Computational Intelligence Lab

Project 2 Introduction
Sentiment Classification

Jakob Nogler

ETH Zürich

(Slides adapted from Dario Pavllo)

April 25th, 2024

Outline

- Project on sentiment classification
  - Introduction to task
  - Baselines and recent literature
  - Useful tips for project & report
- Review of NLP concepts
  - Word embeddings
  - Patterns for sentence classification
  - Implementation of some techniques in PyTorch
  - Practical considerations

Section 1

Project on Sentiment Classification

Sentiment Classification

Given a tweet, predict whether it has a positive or negative sentiment

- Labels obtained using **distant supervision** as opposed to human annotators
- If a tweet originally contained a :) it is positive, if it contained a :(it is negative
- Expect some label noise (e.g. sarcasm)!

Examples:

- "i know android sucks :("
- "twitter is dead right now :("
- "my sis made apple crisp with extra crisp ! it's awesome :)"
- "i hope your wednesday was awesome :)"

Project 2: The Dataset Provided

- Twitter data:
  - train_pos_full.txt $\sim$ 1M tweets that contained :)
  - train_neg_full.txt $\sim$ 1M tweets that contained :(
  - train_pos.txt - 10% from the positive tweets for training
  - train_neg.txt - 10% from the negative tweets for training
  - test_data.txt - 10K unlabeled tweets for the Kaggle submission
- Each tweet contains at most 140 characters
- All tweets are tokenized - words are separated by a single whitespace
- All labels (smileys) are removed
- User mentions replaced with <user>
- Links replaced with <url>
- Hashtags are preserved as-is (beware!)

Some Examples

Positive Tweets

```
<user> anddd to cheer #nationals2013 ?
<user> i love you ! if you love me to please rt
omg ! ! ! just did mi hair ! ! ! im amazing ! ! : oxoxox to mi followers ! ! ! muaaa
<user> good to know <3
```

Negative Tweets

```
rt if #justinbieber is not following you
not enough sleep
hmm , barca to win 3-1 i think
```

Kaggle competition

`https://www.kaggle.com/competitions/ethz-cil-text-classification-2024`

- **Goal:** classify test_data.txt
- **Metric:** accuracy
  - Classes are already balanced

Dataset size & Overfitting

- The dataset size (2M tweets) would be considered small by today's standards!
- State-of-the-art NLP models (transformers) are likely to overfit
  - Tip #1: try simpler models & explore various regularization strategies
  - Tip #2: fine-tune instead of training from scratch (many approaches are possible; finding the best one for this task is up to you)

Large language models (LLMs)

- The use of large language models is **allowed**
- However, the pipeline should be fully reproducible
  - This means you should use publicly available models, understand the techniques to correctly prompt these models, and write the implementation yourself
  - You should not adopt ready-to-use APIs (e.g. ChatGPT API)

Expectations and experimental soundness

- For the final report, it is important for us to see that you followed a sound scientific process
- Start from simple baselines (e.g. bag-of-words, linear models), then progressively move to more complex models
  - Using a more complex model should be motivated quantitatively! No need to use a complex model if the same accuracy can be achieved by a simpler model
  - Sometimes computational cost is also a factor (100x computation time for a 0.1% accuracy boost: is it really worth it?)
- Do not validate your model on the Kaggle leaderboard! Use a local validation set or k-fold cross validation
- Inspect your results manually to understand failure modes

Creativity

- Creativity is a relevant part of the grading scheme
  - What does it mean?
- Quantitative results are important, but we would like to see something more in the report
  - Methodical interpretation of results and failure modes
  - Elegant handling of certain aspects (e.g. hashtags)
  - ...

Grading Scheme

- Accounts for 30% of course grade.
- Competitive grade (30%):

$$4 + 2 \times (\mathtt{x} - \mathtt{baseline})/(\mathtt{max\_score} - \mathtt{baseline})$$

- Non-competitive grade (70%):

  paper quality (30%) + novelty (20%) + implementation quality (20%)

- Details on moodle.
- Competitive performace is not the most important!

Section 2

Review of NLP concepts

Embeddings

- Embeddings are central to NLP/ML techniques
- A real-valued **vector** that captures the meaning of something
  - Not limited to words! The idea can also be applied to images, videos, whole sentences, etc.
  - Multi-modal embeddings: embed different modalities in the same space (e.g. images and text)
  - **Thought vectors** (Geoffrey Hinton)

- An embedding on its own is useless; it becomes useful when it is related to other embeddings
  - Concepts that are semantically similar should be close to each other in the embedding space
- "Semantically similar" and "close" are arbitrary concepts that depend on the adopted training algorithm

Word Embeddings

Suppose we are given a dictionary of words $\mathcal{V} = \{w_1, w_2 \dots\}$.
The *i*-th word

$$w_i \in \mathcal{V}$$

is represented by an embedding

$$\mathbf{x}_{w_i} \in \mathbb{R}^d,$$

a *d*-dimensional latent vector.

