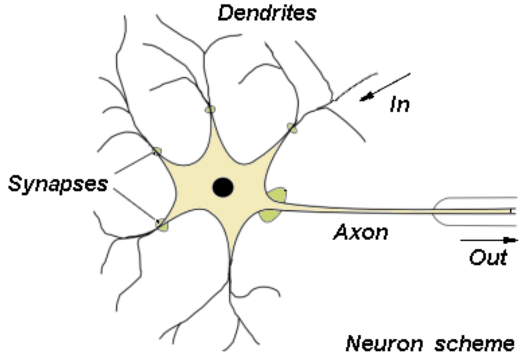
Neural Networks and Word Embeddings

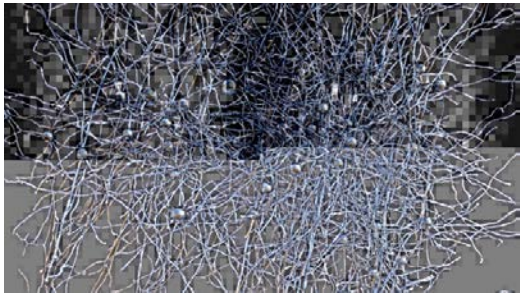
Ulf Leser, Humboldt-Universität zu Berlin

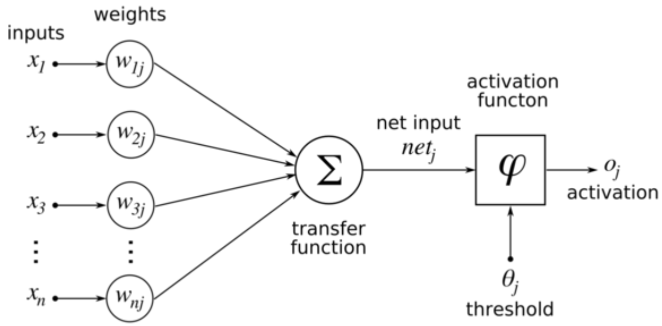
Table of Contents

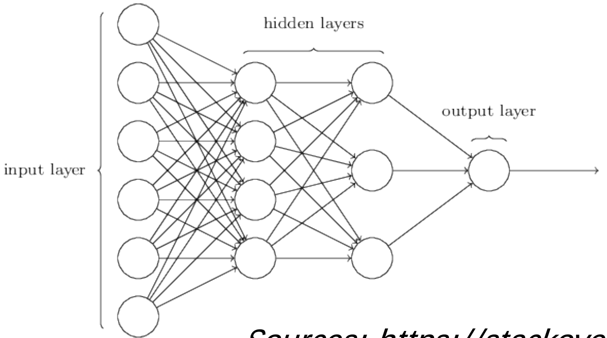
Artificial Neural Networks (ANN)

- A method for non-linear classification
- Quite old, always present, extremely hyped since ~2015
- Inspired by biological neural networks



