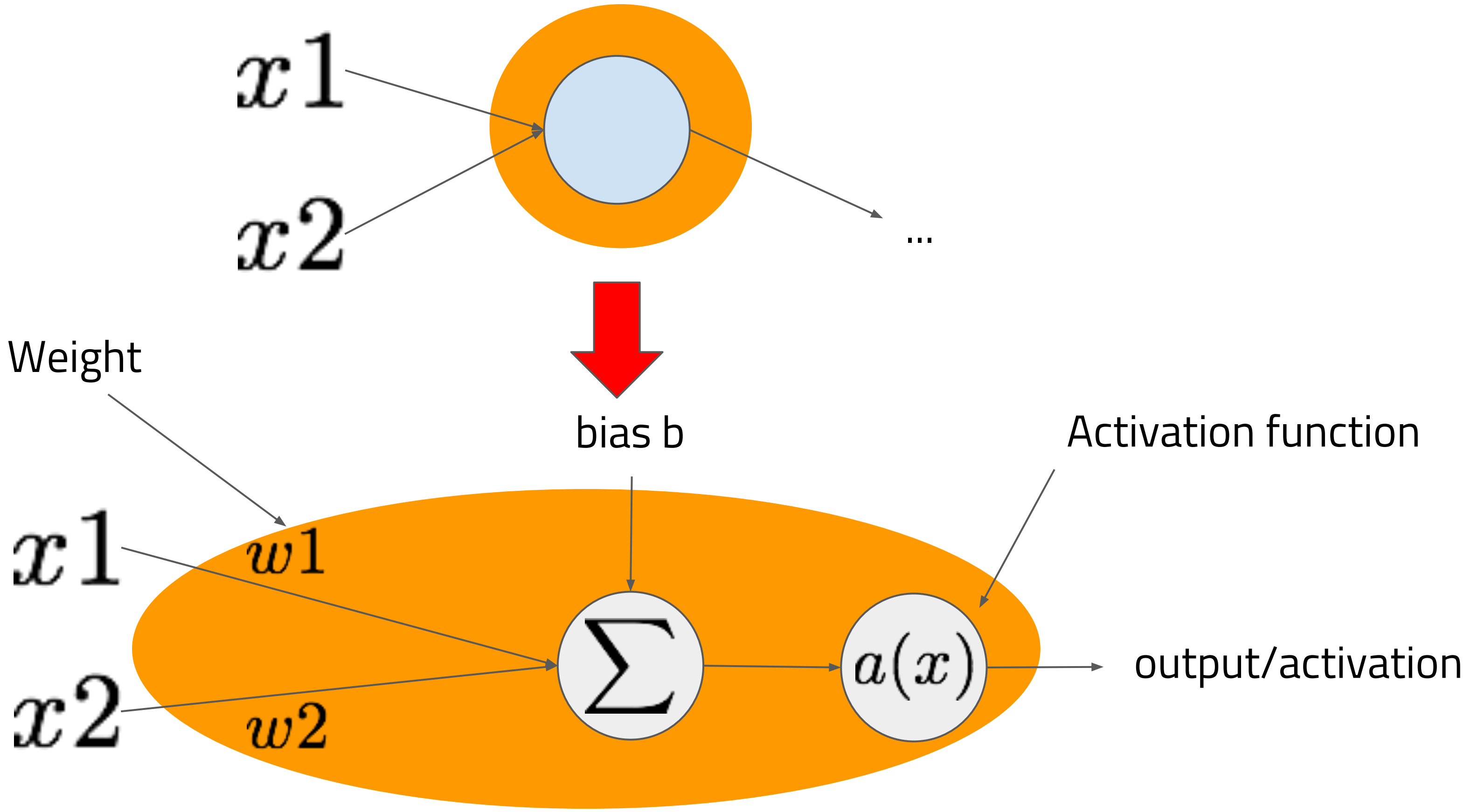
# Neural Network: Building Blocks



## Neural Network: Building Blocks



## Neural Network: Building Blocks



## Neural Network: Universal Approximation Theorem I

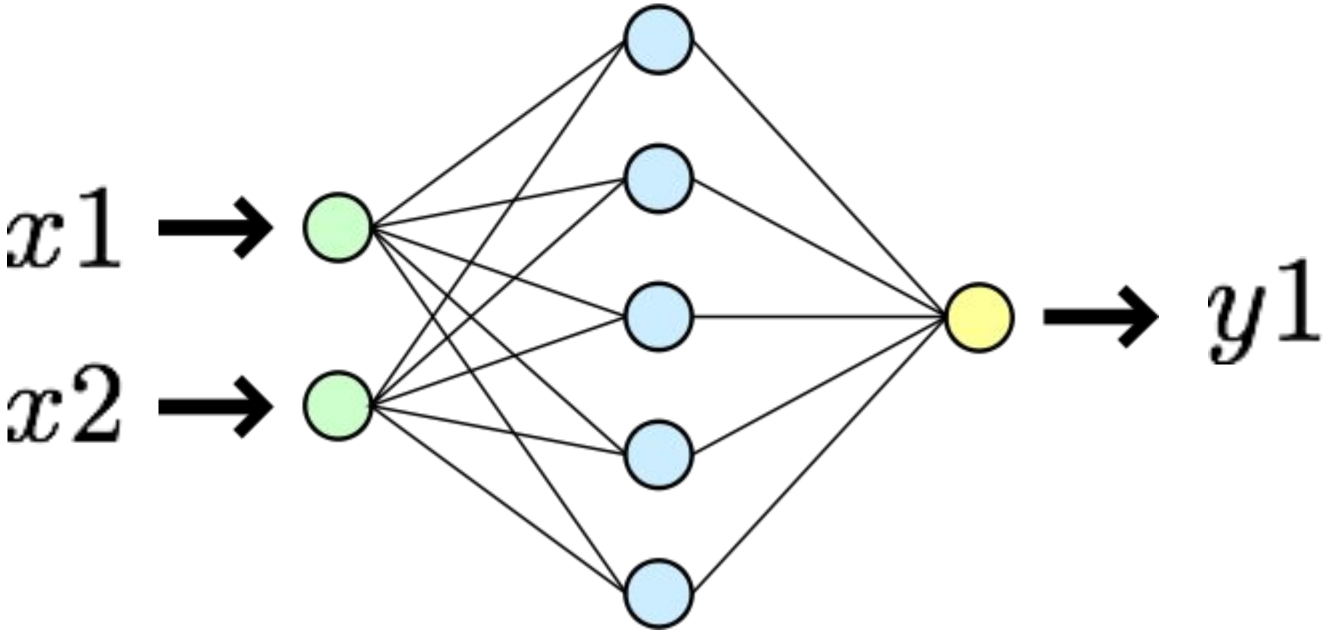
"A feed-forward network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of Rn"

- Wikipedia

➡ one hidden layer with lots of neurons can approximate any natural continuous function

# Neural Network: Universal Approximation Theorem II

$y = \sqrt{x1} + x2 + e^{x1} * x2$ $\qquad$ $y = \sqrt{x1 * x2} + \frac{x2}{x1} + x1 * x2$

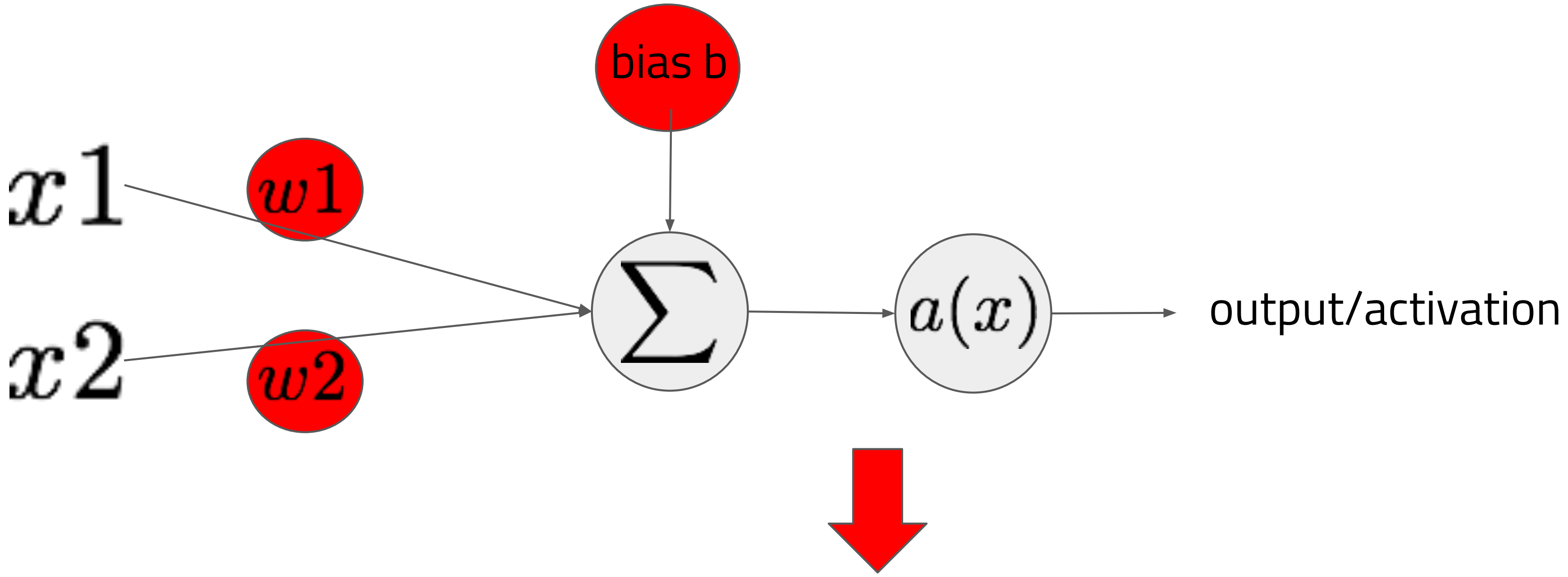


## Neural Network: Universal Approximation Theorem III

A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
**Ian Goodfellow**

We want our function to generalize well thus we will use more complex architectures containing several hidden layers

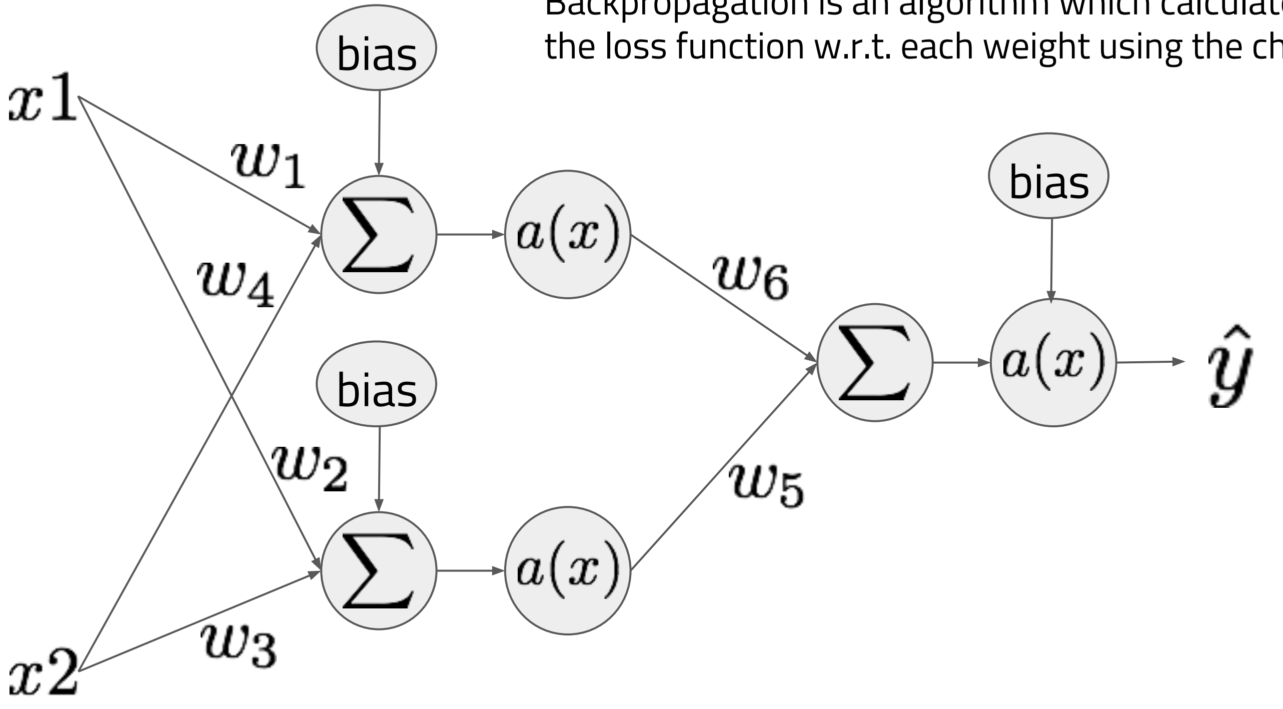
## Neural Network: How to find ideal weights?



Backpropagation

# Neural Network: Backpropagation

Backpropagation is an algorithm which calculates the partial derivatives of the loss function w.r.t. each weight using the chain rule.



$$\text{Loss } L = |\hat{y} - y|$$

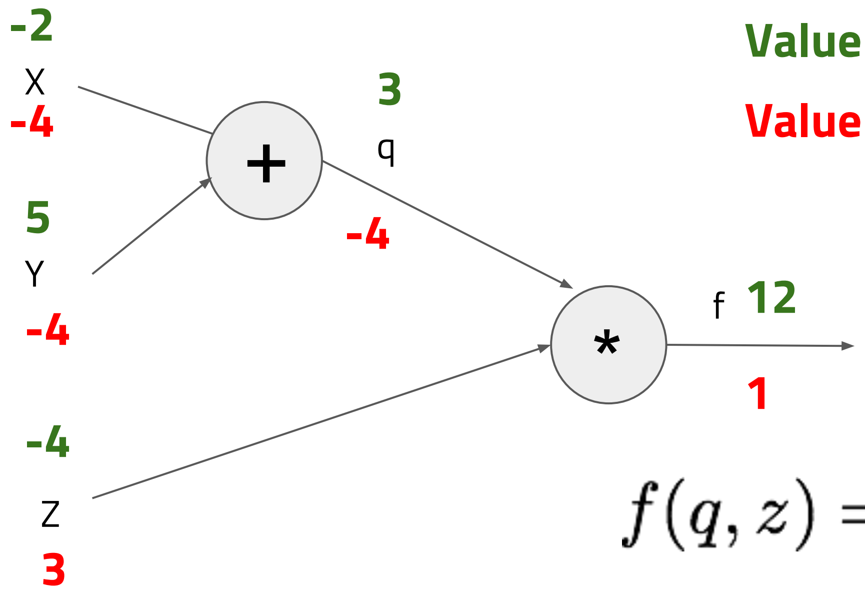
$$\frac{\partial L}{\partial w_1}$$

...