Embeddings capture the meaning of the words:

- Related words should have similar embeddings $\mathbf{x}_{w_i} \approx \mathbf{x}_{w_j}$
- Similarity typically expressed as dot product $\langle \mathbf{x}_{w_i}, \mathbf{x}_{w_j} \rangle$

Discrete Representation

Represent a vector by its index in the vocabulary

$$[0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ \ldots\ 0\ 0\ 0]$$

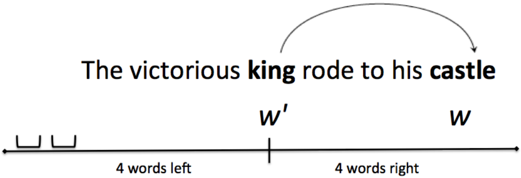
— "one-hot" vector representation.

Problems:

- Dimensionality
  English Language (1M vocab),
  Twitter-2B tweets (1.2M vocab)

- Does not capture similarity of words
  good $= [0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  great $= [0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  milk $= [0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
  (good AND great) $=$ (good AND milk) $= 0$

Distributional similarity-based representations



- **Words that appear in similar contexts have a similar meaning**
- word2vec, GloVe, FastText
  - Same goal, slightly different formulation & training algorithm
- Almost obsolete by today's standards, but the concept has greatly influenced subsequent works

Example: GloVe

Summarize data in **co-occurrence matrix**

$$\mathbf{N} = (n_{ij}) \in \mathbb{R}^{|\mathcal{V}|\cdot|\mathcal{V}|},$$

$$n_{ij} = \#\text{ occurrences of } w_i \in \mathcal{V} \text{ in context of } w_j \in \mathcal{V}$$

Example corpus:

- The king rode to his castle.
- The king lives in the castle.

counts	the	king	rode	lives	to	in	his	castle
the	0	2	0	0	0	1	0	1
king	2	0	1	1	0	0	0	0
rode	0	1	0	0	1	0	0	0
lives	0	1	0	0	0	1	0	0
to	0	0	1	0	0	0	1	0
in	1	0	0	1	0	0	0	0
his	0	0	0	0	1	0	0	1
castle	1	0	0	0	0	0	1	0

GloVe Objective (matrix factorization)

Weighted least squares fit of log-counts

$$\mathcal{H}(\theta; \mathbf{N}) = \sum_{i,j} f(n_{ij}) \left(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta(w_i | w_j)}_{\text{model}} \right)^2,$$

with **unnormalized** distribution

$$\tilde{p}_\theta(w_i | w_j) = \exp\left[\langle \mathbf{x}_i, \mathbf{y}_j \rangle + b_i + d_j\right]$$

- No need to compute normalization constant (great speed up)

and **weighting function** f.
$\mathbf{x}_i$: word embeddings
$\mathbf{y}_i$: context embeddings

Embeddings & deep learning frameworks

Deep learning frameworks provide efficient embedding layers

- E.g. `nn.Embedding` in PyTorch
- Takes as input an integer (word index), returns vector from lookup table
- Typically the very first layer in NLP architectures

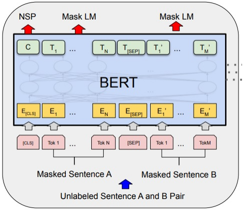
An embedding layer is mathematically equivalent to multiplying a one-hot vector with a matrix, but requires less memory!

$$\underset{\text{One-hot vector}}{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}} \times \underset{\text{Embedding Weight Matrix}}{\begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix}} = \underset{\text{Hidden layer output}}{\begin{bmatrix} 1 & 3 & 5 & 8 \end{bmatrix}}$$

Source

Contextualized embeddings

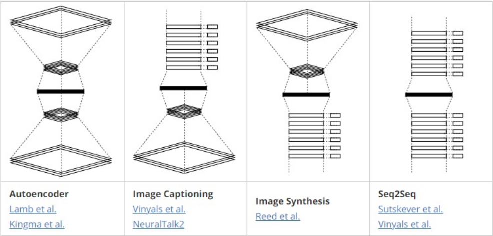
- Words are sometimes ambiguous...
- A single embedding cannot capture all meanings of a word
  - "light": illumination, lightweight, or low-calorie?
- The meaning of a word often depends on the context
- Representations learned by state-of-the-art NLP models can be seen as **contextualized embeddings**.
  - Initial layers learn representations similar to what we have seen so far (word2vec, GloVe)
  - Final layers: custom representations that depend on the surrounding words

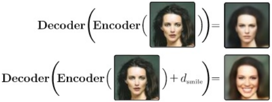


Beyond words

Everything is an embedding!

