Deep Learning for Natural Language Processing

Jindřich Libovický

April 3, 2025

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Today's Learning Outcomes

After this lecture you should be able to ...

1. Describe neural networks as **continuous functions** and takes inputs and generates outputs
2. Describe how neural networks **are trained** and reason about training neural networks with respect to **gradient flow**
3. Statically represent words and tell how to neural networks **represent words in context**
4. Describe **pre-training** of neural networks for NLP

Outline

Deep Learning in NLP

- NLP tasks learn end-to-end using deep learning — the number-one approach in current research
- State of the art in POS tagging, parsing, named-entity recognition, machine translation, ...

☺ Good news: training without *almost* any linguistic insight
although it is not always a good idea

☹ Bad news: requires enormous amount of training data and really big computational resources
although that changes with pre-trained models

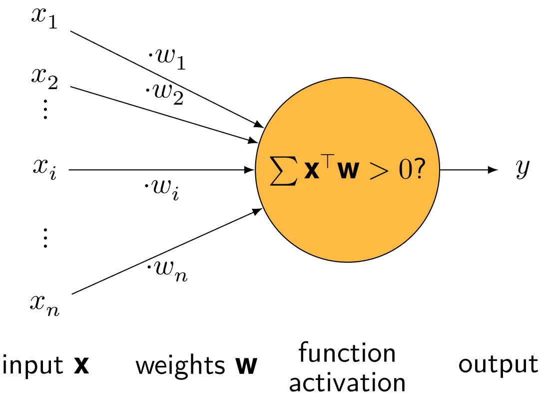
Neural Networks Basics

What is deep learning?

- Buzzword for machine learning using **neural networks** with many layers using back-propagation
- Learning of a **real-valued function** with millions of parameters that solves a particular problem
- Learning more and more abstract **representation of the input data** until we reach such a suitable representation for our problem

Neural Networks Basics

Single Neuron (Perceptron)

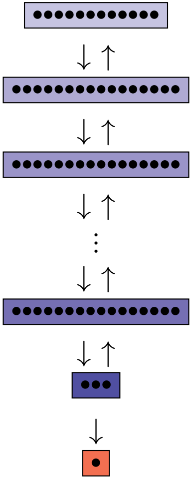


The old view:

$\boxed{\textbf{a network of artificial neurons}}$

- simplistic model of a neuron from the 1940's
- a neuron has some (weighted) inputs, when the input is high enough, it fires a signal
- focus on single neurons does not allow thinking about layers as vector representations

The current view: **a network of layers**



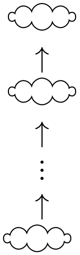
input vector

n hidden layers

output vector

error

$$x$$
$$\downarrow$$
$$h_1 = f(W_1 x + b_1)$$
$$\downarrow$$
$$h_2 = f(W_2 h_1 + b_2)$$
$$\downarrow$$
$$\vdots$$
$$\downarrow$$
$$h_n = f(W_n h_{n-1} + b_n)$$
$$\downarrow$$
$$o = g(W_o h_n + b_o)$$
$$\downarrow$$
$$E = e(o, t) \quad \rightarrow$$



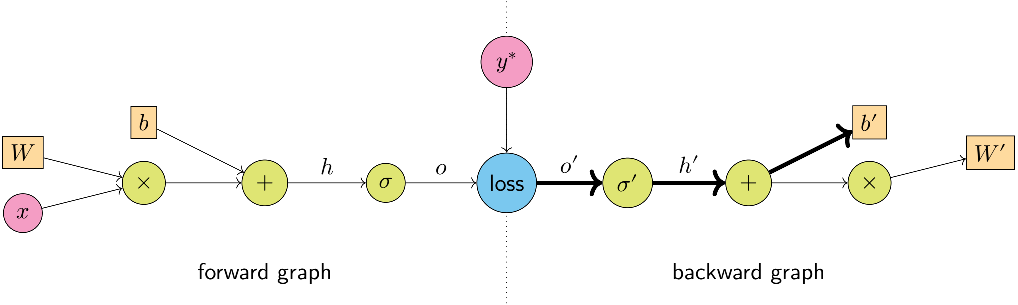
$$\uparrow$$
$$\frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial W_o}$$
$$\uparrow$$
$$\frac{\partial E}{\partial o}$$

Implementation: Computation graph

Logistic regression:

$$y = \sigma\left(Wx + b\right) \tag{1}$$

Computation graph:



Deep learning frameworks – TensorFlow, Pytorch – do it automatically.

Representing Words

Representing Words

General Architecture Overview

1.

Embed input words.

