NLP Tasks and Applicaitons Using Word Embeddings End-to-end problem solving

Venelin Kovatchev

Lecturer in Computer Science

v.o.kovatchev@bham.ac.uk

Outline

- NLP Applications

- Using embeddings. Compositionality.

- End to end neural models

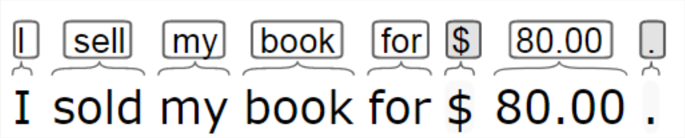
NLP Applications
Classical NLP Pipeline and Features

What we have learned so far

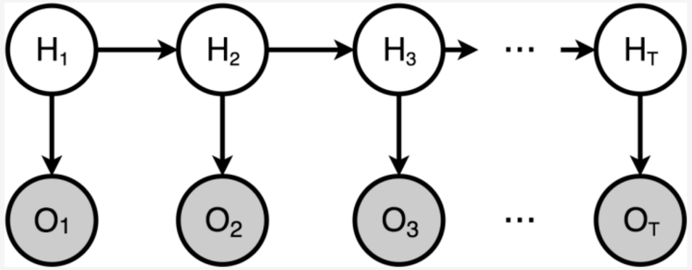
- Tokenization
- Pos tagging
- Chunking / Syntactic parsing / Dependency parsing
- Word co-occurrence and distributional semantic models
- Word embeddings

“Linguistic” tasks: Text tokenization and POS tagging

- Tokenization: output is a text segmented in tokens
  - Regular expressions, BPE



- POS tagging: output is a sequence of hidden states:
  - noun, verb, adjective
  - Hidden Markov Models (HMM



"Linguistic" tasks: Chunking and constituent analysis

- Grouping words together based on their shared "function" in the text
- Find all groups that "function together" in the sentence
- I went to the movies with a friend who I know from high school.
  - **[I]** went to the movies with a friend who I know from high school.
  - I **[went]** to the movies with a friend who I know from high school.
  - I went **[to the movies]** with a friend who I know from high school.
  - I went to the movies **[with a friend who I know from high school]** .

“Linguistic” tasks: The problem of syntax

“What combinations can we get with the constituents “dog”, “human”, and “bites”

- “Dog bites human” – (statistically) most common
- “Human bites dog” – meaningful, possible, but unlikely
- “Bites dog human”, “Human dog bites”, etc. – ungrammatical
- Same constituents, different rules -> different (im-)possible complex expression

"Linguistic" tasks: Full syntactic parsing

- "Colorless green ideas sleep furiously in love

S -> NP VP

NP -> A N

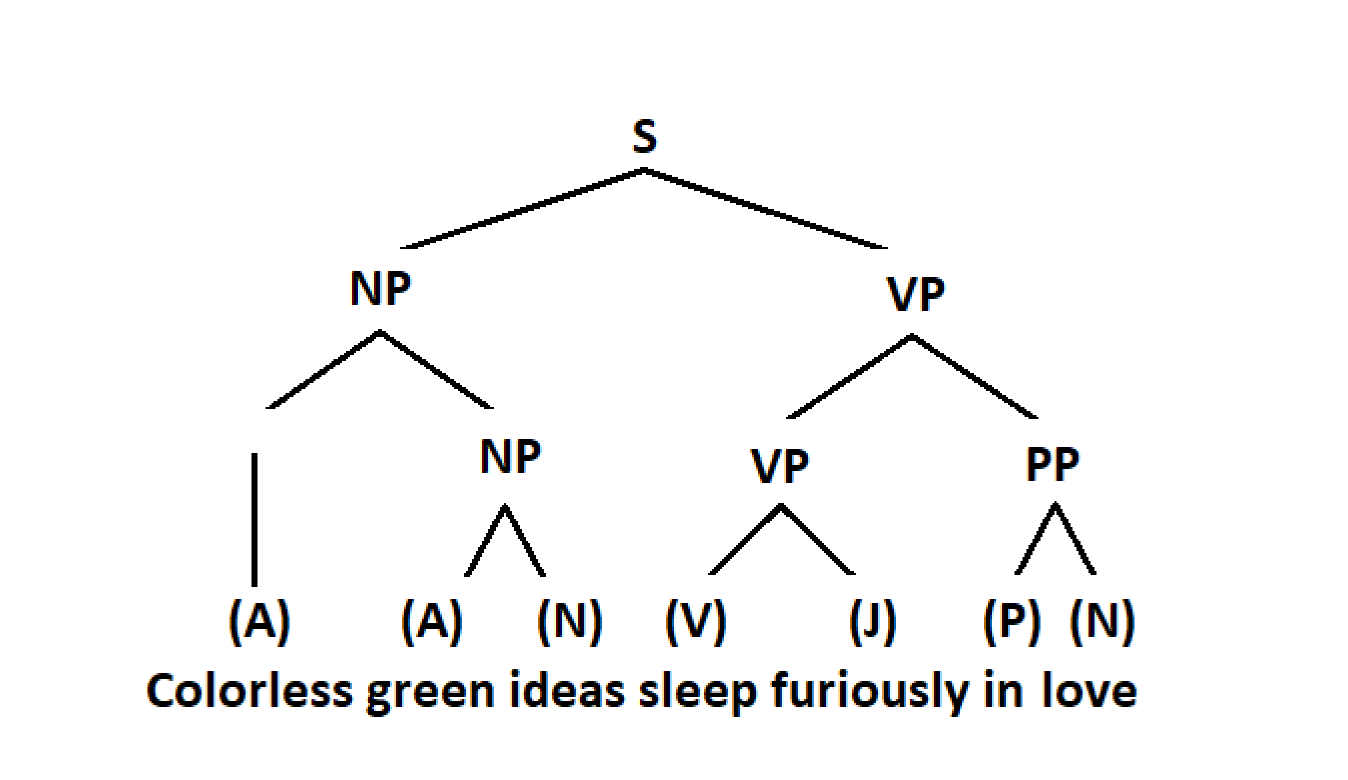
NP -> A NP

VP -> V J

VP -> VP PP

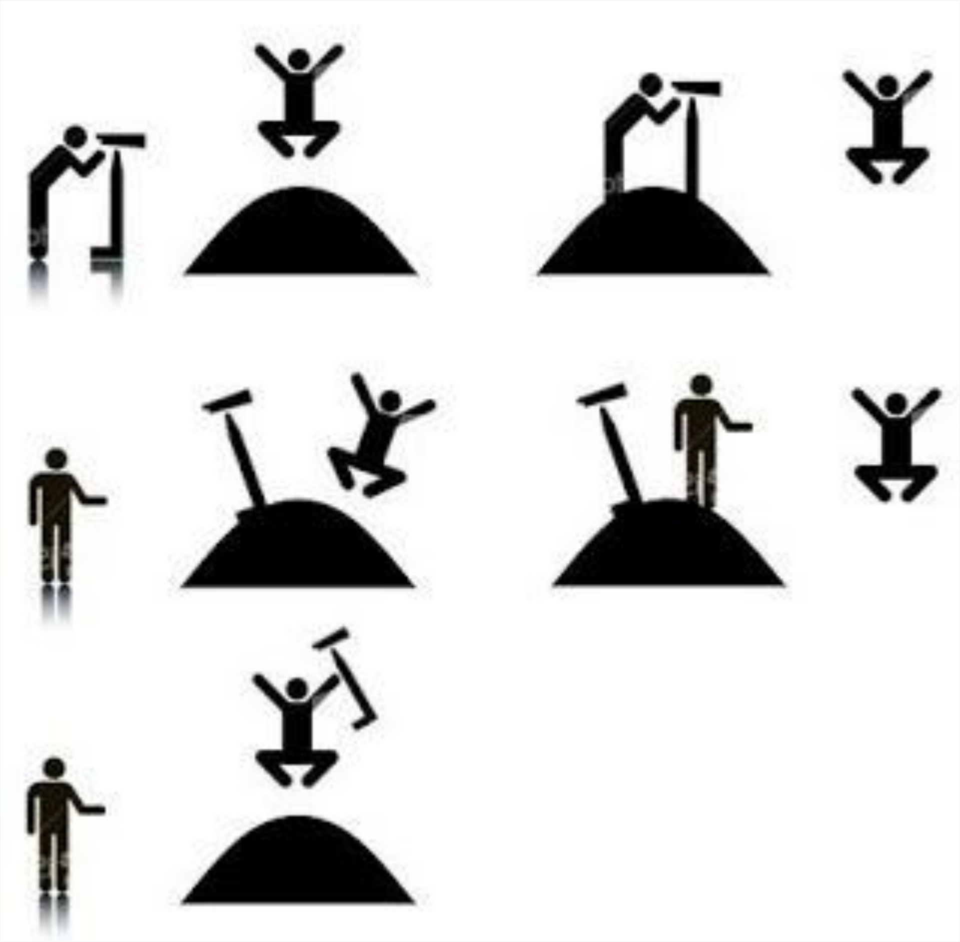
PP -> P N

(NP -> N)



