**Neural Networks & NLP**

1. Natural Language Processing
   1. A field at the intersection of Artificial Intelligence, Linguistics, and Psycholinguistics
   2. Goal: for computers to “process” or “understand” natural language to carry out useful tasks, such as
   3. Performing useful tasks / answer questions (Apple SIRI, Amazon Alexa)
   4. Translate from one language to the other (Google Translate)
   5. Full understanding of human language (or even a proper definition of language meaning) still a distant goal.
2. Why is NLP difficult?
   1. Language is AMBIGUOUS
   2. Real headlines:
   3. Juvenile Court to Try Shooting Defendant
   4. Scientists study whales from space
   5. Language is DYNAMIC
   6. New words and constructions created every day
   7. Language is NOISY
   8. Happy bihday your majesty
   9. Language is ZIPFIAN
3. ML methods in NLP until 2012-13
   1. Starting from around 1990, NLP researchers moved from hand-coding algorithms for interpreting certain aspects of language to using machine learning methods that could learn how to carry out those types of interpretation from large sets of data (‘corpora’)
   2. The field moved on quite rapidly from using simpler ML methods such as decision trees to using generative methods (Naïve Bayes, Hidden Markov Models) and then discriminative methods (logistic regression / maximum entropy, Conditional Random Fields)
   3. But possibly the oldest ML methods in AI, neural networks (originally developed in the ‘40s) remained little used except for speech
4. The return of neural networks
   1. Starting from 2010, NN models started outperforming other types of ML in NLP
   2. Some reasons:
      1. Large amounts of training data became available
      2. Faster machines and multicore CPU/GPUs
      3. New models, algorithms, ideas
5. Deep learning for speech Deep Learning for Speech
   1. The first breakthrough results of “deep learning” on large datasets happened in speech recognition
   2. Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition Dahl et al. (2010)
6. Deep learning for vision
   1. Deep Learning for Computer Vision Most deep learning groups have focused on computer vision (at least till 2 years ago) The breakthrough DL paper: ImageNet Classification with Deep Convolutional Neural Networks by Krizhevsky, Sutskever, & Hinton, 2012, U. Toronto. 37% error red.
7. Syllabus
   1. Week 1: Word models with NN & Intro to NNs
   2. Week 2: Sequence models with Recursive Neural Networks (RNN) and LSTMs
   3. Week 3: Text Classification with NNs
   4. Week 4: Sentiment analysis
   5. Week 5: Machine Translation 1: the Encoder/Decoder model
   6. Week 6: Machine Translation 2: Attention
   7. Week 7: Reading Week
   8. Week 8: Transformers and BERT
   9. Week 9: NER and Information Extraction
   10. Week 10: Coreference
   11. Week 11: Conversational Agents & Dialogue 1
   12. Week 12: Conversational agents & Dialogue 2
8. Times & locations
   1. Pre-recorded lectures from last year
      1. On QM+
   2. Lecture+Q&A (online):
      1. Tuesdays, 9-11
   3. Labs:
      1. Wednesdays, 3-5 (Engineering 3.52)
      2. Thursdays, 4-6 (Engineering 3.56)
9. Neural networks as linear classifiers: the perceptron
   1. Model network as a graph with cells as nodes and synaptic connections as weighted edges from node i to node j, w\_ji
   2. Model net input to cell as:
      1. Net\_j = sum of (w\_ji \* o\_i)
   3. Cell output is (T\_j is threshold for unit j):
      1. o\_j = 0 if net\_i < T\_j
      2. o\_j = 1 if net\_i >= T\_j
10. The perceptron as a linear classifier
    1. Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.  
         
       w\_12 \* o\_2 + w\_13 \* o\_3 > T\_1  
         
       o\_3 > (-(w\_12/w\_13) \* o\_2) + (T\_1/w\_13)
11. Modern networks: non-linear output functions
    1. Need non-linear output function to move beyond linear functions.
    2. A multi-layer linear network is still linear.
    3. Traditional solution is to use the non-linear, differentiable sigmoidal “logistic” function:  
         
