IAAA / PSTALN

Introduction to natural language processing

Benoit Favre <benoit.favre@univ-amu.fr>

Aix-Marseille Université, LIS/CNRS

last generated on January 3, 2021

Outline

PSTALN: class outline

- 2020-11-23
  - *Class:* intro on tasks and problems
  - *Practicals:* intro to pytorch
- 2020-11-30
  - CNN, RNN
  - Text classification
- 2020-12-02
  - Sequence tagging, CRF
  - Noun phrase detection
- 2020-12-04
  - LM and translation
  - Phonetization
- 2020-12-07
  - Non-contextual embeddings
  - LDA & GloVe training
- 2021-01-04
  - Contextual embeddings
  - BERT fine-tuning
- 2021-01-06
  - Embedding analysis
  - Probing BERT embeddings
- 2021-01-15
  - Exam

What is Natural Language Processing?

What is Natural Language Processing (NLP)?

- Allow computer to communicate with humans using everyday language
- Teach computers to reproduce human behavior regarding language manipulation
- Linked to the study of human language through computers (Computational Linguistics)

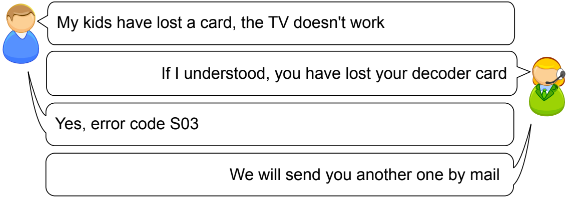
Why is it difficult?

- People do not follow rules strictly when they talk or write: “r u ready?"
- Language is ambiguous: “time flies like an arrow"
- Input can be noisy: speech recognition in the subway

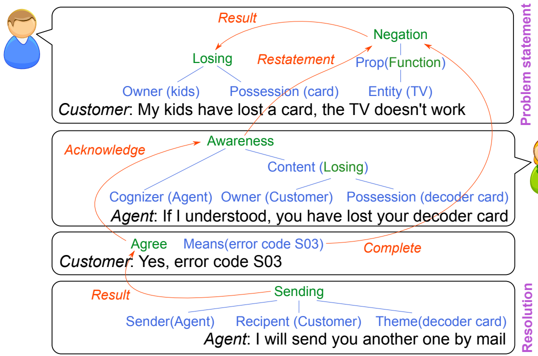
Fundamental question

- What is the model of language?

Example: generate annotations



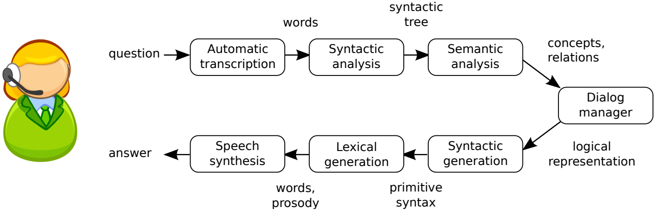
Example: generate annotations



NLP is everywhere

- Spell checker / grammar correction
- Information retrieval / search
- Machine translation
- Information extraction
- Question answering
- Automatic summarization
- Call routing
- Sentiment analysis
- Spam filtering
- Writing recognition
- Voice dictation
- Speech synthesis
- Dialog systems

Modular approach



- Divide and conquer
  - Better understand domain
  - Help systems generalize when there is little data (vs end-to-end)
  - Generate explanations
- However, also a limiting factor
  - Error propagation
  - Fails to model task specificity

Processing levels

- A linguistic description of language

"John loves Mary"

1. **Lexical** : segment character stream in words, identify linguistic units
   John/firstname-male *loves*/verb-love *Mary*/firstname-female
2. **Syntax** : identify grammatical structures
   (S (NP (NNP John)) (VP (VBZ loves) (NP (NNP Mary))) (. .))
3. **Semantic** : represent meaning
   love(person(*John*), person(*Mary*))
4. **Pragmatic** : what is the function of that sentence in context?
   Is it reciprocal ?
   Since when ?
   What does it entail ? know(*John*, *Mary*)

Language ambiguity

- Phonetic
  - I don't know! – I don't - no!
- Graphical
- Phonetic and graphical
  - I live by the bank (river bank or financial institution)
- Etymology
  - I met an Indian (from India or native American)
  - I love American wine (from USA or from the Americas)
- Syntactic
  - He looks at the man with a telescope
  - He gave her cat food
- Referential
  - She is gone. Who?
- Notational conventions
  - Birth date: 08/01/05

(wikipedia)

Notion of corpus

- Language in the wild
  - Email
  - Forums
  - Chats
  - Speech recordings
  - Video
- **Manual Annotation** of all elements we want to predict
  - Text $\rightarrow$ topic
  - Sentence $\rightarrow$ parse tree
  - Review $\rightarrow$ sentiment

General approach

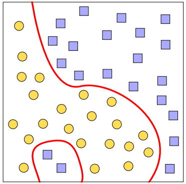
- How to approach natural language processing?
  - Introspective approach
    - *Study what you know about language*
    - Originated from linguistics
    - Formal models, grammar engineering
  - Corpus-based approach
    - *Study how people produce language in the wild*
    - The de facto modern approach
- An empirical approach
  1. Task
  2. Corpus
  3. Guide
  4. Annotation
  5. System
  6. Evaluation

NLP Systems

- Input
  - Raw text, audio...
  - Sentences, contextualized words
  - Output of another system
- Output
  - n classes (ex: topics)
  - Structure (ex: syntactic parse)
  - Novel text (ex: translation, summary)
  - Commands for a system (ex: chatbots)
- Process
  - $\text{output} = f(\text{input})$
  - Deterministic vs random (evaluations need to be repeatable)
  - Parametrisable: $\text{output} = f(\text{input}, \text{parameters})$

NLP as machine learning

- How to design a system from examples of annotated data?
  - For some problems, it's difficult to engineer an algorithm
  - Rely on statistical regularities for performing the task
- Machine learning (ML)
  - find parameters of a function to minimize *errors* on a training corpus
  - loss/cost function minimization
- NLP as ML tasks
  - Labeling (i.e. word senses)
  - Segmentation (i.e. words in Chinese)
  - Regression (i.e. sentiment intensity)
  - Linking (i.e. dependency parsing)
  - Generation (i.e. machine translation)



Basic NLP tasks

- Meta
  - Author/speaker traits recognition
  - Genre classification
- Syntax
  - Word / sentence segmentation
  - Morphological analysis
  - Part-of-speech tagging
  - Syntactic chunking
  - Syntactic parsing
- Semantic
  - Word sense disambiguation
  - Semantic role labeling
  - Logical form creation
- Pragmatic
  - Coreference resolution
  - Discourse parsing

Syntax: Word segmentation

Character sequence $\rightarrow$ word sequence (tokenization)

- Split according to delimiters [:,.!?']
- What about compounds? Multiword expressions?
- URLs (http://www.google.com), variable names (theMaximumInTheTable)
- In Chinese, no spaces between words:
  - $\rightarrow$ (the boy) (likes) (ice cream)

Syntax: Morphological analysis

Split words in relevant factors

- Gender and number
  - flower, flower+s, floppy, flopp+ies
- Verb tense
  - parse, pars+ing, pars+ed
- Prefixes, roots and suffixes
  - geo+caching
  - re+do, un+do, over+do
  - pre+fix, suf+fix
  - geo+local+ization
- Agglutinative languages
  - pronouns are glued to the verb (Arabic, spanish...)
- Rich morphology
  - Turkish, Finish
- $\rightarrow$ Lemmatization task: find canonical word form

Syntax: Part-of-speech tagging

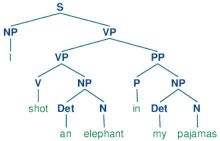
- Syntactic categories

Noun	Adverb	Discourse marker
Proper name	Determiner	Foreign words
Verb	Preposition	Punctuation
Adjective	Conjunctions	Pronouns

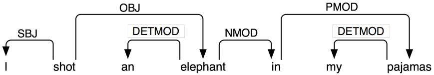
- Each word can have multiple categories
- Example : *time flies like an arrow*
  - flies: verb or noun?
  - like: preposition or verb?

Syntax: Syntactic analysis

- Constituency parsing



- Dependency parsing



Semantics: Word sense disambiguation (WSD)

What is the sense of each word in its context?

- **red**: color? wine? communist?
- **fly**: what birds do? insect?
- **bank**: river? financial institution?
- **book**: made of paper? make a reservation?

Word meaning highly depends on domain

- **apple**: fruit? company?
- **to pitch**: a ball? a product? a note?

Semantics: Semantic parsing

Syntax is ambiguous

- The man **opens** the door
- The door **opens**
- The key **opens** the door

Semantic roles

- Who performed the action? **the agent**
- Who receives the action? **the patient**
- Who helps making the action? **the instrument**
- When, where, why?

John	sold	his car	to his brother	this morning
agent	predicate	instrument	patient	time

Pragmatics: Reference resolution

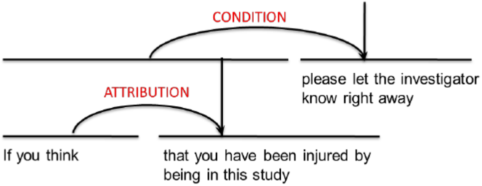
- Link all references to the same entity
  - “Alexander Graham Bell (March 3, 1847 August 2, 1922)[4] was a Scottish-born[N 3] scientist, inventor, engineer, and innovator who is credited with patenting the first practical telephone." (Wikipedia)

Ambiguity

- Pronouns (it, she, he, we, you, who, whose, both...)
- Noun phrases (the young man, the former president, the company...)
- Proper names ("Victoria": South-African city, Canadian region, Queen, model...)

Pragmatics: Discourse analysis

Relationship between sentences of a text, argument structure.



“Fully Automated Generation of Question-Answer Pairs for Scripted Virtual Instruction", Kuyten et al, 2012

Relation type (Rhetorical Structure Theory)

- Background
- Elaboration
- Preparation
- Contrast
- Objective
- Cause
- Circumstances
- Interpretation
- Justification
- Reformulation

Pragmatics: Create a logical form

- Predicate representation
  - Can be used to infer new
- *John loves Mary but it is not reciprocal.*

$$\exists x, y, name(x, \text{“}John\text{”}) \wedge name(y, \text{“}Mary\text{”}) \wedge loves(x, y) \wedge not(loves(y, x))$$

- *John sold his car this morning to his brother.*

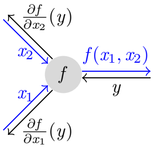
$$\begin{aligned} \exists x, y, z, name(x, \text{“}John\text{”}) \wedge brother(x, y) \wedge car(z) \\ \wedge owns(x, z) \wedge sell(x, y, z) \wedge time(\text{“}morning\text{”}) \end{aligned}$$

History of natural language processing

- 1950: Theory (Turing test, Chomsky grammars)
  - Automatic translation during the cold war
- 1960: Toy systems
  - SHRDLU "place the red box next to the blue circle", ELIZA "the therapist"
- 1970:
  - Prolog (logic-base language for NLP), Dictionaries of semantic frames
- 1980: Dictation, Development of grammars
- 1990
  - Transition "introspection" $\rightarrow$ "corpus"
  - Evaluation campaigns
  - Neural networks are "forgotten"
- 2000
  - Machine learning
  - Applications: speech recognition, machine translation
- 2010...
  - Deep learning, representation learning

Deep learning

- Non-linear compositions of differentiable functions over tensors
  - Also called neural networks for historical reasons
- Minimize loss by gradient descent
  - Chain rule: derivative of composition as product of local derivatives
  - Iteratively improve parameters by taking small steps towards gradient direction
- $\rightarrow$ Build complex functions like lego
  - Much more expressive than before

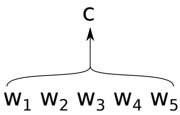


- The work-horse of current NLP research and applications
  - Multitask learning, zero-shot learning, adversarial learning, reinforcement learning, representation learning...