$$\frac{\partial L}{\partial w_6}$$

# Neural Network: Backpropagation

$$\frac{df}{dx} = \frac{df}{dq} * \frac{dq}{dx}$$



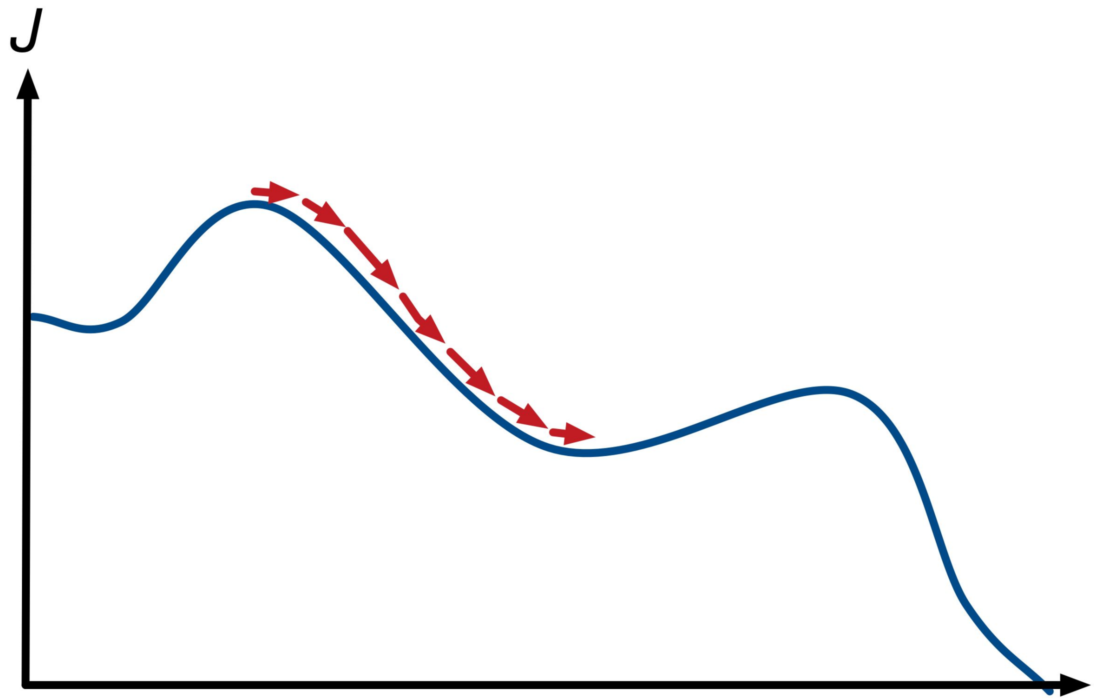
Value

Value of partial derivative

$$f(q, z) = q * z$$

$$q = x + y$$

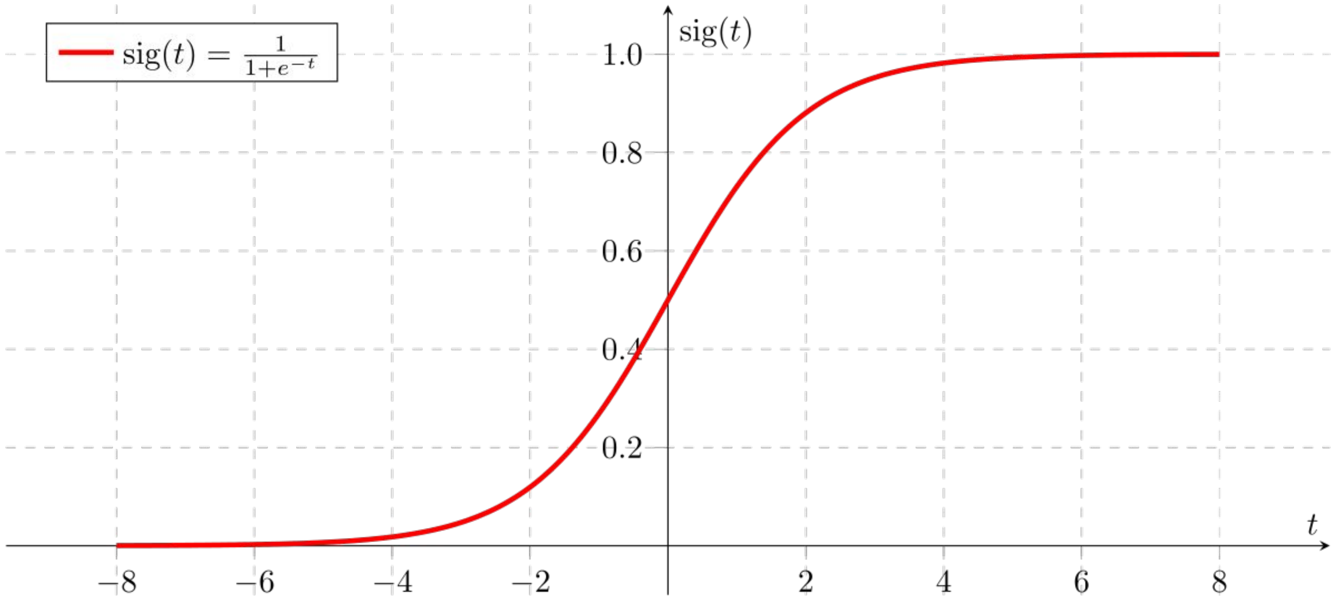
## Neural Network: Gradient Descent



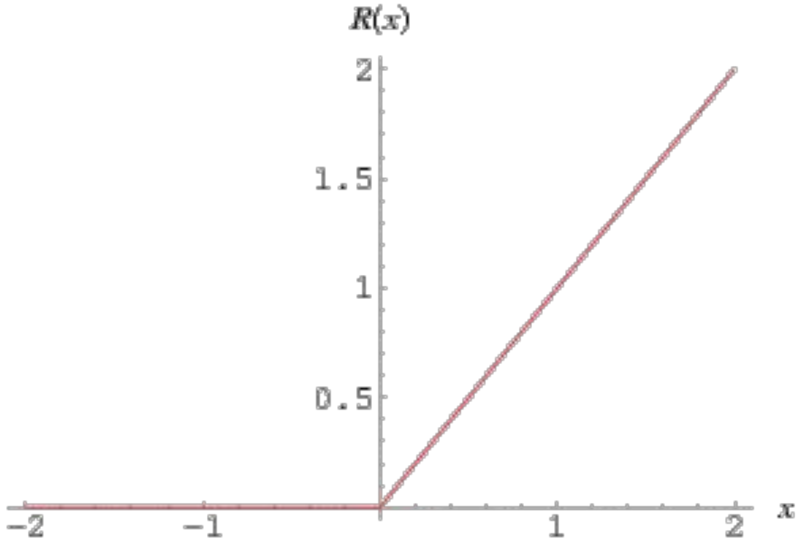
## Neural Network: Different activation functions

- Why do we need non-linear activation functions?
- Hidden Layer:
    - Sigmoid function
    - Rectified Linear Unit (ReLu)
    - Leaky Rectified Linear Unit (Leaky ReLu)
- Output Layer:
    - Softmax function

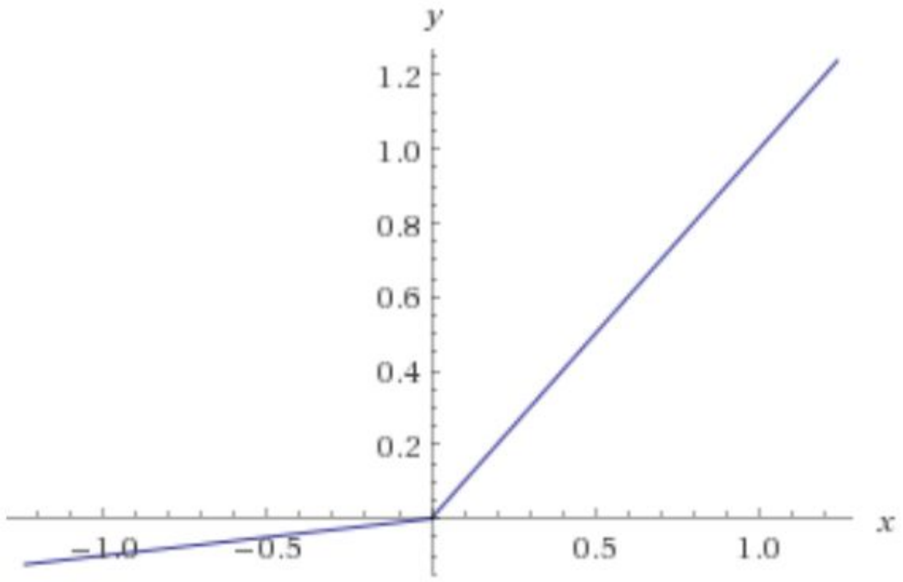
# Neural Network: Sigmoid



## Neural Network: ReLu



## Neural Network: Leaky ReLu



## Overview Deep Learning Frameworks

Microsoft CNTK

TensorFlow™

Caffe2

PYTORCH

K Keras

theano

## Typical Features of Deep Learning frameworks

- Compute gradients of the cost function with respect to the weights **automatically**
- Predefined activation functions
- Predefined cost functions
- Predefined optimizers
- Dataloading utilities
- Option to move all computations to the GPU

## How do Deep Learning frameworks differ?

- Low level vs high level framework
- Static/dynamic computational graph
- More or less user friendly API
- Help from Community
- Speed of adopting new techniques

## Overview Deep Learning Frameworks

Microsoft CNTK

TensorFlow™

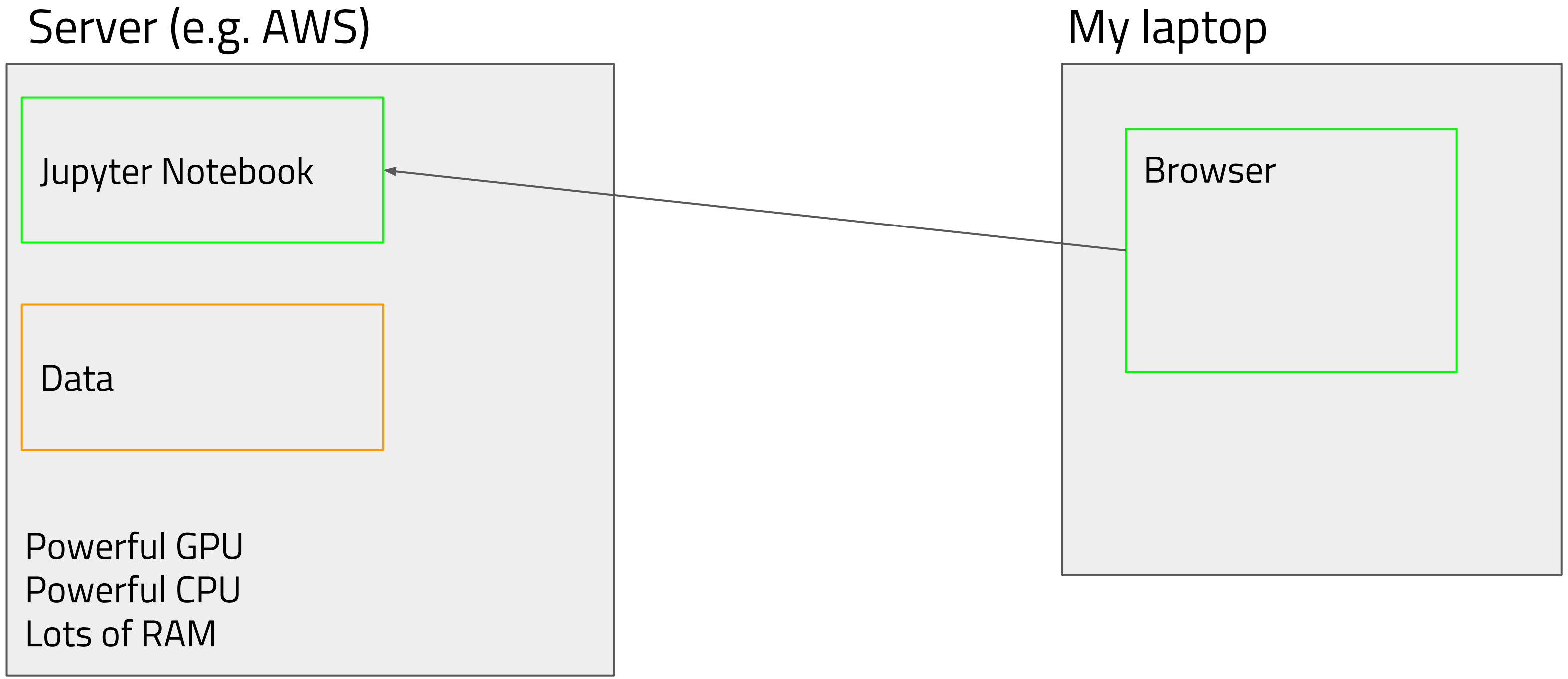
Caffe2

Keras

PYTORCH

theano

# Introduction: Deep Learning in the cloud



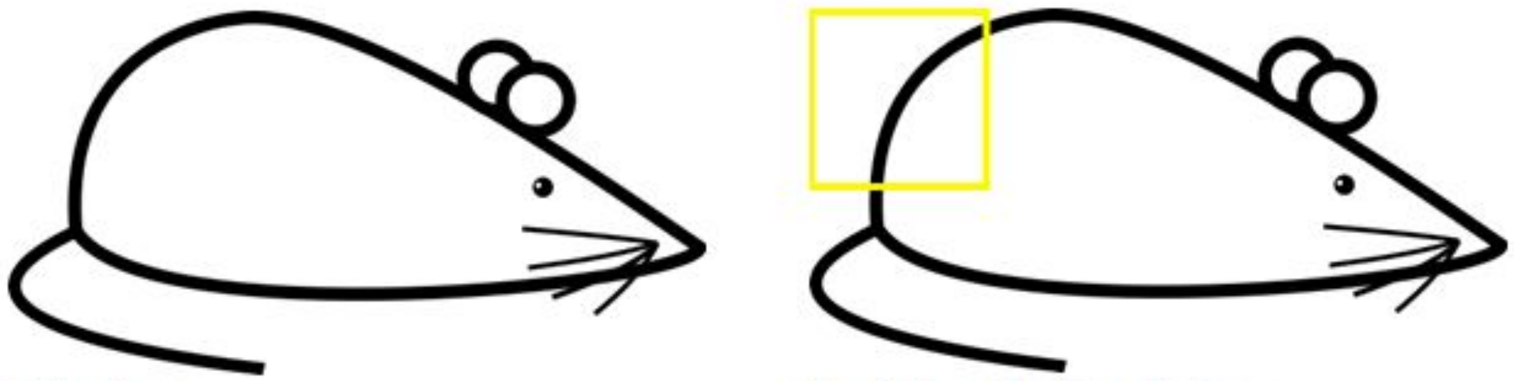
## Introduction to PyTorch

Notebook: Introduction_to_PyTorch.ipynb

## Exercise I: Linear Regression with PyTorch

Notebook: Exercise_1.ipynb

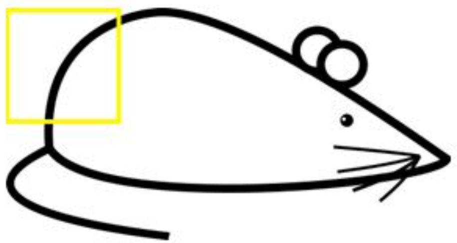
## Introduction Convolutional Neural Networks (CNN)



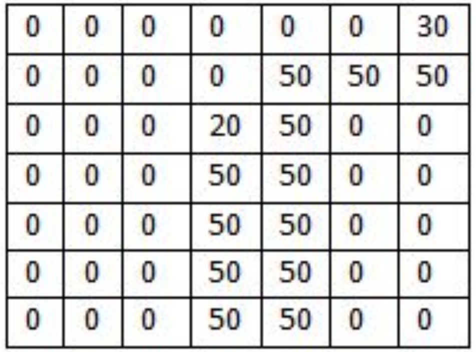
Humans use forms to classify image

→ CNNs use the same concept

# Introduction Convolutional Neural Networks (CNN)



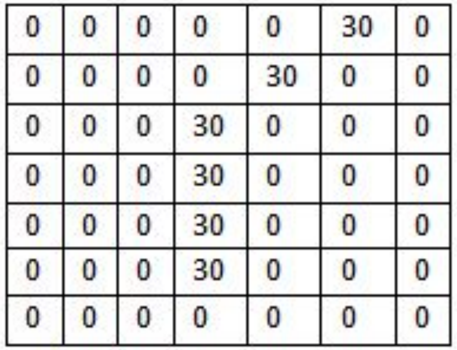
Part of an image



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

Matrix of part of an image

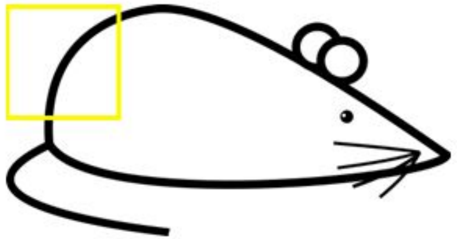
*



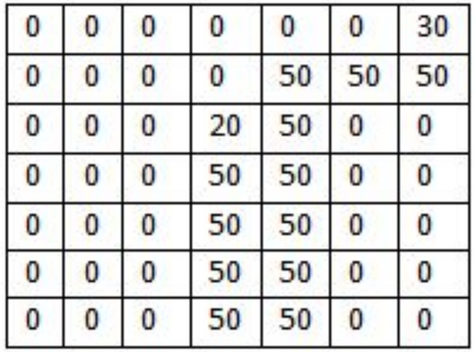
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter