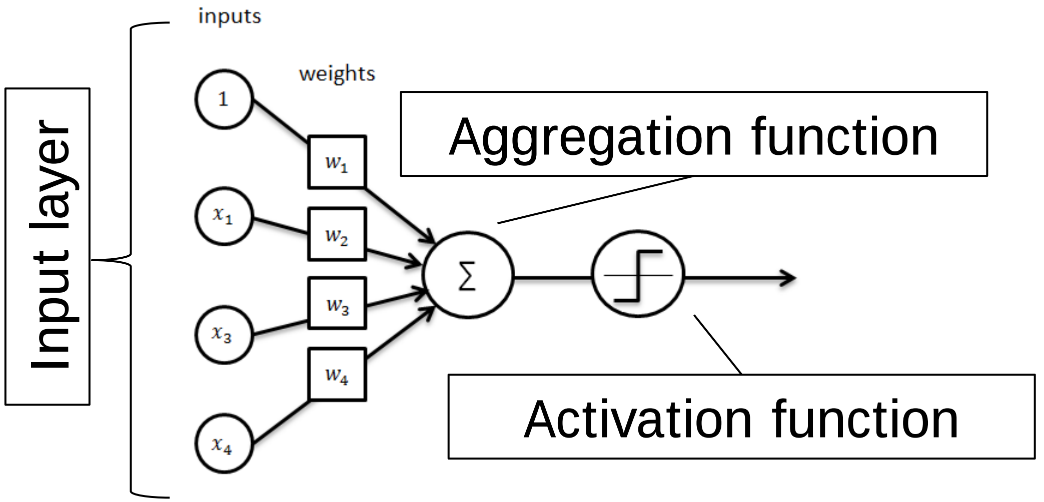


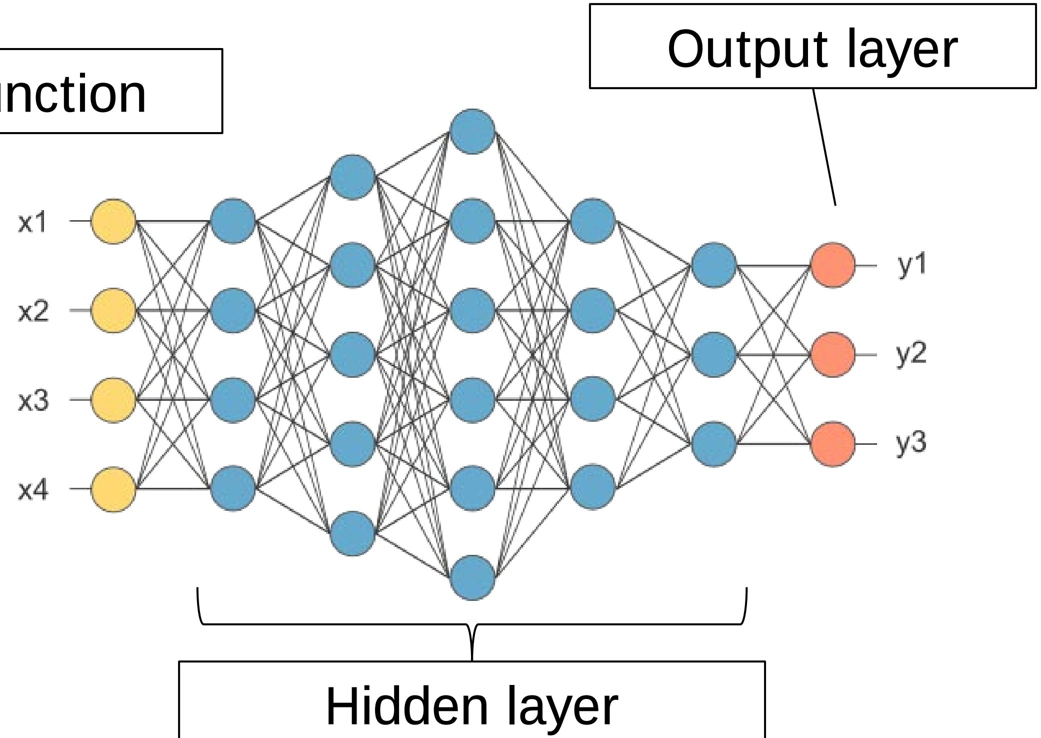




Sources: https://stackoverflow.com
http://neuralnetworksanddeeplearning.com
https://alleninstitute.uk

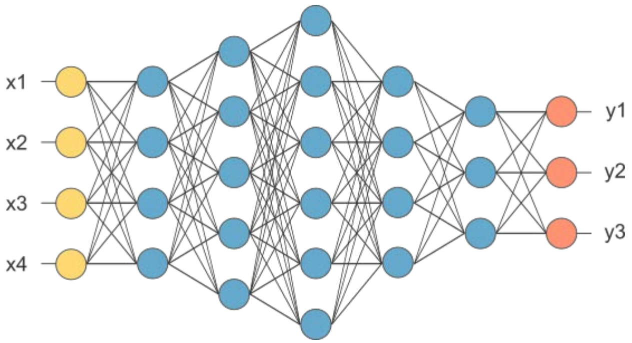
Concepts





Source: http://www.opennn.net/

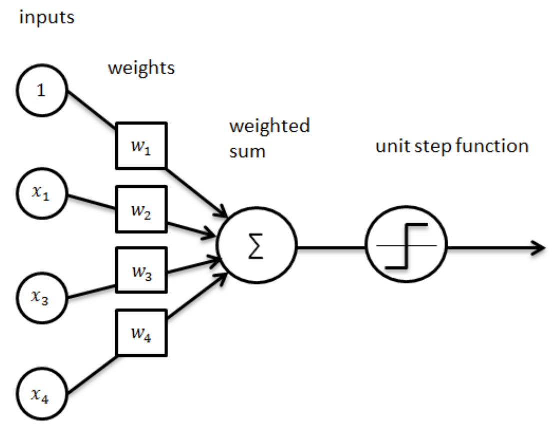
Usage

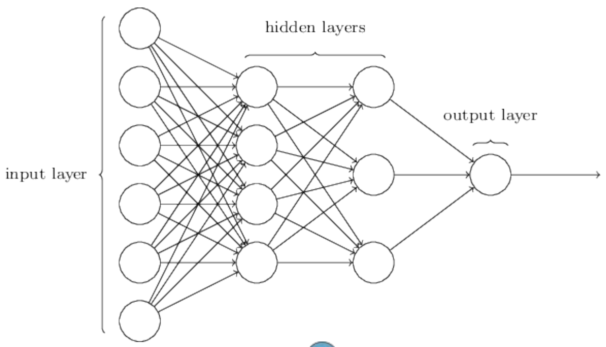


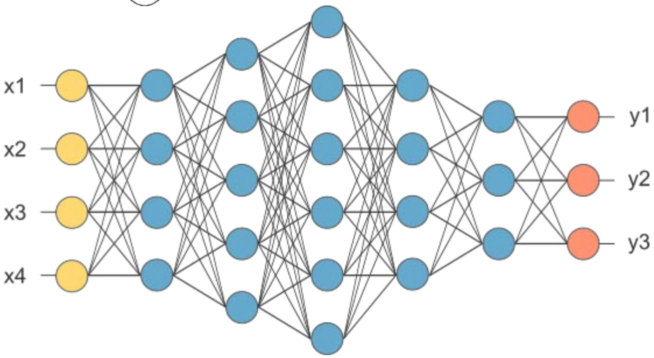
- Objects are described as sets of features
- Binary classification: One output unit and a threshold
  - Multi-class: One output unit per class producing the probability of belonging to this class
- Training: Find weights for all connections between units such that the error of the output on the training data is minimized
  - Performed backwards through the network: Training
- Application: Compute output based on to-be-classified input using the learned weights
  - Performed forward through the network: Prediction