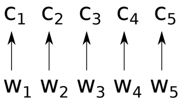
Outline

NLP text and word classification tasks

- Text classification: one prediction per text
  - Predict language of text
  - Predict sentiment of tweet / review
  - Email routing

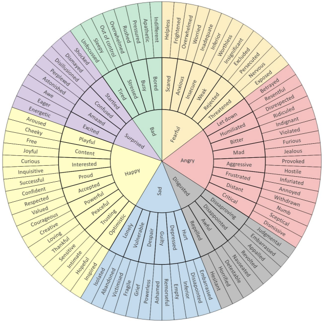


- Word classification: one prediction per word in context
  - Part-of-speech tagging
  - Named entity recognition
  - Information extraction



An example: sentiment analysis

- Determine the **internal state** of
  - a speaker
  - a writer
  - a person mentioned in a text
- Subjectivity
  - “John has a car"
  - “John doesn’t like cars"
  - “This car pollutes a lot"
- Emotions and feelings
- Intention
  - Trust and truth
  - Fake news



Another example: named entity recognition

- Named entities
  - Unique (named) references to real-world entities
- Mention detection (referential expressions)
  - Pronominal (pronouns)
  - Nominals (nouns)
  - Named (proper names)
- Coreference resolution
  - Are two mentions referring to the same entity?
- Entity typing
  - Person
  - Organisation
  - Location
  - ...
- Linking
  - Link entities to a dictionary of known entities
  - e.g. Wikipedia / DBpedia

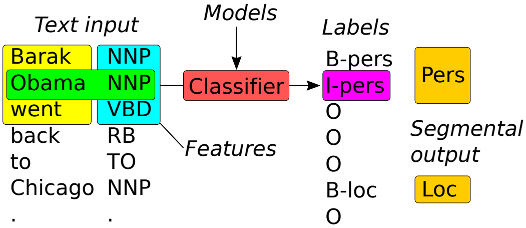
Named entities: exemples

- Le président Obama était en visite à Paris. Il a quitté la ville ce matin.
  - *Le président* (nominal), *Obama* (nommé), *Il* (pronominal)
  - *Paris* (nommé), *la ville* (nominal)
- Paris Hilton est à Paris pour faire des paris avec ses amis.
  - *Paris Hilton* (nommé)
  - *Paris* (nommé)
  - *ses amis* (nominal)
- Jean joue avec le chien de la mère de son frère.
- Il a dit avoir l'avoir connue alors qu'elle était chez eux.
- Il fait trop chaud alors il fait entrer de l'air frais en ouvrant la fenêtre.

Named entities: BIO detector

Convert segmental representation to word-level representation

- (BIO : Begin Inside Outside)
- Begin = first word of segment
- Inside = inside segment
- Outside = outside



Extracting basic features from text

- Historical approaches
  - Text classification
  - Information retrieval
- The bag-of-word model
  - A document is represented as a vector over the lexicon
  - Its components are weighted by the frequency of the words it contains
  - Compare two texts as the cosine similarity between
- Useful features
  - Word n-grams
  - tf$\times$idf weighting
  - Syntax, morphology, etc
- Limitations
  - Each word is represented by one dimension (no synonyms)
  - Word order is only lightly captured
  - No long-term dependencies

$\rightarrow$ word embeddings

Historical approaches

- Text classification
  - Features
    - Bag of words weighted with tf$\times$idf (word space model, WSM), bag of n-grams
    - Latent Semantic Analysis (LSA)
    - Latent Dirichlet Allocation (LDA)
  - Classifiers
    - KNN
    - SVMs with string kernel
    - Random forests
    - Adaboost
- Tagging
  - Hidden Markov Models
  - Maxent models
  - Structured Perceptron
  - Conditional Random Fields

MLP for text classification

- Main idea:
  - Concatenate word embeddings as one big vector
  - Use MLP to predict class label
- Problem 1: texts of varying length
  - Padding: add `<pad>` symbol to vocabulary

w_1	w_2	w_3	$\langle$pad$\rangle$	$\langle$pad$\rangle$
w_1	w_4	$\langle$pad$\rangle$	$\langle$pad$\rangle$	$\langle$pad$\rangle$
w_5	w_2	w_1	w_3	$\langle$pad$\rangle$

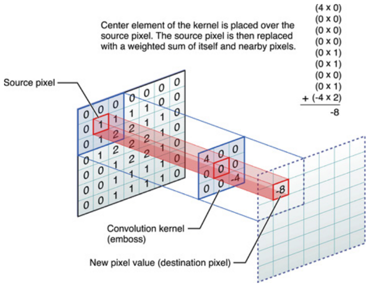
- Problem 2: size of input
  - Documents can be very large
  - Use window over input: cut end of text / randomly sample window
- Problem 3: word order
  - Assume words occur at a given position
  - Does not share parameters for words at different positions

MLP for tagging

- Main idea:
  - Use moving window centered on prediction
  - Concatenate embeddings of words $w_{i-n} \dots w_i \dots w_{i+n}$
  - Works quite well for local phenomena
- Problem 1: boundary effects
  - What to do at beginning and end of text?
  - Replace missing words with padding
    - $w_{-2}, w_{-1}, w_0, w_1, w_2 \rightarrow \langle start \rangle, \langle start \rangle, w_0, w_1, w_2$
    - $w_{l-3}, w_{l-2}, w_{l-1}, w_l, w_{l+1} \rightarrow w_{l-3}, w_{l-2}, w_l, \langle end \rangle, \langle end \rangle$
- Problem 2: word order
  - Same as for classification / beneficial in small window
- Problem 3: dependence on other decisions
  - No coherency between subsequent decisions
  - Can include previous decisions: $(w_{i-2}, d_{i-2}), (w_{i-1}, d_{i-1}) w_i w_{i+1} w_{i+2}$
  - How do we reconcile decisions from forward and backward models?
  - How do we account for errors in decision history?

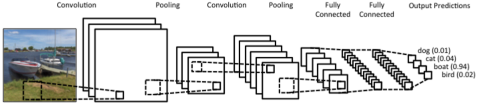
Convolutional Neural Networks (CNN)

- Main idea
  - Created for computer vision
  - How can location independence be enforced in image processing?
  - Solution: split the image in overlapping patches and apply the classifier on each patch
  - Many models can be used in parallel to create filters for basic shapes

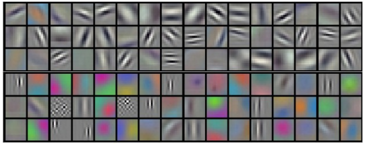


CNN for images

- Typical network for image classification (Alexnet)

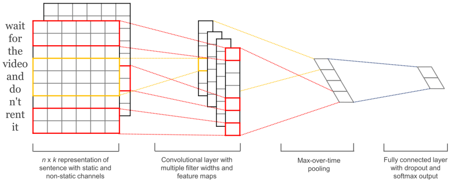


- Example of filters learned for images



CNN for text

- In the text domain, we can learn from sequences of words
  - Moving window over the word embeddings
  - Detects relevant word n-grams
  - Stack the detections at several scales



CNN Math

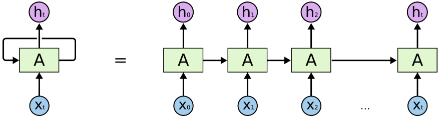
- Parallel between text and images
  - Images are of size (width, height, channels)
  - Text is a sequence of length n of word embeddings of size d
  - $\rightarrow$ Text is treated as an image of width n and height d (1 channel)
- x is a matrix of n word embeddings of size d
  - $x_{i-\frac{l}{2}:i+\frac{l}{2}}$ is a window of word embeddings centered in i, of length l
  - First, we reshape $x_{i-\frac{l}{2}:i+\frac{l}{2}}$ to a size of $(1, l \times d)$ (vertical concatenation)
  - Use this vector for $i \in [\frac{l}{2} \dots n - \frac{l}{2}]$ as CNN input
- A CNN is a set of k convolution filters
  - $\mathrm{CNN}_{out} = activation(W\,\mathrm{CNN}_{in} + b)$
  - CNN_{in} is of shape $(l \times d, n - l)$
  - W is of shape $(k, l \times d)$, b is of shape $(k, 1)$ repeated $n - l$ times
  - CNN_{out} is of shape $(k, n - l)$
- Interpretation
  - If $W(i)$ is an embedding n-gram, then $\mathrm{CNN}_{out}(i, j)$ is high when this embedding n-gram is in the input

Pooling

- A CNN detects word n-grams at each time step
  - We need position independence (bag of words, bag of n-grams)
  - Combination of n-grams
- Position independence (pooling over time)
  - Max pooling $\rightarrow max_t(\mathrm{CNN}_{out}(:, t))$
  - Only the highest activated n-gram is output for a given filter
- Decision layers
  - CNNs of different lengths can be stacked to capture n-grams of variable length
  - CNN+Pooling can be composed to detect large scale patterns
  - Finish by fully connected layers which input the flatten representations created by CNNs

Recurrent Neural Networks

- CNNs are good at modeling topical and position-independent phenomena
  - Topic classification, sentiment classification, etc
  - But they are not very good at modeling order and gaps in the input
    - Not possible to do machine translation with them
- Recurrent definition
  - $h_0 = 0$
  - $h(w_1 \dots w_{i-1}) = h_i = f(h_{i-1})$
  - Uses its previous output to create a representation for the current word



Simple RNNs

- Back to the $y = \text{neural\_network}(x)$ notation
  - $x = x_1 \ldots x_n$ is a sequence of observations
  - $y = y_1 \ldots y_n$ is a sequence of labels we want to predict
  - $h = h_0 \ldots h_n$ is a hidden state (or history for language models)
  - t is discrete time (so we can write x_t for the t-th timestep)
- We can define a RNN as

$$h_0 = 0 \quad (1)$$

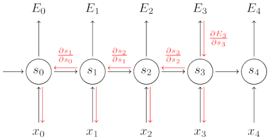
$$h_t = tanh(Wx_t + Uh_{t-1} + b) \quad (2)$$

$$y_t = softmax(W_o h_t + b_o) \quad (3)$$

- Tensor shapes
  - x_t is of shape $(1, d)$ for embeddings of size d
  - h_t is of shape $(1, H)$ for hidden state of size H
  - y_t is of shape $(1, c)$ for c labels
  - W is of shape (d, H)
  - U is of shape (H, H)
  - W_o is of shape (c, H)

Training RNNs

- Back-propagation through time (BPTT)
  - Unroll the network
  - Forward
    - Compute h_t one by one until end of sequence
    - Compute y_t from h_t
  - Backward
    - Propagate error gradient from y_t to h_t
    - Consecutively back-propagate from h_n to h_1



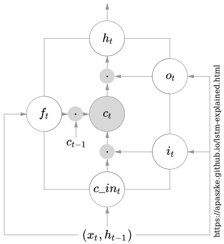
- What if the sequence is too long?
  - Cut after n words: truncated-BPTT
  - Sample windows in the input
  - How to initialize the hidden state?
    - Use the one from the previous window (statefull RNN)

Long-short term memory

- Idea: use gating mechanism to keep information in the hidden state
  - RNN would have to refresh its memory with every input
  - LSTM output depends on gates which are trained to open at the right time
- Gating mechanism

$$g = f(x_t, h_t) \in [0, 1]$$
$$x_{\text{gated}} = g \odot x_t$$

- LSTMs have two hidden states: h and c



LSTM Math

- LSTM

$$
\begin{aligned}
i_t =& \sigma(W_i x_t + U_i h_t + b_i) && \text{input} \\
f_t =& \sigma(W_f x_t + U_f h_t + b_f) && \text{forget} \\
o_t =& \sigma(W_o x_t + U_o h_t + b_o) && \text{output} \\
c'_t =& \tanh(W_c x_t + U_c h_t + b_c) && \text{cell state} \\
c_{t+1} =& f_t \odot c_t + i_t \odot c'_t \\
h_{t+1} =& o_t \odot \tanh(c_{t+1}) \\
\text{LSTM}(x_t, h_t, c_t) =& h_{t+1}
\end{aligned}
$$

- Parameters
  - $W_i, U_i, b_i, W_f, U_f, b_f, W_o, U_o, b_o, W_c, U_c, b_c$
- LSTMs output their hidden state like simple RNNs
  - Need to add a dense layer to predict labels

LSTMs: how can they memorize things?