Get a sequence of continuous vectors

2.

Contextualize input.

Apply a sequence processing architecture and get contextual representation.

3.

Get some output.

Typically classification or labeling.

The problem of representing words

Problem:

Words (and characters) are discrete $\times$ **Inputs to neural nets must be continuous**

Spoiler: The solution is called **embeddings**

Let's discuss it on the problem of language modeling

Language Modeling

- estimate probability of a next word in a text

$$\mathrm{P}(w_i | w_{i-1}, w_{i-2}, \dots, w_1)$$

- standard approach: n-gram models with Markov assumption

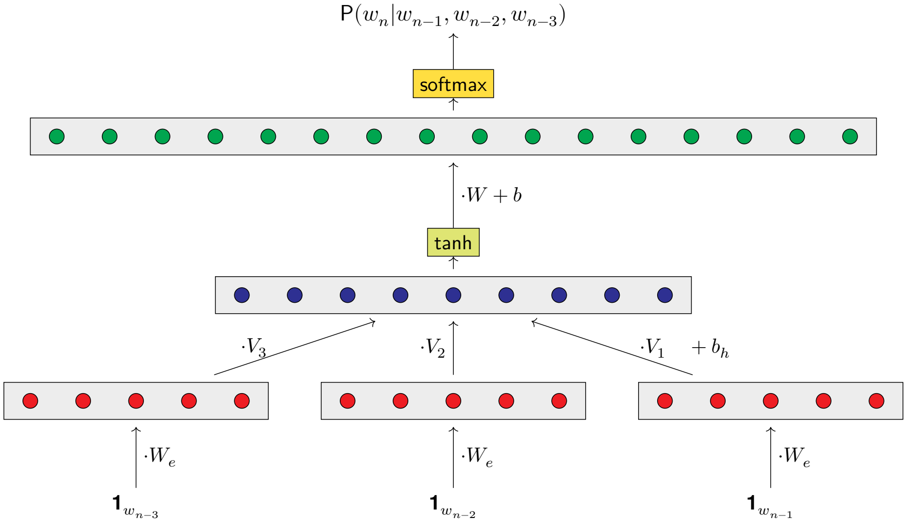
$$\approx \mathrm{P}(w_i | w_{i-1}, w_{i-2}, \dots, w_{i-n}) \approx \sum_{j=0}^{n} \lambda_j \frac{c(w_i | w_{i-1}, \dots, w_{i-j})}{c(w_i | w_{i-1}, \dots, w_{i-j+1})}$$

- Let's simulate it with a neural network:

$$\dots \approx F(w_{i-1}, \dots, w_{i-n} | \theta)$$

θ is a set of trainable parameters.

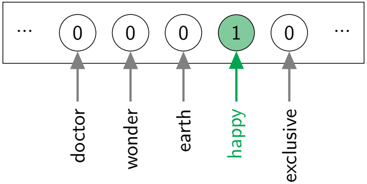
Simple Neural Language Model



Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3 (Feb):1137–1155, 2003. ISSN 1532-4435

Neural LM: Word Representation

- limited vocabulary (hundred thousands words): indexed set of words
- words are initially represented as one-hot-vectors $\mathbf{1}_w = (0, \ldots, 0, 1, 0, \ldots 0)$



- projection $\mathbf{1}_w \cdot V$ corresponds to **selecting one row** from matrix V
- V: is a table of learned word vector representations

so-called **word embeddings**

The first hidden layer is then (matrix V is shared for all words):

$$h_1 = V_{w_{i-n}} \oplus V_{w_{i-n+1}} \oplus \ldots \oplus V_{w_{i-1}}$$

Neural LM: Next Word Estimation

- optionally add extra hidden layer:

$$h_2 = f(h_1 W_1 + b_1)$$

- last layer: probability distribution over vocabulary

$$y = \text{softmax}(h_2 W_2 + b_2) = \frac{\exp(h_2 W_2 + b_2)}{\sum \exp(h_2 W_2 + b_2)}$$

- training objective: cross-entropy between the true (i.e., one-hot) distribution and estimated distribution

$$E = -\sum_i p_{\text{true}}(w_i) \log y(w_i) = \sum_i -\log y(w_i)$$

- learned by error back-propagation

Learned Representations

- word embeddings from LMs have interesting properties
- cluster according to POS & meaning similarity

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using the Euclidean metric, which was chosen arbitrarily).