Many Design Choices

- Activation (aggregation) function?
- Number of hidden layers?
- Number of units per hidden layer?
- Connections only between adjacent layers?
- Only “forward” connections?
- Loss function for learning
- Central issue: “Learnability”
  - Different choices lead to different problems
  - Especially back-links increase complexity (and expressiveness)

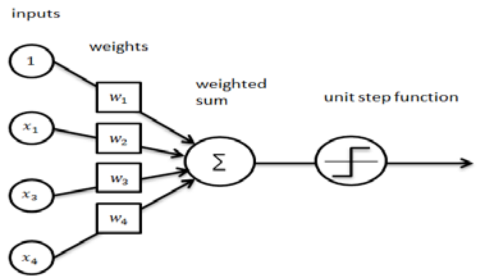


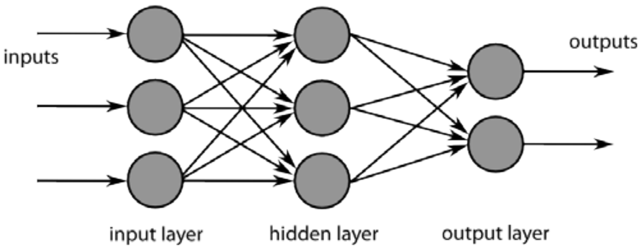


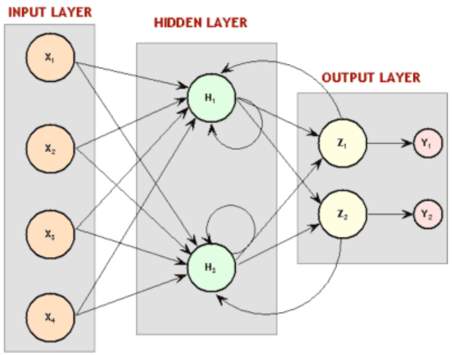


Classical Examples

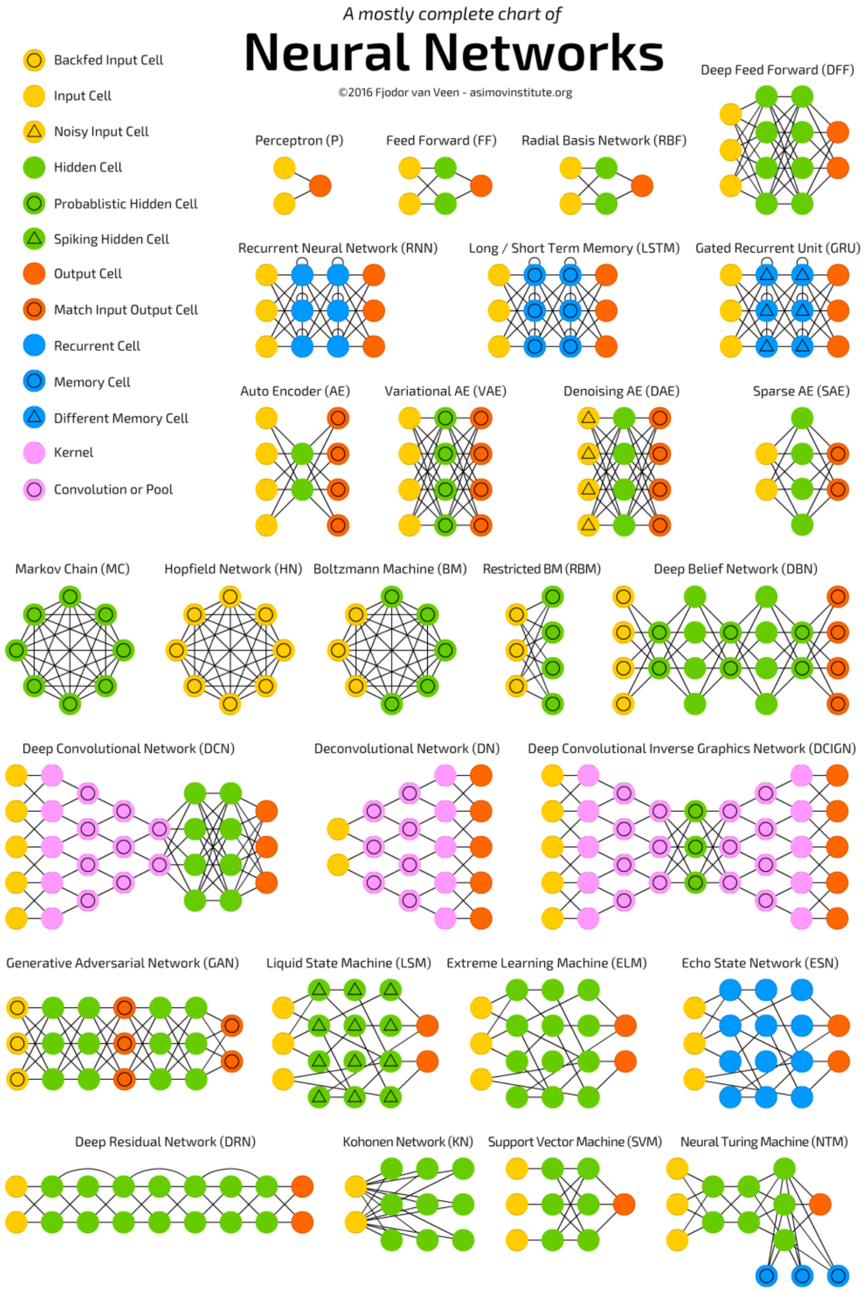
- Perceptron
  - Dead for some time: XOR problem
- Feed-forward ANN
  - Directional, level-wise information flow
  - Can learn almost arbitrary functions (depending on AF)
- Recurrent ANN (RNN)
  - Information may flow back
  - Can learn state for sequential inputs (like in NER)
- Convolutional neural networks
- AutoEncoder
- …





