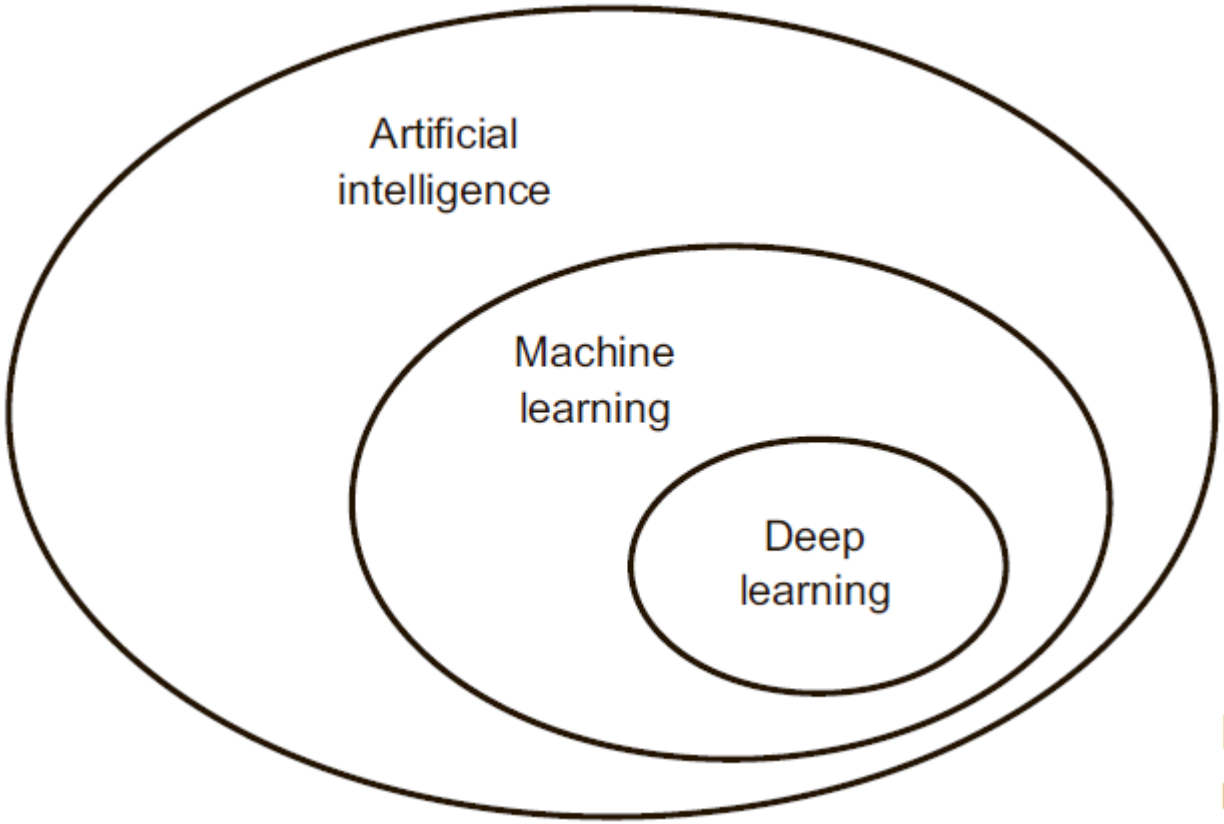
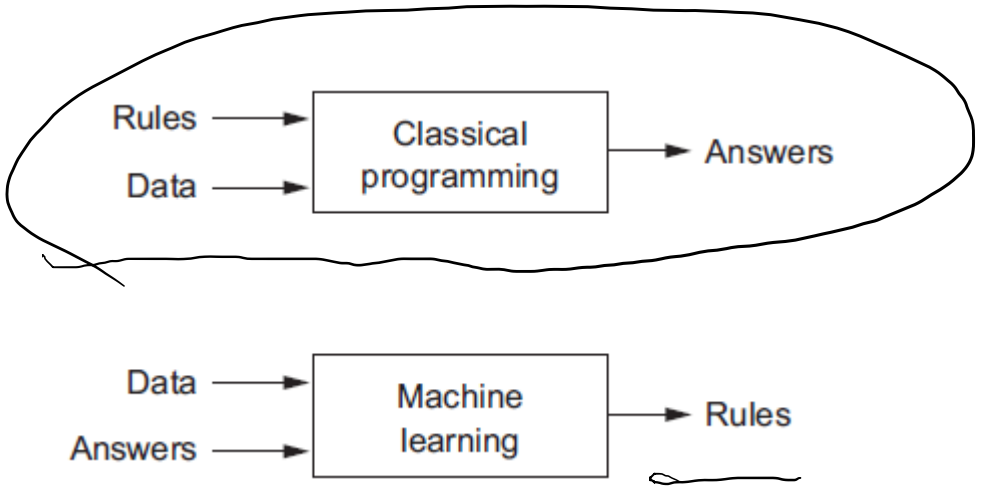
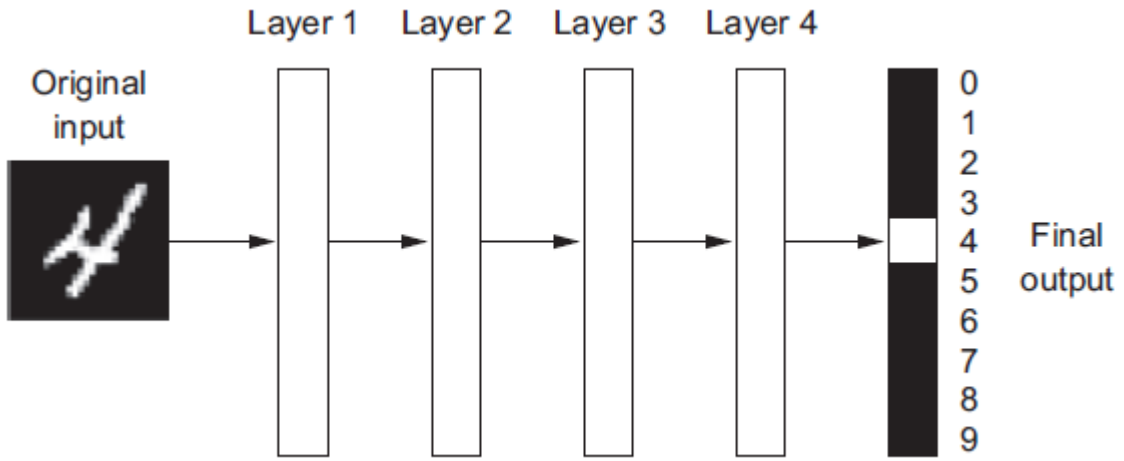
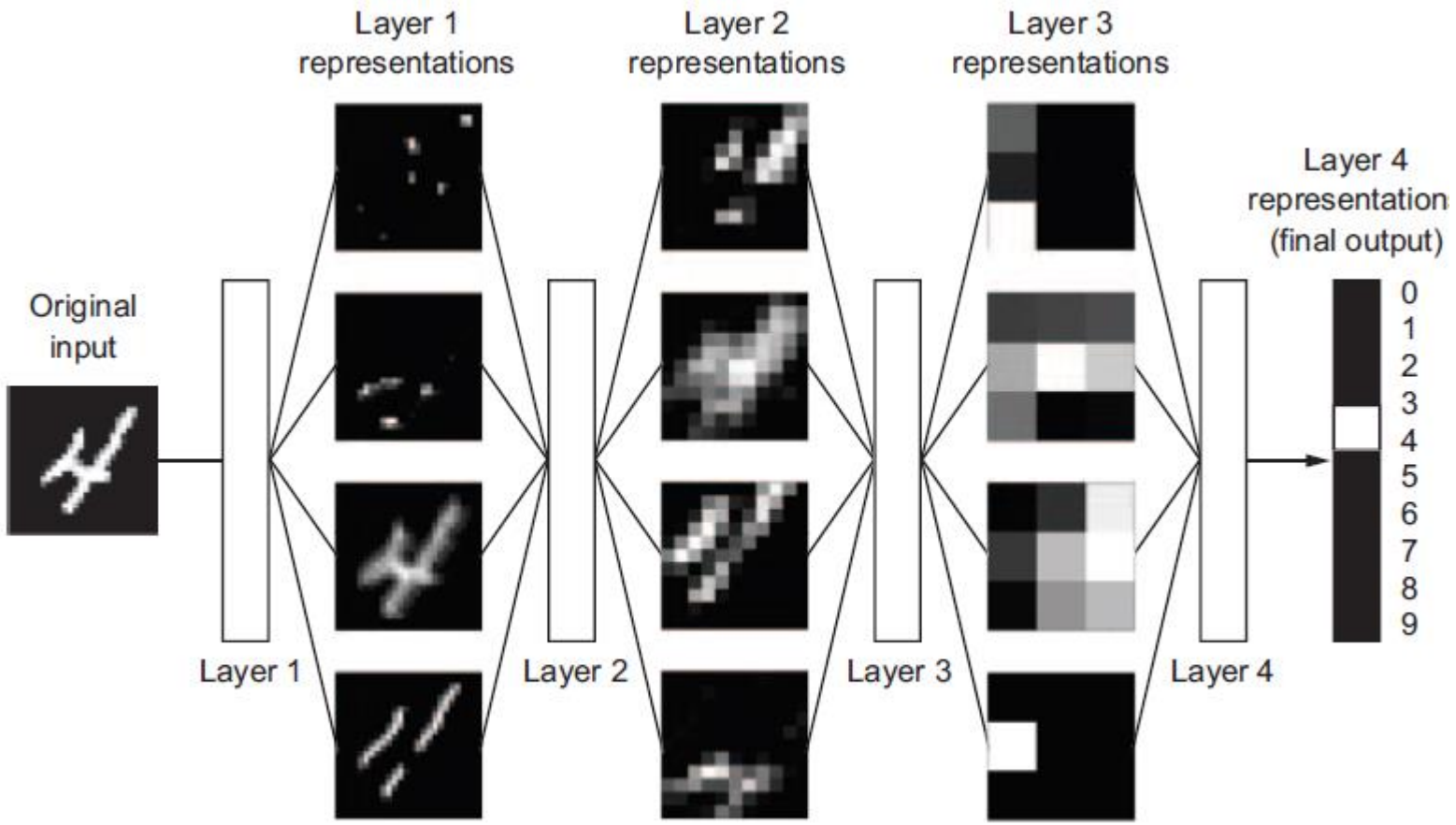
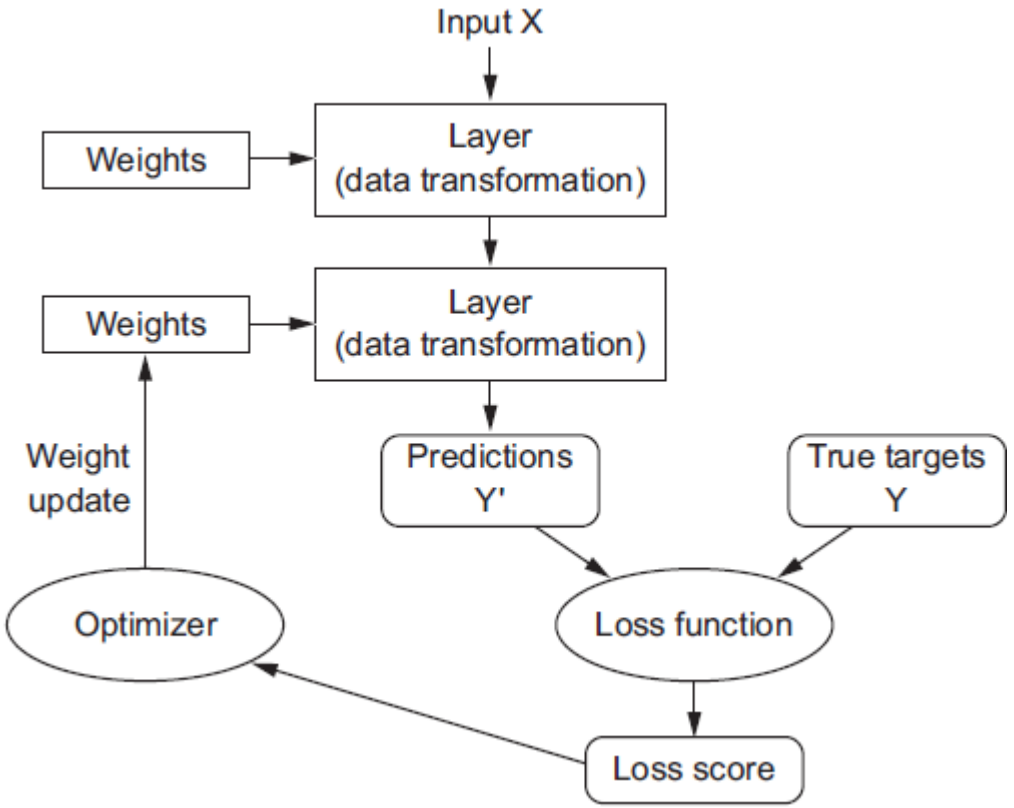
# Keras
# and basics of
# Deep Learning


Artificial
intelligence
Machine
learning
Deep
learning


Rules
Data
Classical
programming
Answers
Data
Answers
Machine
learning
Rules

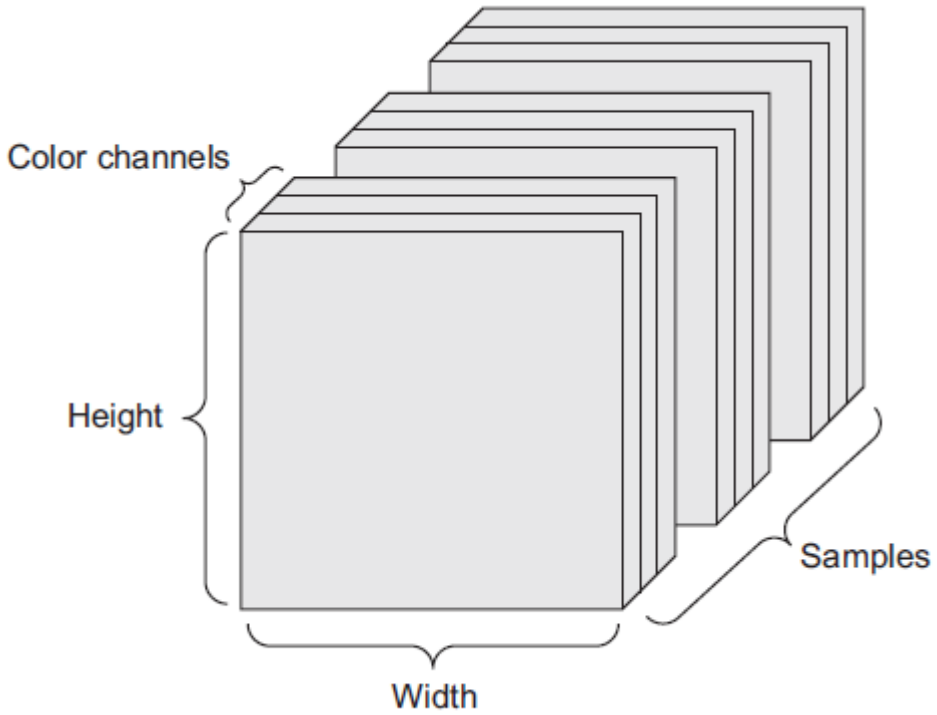

Layer 1
Layer 2
Layer 3
Layer 4
Original input
0
1
2
3
4
5
6
7
8
9
Final output


Layer 1 representations
Layer 2 representations
Layer 3 representations
Layer 4 representation (final output)
Original input
Layer 1
Layer 2
Layer 3
Layer 4
0
1
2
3
4
5
6
7
8
9

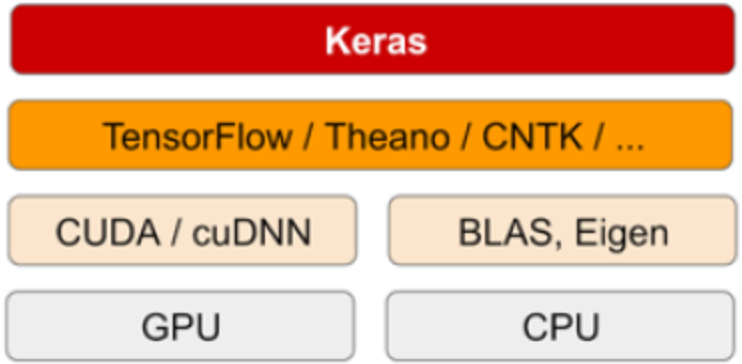

Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Weight
update
Predictions
Y'
True targets
Y
Optimizer
Loss function
Loss score