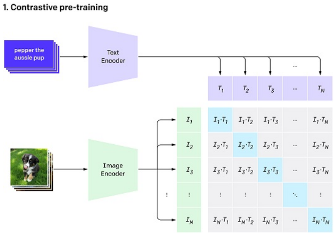
- The final activations of a CNN for image classification
- The hidden state of a recurrent neural network
- The latent representation of an autoencoder
  - `https://gabgoh.github.io/ThoughtVectors/`

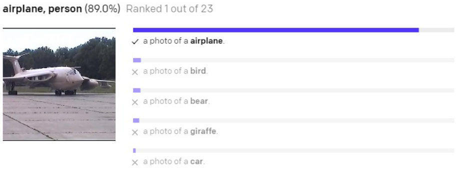




Multi-modal embeddings

- E.g. OpenAI CLIP
- Embed images and sentences in the same space
- Trained via contrastive learning on a huge corpus of (image, caption) pairs from the web
- Can be used for zero-shot classification





Source: `https://openai.com/blog/clip/`

The NLP pipeline in one slide

- Split the dataset into training/validation set (e.g. 90/10)
- Tokenize each sentence
  - "I am a sentence. Another sentence."
    ["I", "am", "a", "sentence", ".", "Another", "sentence", "."]
- Build a vocabulary and convert each token to an index
  - [0, 1, 2, 3, 4, 5, 3, 4]
- Train a machine learning model to predict the label
  - Input layer: embedding, one-hot vector, or bag-of-words
- Evaluate on validation set, tune hyperparameters, or try out a different model
- **Use libraries, do not reinvent the wheel!**

Simple linear baselines (to get started)

- Bag-of-words (each word has a positive or negative weight)
  - Variant: remove stop-words
  - Variant: term weighting (e.g. `tf-idf` features)
  - Variant: bigrams (pairs of words) instead of single words
- Embed each word using pretrained word embeddings (GloVe), then average embeddings across a tweet and train a linear classifier on top of it
  - Freeze embeddings or fine-tune them? The only way to know for sure is to try it out
- These baselines can be implemented using existing libraries (e.g. `sklearn`): highly recommended to do so

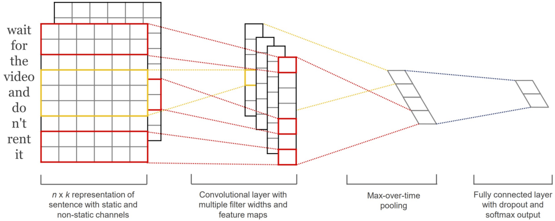
Neural Network baselines

In order of complexity...

- MLP on top of embeddings
  - As before, can initialize using GloVe embeddings
- 1D Convolutional neural networks
- (Bidirectional) Recurrent neural networks
  - LSTM, GRU
- Self-attention models (transformers)
  - BERT, XLNet, etc.
  - Basically all state-of-the-art approaches in NLP are based on transformers
  - Data hungry!

Baselines: 1D CNN

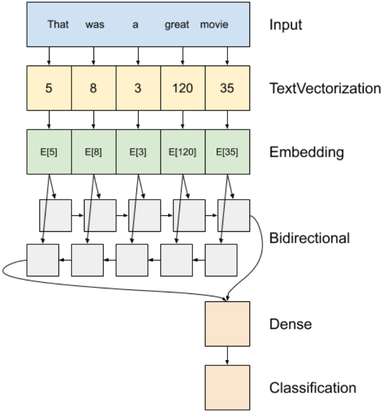
- Input layer: embedding layer (shared across word position)
- 1D Convolutional Layers (`nn.Conv1d` in PyTorch) + ReLU
- Some form of pooling (max or average) to reduce the sentence to a single vector
- Final layer: fully-connected + softmax for classification



Source: Yoon Kim 2014

Baselines: RNN

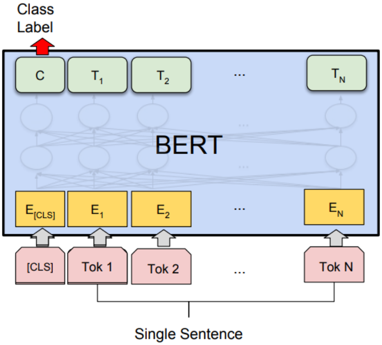
- Use LSTM or GRU architectures, not vanilla RNN!
- Bidirectional: Left-to-right and right-to-left stream
- The final representation is fed through a fully-connected layer