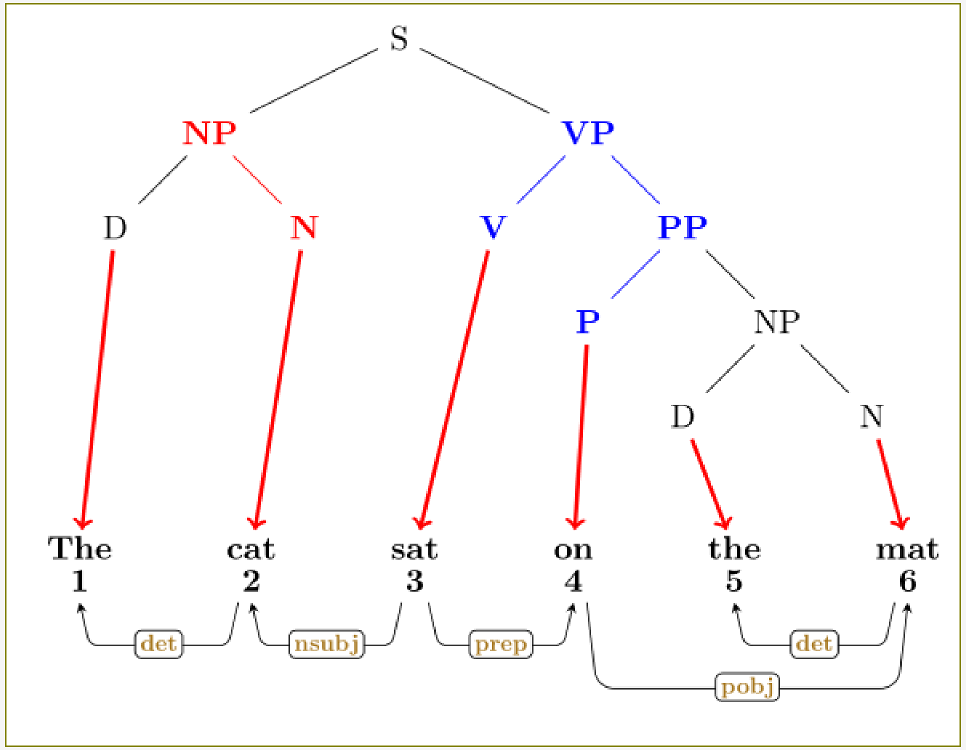
Syntactic ambiguity

- Sentences are often ambiguous
- How many interpretations for the following sentence:
  - "I saw a man in the park with the telescope"



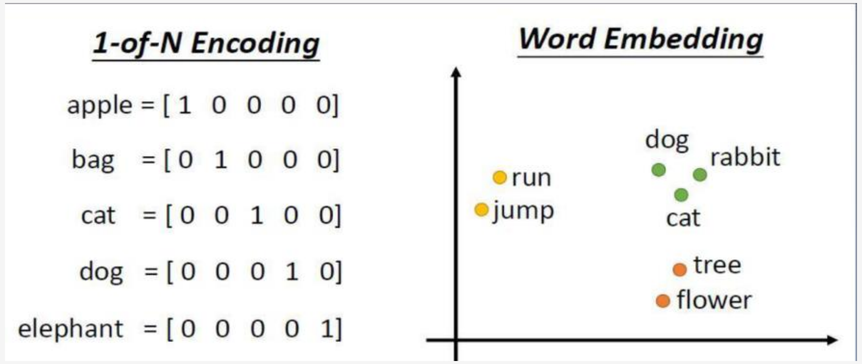
"Linguistic" tasks: Dependency parsing

- The – (det) -> cat
- cat – (subj) -> sat
- on – (prep) -> sat
- mat – (pobj) -> on
- the – (det) mat



"Linguistic" tasks: word embeddings

- What do the words "mean"?
- How can we "measure" the meaning?
- Distributional semantics:
    - the meaning is a function of the context
- Word vectors: one-hot, count based, embeddings



“Linguistic” NLP tasks

- Why do we need linguistic tasks?
  - To help computers make sense of language
    - Pre-processing and feature extraction
  - To help humans make sense of language
    - Linguistic and cognitive science experiments
    - For some problems “linguistic” NLP tasks are end goals

Linguistic tasks and machine learning

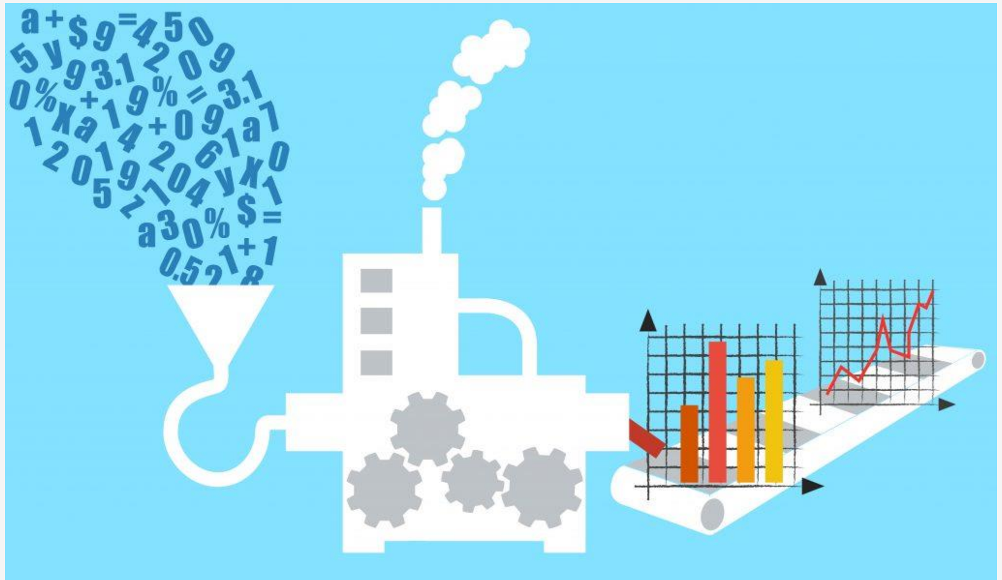
- Linguistic tasks are a goal on their own
- Linguistic tasks are tools in the (classical) NLP toolbox
- The bi-direction interaction between ML and linguistic tasks and data
  - Many linguistic tasks require ML
  - The output of linguistic analysis is used in practical applications
  - Word embeddings and transfer learning

Everyday tasks that use language / NLP

- Marketing
  - Sentiment analysis
  - Recommender systems
  - Content creation

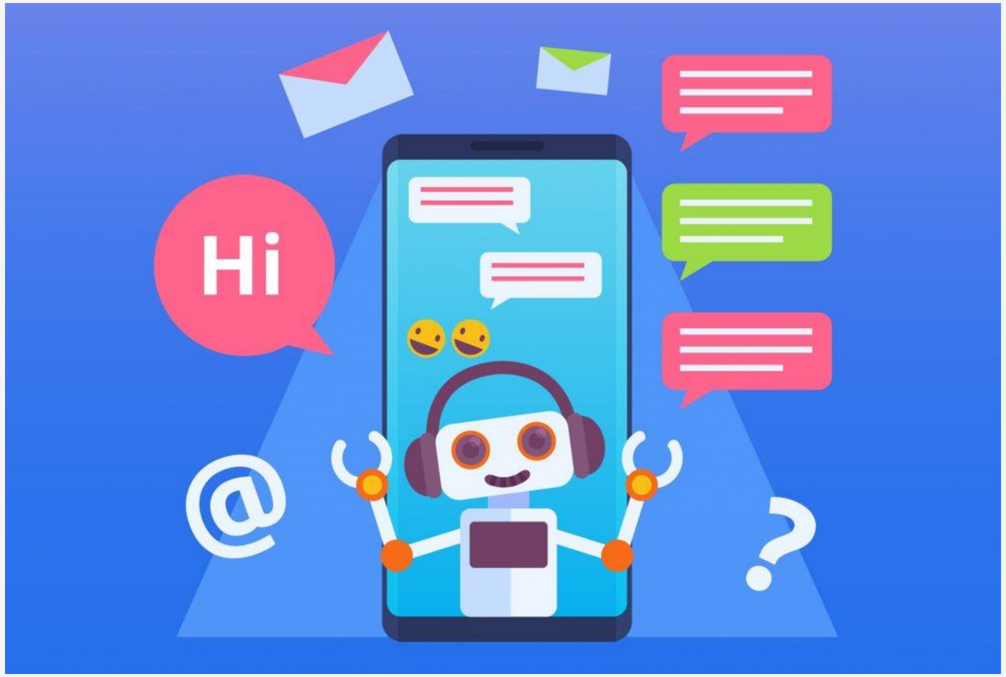
Everyday tasks that use language / NLP (2)

- News articles, social media, and search
  - Information extraction
  - Question answering
  - Inference and Fact checking
  - Content moderation



Everyday tasks that use language / NLP

- User experience and assistance
  - Conversational Agents / Chat bots
  - Machine translation
  - Personal assistants
  - Autocomplete, copilot



Extra-linguistic tasks

- Extra-linguistic tasks are not “internal” to language
  - Part-of-speech tagging vs book recommendation
- Language can be used to solve extra-linguistic tasks (in part)
- Rest of the module: taking a more practical direction
  - How do we solve problems using language
  - How do we define (and evaluate) problems and solutions
  - Feature-based solutions vs end-to-end solutions

Text classification
Feature Engineering

Machine learning and NLP – supervised and unsupervised NLP

- Types of machine learning problems: supervised, unsupervised, reinforcement

- Pop quiz: can you name ML problems of each of the three types?