- Let's have a closer look at the gated output

$$\text{cell}_{t+1} = \text{forget}_t \odot \text{cell}_t + \text{input}_t \odot \text{cell}'_t$$
$$\text{hidden}_{t+1} = \text{output}_t \odot \tanh(\text{cell}_{t+1})$$

- Interpretation
  - if $\text{forget}_t = 1$ and $\text{input}_t = 0$: previous cell state is used
  - if $\text{forget}_t = 0$ and $\text{input}_t = 1$: previous cell state is ignored
  - if $\text{output}_t = 1$: output is set to cell state
  - if $\text{output}_t = 0$: output is set to 0

Gated recurrent units (GRU)

- Same principle but less operations / parameters (Cho et al, 2014)
  - s_t is the hidden state
  - Has to balance between update and forget
- GRU

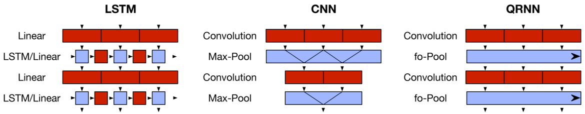
$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z s_t + b_z) && \text{update} \\
r_t &= \sigma(W_r x_t + U_r s_t + b_r) && \text{forget} \\
h_t &= \tanh(W_h x_t + U_h(r_t \odot s_t) + b_h) && \text{input} \\
s_{t+1} &= (1 - z_t) \odot h_t + z_t \odot s_t && \text{new state} \\
\mathrm{GRU}(s_t, x_t) &= s_{t+1}
\end{aligned}
$$

- Parameters
  - $W_z, U_z, b_z, W_r, U_r, b_r, W_h, U_h, b_h$
- Interpretation
  - If $r_t = 0$, h_t does not depend on s_t
  - If $z_t = 0$, use h_t as new state
  - If $z_t = 1$, use s_t as new state

RNNs as fast as CNNs

- RNNs do not parallelize very well (one time step at a time)
  - Quasi-RNNs (Socher et al, ICLR 2017)
  - Simple Recurrent Unit (Lei et al, ICLR 2018)

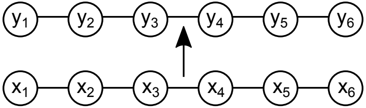
$$\begin{aligned}
\tilde{x}_t &= W_x x_t && \text{input convolution}\\
f_t &= \sigma(W_f x_t + b_f) && \text{forget gate}\\
r_t &= \sigma(W_r x_t + b_r) && \text{reset gate}\\
c_t &= f_t \odot c_{t-1} + (1 - f_t) \odot \tilde{x}_t && \text{recurrent state}\\
h_t &= r_t \odot \tanh(c_t) + (1 - r_t) \odot x_t && \text{skip connection}
\end{aligned}$$



How to use RNNs

- Sentence/document-level classification
  - Drop the prediction of y_t
  - Build hidden state
  - Use the final hidden state as representation for classification
- Word-level classification
  - predict one label y_t per word
  - Useful for part-of-speech tagging, named entity detection, etc.
  - Can do segmentation with (Begin, Inside, Outside) labels
- Language models
  - x_t is the current word
  - y_t is the next word
  - So we estimate $P(w_i|w_{i-1}, h_{i-1})$

Sequence prediction (linear-chain)



Definitions

- Slots:
  - $i = 1 \dots i = n$ for a sequence of length n
- Inputs: $\boldsymbol{x} = x_1 \dots x_n$
  - $x_i \in \mathbb{R}^m$
- Labels: $\boldsymbol{y} = y_1 \dots y_n$
  - $y_i \in \mathcal{Y}$, all possible labels for a given slot
  - $\boldsymbol{y} \in \mathcal{Y}^n$, all possible labellings for length n
- Scoring function: $f(\boldsymbol{y}, \boldsymbol{x})$
  - Best labelling: $\hat{\boldsymbol{y}} = \operatorname{argmax}_{\boldsymbol{y}} f(\boldsymbol{y}, \boldsymbol{x})$

Learning with linear-chain predictions

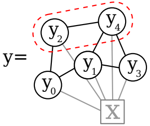
- Loss: $l(\boldsymbol{y}, \boldsymbol{y}')$
  - Assumes both sequences have same length
  - Generally decomposable on slots
  - Hamming loss: $l_h(\boldsymbol{y}, \boldsymbol{y}') = \sum_{i=1}^{n} \mathbb{1}\left[y_i \neq y'_i\right]$
- Parametrizable function $f_\theta(\boldsymbol{y}, \boldsymbol{x})$
- Inference:
  - $\hat{\boldsymbol{y}} = \mathrm{argmax}_{\mathbf{y}}\, f_\theta(\boldsymbol{y}, \mathbf{x})$
- Learning:
  - Given a corpus $\{\boldsymbol{x}, \boldsymbol{y}^\star\}$
  - $\hat{\theta} = \mathrm{argmin}_\theta \sum_{(\mathbf{x}, \boldsymbol{y}^\star)} l\left(\mathbf{y}^\star, \mathrm{argmax}_{\mathbf{y}}\, f_\theta(\mathbf{y}, \mathbf{x})\right)$
- Linear-chain prediction is a special case of *structured* prediction

Potential settings

- Independent predictions
  - $\hat{y}_i = \text{argmax}_{y_i} f(y_i, \boldsymbol{x}) \quad \forall i$
  - Does not depend on neighboring decisions
- Sequential predictions
  - $\hat{y}_i = \text{argmax}_{y_i} f(y_i, \hat{y}_{i-1}, \boldsymbol{x}) \quad \forall i$
  - Depends on past xor future
  - How to reconcile forward and backward predictions?
- Global predictions
  - $\widehat{y_1 \dots y_n} = \text{argmax}_{y_1 \dots y_n} f(y_1 \dots y_n, \boldsymbol{x})$
  - Intractable for large $\mathcal{Y}$ or large n

Decomposable inference

- Global predictions is attractive
  - But each decision depends on all other decisions
  - Requires evaluating $|\mathcal{Y}|^n$ labellings
- Can we decompose f so that it becomes tractable?
  - Make y_i only depend on neighboring predictions
    - $\rightarrow$ can use dynamic programming for inference
  - Assume factors over $\boldsymbol{y}$
    - for example $f(\boldsymbol{y}, \boldsymbol{x}) = \sum_i^{n-1} g(y_i, y_{i+1}, \boldsymbol{x})$
  - Works for graph structures as well
    - Factors need to be cliques around y_i



Probabilistic sequence model

- Reminding that $P(A|B) = \frac{P(A,B)}{P(B)} \rightarrow P(A,B) = P(A|B)P(B)$

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= P(y_1 \dots y_n|\mathbf{x}) \\ &= P(y_n|y_1 \dots y_{n-1}, \mathbf{x})P(y_1 \dots y_{n-1}|\mathbf{x}) \\ &= P(y_1|\mathbf{x}) \prod_{i=2}^{n} P(y_i|y_1 \dots y_{i-1}, \mathbf{x}) \end{aligned}$$

- Limited horizon hypothesis:

$$P(y_i|y_1 \dots y_{i-1}, \mathbf{x}) \simeq P(y_i|y_{i-1}, \mathbf{x})$$

- Independent observation hypothesis:

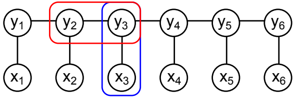
$$P(y_i|y_{i-1}, \mathbf{x}) \simeq P(y_i|y_{i-1})P(y_i|x_i)$$

Hidden Markov Models

- Definition of hidden Markov model:

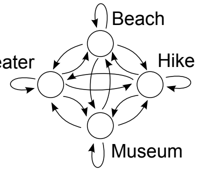
$$P(\mathbf{y}|\mathbf{x}) = P(y_1|x_1)\prod_{i=2}^{n} P(y_i|y_{i-1})P(y_i|x_i)$$

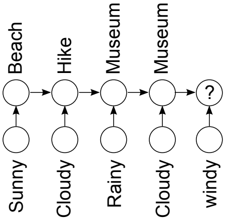
- Hypotheses
  - Limited horizon and independent observations are called *Markov assumptions*
- Components
  - $P(y_i|x_i)$ is called *emission* probability
  - $P(y_i|y_{i-1})$ is called *transition* probability



HMM: Example

- x_i : Weather (Rainy, Cloudy, Sunny, Windy)
- y_i : Activity (Beach, Hike, Museum, Theater)





		y_{i-1}				x_i			
		B	H	M	T	R	C	S	W
y_i	B	0.2	0.3	0.4	0.1	0.1	0.3	0.2	0.4
	H	0.1	0.2	0.3	0.4	0.3	0.2	0.4	0.1
	M	0.4	0.1	0.2	0.3	0.2	0.4	0.1	0.3
	T	0.3	0.4	0.1	0.2	0.4	0.1	0.3	0.2

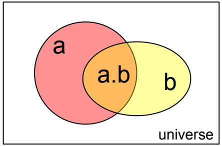
How to estimate HMM parameters?

- Let Ω be the event space, and $count(a)$ the number of times a was observed in Ω.
- Maximum likelihood estimation (frequentist approach):

$$P(a) = \frac{count(a)}{|\Omega|} = \frac{count(a)}{\sum_x count(x)}$$

$$P(a, b) = \frac{count(a \wedge b)}{|\Omega|}$$

$$P(a|b) = \frac{count(a \wedge b)}{count(b)}$$



(Computation with probabilities)

Probability for 1000 independent events to happen together

- $P(a_1, \ldots, a_{1000}) = P(a_1) \times \ldots P(a_{1000})$

If $P(a)$ is uniform, $P(a_i) = \frac{1}{1000}$

- $P(a_1, \ldots, a_{1000}) = \frac{1}{1000 \times 1000 \times \cdots \times 1000}$

Problem

- *double* is only accurate down to 10^{-11} (*float* $\rightarrow 10^{-5}$)
- Multiplying is slow (or used to be)

Solution: Compute in log space

- $\log_{10} \frac{1}{1000} = \log_{10} 1 - \log_{10} 1000 = -3$
- $\log_{10} P(a_1, \ldots, a_{1000}) = \sum_i \log_{10} P(a_i) = -3000$

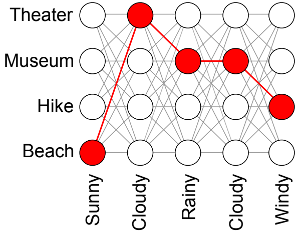
Viterbi algorithm

- How to find the most likely labelling given input/observations only?

$$\hat{\boldsymbol{y}} = \underset{y_0,\dots,y_n}{\operatorname{argmax}} P(y_1|x_1) \prod_{i=2}^{n} P(y_i|y_{i-1})P(y_i|x_i)$$

$$\hat{\boldsymbol{y}} = \underset{y_0,\dots,y_n}{\operatorname{argmax}} \log P(y_1|x_1) \sum_{i=2}^{n} \log P(y_i|y_{i-1}) + \log P(y_i|x_i)$$

Equivalent to the shortest path in a graph where each node has a cost



Viterbi algorithm (pseudo-code)

```
def viterbi(x[]):
  # compute score matrix
  score = [][]
  backtrack = [][]
  for i in 1 .. n:
    for y in labels:
      score[i][y] = max_yprev score[i-1][yprev] + logP(y, yprev) + logP(y, x[i])
      backtrack[i][y] = yprev

  # find highest scoring path with backtracking
  output = []
  i = n
  while i > 1:
    output[i] = backtrack[i][y2]
    y2 = backtrack[i][y2]
    i --
  return output
```

