Machine Learning – Lecture 16

Word Embeddings

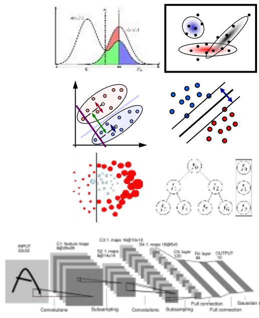
17.01.2019

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

Course Outline

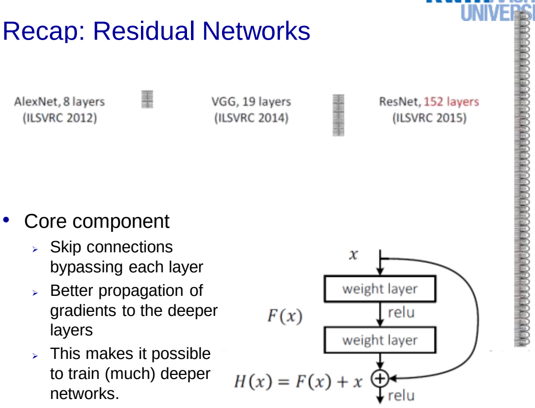
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Random Forests
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks



Topics of This Lecture

- Recap
  - ResNets
  - Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Outlook: Recurrent Neural Networks

Recap: Residual Networks



- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - This makes it possible to train (much) deeper networks.

Recap: Analysis of ResNets

- The effective paths in ResNets are relatively shallow
  - Effectively only 5-17 active modules
- This explains the resilience to deletion
  - Deleting any single layer only affects a subset of paths (and the shorter ones less than the longer ones).
- New interpretation of ResNets
  - ResNets work by creating an ensemble of relatively shallow paths
  - Making ResNets deeper increases the size of this ensemble
  - Excluding longer paths from training does not negatively affect the results.

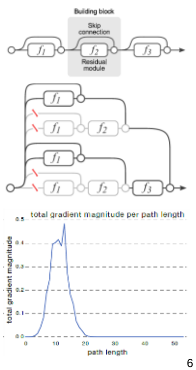
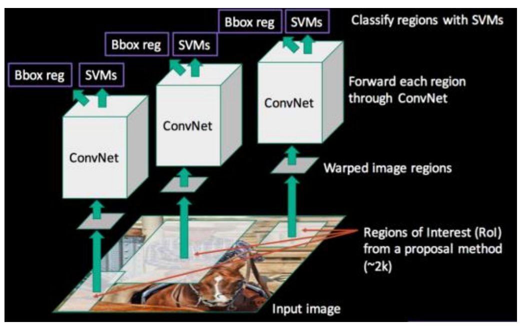


Image source: Veit et al., 2016

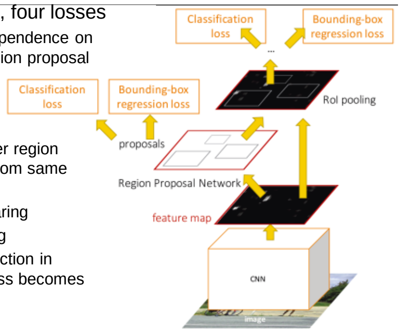
Recap: R-CNN for Object Detection



Slide credit: Ross Girshick

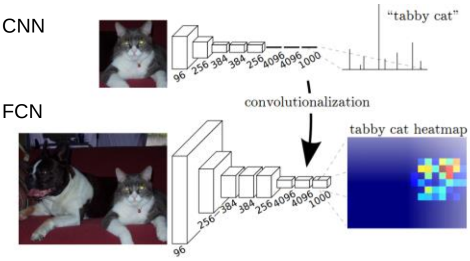
Recap: Faster R-CNN

- One network, four losses
  - Remove dependence on external region proposal algorithm.
  - Instead, infer region proposals from same CNN.
  - Feature sharing
  - Joint training
  - ⇒ Object detection in a single pass becomes possible.