- NLP problems are predominantly (represented as) supervised
    - Text classification: Sentiment analysis, Textual Inference, Fact checking, Toxic language detection
    - Text generation: Question answering, Chatbots, Machine translation

Text classification

- Observations are independent from each other (e.g. single tweet)
- Observations are preprocessed and fed into (a trained) classifier
- The classifier assigns the correct class-label to the observation
- Classes are discreet and often disjointed:
  - an email is either spam or ham, never both
- Different types of classification: binary, multi-class, multi-label

Extra-linguistic problems and feature engineering

- Historically, we approached extra-linguistic problems via feature engineering

- Feature engineering
  - Converting language data into relevant data that is easy to process for machines
  - A “faulty” translation from “human” to “computer”
  - Loses some (often a lot of) information
  - Requires a lot of human intervention

Feature engineering

- Analyze the problem, the input, and the desired outcome
- Explore existing resources and processing techniques
- Select the most relevant features and feature-extraction methods
- Empirically test what works best

Feature engineering

- Various features can be extracted from texts
- Bag-of-words (+ tf-idf)
- N-grams
- Part-of-speech tags
- Named entities
- Sentiment words
- Stop words
- Length

Feature engineering (example)

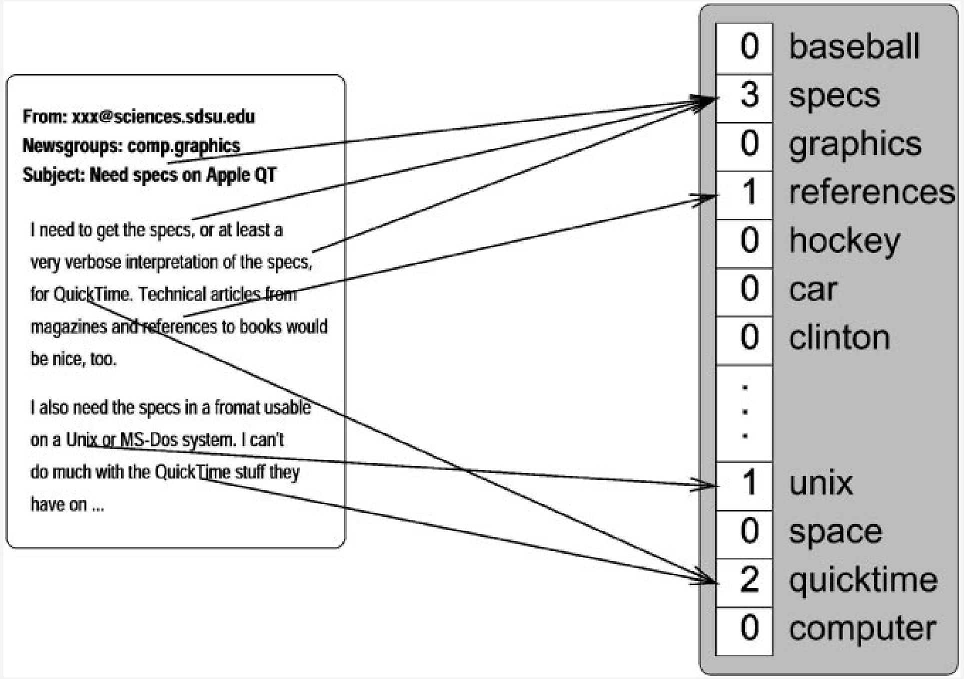
- Consider the task of sentiment analysis
- What features can you extract from the following examples?
- Are these tweets positive or negative?

1."🌟 Absolutely in love with my new headphones from SoundWave! The sound quality is top-notch, crystal clear, and the noise cancellation is a game-changer! 🎧 ✨ Highly recommend to all music lovers out there! #SoundWave #MusicLife 👍🏽 😊 "

2."Really disappointed with my purchase from QuickTech. The laptop crashes constantly and the battery life is a joke. 😡 💻 Worst customer service ever - they just don't care. Totally regret this buy. #QuickTechFail #Frustrated 😞 👎🏻 "

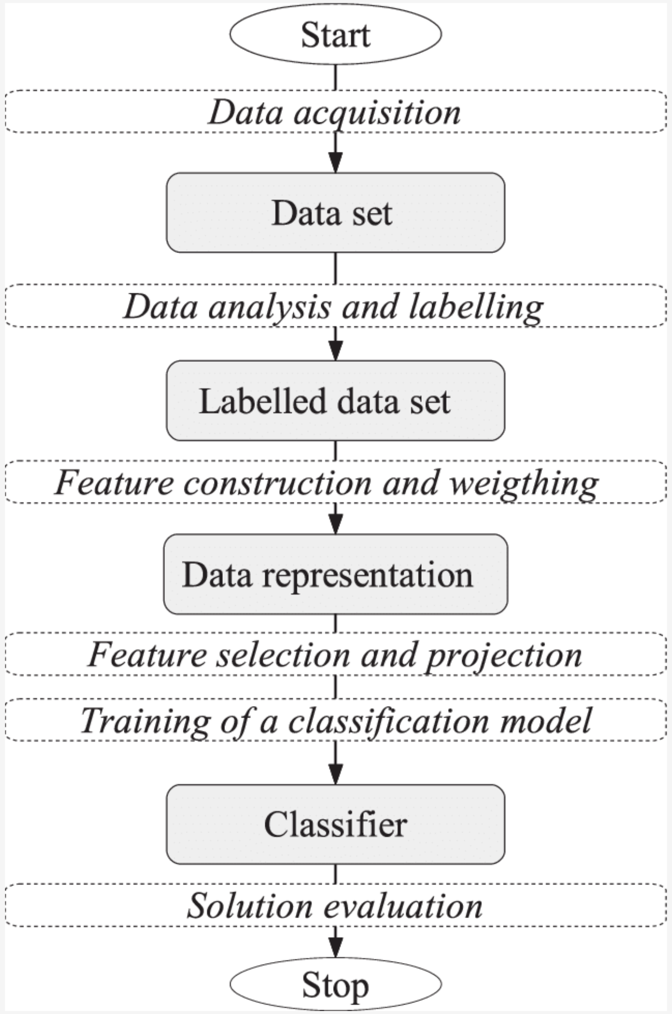
Feature extraction

- Define the set of relevant features
- Train (or program) algorithms to process the text
- Extract features
- Represent the text via the features



Text classification using features

- Step by step process
- Involves active human engagement
    - Feature selection and extraction
- Data is fed into a classifier (Logistic, NB, SVM)
- Iteratively improve feature selection and model (hyper) parameters



The shift towards data-driven approaches

- DSM and embeddings extract (relevant) features from the data
  - Minimal supervision
  - Easier maintenance
- Allow computers to “read” and use language in a more direct way
- The cost is loss of control and transparency

New NLP paradigm – the rise of end-to-end neural models

- Word embeddings mark a major shift in NLP
- What do you think is an "end-to-end" neural model?
- End-to-end neural model represent the complete target system
  - No external preprocessing
  - No explicit pipelines
  - Input-output mapping
  - Training for a specific task (with some transfer learning)
- What would be some limitations of end-to-end models?

Using embeddings
Compositionality

Use of word embeddings

- Embeddings are high dimensional vector representations of words

- All words in the vocabulary can be represented as a high dimensional vector

- What can we use embeddings for?
  - Write five different applications

Querying embeddings for lexical information

- Embeddings are originally a lexical resource
- Performing operations at word level
  - Find words that are semantically similar to a given word
    - Expand existing dictionaries by automatically searching for similar words
  - Explore relational similarity (e.g., UK – London: France – Paris)
  - Compare the meaning of a word to its context
    - Automatic error correction
-

Querying embeddings for lexical information

- Learning from embeddings
  - Learn specific semantic relations (e.g. hypernymy) (Shwartz, et al. 2016)
  - Learn compositionality rules (Baroni and Zampareli, 2010, Socher et al. 2013)
    - Learn representations for phrases and compare with words
  - Clustering (Kovatchev et al. 2016)
    - Topics
    - Part of speech

Evaluating word embeddings

- Embeddings are trained on word prediction (or noise detection)
- Embedding algorithms converge to the best (mathematical) solution
- We throw away the original task!
- How do we evaluate embeddings?
  - Word similarity and word analogy
  - Downstream tasks

From words to text

- Embeddings represent words
- NLP is about processing text
- “The cat sat on a mat” vs “the mat sat on a cat”

Compositionality of meaning

- “The meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them”

- Two key questions:
  - How do we combine individual word meaning?
  - Does the word meaning remain static?

How to combine word meaning

- Assume that the word meaning is a vector or a tensor

- How can you calculate the meaning of a phrase?
  - Addition/aggregation
  - Complex (hierarchical) operations
  - Via a deep neural network

Vector addition

- The simplest form of compositionality
  - “The cat sat on a mat” = “The”+ “cat” + “sat” + “on” + a” + “mat”
- Advantages
  - Easy to calculate
  - Fixed vector length, regardless of text length
- Disadvantages
  - Loses word order
  - Lower impact of individual words (e.g., “not”)

Vector concatenation

- Alternative to vector addition
  - Instead of adding dimensions, we concatenate the vectors
- Can you identify advantages and disadvantages of this approach?