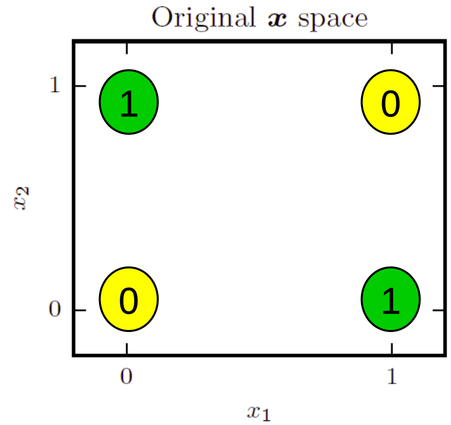


... and many more variations

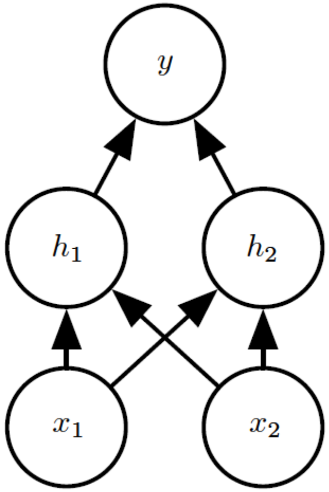


Source: http://www.asimovinstitute.org/

Non-Linear Activation Functions



- How can we learn this decision (XOR)?
- No linear combination of x_1, x_2 will work
  - There is no straight line partitioning the space in the correct "green" and "yellow" parts
- Trick: Use a two-level ANN and a non-linear AF



"Rectified linear activation": out = max(0, W^T*x+c)+b

h_1=max(0, x_1+x_2+0)

h_2=max(0, x_1+x_2-1)

y=max(0,h_1-2*h_2)

x_1	x_2	h_1	h_2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

Learning a ANN

- Feedforward (and many other) ANN can be efficiently learned using back-propagation
- Idea
  - Initialize weights at random
  - Compute loss function for training samples
  - Adjust weights level wise along the gradient of the loss function
  - Repeat until convergence
  - Trick: Fast and repeated computation of the gradients
- Variation of stochastic gradient descent (SGD)

Deep Learning

- ANN for long did not outperform other yet faster methods
- Two trends since roughly 2012
  - Build deeper networks – more and wider hidden layers capture more signals
    - It is not true that "more is always better"
    - Still much art (not science) in tuning hyper-parameters
  - Learn on much more data
    - Deep learning is only good if a lot training data is available
    - Include unsupervised data – pre-training to obtain good initial weights
  - Both require much longer training times – prohibitive in the past
  - Today: Optimized algorithms, stronger machines, accelerators (GPU), distributed learning, pre-trained models, ...
- Now very successful in machine translation, image recognition, gaming, machine reading, ...

Table of Contents

- A very brief introduction to Neural Networks
- Word Semantic
- Word Embeddings with Word2Vec
- Applications

Word Semantics

- All approaches we considered so far consider two tokens as different whenever they have different spelling
  - No shades: Equal or not, dimensions in VSM are orthogonal
  - King, princess, earl, milk, butter, cow, white, crown, emperor, ...
- This makes models very specific – bad generalization
  - If we know that p(milk|cow) is high, this doesn't tell us that p(butter|cow) is probably also high (higher than p(crown|cow))
  - We have to see all words sufficiently often during training – seeing semantically similar words doesn't help
- Humans do compare words in a multi-facetted way
  - King is similar to princess to earl to queen, but not to cow
    - But both are mammals
  - King uses crowns much more often than cows
- How can we capture word semantics to derive meaningful similarity scores?

Knowledge-based: WordNet, Wikipedia, ...

- Let's dream: A comprehensive resource of all words and their relationships
  - Specialization, synonymy, partonomy, relatedness, is_required_for, develops_into, is_possible_with, ...
- Example: WordNet
  - Roughly 150K concepts, 200K senses, 117K synsets
  - Specialization, partonomy, antonomy
- Can be turned into a semantic similarity measure, e.g., length of shortest path between two concepts
- Problem: Incomplete, costly, outdated
  - Especially in specific domains like Biomedicine
- Much research to automatically expand WordNet, but no real breakthrough

Distributional Semantics

- „You shall know a word by the company it keeps" [Firth, 1957]
  - The distribution of words co-occurring (context) with a given word X is characteristic for X
  - To learn about X, look at its context
  - If X and Y are semantically similar, also their contexts are similar
  - If X and Y are a bit different, also their contexts will be a bit different
  - Holds in all domains and all corpora of sufficient size
- Central idea: Represent a word by its context
- For similarity: Compare contexts, not strings
- How can we do this efficiently and effectively?