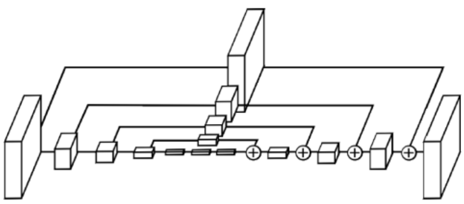
Recap: Fully Convolutional Networks

- CNN
- FCN



- Intuition
  - Think of FCNs as performing a sliding-window classification, producing a heatmap of output scores for each class

Recap: Semantic Image Segmentation



- Encoder-Decoder Architecture
  - Problem: FCN output has low resolution
  - Solution: perform upsampling to get back to desired resolution
  - Use skip connections to preserve higher-resolution information

Topics of This Lecture

- Recap
  - ResNets
  - Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Outlook: Recurrent Neural Networks

Neural Networks for Sequence Data

- Up to now
  - Simple structure: Input vector → Processing → Output
- In the following, we will look at sequence data
  - Interesting new challenges
  - Varying input/output length, need to memorize state, long-term dependencies, ...
- Currently a hot topic
  - Early successes of NNs for text / language processing.
  - Very good results for part-of-speech tagging, automatic translation, sentiment analysis, etc.
  - Recently very interesting developments for video understanding, image+text modeling (e.g., creating image descriptions), and even single-image understanding (attention processes).

Motivating Example

- Predicting the next word in a sequence
  - Important problem for speech recognition, text autocorrection, etc.
- Possible solution: The trigram (n-gram) method
  - Take huge amount of text and count the frequencies of all triplets (n-tuples) of words.
  - Use those frequencies to predict the relative probabilities of words given the two previous words

$$\frac{p(w_3 = c | w_2 = b, w_1 = a)}{p(w_3 = d | w_2 = b, w_1 = a)} = \frac{\mathrm{count}(abc)}{\mathrm{count}(abd)}$$

  - State-of-the-art until not long ago...

Problems with N-grams

- Problem: Scalability
  - We cannot easily scale this to large N.
  - The number of possible combinations increases exponentially
  - So does the required amount of data
- Problem: Partial Observability
  - With larger N, many counts would be zero.
  - The probability is not zero, just because the count is zero!
  - ⇒ Need to back off to (N-1)-grams when the count for N-grams is too small.
  - ⇒ Necessary to use elaborate techniques, such as Kneser-Ney smoothing, to compensate for uneven sampling frequencies.

Let's Try Neural Networks for this Task

"softmax" units (one per possible next word)

internal NN structure

index of word at t-2 | index of word at t-1

- Important issues
  - How should we encode the words to use them as input?
  - What internal NN structure do we need?
  - How can we perform classification (softmax) with so many possible outputs?

Neural Probabilistic Language Model

"softmax" units (one per possible next word)

skip-layer connections

units that learn to predict the output word from features of the input words

learned distributed encoding of word t-2 | learned distributed encoding of word t-1

table look-up | table look-up

index of word at t-2 | index of word at t-1

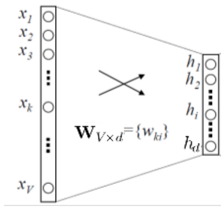
- Core idea
  - Learn a shared distributed encoding (word embedding) for the words in the vocabulary.

Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A Neural Probabilistic Language Model, In JMLR, Vol. 3, pp. 1137-1155, 2003.

