Deep Learning

Lecture 9 – Natural Language Processing

Kumar Bipin

BE, MS, PhD (MMMTU, IISc, IIIT-Hyderabad)

Robotics, Computer Vision, Deep Learning, Machine Learning, System Software

Agenda

9.1 Language Models

9.2 Traditional Language Models

9.3 Neural Language Models

9.4 Neural Machine Translation

9.1
Language Models

Language Models

$$\begin{aligned} p(\mathbf{x}) = p(x_1, \ldots, x_T) &= \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \\ &= p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \ldots \end{aligned}$$

Language Model:

- Probability distribution over a sequence of **discrete tokens** $\mathbf{x} = (x_1, \ldots, x_T)$ where each token can take a value from a **vocabulary** $\mathcal{V}$ $(x_t \in \mathcal{V})$
- Decomposes into a sequence via the **chain rule of probability / product rule**
- Depending on the model, a token can be a word, character or byte
- A special <`EOS`> token indicates the end of the sentence ($x_T =$ <`EOS`>)

Language Models

Word Language Model Example:

$$
\begin{aligned}
p(\text{The dog ran away } \texttt{<EOS>}) &= p(\text{The})\, p(\text{dog}|\text{The})\, p(\text{ran}|\text{The dog}) \\
&= p(\text{away}|\text{The dog ran})\, p(\texttt{<EOS>}|\text{The dog ran away})
\end{aligned}
$$

- Word language models are **auto-regressive** models that predict the next word given all previous words in the sentence
- A good model has a high probability of predicting likely next words

Applications

Applications of Language Models

Language Recognition:

Assume two language models over sentences:

$$p(\mathbf{x}) = p(x_1, \ldots, x_T) \qquad p'(\mathbf{x}) = p'(x_1, \ldots, x_T)$$

Where p has been trained on English and p' on French sentences.

We can determine which sentence a language is from by classifying according to:

$$\text{Language}(\mathbf{x}) = \begin{cases} \text{English} & \text{if } p(\mathbf{x}) > p'(\mathbf{x}) \\ \text{French} & \text{otherwise} \end{cases}$$

Applications of Language Models

Generative Model:

Assume a language model over sentences $\mathbf{x}$:

$$p(\mathbf{x}) = p(x_1, \ldots, x_T)$$

Using the **decomposition into conditional distributions**

$$p(x_1, \ldots, x_T) = p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \ldots$$

we can efficiently **sample new sentences** from the model distribution.

Applications of Language Models

Bayesian Inference: (Machine Translation)

Assume a **prior** over sentences $\mathbf{x}$ in the form of a language model:

$$p(\mathbf{x}) = p(x_1, \ldots, x_T)$$

Assume a **likelihood** that tells us how likely sentence $\mathbf{x}$ translates to sentence $\mathbf{y}$:

$$p(\mathbf{y}|\mathbf{x}) = p(y_1, \ldots, y_{T'}|x_1, \ldots, x_T)$$

We can use Bayes rule to infer the **posterior** over translated sentences:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})\, p(\mathbf{x})}{p(\mathbf{y})}$$

Training

Learning Language Models

Let $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$ denote a training set with sentences $\mathbf{x}_i = \{x_1^i, \ldots, x_{T_i}^i\}$.
We train the unconditional language model $p_{model}(\mathbf{x}_i|\mathbf{w})$ via **maximum likelihood:**

$$\begin{aligned}\hat{\mathbf{w}}_{ML} &= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^N p_{model}(\mathbf{x}_i|\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^N \log p_{model}(\mathbf{x}_i|\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \ -\mathbb{E}_{p_{data}}\left[\log p_{model}(\mathbf{x}|\mathbf{w})\right]\end{aligned}$$

We **minimize the cross entropy** between the data and the model distribution.

Remark: We now use $p_{model}(\mathbf{x}|\mathbf{w})$ to distinguish the model from the data distribution.

Evaluation

Evaluating Character Language Models

Character language models typically measure performance in bits per character.

Shannon Information:

Given a character sequence $\mathbf{x}$ of length T with probability $p(\mathbf{x})$, the **surprise** (= shannon information) normalized by the sequence length T is the normalized negative log-likelihood of $\mathbf{x}$:

$$I(\mathbf{x}) = -\frac{1}{T}\log_2 p(\mathbf{x}) = -\frac{1}{T}\sum_{t=1}^{T}\log_2 p(x_t|x_1,\ldots,x_{t-1}) \quad [bits]$$

Cross Entropy:

The **expected surprise** of the model under the data distribution (for sequences of length T) is thus given by the (normalized) cross entropy:

$$H(p_{data}, p_{model}) = \mathbb{E}_{p_{data}}\left[-\frac{1}{T}\log_2 p_{model}(\mathbf{x})\right]$$

Evaluating Character Language Models

Cross Entropy:

Let us now consider sequences of arbitrary length by taking the limit $T \to \infty$.

By the Shannon-McMillan-Breiman theorem, the cross-entropy simplifies further

$$\begin{aligned} H(p_{data}, p_{model}) &= \mathbb{E}_{p_{data}} \left[-\frac{1}{T} \log_2 p_{model}(\mathbf{x}) \right] \\ &\approx -\frac{1}{T} \log_2 p_{model}(\mathbf{x}) \quad [bits] \end{aligned}$$

as each sequence occurs in proportion to its probability anyways if we consider long enough sequences (think of an infinite text $\mathbf{x}$ generated from the data distribution).

Remark: In practice, $H(p_{data}, p_{model})$ is evaluated on a test or validation sequence $\mathbf{x}$.

Evaluating Character Language Models

Example 1:

Consider a vocabulary $\mathcal{V} = \{A, B\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t) = \frac{1}{2}$ for both data and model distribution.