# Data representation as tensors

Image Data

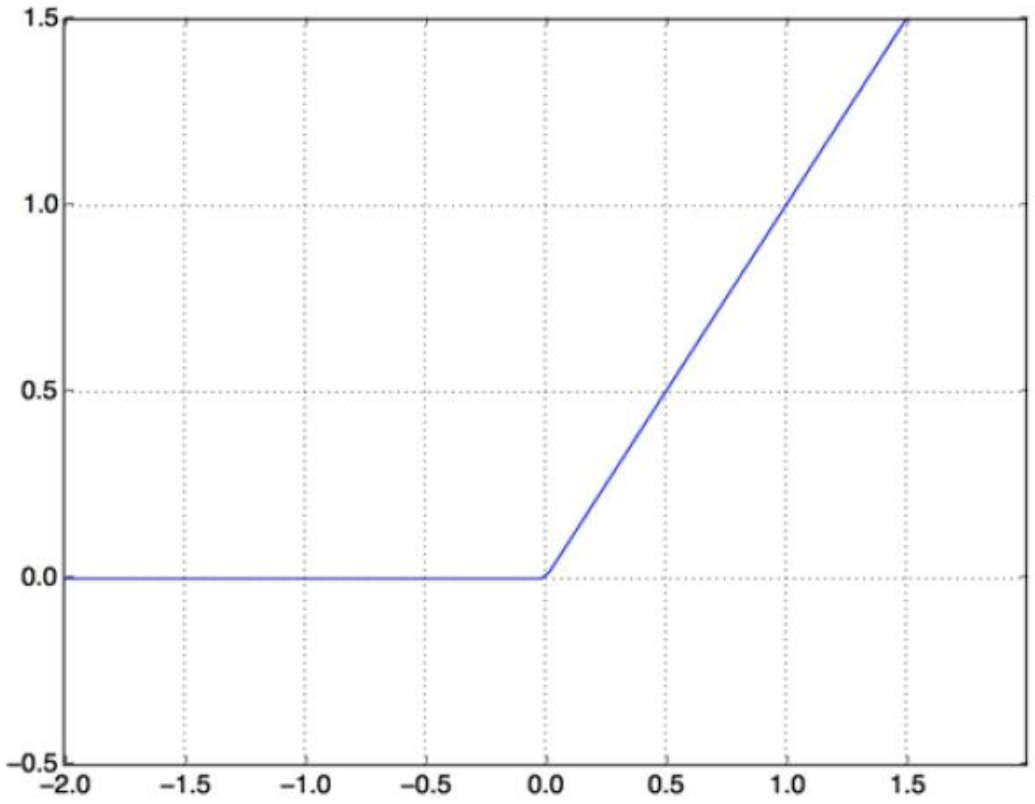


# Keras Library

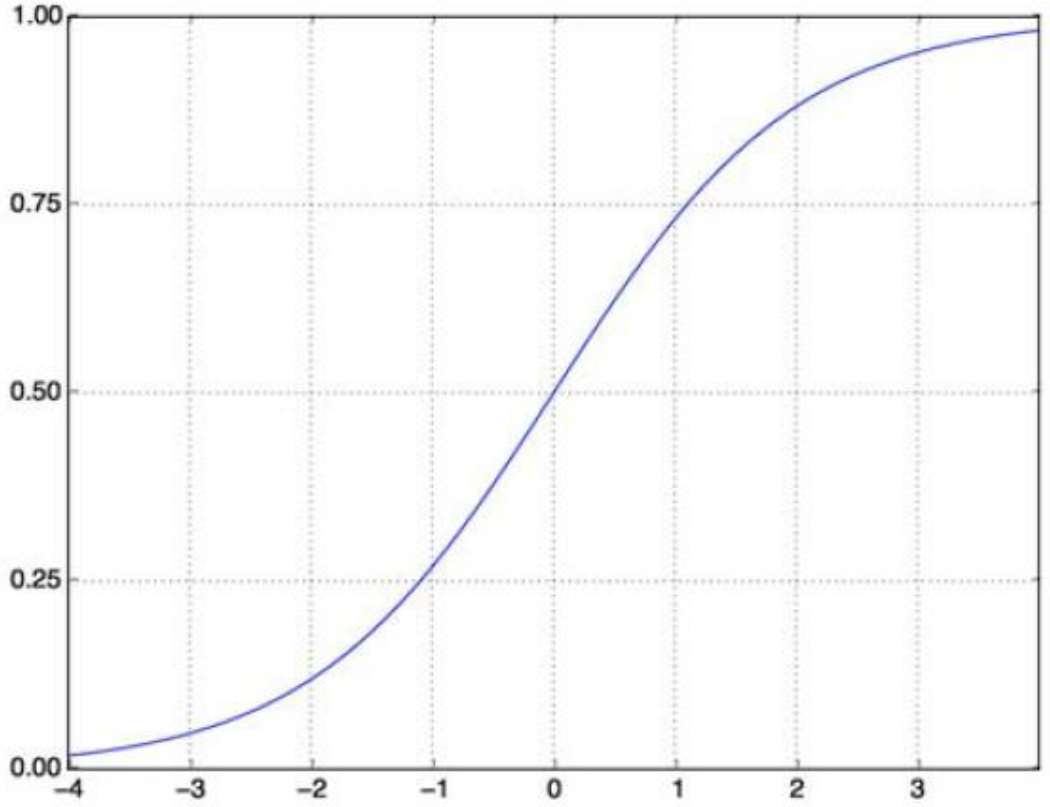


# Activation functions

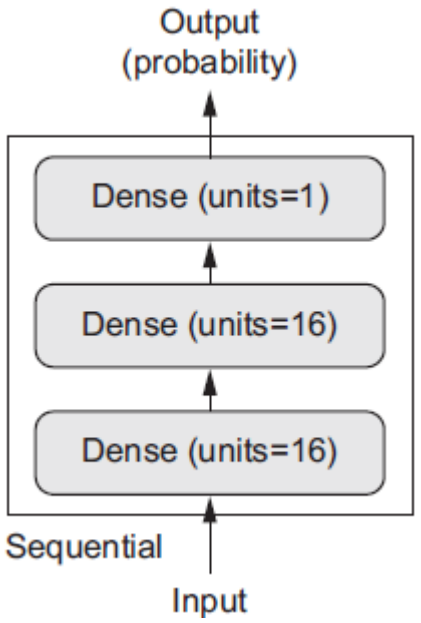
RELU




Sigmoid function



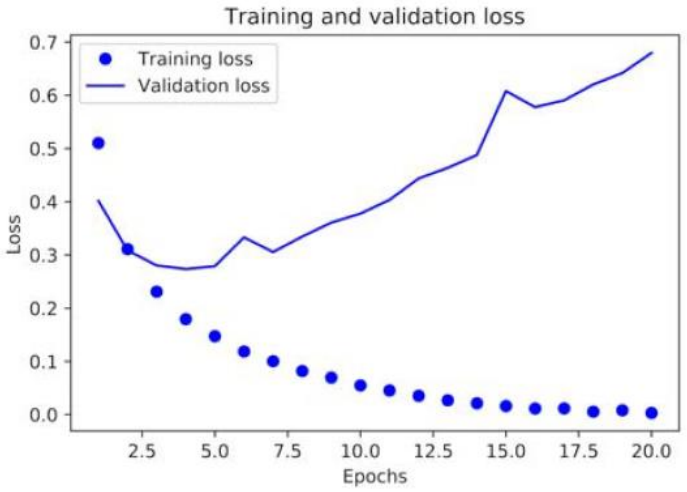
# Example. Three-layer network




```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Example: movie reviews

# Plotting training and validation loss

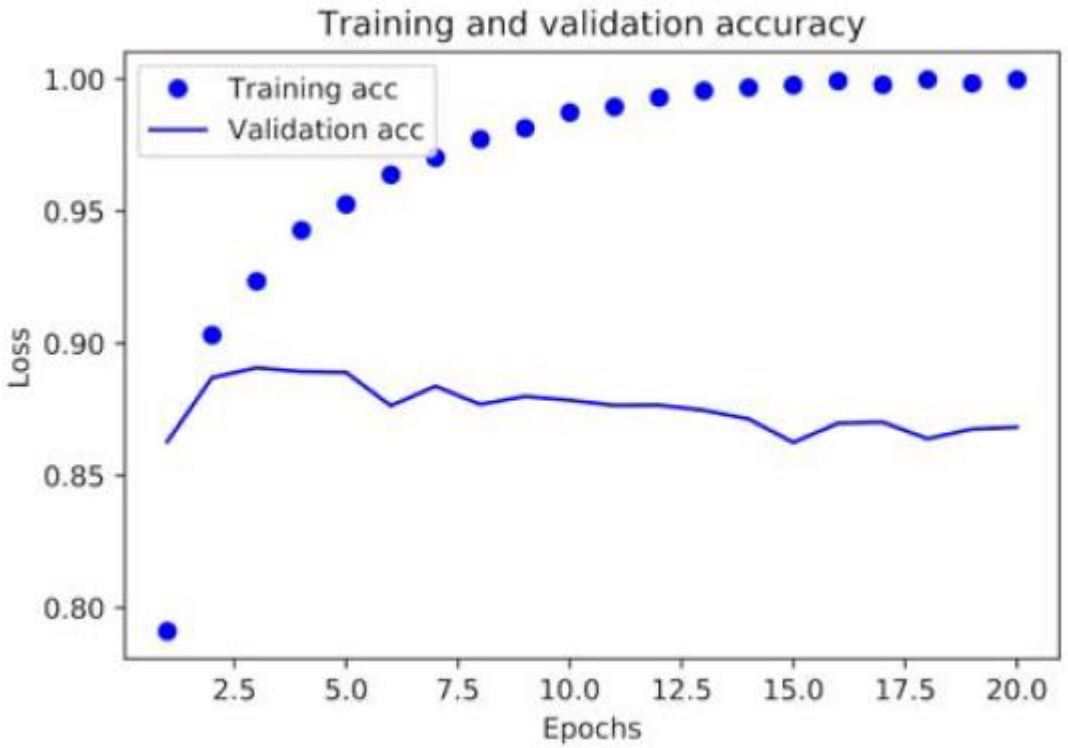



```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

"bo" is for "blue dot."

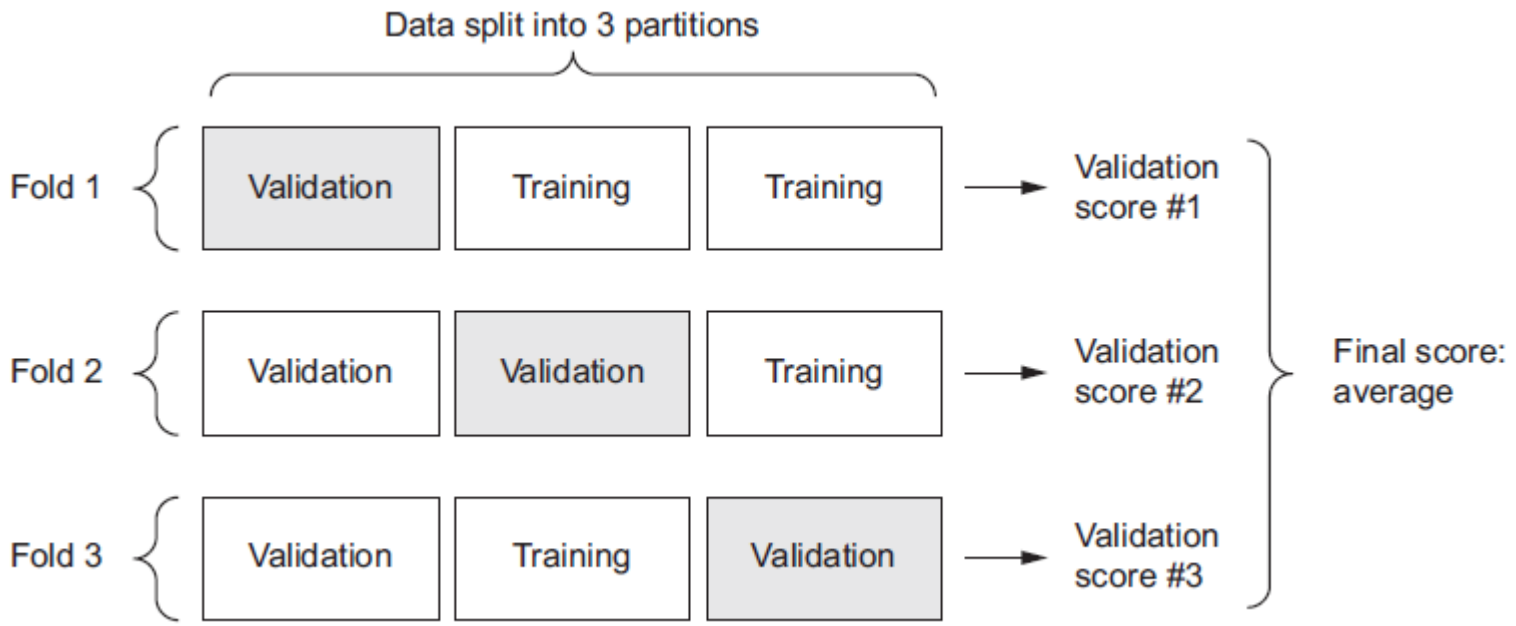
"b" is for "solid blue line."

# Training and validation accuracy




```
plt.clf()                    <-- Clears the figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

# K-fold cross-validation




Example: regression problem

# Data preprocessing

- Vectorization
- Normalization
- Handling missing values (skikit Imputer class)
- Data augumentation

# Data preprocessing. Vectorization

- All inputs and targets in a neural network must be tensors of floating-point data (or, in specific cases, tensors of integers). Whatever data you need to process—sound, images, text—you must first turn into tensors, a step called *data vectorization*.
- Example: one-hot-encoding, word2vec

# Data preprocessing. Value normalization

- You have to normalize each feature independently so that it had a standard deviation of 1 and a mean of 0.
- In general, it isn't safe to feed into a neural network data that takes relatively large values (for example, multidigit integers, which are much larger than the initial values taken by the weights of a network) or data that is heterogeneous (for example, data where one feature is in the range 0–1 and another is in the range 100–200). Doing so can trigger large gradient updates that will prevent the network from converging. To make learning easier for your network, your data should have the following characteristics:
  - *Take small values*—Typically, most values should be in the 0–1 range.
  - *Be homogenous*—That is, all features should take values in roughly the same range.

```
x -= x.mean(axis=0)
x /= x.std(axis=0)
```

Assuming x is a 2D data matrix of shape (samples, features)

# Data preprocessing. Missing values

- In general, with neural networks, it's safe to input missing values as 0, with the condition that 0 isn't already a meaningful value. The network will learn from exposure to the data that the value 0 means *missing data* and will start ignoring the value.

Note that if you're expecting missing values in the test data, but the network was trained on data without any missing values, the network won't have learned to ignore missing values! In this situation, you should artificially generate training samples with missing entries: copy some training samples several times, and drop some of the features that you expect are likely to be missing in the test data.

# Feature engineering

- *Feature engineering* is the process of using your own knowledge about the data and about the machine-learning algorithm at hand to make the algorithm work better by applying hardcoded (nonlearned) transformations to the data before it goes into the model:
  - More elegant solution
  - Far less data

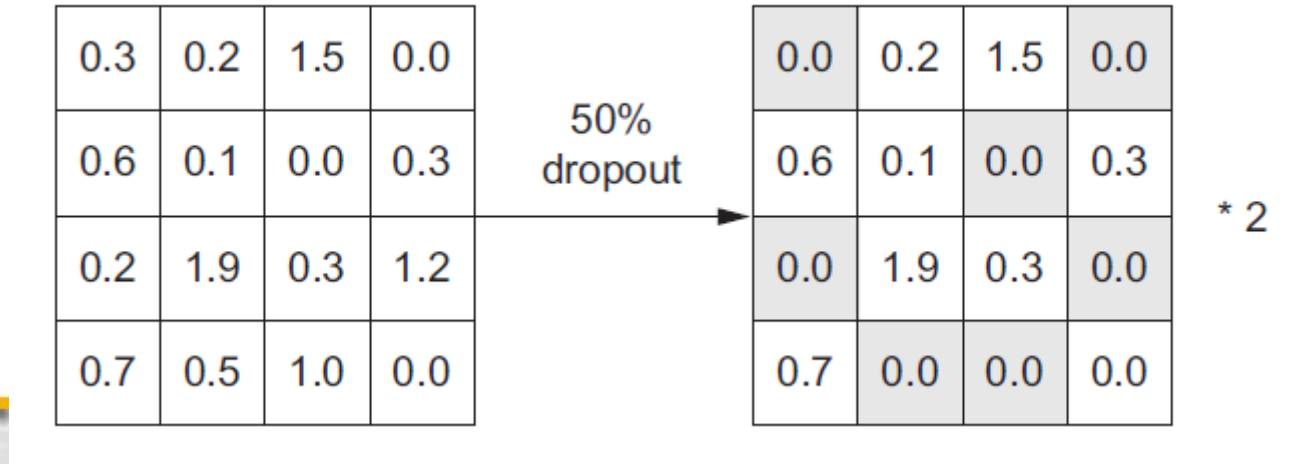
| | | |
|---|---|---|
| Raw data: pixel grid |  |  |
| Better features: clock hands' coordinates | {x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0} | {x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32} |
| Even better features: angles of clock hands | theta1: 45 theta2: 0 | theta1: 90 theta2: 140 |

# Overfitting avoidance

- Take more data
- Reduce the network size
- Regularization (L1, L2)

```
from keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

- Dropout



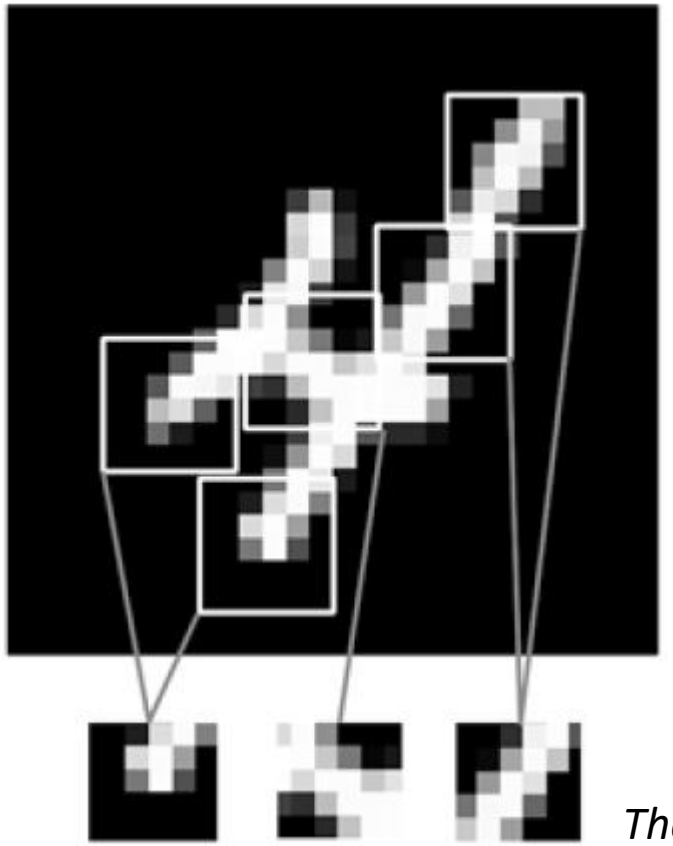
# The universal workflow of machine learning