Naive Approach

- Given a large corpus D and a vocabulary K
- Define a context window (typically sentence)
- Represent every $k \in K$ as a $|K|$-dimensional vector v_k
  - Find set W of all context windows containing k
  - For every $k' \neq k$, count frequency of k' in W: $v_k[k'] = freq(k', W)$
  - May be normalized, e.g. tf*idf
- Similarity: Compute cosine similarity between word-vectors
- Problem: Our model for each $d \in D$ grew from $|K|$ to $|K|^2$
  - Infeasible
  - We need an efficient and conservative dimensionality reduction
    - Efficient: Fast to compute; conservative: Distances are preserved

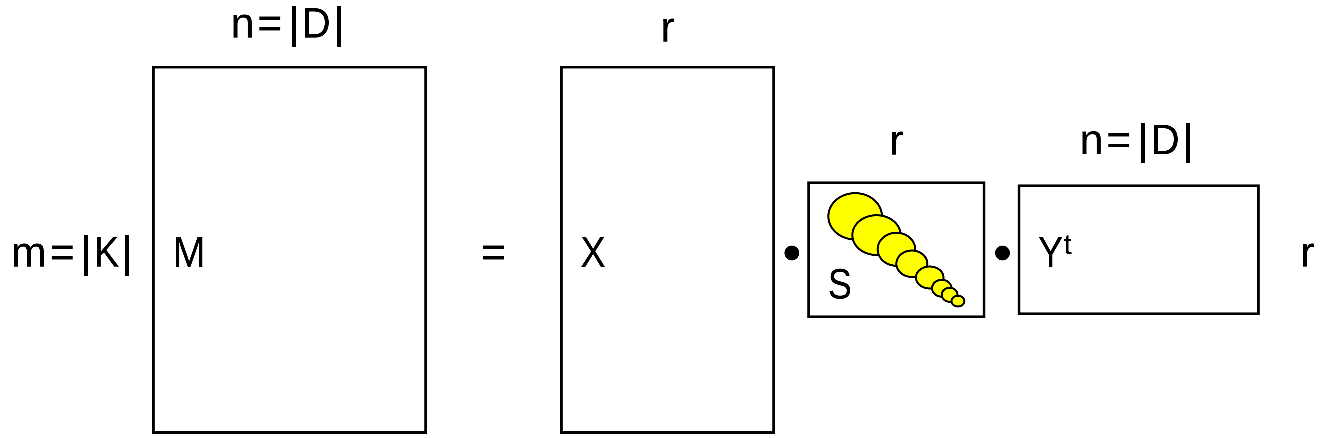
Latent Semantic Indexing

- Recall from Information Retrieval ...
- Goal: Represent documents as a distribution over concepts
  - "Concepts" should be computed automatically
  - LSI models concepts as linear combinations of document/term vectors with certain properties
  - Number of concepts is a hyper parameter
  - Search in concept space, not in term space
- Start from *term-document matrix M*
- Approximate M by a particular M'
  - M' has much less dimensions than M
  - M' should abstract from terms to concepts
  - M' should be such that $M'^t * q \approx M^t * q$
    - Produce the least error among all M' of the same dimensionality

Begriff	Dokument 1	Dokument 2	Dokument 3
Access	1	0	0
Document	1	0	0
Retrieval	1	0	1
Information	0	1	1
Theory	0	1	0
Database	1	0	0
Indexing	1	0	0
Computer	0	1	1

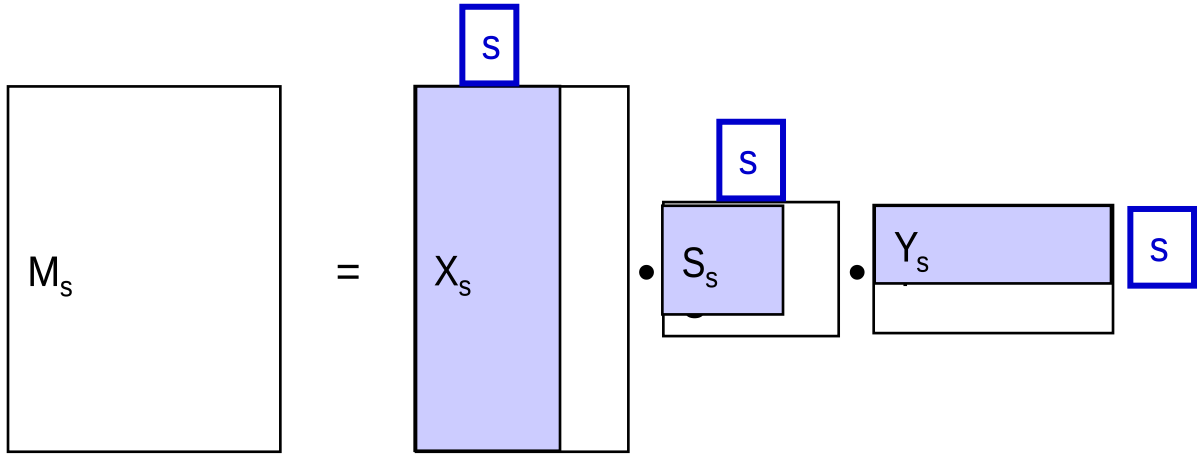
Singular Value Decomposition (SVD)

- SVD decomposes a matrix into $M = X \bullet S \bullet Y^t$
  - S is the diagonal matrix of the singular values of M in descending order and has size r x r (with r=rank(M))
  - X is the matrix of Eigenvectors of $M \bullet M^t$
  - Y is the matrix of Eigenvectors of $M^t \bullet M$
  - This decomposition is unique and can be computed in $O(r^3)$
    - Use approximations in practice



Approximating M

- LSI: Use SVD to approximate M
- Fix some s<r; Compute $M_s = X_s \bullet S_s \bullet Y_s^t$
  - X_s : First s columns in X
  - S_s : First s columns and first s rows in S
  - Y_s : First s rows in Y
- M_s is the matrix where $||M-M_s||_2$ is minimal
- Columns in Y_s^t are low-dimensional representations of docs