Then

$$H(p_{data}, p_{model}) = -\frac{1}{10} \log_2 \left(\frac{1}{2}\right)^{10} = \log_2 2 = 1 \; bit$$

The amount of information needed to predict the next character is 1 bit with this simple (unigram) model as the next character is either A or B with equal probability. In other words, we can't find a better coding of this language than using 1 bit per character.
Remark: A uniform distribution maximizes the entropy ($\Rightarrow$ upper bound for $|\mathcal{V}|$).

Evaluating Character Language Models

Example 2:

Consider a vocabulary $\mathcal{V} = \{A, B\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 1$ and $p(x_t = B) = 0$
for both data and model distribution.

Then

$$H(p_{data}, p_{model}) = -\frac{1}{10} \log_2 1^{10} = \log_2 1 = 0 \; bits$$

In this case, the amount of information needed to predict the next character is 0 bits as the next character is always A. In other words, we don't need any capacity to transmit this language through some channel, it contains no information.
Remark: 0 bits is the minimal value for the entropy or cross-entropy ($\Rightarrow$ lower bound).

Evaluating Character Language Models

Example 3:

Consider a vocabulary $\mathcal{V} = \{A, B\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 0.1$ and $p(x_t = B) = 0.9$
for both data and model distribution.

Then

$$H(p_{data}, p_{model}) = -\frac{1}{10} \log_2 \left(\frac{1}{10}\right)^1 \left(\frac{9}{10}\right)^9 = 0.47\ bits$$

We need 0.47 bits now as we sometimes observe A, but most often B. Thus, the information conveyed in this language is larger than 0 bits and smaller than 1 bit.

Evaluating Character Language Models

Example 4:

Consider a vocabulary $\mathcal{V} = \{A, B\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 0.1$ and $p(x_t = B) = 0.9$ for the model distribution and $p(x_t = A) = 1$ for the data distribution.

Then

$$H(p_{data}, p_{model}) = -\frac{1}{10} \log_2 \left(\frac{1}{10}\right)^{10} = \log_2 10 = 3.32\ bits$$

We need more than 1 bit now as the model fits the data badly. We need 0 bits to encode any possible outcome of p_{data} using the code optimized for p_{data} and 3.32 bits to encode any possible outcome of p_{data} using the code optimized for p_{model}:

$$H(p_{data}, p_{model}) = \underbrace{H(p_{data})}_{\geq 0} + \underbrace{D_{KL}(p_{data} || p_{model})}_{\geq 0}$$

Evaluating Word Language Models

It would be natural to measure **word language models** in bits per word.

However, word language models are traditionally measured in **perplexity:**

$$\begin{aligned}\text{Perplexity}(p_{data}, p_{model}) &= 2^{H(p_{data}, p_{model})} \\ &\approx 2^{-\frac{1}{T}\log_2 p_{model}(\mathbf{x})} \\ &= p_{model}(\mathbf{x})^{-\frac{1}{T}} \\ &= \left(\prod_{t=1}^{T} p_{model}(x_t | x_1, \ldots, x_{t-1})\right)^{-\frac{1}{T}}\end{aligned}$$

Thus, perplexity can be interpreted as the **inverse probability of the test set**, normalized by the sequence length T (geometric mean).

Again, $\text{Perplexity}(p_{data}, p_{model})$ is evaluated on a test or validation sequence $\mathbf{x}$.

Evaluating Word Language Models

Example 1:

Consider a vocabulary $\mathcal{V} = \{A, B, C\}$ and sequences of length $T = 10$.

Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t) = \frac{1}{3}$ for both data and model distribution.

Then

$$\text{Perplexity}(p_{data}, p_{model}) = \left(\left(\frac{1}{3}\right)^{10}\right)^{-\frac{1}{10}} = 3$$

We see that the perplexity models the number of possible next tokens to choose from (i.e., here the model is maximally confused which of the 3 tokens A, B or C to pick).

Thus, perplexity is often also called the **average weighted branching factor.**

Remark: A uniform distribution maximizes the perplexity ($\Rightarrow$ upper bound for $|\mathcal{V}|$).

Evaluating Word Language Models

Example 2:

Consider a vocabulary $\mathcal{V} = \{A, B, C\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 1$ and $p(x_t \in \{B, C\}) = 0$,
for both data and model distribution.

Then

$$\text{Perplexity}(p_{data}, p_{model}) = \left(1^{10}\right)^{-\frac{1}{10}} = 1$$

We see that the perplexity reduces in this case as the next choice is certain. The model is not surprised to see the test set as it is able to predict the test set exactly (all A's).
Remark: 1 is the minimal value for the perplexity measure ($\Rightarrow$ lower bound). However, this is only achievable for languages that contain only a single token.

Evaluating Word Language Models

Example 3:

Consider a vocabulary $\mathcal{V} = \{A, B, C\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 0.1$, $p(x_t = B) = 0.9$, and $p(x_t = C) = 0$ for both data and model distribution.

Then

$$\text{Perplexity}(p_{data}, p_{model}) = \left(\left(\frac{1}{10}\right)^1 \left(\frac{9}{10}\right)^9\right)^{-\frac{1}{10}} = 1.38$$

In this case, the perplexity is slightly larger than 1 as the model is quite certain to predict B as the next character, but sometimes it should predict A.

Evaluating Word Language Models

Example 4:

Consider a vocabulary $\mathcal{V} = \{A, B, C\}$ and sequences of length $T = 10$.
Assume $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$ with $p(x_t = A) = 0.1$, $p(x_t = B) = 0.9$, and $p(x_t = C) = 0$ for the model distribution and $p(x_t = A) = 1$ for the data distribution.