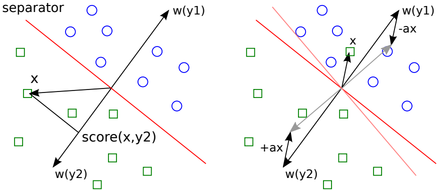
HMM: conclusions

- Uses in NLP
  - Was state-of-the-art (SOTA) for part-of-speech tagging for a long time
  - But input independence hypothesis is wrong in NLP
  - Still used in low-resource speech recognition systems
- Horizon hypothesis
  - Can be extended to longer contexts (HMM of order 1, 2, 3...)
  - What if two non-adjacent labels are dependent?
- Maximum estimation is simple but inaccurate
  - How to handle rare, unseen events?
  - Emissions and transitions can also be estimated with deep models

Perceptron algorithm

Remember the (multiclass) Perceptron algorithm

- For each training instance $(x, y^\star)$
  - Compute $\hat{y} = \text{argmax}_y \, \theta_y^T x$
  - If $\hat{y} \neq y^\star$, update parameters
    - $\theta_{\hat{y}} = \theta_{\hat{y}} - x$
    - $\theta_{y^\star} = \theta_{y^\star} + x$



Strucured Perceptron

The structured Perceptron is an extension for sequences

- Let's extract local features around slot i
  - $\text{feature}_k(y_i, \boldsymbol{x})$
  - For example $\mathbb{1}\left[y_i = \text{DET} \wedge x_i = \text{"the"}\right]$
  - Their weighted sum for a slot plays the role of *emissions*
- Let's define *transition* weights in the parameter vector
  - $\theta_{y_i, y_{i-1}}$
  - For example $\theta_{y_i=\text{NN}, y_{i-1}=\text{DET}}$
- Inference (with Viterbi)

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y}} \sum_i \left(\theta_{y_i, y_{i-1}} + \sum_k \theta_{y_i, k} \text{feature}_k(y_i, \boldsymbol{x}) \right)$$

Structured Perceptron training

Training

- Init with $\theta = 0$
- For each instance $(\boldsymbol{x}, \boldsymbol{y}^\star)$
  - Compute $\hat{\boldsymbol{y}} = \mathrm{argmax}_{\boldsymbol{y}} \sum_i \left(\theta_{y_i, y_{i-1}} + \sum_k \theta_{y_i,k} \text{feature}_k(y_i, \boldsymbol{x})\right)$
  - For $i = 1..n$
    - Update transition parameters

$$\theta_{\hat{y}_i, \hat{y}_{i-1}} = \theta_{\hat{y}_i, \hat{y}_{i-1}} - 1$$
$$\theta_{y^\star_i, y^\star_{i-1}} = \theta_{y^\star_i, y^\star_{i-1}} + 1$$

    - Update emission parameters

$$\theta_{\hat{y}_i, k} = \theta_{\hat{y}_i, k} - \text{feature}_k(\hat{y}_i, \mathbf{x}) \qquad \forall k$$
$$\theta_{y^\star_i, k} = \theta_{y^\star_i, k} + \text{feature}_k(y^\star_i, \mathbf{x}) \qquad \forall k$$

Trick: the averaged perceptron

- Average θ over updates (a kind of regularization)

$$\theta_{avg} = \frac{1}{N}\sum_{t=1}^{N}\theta^{(t)}$$

- Implementation
  - Two parameter vectors: current and accumulator
  - Only update changed weights

$$\begin{aligned}
\theta^{(t)} &= \theta^{(t-1)} + \mathbf{z}^{(t)} \\
\theta_{avg} &= \frac{1}{N}\sum_i \theta^{(i)} \\
&= \frac{1}{N}\left(\theta^{(0)} + \dots + \theta^{(N-1)} + \theta^{(N)}\right) \\
&= \frac{1}{N}\left(\theta^{(0)} + \dots + \theta^{(N-1)} + (\theta^{(N-1)} + \mathbf{z}^{(N)})\right) \\
&= \frac{1}{N}\left(\theta^{(0)} + \dots + 2 \times (\theta^{(N-2)} + \mathbf{z}^{(N-1)}) + \mathbf{z}^{(N)}\right) \\
&= \frac{1}{N}\sum_i (N - i + 1) \times \mathbf{z}^{(i)}
\end{aligned}$$

Structured Perceptron: conclusions

- Log-linear model over sequences
  - Instead of estimating maximum likelihood probabilities, learn weights with Perceptron algorithm
  - Emission probabilities can look at complete $\boldsymbol{x}$ (no more independent input hypothesis)
- In practice
  - Better expressivity than HMMs thanks to decomposable features of the input
  - Very fast parameter update (with respect to inference)
  - Compact models due to $\theta^{(0)} = 0$
  - Best approach (over trees) in syntactic parsing for a long time
  - But coordinate-wise Perceptron updates lack nuance
- Can be extended with max-margin learning (SVM-like)
  - MIRA (Crammer et al, 2003)
  - Adagrad (Duchi et al, 2011)

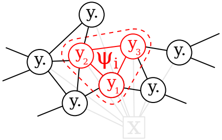
Conditional Random Fields

Probabilistic model of labelling $\boldsymbol{y}$ given observations $\boldsymbol{x}$

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_i \exp\left(\psi_i(\boldsymbol{y}, \boldsymbol{x})\right)$$

$$Z(\boldsymbol{x}) = \sum_{y'} \prod_i \exp\left(\psi_i(\boldsymbol{y}', \boldsymbol{x})\right)$$

- $\boldsymbol{y}$ lies on a graph of slots
- ψ_i is a scoring function on a clique around slot i
- $Z(\boldsymbol{x})$ is a normalization function to make probabilities



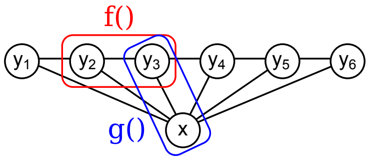
Linear-chain CRF

Linear-chain CRFs are defined over sequences of labels

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_i \exp\left(f(y_i, y_{i-1}) + g(y_i, \boldsymbol{x})\right)$$

$$Z(\boldsymbol{x}) = \cdots$$

- $f(y_i, y_{i-1})$ is a *transition* scoring function
- $g(y_i, \boldsymbol{x})$ is an *emission* scoring function



Inference for CRF

- Express model in log-space

$$\log P(\boldsymbol{y}|\boldsymbol{x}) = -\log Z(\boldsymbol{x}) + \sum_i f(y_i, y_{i-1}) + g(y_i, \boldsymbol{x})$$

- Find best labeling
  - No need to compute Z as it only depends on $\boldsymbol{x}$
  - Viterbi algorithm

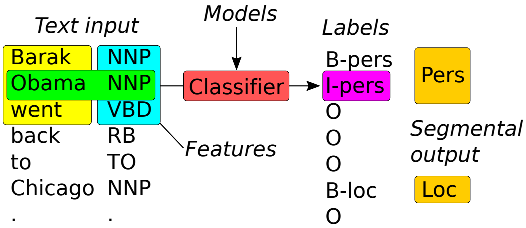
$$\hat{\boldsymbol{y}} = \underset{y_1 \dots y_n}{\operatorname{argmax}} \sum_i f(y_i, y_{i-1}) + g(y_i, \boldsymbol{x})$$

CRF'2003 (the old way)

- Log-linear feature functions

$$f_\theta(y_i, y_{i-1}) = \theta_{y_i, y_{i-1}}$$
$$g_\theta(y_i, \mathbf{x}) = \sum_k \theta_{y_i,k} \mathbb{1}\left[\text{feature}_k(x_{i-n}, \ldots, x_{i+n})\right]$$

- Example from named entity recognition



CRF'2017 (with deep learning toolkits)

- Use deep representation for emissions (LSTM example)
  - The transition layer is a simple 1-parameter activation per label pair
  - The emission layer is projected to generate activations of size m (number of labels).

$$f_\theta(y_i, y_{i-1}) = \theta_{y_i, y_{i-1}}$$
$$g_\theta(y_i, \mathbf{x}) = \mathsf{Linear}_\theta(BiLSTM_\theta(\mathbf{x}))$$

- Inference
  1. Compute activations for emissions g
  2. Compute Viterbi over transition parameters ($\theta_{y_i, y_{i-1}}$) and emissions
  3. Return highest scoring sequence of labels

CRF training

- Parameterize $f_\theta(y_i, y_{i-1})$ and $g_\theta(y_i, \boldsymbol{x})$
- Maximize log-likelihood of training data

$$LL(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y}^\star)} \log P_\theta(\boldsymbol{y}^\star|\boldsymbol{x})$$

$$LL(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y}^\star)} \sum_i f_\theta(y_i^\star, y_{i-1}^\star) + g_\theta(y_i^\star, \boldsymbol{x}) - \log Z_\theta(\boldsymbol{x})$$

- Computing the score of the reference labelling (green)
  - Just sum over the observed cliques in $\boldsymbol{y}^\star$
- Computing the score of all labellings Z (purple)
  - That's a bit more involved

The forward algorithm

$$Z_\theta(\boldsymbol{x}) = \sum_{\boldsymbol{y}'} \exp \sum_i f_\theta(y'_i, y'_{i-1}) + g_\theta(y'_i, \boldsymbol{x})$$

We can use dynamic programming to compute Z efficiently

$$\log \alpha_1(y_2) = \log \sum_{y_1} \exp\left(f_\theta(y_2, y_1) + g_\theta(y_1, \boldsymbol{x})\right)$$

$$\ldots$$

$$\log \alpha_k(y_{k+1}) = \log \sum_{y_k} \exp\left(f_\theta(y_{k+1}, y_k) + g_\theta(y_k, \boldsymbol{x}) + \log \alpha_{k-1}(y_k)\right)$$

$$\ldots$$

$$\log Z_\theta(\boldsymbol{x}) = \log \sum_{y_n} \exp\left(g_\theta(y_n, \boldsymbol{x}) + \log \alpha_{n-1}(y_n)\right)$$

- Need to compute $\log \sum \exp$

(Adding probabilities in log space)

- Computing $\log(a+b)$ as $f(\log(a), \log(b))$? :

$$\begin{aligned}
a+b &= \exp(\log(a+b)) \\
&= \exp(\log(a \times (1+\frac{b}{a}))) \\
&= \exp(\log(a) + \log(1+\exp(\log(\frac{b}{a})))) \\
&= \exp(\log(a) + \log(1+\exp(\log(b) - \log(a)))))) \\
\log(a+b) &= \log(a) + \log(1+\exp(\log(b) - \log(a)))
\end{aligned}$$

- Numerically stable implementation

```
def logsumexp(x, y):
    min = min(x, y)
    max = max(x, y)
    if max - min > epsilon: return max
    else: return max + log(1 + exp(min - max))
```

CRF training

Compute gradient of log likelihood

$$\frac{\partial LL}{\partial \theta} = \sum_i \left(\frac{\partial f(y_i, y_{i-1})}{\partial \theta} + \frac{\partial g(y_i, \mathbf{x})}{\partial \theta} \right) - \frac{\partial \log Z(\mathbf{x})}{\partial \theta}$$

- With deep learning
  - Minimize $-LL$, the negative log-likelihood function
  - Use dynamic toolkits such as pytorch
  - Auto-differentiation finds its way through the forward algorithm
  - Example implementation: `https://pytorch-crf.readthedocs.io`
- Before deep learning
  - Derive analytical solution (not covered in this class)
  - Forward-backward algorithm for Z
  - Numerical optimization with quasi-Neutonian methods (LBFGS)
  - Example implementation: `https://wapiti.limsi.fr/`

Take home message

- Account for decisions in sequence models
  - Independent (i.e. basic RNN)
  - Sequential (forward xor backward)
  - Global: needs a decoder
- Hidden Markov Models (HMM)
  - Markov assumptions: limited horizon and input independence
  - Maximum likelihood estimation of emissions and transitions
  - Viterbi algorithm for best sequence
- Structured Perceptron
  - Feature-based, relax input independence
  - Learning with the Perceptron algorithm
- Conditional random fields (CRF)
  - $P(\boldsymbol{y}|x)$ is a normalized product of exponentials
  - Factorize over cliques of the decisions graph
  - Can be implemented on top of deep contextual representation