# Introduction Convolutional Neural Networks (CNN)



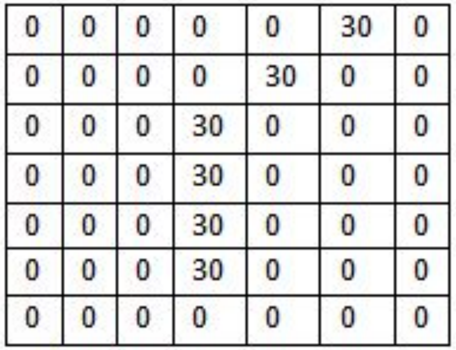
Part of an image



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

Matrix of part of an image

*

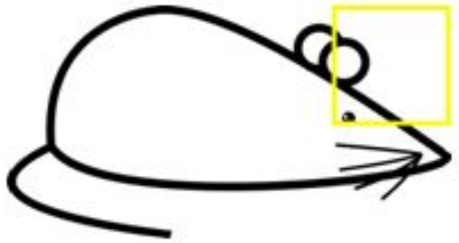


| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

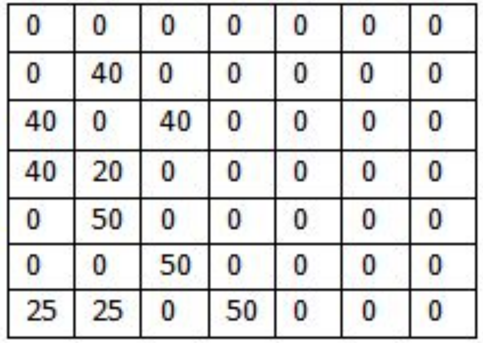
Filter

Multiply + Add = (50*30) + (50*30) + (50*30) + (50*30) + (20*30) + (50*30) = 6600

# Introduction Convolutional Neural Networks (CNN)



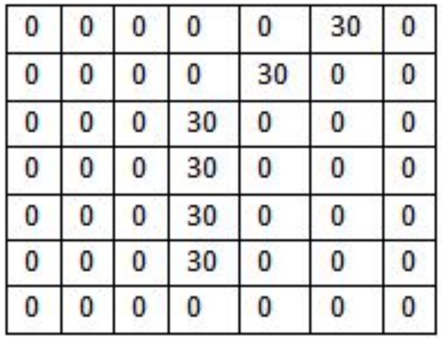
Part of an image



| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 40 | 0 | 0 | 0 | 0 | 0 |
| 40 | 0 | 40 | 0 | 0 | 0 | 0 |
| 40 | 20 | 0 | 0 | 0 | 0 | 0 |
| 0 | 50 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 50 | 0 | 0 | 0 | 0 |
| 25 | 25 | 0 | 50 | 0 | 0 | 0 |

Matrix of part of an image

$*$

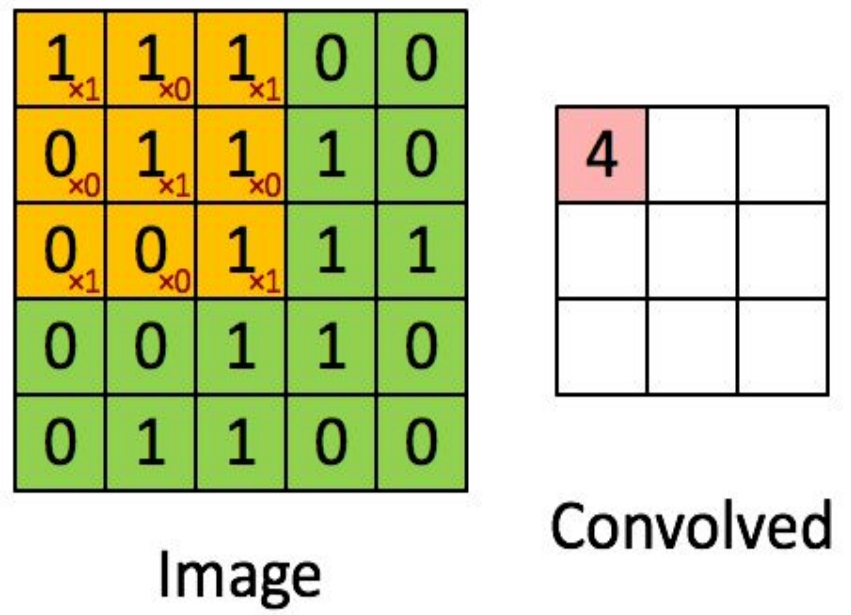


| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter

Multiply + Add = 0

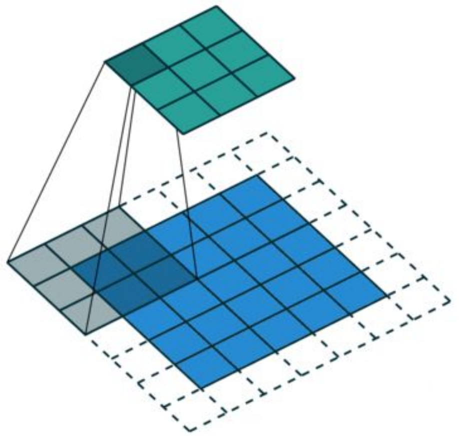
## Introduction Convolutional Neural Networks (CNN)



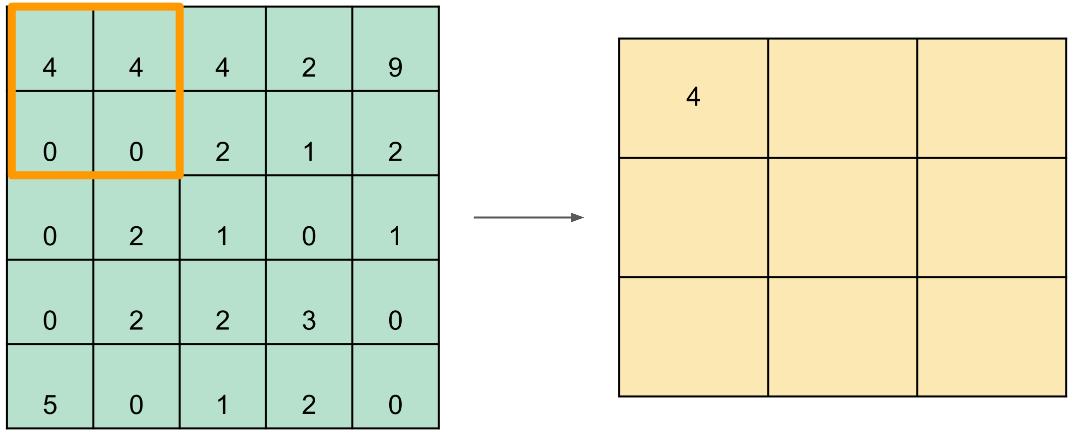
➡ Filter operation is performed by a new layer: Convolutional Layer

## Convolutional Layer

- Kernel-size (aka filter size): size of the filter matrix (always quadratic)
- Stride (aka step size): step size of filter movement
- Padding: adds pixel around the border of an image (0s most often)



# Max Pooling Layer subsamples feature map



Feature Map

Max Pooling Layer

## Convolutional Layer

- Kernel-size/filter size: 3x3, 5x5, 7x7,...
- Stride: step size of the convolution
- Padding: adds pixel around the border of an image (0s most often)

Calculate output size:

$$\frac{height/width + 2 * padding - filtersize}{stride} + 1$$

Number of weights:

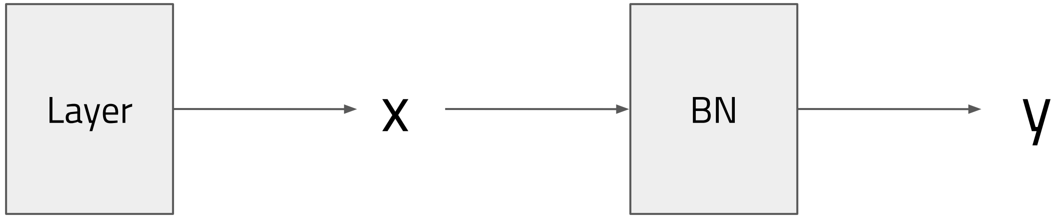
filter size * filter size * # channels + 1 bias

# Batch Normalization Layer

- Same idea as input normalization
- Reduces dependency on initialization
- Useful side effect: added regularization
- Two step calculation:
  - Calculate mean over batch and subtract it
  - Calculate standard deviation over batch and divide by it
- two learnable parameters enable the network switch off BN

$$\hat{x} = \frac{x - \mu}{\sigma}$$

$$y = \gamma \hat{x} + \beta$$



## Common layers of a CNN

- Conv (convolutional layer): extracts features from input/feature map
- Pool (max pooling layer): subsamples feature map
- FC (fully connected layer): combines features
- ReLu: adds non-linear relationship
- Softmax: output layer for multiclass classification
- BN (batch normalization layer): helps the gradient to flow better

## LeNet (1998)

- Input -> Conv -> Sigm -> Pool -> Conv -> Sigm -> Pool -> FC -> Sigm -> Output
- Task: recognize handwritten digits for banks

Yann LeCun, first author of LeNet paper; now at facebook

## Next Notebook: LeNet

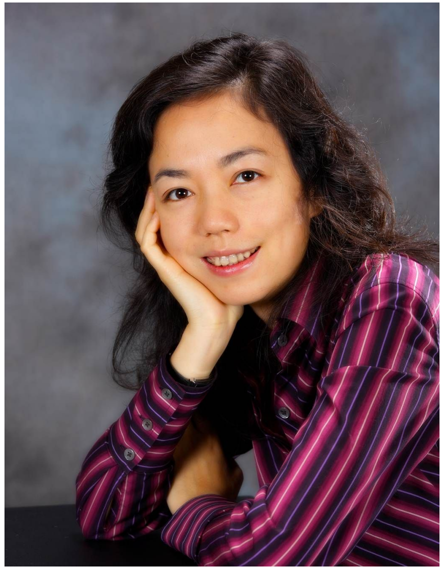
Notebook: LeNet.ipynb

## History of CNN architectures

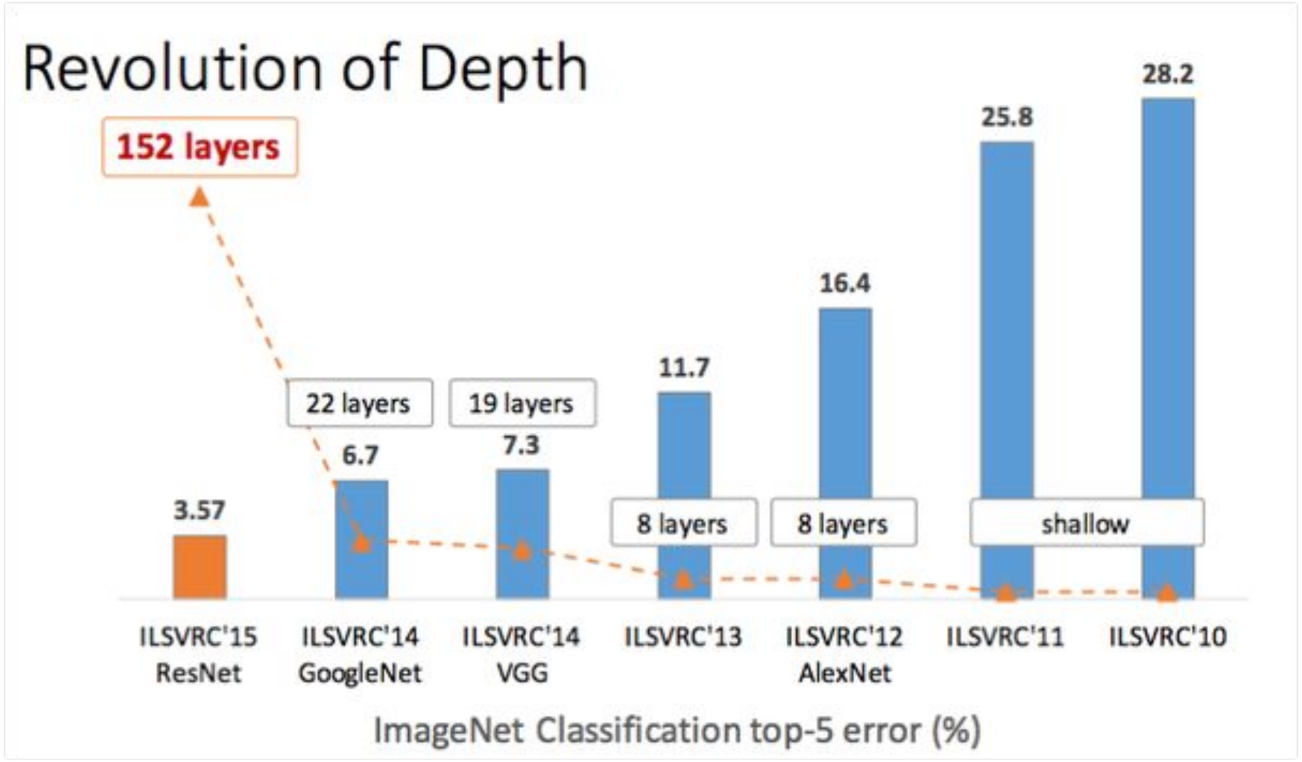
- LeNet (1998)
- ImageNet challenge (2010)
- AlexNet (2012)
- VGGNet (2014)
- Inception (2014)
- ResNet (2015)
- Xeption (2016)

## Introduction ImageNet Classification Challenge (ILSVRC)

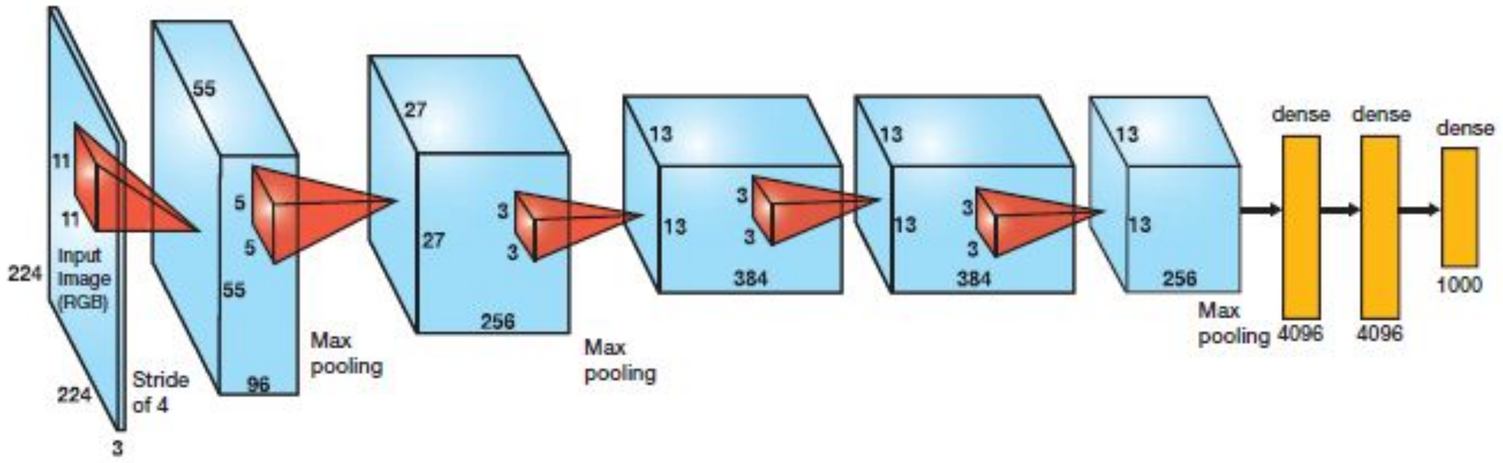
- started in 2006 with the help of Amazon Mechanical Turk
- 3.2 Mio labelled images; 5,247 categories
- ended in 2017
- > 13 Mio labelled images; > 1000 categories
- **free** to download