Then

$$\text{Perplexity}(p_{data}, p_{model}) = \left(\left(\frac{1}{10}\right)^{10}\right)^{-\frac{1}{10}} = 10$$

In this case the perplexity is larger than 3 as the model fits the data badly.

Evaluating Language Models

- Shannon estimated that English text has $0.6 - 1.3$ bits per character
- For character language models, current performance is roughly 1 bit per character
- For word language models, perplexities of about 60 were typical until 2017
- According to Quora, there are 4.79 letters per word (excluding spaces)
- Assuming 1 bit per character, we have a perplexity of $2^{5.79} = 55.3$
- State-of-the-art models (GPT-2, Megatron-LM) yield perplexities of $10 - 20$
- Be careful: Metrics not comparable across vocabularies or datasets

Additional Resources:

`https://thegradient.pub/understanding-evaluation-metrics-for-language-models/`

`https://towardsdatascience.com/`

`the-relationship-between-perplexity-and-entropy-in-nlp-f81888775ccc`

9.2
Traditional Language Models

Language Models

$$
\begin{aligned}
p(\mathbf{x}) = p(x_1, \dots, x_T) &= \prod_{t=1}^{T} p(x_t | x_1, \dots, x_{t-1}) \\
&= p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \dots
\end{aligned}
$$

Language Model:

- Probability distribution over a sequence of **discrete tokens** $\mathbf{x} = (x_1, \dots, x_T)$ where each token can take a value from a **vocabulary** $\mathcal{V}$ $(x_t \in \mathcal{V})$
- Decomposes into a sequence of conditional distributions
- The history (conditioning variables/tokens) is called **context** in NLP

Language Models

$$\begin{aligned} p(\mathbf{x}) = p(x_1, \ldots, x_T) &= \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \\ &= p(x_1)\, p(x_2|x_1)\, p(x_3|x_1, x_2) \ldots \end{aligned}$$

Language Model:

- We can represent each conditional distribution with a **probability table** and learn the entries of these tables, but this becomes intractable as T grows
- The probability table for $p(x_T|x_1, \ldots, x_{T-1})$ comprises $|\mathcal{V}|^T$ entries where the vocabulary size $|\mathcal{V}|$ of word language models is roughly $|\mathcal{V}| \approx 30.000$
- Huge **memory** and **training sets** needed (many long sentences are very rare)

n-gram Models

$$p(\mathbf{x}) = p(x_1, \ldots, x_T) = p(x_1, \ldots, x_{n-1}) \prod_{t=n}^{T} p(x_t | x_{t-n+1}, \ldots, x_{t-1})$$

n-gram Models:

- An n-gram is a sequence of n tokens (e.g., “The dog”, “dog ran”, “ran away”)
- n-gram models **approximate the history context** by the last $n-1$ tokens
- In other words, they make a **Markov assumption** (the model is **memoryless**)
- This idea is similar to the autoregressive models that we have introduced, except that each conditional $p(x_t | x_{t-n+1}, \ldots, x_{t-1})$ is represented by a probability table
- Early language models considered bigrams $(n = 2)$ and trigrams $(n = 3)$

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

n-gram Models

Word Language Model Examples:

- **bigram:** ($n = 2$, history length $= 1$)

$$p(x_1, x_2, x_3, x_4) = p(x_1)\, p(x_2|x_1)\, p(x_3|x_2)\, p(x_4|x_3)$$

$$p(\text{The dog ran away}) = p(\text{The})\, p(\text{dog}|\text{The})\, p(\text{ran}|\text{dog})\, p(\text{away}|\text{ran})$$

- **trigram:** ($n = 3$, history length $= 2$)

$$p(x_1, x_2, x_3, x_4) = p(x_1, x_2)\, p(x_3|x_1, x_2)\, p(x_4|x_2, x_3)$$

$$p(\text{The dog ran away}) = p(\text{The dot})\, p(\text{ran}|\text{The dog})\, p(\text{away}|\text{dog ran})$$

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

Training of n-gram Models

The **conditional probability** can be written as:

$$p(x_t|x_{t-n+1},\ldots,x_{t-1}) = \frac{p(x_{t-n+1},\ldots,x_t)}{p(x_{t-n+1},\ldots,x_{t-1})} = \frac{p(x_{t-n+1},\ldots,x_t)}{\sum_{x_t} p(x_{t-n+1},\ldots,x_t)}$$

For a **bigram** model this would yield:

$$p(x_t|x_{t-1}) = \frac{p(x_{t-1},x_t)}{p(x_{t-1})} = \frac{p(x_{t-1},x_t)}{\sum_{x_t} p(x_{t-1},x_t)}$$

We see that we simply need to **count** the number of n-grams and (n-1)-grams in the training set to populate the probability table of the n-gram model.

Smoothing: For large n, the n-gram probabilities are often zero as they haven't been observed in the training set. A simple heuristic is to add one to all n-gram counts.

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

Sampling from n-gram Models

Example of a random sentence drawn from a Jane Austen trigram model:

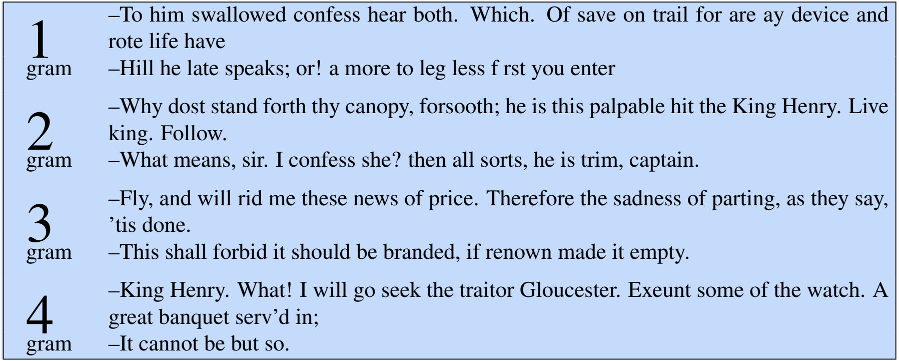
“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.”