Extensions

- Trees
  - Inside-outside algorithm
  - See A. Nasr's class on parsing
- Graphs
  - Markov random fields
  - Junction tree algorithm
  - Loopy belief propagation
- And beyond: general algorithm for max-margin learning
  1. Inference for $\hat{y}$
  2. Compute score of $y^\star$
  3. Make sure score$(y^\star)$ − score$(\hat{y}) >$ errors$(\hat{y}, y^\star)$

Language model (LM)

- Objective
  - Find function that ranks a word sequence according to its likelihood of being proper language
  - Compute probability of text to originate from a corpus

$$
\begin{aligned}
P(w_1 \dots w_n) &= P(w_n | w_{n-1} \dots w_1) P(w_{n-1} \dots w_1) \\
&= P(w_n | w_{n-1} \dots w_1) P(w_{n-1} | w_{n-2} \dots w_1) \\
&= P(w_1) \prod_i P(w_i | w_{i-1} \dots w_1)
\end{aligned}
$$

$$
\begin{aligned}
P(\text{le chat boit du lait}) = & P(\text{le}) \\
& \times P(\text{chat}|\text{le}) \\
& \times P(\text{boit}|\text{le chat}) \\
& \times P(\text{du}|\text{le chat boit}) \\
& \times P(\text{lait}|\text{le chat boit du})
\end{aligned}
$$

N-gram LM

- Apply Markov chain limited-horizon approximation

$$P(mot(i)|historique(1, i-1)) \simeq P(mot|historique(i-k, i-1))$$
$$P(w_i|w_1 \dots w_{i-1}) \simeq P(w_i|w_{i-k} \dots w_{i-1})$$

- For $k = 2$

$$P(\text{le chat boit du lait}) \simeq P(\text{le}) \times P(\text{chat}|\text{le}) \times P(\text{boit}|\text{le chat}) \times P(\text{du}|\text{chat boit}) \times P(\text{lait}|\text{boit du})$$

- Estimation

$$P(\text{boit}|\text{le chat}) = \frac{nb(\text{le chat boit})}{nb(\text{chat boit})}$$

- N-gram LM ($n = k + 1$), uses n words for estimation

LM Smoothing

- Example, bigram model (2-gram) :

$$P(\text{la chaise boit du lait}) = P(\text{la}) \times P(\text{chaise}|\text{la}) \times P(\text{boit}|\text{chaise}) \times \ldots$$

- How to deal with unseen events
- Method of pseudo-counts (Laplace smoothing) (N = number of simulated events)

$$P_{pseudo}(\text{boit}|\text{chaise}) = \frac{nb(\text{chaise boit}) + 1}{nb(\text{chaise}) + N}$$

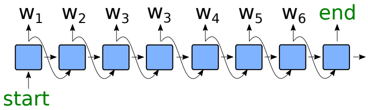
- Interpolation methods

$$P_{interpol}(\text{boit}|\text{chaise}) = \lambda_{\text{chaise}} P(\text{boit}|\text{chaise}) + (1 - \lambda_{\text{chaise}}) P(\text{chaise})$$

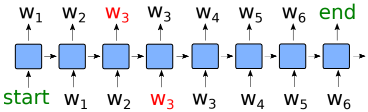
- Backoff methods: like interpolation but only when events are not observed
- Most popular approach: “modified Kneser-Ney" [James et al, 2000]

Neural language model

- Train a (potentially recurrent) classifier to predict the next word

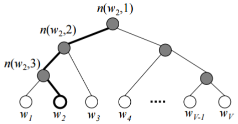


- In training, two possible regimes:
  - Use true word to predict next word
  - Use predicted word from previous slot



Softmax approximations

- When vocabulary is large (> 10000), the softmax layer gets too expensive
  - Store a $h \times |V|$ matrix in GPU memory
  - Training time gets very long
- Turn the problem to a sequence of decisions
  - Hierarchical softmax



- Turn the problem to a small set of binary decisions
  - Noise contrastive estimation, sampled softmax...
  - $\rightarrow$ Pair target against a small set of randomly selected words
- More here: http://sebastianruder.com/word-embeddings-softmax

Perplexity

- How good is a language model?
  1. Intrinsic metric: compute the probability of a validation corpus
  2. Extrinsic metric: use it in a system and compute its performance
- Perplexity (PPL) is an intrinsic measure
  - If you had a dice with one word per face, how often would you get the correct next word for a validation context?
  - Lower is better
  - Only comparable for LM trained with the same vocabulary

$$PPL(w_1 \dots w_n) = p(w_1 \dots w_n)^{-\frac{1}{n}}$$
$$= \prod_{i=1}^{n} p(w_i | w_{i-1} \dots w_1)^{-\frac{1}{n}}$$
$$PPL(w_1 \dots w_n) = \exp_2 \left(-\frac{1}{n} \sum_{i=1}^{n} \log_2 \text{score}(i) \right)$$

Byte-pair encoding (BPE)

Word language models	Character language models
Large decision layer	Don't know about words
Unknown words problem	Require stability over long history

- Word-piece models
  - Split words in smaller pieces
  - Frequent tokens are modeled as one piece
  - Can factor morphology
- Byte pair encoding [Shibata et al, 1999]
  1. Start with alphabet containing all characters
     - ⋆ Split words as characters
  2. Repeat until up to desired alphabet size (typically 10-30k)
     1. Compute most frequent 2-gram (a, b)
     2. Add to alphabet new symbol $\gamma_{(a,b)}$
     3. Replace all occurrences of (a, b) with $\gamma_{(a,b)}$ in corpus

Generation from LM

Given a language model, how can we generate text?

- Start with input $x = \langle\text{start}\rangle$, hidden state $h = 0$
- Repeat until $x = \langle\text{end}\rangle$:
  1. Compute logits and new hidden state $y, h \leftarrow \text{model}(h, x)$
  2. Introduce temperature $y' = y/\theta$
  3. Make distribution $p = \text{softmax}(y)$
  4. Draw symbol from multinomial distribution $\tilde{s} \sim p$
     1. Draw $v \sim \text{Uniform}(0, 1)$
     2. Compute $\tilde{s} = \text{argmax}_s\, v > \sum_{i=0}^{s} p_i$
  5. $x \leftarrow \tilde{s}$
- Temperature θ modifies the distribution ($\theta = 0.7$ is a good value)
  - $\theta < 1$ is more conservative results
  - $\theta > 1$ leads to more variability

Neural LM: conclusions

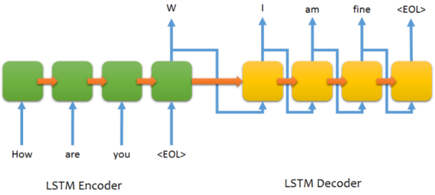
- Use (recurrent) classifier to predict next word given history
  - Typically train on true history
- Evaluation
  - Perplexity, but not really related to downstream usefulness
- Large decision layer for realistic vocabulary
  - Softmax approximations
  - Maybe words are not the best representation

Neural machine translation (NMT)

- Phrase-based translation
  - Same coverage problem as with word-ngrams
  - Alignment still wrong in 30% of cases
  - A lot of tricks to make it work
  - Researchers have progressively introduced NN
    - Language model
    - Phrase translation probability estimation
  - The google translate approach until mid-2016
- End-to-end approach to machine translation
  - Can we directly input source words and generate target words?

Encoder-decoder framework

- Generalisation of the conditioned language model
  - Build a representation, then generate sentence
  - Also called the seq2seq framework

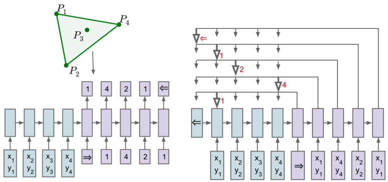


- But still limited for translation
  - Bad for long sentences
  - How to account for unknown words?
  - How to make use of alignments?

Interlude: Pointer networks

- Decision is an offset in the input
  - Number of classes dependent on the length of the input
  - Decision depends on hidden state in input and hidden state in output
  - Encoder state e_j, decoder state d_i

$$y_i = softmax(v^\intercal tanh(We_j + Ud_i))$$



(a) Sequence-to-Sequence (b) Ptr-Net

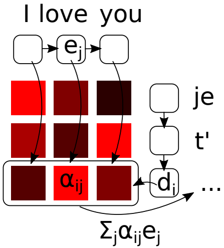
Oriol Vinyals, Meire Fortunato, Navdeep Jaitly, "Pointer Networks", arXiv:1506.03134

Attention mechanisms (Bahdanau et al, 2014)

- Loosely based on human visual attention mechanism
  - Let neural network focus on aspects of the input to make its decision
  - Learn what to attend based on what it has produced so far

- Given $\mathbf{e}$ and $\mathbf{d}$ two vectors for encoder and decoder states:

$$\alpha_i = \text{softmax}_j(f_{\text{align}}(d_i, e_j))$$
$$\text{attn}_i = \sum_j \alpha_{i,j} e_j$$



- Additive attention

$$f^{+}_{\text{align}}(d_i, e_j) = v^{\intercal}\, tanh(Wd_i + Ue_j)$$

- Multiplicative attention

$$f^{\times}_{\text{align}}(d_i, e_j) = d_i^{\intercal}\, We_j$$

Attention is all you need (Vaswani et al, 2017)

- A replacement for seq2seq RNNs
  - Attention treats words as a bag
    - ★ RNN allows convey word order
    - ★ Need to encode position information as embeddings
  - Also attend on same sequence: self-attention
- Increase capacity
  - Multiple attention heads: allows to focus on multiple phenomena
  - Multiple layers of attention: encode variables conditioned on subsets of inputs
- The baby is named “Transformers"[1]
  - Encoder-decoder with multiple layers of multi-head attention
  - State of the art on Machine Translation

[1] `http://jalammar.github.io/illustrated-transformer/`

Conclusion: tasks

- Text classification
  - Input: sequence of words
  - Task: one prediction/class label per text
  - Compromise
    - take into account word order
    - modeling long inputs
- Word classification
  - Input: sequence of words
  - Task: one prediction per word in context
  - Challenges
    - Use non-local information
    - Use other decisions from model
  - Best solved as sequence classification
- Segmentation: convert sequence tagging with “BIO" labels
- Relation prediction: classify pairs of texts

Conclusion: models

- Convolutional Neural Networks (CNN)
  - Learn to apply a filter on a moving window of the input
  - Position independent
  - Interpretable as word n-grams
- Recurrent Neural Networks (RNN)
  - State depends on previous state
  - Can model varying length history
  - Potentially model the whole history
- Sequence models
  - Decompose label graph as cliques
  - Optimize for best labeling
  - Extends to trees and graphs
- Translation models
  - Sequence to sequence
  - Attention mechanisms

Outline

Representation learning

- What is a representation?
  - Mathematical object to represent an input (symbolic or continuous) in a system
  - Why we need them?
    - Input of mathematical models, systems
    - Visualize, analyse and better understand phenomena
- What's a good representation for words?
  - One-hot representation, bag-of-word representations
  - Enriched with output of linguistic annotation process
    - How to handle ambiguity?
    - What about errors?
- Entails a few more questions
  - How does it interact with higher-level linguistics (syntax, semantics, pragmatics...)?
  - How universal such a representation can be?
    - Across domains and applications
    - Across languages and cultures

Learning representations?