Table taken from Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011. ISSN 1533-7928

- in IR: query expansion by nearest neighbors
- in deep learning models: embeddings initialization speeds up training / allows complex model with less data

Implementation in PyTorch I

```
import torch
import torch.nn as nn

class LanguageModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.hidden_layer = nn.Linear(3 * embedding_dim, hidden_dim)
        self.output_layer = nn.Linear(hidden_dim, vocab_size)
        self.loss_function = nn.CrossEntropyLoss()

    def forward(self, word_1, word_2, word_3, target=None):
        embedded_1 = self.embedding(word_1)
        embedded_2 = self.embedding(word_2)
```

Implementation in PyTorch II

```
        embedded_3 = self.embedding(word_3)

        hidden = torch.tanh(self.hidden_layer(
            torch.cat(embedded_1, embedded_2, embedded_3)))
        logits = self.output_layer(hidden)

        loss = None
        if target is not None:
            loss = self.loss_function(logits, targets)

        return logits, loss
```

Representing Sequences

Representing Sequences

General Architecture Overview

1.

Embed input words.

Get a sequence of continuous vectors

2.

Contextualize input.

Apply a sequence processing architecture and get contextual representation.

3.

Get some output.

Typically classification or labeling.

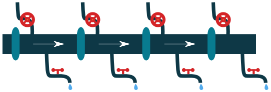
Contextual representation

Word embeddings represent words in isolation.

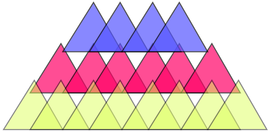
Meaning is context-dependent —

We need an $\boxed{\textbf{encoder}}$ providing $\boxed{\textbf{contextual representation}}$

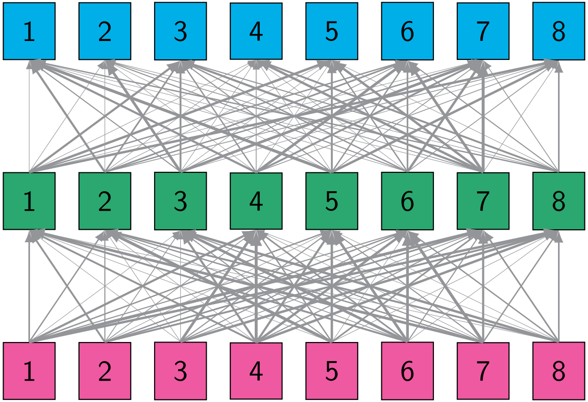
Transformers: Complete graph metaphor



RNN = information pipeline



CNN = information in tree-like data structure

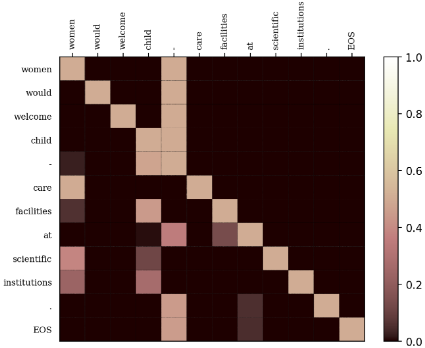


Self-attentive = $\boxed{\textbf{Transformers}}$ =

Information flow in a weighted complete bipartite graph

Transformers

- In some layers: states are linear combination of previous layer states
- Originally for the Transformer model for machine translation



$\Leftarrow$ attention weights $=$ similarity matrix between all pairs of states

- $O(n^2)$ memory, $O(1)$ time (when paralelized)
- next layer: sum by rows

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 6000–6010, Long Beach, CA, USA, December 2017. Curran Associates, Inc

Multi-head scaled dot-product attention

Scaled dot-production attention

$Q = (q_1, \dots, q_n)$: queries, K: keys, V: values

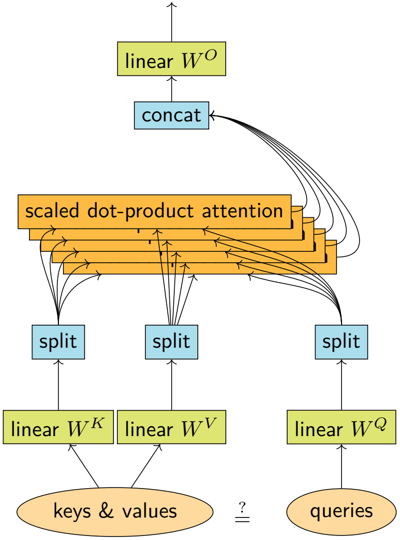
$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\overbrace{\left(\frac{QK^\top}{\sqrt{d}}\right)}^{\text{similarity matrix}} V$$