- As n-gram models are autoregressive models, **sampling is easy**
- We just need to iteratively draw tokens from $p(x_t|x_{t-n+1},\dots,x_{t-1})$ until we reach the end of sentence symbol $<\texttt{EOS}>$

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

Sampling from n-gram Models

Samples from a Shakespeare language model:



1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

`https://web.stanford.edu/~jurafsky/slp3/`

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

Summary

- n-gram models are simple models that make a **Markov assumption** to model a distribution over sequences via probability tables
- However, they have **limited history context** and **parameters grow exponentially**
- **Smoothing heuristics** are required to deal with the resulting sparsity
- They can't directly model conditional distributions (e.g., for translating sentences)
- In contrast to neural language models, they can be considered as **local non-parametric** predictors (thus suffering the curse of dimensionality)
- Tokens are encoded as **discrete items**
  - Large vocabularies are typically reduced to a shortlist (removing infrequent words)
  - Any two words have the same distance ($\sqrt{2}$ in one-hot vector space)
  - Thus, n-gram models **can't share information** between related (=close) words

Shannon: A mathematical theory of communication. The Bell System Technical Journal, 1948.

9.3
Neural Language Models

Local Word Representations

- **n-gram** models (probability tables) suffer from the **curse of dimensionality**
- Modeling the joint distribution of $n = 10$ consecutive words with a vocabulary of size $|\mathcal{V}| = 10000$ results in intractable $|\mathcal{V}|^n = 10^{40}$ parameters
- When modeling continuous variables, we obtain generalization more easily because the function to be learned is expected behave **locally smoothly**
- But discrete word representations assume the same **distance between words:**

$$\left\| \underbrace{(1, 0, 0, \dots, 0)^\top}_{\mathbf{w}_1} - \underbrace{(0, 1, 0, \dots, 0)^\top}_{\mathbf{w}_2} \right\|_2 = \sqrt{2}$$

- Remark: In this unit we use $\mathbf{w}$ to denote a one-hot encoding of a word

Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Word Representations

However, **some words are more similar than others**. Consider the sentences:

- “The cat is walking in the bedroom”
- “The dog was running in a room”

Having seen the first sentence (in the training set) should help to generalize making the second sentence very likely as

- “cat” and “dog”,
- “walking” and “running”,
- “the” and “a”, . . .

have similar semantic roles.

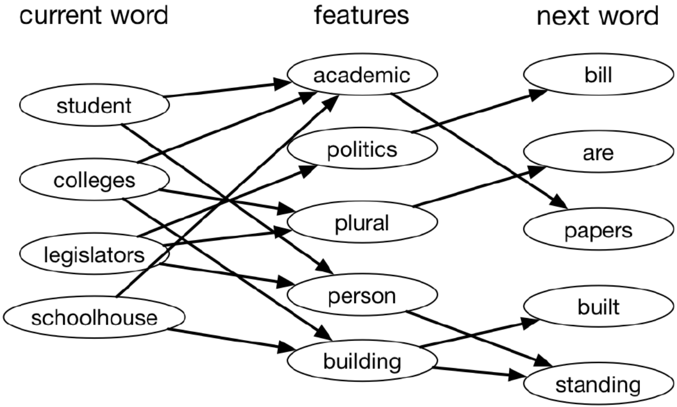
Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Distributed Word Representations

- **Distributed representations** map one-hot vectors $\{0,1\}^{|\mathcal{V}|}$ of high dimensions (e.g., $|\mathcal{V}| = 10000$) to word embeddings $\mathbb{R}^M$ of lower dimensions (e.g., $M = 30$)
- Thus, they distribute the representation along all dimensions (of the embedding vector) and are able to **model similarity between words**
- They distribute probability mass where it matters rather than uniformly in all directions around each training point
- They therefore allow **generalization across sentences**
- Word embeddings are often realized through a simple **linear mapping** which can then be further processed by a feedforward or recurrent neural network

Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Distributed Word Representations



Remark: Unlike in this illustration, we do not attach actual labels to words

Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Word Representations

Example: Assume a training set with the following 3 sentences:

- “The cat is walking”
- “The dog is walking”
- “The cat is sitting”

A distributed representation learns to embed “cat” and “dog” nearby.

Given that p(The cat is sitting) is high, and “cat” and “dog” are related

- “The dog is sitting”

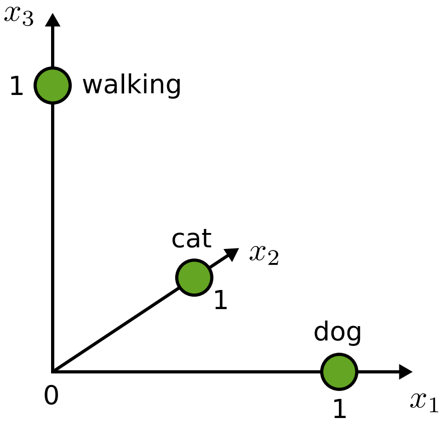
is also likely despite not being part of the training set.

A n-gram model can’t generalize in this setting.