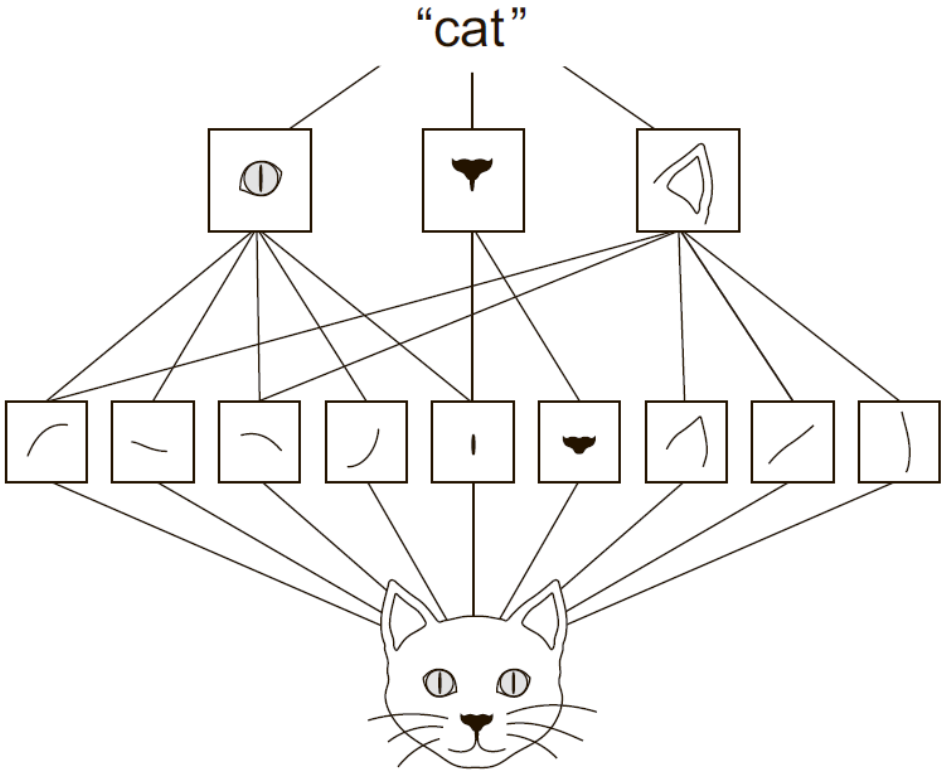
1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Deciding on an evaluation protocol
   - *Maintaining a hold-out validation set*
   - *Doing K-fold cross-validation*
   - *Doing iterated K-fold validation*
4. Preparing your data
5. Developing a model that does better than a baseline
   - *Last-layer activation*
   - *Loss function*
   - *Optimization configuration*
6. Scaling up: developing a model that overfits (add layers, make layers bigger, train more epochs)
7. Regularizing your model and tuning your hyperparameters

# Last-layer activation

| Problem type | Last-layer activation | Loss function |
|---|---|---|
| Binary classification | `sigmoid` | `binary_crossentropy` |
| Multiclass, single-label classification | `softmax` | `categorical_crossentropy` |
| Multiclass, multilabel classification | `sigmoid` | `binary_crossentropy` |
| Regression to arbitrary values | None | `mse` |
| Regression to values between 0 and 1 | `sigmoid` | `mse` or `binary_crossentropy` |

# Deep Learning. Convolution

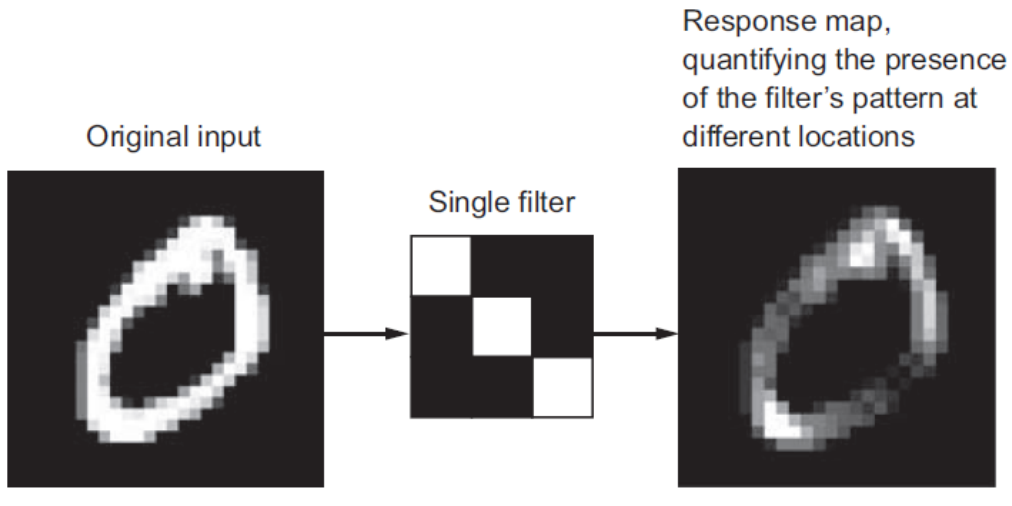




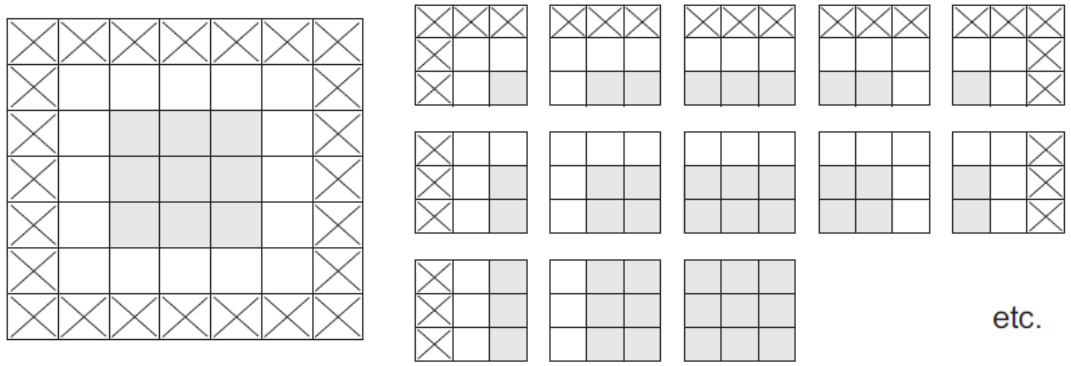

*The patterns they learn are translation invariant.*
*They can learn spatial hierarchies of patterns*

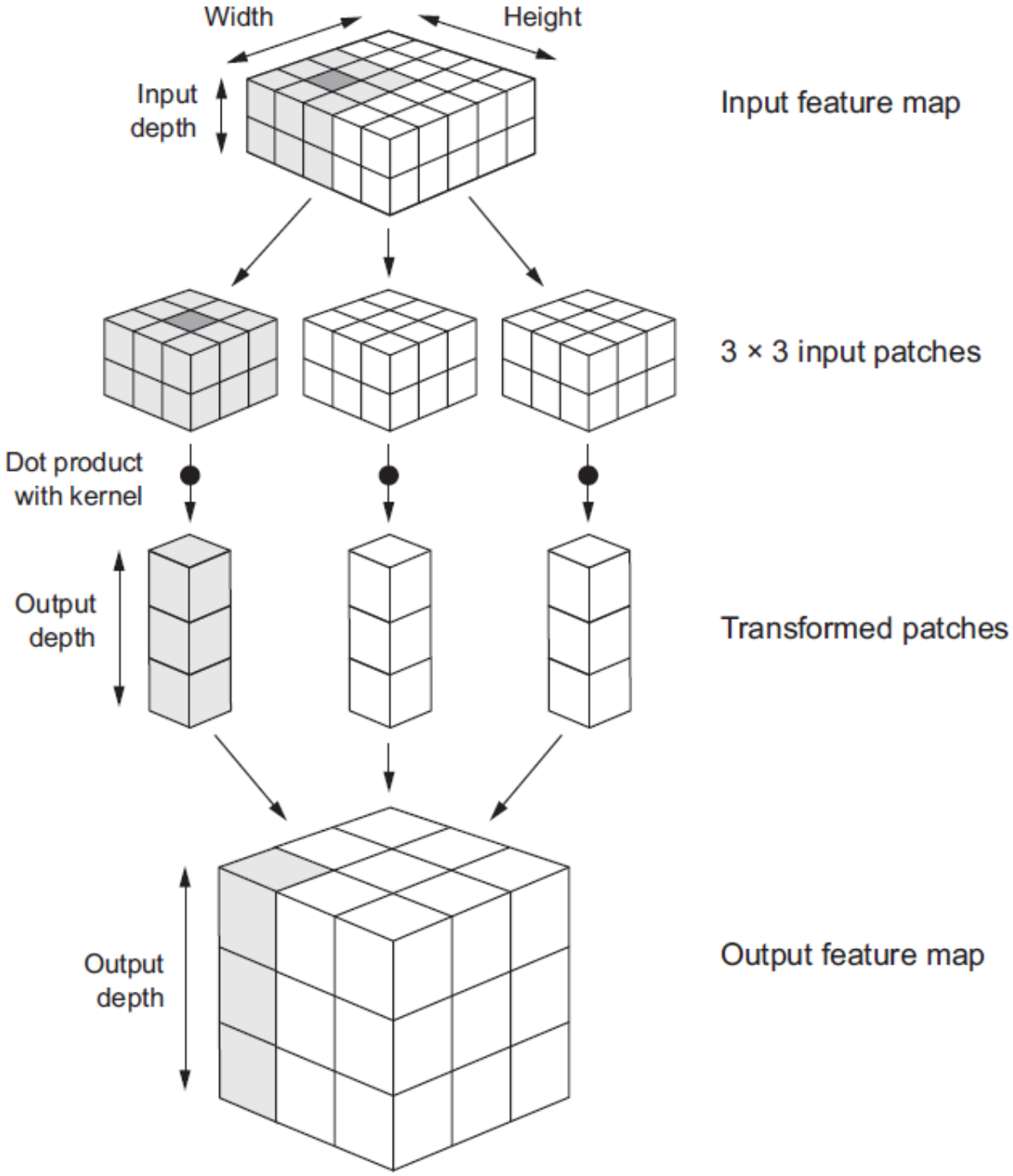
# Convolution properties

- *Size of the patches extracted from the inputs*—These are typically 3 × 3 or 5 × 5. In the example, they were 3 × 3, which is a common choice.
- *Depth of the output feature map*—The number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

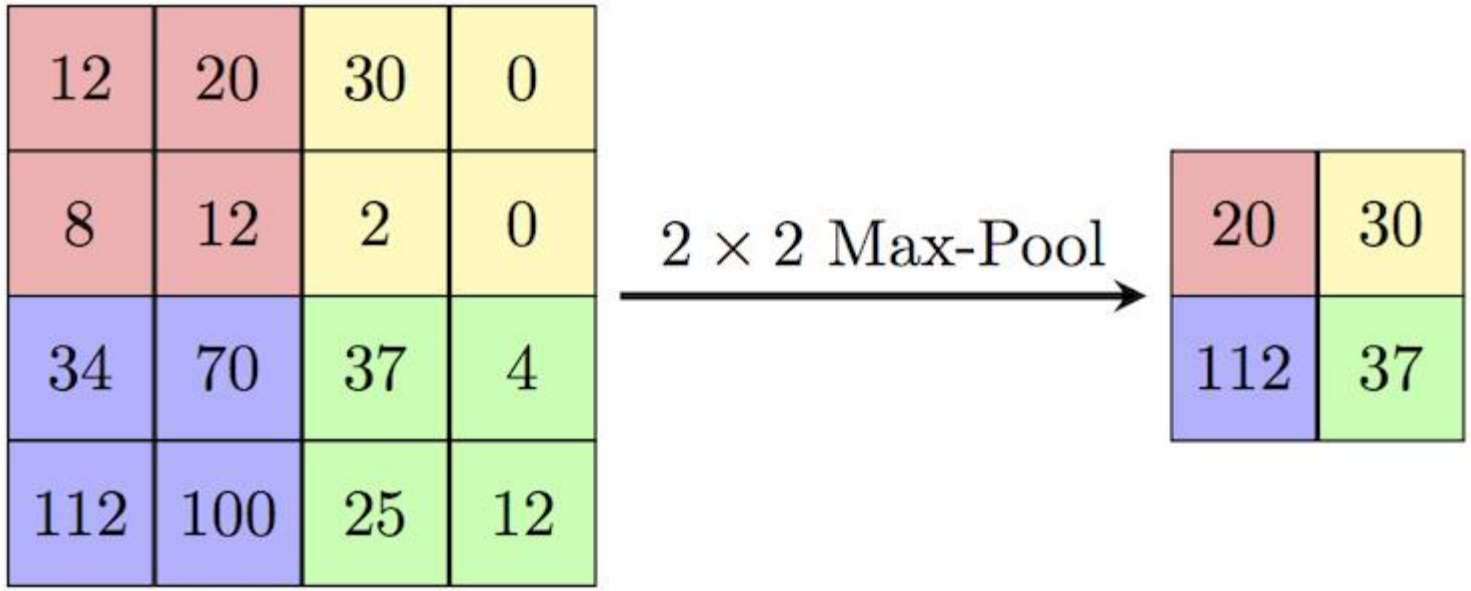


# How it works?

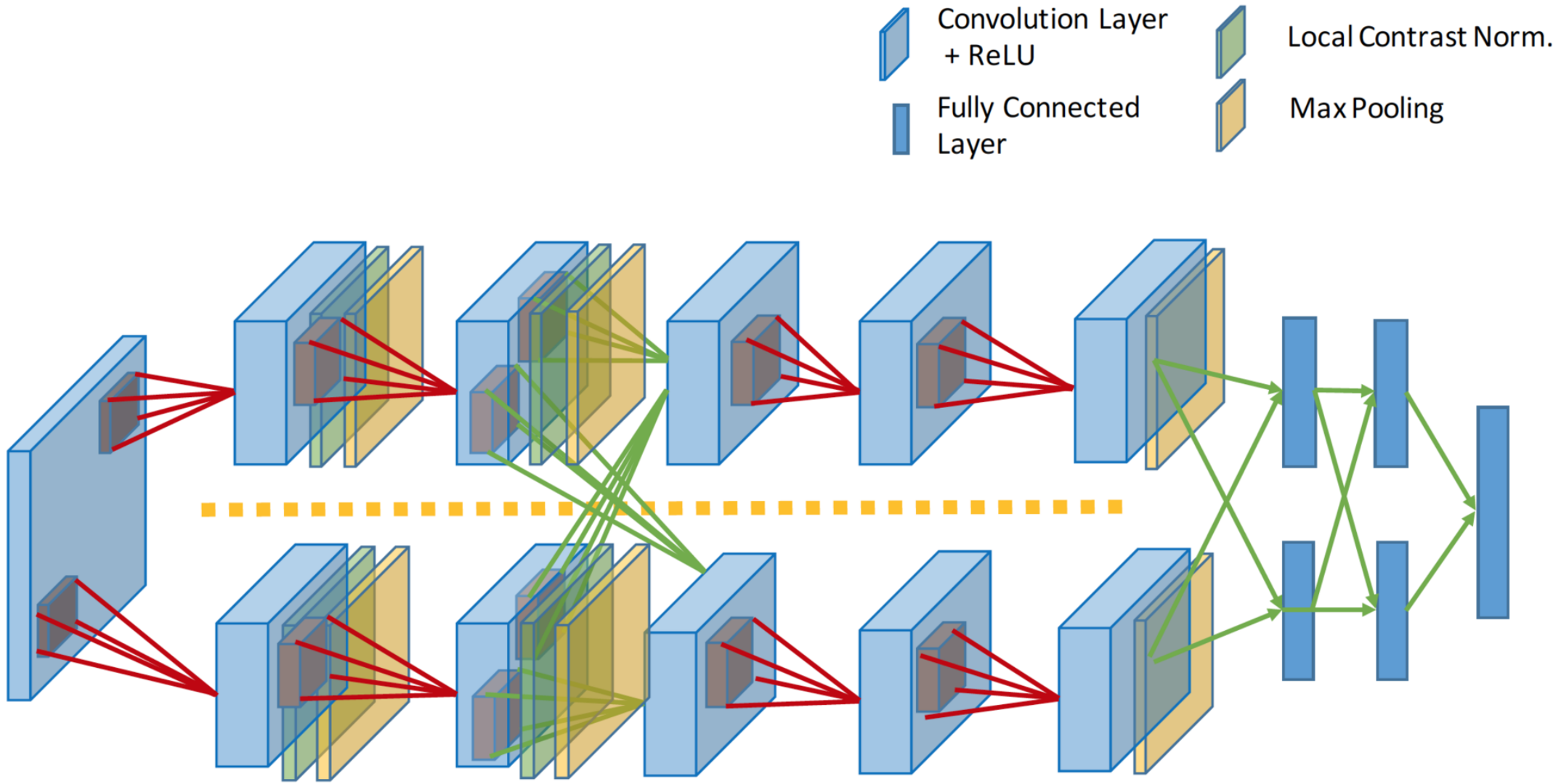

etc.