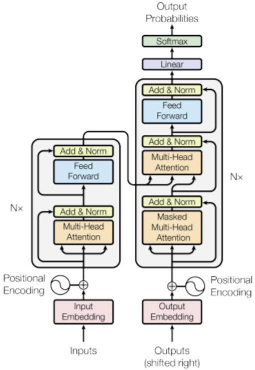


Source:
https://www.tensorflow.org/tutorials/text/text_classification_rnn

Baselines: self-attention / transformers

- As opposed to RNNs, which process the sentence sequentially, self-attention layers look at the entire sentence at once
- The sentence is processed through multiple attention layers (fixed length from input to output)





Types of transformer-based language models

- Masked language models (full self-attention)
  - E.g. BERT, XLNet, RoBERTa, T5
  - Useful for representation learning; these are not generative models!
  - Typically need to fine-tune on a specific task
- Causal language models (large language models)
  - E.g. GPT-1/2/3/4, LLaMA
  - Sequential generation
  - Can be used for downstream tasks in zero-shot or few-shot fashion (see later); not very useful for fine-tuning
- Conversational language models
  - E.g. ChatGPT, Open Assistant
  - Causal language models fine-tuned to generate conversational-style text

Querying language models

- **Prompting** $\rightarrow$ extracting knowledge from LLMs
- Few-shot classification
  - Describe the task and provide a few examples, then let the model answer
  - `Classify the sentiment of the following tweet.`
    `Answer with a single word: [positive, negative]`
    `Examples:`
    `‘‘i am happy’’: positive`
    `‘‘i am sad’’: negative`
    `Tweet: ‘‘i am great’’`
    `Answer:` [let the model complete]
- Zero-shot classification
  - Describe the task (no examples) and let the model answer directly

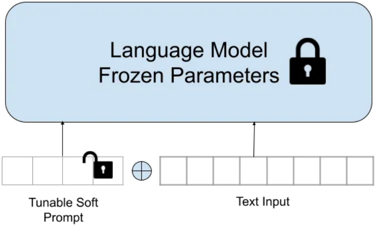
Prompt engineering

- The way you prompt the model has a great impact on the final result
- Better prompting techniques can lead to improved results
- A lot of recent literature on **prompt engineering**
- Up to you to find what works best

Advanced prompting techniques

- Prompting vs fine-tuning
  - Can we get the best of both worlds?
  - Fine-tuning LLMs is too expensive and might overfit easily

- Prompt tuning (soft prompting)
  - Optimize a prompt in embedding space using labeled data
  - The model itself is frozen



Source: `https://savasy-22028.medium.com/prompting-in-nlp-prompt-based-zero-shot-learning-3f34bfdb2b72`

Transformers tips

- Use `huggingface` transformers library
  - `https://github.com/huggingface/transformers`
  - Support for both TF and PyTorch
- Explore various fine-tuning strategies
  - Fine-tune entire model
  - Freeze all layers except the last; fine-tune the last layer
- Training from scratch is likely to overfit
- Consider smaller variants of the model to reduce overfitting and computation time
- Tweets are short (max 140 chars in the dataset): pointless to use model variants devised for large receptive fields

Publicly available language models

- Masked language models (for fine-tuning)
  - BERT, XLNet, RoBERTa
- Causal language models (zero-shot/few-shot)
  - GPT-2 – OpenAI (up to 1.5B parameters)
  - LLaMA – FAIR (7B–65B parameters)
  - Stable LM – Stability AI (3B–7B parameters)
- More exhaustive list:
  `https://github.com/Hannibal046/Awesome-LLM`

Domain expertise

- Remember that we are dealing with tweets, not news or "clean" language
- Hash tags, "Twitter language", memes, slang, typos
- Are off-the-shelf tokenizers good enough?
- Are pretrained embeddings such as word2vec GloVe good enough for tweets?
  - Do a proper literature review / research before getting started with the project
  - E.g. FastText embeddings (uses character-level information; more robust to typos / rare words)
- Inspect your data! Inspect your trained model!

Sample implementation

- See notebook...
  `https://colab.research.google.com/github/dalab/lecture_cil_public/blob/master/exercises/2021/Project_2.ipynb`
  - Bag-of-words baseline with logistic regression
  - Simple model interpretation (visualizing words associated with highest/lowest weights)
  - Coming up with better models is your job
  - Have fun!