Image source: Geoff Hinton

Word Embedding

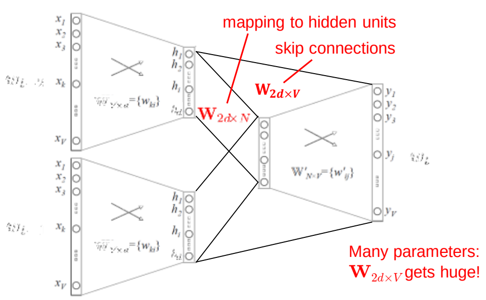
- Idea
  - Encode each word as a vector in a d-dimensional feature space.
  - Typically, $V \sim 1\mathrm{M}$, $d \in (50, 300)$



- Learning goal
  - Determine weight matrix $\mathbf{W}_{V\times d}$ that performs the embedding.
  - Shared between all input words
- Input
  - Vocabulary index $\mathbf{x}$ in 1-of-K encoding.
  - For each input $\mathbf{x}$, only one row of $\mathbf{W}_{V\times d}$ is needed.
  - ⇒ $\mathbf{W}_{V\times d}$ is effectively a look-up table.

Image source: Xin Rong, 2015

Word Embedding: Full Network



- Train on large corpus of data, learn $\mathbf{W}_{V\times d}$.
  - ⇒ Shown to outperform n-grams by [Bengio et al., 2003].

Visualization of the Resulting Embedding

(part of a 2.5D map of the most common 2500 words)

Image source: Geoff Hinton

Visualization of the Resulting Embedding

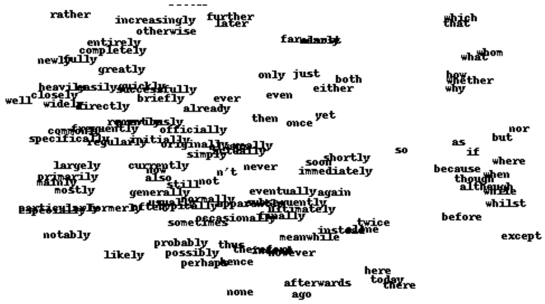
Machine Learning Winter '18

B. Leibe

21

Image source: Geoff Hinton

Visualization of the Resulting Embedding



Machine Learning Winter '18

B. Leibe

22

Image source: Geoff Hinton

Popular Word Embeddings

- Open issue
  - *What is the best setup for learning such an embedding from large amounts of data (billions of words)?*
- Several recent improvements
  - word2vec [Mikolov 2013]
  - GloVe [Pennington 2014]

  ⇒ Pretrained embeddings available for everyone to download.

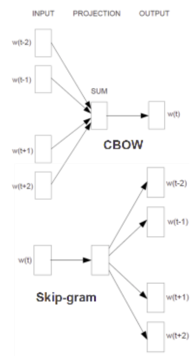
Machine Learning Winter '18

B. Leibe

23

word2vec

- Goal
  - Make it possible to learn high-quality word embeddings from huge data sets (billions of words in training set).
- Approach
  - Define two alternative learning tasks for learning the embedding:
    - "Continuous Bag of Words" (CBOW)
    - "Skip-gram"
  - Designed to require fewer parameters.



Machine Learning Winter '18

B. Leibe

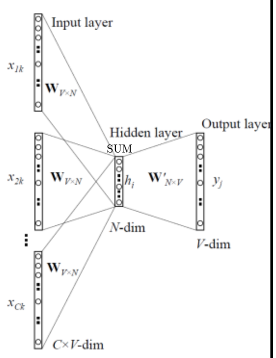
25

Image source: Mikolov et al., 2015

word2vec: CBOW Model

- Continuous BOW Model
  - Remove the non-linearity from the hidden layer
  - Share the projection layer for all words (their vectors are averaged)

  ⇒ Bag-of-Words model (order of the words does not matter anymore)



Machine Learning Winter '18

B. Leibe

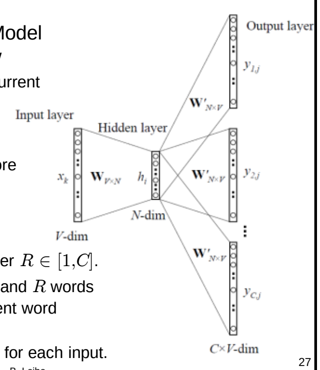
26

Image source: Xin Rong, 2015

word2vec: Skip-Gram Model

- Continuous Skip-Gram Model
  - Similar structure to CBOW
  - Instead of predicting the current word, predict words within a certain range of the current word.
  - Give less weight to the more distant words
- Implementation
  - Randomly choose a number $R \in [1,C]$.
  - Use R words from history and R words from the future of the current word as correct labels.