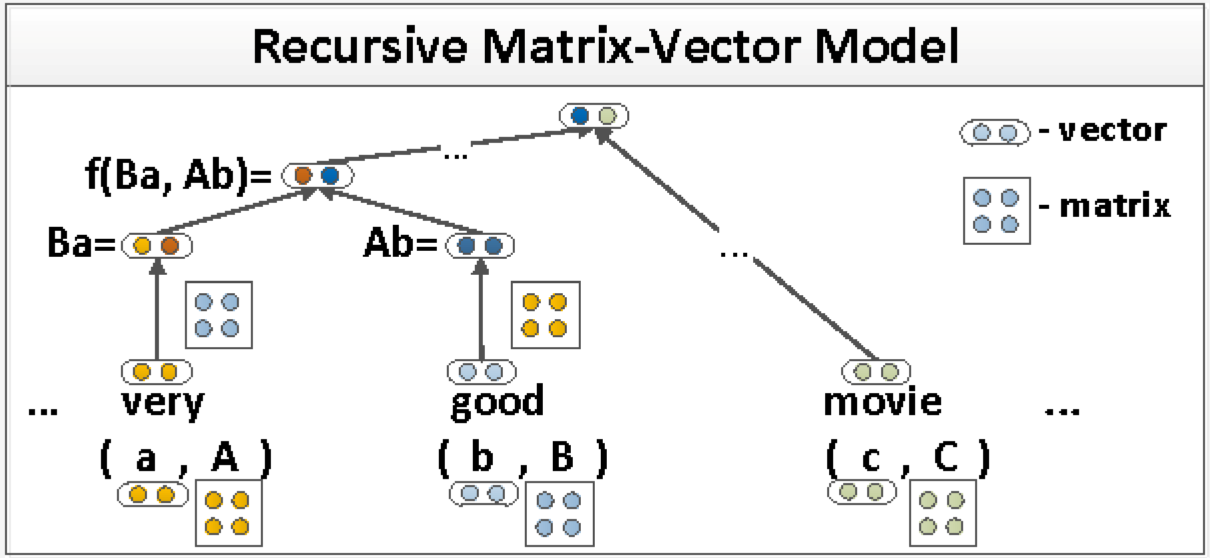
Recursive compositionality

- Baroni and Zampareli (2010), Socher et al. (2012, 2013)

- Meaning is not just a vector, but can be a vector + matrix

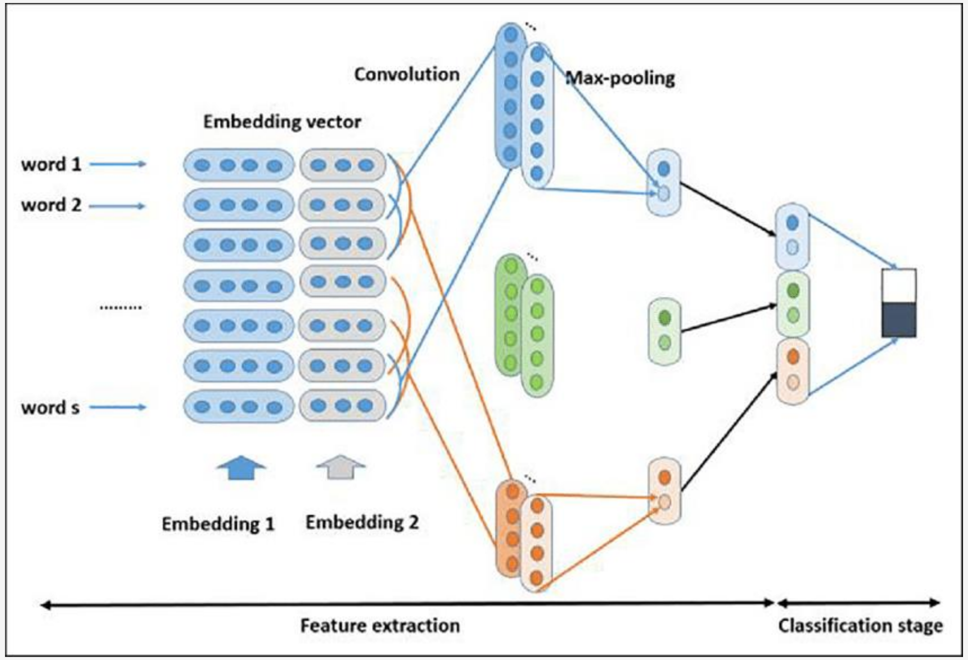
- Compositionality is a recursive vector-matrix operation

- Follow the syntactic structure



Compositionality via deep neural networks

- The current state-of-the-art
- Input the embeddings into a neural network
- Let the network handle the interactions
  - The network architecture determines the compositionality



Task specific embeddings

- General purpose word embeddings
- Polysemy (“blue cat”, “cat myfile.sh”, “CAT scan”)
- Retraining embeddings
  - Domain: news, medical, social media (Major et al. 2018, Soares, 2019)
  - Task: sentiment, NER (Siencnik, 2015)
  - Languages

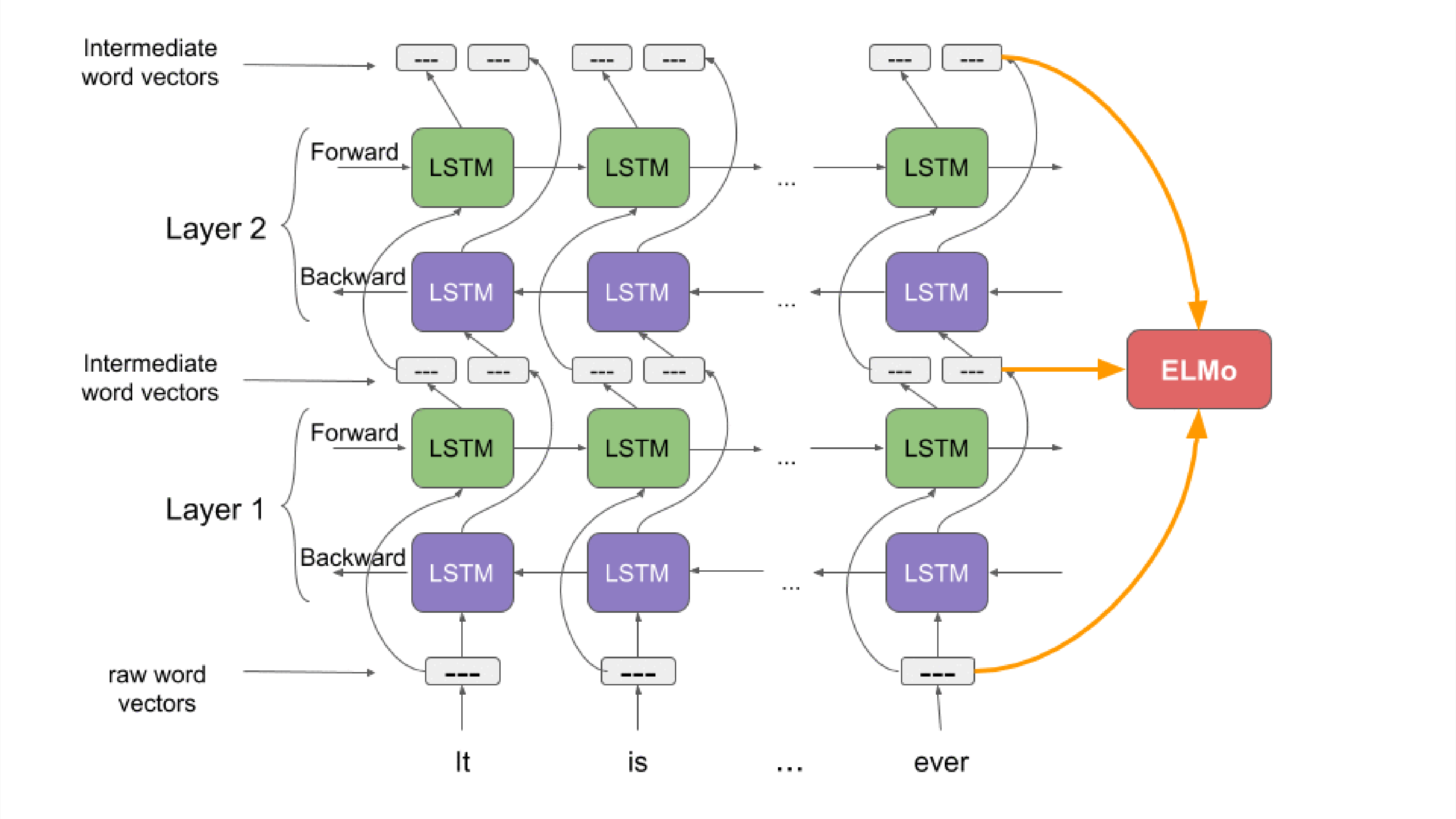
Contextual word embeddings

- The problem of polysemy
  - “The **cat** sat on a mat”
  - “You can **cat** this text file”
  - “I just got the results from my **cat** scanner”
- Are these the same word?
- Should they have the same embedding?
- Task specific embeddings may solve the problem, but can we do better?

ELMO – Deep contextualized word representations

- Key idea: generate a dynamic embedding, based on the context a word appears
  - “The cat sat on a mat” -> W2V -> the vector of “cat” depends only on “cat”
  - “The cat sat on a mat” -> ELMO -> the vector of “cat” depends on all words
- How? Bi-directional (LSTM) language model
- Concatenation of different layers
- Task-specific weights

Bi-directional language model



Source: https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Bi-directional language model

- Forward language model:

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1}).$$

- Backward language model:

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N).$$

- Bi-directional LM:

$$\sum_{k=1}^{N} \big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \big) .$$

- Predict a word given its left and right context; keep input and output weights shared;

Deep representations

- For each token k, an L-layer bi-directional LM obtains 2L + 1 representations

$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} = \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}$$

- Initial (static) representation x

- For each layer – hidden representation of forward and backward

- ELMO learns a task-specific linear combination:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}. \quad (1)$$

- The representations R_k depend only on the context

- The embedding $ELMO_k$ depends on the context and the task

The impact and importance of ELMO

- ELMO (Peters et al. 2017) significantly improves NLP model performance
- Not the first contextual representation
- The first deep contextual representation (prior work only takes last layer)
- The first task-specific representation
- Short lived success due to the appearance of transformers and BERT
- Many of the concepts in ELMO are adopted in BERT

Using embeddings in downstream tasks and student projects

- What kind of vectors to use?
- Use pre-trained or re-train?
- Use as feature vectors or use to learn features?
  - E.g., can you “learn” which vectors are positive?
- Can you combine with other features?
- What would be the classifier?
- Size and scale of vectors?