- Designing representations: feature engineering
  - Expert-motivated choice of relevant factors
  - Does not account for noise sources well, nor generalizes well
  - Example for Part-of-speech tagging[2]:

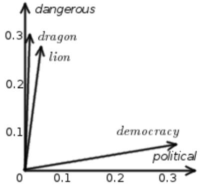
Word	Num	Cap	Sym	P1	P2	P3	P4	S1	S2	S3	S4
Time	N	Y	N	T	Ti	Tim	Time	e	me	ime	Time
flies	N	N	N	f	fl	fli	flie	s	es	ies	lies
like	N	N	N	l	li	lik	like	e	ke	ike	like
an	N	N	N	a	an	∅	∅	n	an	∅	∅
arrow	N	N	N	a	ar	arr	arro	w	ow	row	rrow
.	N	N	Y	.	∅	∅	∅	.	∅	∅	∅

- Learning representations
  - Originates from computer vision where feature design is hard
  - Input a stream of bits and let the system discover relevant representations from data
  - Train on large **unannotated** datasets

[2] `https://pageperso.lis-lab.fr/benoit.favre/jsviterbi/`

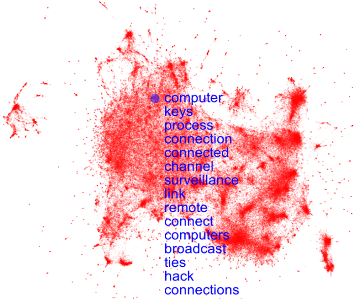
Word embeddings

- Word embeddings in neural networks
  - From one-of-n sparse representations to low dimensional dense vectors
  - Each word is associated with a location in that space
  - Word location can be moved through back-propagation
- Main idea: pre-training
  - Need representations for words not seen in task dataset
  - Pre-train from large quantities of text vs small task dataset
- Distributional semantic hypothesis
  - Two words occurring in the same context are likely to have similar meaning (Harris 1954, Firth 1957)
  - Train representations based on how words co-occur



Example: word embeddings

- Word embeddings (Fasttext-300 trained on 2B words from Opensubtitles[3])



[3] https://pageperso.lis-lab.fr/benoit.favre/opensubtitles-umap/

Latent Semantic Analysis (LSA)

	w_1	w_2	w_3	w_4	w_5
d_1		1		3	
d_2					
d_3	1	2		1	
d_4	2				2
d_5	3	1			

- Latent semantic analysis (LSA, 1998)
  - Create a word by document matrix M: $m_{i,j}$ is the log of the frequency of word i in document j.
  - Perform a SVD on the coocurrence matrix $M = U\Sigma V^T$
  - Use U as the new representation (U_i is the representation for word i)
  - Since M is very large, optimize SVD (Lanczos' algorithm...)
- Extensions
  - Build a word-by-word cooccurrence matrix within a moving window
  - Only use most silent dimensions of U
  - Random indexing (Sahlgren, 2005)

Historical approaches: Random indexing

- Random indexing (Sahlgren, 2005)
  - Associate each word with a random $n - hot$ vector of dimension m (example: 4 non-null components in a 300-dim vector)
  - It is unlikely that two words have the same representation, so the vectors have a high probability of being an orthogonal basis
  - Create a $|vocab| \times m$ cooccurrence matrix
  - When words i and j cooccur, add the representation for word j to row i
  - This approximates a low-rank version of the real coocurrence matrix
  - After normalization (and optionally PCA), row i can be used as new representation for word i
- Need to scale to very large datasets (billions of words)

Global vectors (GloVe)

- Main idea (Pennington et al, 2014)
  - Scalar product between word vectors should reflect their coocurrence

	w$_1$	w$_2$	w$_3$	w$_4$	w$_5$
w$_1$		1		3	
w$_2$					
w$_3$	1	2		1	
w$_4$	2				2
w$_5$	3	1			

- Training
  - Start from (sparse) cooccurrence matrix $\{m_{ij}\}$
  - Then minimize following loss function

$$Loss = \sum_{i,j} f(m_{ij}) \left(w_i^T w_j + b_i + b_j - \log m_{ij} \right)^2$$

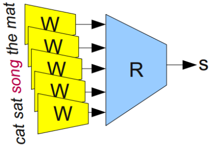
- f dampers the effect of low frequency pairs, in particular $f(0) = 0$
- Worst-case complexity in $|vocab|^2$, but
  - Since $f(0) = 0$ only need to compute for seen coocurrences
  - Linear in corpus size on well-behaved corpora

Corrupted n-grams (Bottou 2011)

- Approach: learn to discriminate between existing word n-grams and non-existing ones
  - Input: 1-hot representation for each word of the n-gram
  - Output: binary task, whether the n-gram exists or not
  - Parameters W and R (W is shared between word positions)
  - Mix existing n-grams with corrupted n-grams in training data

$$r_i = Wx_i \quad \forall i \in [1 \ldots n]$$

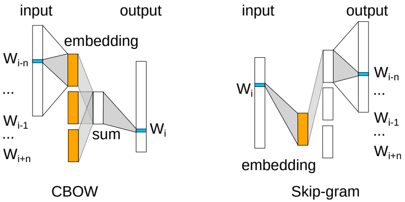
$$y = softmax(R\sum_{i=1}^{n} r_i)$$



- Extension: train any kind of language model
  - Recurrent language models (Bengio 2003)
  - Continuous-space language model (Schwenk 2007)
  - Multi-task systems (tagging, named entity, chunking, etc) (Collobert 2011)

Word2vec

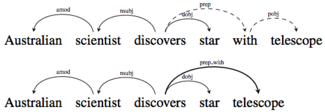
- Proposed by (Mikolov et al, 2013)
- Task
  1. CBOW: Given bag-of-word from window, predict central word
  2. Skip-gram: Given central word, predict another word from the window



- Training (simplified)
  - For each word-context (x, y) :
    - $\hat{y} = softmax(Wx + b)$
    - Update W and b via error back-propagation

Syntax-aware embeddings

- Dependency embeddings (Levy et al, 2014)
  - Use dependency tree instead of context window
  - Represent word with dependents and governor
  - Makes much more syntactic embeddings



WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

Morphological embeddings

- Account for morphology
  - Generate representations for unseen words
  - Account for the functional role of morphology
- RNN:
  - Accumulate character-level representations
- Predict morphological features (Cotterell et al, 2015)
  - Features: tense, genre, case, animacy...
  - Requires large training set
- Fasttext (Bojanowski et al, 2016)
  - word2vec on (form + character n-grams)
  - (forest, ˆfo, for, ore, res, est, st$) $\rightarrow$ context

Enforcing linguistic relations

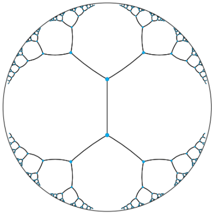
- How to introduce linguistic knowledge in embeddings?
  - Word categories
  - Wordnet relations
  - Domain-specific ontologies
- Extract cooccurrences from ontology (RDF2Vec, OWL2Vec...)
  - word $\rightarrow$ definition
  - works for large ontology
- Retrofit with semantic lexicons (Faruqui et al, 2015)
  - Pretrain a regular embedding space $\hat{q}$
  - Move vectors so that neighbors in $G = \{E, V\}$ are close

$$\min_{w} \sum_{i \in V} \left(\alpha ||w_i - \hat{q}_i|| + \beta \sum_{(i,j) \in E} ||w_i - w_j||^2 \right)$$

- Joint training (Alsuhaibani et al, 2018)
  - Modify embedding training loss to account for knowledge source

Pointcaré embeddings

- Poincaré Embeddings for Learning Hierarchical Representations (Nickel et al 2017)
  - Can embed taxonomies such as Wordnet, and graphs
- Use Pointcaré-ball as hyperbolic space
  - Replace squared error (euclidian distance) by Pointcaré distance in loss softmax
  - Optimize Riemman gradient



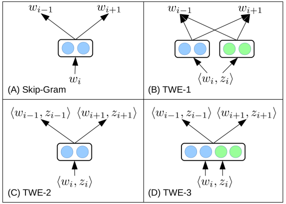
$$L(\theta) = \sum_{u,v \in D} log \frac{exp\left(-d(u,v)\right)}{\sum_{v' \in N(u)} exp\left(-d(u,v')\right)}$$

$$d(u,v) = arcosh\left(1 + 2\frac{||u-v||^2}{(1-||u||^2)(1-||v||^2)}\right)$$

- u, v are word vectors, D is a set of observed hierarchical relationships

Sense-aware embeddings

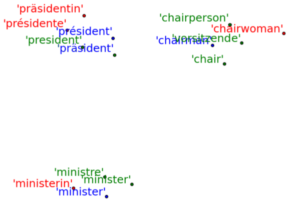
- Multi-prototype embeddings (Huang et al, 2012; Liu et al, 2015)
  - Each word shall have one embedding for each of its senses
  - Hidden variables: a word has n embeddings
  - Can pre-process with topic tagging (LDA)



Word	Prior Probability	Most Similar Words
apple_1	0.82	strawberry, cherry, blueberry
apple_2	0.17	iphone, macintosh, microsoft
bank_1	0.15	river, canal, waterway
bank_2	0.6	citibank, jpmorgan, bancorp
bank_3	0.25	stock, exchange, banking
cell_1	0.09	phones, cellphones, mobile
cell_2	0.81	protein, tissues, lysis
cell_3	0.01	locked, escape, handcuffed

Multilingual embeddings

- Can we create a single embedding space for multiple languages?
  - Train bag-of-word autoencoder on bitexts (Hermann et al, 2014)
    - ★ Force sentence-level representations (bag-of-words) to be similar
    - ★ For instance, sentence representations can be bag-of-words

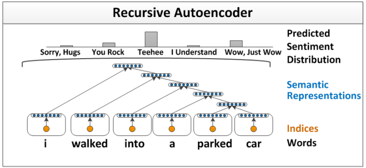


- Align embedding spaces (Conneau et al, 2017)
  - Use multilingual dictionary to map embedding spaces

$$M = \operatorname*{argmin}_W \sum_i ||x_i - Wy_i||^2$$

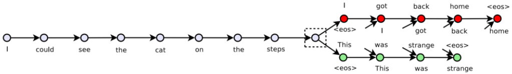
Compositional meaning

- Task-adapted embeddings (Socher et al.)
  - Combine word-level embeddings
  - Follow parse tree, learn constituent-specific combiners
  - Sentence representation is supervised by task (Sentiment analysis)



Sentence and document embeddings

- Skip-Though vectors (Kiros et al, 2015)
  - Train a system to generate the next and previous sentence from the current sentence
  - Sentences that appear in the same context will have similar embeddings



- Doc2vec / paragraph vectors (Le & Mikolov, 2014)
  - Represent sentences in one-hot vector (very high dimensional)
  - Train word2vec or similar algorithm

Take home: non-contextual embeddings

- One-hot encoding for words is limited
  - Need to account for semantic / syntactic proximity
  - Results in very large sparse vectors
- Can model words according to their “average" neighborhood
  - Words that appear in similar context are given the same representation
  - Based on co-occurrence matrix approximations
  - No linguistic knowledge required
  - Small dense vectors
- Pretraining on large corpora
  - Representation learning
  - Plug “frozen" embedding layer while training system
  - Virtually no unseen words in test
- Limits
  - Single sense hypothesis
  - Independent of context
  - Requires large unannotated datasets

Outline

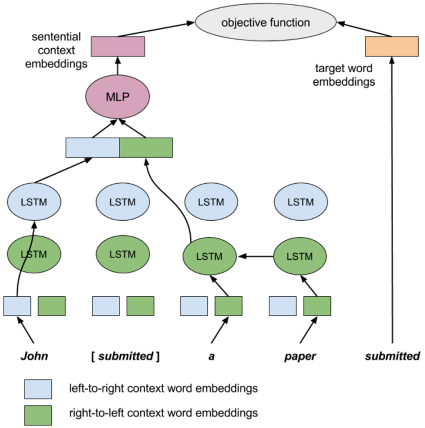
Contextualizing embeddings

- Limitations of non-contextual embeddings
  - Representation independent of context
  - Single-sense assumption / how to disambiguate senses?
  - How to account for domain?
  - How to represent unknown words?
  - Also a limitation for sentences and documents
- But good ideas to keep
  - Large quantities of text available
  - Language modeling self-supervision tasks