# Winning architectures became deeper every year



## AlexNet (2012)



Architecture of AlexNet

## AlexNet (2012) - BEGIN OF DEEP LEARNING

- 60 Mio. parameters, 650,000 neurons
- Top 5 error rate of 15,3% on ImageNet challenge
- 1.2 Mio. trainings images => training took 5-6 days on 2 GPUs (90 Epochs)
- Key contribution: kick off deep learning interest + data augmentation

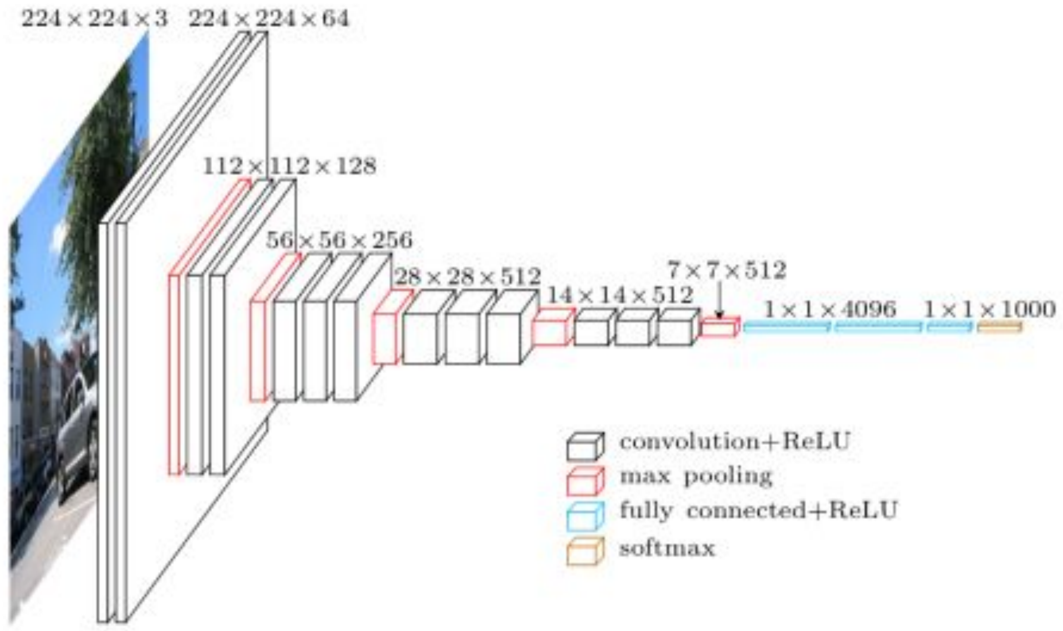
Next notebook: AlexNet.ipynb

Alex Krizhevsky,
now at google

## VGGNet (2014)

- Invented by **V**isual **G**eometry **G**roup at Oxford
- VGG16 => 16 layer; VGG19 => 19 layer
- 138 Mio. parameters
- Top 5 error rate of 7,3% on ImageNet challenge
- Training for ImageNet took two to three weeks using four Nvidia Titan Black GPUs
- Easy to understand => good baseline
- Available in most libraries
- Key contribution: Deeper networks perform better

# VGGNet (2014)



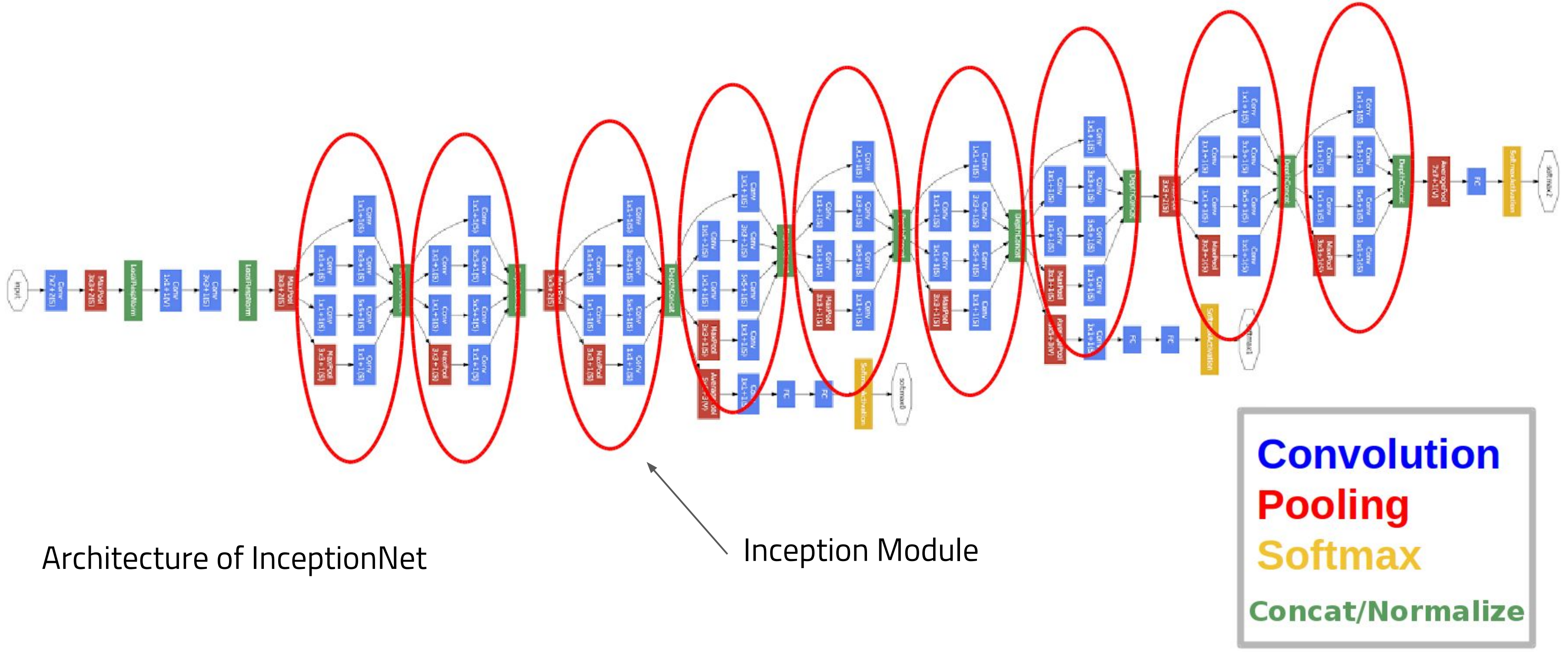
Architecture of VGG

## InceptionNet/GoogLeNet (2014)

- 22 layers/100 layers (depending on how you count)
- Main advantage: only 5 Mio. parameters (no FC layers)
- Top 5 error rate of 6.7% on ImageNet challenge
- Key contribution: Inception module

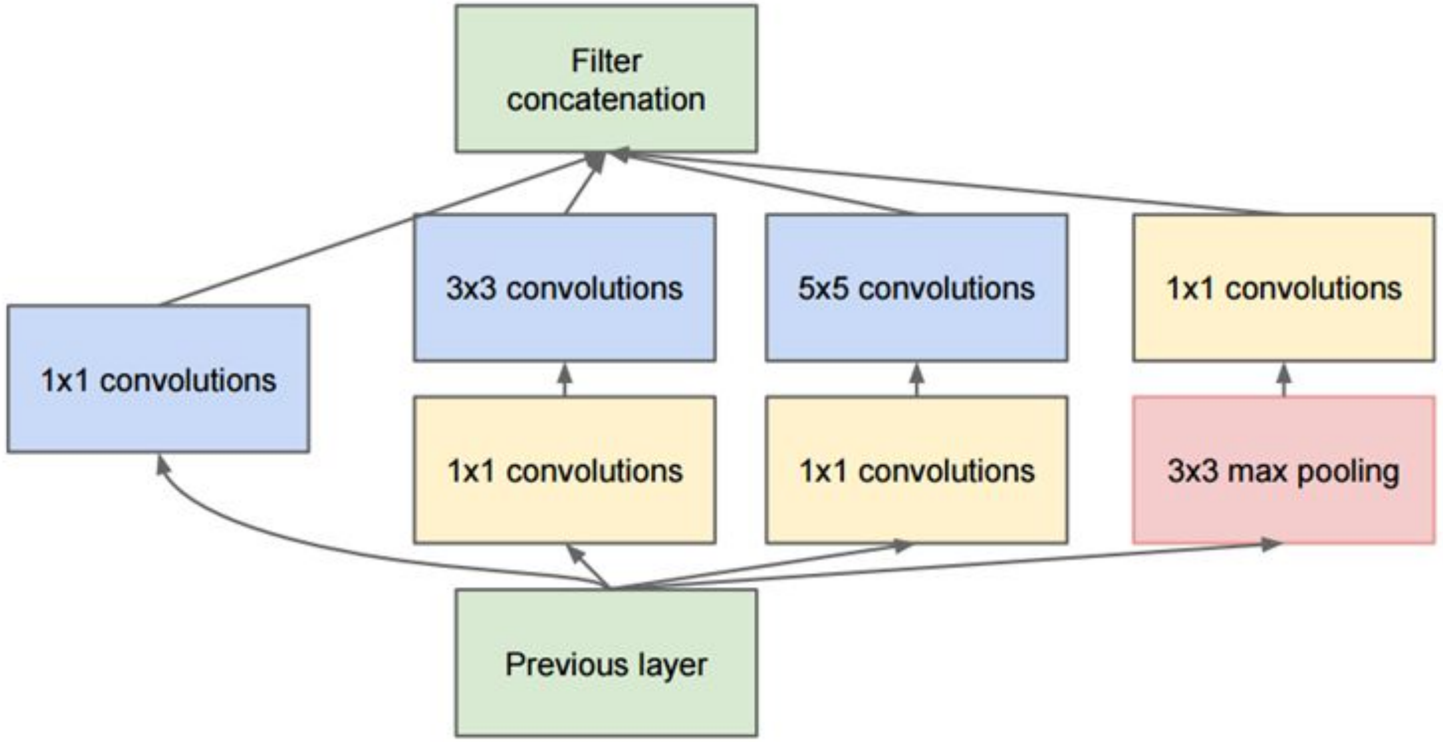
Christian Szegedy,
now at google

# InceptionNet/GoogLeNet (2014)



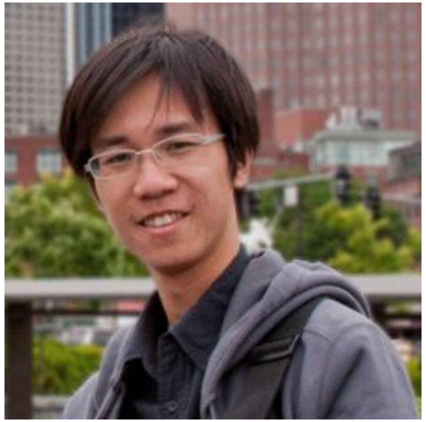
Architecture of InceptionNet

# Inception module aka “network in network”



## ResNet (2015)

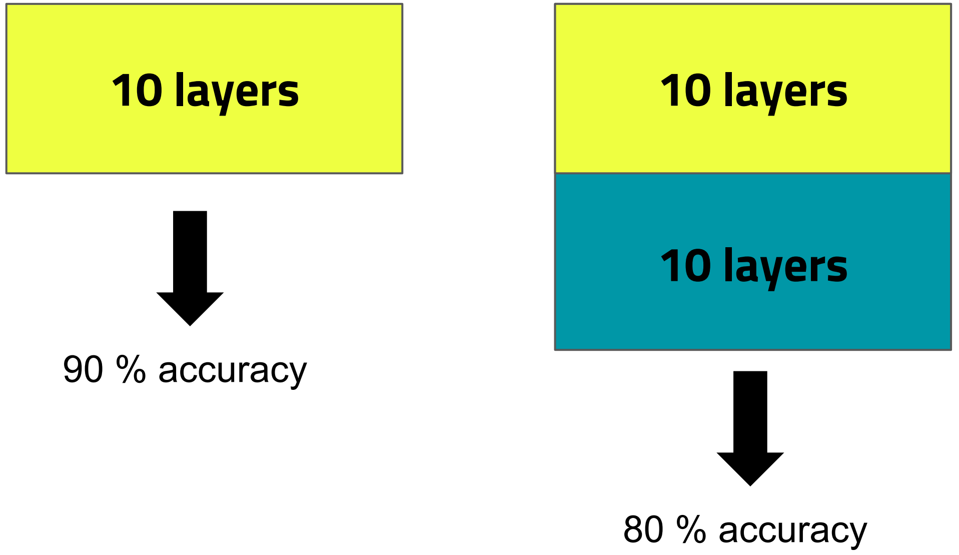
- ResNet50 -> 50 layers; ResNet152 -> 152 layers
- Microsoft Research team
- Top 5 error rate of 3.6% on ImageNet challenge
- Key contribution: Residual block (enables us to make network deeper)



Kaiming He,
now at facebook

# ResNet - degradation problem

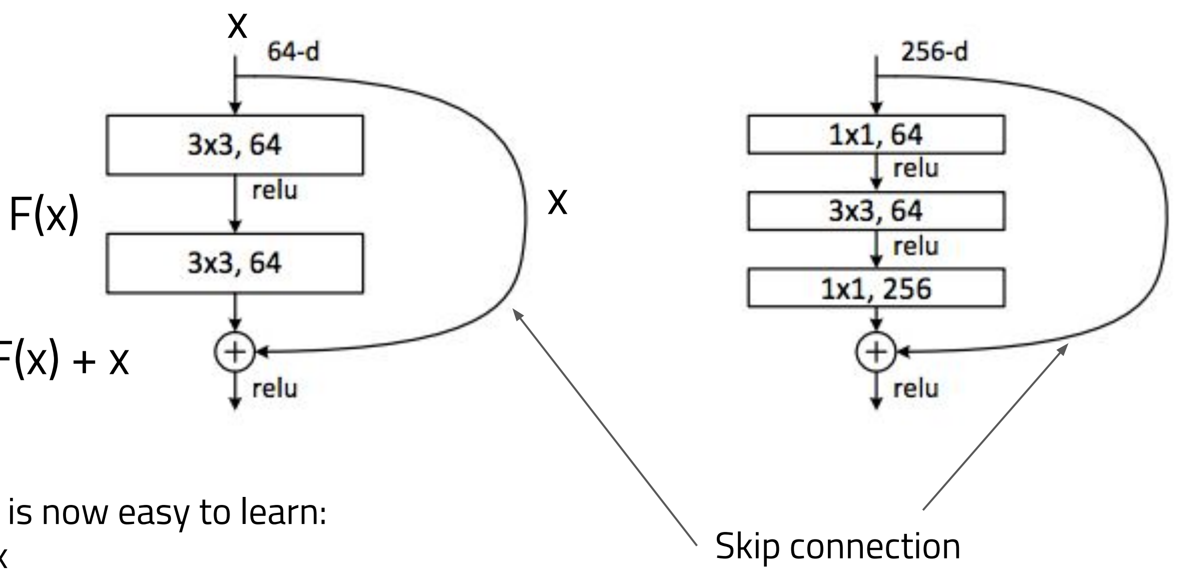
Thought experiment



This problem is called the degradation problem. Deeper layers struggle to even find an identity mapping.