       o\_j = 1 / (1 + e^{-(net\_j – T\_j)})
    4. The derivative of the sigmoid is just:
       1. y\_i \* (1 - y\_i) [We can derive gradient decent rules to train]
12. Multi-Layer Networks
    1. A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.
    2. The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.
13. Real Neural Learning
    1. Synapses change size and strength with experience.
    2. Hebbian learning: When two connected neurons are firing at the same time, the strength of the synapse between them increases.
    3. “Neurons that fire together, wire together.”
14. Neural Network Learning
    1. Each (artificial) neural network layer performs a tensorial operation on its input: output = F(dot(W,input) + b)
       1. where W and b are tensors specifying the WEIGHTS or TRAINABLE PARAMETERS of the network
    2. The objective of learning is to acquire the values of these tensors
    3. The ANN learning algorithms do this by following the Hebbian learning idea - `strengthening’ (= increasing) certain weights and `weakening’ (= decreasing) other ones
15. Artificial Neural Network Learning
    1. Perceptron: Initial algorithm for learning simple neural networks (single layer) developed in the 1950’s.
    2. Backpropagation: More complex algorithm for learning multi-layer neural networks developed in the 1980’s.
16. Learning by gradient descent
    1. Both the perceptron algorithm and backpropagation are examples of an approach to learning called GRADIENT DESCENT
    2. Gradient descent is learning a function iteratively, by progressively modifying its parameters to minimize error.
17. Reminder: gradient descent for (linear) regression:
    1. The (linear) function that we want to learn:  
         
       h\_theta(x) = theta\_0 + theta\_1 \* x  
       Parameters: theta\_0, theta\_1
18. Reminder: gradient descent for (linear) regression
    1. The cost (or loss) function (Mean Square Error):   
         
       J(theta\_0, theta\_1) = (1/2m) \* sum of (h\_theta(x^{(i)}) – y^{(i)})^2
    2. The goal: minimise J(theta\_0, theta\_1)
19. Gradient descent for NNs
    1. Randomly choose the initial weights
    2. While error (E) is too large
    3. For each training pattern
    4. Apply the inputs to the network
    5. Calculate the error at the outputs
    6. Use the output error to compute error signals for pre-output layers
    7. Use the error signals to compute weight adjustments
    8. Apply the weight adjustments
    9. Periodically evaluate the network performance
    10. Key question: How do we compute the adjustments?
20. Derivatives and gradients
    1. A GRADIENT is the generalization to multi-input functions (TENSORS) of the notion of DERIVATIVE
    2. Given a differentiable (smooth and continuous) function f(x), the derivative f’(x) gives us the SLOPE of f(x)
    3. So we can reach a minimum by ‘travelling along’ the direction indicated by the derivative
    4. In learning, we reach the minimum of the LOSS (ERROR) FUNCTION by travelling along its curve
21. Gradient descent algorithm:
    1. Repeat until convergence:
       1. theta\_j := theta\_j – alpha \* (del J(theta\_0, theta\_1) / del theta\_j)
       2. Simultaneously update j=0 and j=1
22. Embeddings and lexical semantics
    1. One of the key technical developments leading to neural networks becoming the dominant paradigm for NLP in the last ten is the idea of representing the input to NLP models – words and sentences – using dense vectorial representations called EMBEDDINGS
23. Lexical semantics and lexical resources
    1. Quite a lot of work in (lexical) semantics and lexicography in the ‘70s, ‘80s and ’90s attempted to develop theories of lexical meaning encoding meaning relations, and create resources that could be used by NLP systems to associate more complex meanings to words
    2. Possibly the most rigorous approach of this type was pioneered by Pustejovsky and colleagues
       1. Reference: J. Pustejovsky (1995), The Generative Lexicon, MIT Press
       2. Resource: The LKB, <https://www.cl.cam.ac.uk/research/nl/acquilex/>
    3. The resource best known and most widely used in NLP: WordNet
       1. Reference: C. Fellbaum, ed. (1998) WordNet: An Electronic Lexical Database. MIT Press.
       2. Website: <https://wordnet.princeton.edu/>
24. Towards dense word representations: distributional semantics
    1. You can get a lot of value by representing a word by means of its neighbors
       1. “You shall know a word by the company it keeps” (J. R. Firth 1957: 11)
    2. One of the most successful ideas of modern statistical NLP
25. Words as vectors in a word space: count-based vectors from a co-occurrence matrix
    1. Corpus = {“I like deep learning”, “I like NLP”, “I enjoy flying”}
    2. Context = previous word and next word

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Counts | I | Like | Enjoy | Deep | Learning | NLP | Flying | . |
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| Like | 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| Enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| Learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| Flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| . | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