Multi-head setup

$$\mathrm{Multihead}(Q, V) = \overbrace{(H_1 \oplus \cdots \oplus H_h)}^{\text{concatenate head outputs}} W^O$$

$$H_i = \mathrm{Attn}(QW_i^Q, VW_i^K, VW_i^V)$$

W_i^Q, W_i^K, W_iV head-specific projections

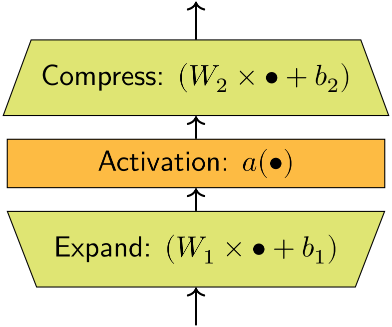


Dot-Product Attention in PyTorch

```
def attention(query, key, value, mask=None):
    d_k = query.size(-1)
    scores = (
        torch.matmul(query, key.transpose(-2, -1)) /
        math.sqrt(d_k))
    p_attn = F.softmax(scores, dim = -1)
    return torch.matmul(p_attn, value), p_attn
```

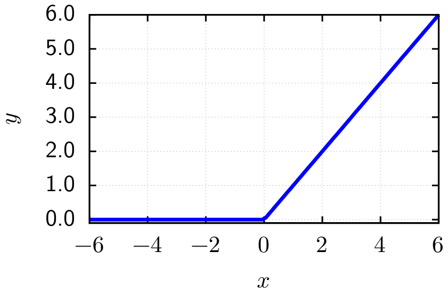
Feed-forward Layer

$$\text{FeedForward}(X) = W_2 \cdot a(W_1 \cdot X + b_1) + b_2$$

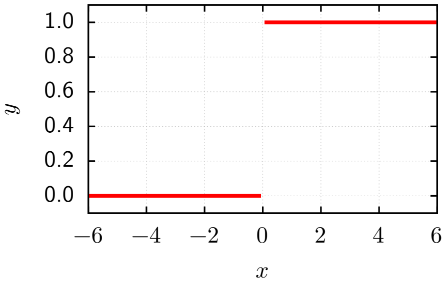


- Applied element-wise
- W_1, W_2 are learned; b_1, b_2 are bias terms
- Upsample to $4\times$ model dimension
  $\Rightarrow$ Many parameters here
- Place to store knowledge about the data

ReLU:



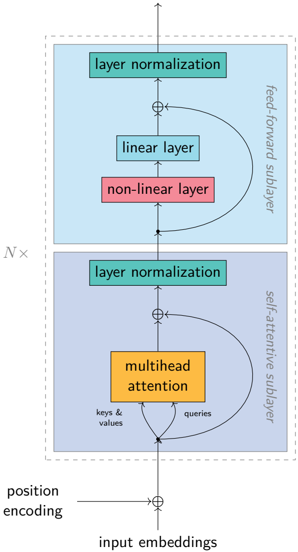
Derivative of ReLU:



faster, suffer less with vanishing gradient

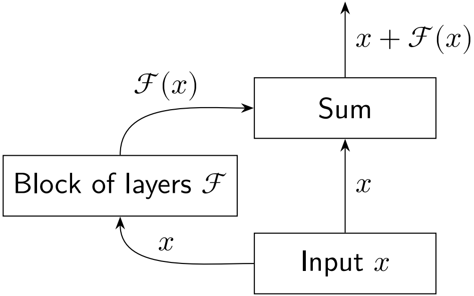
Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, Haifa, Israel, June 2010. JMLR.org

Stacking self-attentive Layers



- several layers (original paper 6)
- each layer 2 sub-layers: self-attention and feed-forward layer
- everything inter-connected with residual connections

Residual Connections



$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

Make sure there is always a path for the gradient to flow through the network.

Because summation is linear w.r.t. the gradient.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE Computer Society. ISBN 9781467388511

Residual Connections: Numerical Stability

Numerically unstable, we need activation to be in similar scale ⇒ layer normalization. Activation before non-linearity is normalized:

$$\overline{a}_i = \frac{g_i}{\sigma_i}(a_i - \mu_i)$$

...g is a trainable parameter, μ, σ estimated from data.