End to end neural models

Accumulation of errors in a pipeline

- Assume a classifier working at 95%, but…
  - 95% accuracy tokenizer
  - 95% accuracy POS tagger
  - 95% accuracy dependency parser
  - 95% relevant feature mapping
  - 95% accurate classifier
    - What would be the total performance?
    - 77.3%

Why not go end to end?

- Do we need full pipelines?
- Embeddings make it possible to “feed” text directly into models
- Is it possible to go fully end-to-end and eliminate
    - Accumulation of errors
    - Human labor and supervision
    - (In)compatibility issues between elements?

High level idea

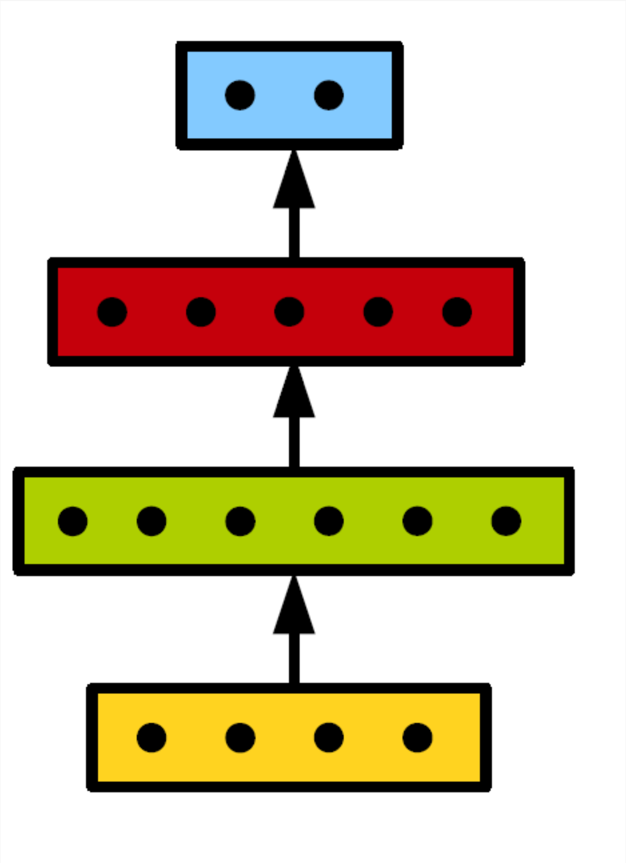
- Embeddings represent individual words
- Neural architecture combines embeddings (“words”) together
- Deeper hidden layers represent interactions between words (phrases, clauses, sentences)
- Backpropagation updates the way we combine words, but can also update embeddings

Neural architectures

- Feed forward neural networks
- Recurrent neural networks
    - LSTM, BiLSTM, GRU
- Convolutional neural networks

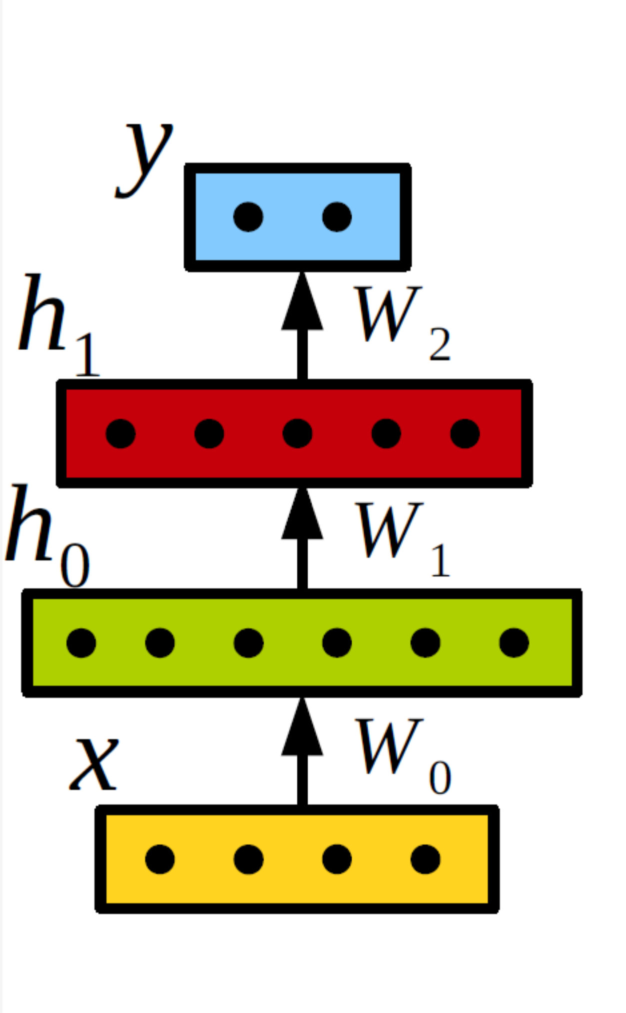
Feed-forward networks
Multi-layer perceptron

Multi-layer perceptron



- An input layer (embedding)
- One or more hidden layers, each computed from the previous layer
- An output layer
  - Based on the last hidden layer
  - Class probabilities (spam / ham)

Multi-layer perceptron



- $y = \text{softmax}(h_1 \cdot W_2 + b_2)$
- $h_1 = f(h_0 \cdot W_1 + b_1)$
- $h_0 = f(x \cdot W_0 + b_0)$
- Non-linear functions f:
  - Sigmoid: $\sigma(x) = \frac{1}{\exp(-x)}$
  - Hyperbolic: $\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$
  - ReLU: $\text{rect}(x) = \max(0, x)$

What is the input X?

- “The cat sat on a mat”
- We can convert all words into vectors
  - w2v(“the”), w2v(“cat”), w2v(“sat”), w2v(“on”), w2v(“a”), w2v(“mat”)
  - We obtain 6 300 dimensional vectors
- How do we add the input in the MLP?
  - Single fixed vector
  - A matrix of vectors
  - What are some advantages and disadvantages of each approach?

Training MLP

- Using Stochastic Gradient Descent
- Start with pretrained embeddings and random weights and biases
- Each epoch
    - Shuffle training data and select a training batch (set of k examples)
    - Compute the “forward” and the loss function
    - Compute the gradient of the loss function (backward)
    - Update the parameters (learning rate η): W = W - η $\frac{1}{K}\sum_{i=0}^{K-1} \nabla J_i(W)$
- Repeat until convergence

Training MLP

- Using Stochastic Gradient Descent
- Start with pretrained embeddings and random weights and biases
- Each epoch
  - Shuffle training data and select a training batch (set of k examples)
  - Compute the “forward” and the loss function
  - Compute the gradient of the loss function (backward)
  - Update the parameters (learning rate η): W = W - η $\frac{1}{K}\sum_{i=0}^{K-1} \nabla J_i(W)$
- Repeat until convergence

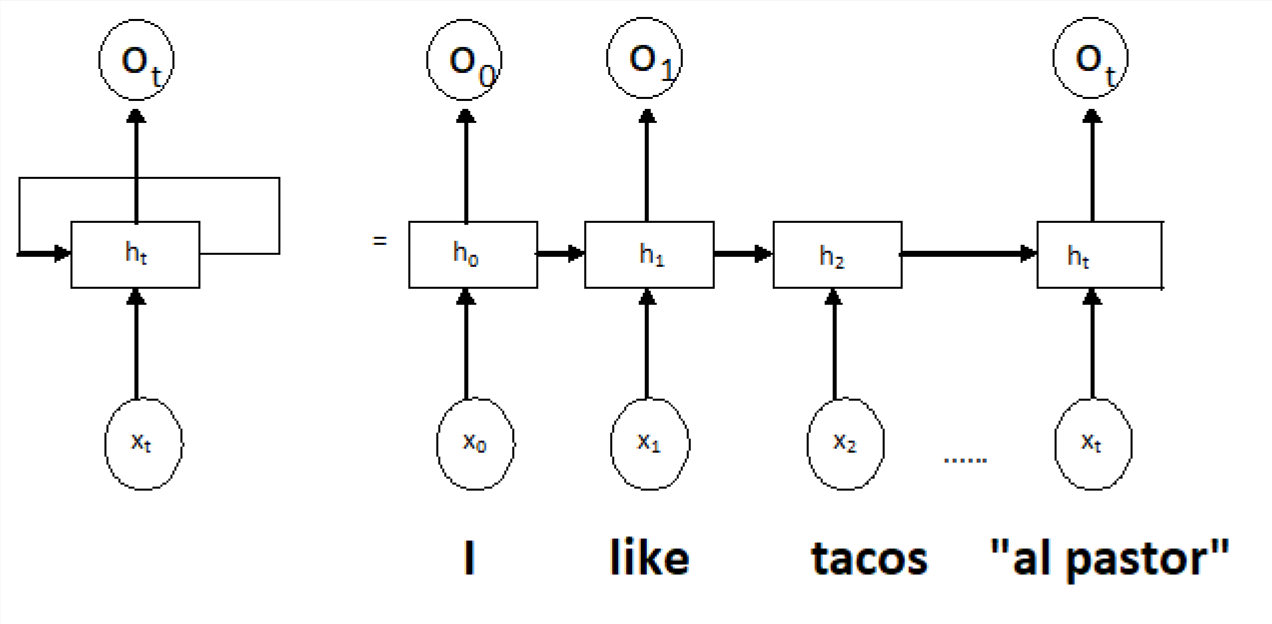
Training MLP

- SGD requires calculating the gradient of the loss function J, ∇J
  - For each example x_i
  - For each parameter w $\in W$
- Do we also update embeddings?
- How do we do that if we want to?