1. LSA: dense vectors using Singular Value Decomposition:
   1. The problem with this method, is that we may end up with matrices having billions of rows and columns, which makes SVD computationally restrictive.
2. Count-based distributional semantics: an evaluation
   1. LSA-like distributional semantics models were the dominant paradigm for data-driven lexical semantics between 1990 and 2010
   2. They achieved good results at tasks involving word similarity
      1. In particular, they achieved near-human performance at the TOEFL synonymy task
   3. But only mediocre results for other aspects of NLP.
3. Word embeddings
   1. The dense vector word representations developed in distributional semantics were the precursors of word embeddings
   2. Bengio et al (2003) proposed a `Neural Probabilistic Language Model’ whose key characteristic is that it uses an EMBEDDING LAYER to learn ‘dense word representations’ aka word embeddings
   3. Two types
      1. Trained while learning a model for a task (Bengio et al, 2003; Collobert & Weston, 2008, 2011)
      2. Pre-computed (Mikolov et al, 2010, 2013; Pennington et al, 2014)
4. Applications of word prediction • Spelling checkers • Mobile phone texting • Speech recognition • Handwriting recognition • Disabled users
5. A statistical formulation of word prediction
   1. The basic idea underlying the statistical approach to word prediction is to use the probabilities of SEQUENCES OF WORDS to choose the most likely next word / correction of spelling error
   2. I.e., to compute: P(w given W1 …. W\_N-1)
   3. For all words w, and predict as next word the one for which this (conditional) probability is highest.
6. From counting to predicting
   1. Mikolov et al (2013) and others discovered that when training a Bengio-style neural language models
   2. … You obtain as a byproduct in the hidden layer a distributional representation for w\_t that performs better as a pretrained word embedding for many tasks that the distributional representation obtained by previous approaches (Baroni et al, 2014)
   3. The research led to the development of Word2Vec, the first neural lexical semantics model
7. Word2Vec key ideas
   1. Focus: develop models that are SCALABLE and thus can be used to train word representations from very large scale corpora
      1. Use the LANGUAGE MODEL TASK as a way to train very large models without relying on annotated data (task: predict probability of nearby word)
   2. Two models:
      1. Continuous Bag of Words (CBOW): predict target word from bag-of-words context (as in Bengio et al, 2003)
      2. SKIPGRAM: predict context from target word
   3. Assign to each word TWO representations: the EMBEDDING vector proper and the CONTEXT vector
   4. Use the SIMILARITY between the vectors to compute probability of proximity
   5. The basic version of this approach is intractable; two solutions to make it tractable:
      1. NEGATIVE SAMPLING
      2. Hierarchical softmax
8. Language modelling with skip-gram
   1. Predict surrounding words of each word
      1. outside context words in window of size 2: P(w\_t-2 given w\_t), P(w\_t-1 given w\_t)
      2. center word at position t
      3. outside context words in window of size 2: P(w\_t+1 given w\_t), P(w\_t+2 given w\_t)
9. Computing the probability of the context word given the target word:
   1. Example windows and process for computing; P(w\_t+j given w\_t)
10. Computing the probability of a context word:
    1. P(o given c) = exp^{u^T\_O \* v\_c} / sum of(exp^{u^T\_w \* v\_c})
    2. Where
       1. exp^{u^T\_O \* v\_c}
          1. exponentiation makes anything positive
          2. u^T\_O \* v\_c : Dot product compares similarity of o and c; Larger dot product = larger probability
       2. sum of(exp^{u^T\_w \* v\_c})
          1. Normalize over entire vocabulary to give probability distribution
    3. This is an example of the softmax function:
       1. Softmax(x\_i) = exp^{x\_i} / sum of(exp^{x\_j})
       2. The softmax function maps arbitrary values x\_i to a probability distribution p\_i.
       3. “max” because amplifies probability of largest
       4. “soft” because still assigns some probability to smaller
       5. Frequently used in Deep Learning
11. Word2vec parameters:
    1. The model makes the same predictions at each position
    2. We want a model that gives a reasonably high probability estimate to all words that occur in the context (at all often)
12. Word2vec objective function
    1. For each position t = 1, … , T), predict context words within a window of fixed size m, given center word w\_j. Data likelihood:
    2. Likelihood = L(theta) = product sum of (P (w\_t+j given w\_t;theta))
       1. theta is all variables to be optimized
    3. The objective function J(theta) is the (average) negative log likelihood (sometimes called a cost or loss function):
       1. J(theta) = -1/T \* log L(theta)
13. Gradient descent with word2vec:
    1. To train a model, we gradually adjust parameters to minimize a loss
    2. Recall: theta represents all the model parameters, in one long vector
    3. In our case, with d-dimensional vectors and V-many words, we have:
    4. Remember: every word has two vectors
    5. We optimize these parameters by walking down the gradient
    6. We compute all vector gradients!
14. Reminder: gradient descent
    1. Optimization: Gradient Descent
    2. We have a cost function J(theta) we want to minimize
    3. Gradient Descent is an algorithm to minimize J(theta)
    4. Idea: for current value of theta, calculate gradient of J(theta), then take small step in direction of negative gradient. Repeat
    5. Update equation (in matrix notation):
       1. Theta^new = theta^old – alpha \* Nabla\_theta \* J(theta)
    6. Update equation (for single parameter):
       1. Theta^new\_j = theta^old\_j – alpha \* del J(theta) / del theta^old\_j
15. Problem: the softmax can be expensive:
    1. Computing the probabilities using the full form of the softmax
    2. Requires computing the denominator for all pairs of words in the vocabulary, which is very expensive for modern vocabularies, often in the order of 10^7
16. Making the model scalable
    1. Several approximated methods for probability computation were tested by Mikolov et al (2013c):
    2. Subsampling
    3. NEGATIVE SAMPLING
    4. Hierarchical softmax
17. Subsampling
    1. In large corpora, highly frequent words (“the”, “a”) occur hundred of millions of times, but their word representation converges very quickly.
    2. To avoid recomputing the probability of such words, discard (= do not recompute the representation of ) words with probability given by:  
         