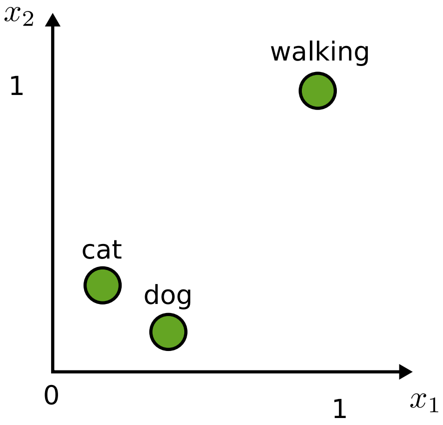
Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Local vs. Distributed Word Representations

Local Representation

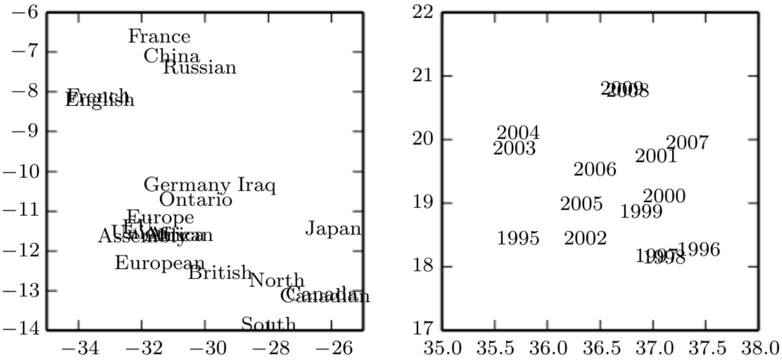


Distributed Representation



Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Learned Word Embeddings



Two-dimensional t-SNE visualization of a word embedding model.

Be careful with 2D visualizations: “In a 30-dimensional grocery store, anchovies can be next to fish and next to pizza toppings.” – Geoffrey Hinton

Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Neural Probabilistic Language Model

First language model that used a distributed representation which have been used before in connectionism (Hinton, Elman) and symbolic reasoning (Paccanaro).

Key Ideas:

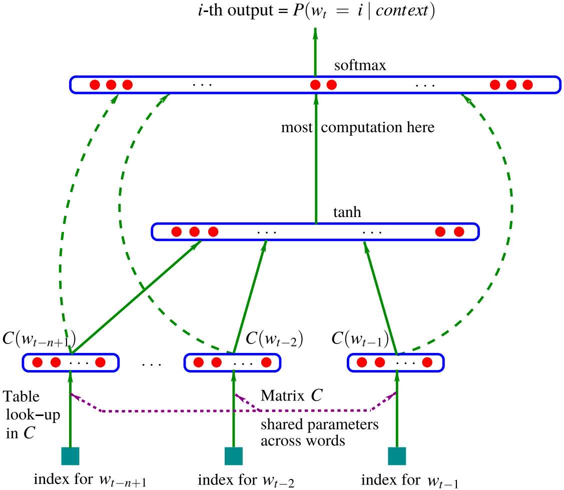
- Associate with each word in the vocabulary a distributed **word feature vector** (a real-valued vector in $\mathbb{R}^M$)
- Express the **joint probability function** of word sequences in terms of the feature vectors of these words in the sequence, and
- Learn simultaneously the **word feature vectors** and the parameters of that **probability function**

Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Neural Probabilistic Language Model

Feedforward Model:

- Input: Sequence of words
- Output: Prob. of next word
- 3 Layers:
  1. Fully connected ⇒ embedding
  2. Fully connected + tanh
  3. Fully connected + softmax
- Input to 2nd layer: vector of concatenated word embeddings
- Optional: direct connections



Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Neural Probabilistic Language Model

Feedforward Model:

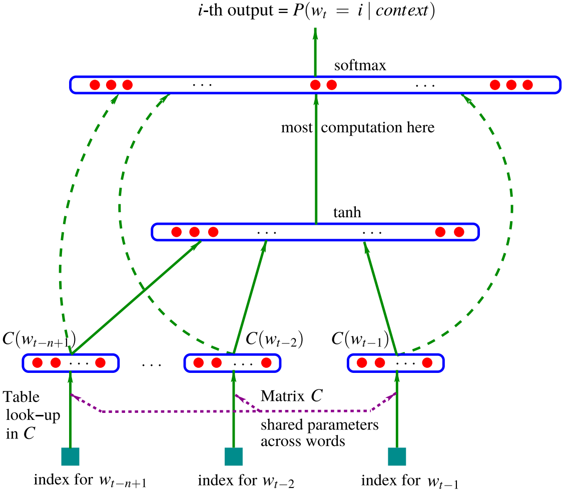
$$P(\mathbf{w}_t|\mathbf{w}_{t-1},\ldots,\mathbf{w}_{t-n+1}) = \frac{e^{y_{\mathsf{idx}(\mathbf{w}_t)}}}{\sum_i e^{y_i}}$$

$$\mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x} + \mathbf{U}\tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$$

$$\mathbf{x} = (\mathbf{C}\,\mathbf{w}_{t-n+1},\ldots,\mathbf{C}\,\mathbf{w}_{t-1})$$

where $\mathbf{C} \in \mathbb{R}^{M\times|\mathcal{V}|}$. Thus the model

- .. scales linearly with $|\mathcal{V}|$
- .. scales linearly with n



Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Neural Probabilistic Language Model

- The **main result** is that significantly better results can be obtained when using the neural network, in comparison with the best of the n-grams, with a test perplexity difference of about 24% on Brown and about 8% on AP News, when taking the MLP versus the n-gram that worked best on the validation set.
- The results also suggests that the neural network was able to take advantage of more context (on Brown, going from 2 words of context to 4 words brought improvements to the neural network, not to the n-grams).
- Also showed that the hidden units are useful (MLP3 vs MLP1 and MLP4 vs MLP2), and that mixing the output probabilities of the neural network with the interpolated trigram always helps to reduce perplexity.