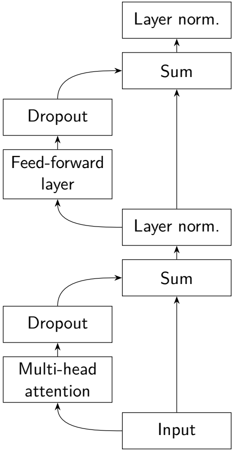
$$\mu = \frac{1}{H}\sum_{i=1}^{H} a_i$$

$$\sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i - \mu)^2}$$

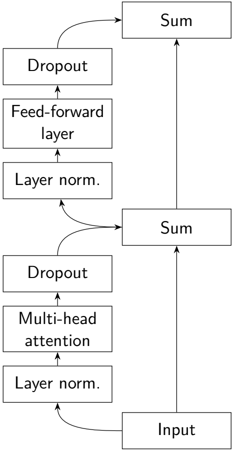
Lei Jimmy Ba, Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. ISSN 2331-8422

Pre- vs. Post-Layer Normalization

Original: Post-normalization



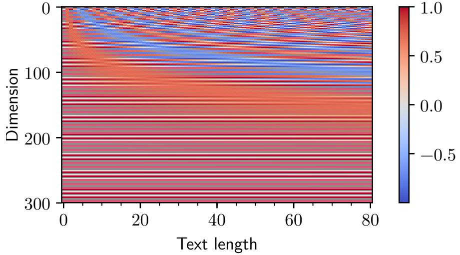
Now more common: Pre-normalization (Xiong et al., 2020)



Position Encoding

Model is not aware of the position in the sequence.

$$\text{pos}(i) = \begin{cases} \sin\left(\frac{t}{10^4}^{\frac{i}{d}}\right), & \text{if } i \mod 2 = 0 \\ \cos\left(\frac{t}{10^4}^{\frac{i-1}{d}}\right), & \text{otherwise} \end{cases}$$



- Just summed with the token embeddings
- More recent alternatives: learned position embeddings (for fixed max length)
- Rotary position embeddings: encode relative distances

Classification and Labeling

Classification and Labeling

General Architecture Overview

1.

Embed input words.

Get a sequence of continuous vectors

2.

Contextualize input.

Apply a sequence processing architecture and get contextual representation.

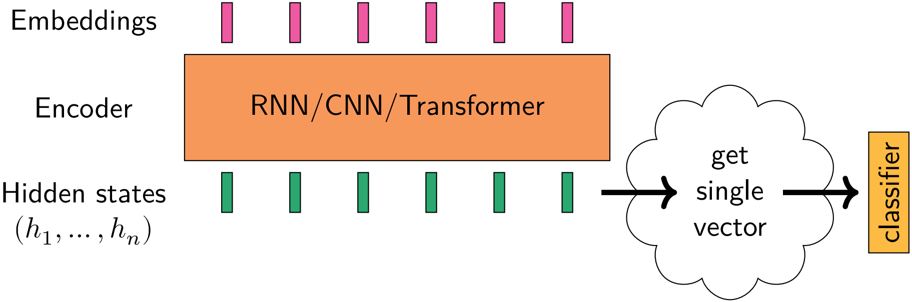
3.

Get some output.

Typically classification or labeling.

Text classification

Tasks such as: sentiment analysis, genre classification, fake news detection, spam detection.



Classifier = feedforward neural net with softmax at the end

Getting a single vector: Two options

1. Pooling

- Squeeze the sequence into a vector
- Usable directly for embeddings and also encoder output
- Mean pooling: $\frac{1}{n}\sum h_i$
- Max pooling (intuitively existential quantifier): take max in each dimension

2. Choose one state

- In RNN the last state, i.e., after reading the whohle input
- In Transformers any (typically the first) state, self-attention will learn to move relevant information there

Softmax & Cross-Entropy

Output layer with softmax (with parameters W, b) — gets categorical distribution:

$$P_y = \mathsf{softmax}(\mathbf{x}) = \mathsf{P}(y = j \mid \mathbf{x}) = \frac{\exp \mathbf{x}^\top W_j + b_j}{\sum \exp \mathbf{x}^\top W + b}$$

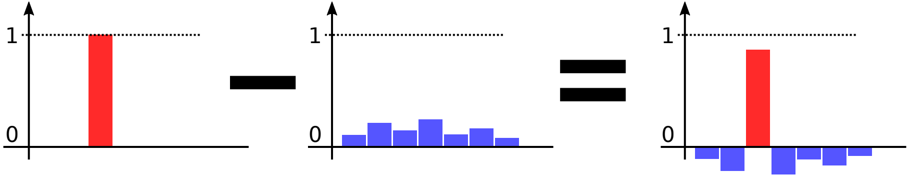
Network error $=$ cross-entropy between estimated distribution and one-hot ground-truth distribution $T = \mathbf{1}(y^*) = (0, 0, \ldots, 1, 0, \ldots, 0)$:

$$\begin{aligned} L(P_y, y^*) = H(P, T) &= -\mathbb{E}_{i\sim T} \log P(i) \\ &= -\sum_i T(i) \log P(i) \\ &= -\log P(y^*) \end{aligned}$$

Derivative of Cross-Entropy

Let $l = \mathbf{x}^\top W + b$, l_{y^*} corresponds to the correct one.