Width
Height
Input depth
Input feature map
3 × 3 input patches
Dot product with kernel
Output depth
Transformed patches
Output depth
Output feature map

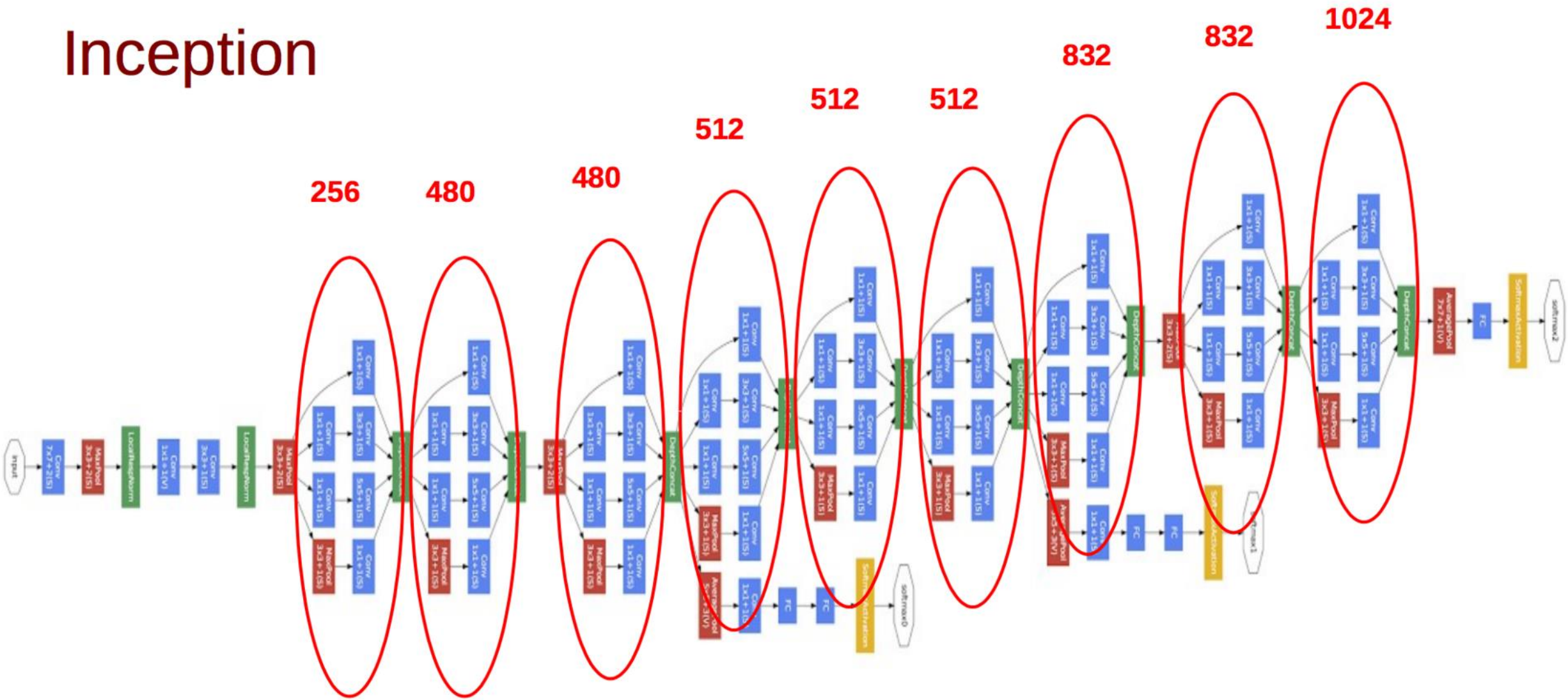
# MaxPooling



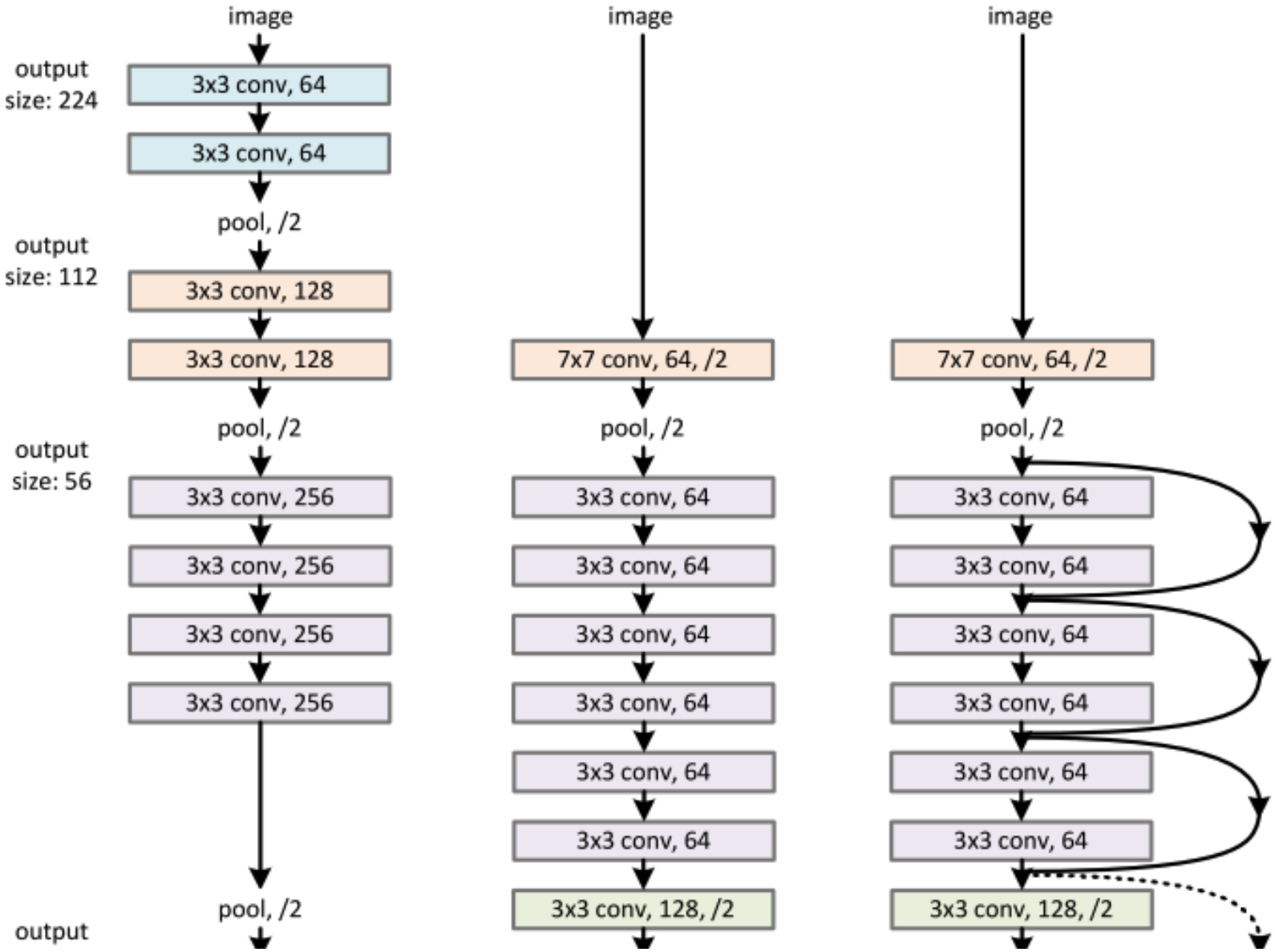
# AlexNet (2012)



# Inception от Google (17 Sep 2014)

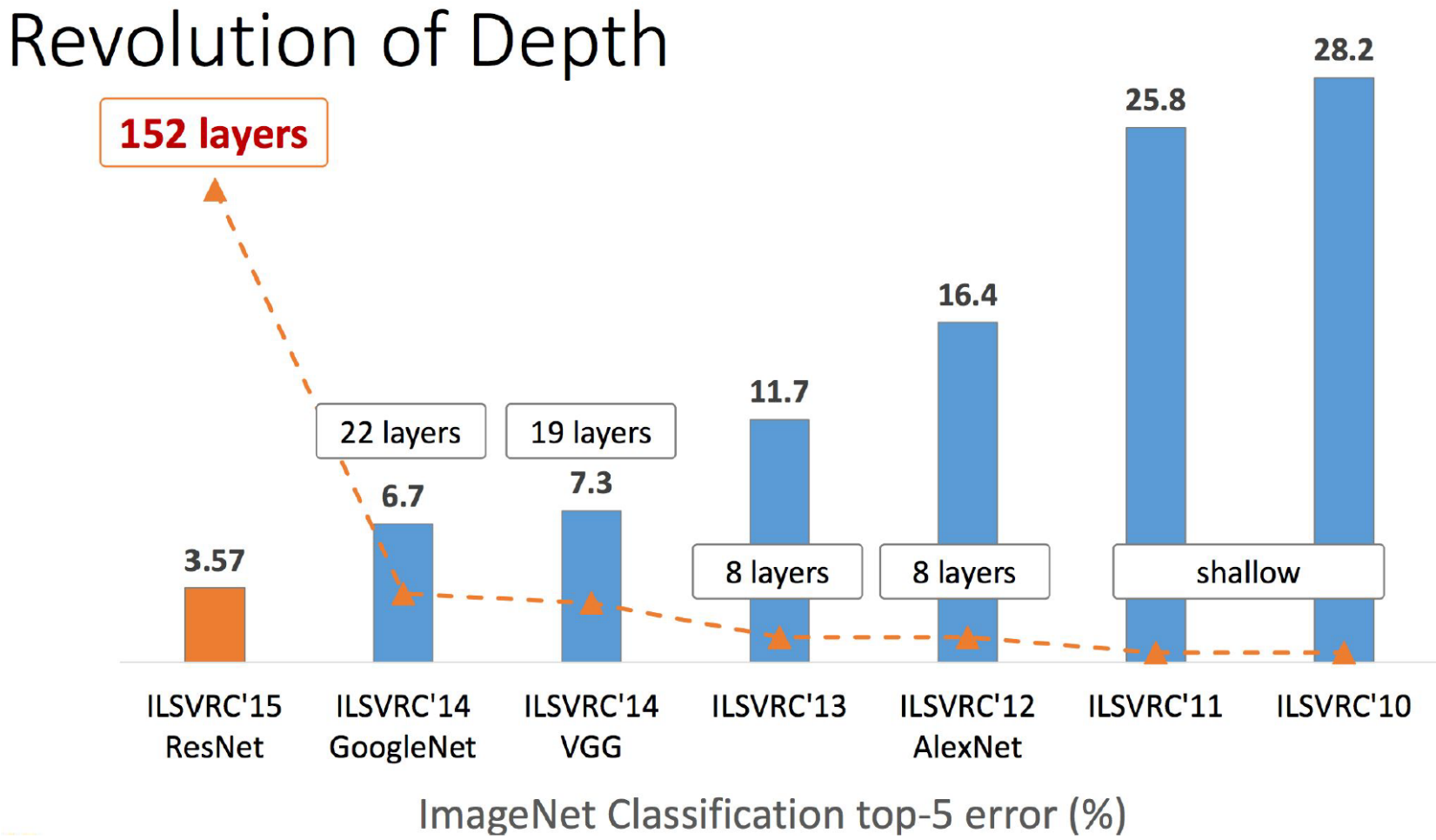


# ResNET (10 Dec 2015)



# ResNET (10 Dec 2015)

Модель, с которой авторы победили на ImageNet, содержит меньше параметров, чем 19-слойный VGG, при глубине 152 слоя:



# Word2vec

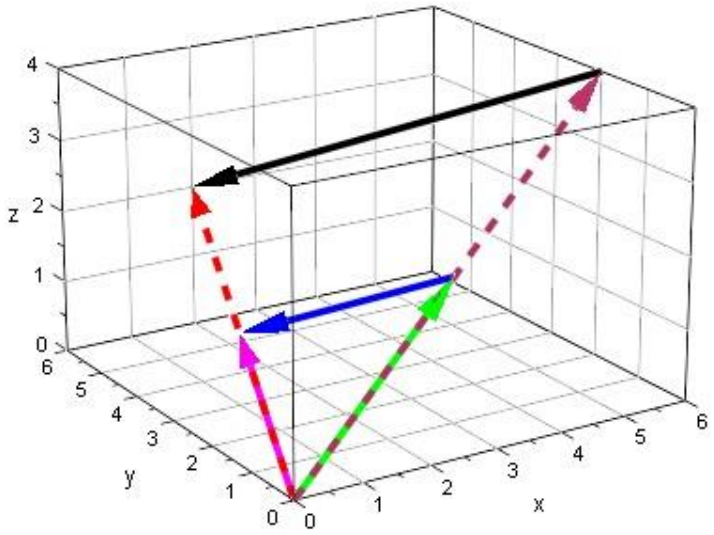
# word2vec

One of the most successful ideas of modern statistical NLP

| | | |
|---|---|---|
| government debt problems turning into | banking | crises as has happened in |
| saying that Europe needs unified | banking | regulation to replace the hodgepodge |

↖ These words will represent *banking* ↗

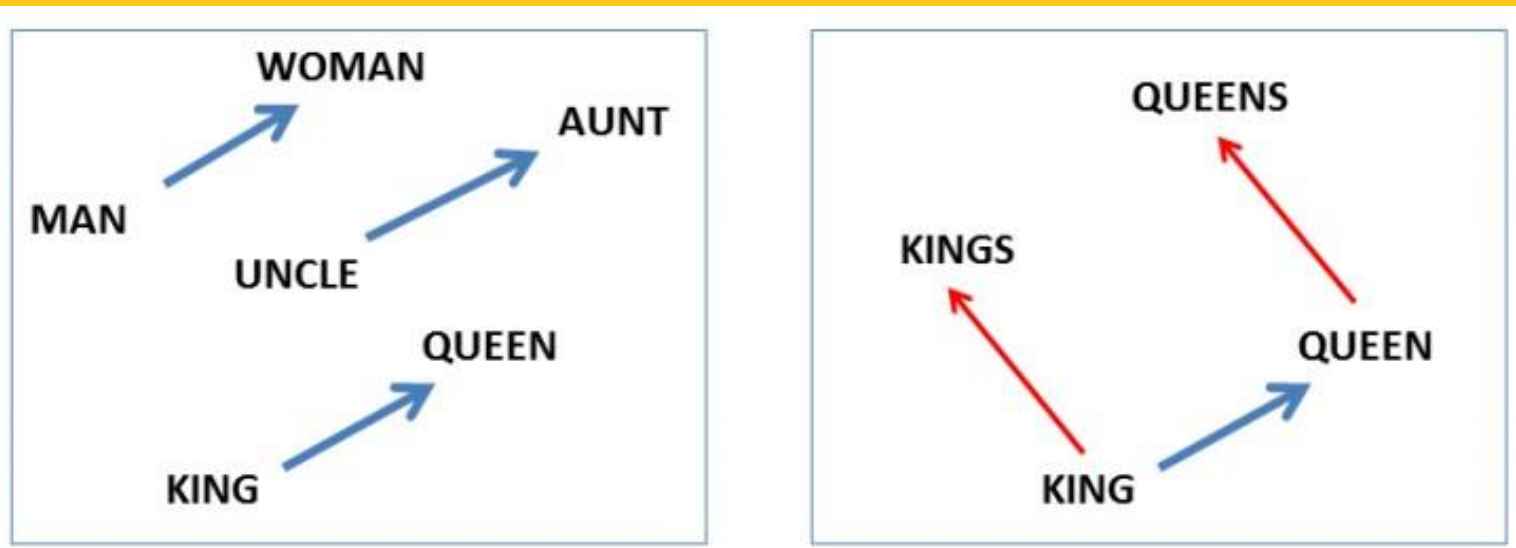
Quoted after Socher




*linguistics* =

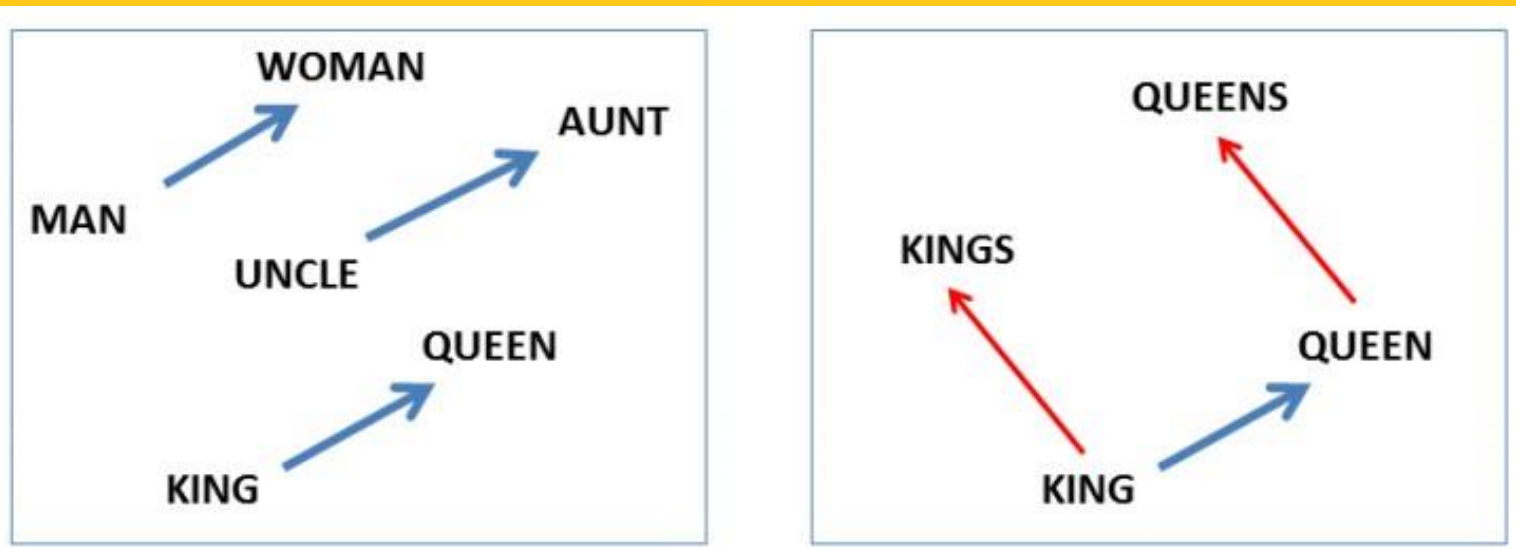
$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Quoted after Socher



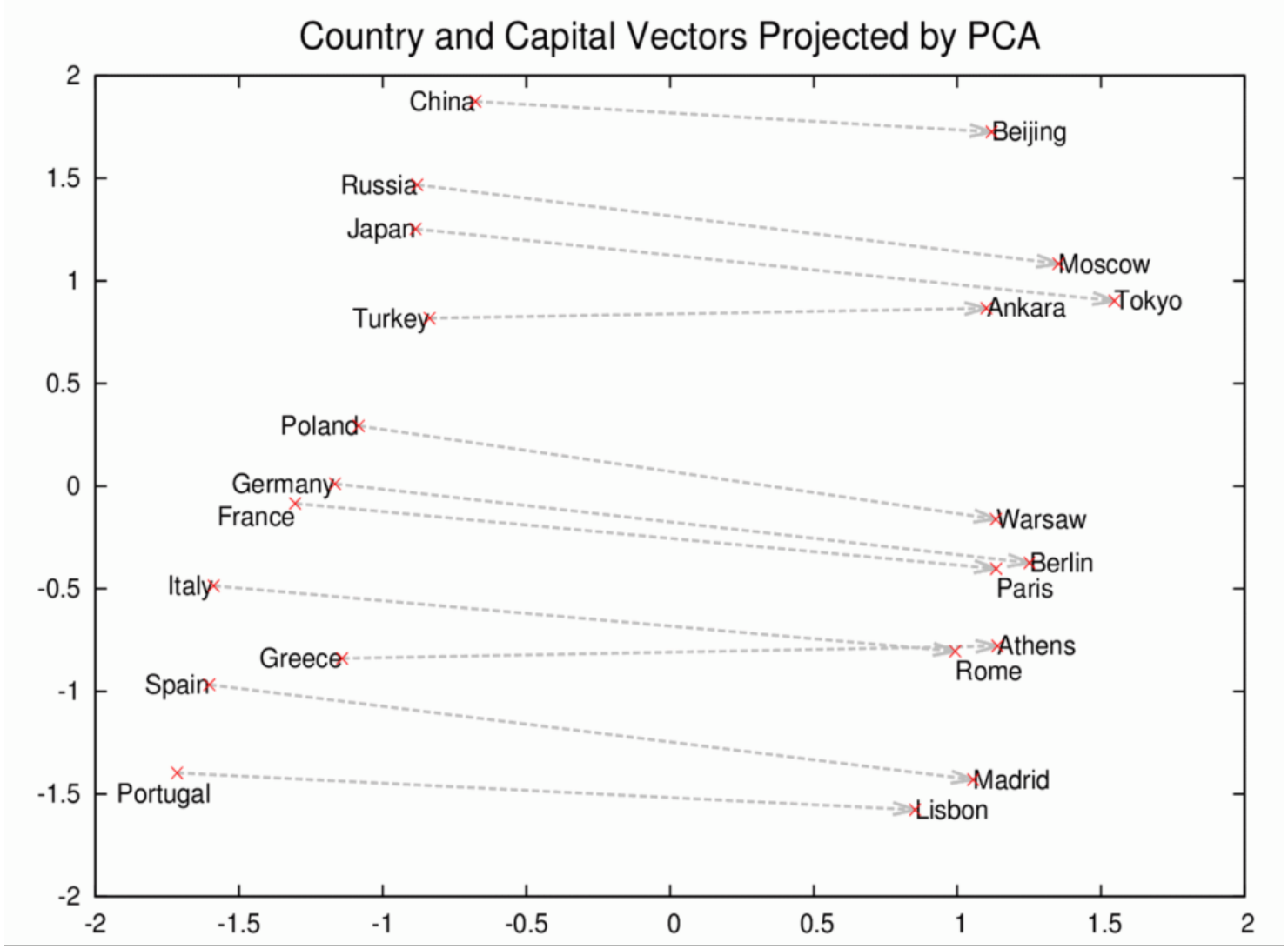

(Mikolov et al., NAACL HLT, 2013)