RNN, LSTM, and BiLSTM
Neural Language Models

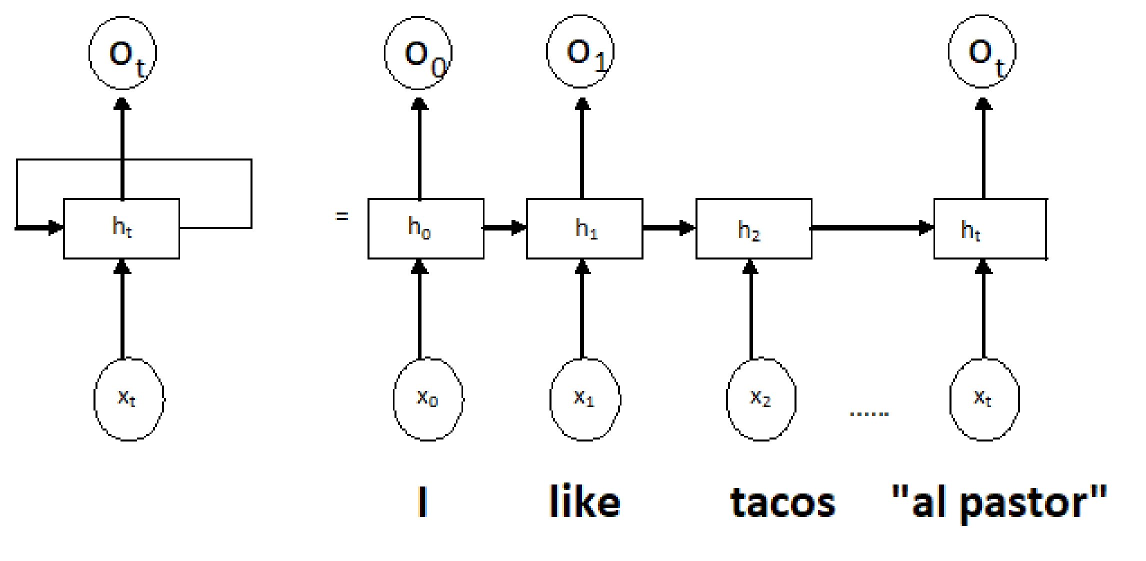
Recurrent neural networks

- How to represent text of varying length?
- Copy the network for each word
- At each timestep t, input is X_t and $h_{(t-1)}$
- I + like + tacos + “al pastor”
- Left to right, combining words one at a time
- Sequence classification
- Sequence to sequence



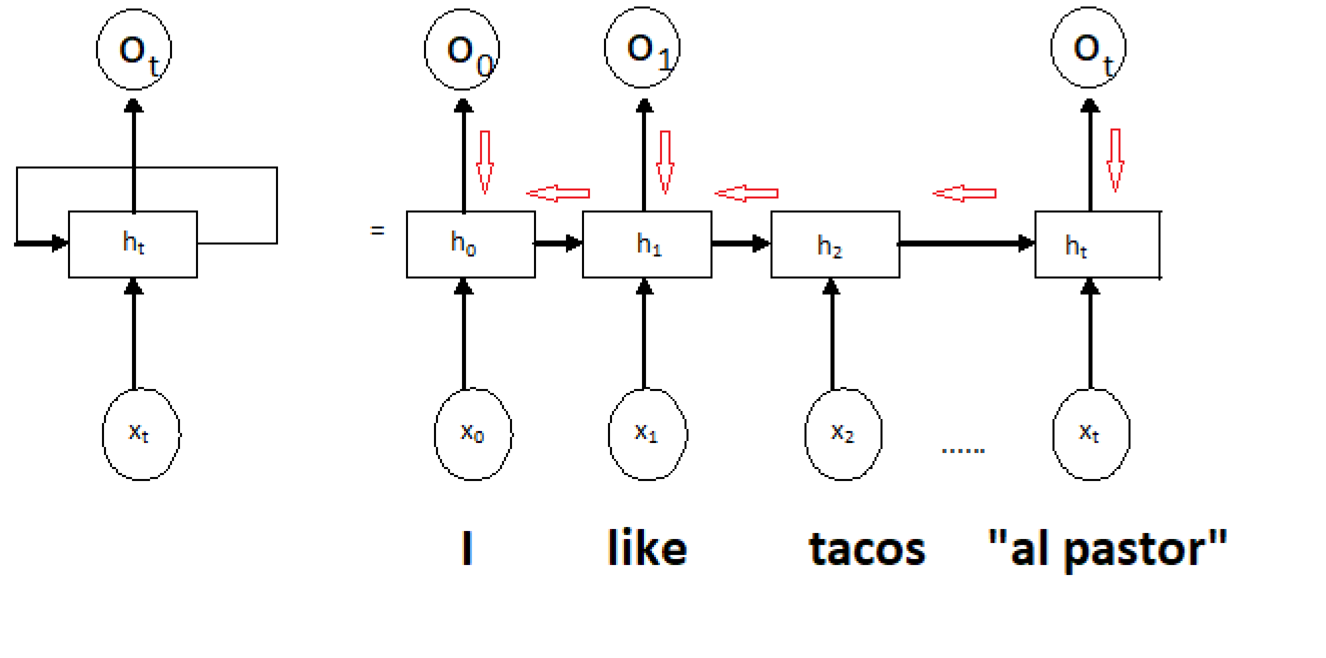
RNNs (formally)

- At each timestep
  - Input x_t, and previous hidden state $h_{(t-1)}$
  - Three sets of weights:
    - input (W_x), hidden (W_h) , and output (W_o)
  - $h_t = f_h(W_x x_t + W_h h_{(t-1)})$
  - $O_t = f_o(W_o h_t)$
  - $J_t = f(O_t, y_t)$
- Pop quiz: are W_x, W_h, and W_o at time t_0 different than at time t_1?



Backpropagation through time

- Gradients flow backwards
- Propagate the gradient through each step, starting from the last
- Update the three weight matrices
- What would be a potential problem?
  - Exploding/vanishing gradients
  - Saturating non-linearity



Language modeling

- Recall: a language model assigns probability to a sequence of words:

  - Traditional Markov model: $p(w_1, \ldots, wm) = \prod_{t=1}^{m} p(w_t | w_1, \ldots, w_{(t-1)}) \approx \prod_{t=1}^{m} p(w_t | w_{(t-n), \ldots,} w_{(t-1)})$

- Estimate the probability of a word given a history of a predefined length

- Estimate the probability of a sequence by multiplying the probabilities of all words

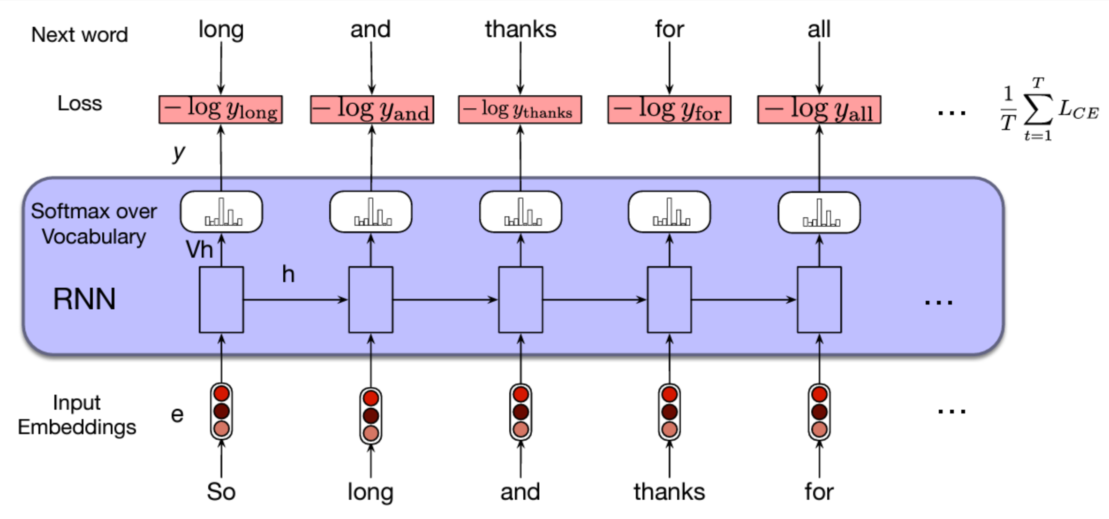
- Pop-quiz: is word2vec a traditional language model?