How can we do depper?

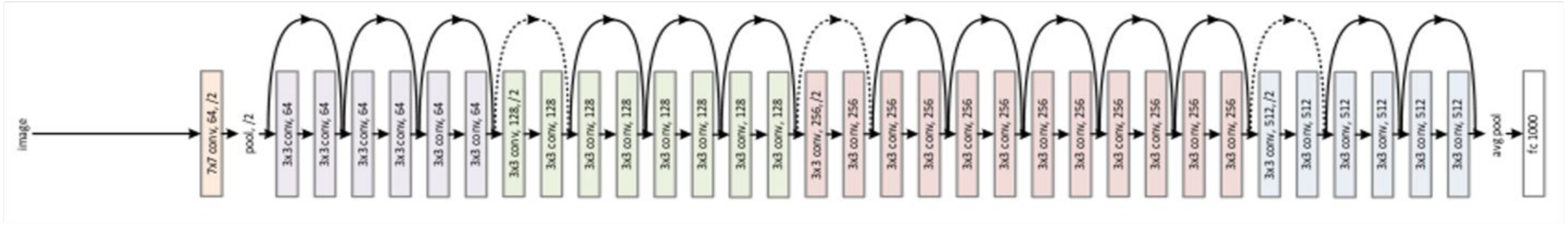
# Residual block



$H(x) = F(x) + x$

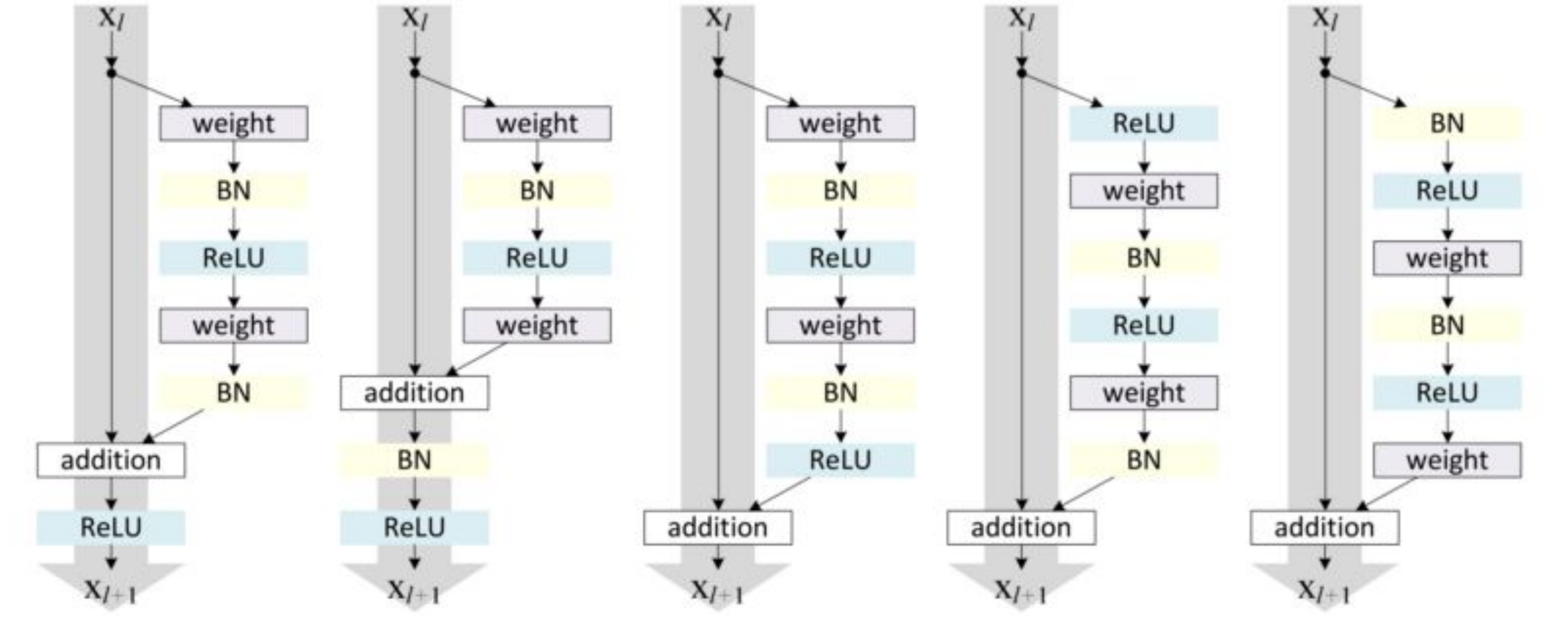
Identity is now easy to learn:
$x = 0 + x$

# ResNet



Architecture of ResNet

# New releases of ResNet



(a) original

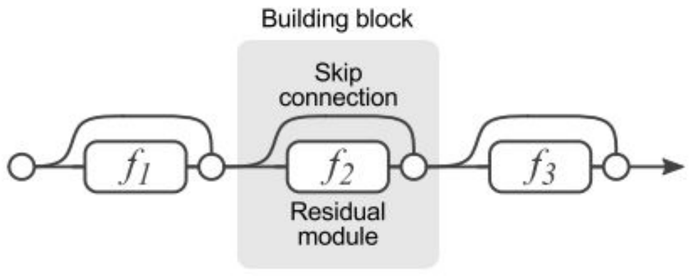
(b) BN after addition

(c) ReLU before addition

(d) ReLU-only pre-activation

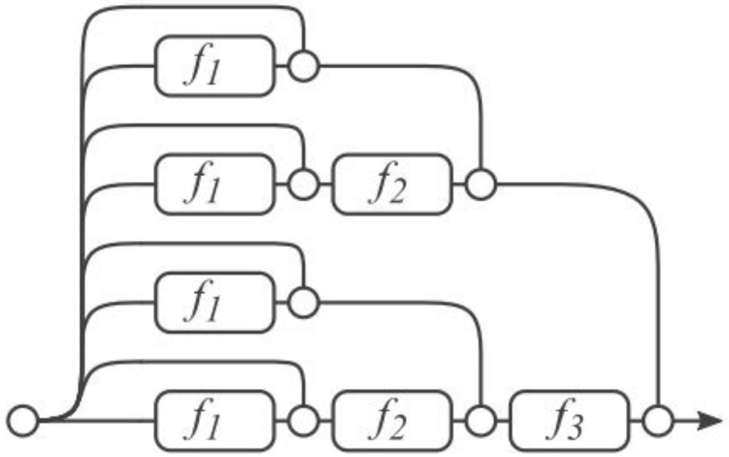
(e) **full pre-activation**

# New Hypothesis why ResNet works so well



(a) Conventional 3-block residual network

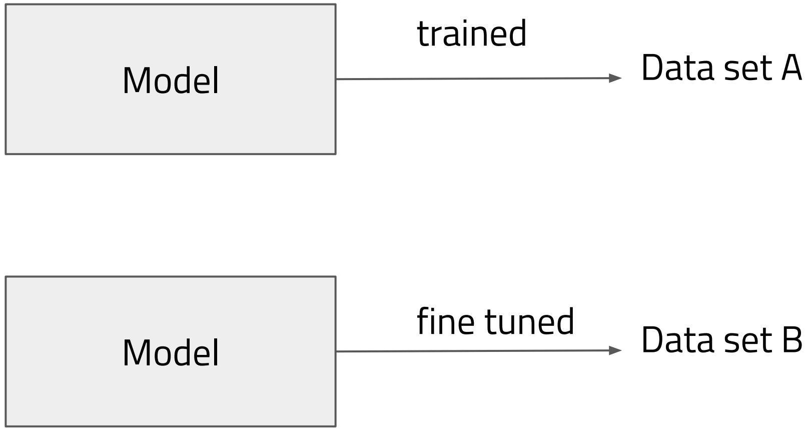
=



(b) Unraveled view of (a)

➡ ResNet works like an ensemble of many paths

# Transfer Learning



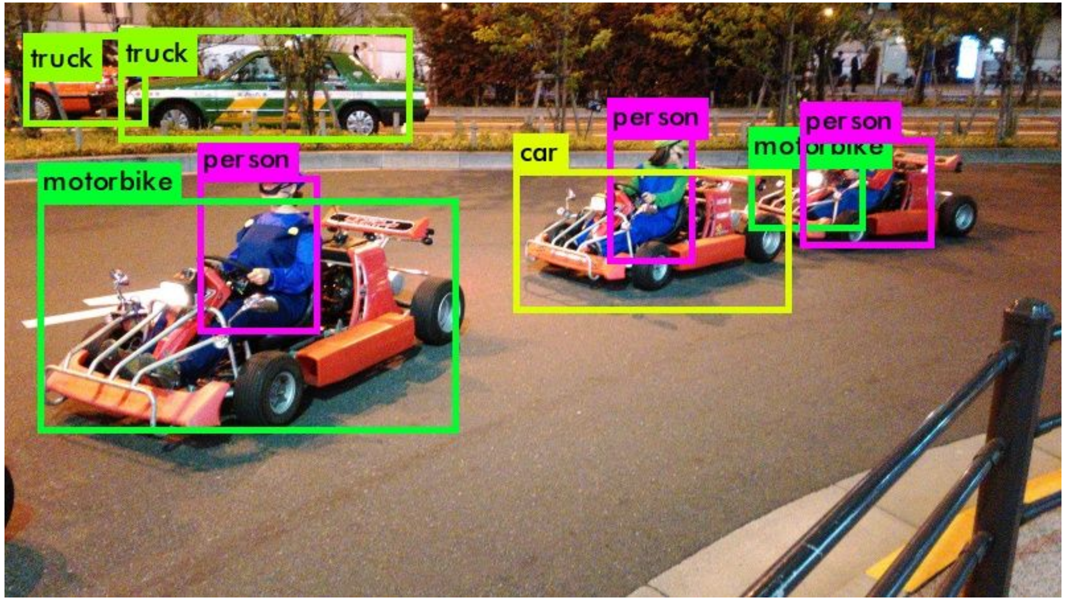
➡ Model can now be used on data set B

## Next notebook: Transfer Learning

## Mittagspause

# Introduction Object detection

cat



classification

object detection

## mean average precision - mAP

https://stackoverflow.com/questions/48461855/understanding-and-tracking-of-metrics-in-object-detection

## Regional-based CNN (R-CNN; 2014)

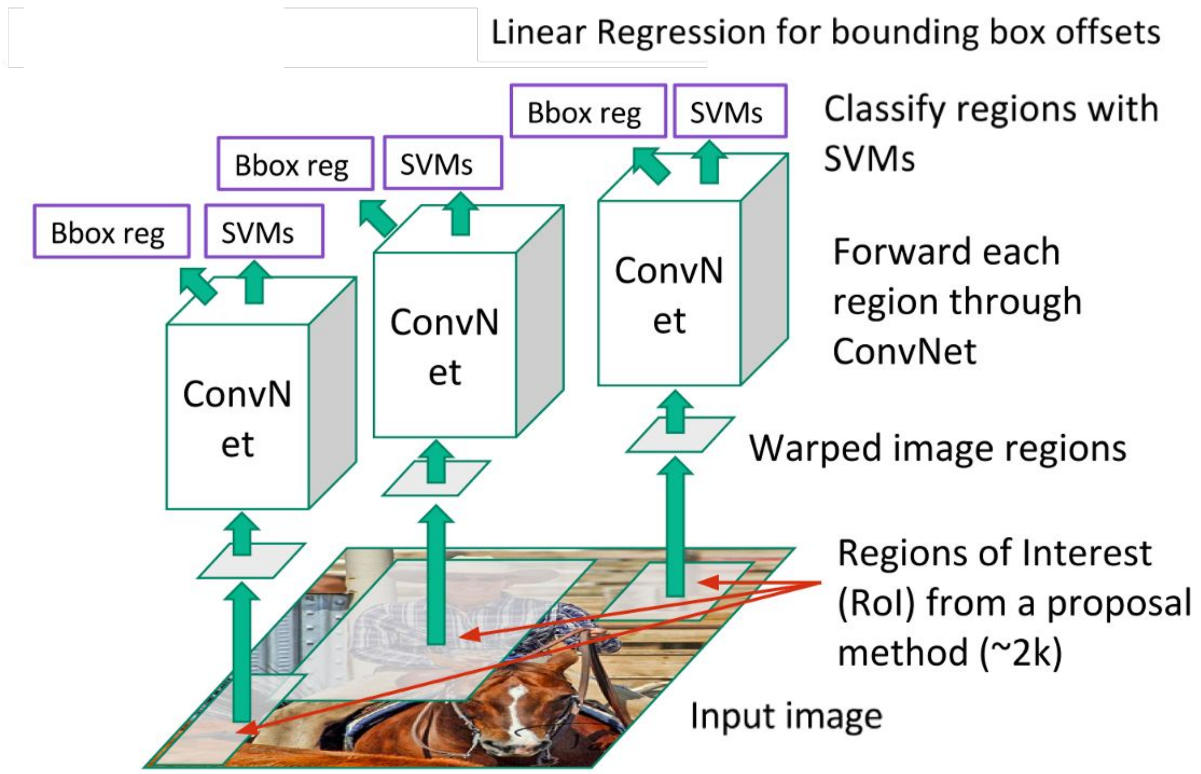
**Overview**

1. Generate ~ 2000 region proposals by using selective search algorithm
2. Transform each region to a 227x227 image
3. Use CNN to extract a fixed-length feature vector from each region
4. Use SVM + feature vector to classify regions
5. Discard all regions when there is a “close” region with a higher prediction score
6. Adjust size of proposed region with correction factor found by linear classifier

**Disadvantage**

~47s/image for a single prediction on a GPU $\Rightarrow$ too slow

## R-CNN visualized


Linear Regression for bounding box offsets
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