Vectors are directions in space
Vectors can encode relationships




(Mikolov et al., NAACL HLT, 2013)

**man** is to **woman** as **king** is to **?**


Country and Capital Vectors Projected by PCA
China
Beijing
Russia
Moscow
Japan
Tokyo
Turkey
Ankara
Poland
Warsaw
Germany
Berlin
France
Paris
Italy
Rome
Greece
Athens
Spain
Madrid
Portugal
Lisbon
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

# word2vec

- by Mikolov, Sutskever, Chen, Corrado and Dean at Google
- NAACL 2013
- takes a **text corpus** as **input** and produces the **word vectors** as **output**

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

## Sweden

Similar words

| Word | Cosine distance |
|---|---|
| norway | 0.760124 |
| denmark | 0.715460 |
| finland | 0.620022 |
| switzerland | 0.588132 |
| belgium | 0.585835 |
| netherlands | 0.574631 |
| iceland | 0.562368 |
| estonia | 0.547621 |
| slovenia | 0.531408 |
| floorball_federation | 0.529570 |
| luxembourg | 0.529477 |
| czech_republic | 0.528778 |
| slovakia | 0.526340 |
| romania | 0.524281 |
| kista | 0.522488 |
| helsinki_vantaa | 0.519936 |
| swedish | 0.519901 |
| balrog_ik | 0.514556 |
| portugal | 0.502495 |
| russia | 0.500196 |
| slovakia_slovenia | 0.496051 |
| ukraine | 0.495712 |
| chur | 0.484225 |
| thailand_togo | 0.479172 |
| crimea_ukraine | 0.478596 |
| panama_papua | 0.477126 |
| latvia | 0.477058 |

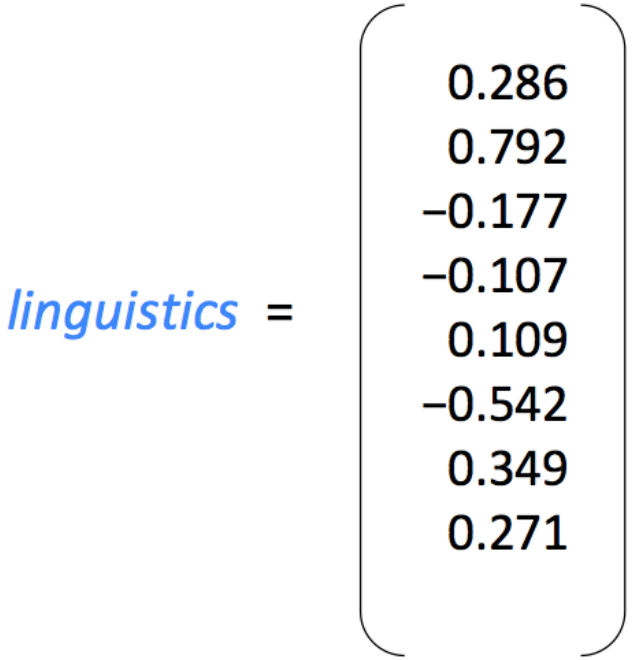
# Harvard

Similar words

| Word | Cosine distance |
|---|---|
| yale | 0.638970 |
| cambridge | 0.612665 |
| university | 0.597709 |
| faculty | 0.588422 |
| harvey_mudd | 0.578338 |
| johns_hopkins | 0.575645 |
| graduate | 0.570294 |
| undergraduate | 0.565881 |
| professor | 0.563657 |
| mcgill | 0.562168 |
| ph_d | 0.558665 |
| california_berkeley | 0.555539 |
| yale_university | 0.550480 |
| harvard_crimson | 0.549848 |
| princeton | 0.544070 |
| college | 0.542838 |
| oxford | 0.531948 |
| barnard_college | 0.530800 |
| professors | 0.529959 |
| princeton_university | 0.529763 |
| ucl | 0.527395 |
| doctorates | 0.526292 |
| doctoral | 0.523317 |
| cambridge_massachusetts | 0.519657 |
| juris_doctor | 0.518845 |
| graduate_student | 0.518815 |
| postgraduate | 0.515757 |

# word2vec

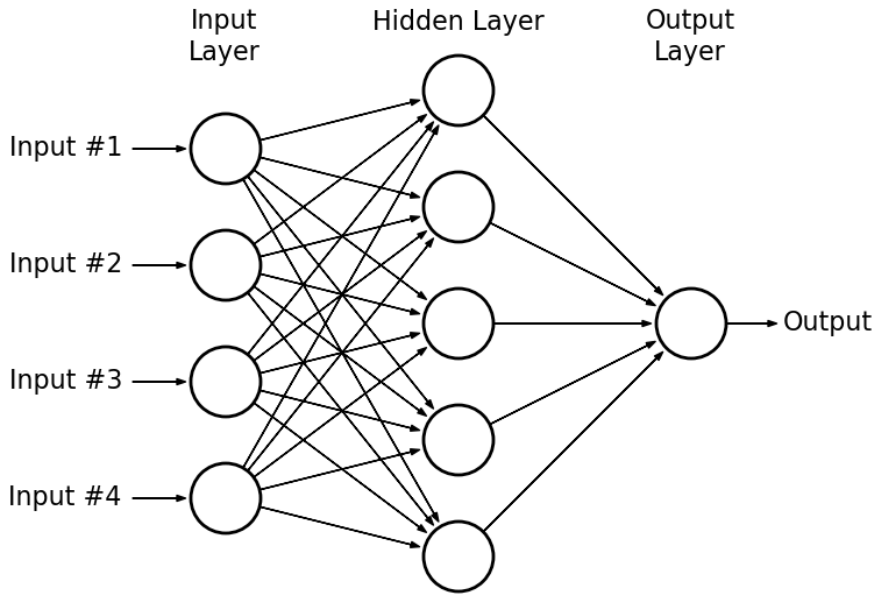
- **word meaning** and **relationships** between words are **encoded spatially**
- two main learning algorithms in word2vec:
  **continuous bag-of-words** and
  **continuous skip-gram**



# Goal

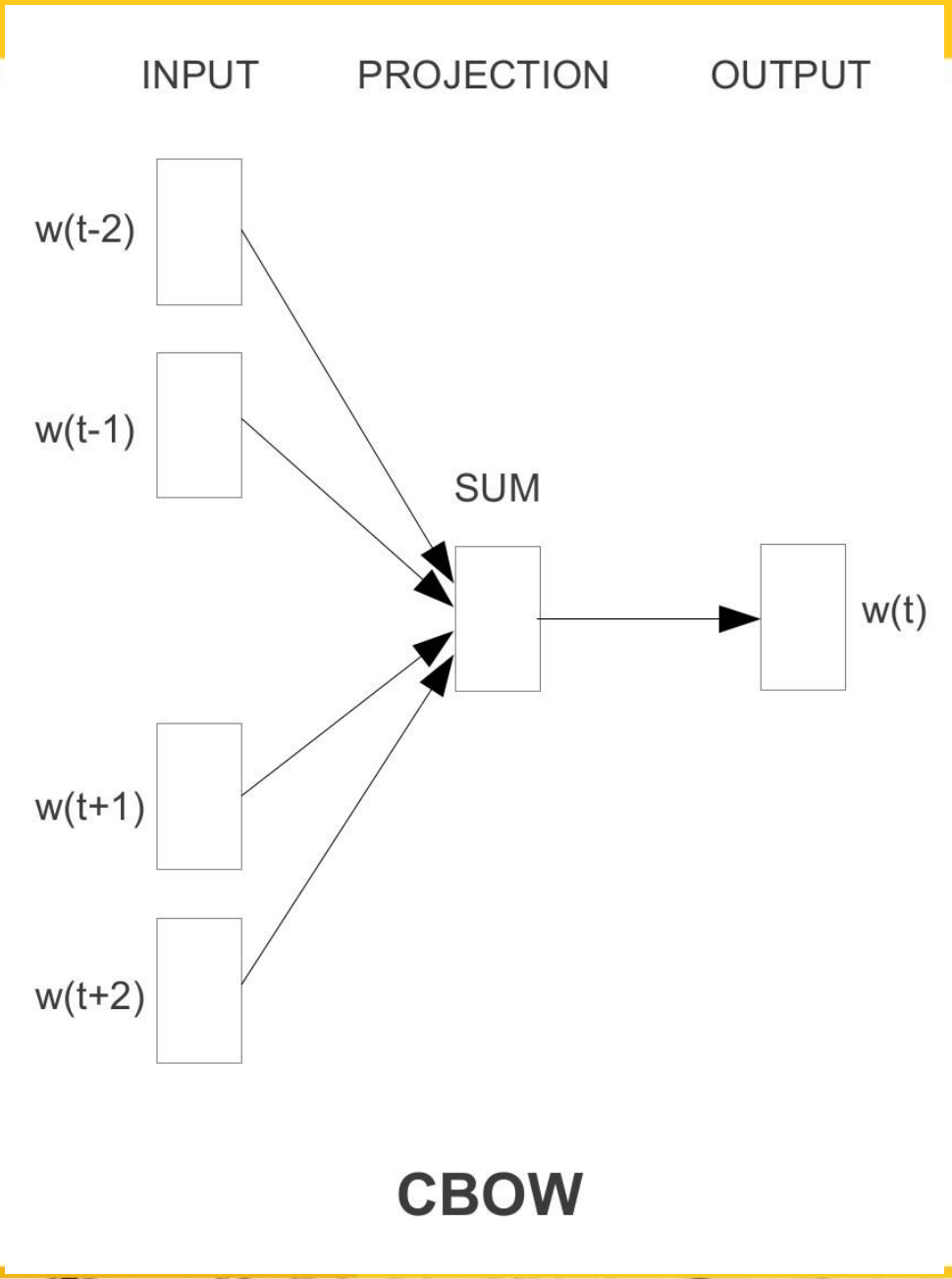
*linguistics* =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$



# continuous bag-of-words

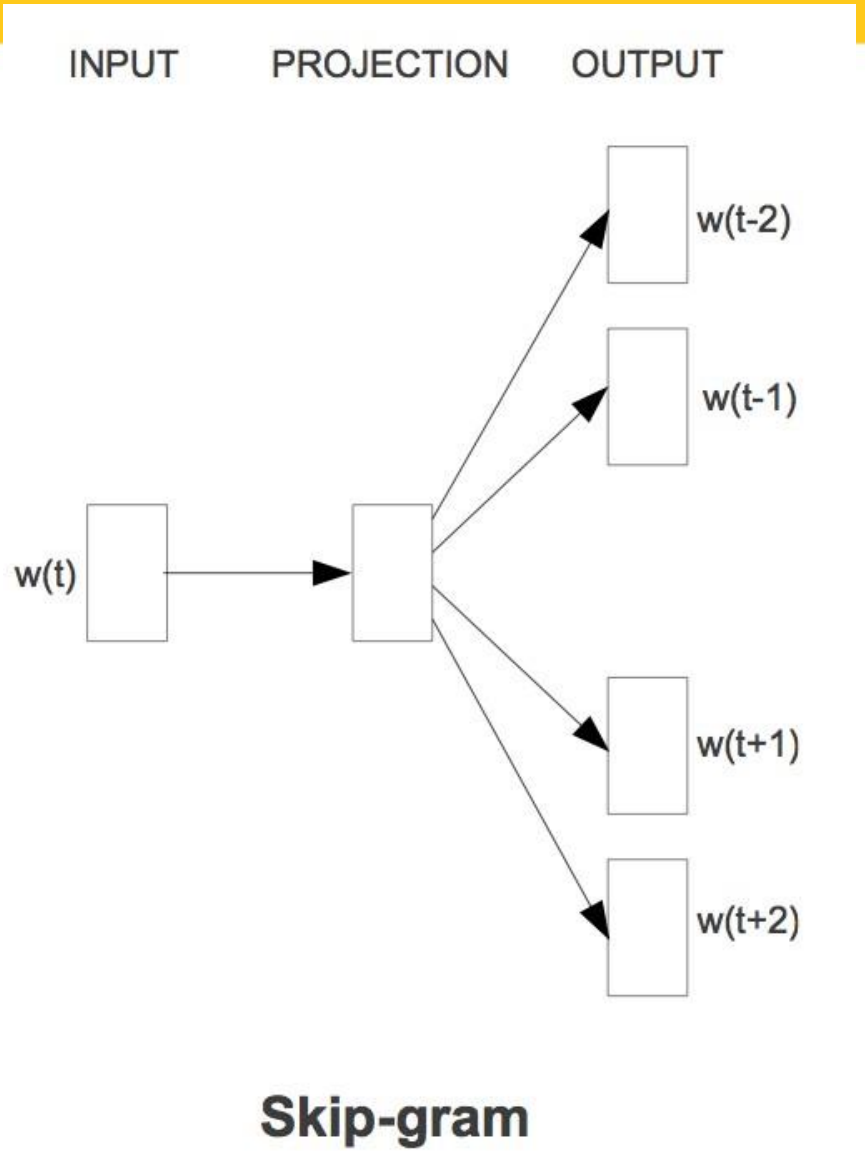
- predicting the current word **based on the context**
- **order** of words in the history **does not influence the projection**
- faster & **more appropriate for larger corpora**




Mikolov et al.

# continuous skip-gram

- maximize classification of a word **based on another word in the same sentence**
- better word vectors for **frequent words**, but slower to train




Mikolov et al.

# Why it is awesome

- there is a **fast open-source implementation**
- can be used as **features**
  for **natural language processing tasks** and
  **machine learning** algorithms