Neural language modeling

- If we use an RNN based model, we can drop Markov assumption
- Hidden state at time t keeps information about everything seen before
- At each step, we can add a classifier on top of the hidden state and predict the next word
- Add special symbols to mark the start and end of the sentence (<s> </s>)

Neural language modeling with RNNs

- More formally:
  - $h_t = \tanh(W_x x_t + W_h h_{(t-1)})$
  - $\hat{y}_t = \mathrm{softmax}(W_y h_t)$
  - $J_t = -\log \hat{y}_{t,\ \mathrm{correct}}$ (- log prob the word at t+1)



- $J_{sent} = \frac{1}{T}\sum_{t=1}^{T} -log\ \hat{y}_{t,\ \mathrm{correct}}$

Figure 9.6 from SLP, chapter 9

Tying the weights

- Pop quiz: What are the dimensions of W_y?
- What are the dimensions of the embedding matrix E?
- We can use a quick "trick" and set the dimension of h to be same as the size of the vectors
- Then we can use E^T instead of W_y
  - Easier to train and performing better

The output layer in RNNs

- The output layer “knows” the history until time step t
- If the output layer predicts words, this the RNN becomes a language model (!)
- However, the output layer does not have to predict words
    - RNN can be used for classification, sequence tagging, generation

RNN for sequence tagging

- RNN can be trained to predict POS tags
- At each step, use the hidden state to predict POS
- Softmax over tags instead of words

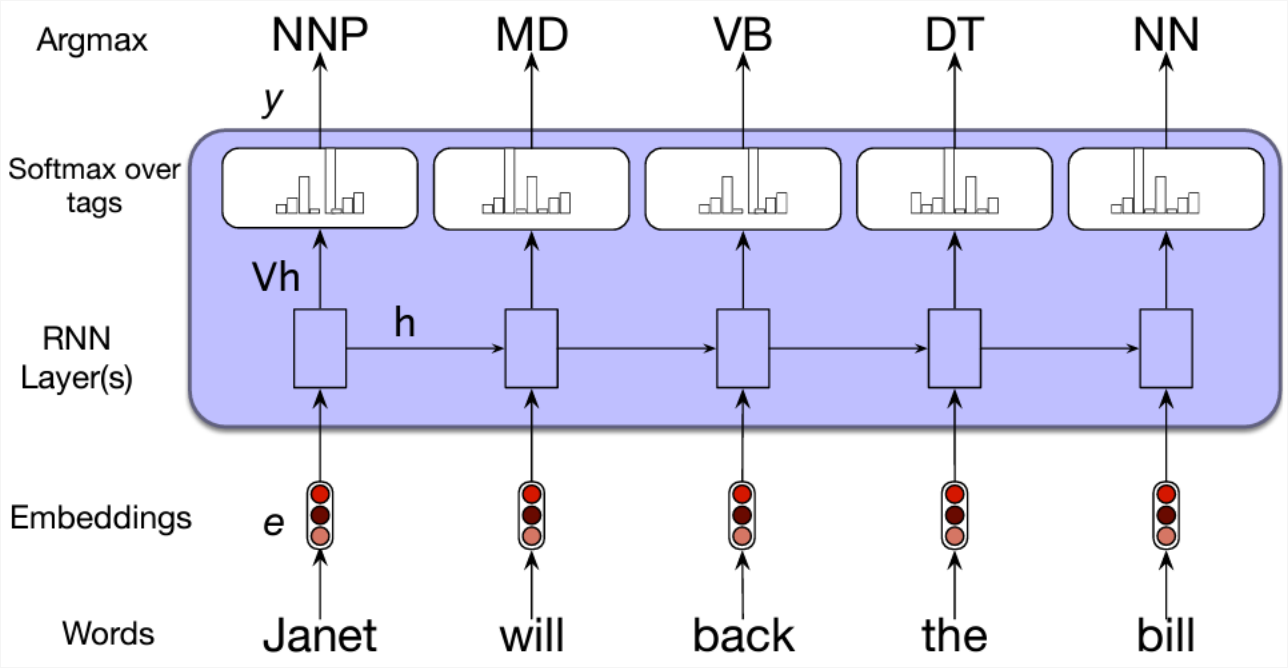


Figure 9.7 from SLP, chapter 9

RNN for text classification

- RNN can be used directly for classification
- Only use the last hidden layer
- Either predict directly, or add a FFN
- Predict intermediate sentiment?

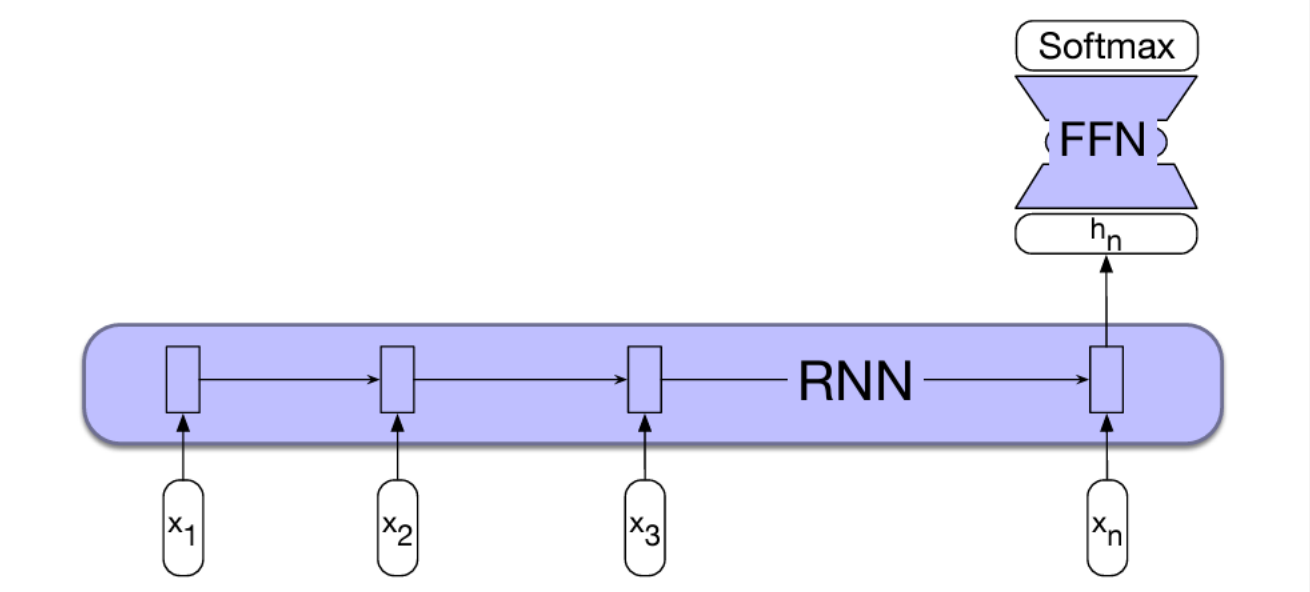


Figure 9.8 from SLP, chapter 9

Using RNN to generate text

- Generative AI

- How do you generate text using a Markov model?

- Generating using RNN
    - Sampling
    - Updating expectations
    - Teacher forcing

Stacked and bidirectional RNNs

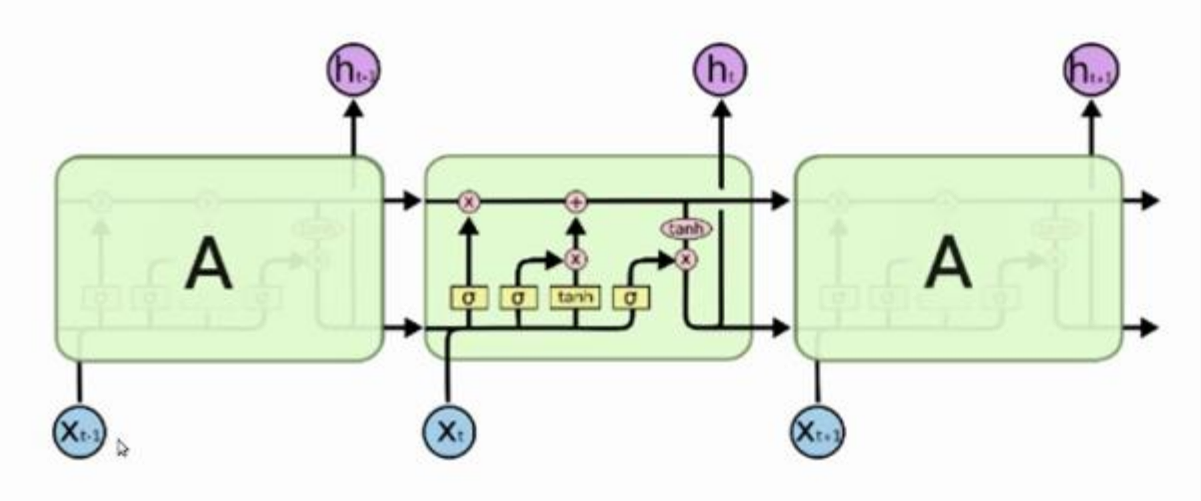
- The basic RNN is a powerful tool
- We can go deeper
- Stacking RNNs
- Bidirectional RNNs
  - Pop quiz: in a BiRNN for classification, which hidden states do we use from forward and backwards?