# Fast R-CNN (2015)

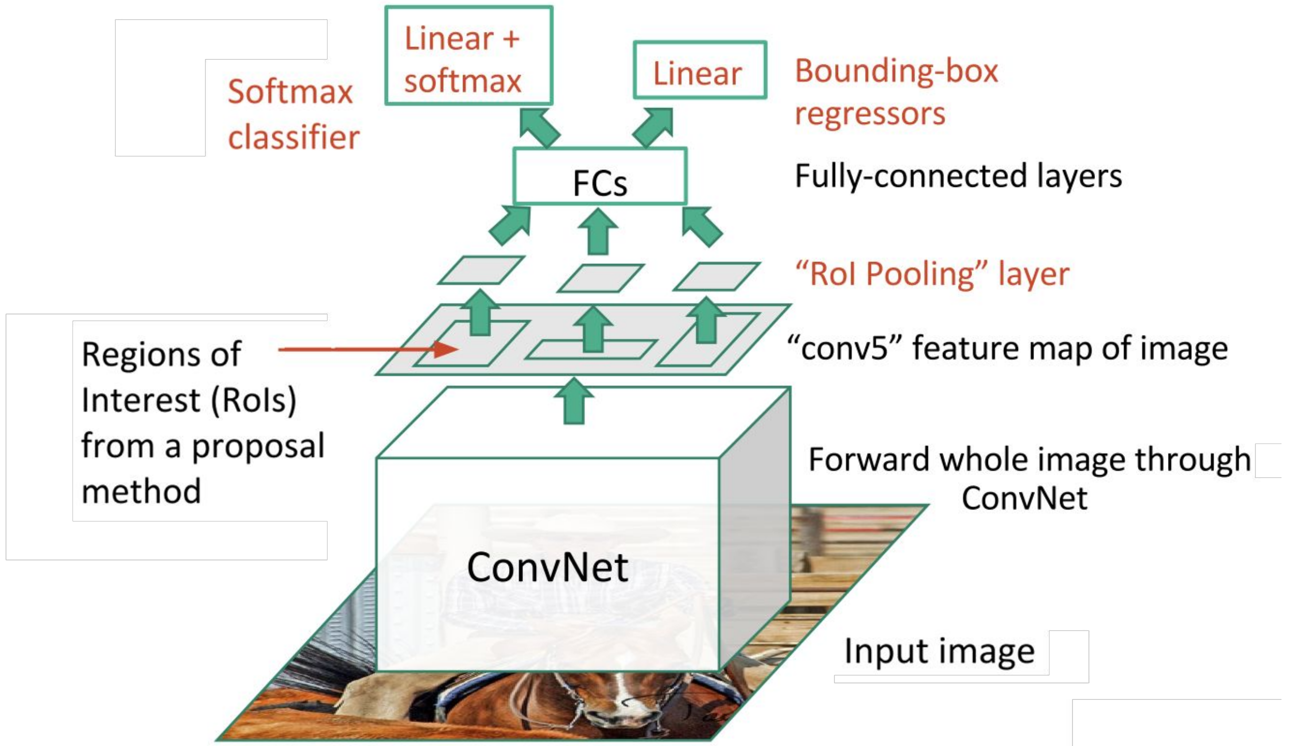
**Overview**

1. Generate regions of interest (ROI) by using selective search algorithm
2. Use one CNN to extract features of the whole input image
3. ROI Pooling: Get extracted features from ROIs
4. Feed ROI features into softmax layer for classification
5. Feed ROI features into another layer and get back four values for the bounding box

**Advantage**

All parameters are part of one network $\Rightarrow$ all weights can be adjusted in one step

# Fast R-CNN visualized



# Faster R-CNN (2015)

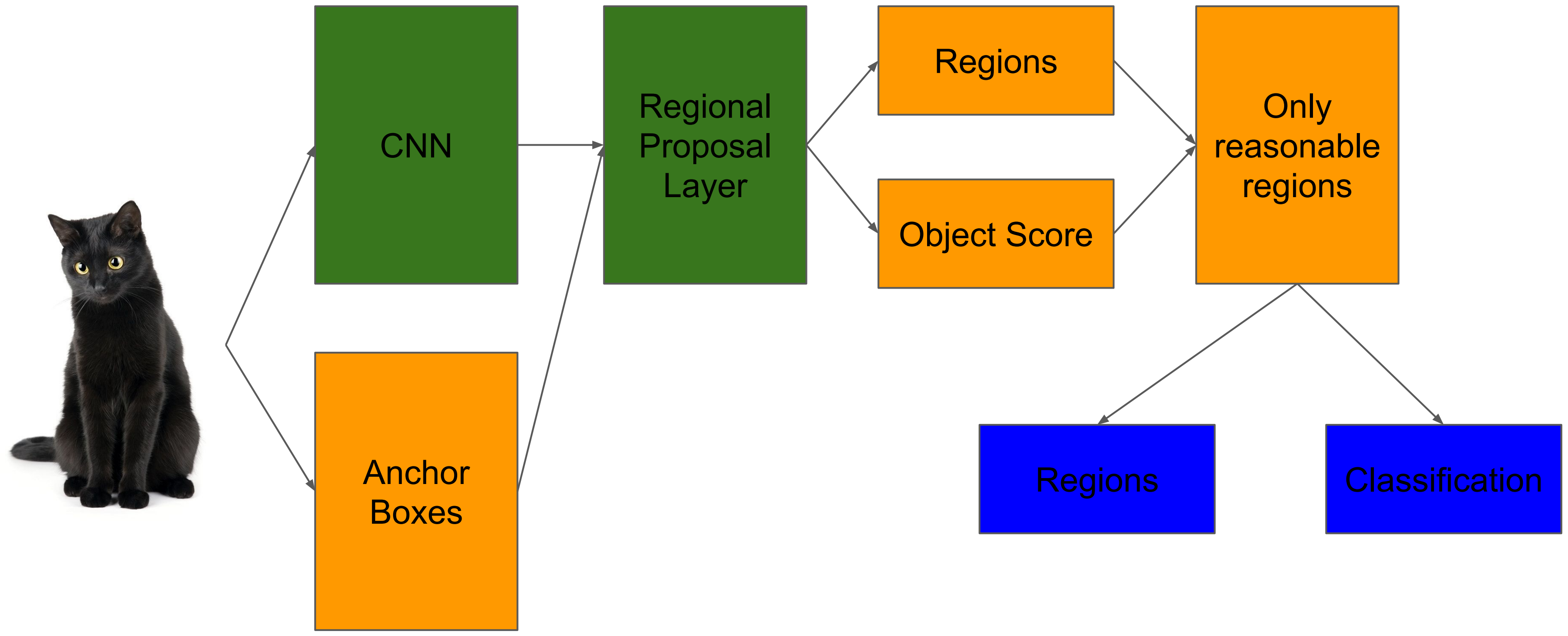
**Overview**

1. Use one CNN to extract features of the whole input image
2. Generate a fixed number of anchor boxes
3. Each anchor box gets a score (object or no object)
4. Depending on the score some anchor boxes will be discarded right away
5. Use the CNN to classify the content of anchor boxes

**Advantage**

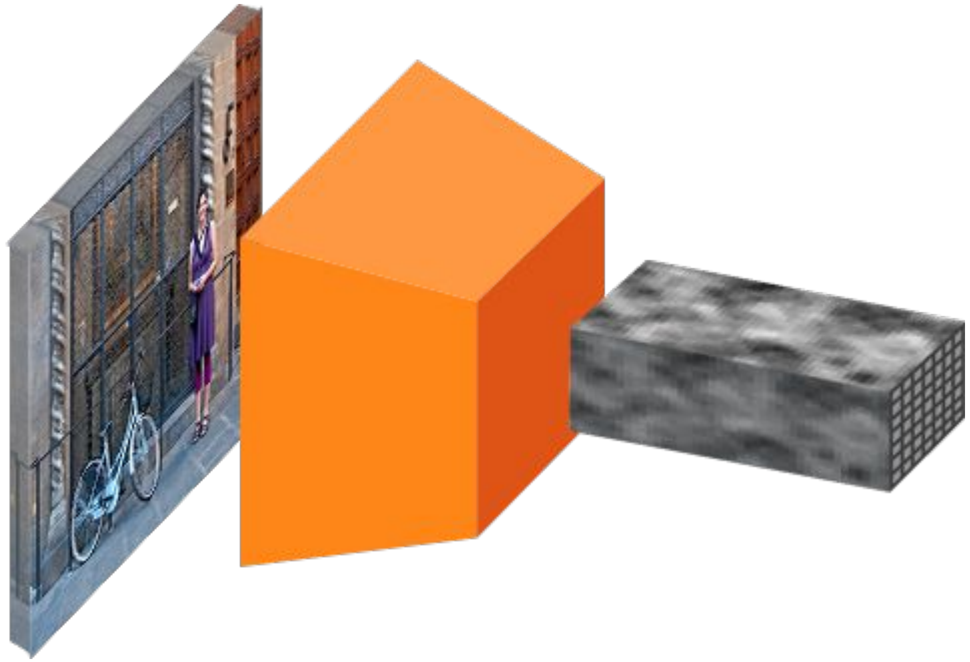
Fully differentiable module all weights can be adjusted in one step

# Faster R-CNN visualized



# Faster R-CNN

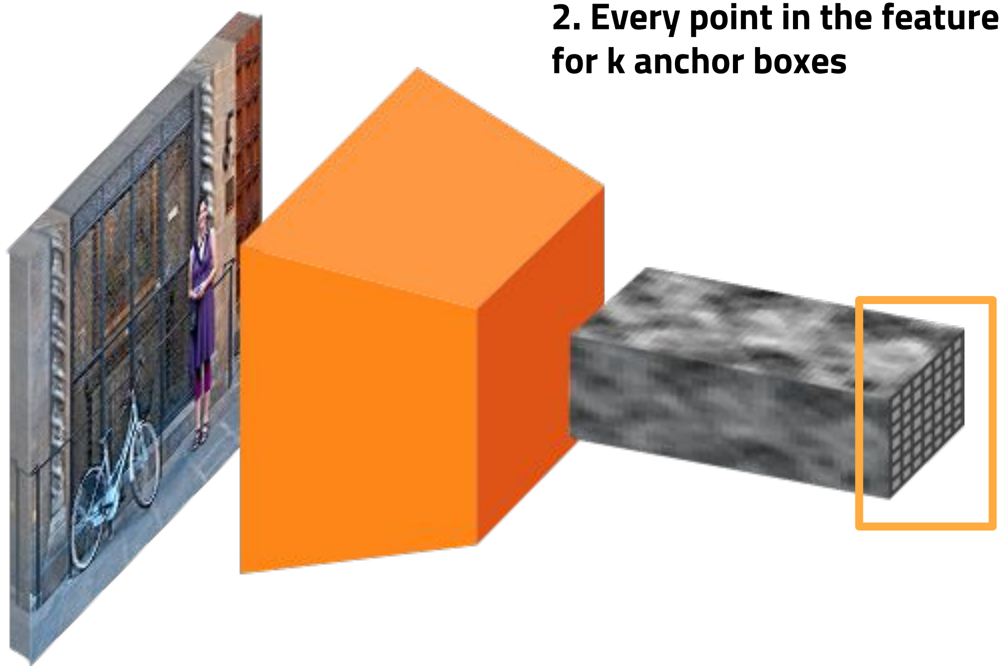
1. **Extract features using a standard CNN**



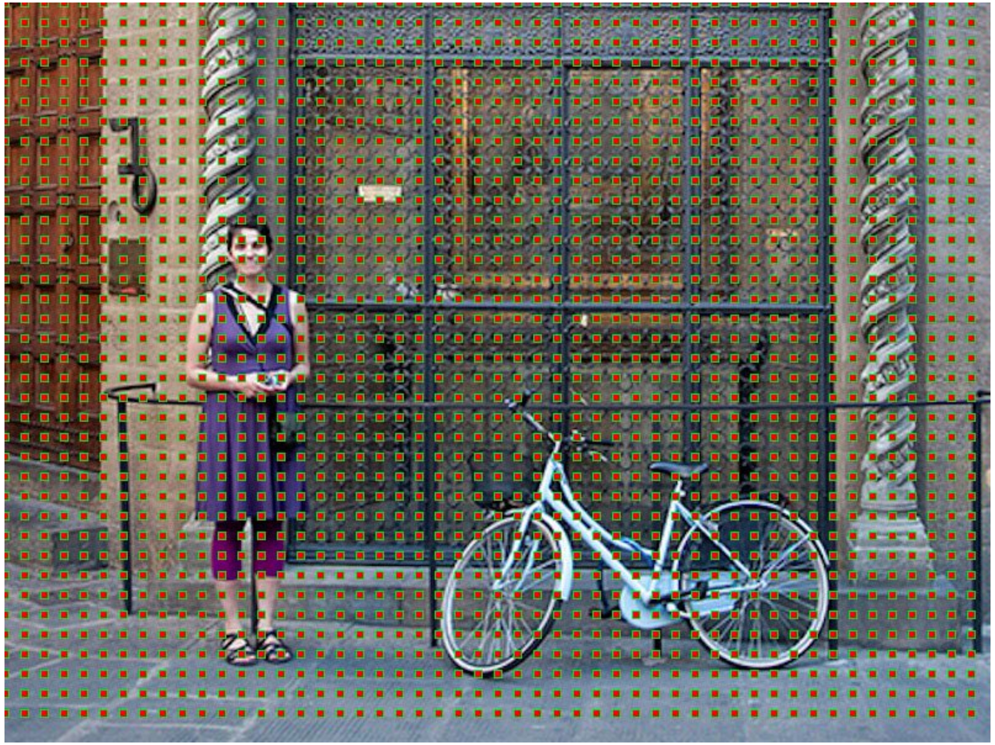
# Faster R-CNN

**1. Extract features using a standard CNN**

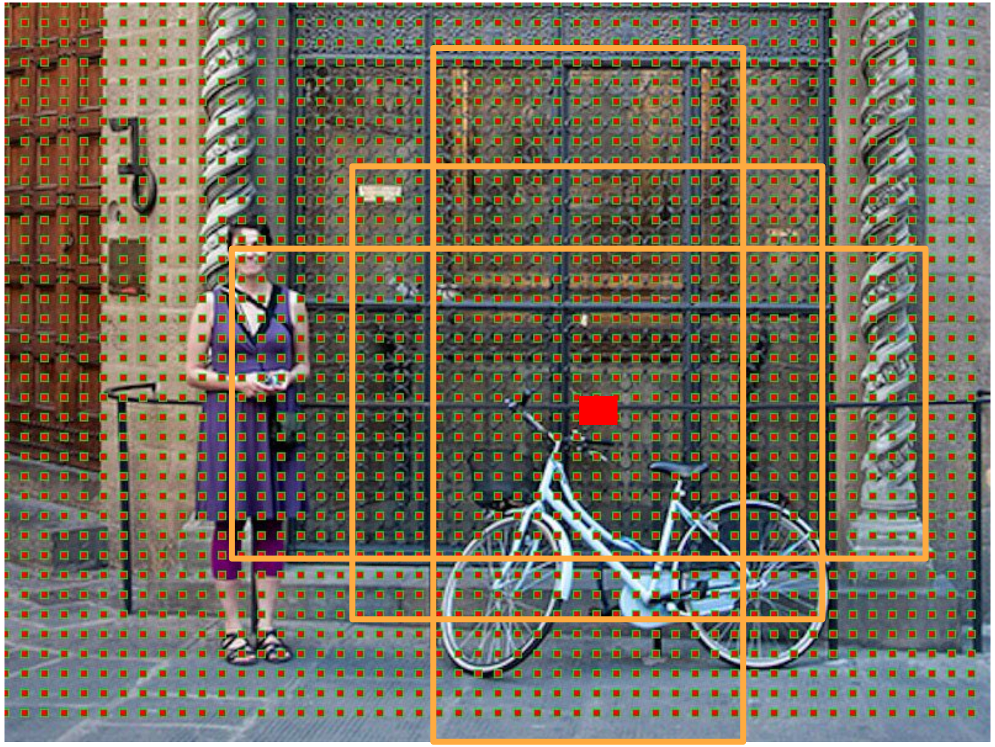
**2. Every point in the feature map is the center point for k anchor boxes**