# Machine Translation

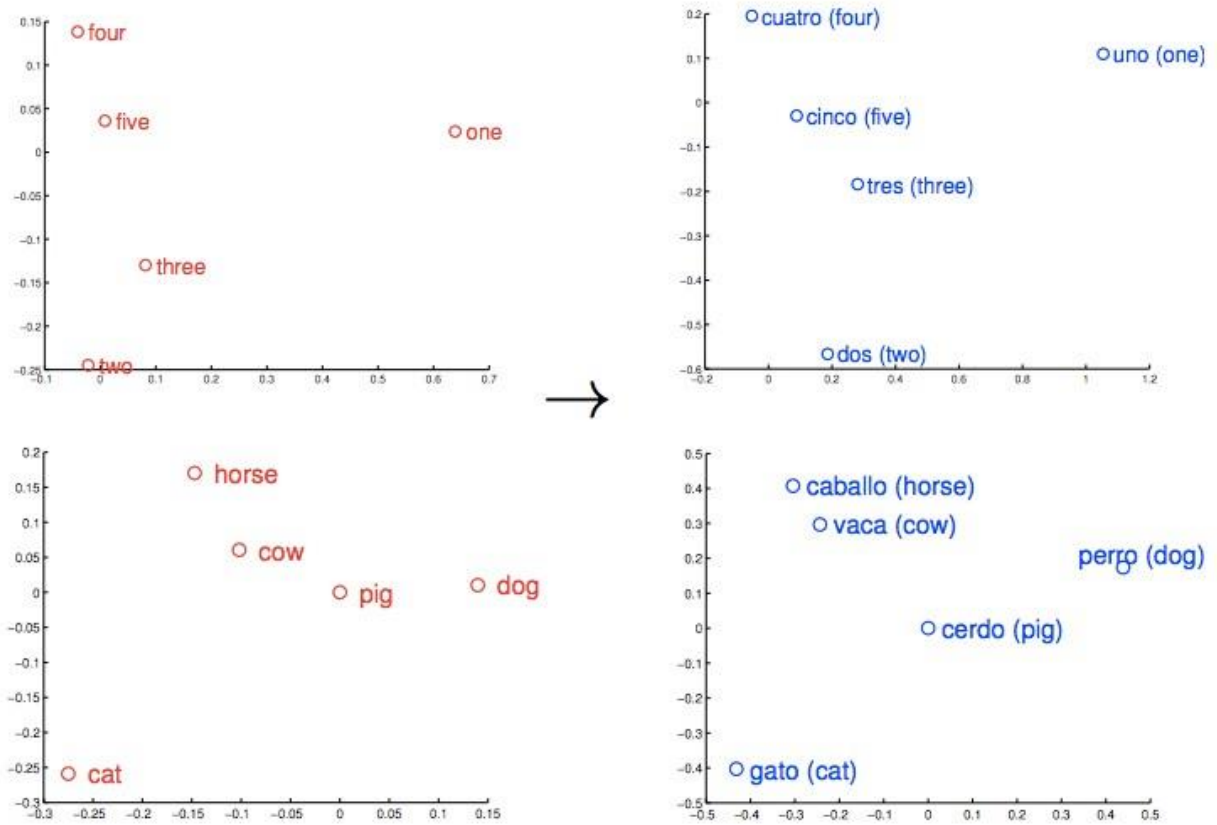
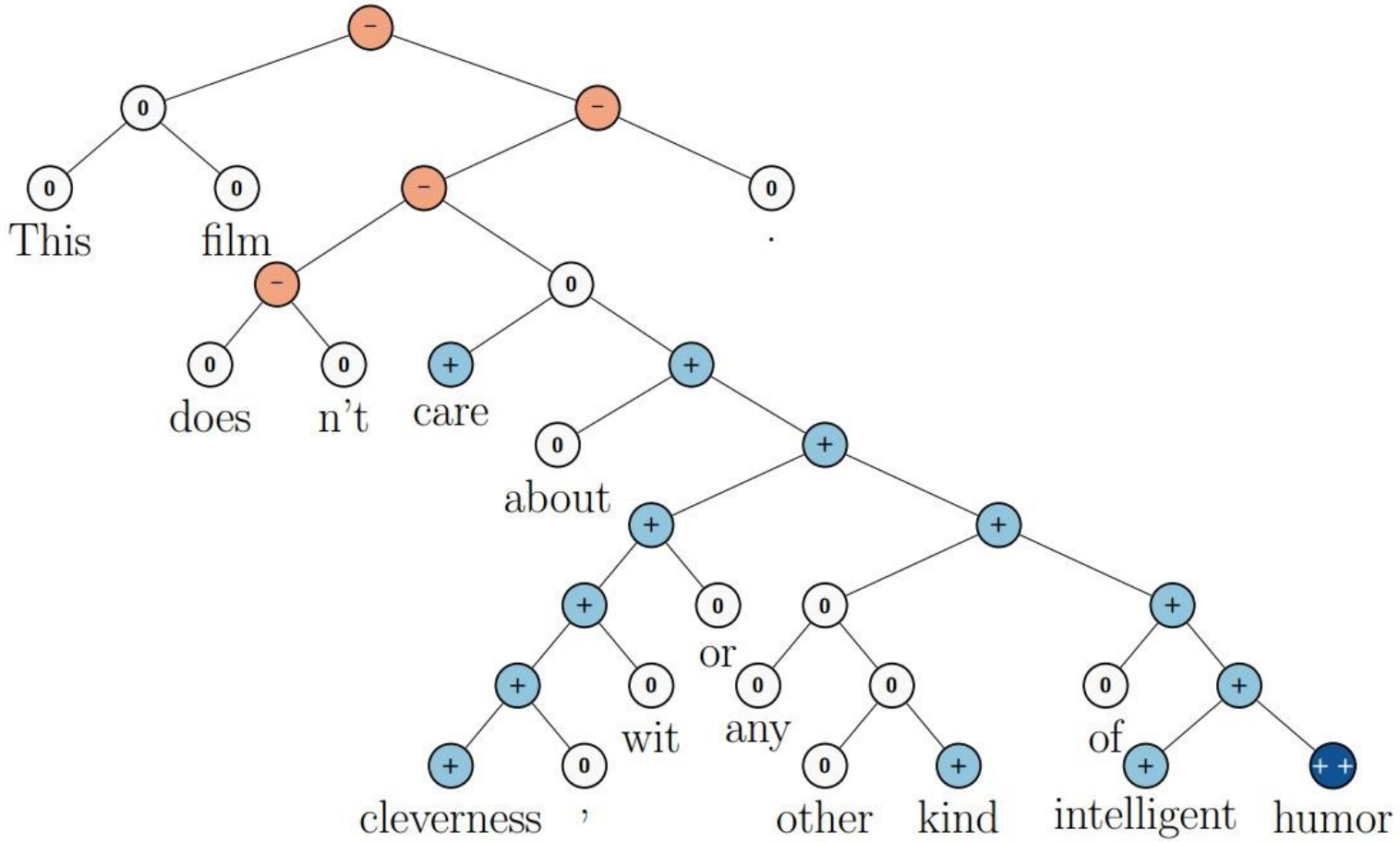



Figure 1: Distributed word vector representations of numbers and animals in English (left) and Spanish (right). The five vectors in each language were projected down to two dimensions using PCA, and then manually rotated to accentuate their similarity. It can be seen that these concepts have similar geometric arrangements in both spaces, suggesting that it is possible to learn an accurate linear mapping from one space to another. This is the key idea behind our method of translation.

Quoted after Mikolov

# Sentiment Analysis




Recursive Neural Tensor Network

Quoted after Socher

# Image Descriptions


Figure 5. A selection of evaluation results, grouped by human rating.

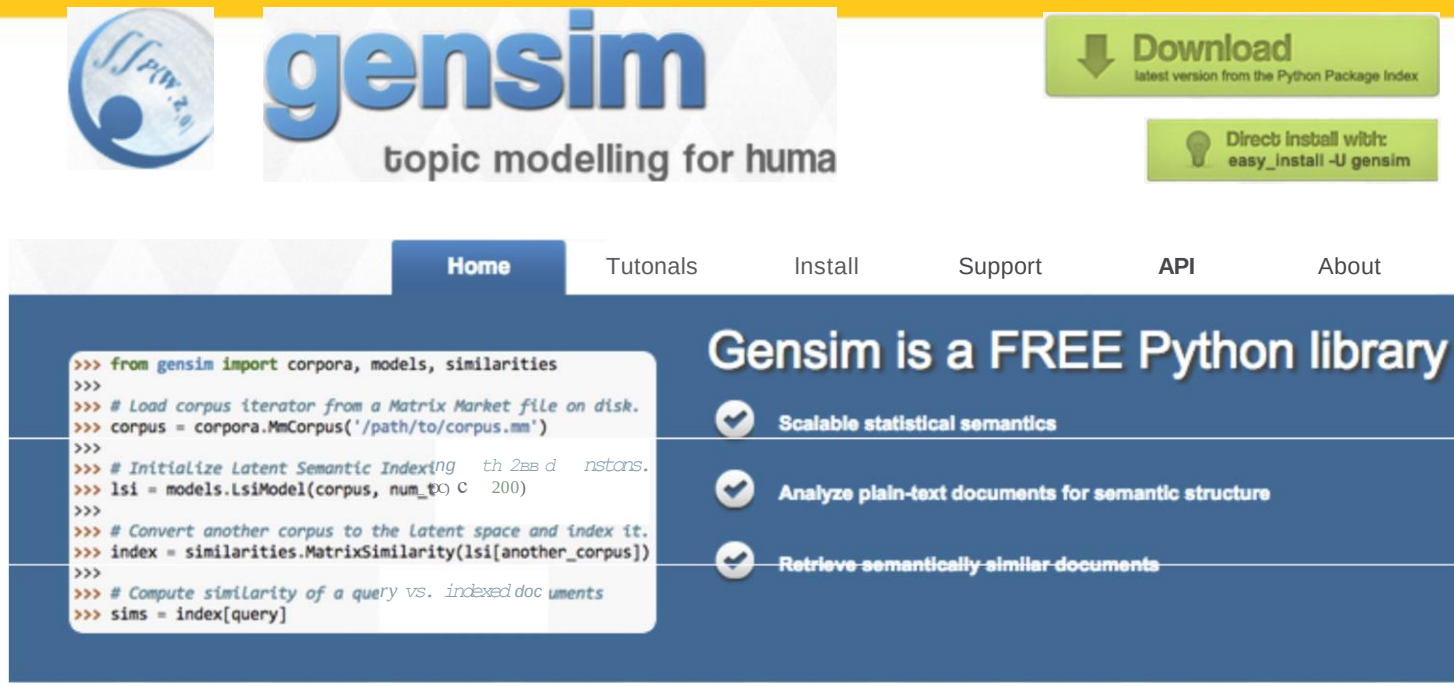
Quoted after Vinyals

# Using word2vec

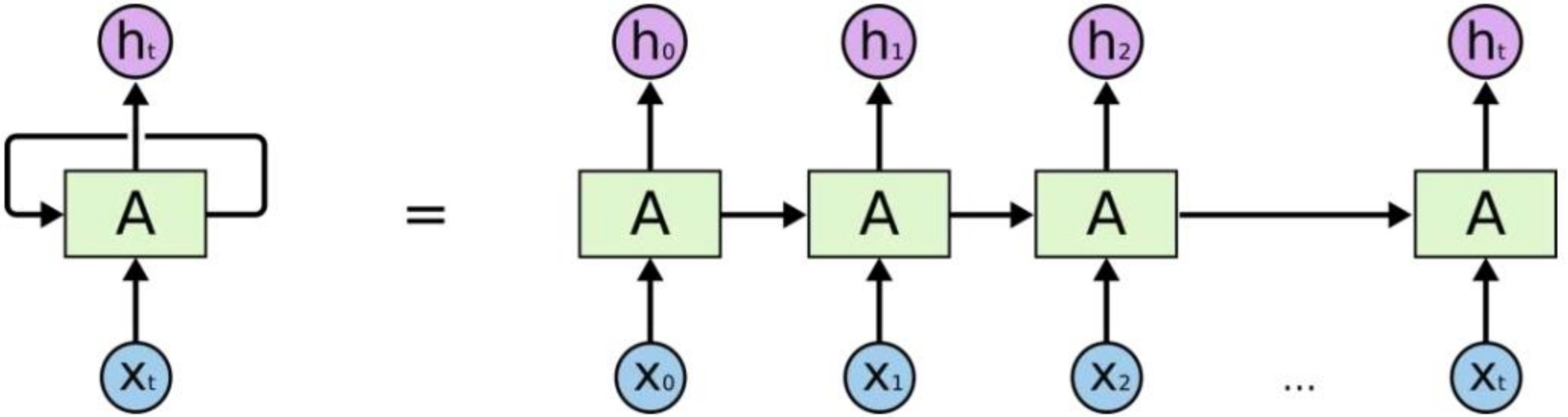
- Original: http://word2vec.googlecode.com/svn/trunk/
- C++11 version: https://github.com/jdeng/word2vec
- Python: http://radimrehurek.com/gensim/models/word2vec.html
- Java: https://github.com/ansjsun/word2vec_java
- Parallel java: https://github.com/siegfang/word2vec
- CUDAversion: https://github.com/whatupbiatch/cuda-word2vec

Quoted after Wang

# Using it in Python

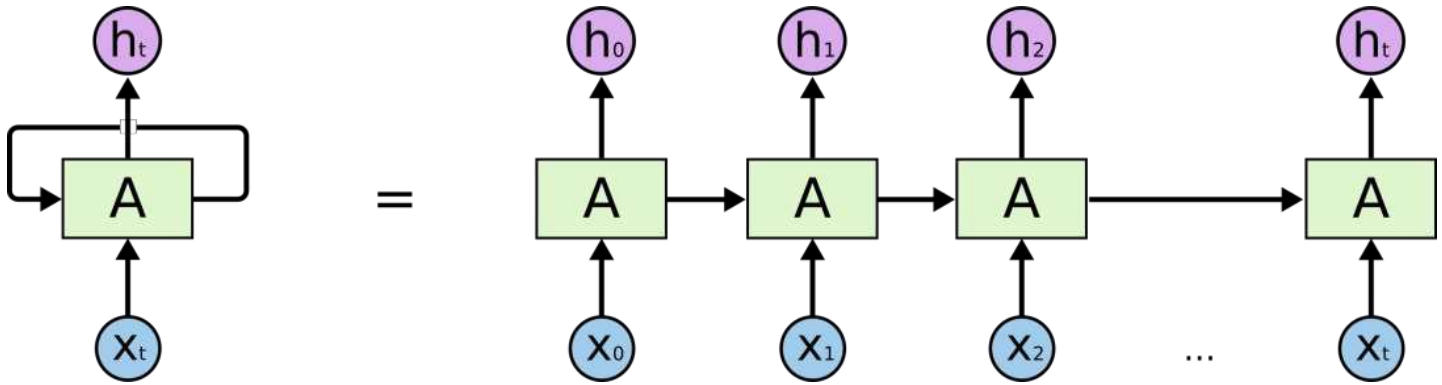

gensim
topic modelling for huma
Download
latest version from the Python Package Index
Direct install with:
easy_install -U gensim
Home
Tutonals
Install
Support
API
About
>>> from gensim import corpora, models, similarities
>>>
>>> # Load corpus iterator from a Matrix Market file on disk.
>>> corpus = corpora.MmCorpus('/path/to/corpus.mm')
>>>
>>> # Initialize Latent Semantic Indexing with 200 dimensions.
>>> lsi = models.LsiModel(corpus, num_topics=200)
>>>
>>> # Convert another corpus to the latent space and index it.
>>> index = similarities.MatrixSimilarity(lsi[another_corpus])
>>>
>>> # Compute similarity of a query vs. indexed documents
>>> sims = index[query]
Gensim is a FREE Python library
Scalable statistical semantics
Analyze plain-text documents for semantic structure
Retrieve semantically similar documents
eatures
Hover your mouse over each feature for more info.
Scalablity
Platform independent
Robust
Open source
Efficient implementations
Converters & I/O formats
Similarity queries
Support

# Recurrent Neural Network (RNN)


$h_t$
A
$x_t$
=
$h_0$
$h_1$
$h_2$
$h_t$
A
A
A
A
$x_0$
$x_1$
$x_2$
...
$x_t$

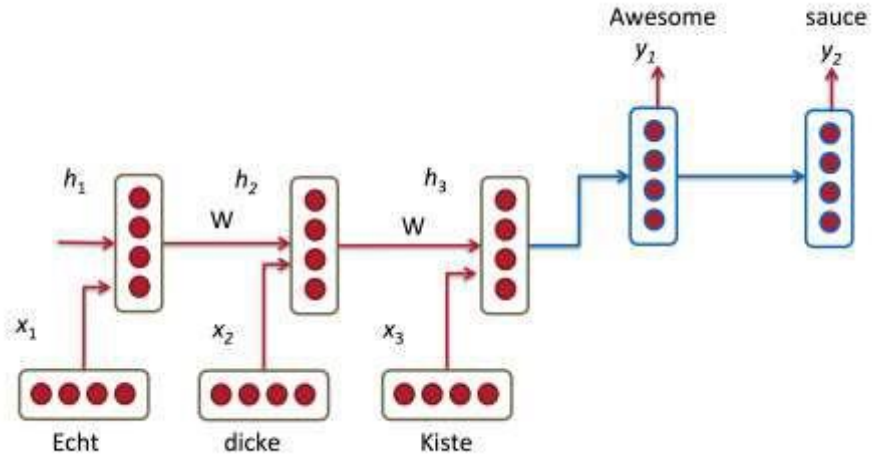
# Benefits

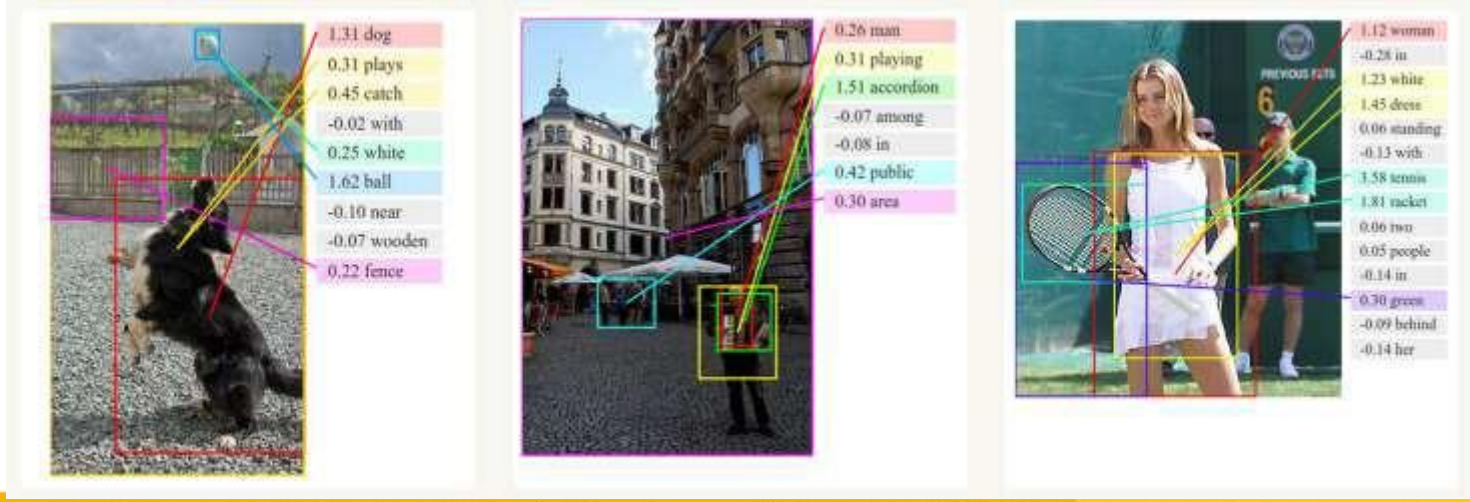
- Deal with sequential information
- Perform the same task for every element of a sequence
- Has memory
- Can be unrolled like a chain



# Application

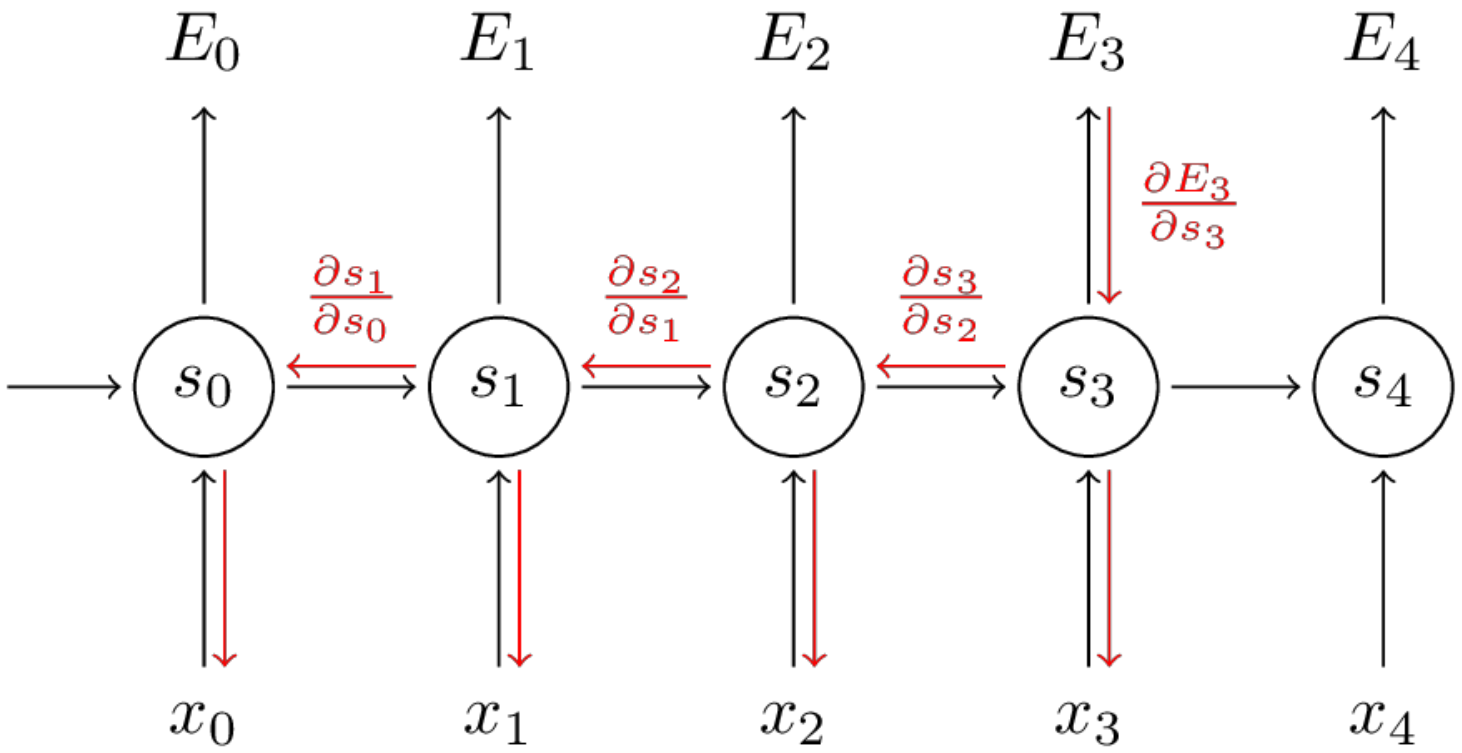
- Language Modeling and Generating Text
- Machine Translation
- Speech Recognition
- Generating Image Descriptions