Usage and Problem

- We can apply the same math to the term-term correlation matrix (computed as $M*M^t$)
- This would yield low-dimensional vectors for each term
- But: We cannot compute anything that requires $O(|K|^3)$

M

	1	2	3	4	5
A	1	1	1		
B	1	1	1		1
C		1	1		
D				1	1

•

M^t

	A	B	C	D
1	1	1		
2	1	1	1	
3	1	1	1	
4				1
5		1		1

=

Term correlation matrix

	A	B	C	D
A	3	3	2	0
B	3	4	2	1
C	2	2	2	0
D	0	1	0	2

Table of Contents

- A very brief introduction to Neural Networks
- Word Semantic
- Word Embeddings (with Word2Vec)
- Applications

Word Embeddings

- Very popular technique since app. 2015
- Goal: Learning word vectors (“word embeddings”)
  - Low dimensional – typically 100-500 (a hyper parameter)
  - Unsupervised learning – may use extremely large corpora
  - Specific techniques to scale-up training (e.g. GPUs)
  - Can be precomputed and used without re-training in apps
- Approach: Use Machine Learning, not algebra
  - Though the border is not clear at all

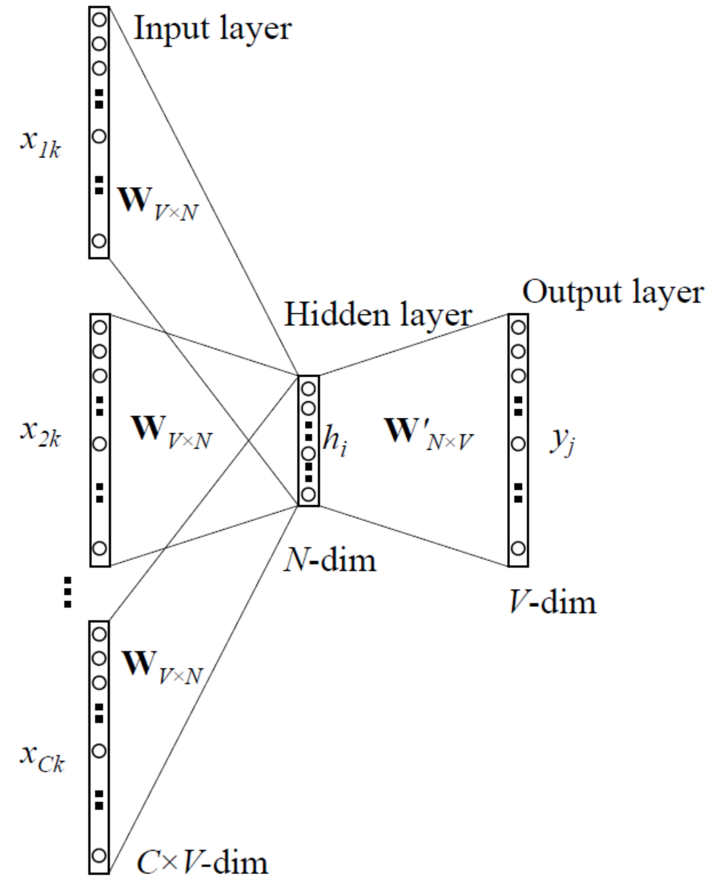
Word2Vec [Mikolov et al. 2013]

- Recall language models
  - Goal: Given a prefix of a sentence, predict next word
  - Can be understood as multi-class classification problem
    - As many classes as words
  - We computed word probabilities using a simple N-gram model
- Idea of Word2Vec
  - Cast the problem as classification
  - Given the context of a word – predict the word
    - Obvously related to language modelling
    - Note the “context” – we are close to word embeddings

K2 is the second ? mountain in the world.