# Faster R-CNN

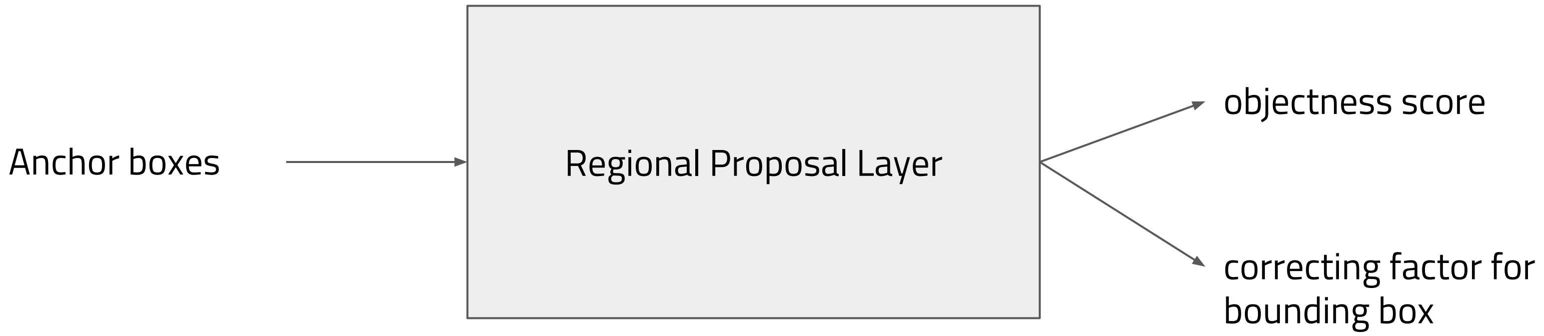


# Faster R-CNN

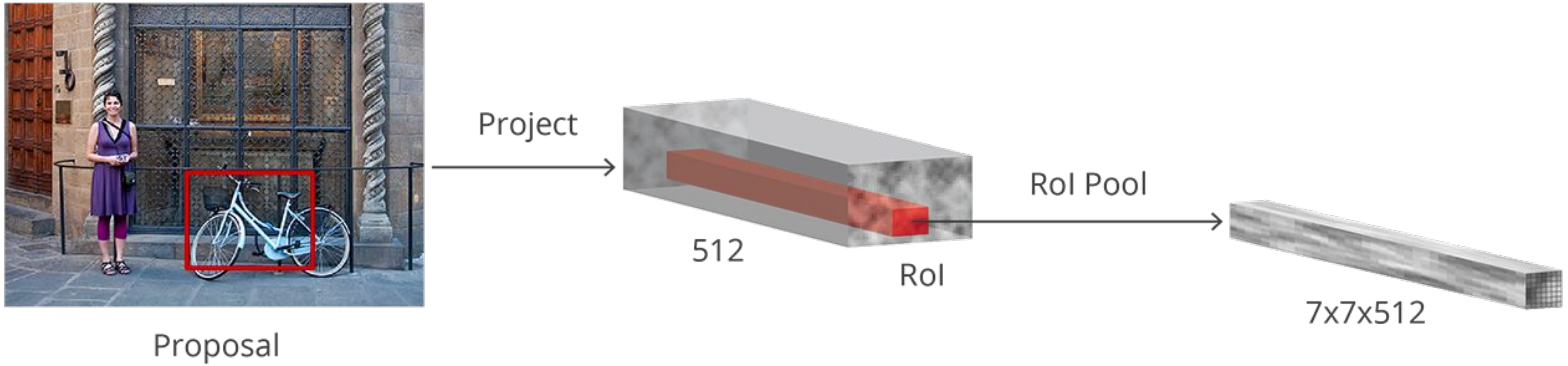


- New center point every 16 pixels
- 9 anchor boxes per center point
- 600 x 800 pic: 17901 anchor boxes

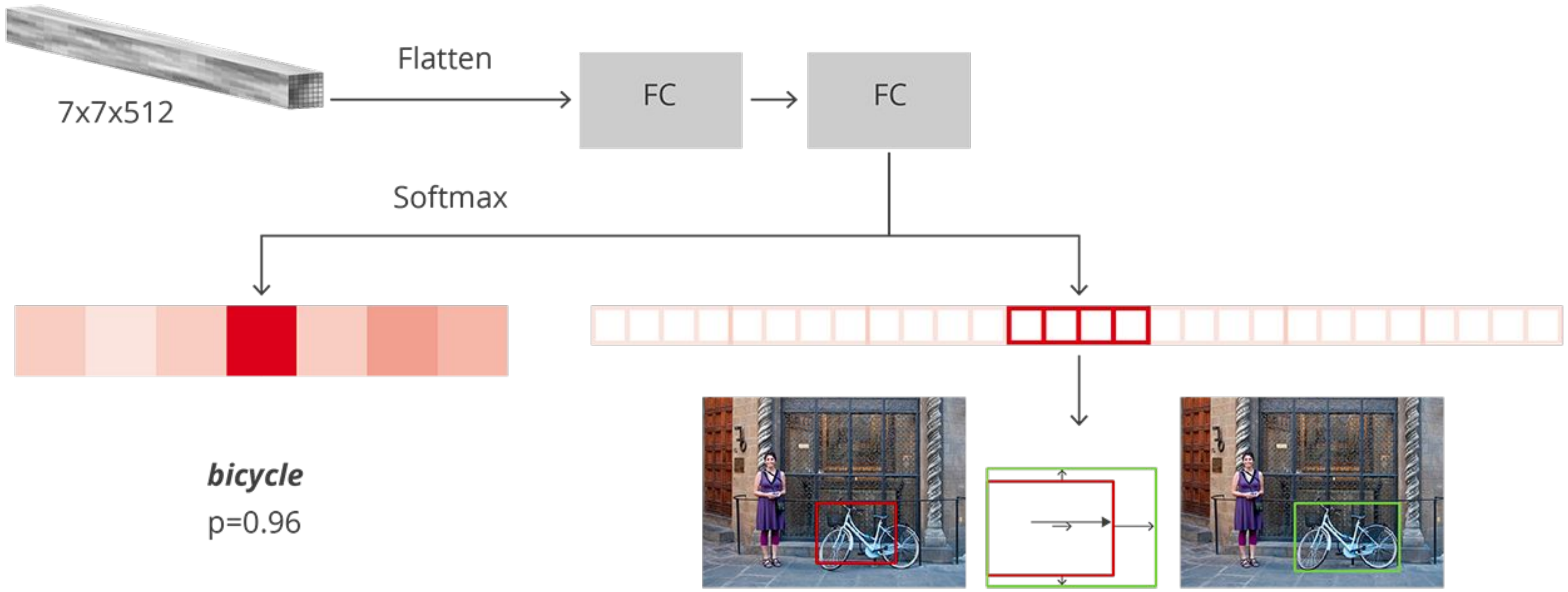
# Regional Proposal Network (RPN)



# Classify ROI



# Final prediction



## Training

- Four different losses
  - object score for RPN
  - correction factor for RPN
  - Softmax for final classification
  - correction factor for final bounding box regression
- Researchers found a way to combine these four losses in one loss function

## Comparison

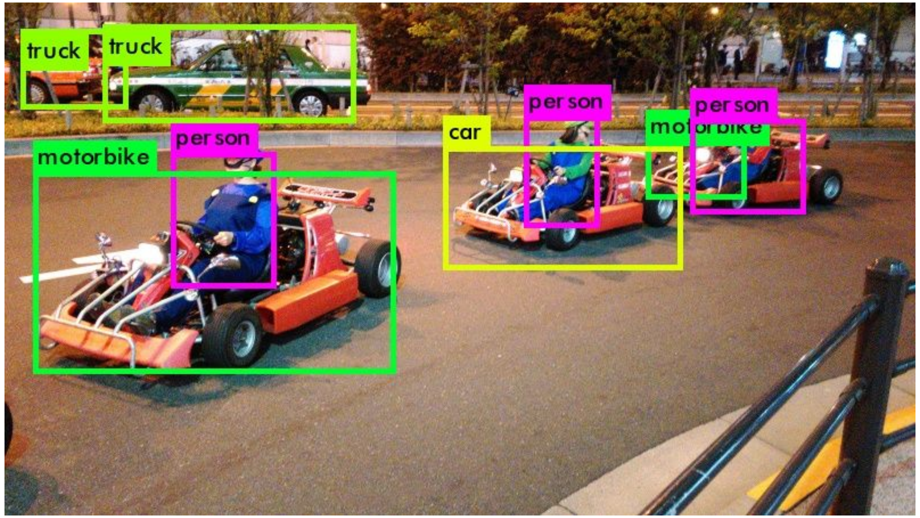
R-CNN: 50 secs/image

Fast R-CNN: 2 sec/image

Faster R-CNN: 0.2sec/image

# Single Shot Detection

- BEFORE: Region proposal + classification = two steps
- NOW: Regression = one step

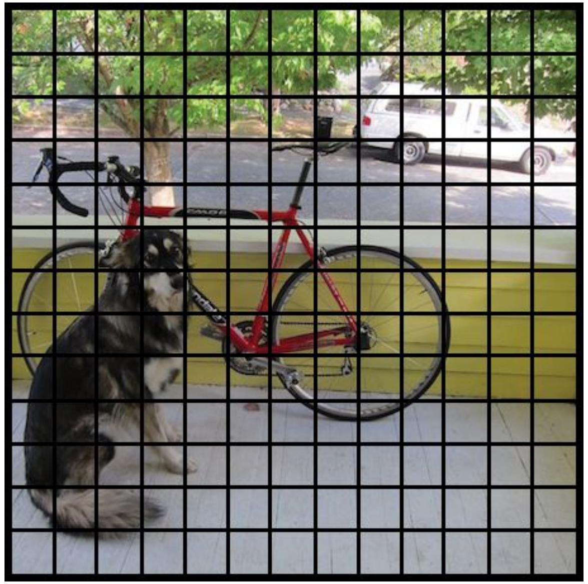


# Yolo v2

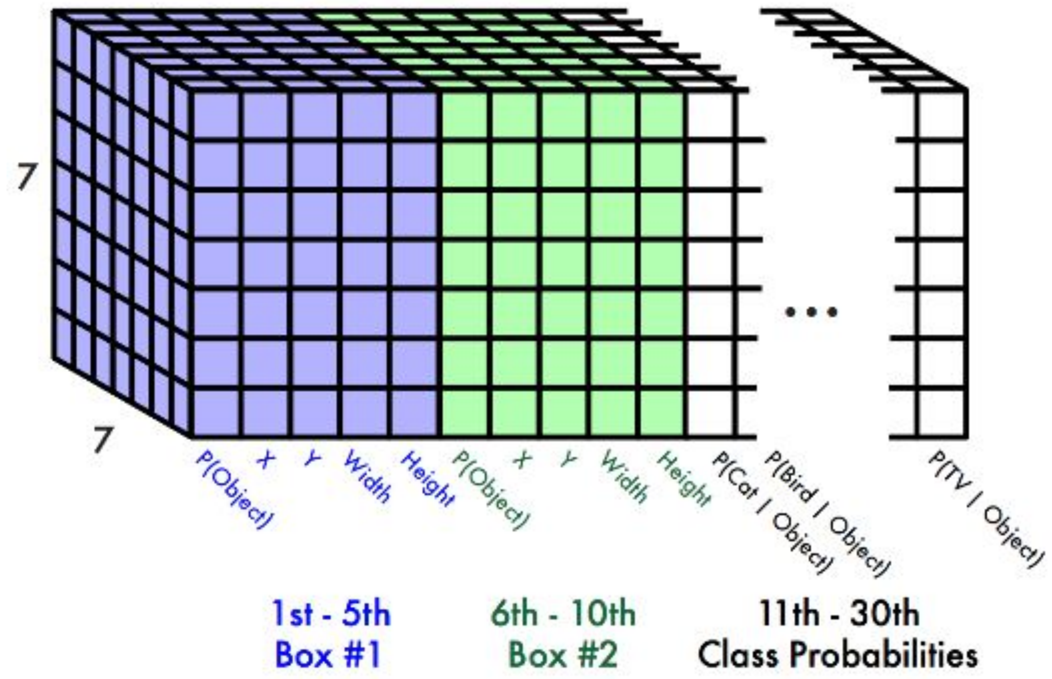
- Divides image in S x S grid
- Each grid cell predicts
  - B bounding boxes
  - a confidence score for each bounding box
  - One set of class predictions

Bsp: Pascal VOC: S=7, B=2

=> 7x7x(2x(4+1)+20 classes) = 7x7x30 output

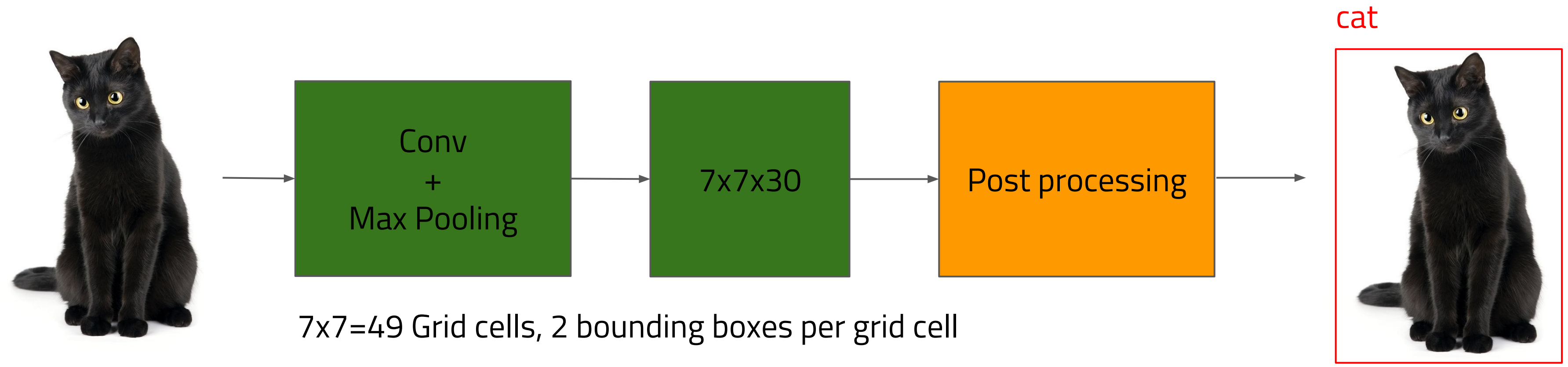


## Yolo v2



7x7x30

## Yolo v2



7x7=49 Grid cells, 2 bounding boxes per grid cell

30 infos per grid cell = 2 bounding boxes (x,y,w.h) + 2 confident scores + 20 object classes

**Hyperparameter**

- Number of grid cells
- Number of bounding boxes per grid cell
- Threshold at which bounding box is considered valid

## Comparison Object detection algorithms

| Name | backbone | AP_small | AP_medium | AP_large |
|---|---|---|---|---|
| Faster RCNN | ResNet-101 | 15.6 | 38.7 | 50.9 |
| Yolo v2 | Darknet-19 | 5.0 | 22.4 | 35.5 |
| Yolo v3 | Darknet-53 | 18.3 | 35.4 | 41.9 |
| RetinaNet | ResNeXt-101 | 24.1 | 44.2 | 51.2 |

## Natural Language Modeling

The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English. The authors claimed that within three or five years, machine translation would be a solved problem.

**"Robot brain translates Russian into King's English"** – IBM, January 7, 1954

## Natural Language Modeling - Tasks

- Semantic analysis
- Translation
- Speech2Text
- Text2Speech
- Text generation
- Chat bots
- Spam detection

## Word embeddings - Why and What?

- Machine Learning Algorithms can usually only work with numbers not strings
- We could convert every word to a one-hot encoded vector

  => But relationships between words will be lost

**Example**

| | | |
|---|---|---|
| Monday | 0 | 0 0 0 1 |
| Tuesday | 1 | 0 0 1 0 |
| Wednesday | 2 | 0 1 0 0 |
| Thursday | 3 | 1 0 0 0 |

# Word embeddings - Why and What?

- Machine Learning Algorithms usually can only work with numbers not strings
- We could convert every word to a one-hot encoded vector

  => But relationship between words will be lost

**Example**

| | | |
|---|---|---|
| Monday | 0 | 0 0 0 1 |
| Tuesday | 1 | 0 0 1 0 |
| Wednesday | 2 | 0 1 0 0 |
| Thursday | 3 | 1 0 0 0 |

order remains

no relationship

# Word embeddings - Why and What?

- Machine Learning Algorithms usually can only work with numbers not strings
- We could convert every word to a one-hot encoded vector
  => But relationship between words will be lost

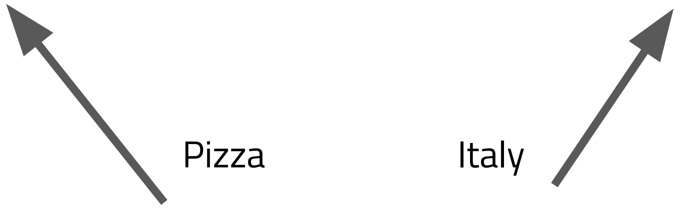
**Example**

| | | |
|---|---|---|
| Monday | 0 | 0 0 0 1 |
| Tuesday | 1 | 0 0 1 0 |
| Wednesday | 2 | 0 1 0 0 |
| Thursday | 3 | 1 0 0 0 |

order remains

no relationship

“monkey” 1… 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …
“flower” 0… 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …



words as 50000 dimensional vectors

## Word embeddings

V(“Italy”) - V(“Pizza”) = V(“Germany”) - (“Bratwurst”)

V(“King”) - V(“Queen”) = V(“Man”) - V(“Woman”)

V(“King”) - V(“Man”) + V(“Woman”) = V(“Queen”)

➡ Equations with word vectors actually make sense

# Word2Vec

Word2Vec describes a family of algorithms which can generate word embeddings.