  ⇒ $R+R$ word classifications for each input.



Machine Learning Winter '18

B. Leibe

27

Image source: Xin Rong, 2015

Interesting property

- Embedding often preserves linear regularities between words
  - Analogy questions can be answered through simple algebraic operations with the vector representation of words.
- Example
  - What is the word that is similar to *small* in the same sense as *bigger* is to *big*?
  - For this, we can simply compute
    X = vec(*"bigger"*) – vec(*"big"*) + vec(*"small"*)
  - Then search the vector space for the word closes to X using the cosine distance.

  ⇒ Result (when words are well trained): vec(*"smaller"*).
- Other example
  - E.g., vec("King") – vec("Man") + vec("Woman") ≈ vec("Queen")

Evaluation on Analogy Questions

	Type of relationship	Word Pair 1		Word Pair 2	
semantic	Common capital city	Athens	Greece	Oslo	Norway
	All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
	Currency	Angola	kwanza	Iran	rial
	City-in-state	Chicago	Illinois	Stockton	California
	Man-Woman	brother	sister	grandson	granddaughter
syntactic	Adjective to adverb	apparent	apparently	rapid	rapidly
	Opposite	possibly	impossibly	ethical	unethical
	Comparative	great	greater	tough	tougher
	Superlative	easy	easiest	lucky	luckiest
	Present Participle	think	thinking	read	reading
	Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
	Past tense	walking	walked	swimming	swam
	Plural nouns	mouse	mice	dollar	dollars
	Plural verbs	work	works	speak	speaks

Image source: Mikolov et al., 2015

Results

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

- Results
  - word2vec embedding is able to correctly answer many of those analogy questions.
  - CBOW structure better for syntactic tasks
  - Skip-gram structure better for semantic tasks

Image source: Mikolov et al., 2015

Problems with 100k-1M outputs

- Weight matrix gets huge!
- Example: CBOW model
  - One-hot encoding for inputs

  ⇒ Input-hidden connections are just vector lookups.
  - This is not the case for the hidden-output connections!
  - State h is not one-hot, and vocabulary size is 1M.

  ⇒ $\mathbf{W}'_{N\times V}$ has 300×1M entries

  ⇒ All of those need to be updated by backprop.

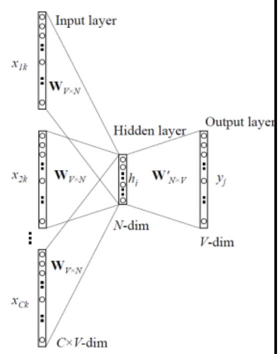


Image source: Xin Rong, 2015

Problems with 100k-1M outputs

- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs

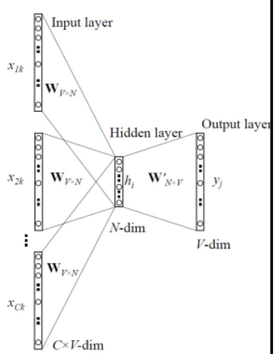
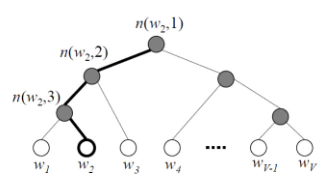


Image source: Xin Rong, 2015

Solution: Hierarchical Softmax



- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word w_0 as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_{n(w,j)} \cdot h$ at each node to decide whether to proceed with left or right child node.