       P(w\_i) = 1 / sqrt(t / f(w\_i))
18. Negative sampling;
    1. Instead of computing the full: log P(w\_O given w\_i)
    2. Draw k negative examples from the probability distribution over words, and compute the probability that object word is distinguished from these using:
    3. Log sigma(v’\_wO T v\_w1) + sum of (E\_wi tilde P\_n(w)) \* log sigma(-v’\_wi T v\_wI)
    4. Where
       1. Log sigma(v’\_wO T v\_w1): This term is the probability of the words actually occurring next to the target word – want to maximize these
       2. log sigma(-v’\_wi T v\_wI): Sample k other words which do NOT occur next to the target according to P(w), and MINIMIZE their prob
19. Assessing Word2Vec
    1. Mikolov et al (2013a): Semantic / syntactic word relationship test
       1. Standard test for distributional models: check that semantically / syntactically closest words are also closest according to the model
          1. “France” more similar to “UK” than to “London”
          2. “small” more similar to “big” than to “red”
       2. New test (sometimes called ‘analogy test’)
          1. Given words that stand in a certain relation (eg France and Paris) find the word that stands in the same relation to a third word (eg UK)
          2. This is done using VECTOR ADDITION:
             1. compute vector(Paris) – vector(France) + vector(UK)
             2. Find vector that is closest
20. Baroni et al 2014
    1. Compared a number of versions of the Skip-gram model with a number of traditional distributional models on an extensive range of tests
21. Models tested
    1. COUNT models (cnt) – 36 in total
       1. Full and compressed (LSA) models
       2. Several weighing measures including PMI
       3. Different window sizes / vector sizes
    2. PREDICT models (pre) – 48 in total
       1. CBOW
       2. Windows: 2 & 5 on either side
       3. Vector dimensions: from 200 to 500
       4. Hierarchical softmax, negative sampling, subsampling
22. Tests
    1. Semantic relatedness (e.g., Rubenstein & Goodenough dataset)
       1. E.g., King/Queen
    2. Synonym detection (e.g., TOEFL dataset)
       1. E.g., levied / imposed
    3. Concept categorization (e.g., Almuhareb & Poesio)
       1. 402 concepts from all 21 WordNet top categories
    4. Selectional preferences (e.g., Baroni & Lenci)
       1. E.g., <eat, food>
    5. Analogy (e.g., Mikolov et al)
23. A word of caution: Levy et al, 2015
    1. Levy et al (2015) found that a lot of the advantages of word2vec came not from the fact that it’s predicting, but from the ideas that come with it:
       1. Hyperparameter tuning
24. Multilingual skip-gram: FastText
    1. A library from Facebook for building word embeddings for MULTIPLE LANGUAGES
    2. It takes SUB-WORD information into account by representing words as bags of CHARACTER N-GRAMS
25. Other models
    1. GLOVE (Pennington, Socher & Manning, 2014)
       1. Use RATIOS of co-occurrence probabilities
    2. More recent models: ELMO (Peters et al, 2018) and BERT (2019)