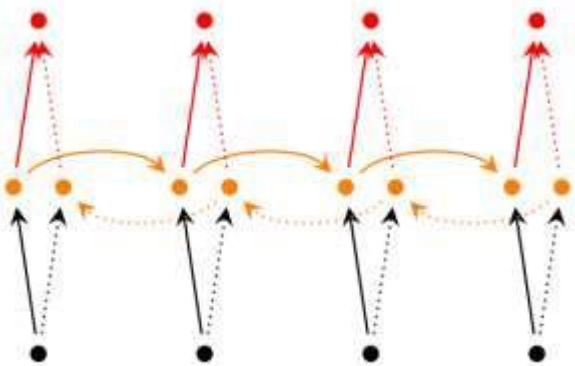
# Training

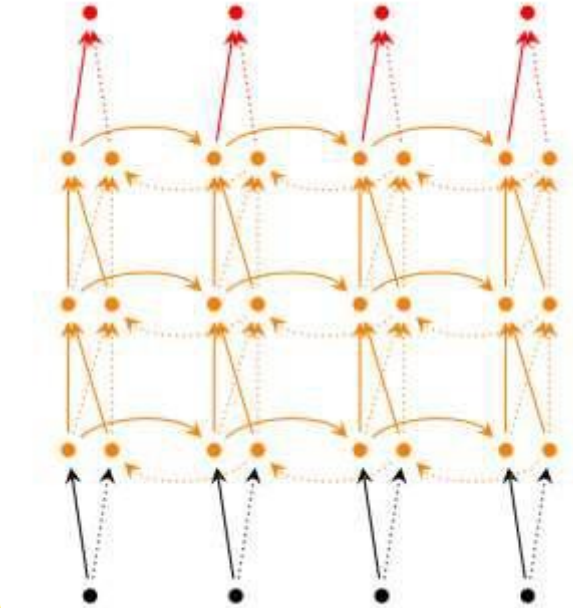
- Backpropagation Through Time (BPTT)

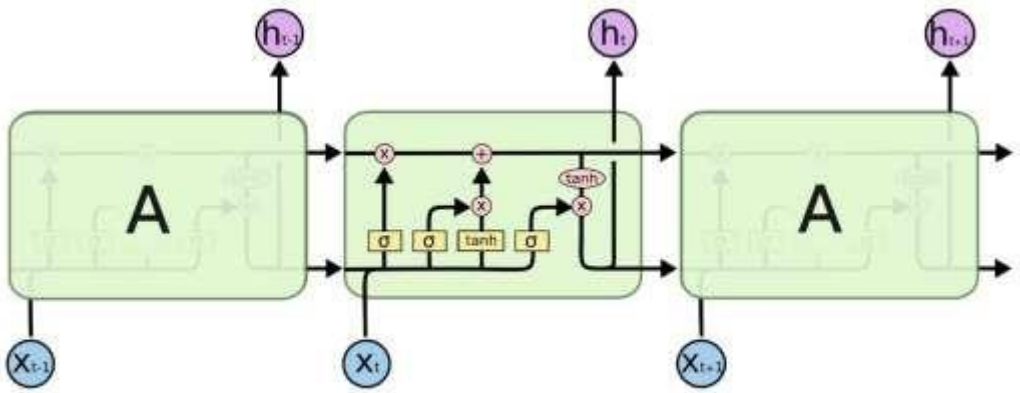


# Variations

- Bidirectional RNNs
- Deep (Bidirectional) RNNs
- **LSTM networks**

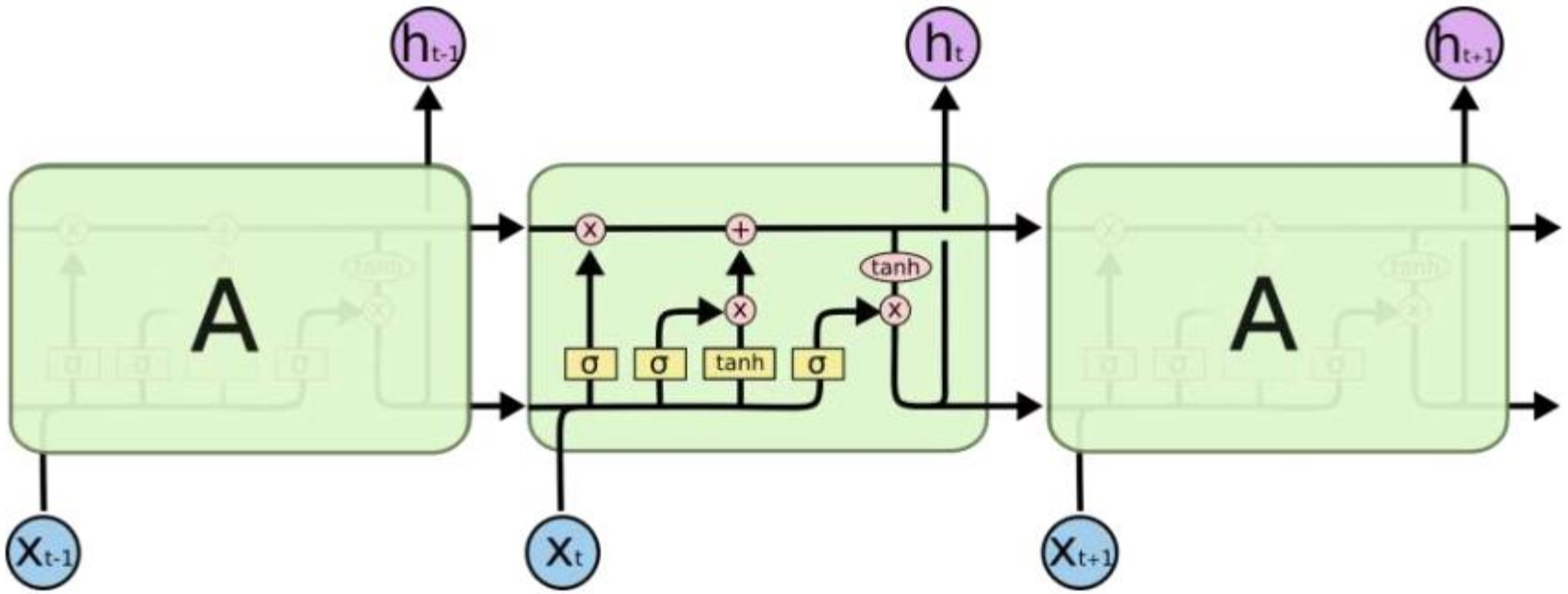






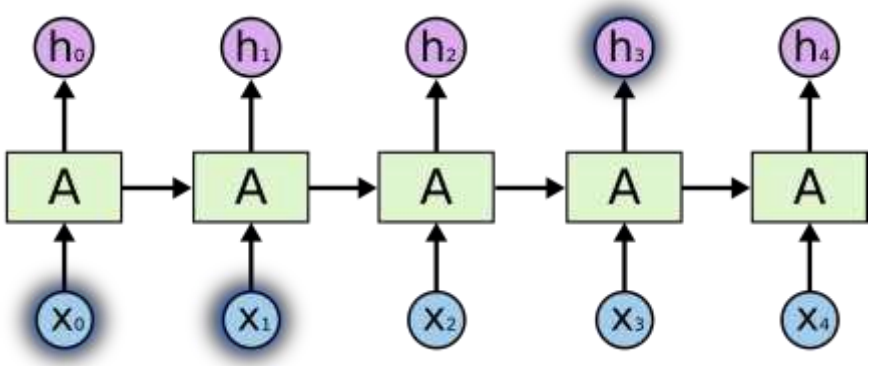

The repeating module in an LSTM contains four interacting layers.

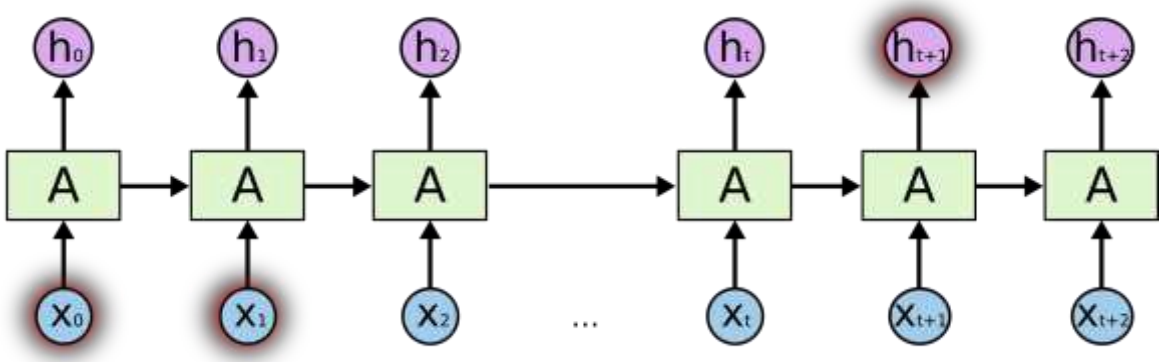
# Long Short Term Memory Network (LSTM)


$h_{t-1}$
$h_t$
$h_{t+1}$
A
A
×
+
tanh
×
×
σ
σ
tanh
σ
$X_{t-1}$
$X_t$
$X_{t+1}$

# Memory Problem of RNN

- Sometimes we need more context
- RNN is unable to connect the information further in the past



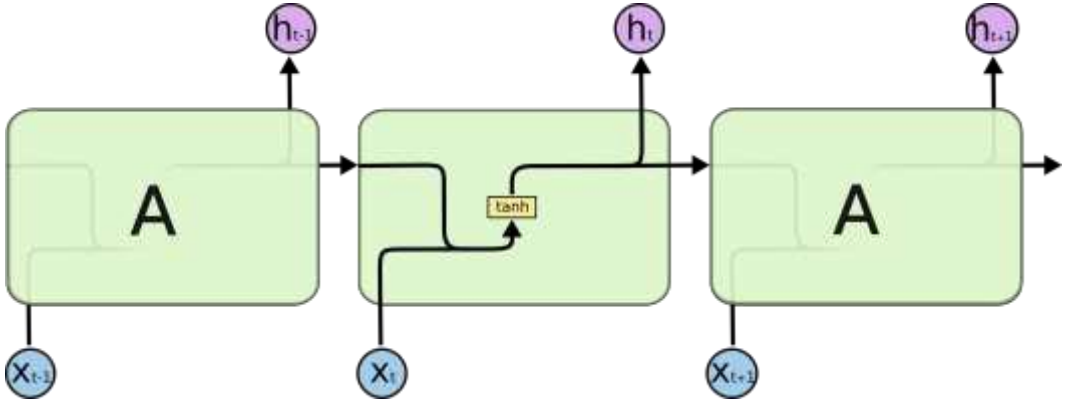



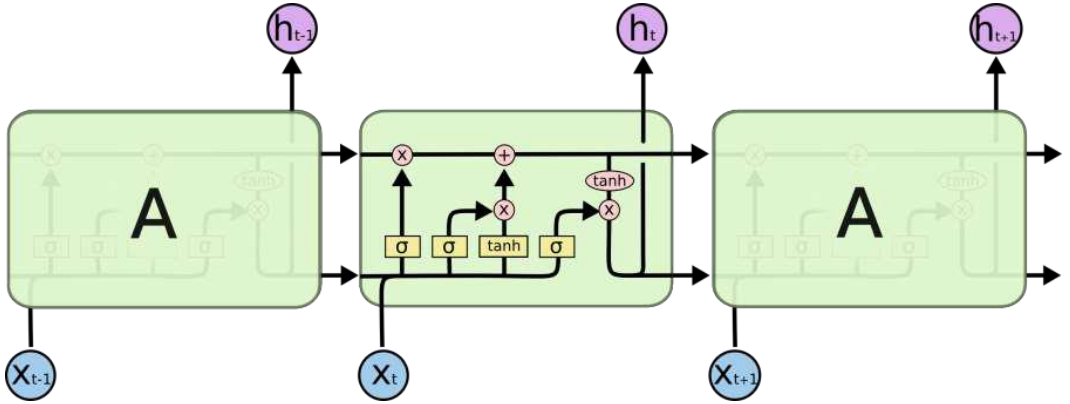
# Benefits of LSTM

- Can learn long-term dependencies

# Difference between RNN & LSTM

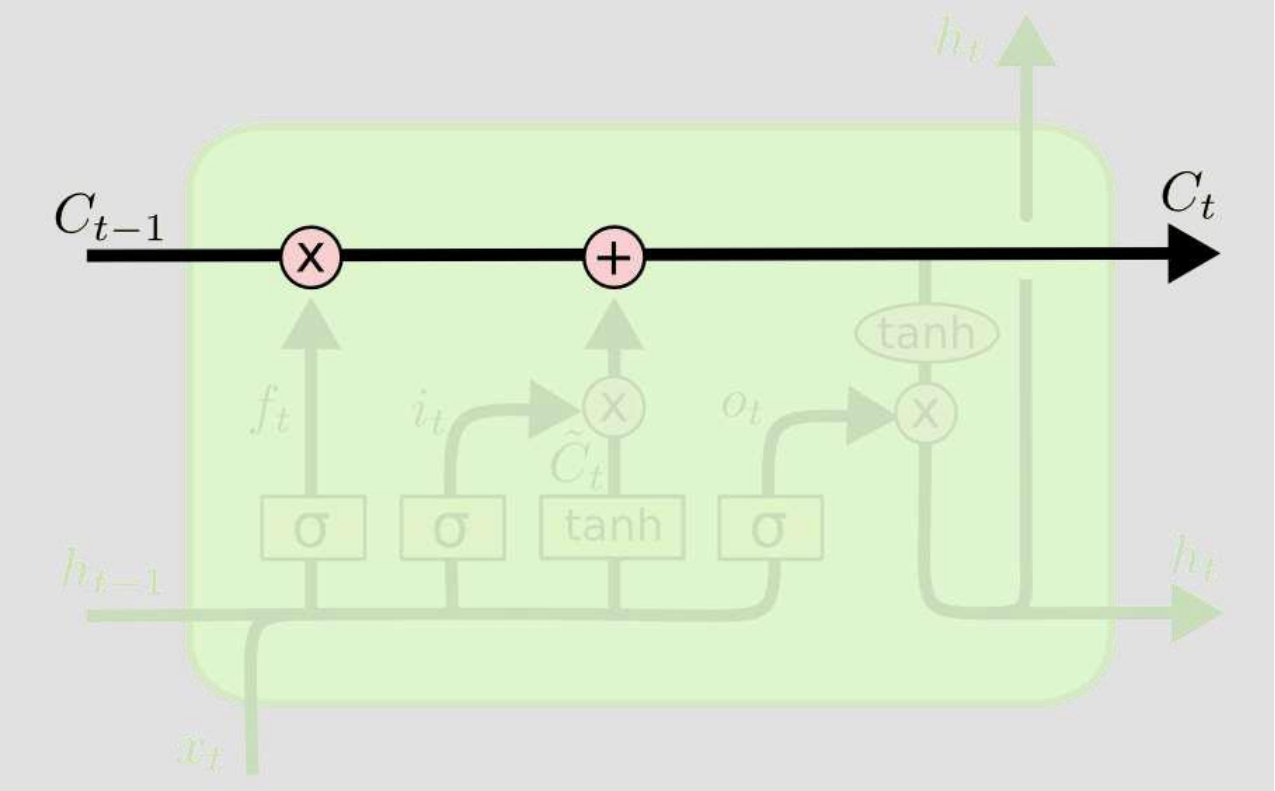
- **RNN**: single layer (tanh)
- **LSTM**: four interactive layers