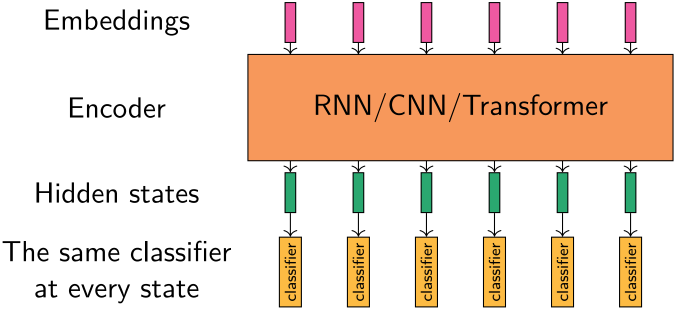
$$
\begin{aligned}
\frac{\partial L(P_y, y^*)}{\partial l} &= -\frac{\partial}{\partial l} \log \frac{\exp l_{y^*}}{\sum_j \exp l_j} = -\frac{\partial}{\partial l} l_{y^*} - \log \sum \exp l \\
&= \mathbf{1}_{y^*} + \frac{\partial}{\partial l} - \log \sum \exp l = \mathbf{1}_{y^*} - \frac{\sum \mathbf{1}_{y^*} \exp l}{\sum \exp l} = \\
&= \mathbf{1}_{y^*} - P_y(y^*)
\end{aligned}
$$



Interpretation: Reinforce the correct logit, suppress the rest.

Sequence Labeling

- assign value / probability distribution to every token in a sequence
- morphological tagging, named-entity recognition, LM with unlimited history, answer span selection



Pre-training Representations

Pre-training Representations

Pre-trained Representations

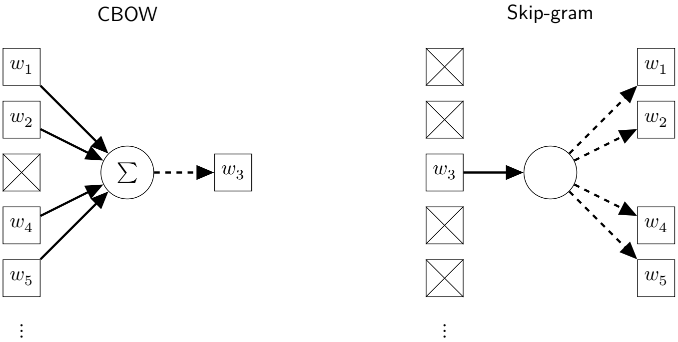
- representations that emerge in models seem to carry a lot of general information about the language
- representations pre-trained on large data can be re-used on tasks with smaller training data

Pre-training Representations

Word2Vec & FastText

Word2Vec

- way to learn word embeddings without training the complete LM



- CBOW: minimize cross-entropy of the middle word of a sliding windows
- skip-gram: minimize cross-entropy of a bag of words around a word (LM other way round)

Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, GA, USA, June 2013. Association for Computational Linguistics

Word2Vec: sampling

1.	**All** human beings are born free and equal in dignity ...	→	(All, humans) (All, beings)
2.	All **human** beings are born free and equal in dignity ...	→	(human, All) (human, beings) (human, are)
3.	All human **beings** are born free and equal in dignity ...	→	(beings, All) (beings, human) (beings, are) (beings, born)
4.	All human beings **are** born free and equal in dignity ...	→	(are, human) (are, beings) (are, born) (are, free)

Word2Vec: Formulas

- Training objective:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j\sim(-c,c)}\log p(w_{t+c}|w_t)$$

- Probability estimation:

$$p(w_O|w_I) = \frac{\exp\left({V'}_{w_O}^{\top} V_{w_I}\right)}{\sum_w \exp\left({V'}_{w}^{\top} V_{w_i}\right)}$$

where V is input (embedding) matrix, V' output matrix

Equations 1, 2. Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, GA, USA, June 2013. Association for Computational Linguistics

Word2Vec: Training using Negative Sampling

The summation in denominator is slow, use noise contrastive estimation:

$$\log \sigma \left({V'}_{w_O}^{\top} V_{w_I} \right) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[\log \sigma \left(-{V'}_{w_i}^{\top} V_{w_I} \right) \right]$$

Main idea: classify independently by logistic regression the positive and few sampled negative examples.