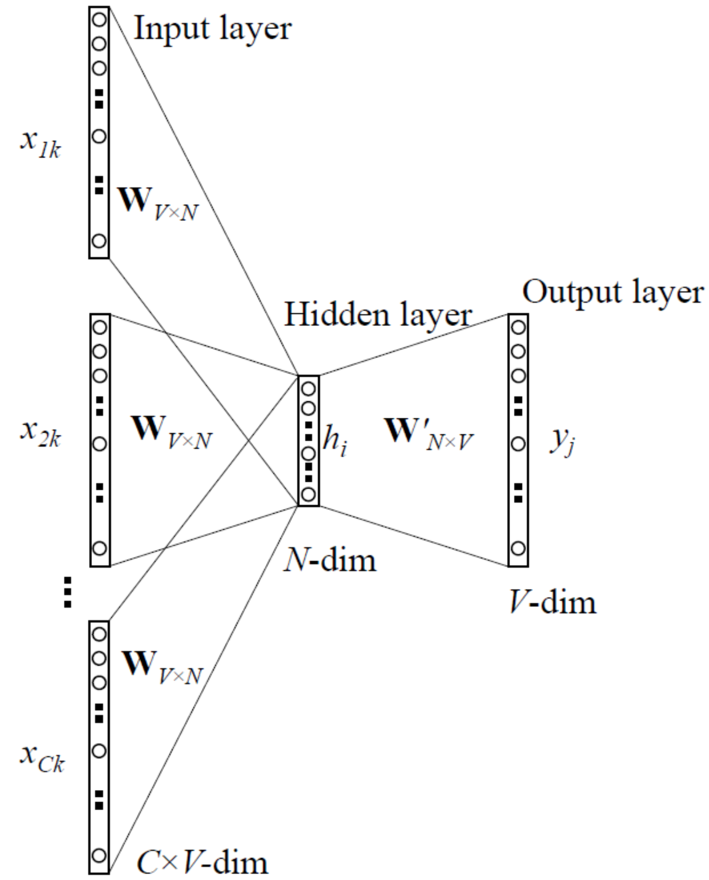
Architecture

- Fix dimensionality N, let V=|K|
- Fix context size C+1
- Solve problem by a 1-layer ANN
  - Input: C vectors of size V (context)
  - Hidden layer: N units
  - Output: V-dimensional layer (target)
- Parameters to learn
  - Input-hidden: V*N weights
    - “Parameter tying”
  - Hidden-output: N*V weights
- Activation functions
  - Hidden units: Weighted sum
  - Output units: softmax

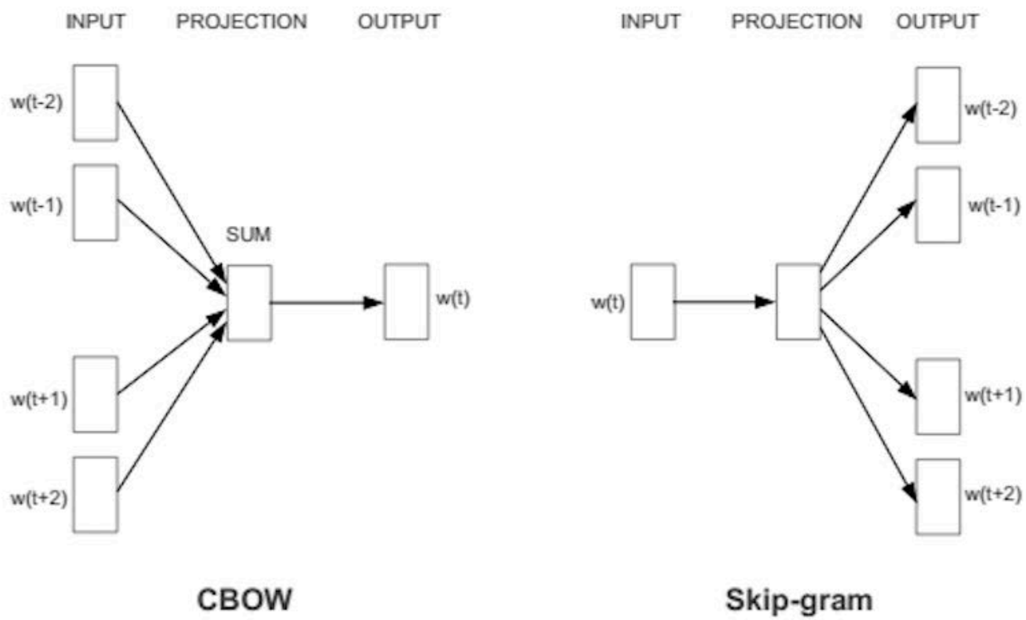


Learning Word2Vec

- Obtain a very large corpus
- Train ANN as usual
  - Random initialization
  - For every context / word
    - Use context as input, word as target
      - All in one-hot representation
    - Compute output, loss and gradient
    - Adjust weights
  - Iterate until convergence