Context2vec [Melamud et al 2016]

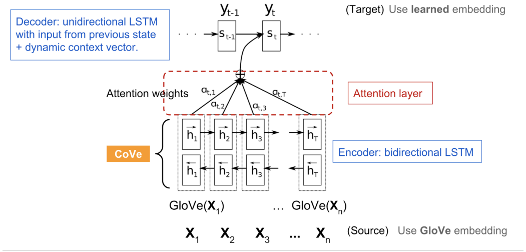
- Main idea: extend word2vec to generate representations for contexts
  - Same objective as CBOW
  - Context embedding taken from single-layer Bi-LSTM

Sentential Context	Closest target words
This [ ] is due	item, fact-sheet, offer, pack, card
This [ ] is due not just to mere luck	offer, suggestion, announcement, item, prize
This [ ] is due not just to mere luck, but to outstanding work and dedication	award, prize, turnabout, offer, gift
[ ] is due not just to mere luck, but to outstanding work and dedication	it, success, this, victory, prize-money



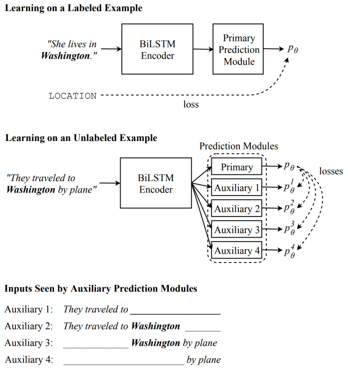
Contextual Word Vectors (CoVe) [McCann et al 2017]

- Machine translation
  - Trained on English-German corpora (7 million sentences)
  - 2-layer Bi-LSTM, attention mechanism, LSTM decoder
  - Input GloVe vectors



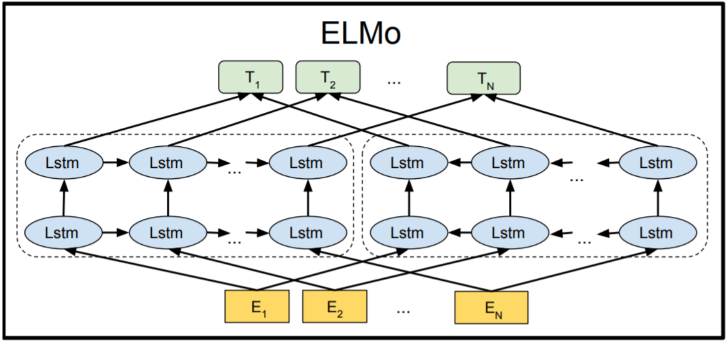
Cross-view training [Clark et al 2018]

- Train on multiple auxiliary tasks
  - Multitask objective
  - Similar to [Colobert et al 2011] “Natural language processing from scratch"
  - Speeds up training by 50%
  - Character CNN + word embeddings, 2-layer BiLSTM, parallel decision layers
  - Tasks are specific to layers
    - Forward and backward next-word prediction
    - Predict word classes given context
    - Predict target task (NER, parsing...)



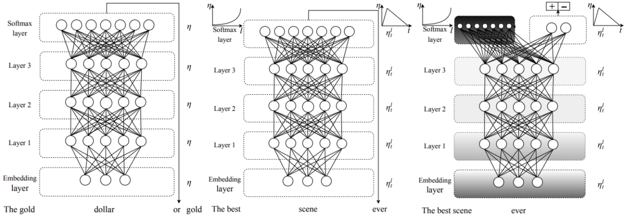
Em-beddings from Language Models (ELMo) [Peters et al, 2018]

- Forward and backward language model
  - CNN for character representations
  - Token embeddings
  - Multiple layers (disjoint) in each direction
  - But share embedding and decision layers
  - Merge outputs to create representations
  - Trained on 5B words (wikipedia + news sources)



ULMFit [Howard and Ruder 2018]

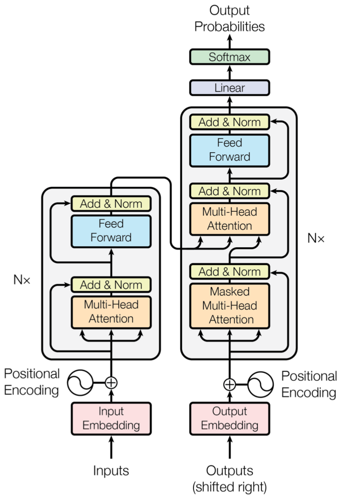
- Universal Language Model Fine-tuning for Text Classification
  - 3-LSTM Language model
  - Next-word prediction objective
  - Corpus: 103 million words (wikipedia)
- Introduces the concept of fine-tuning
  - Learning rate schedule
  - Fine-tune all layers



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Attention is all you need [Vaswani et al 2017]

- “Transformer" architecture
  - Only based on attention mechanism
  - No CNN/RNN at all
  - Multiple parallel attention heads
  - Originally developed for machine translation
- Introduces the concept of self-attention
  - Attention between tokens of the input
- Position encoding
  - Words are treated as a bag
  - Need a way to reintroduce position information

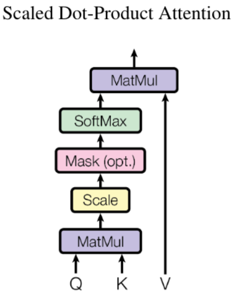


Self-attention

- Idea: allow each input to be conditioned on another input
  - Q, K, V are projections of X
  - Extension of multiplicative attention
  - d is the dimension of the attention head
  - Scale-down prior to softmax for well-behaved computations

$$
\begin{aligned}
Q &= X \cdot W_1 && \text{query} \\
K &= X \cdot W_2 && \text{key} \\
V &= X \cdot W_3 && \text{value}
\end{aligned}
$$

$$Attn_{self} = softmax(\frac{Q \cdot K^\top}{\sqrt{(d)}})V$$

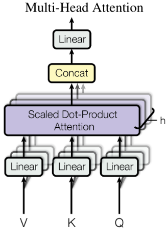


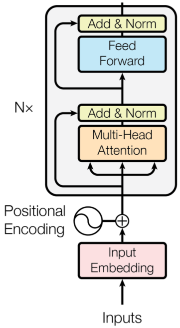
Encoder block

- Parallel attention heads
  - Can condition on multiple parts of the input
- Residual network (θ_1 and θ_2 are vect. of param.)

$$\text{LayerNorm}(x) = \theta_1 + \theta_2 \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}$$
$$B = \text{LayerNorm}(X + Attn_{self})$$

- Decoder block is similar but
  - Also attends encoder token representations
  - Masked so that cannot attend future (not yet generated) tokens



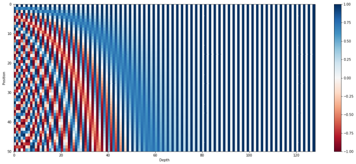


Transformers: position encoding

- Compute embedding for position t

$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k.t), & \text{if } i = 2k \\ \cos(\omega_k.t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$



- Easy to model relative positions

$$M_\phi \cdot \vec{p_t} = \vec{p_{t+\phi}}$$

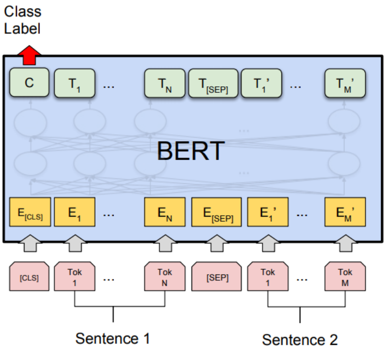
Transformers: Word-piece models

Word language models	Character language models
Large decision layer	Don't know about words
Unknown words problem	Require stability over long history

- Word-piece models
  - Split words in smaller pieces
  - Frequent tokens are modeled as one piece
  - Can factor morphology
- Byte pair encoding (Shibata et al, 1999)
  1. Start with alphabet containing all characters
     - Split words as characters
  2. Repeat until up to desired alphabet size (typically 10-30k)
     1. Compute most frequent 2-gram (a, b)
     2. Add to alphabet new symbol $\gamma_{(a,b)}$
     3. Replace all occurrences of (a, b) with $\gamma_{(a,b)}$ in corpus

Bidirectional Transformers pre-training (BERT) [Delvin et al 2018]

- Representation learning with transformers
  - Encoder-part of a transformer
  - 12 layers, 12 attention heads
  - Byte-pair encoding of input
  - Large context (512 tokens)
- Trained on language modeling tasks
  - Masked language model (MLM): Select 15% of words
    - 80% replaced by `[MASK]`
    - 10% replaced by random word
  - Next-sentence prediction (NSP): 50% actual next sentences
  - Training data: 4B words from Books and Wikipedia
- Fine-tuning on tasks
  - Replace decision layers by task-specific decision layer
  - Adapt **all** parameters to target task



Evaluation

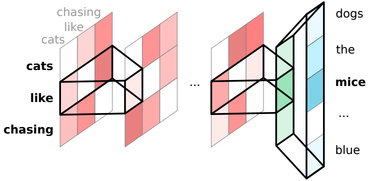
- GLUE benchmark [Wang 2019]
  - CoLA: grammaticality
  - SST-2: sentiment polarity
  - MRPC: paraphrase detection
  - STS-B: text similarity
  - QQP, QNLI: question answering
  - MNLI, RTE: natural language inference
  - WNLI: winograd schema
  - AX: language understanding

Task	**Mean**	**CoLA**	**SST-2**	**MRPC**	**STS-B**	**QQP**	**MNLI**	**QNLI**	**RTE**	**WNLI**	**AX**
CBOW	58.6	0.0	80.0	81.5	61.2	51.4	56.0	72.1	54.1	62.3	9.2
Skip-thought	61.3	0.0	81.8	80.8	71.8	56.4	62.9	72.9	53.1	65.1	12.2
CoVe	63.6	14.5	88.5	81.4	67.2	59.4	64.5	75.4	53.5	61.6	20.6
ELMo	67.7	32.1	89.3	84.7	70.3	61.1	67.2	75.5	57.4	65.1	21.3
BERT	80.2	59.2	94.3	88.7	87.3	71.5	85.4	92.4	71.6	65.1	9.2
SotA[4]	**90.7**	74.8	97.0	94.5	92.8	74.7	91.3	97.8	92.0	94.5	52.6

[4] As of 2020-09-02 `https://gluebenchmark.com/leaderboard`

Generative Pre-trained Transformer

- GPT-1: “Improving Language Understanding by Generative Pre-training" [Radford et al, 2018]
  - Trained on 7000 books
  - 12 Layers (BERT size), 117 million parameters
  - New SotA on 12 tasks
- GPT-2: “Language Models are unsupervised multitask learners" [Radford et al, 2019]
  - Trained on 8 million documents
  - Add token to identify the task
  - 48 layers, 1.5 billion parameters
- GPT-3: “Language models are few-shot learners" [Brown et al, 2020]
  - Trained on Common Craw texts (410 billion tokens)
  - 96 layers, 175 billion parameters
  - Zero-shot: describe task as input to model

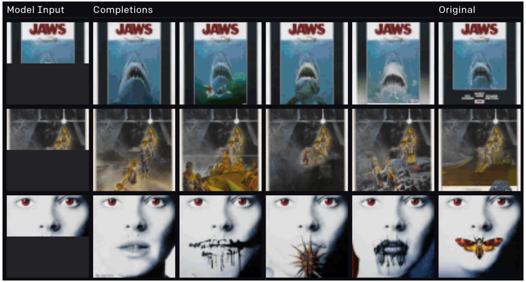


Transformers variants

- DistilBERT, TinyBERT... (Sanh et al 2019; Jiao et al 2019)
  - Train smaller model to achieve same activations as full BERT
  - up to 10 times smaller for small loss in performance
- Multilingual BERT
  - Train on multilingual Wikipedia (104 languages)
  - Can fine-tune cross-lingual tasks
- Very long inputs
  - Hierarchical network: (Zhang et al, 2019)...
  - Sparse attention matrix: Reformer (Kitaev et al, 2020), transformer XL (Dai et al, 2019)...