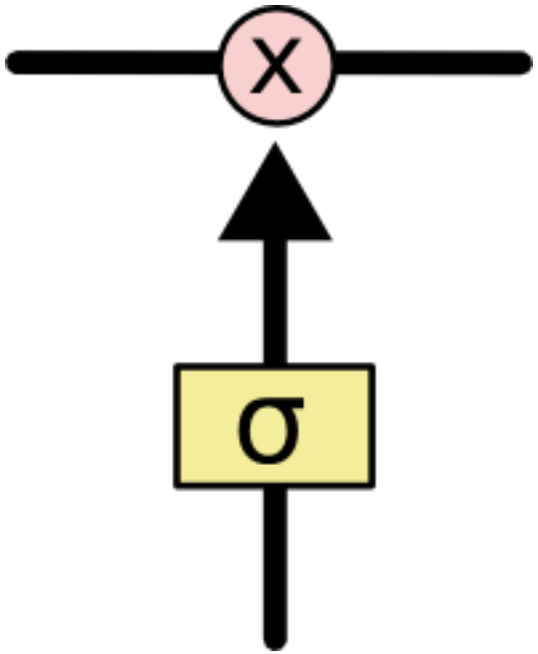
# Cell state

- The conveyor belt



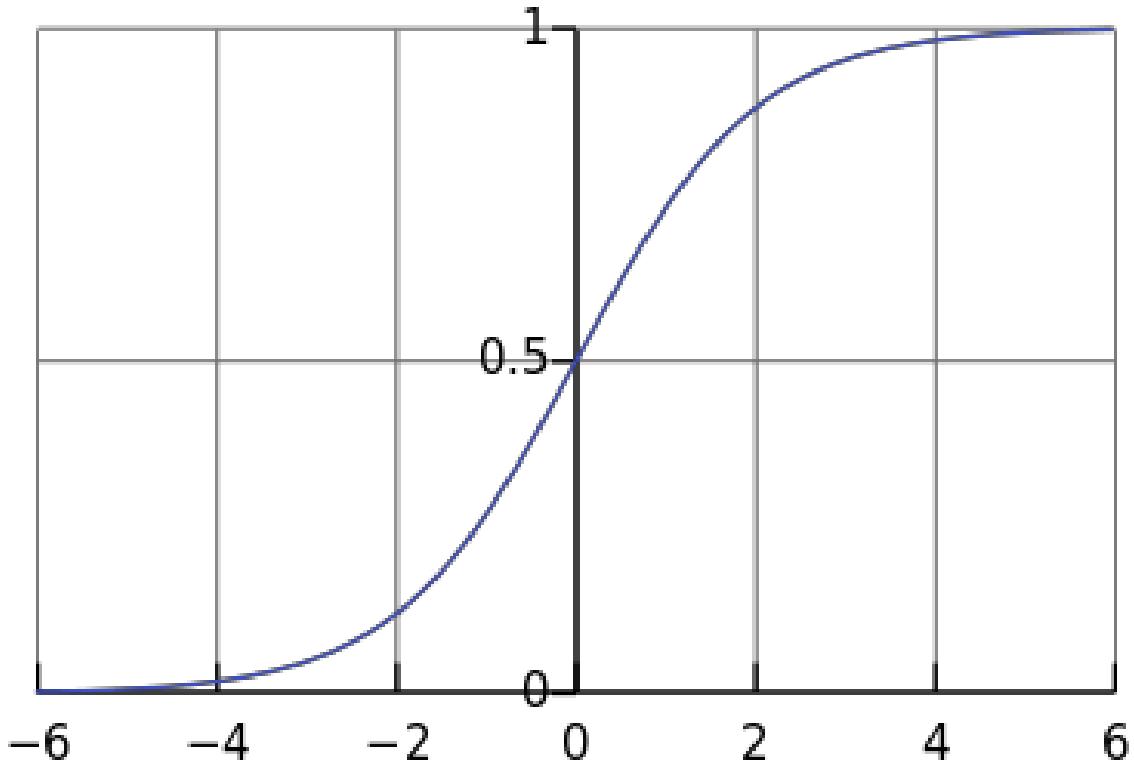
# Gates (3 in total for LSTM)

- A way that let information through
- E.g. A sigmoid neural net layer & a pointwise multiplication operation



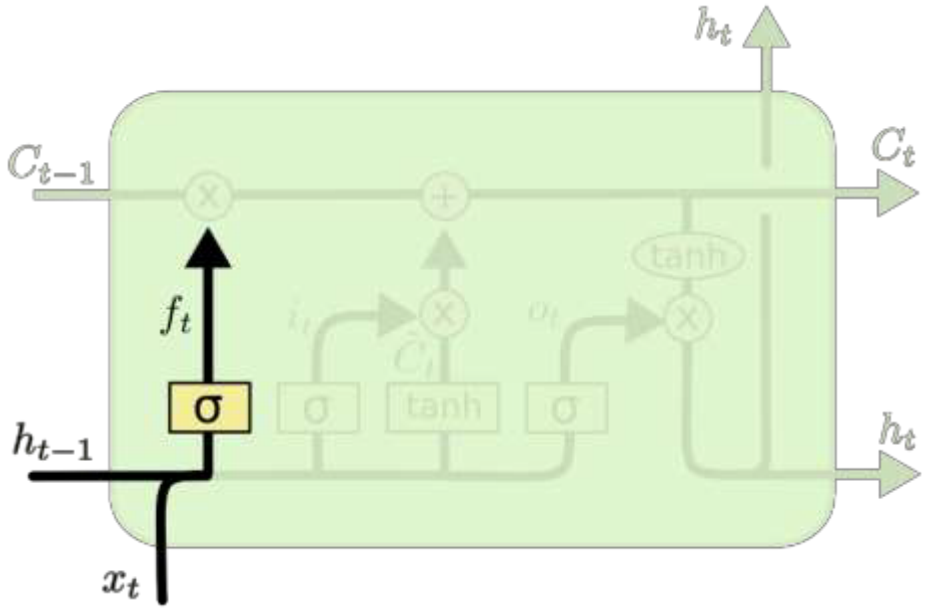
# Optional Math – Sigmoid function

$$S(t) = \frac{1}{1 + e^{-t}}.$$



# Step 1: **Forget Gate Layer**

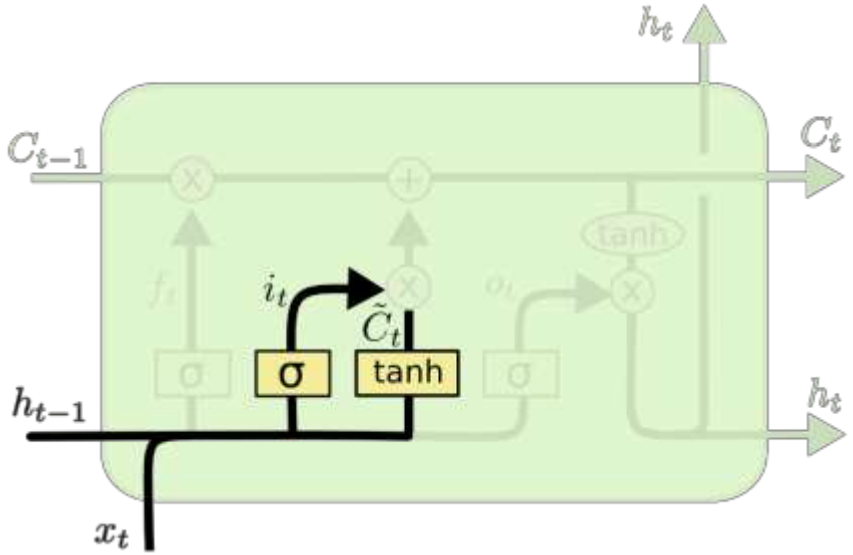
- **Decide what info to throw away**
- Look at h[t-1] and x[t] and output a number 0~1 to decide how much cell state to keep C[t-1]
- E.g. When see a new subject, we want to forget the gender of the old subject




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

# Step 2: **Input Gate Layer**

- **Decide what info to add**
- A sigmoid: decide which value to update
- A tanh layer: create a new candidate value C~[t]
- E.g. add a new gender of the new subject

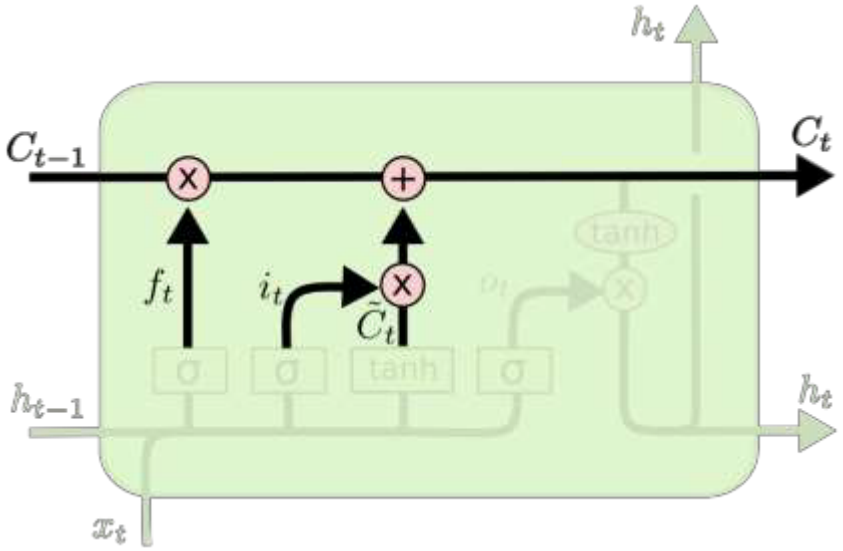



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

# Step 3: **Combine step 1 &2**

- **Combine step 1 & 2**
- Multiply the old state by f[t]: to forget the things
- Add i[t] * C~[t] : to add new candidate value (scaled)



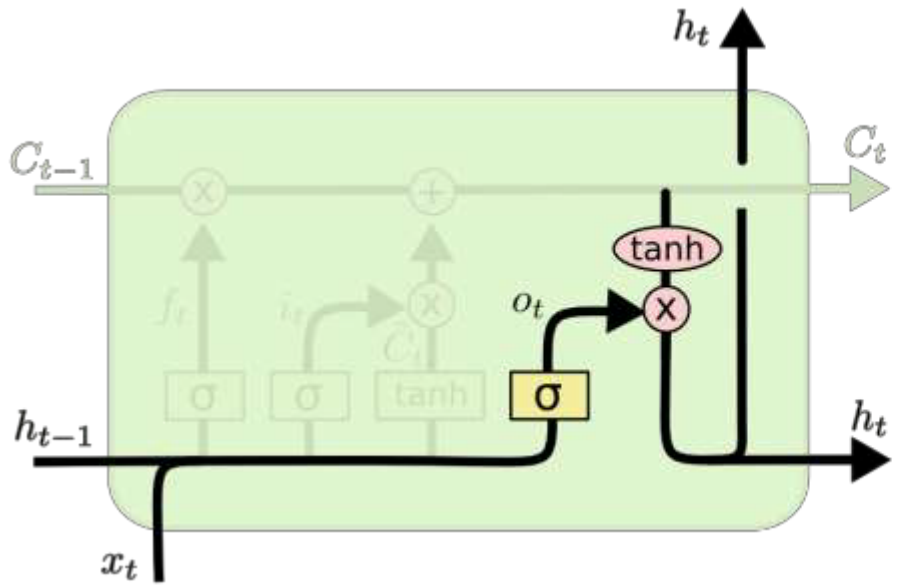

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# Step 4: **Filter/output the Cell state**

- **Decide what to output**
- sigmoid: decide which part to output
- tanh: push the value to be between -1 ~ 1
- Multiply them to only output the part we decided to
- E.g. output a info related to a Verb
- E.g. output whether the subject it singular or plural

# Step 4: **Filter/output the Cell state**

- **Decide what to output**

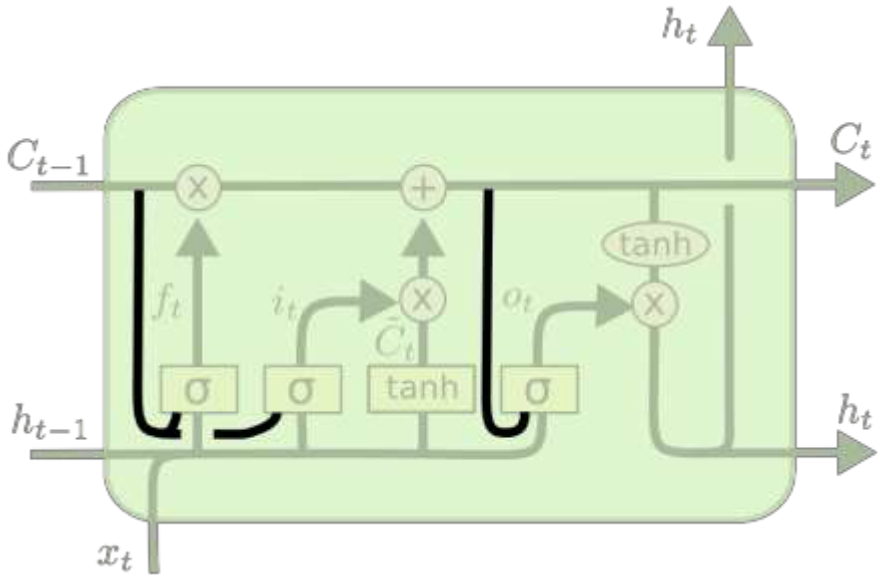



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

# Variants on LSTM (1)

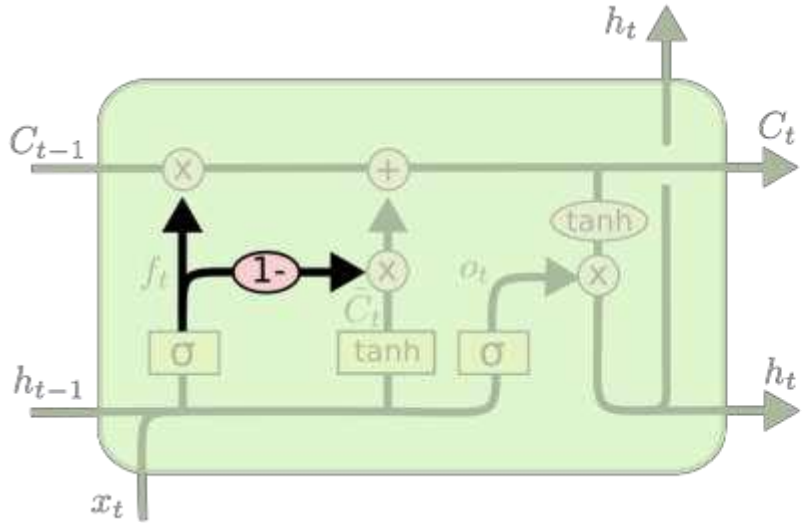
- **Peephole:**
  - let the gate layer look at the cell state (entire/ partial)




$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

# Variants on LSTM (2)

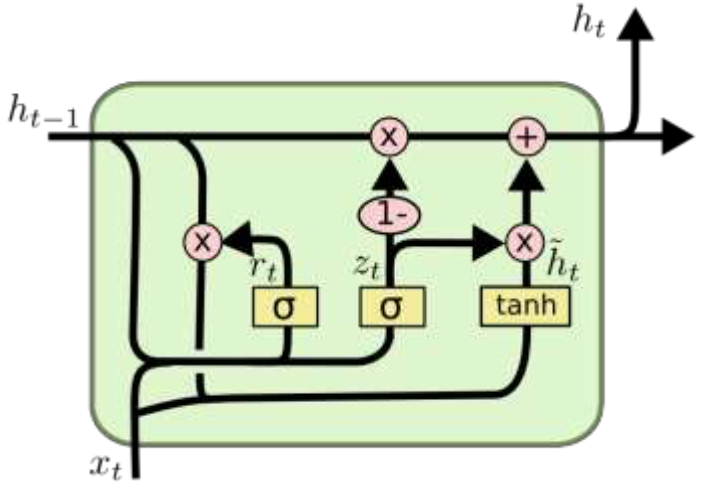
- **Coupled forgot and input gates:**
  - Not deciding separately
  - f[t] * C[t-1] + (1-f[t]) * C~[t]




$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

# Variants on LSTM (3)

- **Gated Recurrent Unit (GRU):**
  - combine the forget and input layer into a single "update gate"
  - merge the cell state and the hidden state
  - simpler and popular




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

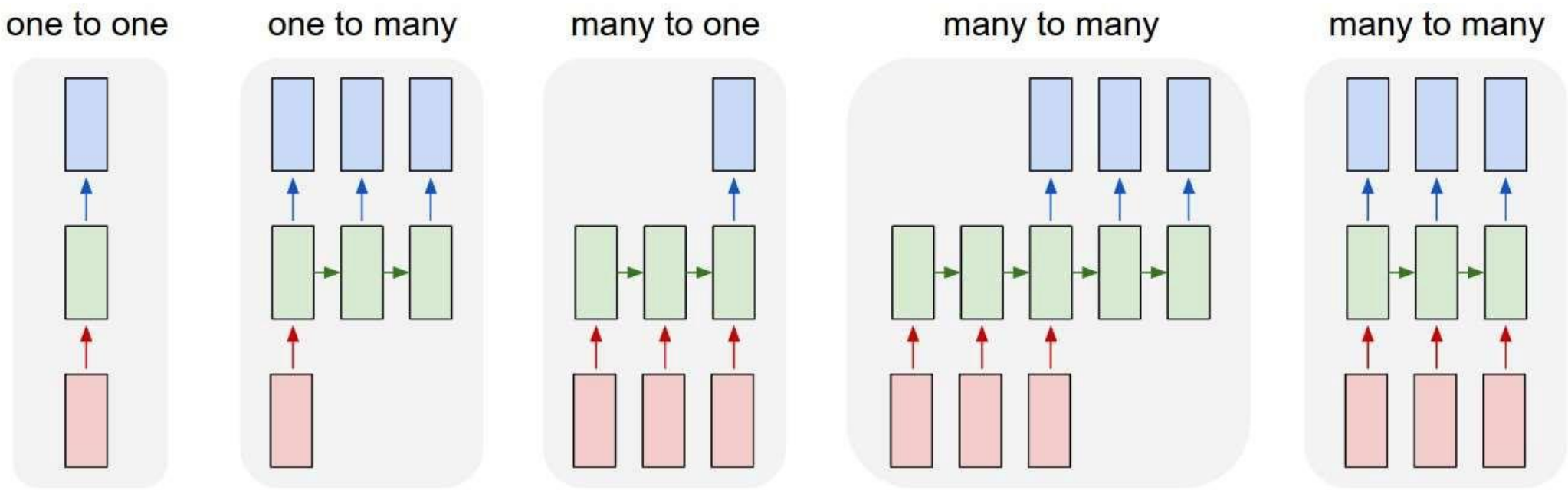
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# RNN / LSTM Effectiveness

# Multiple types of RNN usecases



# Turing-Complete

- Running a fixed program with certain inputs and some internal variables (can simulate arbitrary programs)

- Andrej Karpathy (Ph.D. @ Stanford):

> *If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs.*

# Non-sequential data

- Though the data is not in form of sequences, we can still use RNN by process it sequentially.

# Some cool RNN/LSTM applications

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Great references


- [1] RNN: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/?subscribe=success#blog_subscription-2
- [2] LSTM: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- [3] RNN Effectiveness: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- [4] Backpropagation: http://cs231n.github.io/optimization-2/#backprop
- [5] ML categories: http://enhancedatascience.com/2017/07/19/machine-learning-explained-supervised-learning-unsupervised-learning-and-reinforcement-learning/