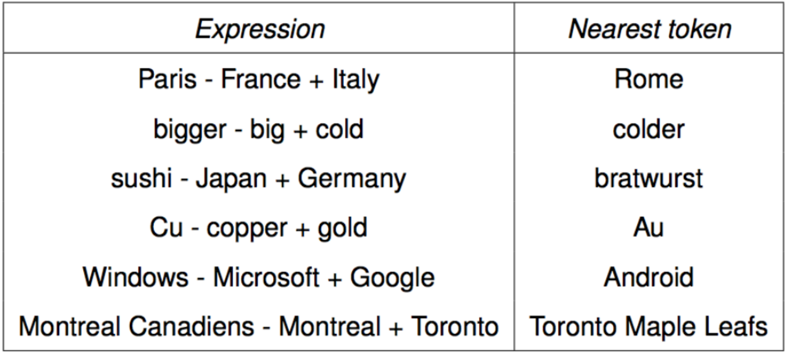
Bengio, Ducharme, Vincent and Janvin: A Neural Probabilistic Language Model. JMLR, 2003.

Word2Vec / Skip-Grams

- Fitting language models is hard (large $|\mathcal{V}| \Rightarrow$ **large softmax**)
- **Skip-grams** predict a word in their surrounding context
- Instead of predicting a distribution over words,
  switch to a **binary prediction problem**
- The model is given pairs of words and needs to distinguish if the words occur next to each other in the training corpus or they are sampled randomly
- Logistic regression on inner product of word embeddings
- Can be **trained very efficiently** with lots of data

Mikolov, Chen, Corrado and Dean: Efficient Estimation of Word Representations in Vector Space. ICLR Workshops, 2013.

Word Vector Arithmetics



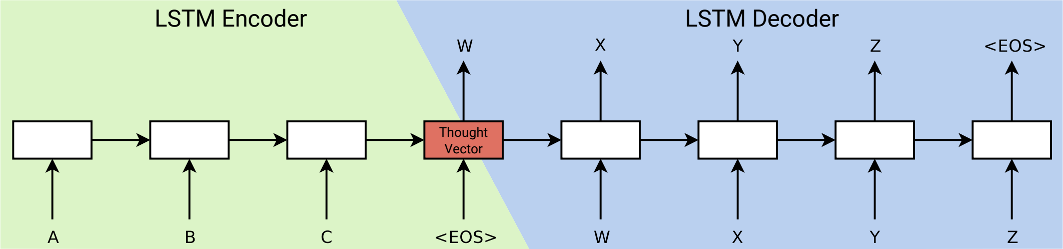
Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Mikolov, Chen, Corrado and Dean: Efficient Estimation of Word Representations in Vector Space. ICLR Workshops, 2013.

9.4

Neural Machine Translation

Sequence to Sequence Learning



- **Two 4-Layer LSTMs** for encoding/decoding the source/target sentence
- Encoding operates in **reverse order** to introduce short-term dependencies
- Intermediate representation produced by the encoder is called **thought vector**
- Encoding using 1000 dim. word embeddings, decoding via **beam search**
- First end-to-end system that outperforms rule-based models ⇒ deployment

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014.

Decoding

Let $\mathbf{w}_1, \ldots, \mathbf{w}_T$ denote the target sentence and let $\mathbf{v}$ denote the thought vector.

Sampling a translation from the LSTM decoder is simple:

$$\mathbf{w}_t \sim p(\mathbf{w}_t | \mathbf{v}, \mathbf{w}_1, \ldots, \mathbf{w}_{t-1})$$

But often we like to compute the **most probable translation:**

$$\mathbf{w}_1, \ldots, \mathbf{w}_T = \underset{\mathbf{w}_1, \ldots, \mathbf{w}_T}{\operatorname{argmax}} \; p(\mathbf{w}_1, \ldots, \mathbf{w}_T | \mathbf{v})$$

This is costly, but a **greedy algorithm** often works well in practice:

$$\mathbf{w}_t = \underset{\mathbf{w}_t}{\operatorname{argmax}} \; p(\mathbf{w}_t | \mathbf{v}, \mathbf{w}_1, \ldots, \mathbf{w}_{t-1})$$

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014.

Beam Search

Failure of Greedy Algorithm:

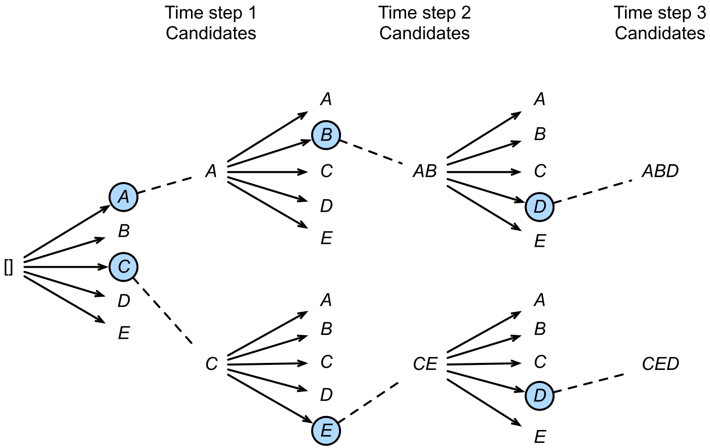
- p(Apples are good) $>$ p(Those apples are good)
- p(Those) $>$ p(Apples)

Idea of Beam Search:

- At each time step, **maintain a list** of the K best words and hidden vectors
- This can be used to produce a list of K best decodings for the following word, which can then be compared to select the most likely one

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014.

Beam Search



https://d2l.ai/chapter_recurrent-modern/beam-search.html

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014.