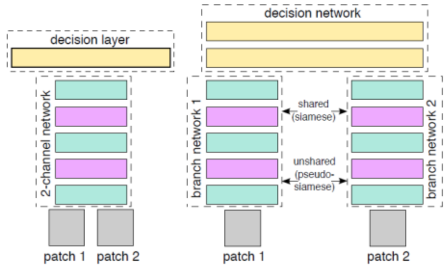
  ⇒ Decision based on output vector of hidden units directly.

Image source: Xin Rong, 2015

Topics of This Lecture

- Recap: CNN Architectures
- Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Embeddings in Vision
  - Siamese networks
  - Triplet loss networks
- Outlook: Recurrent Neural Networks

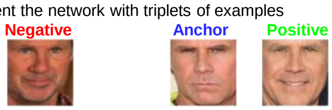
Siamese Networks



- Similar idea to word embeddings
  - Learn an embedding network that preserves (semantic) similarity between inputs
  - E.g., used for patch matching

Recap: Discriminative Face Embeddings

- Learning an embedding using a Triplet Loss Network
  - Present the network with triplets of examples



  - Apply triplet loss to learn an embedding $f(\cdot)$ that groups the positive example closer to the anchor than the negative one.

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2$$

⇒ Used with great success in Google's FaceNet face recognition

Triplet Loss – Practical Implementation

- Triplet loss formulation

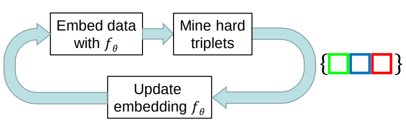
$$\mathcal{L}_{\text{tri}}(\theta) = \sum_{\substack{a,p,n \\ y_a = y_p \neq y_n}} [m + D_{a,p} - D_{a,n}]_+$$

- Practical Issue: How to select the triplets?
  - The number of possible triplets grows cubically with the dataset size.
  - Most triplets are uninformative

  ⇒ Mining hard triplets becomes crucial for learning.

  ⇒ Actually want *medium-hard* triplets for best training efficiency
- Popular solution: Offline hard triplet mining
  - Process the dataset to find hard triplets
  - Use those for learning
  - Iterate

Triplet Loss – Practical Implementation (2)

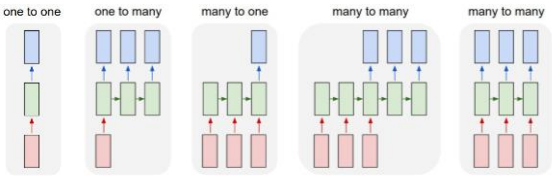


- Popular solution: Offline hard triplet mining
  - Process the dataset to find hard triplets
  - Use those for learning
  - Iterate

Topics of This Lecture

- Recap: CNN Architectures
- Applications of CNNs
- Word Embeddings
  - Neuroprobabilistic Language Models
  - word2vec
  - GloVe
  - Hierarchical Softmax
- Embeddings in Vision
  - Siamese networks
  - Triplet loss networks
- Outlook: Recurrent Neural Networks

Outlook: Recurrent Neural Networks



- Up to now
  - Simple neural network structure: 1-to-1 mapping of inputs to outputs
- Next lecture: Recurrent Neural Networks
  - Generalize this to arbitrary mappings

Image source: Andrej Karpathy

References and Further Reading

- Neural Probabilistic Language Model
  - Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A Neural Probabilistic Language Model, In JMLR, Vol. 3, pp. 1137-1155, 2003.
- word2vec
  - T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, ICLR'13 Workshop Proceedings, 2013.
- GloVe
  - Jeffrey Pennington, Richard Socher, and Christopher D. Manning, GloVe: Global Vectors for Word Representation, 2014.
- Hierarchical Softmax
  - F. Morin and Y. Bengio, Hierarchical probabilistic neural network language model. In AISTATS 2005.
  - A. Mnih and G.E. Hinton (2009). A scalable hierarchical distributed language model. In NIPS 2009.

References: Other Embeddings

- Face Embeddings
  - F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, in CVPR 2015.
  - A. Radford, L. Metz, S. Chintala, Unsupervise Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016.