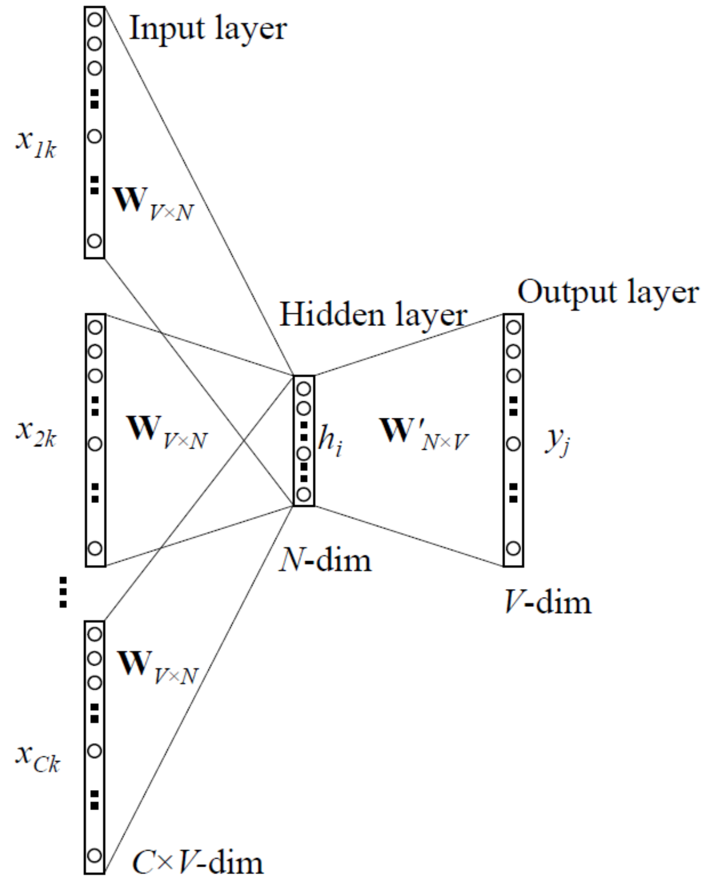
Two Options: CBOW or Skip-Gram



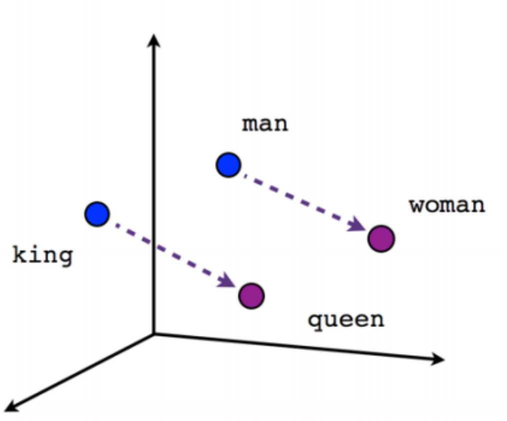
- Continuous Bag of Words: Predict word from its context
- Skip-Gram: Predict context from its center word
  - That’s actually one predictor per output word
  - Tends to produce more accurate results given large corpus

Word Embeddings?

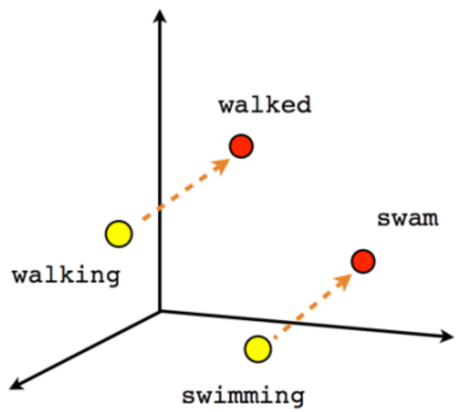
- Nice – but where are our word embeddings?
- Look at the output layer
  - Every word is one output unit
  - With N incoming weights
  - These weights form the word vector for the output word
  - The hidden units are the „concepts“
- Of course: Works only for known words
  - Alternative: Character level input



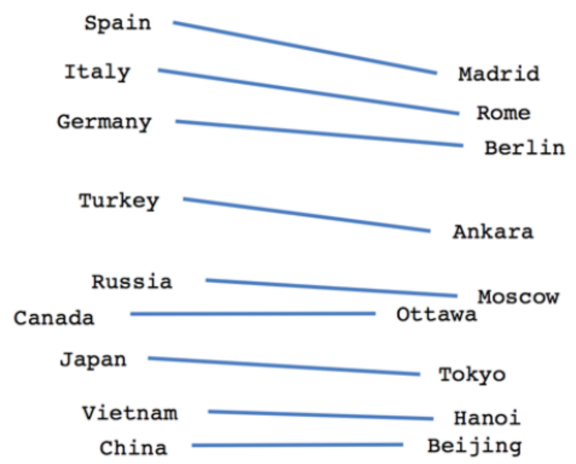
Does it Work?



Male-Female



Verb tense



Country-Capital

king – man ~ queen – woman

walking – walked ~ swimming – swam

Russia – Moscow ~ Vietnam – Hanoi

man - computer programmer ~ woman – homemaker
father - doctor ~ mother - nurse

Table of Contents

- A very brief introduction to Neural Networks
- Word Semantic
- Word Embeddings with Word2Vec
- Applications

Applications of Word Embeddings

- Word Embeddings can be used in essentially all places where words are represented as vectors
- Own experience: An extra 1-5% in F-measure (for NER)
  - That's a lot! Much more effect than classification method
- Very active research area – new papers appear daily
- "Best" methods still rather unclear
- Some examples

Word Embeddings and NER

- Recall NER using token classification
  - Token is represented as feature vector, classes are IOB
  - Features encode the token itself and context words
    - Traditionally: All in one-hot encoding
- Using word embeddings: Represent token and context words using their (precomputed) embeddings
  - Advantage: If token is semantically similar to a token tagged in the training data – additional evidence
  - In the traditional model, the lack of semantics was circumvented by using syntactic features (greek letters, certain suffixes, case, ...) presumably correlated to word semantics
  - Now, we can directly encode word semantics

Word Embeddings and Text Classification

- Recall, for instance, a SVM for classification
  - Every document is a vector of features (tf*idf)
  - SVM finds max-margin separating hyperplane (binary classification)
  - Hyperplane is some linear combination of feature values, i.e. words
- Classification and word embeddings
  - Not so simple; we cannot give a SVM a vector instead of a value
    - Wouldn't help: SVM doesn't compare values in different dimensions
  - Simple: Sum up all word vectors in a doc
    - Generates a low dimensional, "semantically aggregated" doc vector
  - Alternative: Directly learn "doc embeddings"
  - Alternative: Cluster embeddings per doc and use matching quality between clusters as distance in k-NN [or as kernel for a SVM]
  - Alternative: Compute minimal matching between sets of embeddings of two docs and use as distance in k-NN
  - ...

Literature

- LeCun, Y., Bengio, Y. and Hinton, G. (2015). "Deep Learning." Nature 521.
- Goodfellow & Bengio (2016): “Deep Learning”, MIT Press
  - See http://www.deeplearningbook.org/
- Mikolov, Sutskever, Chen, Corrado, Dean, J. (2013): “Distributed representations of words and phrases and their compositionality”, Advances in neural information processing systems
  - >6000 citations until 12/2017
- Mikolov, Chen, Corrado, Dean (2013): “Efficient estimation of word representations in vector space”. arXiv:1301.3781