The Transformer

- **Attention based** model which doesn’t rely on recurrence or convolution
- Process all tokens in **parallel** (not sequentially as in an RNN)
- Leads to significant speed-ups when using modern GPU clusters
- **Self-attention** relates all tokens in a layer with each other
- Thus can more easily capture **long-distance dependencies** compared to an RNN
- Transformer-like architectures have now replaced RNNs in NLP applications, **Defacto standard** for all state-of-the-art models (e.g., on SuperGLUE benchmark)

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

The Transformer

- Each **layer** in the Transformer has shape $L[T, J]$ where t ranges over the position in the input sequence and j ranges over features at that position
- When processing sentences of words, T is the sentence length
- This is the same shape as in an RNN – a sequence of vectors $L[t, J]$
- However, unlike in RNNs, in the Transformer we can compute the layer $L_{\ell+1}[T, J]$ from $L_{\ell}[T, J]$ in **parallel**
- In this respect, the transformer is more similar to a CNN than to an RNN

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

Self-Attention

- The fundamental innovation of the Transformer is the **self-attention layer**
- For each position t in the sequence we compute an attention over the other positions in the sequence
- The transformer uses **multiple heads,** i.e., it computes the attention operation multiple times ($K = 8$ in the original implementation)
- **Self-attention** then constructs a tensor $A[k, t_1, t_2]$ – the strength of the attention weight from t_1 to t_2 for head k
- In the paper, an **embedding dimension** of $D_J = 512$ is chosen per token
- The authors use $K = 8$ heads and a dimension of $D_Q = D_K = D_V = 64$ for the query, key and value embeddings that are used for each token (can be different)

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

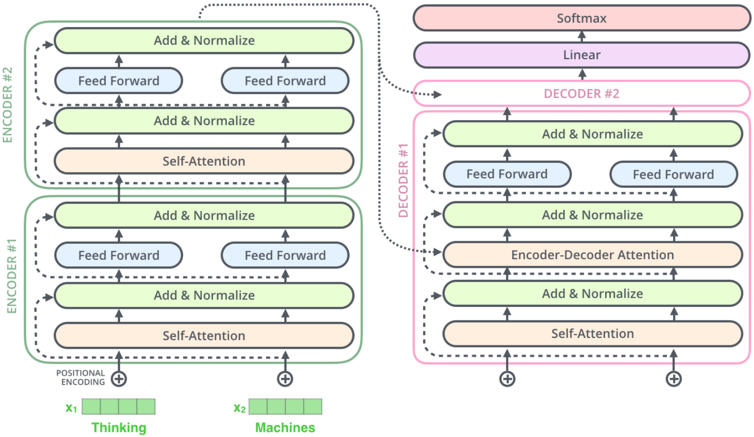
Multi-Headed Self-Attention

For each head k and position t, we compute a **key**, **query** and **value** vector. The querys Q and the keys K are used to compute the self-attention matrix A for head k. A is then multiplied with the values V to yield embedding vectors H that are concatenated.

$$
\begin{aligned}
Q_{\ell+1}[k,t,i] &= W^{Q}_{l+1}[k,i,J]\, L_{\ell}[t,J] \\
K_{\ell+1}[k,t,i] &= W^{K}_{l+1}[k,i,J]\, L_{\ell}[t,J] \\
V_{\ell+1}[k,t,i] &= W^{V}_{l+1}[k,i,J]\, L_{\ell}[t,J] \\
A_{\ell+1}[k,t_1,t_2] &= \underset{t_2}{\mathrm{softmax}} \left[\frac{1}{\sqrt{D_Q}} Q_{\ell+1}[k,t_1,I]\, K_{\ell+1}[k,t_2,I] \right] \\
H_{\ell+1}[k,t,i] &= A_{\ell+1}[k,t,T]\, V_{\ell+1}[k,T,i] \\
L_{\ell+1}[t,j] &= W^{L}_{\ell+1}[j,I]\, (H_{\ell+1}[1,t,I], \ldots, H_{\ell+1}[K,t,I])
\end{aligned}
$$

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

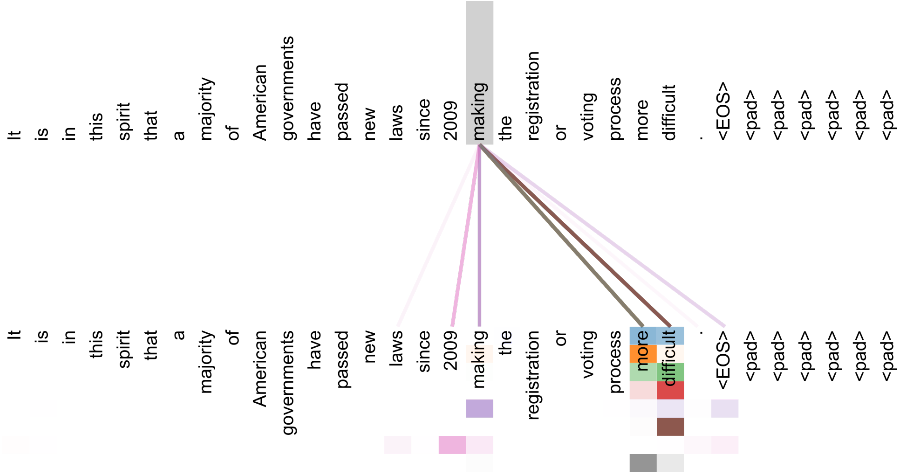
The Transformer



http://jalammar.github.io/illustrated-transformer/

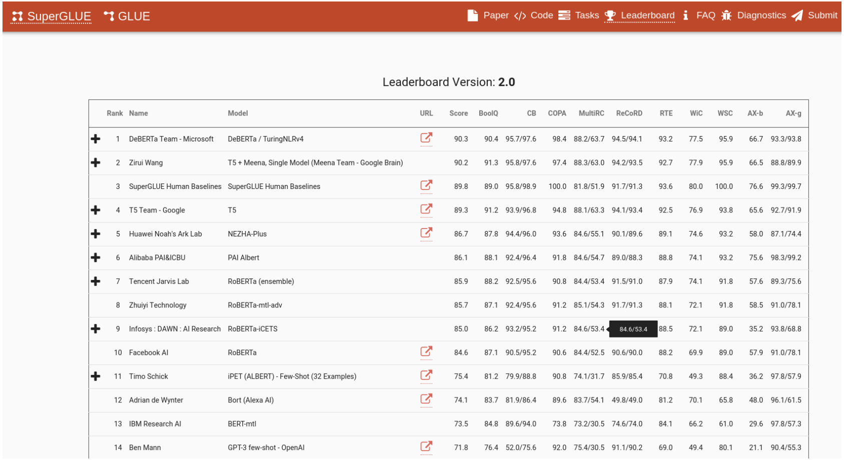
Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