LSTM

- Popular variation of RNN that address native limitations
- Same recurrent concept (copy the network)
- Three different “gates”:
  - Forget gate
  - Input gate
  - Output gate
- Gates manipulate the flow of information through time
  - Addressing conflicting objectives

LSTM. Some Notations

- x_t – input at time t
- h_t – hidden state at t after output gate
  - “short term memory”
- C_t – state at t before output gate
  - “long term memory”
- O_t – output at t
- W and U – weights with respect to x and h



Understanding the gates

- All gates have the same format:
  - A feedforward layer followed by a sigmoid
- The gates are filtering out information at a certain part of the network
- Each gate has two important aspects:
  - How to calculate the filter
  - What is the input that the filter is applied to

LSTM Gates: The forget gate

- Goal: determine how much to keep from the previous information (the long-term memory)
- Formula: $f_t = \sigma(U_f h_{(t-1)} + W_f x_t)$
  - Input at the current state (x_t), weighted by W_f
  - Hidden state at state t-1, weighted by U_f
  - Sigmoid activation (0 – forget everything, 1 – keep everything)
- "How much information to keep from the Long-term memory, given the current input and short-term memory"
- f_t is applied to $C_{(t-1)}$

LSTM Gates: The new information

- Goal: determine how surprising the new information is
- Formula: $g_t = \tanh(U_g h_{(t-1)} + W_g x_t)$
  - This is our “normal” recurrence in RNN
  - Tanh activation: what values does it take?
    - Tanh values are in [-1, 1]
  - If the value of g_t is negative, the new information is subtracted from the memory, rather than added
- Note: this is not a gate, this is just your typical RNN hidden state

LSTM Gates: The input gate

- Goal: determine how much to use from the new information
- Formula: $i_t = \sigma(U_i h_{(t-1)} + W_i x_t)$
  - Similar to the forget gate, but with its own set of weights
- i_t is applied to the value of new information N_t
  - The input at state t is filtered twice: through N_t and i_t
- If we add the information that passes through the input and forget gates, we get the current long-term memory
  - $C_t = f_t * C_{(t-1)} + i_t * g_t$

LSTM Gates: The output gate

- Goal: determine what part of the long-term memory is necessary right now
- Consider machine translation: what is needed to translate current word vs what is needed to translate everything
- Formula: $o_t = \sigma(U_o h_{(t-1)} + W_o x_t)$
  - Very similar to forget and input gates, with its own set of weights
- o_t is applied to the value of C_t after passing it through another tanh
  - $h_t = o_t * \tanh(C_t)$
- h_t is the “traditional” hidden state – it can be used to make predictions at time step t

LSTM Results and improvements

- LSTM quickly became dominant paradigm in NLP
- Used in a variety of tasks:
  - Sentiment analysis (Wang et al. 2016)
  - Sequence labeling (Miwa et al. 2016)
  - Question Answering (Wang et al. 2015)
  - AMR parsing (Foland et al. 2017)
  - NLI (Chen et al. 2017)

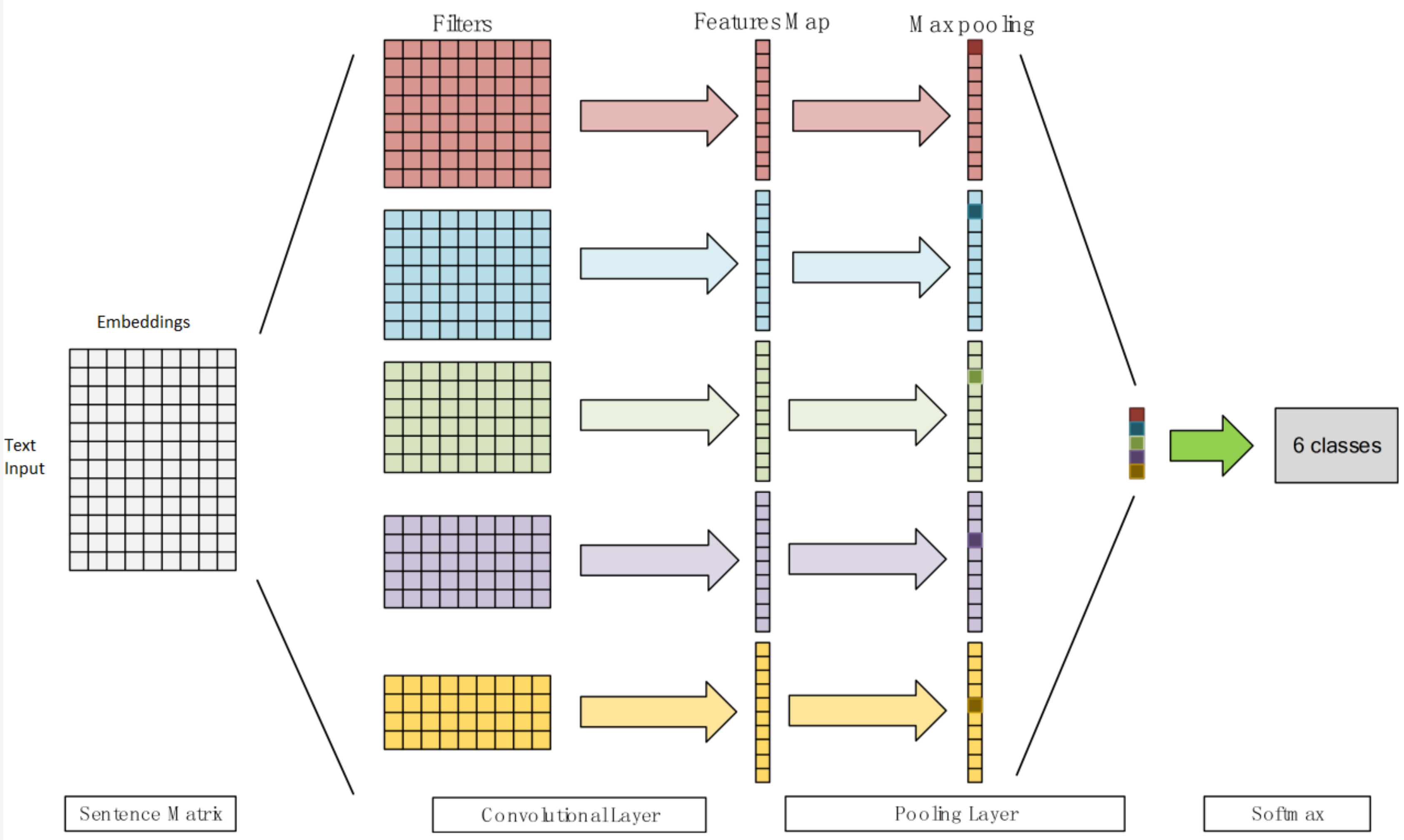
BiLSTM, stacked LSTMs, GRU

- Similar to RNN
- Deep LSTM
- BiLSTM – the de-facto norm of using LSTMs
- GRU – a different variation of gated RNN, slightly simpler than LSTM

Convolutional neural networks (CNN)

- Another popular architecture for NLP
  - Deals with text of various size
- Inspired by computer vision success
- Applying filters of different size to the input (2,3,4) + pooling
  - Analogous to n-grams

Convolutional neural networks (CNN)



Multimodality and multilinguality

- Embeddings and multilinguality
  - Words of any language can be mapped to vectors
  - Words of different languages can be mapped to the same space
  - Mapping and similarity across languuages
- Embeddings and multimodality
  - Images, sound, and other modalities can also be mapped to vectors (e.g., using CNNs)
  - Shared multimodal spaces (vision + language)

The first “should I worry about my job”

- Rapid change in technologies can be stressful

- Three “major” milestones in NLP in the past 10 years
  - Word2Vec and end-to-end models (BiLSTM, CNN) – 2013
  - BERT (and the transformer family) – 2018
  - Large generative language models (GPT3, ChatGPT, Bard) – 2022 - 2023

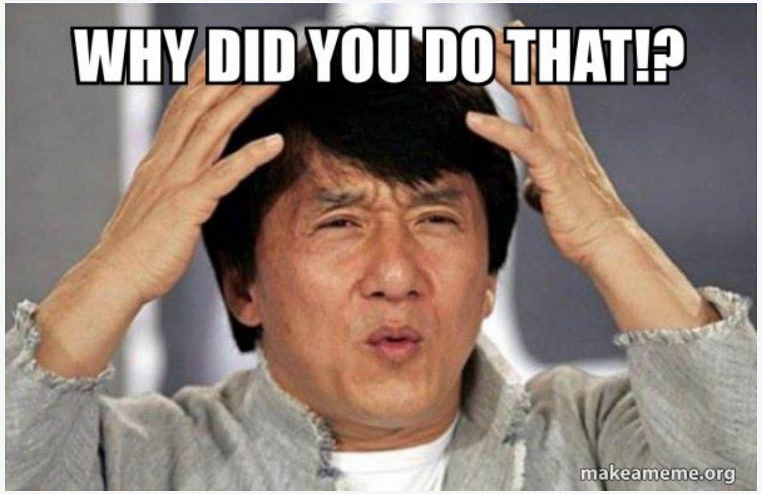
- There are still many unsolved problems

Some concerns

Explainability and Interpretability

- Interpreting feature-based models
  - Feature values (“v1agra”) + weights = prediction (“spam”)

- Interpreting end-to-end neural networks
  - Feature values (300d dense vector)
  - weights (input, forget, output gates)
  - different types of nonlinearity



Explainability and Interpretability

- Why did you do that?
- How did you do that? What makes you say that?
- How can I verify your results?

Bias, Guarantees, and Robustness

- Does the algorithm discriminate?
- Does the algorithm contain bias with respect to race, gender, religion, sexual orientation?
- Does the algorithm guarantee consistent and robust performance?
- Is the algorithm secure from adversarial attacks?

New fields of study in NLP in the are of deep learning

- Explainability of neural networks
- Algorithmic fairness
- Evaluation, unit testing, and adversarial attacks for NLP
- Data centric AI

Conclusions

Embeddings in NLP

- Embeddings changed the way we do NLP

- Valuable stand-alone resource
  - Can be used to query lexical information
  - Can be used as automatically extracted features
  - Can be used for a simple text representation
  - Static and dynamic embeddings, polisemy

- Enable end-to-end neural models

End-to-end neural models

- Minimizing human interaction and supervision
  - Remove the need of feature engineering
  - Only require labeled data for classification tasks
- Improving efficiency and removing accumulation of errors in pipeline
- Enabling full training without depending on external resources

End-to-end neural models

- Introducing new challenges and problems
  - Increased computational complexity
  - Need for more data
  - Error accumulation becomes internal
  - Difficult to interpret and debug
  - Potentially containing biases
  - Ultimately re-invent many of the pipeline parts during training