Transformers beyond text

- Music transformers[5] (Huang et al 2018)
  - Train from midi files played by pianists
  - Attention adds long term coherence to generated pieces
- Image GPT (Chen et al, 2020), Image transformers (Parmar et al, 2018)



[5] https://magenta.tensorflow.org/piano-transformer

Take-home: self-supervision tasks

- NWP: next word prediction (autoregressive language model)
  - Classical language model criterion
  - Learn to predict next word given history
- MLM: masked language model
  - Replace random words with `[MASK]` in input
  - Given representation for `[MASK]` token, predict the original identity of masked word
- NSP: next sentence prediction
  - Given two sentences, do they follow each other in the text?
  - Add tokens: `[CLS]` sentence1 `[SEP]` sentence 2
  - Add segment identity bit to input
  - Given representation for `[CLS]` token, preform binary prediction
- Many variants
  - Document rotation prediction
  - Emoji prediction
  - Punctuation prediction...

Take-home: contextual embeddings

- Non-contextual word representation limitations
  - Single-sense assumption
  - Does not account for context
  - Requires large input layers ($>$ 1 million tokens)
  - Handling unseen words
- Variants on BERT/GPT are the current state of the art
  - Transformer architecture with self-attention
  - Train on large dataset
  - Fine-tuning on target tasks
  - Extensions: handle long texts, reduce model size
- Emerging properties
  - Generalization across tasks, domains and languages
  - Can describe the task as input of the model

Outline

Evaluation of word embeddings

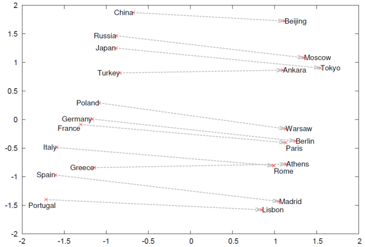
- How do word embeddings perform on typical NLP tasks?
- Task-oriented (Ghannay et al. 2016)
  - POS: Part-of-speech tagging
  - CHK: Syntactic Chunking
  - NER: Named-entity recognition
  - MENT: Mention detection for coreference resolution

Model	POS	CHK	NER	MENT
W2V CBOW	96.01	90.48	78.32	55.49
W2V Skip-gram	96.43	89.64	77.65	57.80
GloVe	95.79	86.90	76.45	54.49
CSLM	96.24	90.11	76.20	57.34
w2vf-deps	**96.66**	**92.02**	**79.37**	**58.06**

- Small differences between models
- Overall, linguistically-informed models seem to perform better

Linguistic regularities

- Linear relations
  - king + (woman - man) = queen
  - paris + (italy - france) = rome
- Inflection
  - Plural, gender
  - Comparatives, superlatives
  - Verb tense
- Semantic relations
  - Capital / country
  - Leader / group
  - analogies



Evaluating linguistic regularities

Subtask	**CBOW**	**Skip-gram**	**GloVe**	**CSLM**	**w2vf-deps**
Capital cities	89.5	88.3	**93.1**	34.0	71.5
Capital-world	81.0	88.2	**92.2**	16.1	34.4
Currency	9.5	**17.6**	16.6	1.1	8.8
City-in-state	22.9	27.2	**36.2**	3.5	6.2
Family	**86.8**	76.5	81.4	66.8	74.3
Adjective-to-adverb	13.3	18.2	**22.3**	7.3	5.3
Opposite	24.9	34.1	22.5	20.4	**36.9**
Comparative	81.4	79.3	84.8	70.1	**87.6**
Superlative	61.2	69.4	65.0	45.0	**71.5**
Present-participle	62.6	65.3	**66.7**	39.9	60.1
Nationality-adjective	81.1	86.7	**91.2**	25.6	25.8
Past-tense	55.1	56.7	**59.2**	54.1	55.9
Plural	54.0	55.0	**69.8**	17.0	59.9
Plural-verbs	37.8	61.8	48.4	48.5	**86.8**

- Overall, GloVe seem to perform better at this task

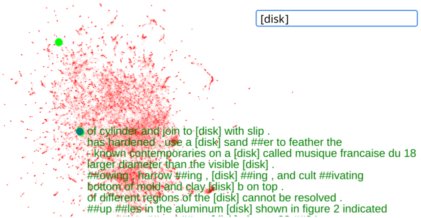
Task-specific embeddings

- Variants in embedding training
  - Lexical: words
  - Part-of-speech: joint model for (word, pos-tag)
  - Sentiment: also predict smiley in tweet

Lexical		Part-of-speech		Sentiment	
good	**bad**	**good**	**bad**	**good**	**bad**
great	**good**	great	good	great	terrible
bad	terrible	bad	terrible	goid	horrible
goid	baaad	nice	horrible	nice	shitty
gpod	horrible	gd	shitty	goood	crappy
gud	lousy	goid	crappy	gpod	sucky
decent	shitty	decent	baaaad	gd	lousy
agood	crappy	goos	lousy	fantastic	horrid
goood	sucky	grest	sucky	wonderful	stupid
terrible	horible	guid	fickle-minded	gud	:/
gr8	horrid	goo	baaaaad	bad	sucks

What do contextual embeddings learn?

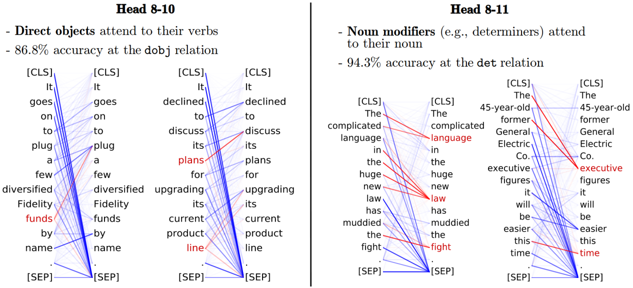
- BERT-like models are good when fine-tuned on target tasks
  - But how do they obtain so good performance?
  - Do they simulate a linguistic pipeline?
  - How do they encode linguistic/general knowledge?
- Potential analysis
  - Outputs for ambiguous/synonym words[6]
  - Attention mechanism
  - Probing tasks



[6] https://pageperso.lis-lab.fr/benoit.favre/bert-umap/

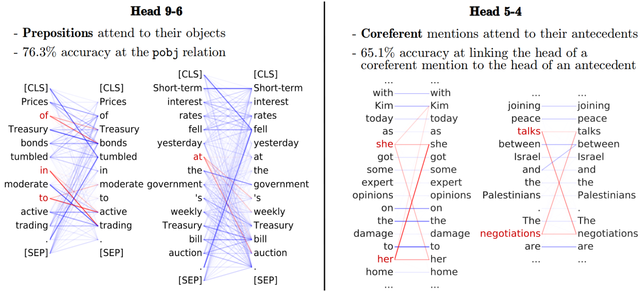
Attention heads

- What Does BERT Look At? An Analysis of BERTs Attention (Clark et al, 2019)



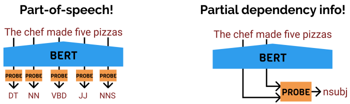
Attention heads (2)

- What Does BERT Look At? An Analysis of BERTs Attention (Clark et al, 2019)



Linguistic probing

- Probing tasks
  - Address simple linguistic task
  - Train simple model from untouched representations



- Warning: confounding factors
  - Is the simple model learning everything?
  - Can the task be achieved from non-relevant factors (memorizing words)
- Control
  - Learn from random representations
  - Learn on arbitrary tasks that depend on word types
  - Look at unseen words
- Designing and Interpreting Probes with Control Tasks (Hewitt & Liang 2019)

Linguistic probes in action

- Parse trees?
  - Idea: find subspace isomorphic to known syntactic trees
    - tree distances $\Leftrightarrow$ squared distance between word vectors
    - node depth $\Leftrightarrow$ squared norm of word vectors
  - Note: not a property specific to transformers

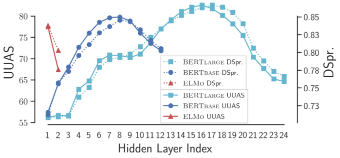


Figure 1: Parse distance UUAS and distance Spearman correlation across the BERT and ELMo model layers.

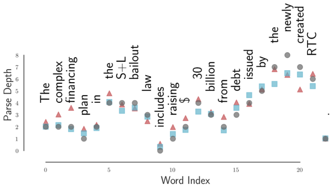
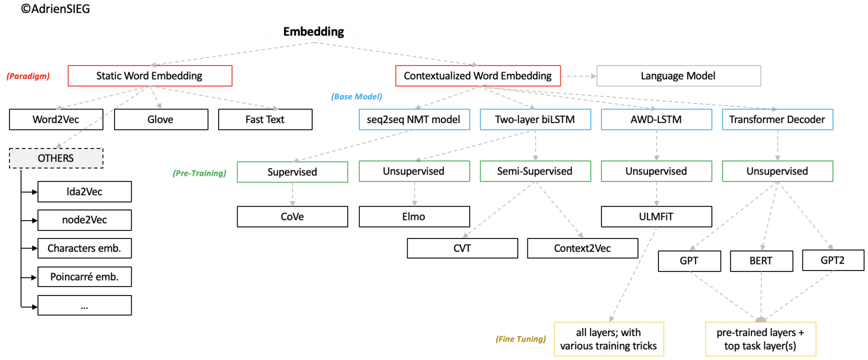


Figure 3: Parse tree depth according to the gold tree (black, circle) and the norm probes (squared) on ELMo1 (red, triangle) and BERTLARGE16 (blue, square).

- A Structural Probe for Finding Syntax in Word Representations (Hewitt, Manning 2019])

Outline

A typology of language representations



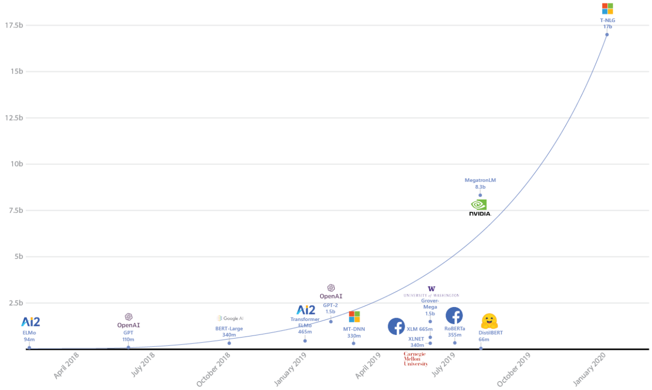
Take home message

- Computer scientists' model of language
  - Pre-training of representations
  - Non-contextual vs contextual embeddings
  - Emergence of linguistic properties
- The *current* recipe for natural language processing
  1. a spoon of transformers
  2. an ounce of self training
  3. a cup of large datasets
  4. a dash of finetuning on a target task
  5. cook for a few weeks in a datacenter

Open questions

- What is the correct NN architecture?
  - Why does it get better with more data and more parameters?
  - Why can't it learn a compact model from the start?
- What are the correct set of self-training tasks?
  - How to go beyond data biases?
- What is *discovered* through pre-training?
  - Linguistic regularities
  - Few-shot learning
- How to ground the model in reality?
  - Cross-reference multimodal inputs?
  - Is the neural network also learning the world?
- What does it mean for other disciplines if the model of language is not *bounded*?

Model size


17.5b
15b
12.5b
10b
7.5b
5b
2.5b
T-NLG
17b
MegatronLM
8.3b
NVIDIA
ELMo
94m
OpenAI
GPT
110m
Google AI
BERT-Large
340m
Transformer
ELMo
465m
OpenAI
GPT-2
1.5b
MT-DNN
330m
XLM 665m
XLNET
340m
Carnegie Mellon University
UNIVERSITY of WASHINGTON
Grover-
Mega
1.5b
RoBERTa
355m
DistilBERT
66m
April 2018
July 2018
October 2018
January 2019
April 2019
July 2019
October 2019
January 2020