Equations 1, 3. Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, GA, USA, June 2013. Association for Computational Linguistics

Word2Vec: Vector Arithmetics

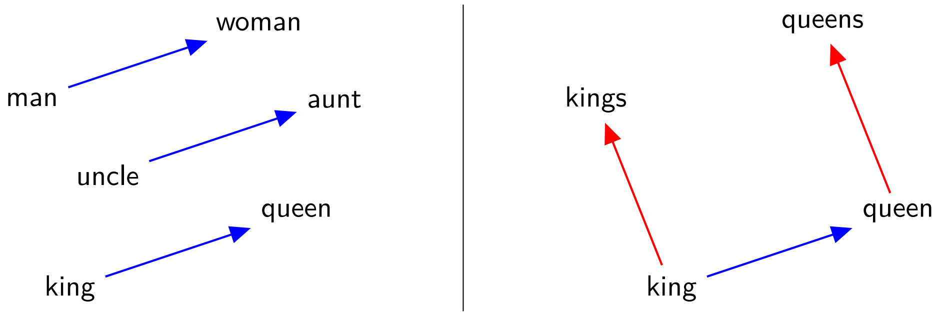


Image originally from Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, GA, USA, June 2013. Association for Computational Linguistics

State-of-the-art pre-trained word embeddings

- Word2Vec training treats words as independent entities
- There are regularities in how words look like — it is called morphology

FastText tackles this

1. Represent each word as a bag of character n-grams
2. Keep a table of character n-grams embeddings instead of words
3. Word embedding $=$ average of character n-gram embeeddings

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051. URL `https://www.aclweb.org/anthology/Q17-1010`

Training models

FastText — model implementation by Facebook

https://github.com/facebookresearch/fastText

Just throw in raw text.

```
./fasttext skipgram -input data.txt -output model
```

The tool allows training simple classifiers using the embeddings.

Pre-training Representations

BERT

What is BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

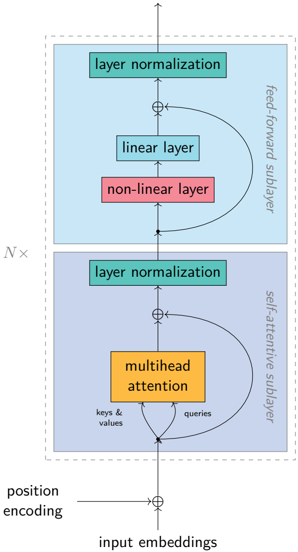
- pre-trained representation capturing context

  contextual embeddings

- Transformer with a masked language model objective
- originally by Google, published in November 2018
- since then better versions: e.g., RoBERTa by Facebook, language-specific variants, multilingual versions

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`

Achitecture



- a stack of 12 Transformer layers
- trained as sequence labeling: change some input token, labeler guesses original tokens

$\boxed{\textbf{masked language modeling}}$

- being able to predict missing words: proxy for language understanding

When trained, throw away labeling, use last layer as the representation.