- CBOW (Continuous Bag of words)
  - Predict a word given its context words
  - Works well with small amount of the training data, represents well even rare words or phrases
- Skip – Gram model
  - Predict the context words of a given word
  - Several times faster to train than the skip-gram, slightly better accuracy for the frequent words

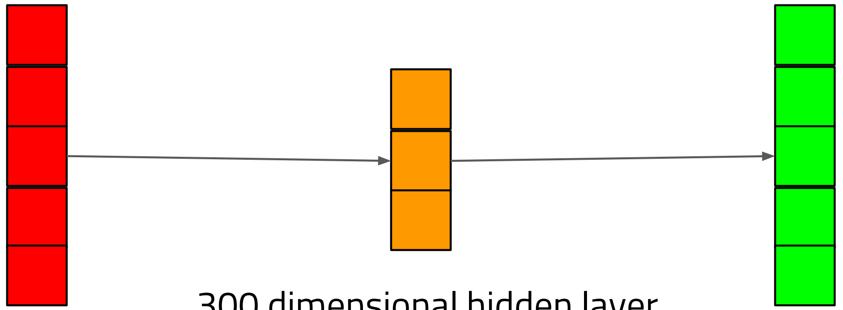
Tomas Mikolov, currently at Facebook

➡ Tomas Mikolov et al managed to speed up the computation of both algorithms tremendously

# Word2Vec - Skip gram

Donald Trump really likes watching Fox News and being on the Times cover.

target: watching          context: really, liked, Fox News



In the paper a vocabulary size of 3 millionen words and a training corpus of 100 billionen words were used.

|v|: vocabulary size

300 dimensional hidden layer

onehot encoded input layer size |v|

softmax output layer of size |v|

# Global vectors for word representations (GloVe)

GloVe is also an algorithm to build word representations.

1. Build a co-occurrence matrix X
2. Find vectors for the words i and j so that: $\vec{w}_i^T \vec{w}_j + b_i + b_j = \log X_{ij}$
3. $$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f\left(X_{ij}\right)\left(\vec{w}_i^T \vec{w}_j + b_i + b_j - \log X_{ij}\right)^2$$

$$f\left(X_{ij}\right) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^{\alpha} & \text{if } X_{ij} < x_{\max} \\ 1 & \text{otherwise.} \end{cases}$$

The function prevents common word pairs to skewing the result

# spaCy

spaCy is a free, open-source library for Natural Language Processing (NLP) in Python.

**Main features**

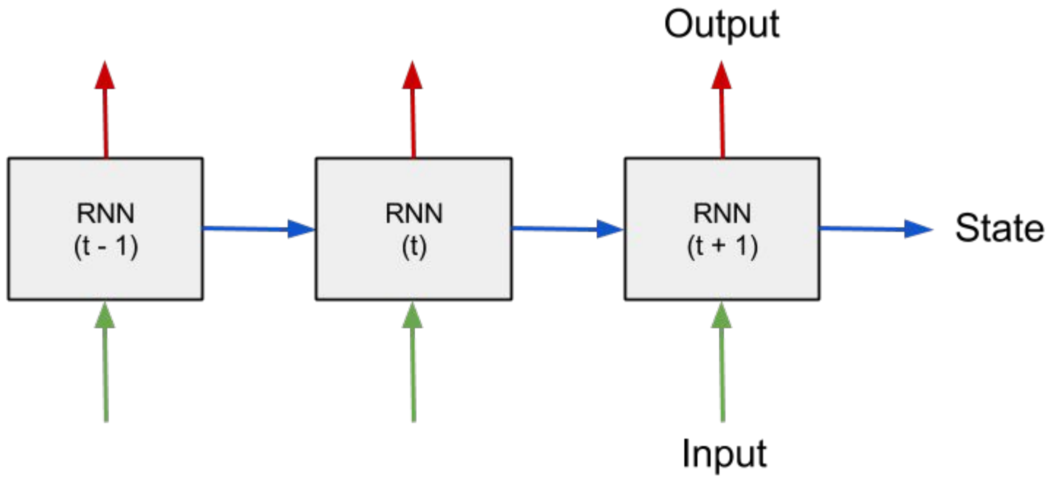
- Tokenization
- Lemmatization
- Part-of-Speech tagging
- Entity Recognition
- Word vectors

Matt Honnibal, currently at ExplosionAI

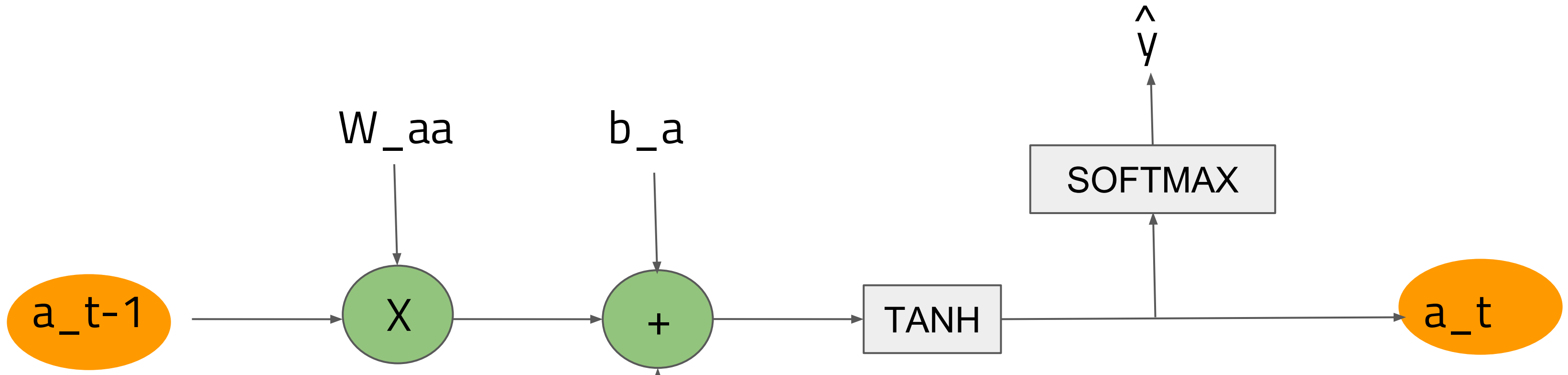
Next Notebook: Word Embeddings.ipynb

## Recurrent Neural Network (RNN)

- One RNN consists of several RNN cells
- Useful when data at several timesteps t is available (speech, music, video)

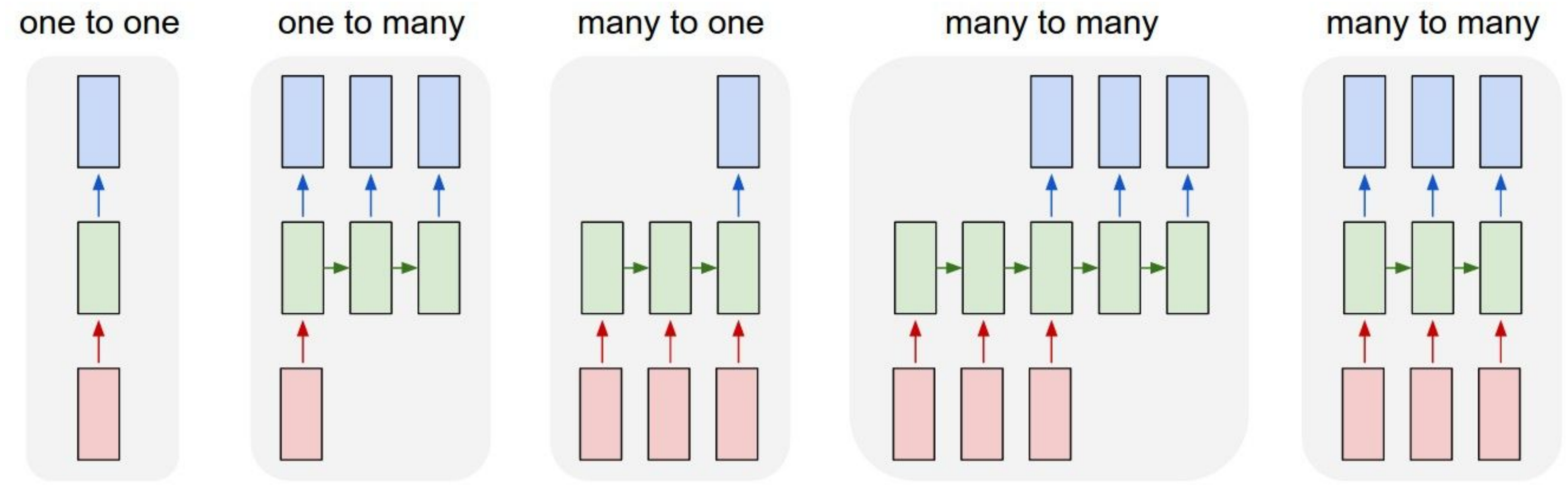


# Recurrent Neural Network Cell (RNN)



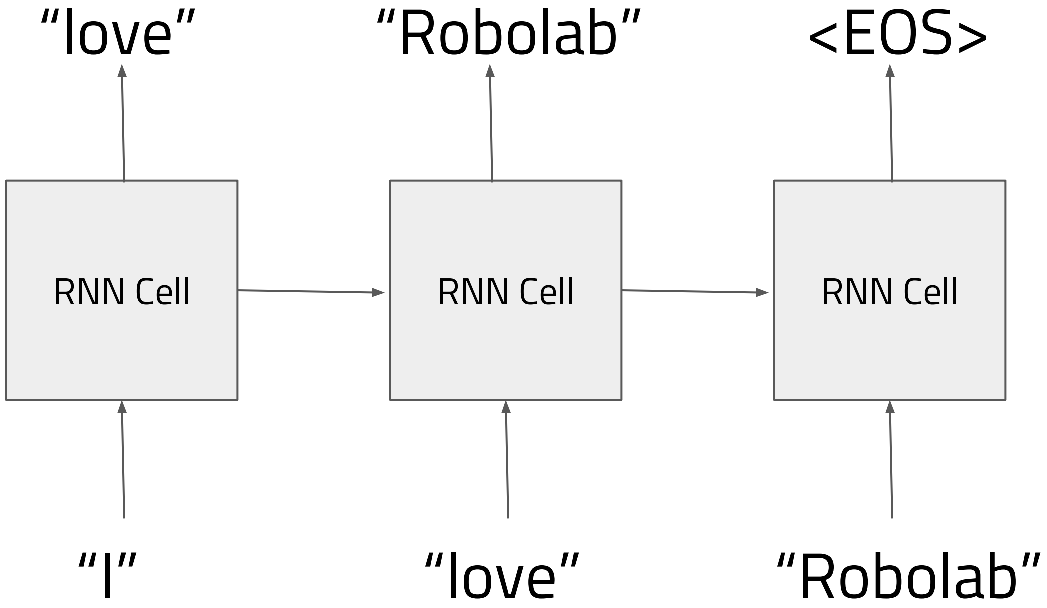
a_t-1: activation from previous cell
x_t: data at timestep t
a_t: activation of current cell
b_a: bias
W_aa: weights for activation
W_ax: weights for input x
$\hat{y}$: prediction

# RNN - combining several cells



Input layer RNN cell Output layer

# Generate text with RNNs

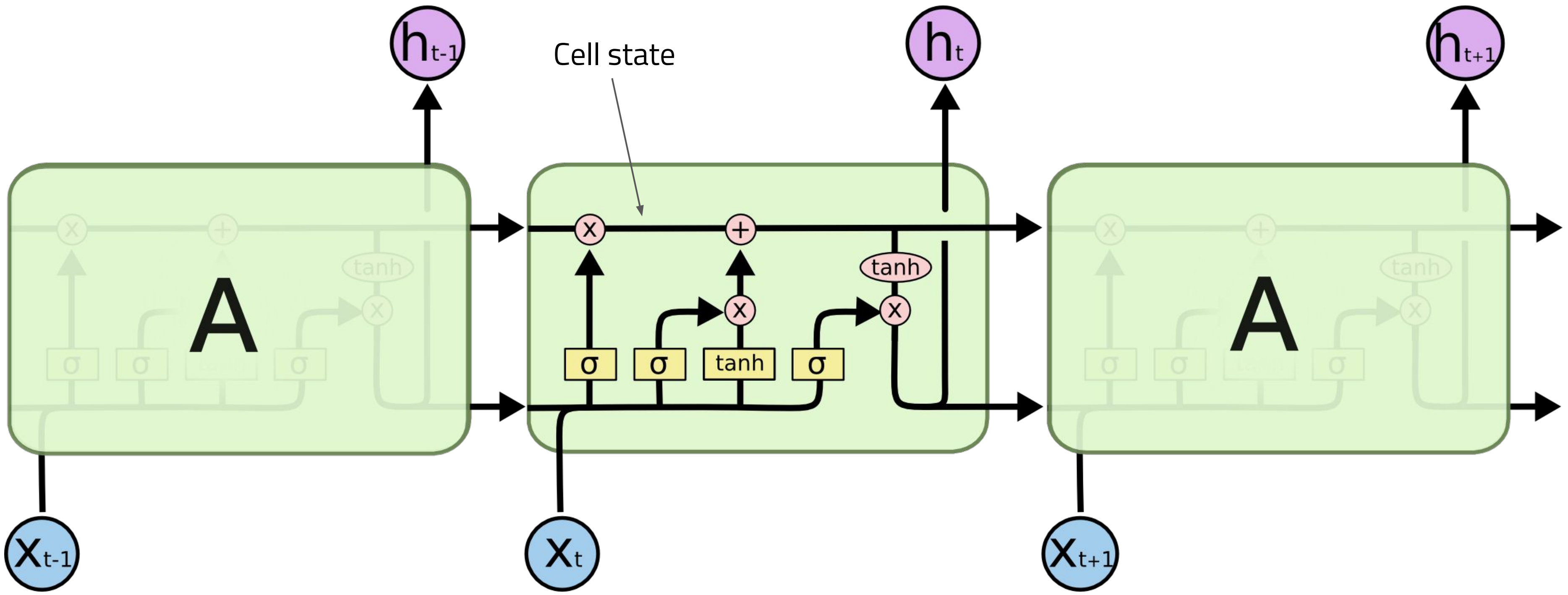


## Long short-term memory (LSTM)

- RNNs have problems storing a state in a_t over a long period of time
- LSTMs ~ RNNs on steroids
- LSTMs are much more complicated than RNNs
- LSTMs are quite old (1997)

Juergen Schmidhuber, SWISS AI LAB

# Long short-term memory (LSTM)



# Final Exercise

1. Plant seedling competition (image classification)
2. Rebuild Yolo v2
3. Play around with RNN/LSTMs