Self-Attention



Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

SuperGLUE



Leaderboard Version: **2.0**

	Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
+	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4	↗	90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
+	2	Zirui Wang	T5 + Meena, Single Model (Meena Team - Google Brain)		90.2	91.3	95.8/97.6	97.4	88.3/63.0	94.2/93.5	92.7	77.9	95.9	66.5	88.8/89.9
	3	SuperGLUE Human Baselines	SuperGLUE Human Baselines	↗	89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	4	T5 Team - Google	T5	↗	89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
+	5	Huawei Noah's Ark Lab	NEZHA-Plus	↗	86.7	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	58.0	87.1/74.4
+	6	Alibaba PAI&ICBU	PAI Albert		86.1	88.1	92.4/96.4	91.8	84.6/54.7	89.0/88.3	88.8	74.1	93.2	75.6	98.3/99.2
+	7	Tencent Jarvis Lab	RoBERTa (ensemble)		85.9	88.2	92.5/95.6	90.8	84.4/53.4	91.5/91.0	87.9	74.1	91.8	57.6	89.3/75.6
	8	Zhuiyi Technology	RoBERTa-mtl-adv		85.7	87.1	92.4/95.6	91.2	85.1/54.3	91.7/91.3	88.1	72.1	91.8	58.5	91.0/78.1
+	9	Infosys : DAWN : AI Research	RoBERTa-iCETS		85.0	86.2	93.2/95.2	91.2	84.6/53.4	84.6/53.4	88.5	72.1	89.0	35.2	93.8/68.8
	10	Facebook AI	RoBERTa	↗	84.6	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	57.9	91.0/78.1
+	11	Timo Schick	iPET (ALBERT) - Few-Shot (32 Examples)	↗	75.4	81.2	79.9/88.8	90.8	74.1/31.7	85.9/85.4	70.8	49.3	88.4	36.2	97.8/57.9
	12	Adrian de Wynter	Bort (Alexa AI)	↗	74.1	83.7	81.9/86.4	89.6	83.7/54.1	49.8/49.0	81.2	70.1	65.8	48.0	96.1/61.5
	13	IBM Research AI	BERT-mtl		73.5	84.8	89.6/94.0	73.8	73.2/30.5	74.6/74.0	84.1	66.2	61.0	29.6	97.8/57.3
	14	Ben Mann	GPT-3 few-shot - OpenAI	↗	71.8	76.4	52.0/75.6	92.0	75.4/30.5	91.1/90.2	69.0	49.4	80.1	21.1	90.4/55.3

`https://super.gluebenchmark.com/leaderboard`

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

GPT-2

Uses a Transformer model with unsupervised pre-training and supervised fine-tuning.

Continue from:
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

`https://openai.com/blog/better-language-models/#sample1`

Radford, Wu, Child, Luan, Amodei and Sutskever: Language models are unsupervised multitask learners. OpenAI, 2019.

GPT-2

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Perez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when ... Perez and his friends were astonished to see the unicorn herd. ... While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Perez stated, "We can see, for example that they have a common 'language,' something like a dialect or dialectic." Dr. Perez believes that the unicorns may have originated in Argentina. Some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. ... However, Perez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. . . .

Radford, Wu, Child, Luan, Amodei and Sutskever: Language models are unsupervised multitask learners. OpenAI, 2019.

GPT-3

Continue from:

Feeling unproductive? Maybe you should stop overthinking.

`https://adolos.substack.com/p/feeling-unproductive-maybe-you-should`

`https://www.technologyreview.com/2020/08/14/1006780/ai-gpt-3-fake-blog-reached-top-of-hacker-news/`

Brown et al.: Language Models are Few-Shot Learners. NeurIPS, 2020.

GPT-3

In order to get something done, maybe we need to think less. Seems counter-intuitive, but I believe sometimes our thoughts can get in the way of the creative process. We can work better at times when we ”tune out” the external world and focus on what’s in front of us. I’ve been thinking about this lately, so I thought it would be good to write an article about it. So what exactly does this mean? Let’s start with some definitions.

Definition #1: Creative Thinking (CT) is the act of using your mind to come up with ideas that are not already present in reality or that have never existed before. It is a form of mental activity that involves the use of logic and reason.

Definition #2: Over-Thinking (OT) is the act of trying to come up with ideas that have already been thought through by someone else. OT usually results in ideas that are impractical, impossible, or even stupid. . . .

Brown et al.: Language Models are Few-Shot Learners. NeurIPS, 2020.

GPT-3

- GPT-2 has 1.5 billion parameters
- GPT-3 has 175 billion parameters

- Critique by Rob Toews (Forbes):
  “Deep Learning’s Carbon Emissions Problem”
  `https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/?sh=6cbaef9c6b43`

- Critique by Gary Marcus and Ernest Davis (MIT Technology Review):
  “OpenAI’s language generator has no idea what it’s talking about”
  `https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/`

Brown et al.: Language Models are Few-Shot Learners. NeurIPS, 2020.