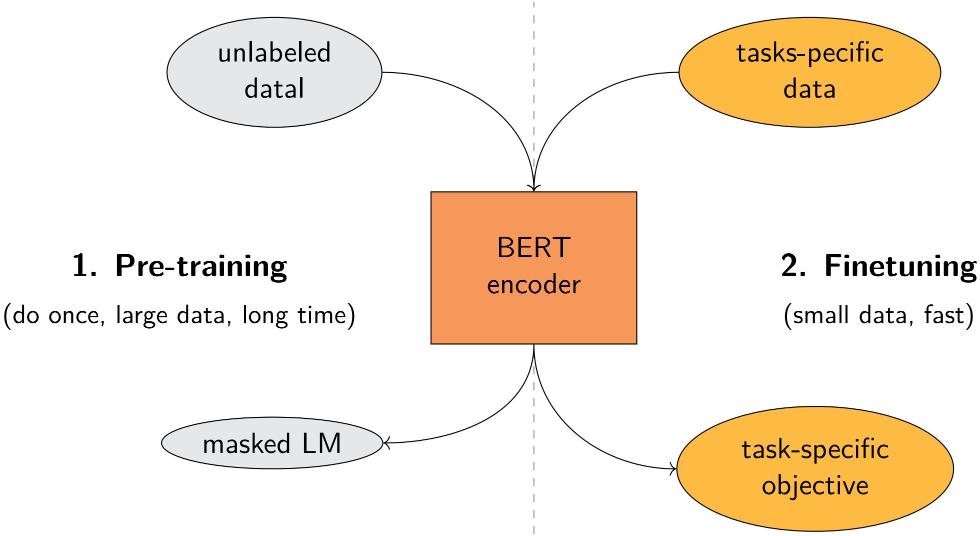
Masked Language Model

All human being are born free fr and A equal i in f dignity and rights

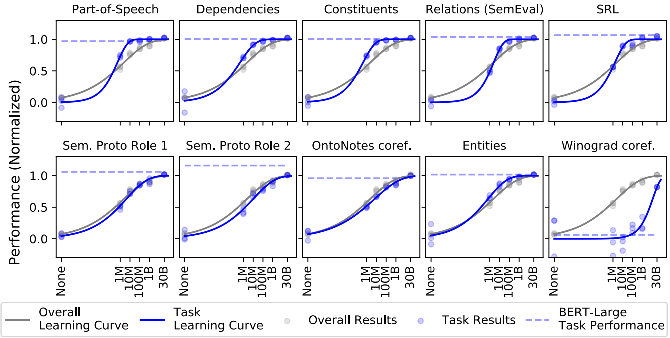
1. Randomly sample a word → free
2. With 80% change replace with special `MASK` token.
3. With 10% change replace with random token → hairy
4. With 10% change keep as is → free

Then a classifier should predict the missing/replaced word free

Pre-train and finetune paradigm



Training data size



- BERT — 3.4B words
- RoBERTa — 30B words

← How much training data is needed to master a task

Source: Yian Zhang, Alex Warstadt, Xiaocheng Li, and Samuel R. Bowman. When do you need billions of words of pretraining data? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1112–1125, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.90. URL `https://aclanthology.org/2021.acl-long.90`

Implementation: Huggingface Transformers

https://github.com/huggingface/transformers

Implements most existign pre-trained BERT-like models for both PyTorch and TensorFlow.

Transformers

Biases from Training Data

Trained on crawled web data $\Rightarrow$ **replicate** biases from the data.

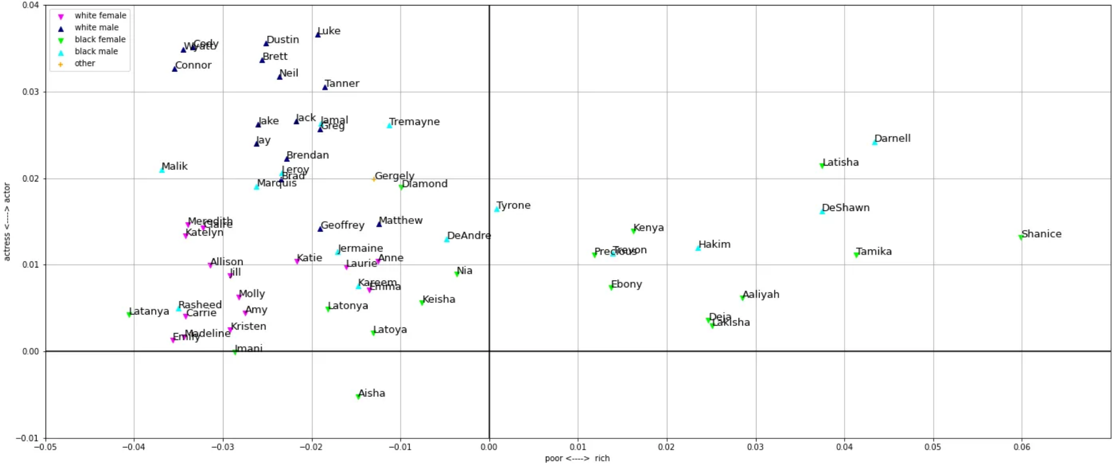
- Web is full of toxic content
- People with extreme opinions tend to write more than others
- Data is not representative of demography

Biases may leak into / influence the downsteram tasks.

Example: Racial Bias

Being black is the same as being poor

`[MASK]` is an actor / actress. `[MASK]` is poor / rich.



Source: `https://towardsdatascience.com/racial-bias-in-bert-c1c77da6b25a`

Today's Learning Outcomes

After this lecture you should be able to ...

1. Describe neural networks as **continuous functions** and takes inputs and generates outputs
2. Describe how neural networks **are trained** and reason about training neural networks with respect to **gradient flow**
3. Statically represent words and tell how to neural networks **represent words in context**
4. Describe **pre-training** of neural networks for NLP

Summary

1. Discrete symbols $\rightarrow$ continuous representation with trained embeddings
2. Architectures to get suitable representation: recurrent, convolutional, self-attentive
3. Output: classification, sequence labeling
4. Representations pretrained on large data helps on downstream tasks

http://ufal.cz/courses/npfl124