Sequencing Legal DNA

NLP for Law and Political Economy

11. Text Generators

Outline

Language Modeling

- “Language Modeling” is a somewhat confusing label for the task of teaching an algorithm to predict/generate language.

Language Modeling

- "Language Modeling" is a somewhat confusing label for the task of teaching an algorithm to predict/generate language.
- The standard approach uses the Markov assumption: future words are independent of the past given the present and some finite number of previous rounds.
  - A kth order markov-assumption assumes that the next word in a sequence depends only on the last k words:

$$\Pr(w_{i+1}|w_{1:i}) \approx \Pr(w_{i+1}|w_{i-k:i})$$

Language Modeling

- "Language Modeling" is a somewhat confusing label for the task of teaching an algorithm to predict/generate language.
- The standard approach uses the Markov assumption: future words are independent of the past given the present and some finite number of previous rounds.
  - A kth order markov-assumption assumes that the next word in a sequence depends only on the last k words:

$$\Pr(w_{i+1}|w_{1:i}) \approx \Pr(w_{i+1}|w_{i-k:i})$$

- Estimating the probability of a sentence is

$$\Pr(w_{1:n}) \approx \prod_{i=1}^{n} \Pr(w_i|w_{i-k:i-1})$$

where $w_{-k+1}, \ldots, w_0$ are replaced with the padding symbol.

Language Modeling

- "Language Modeling" is a somewhat confusing label for the task of teaching an algorithm to predict/generate language.
- The standard approach uses the Markov assumption: future words are independent of the past given the present and some finite number of previous rounds.
  - A kth order markov-assumption assumes that the next word in a sequence depends only on the last k words:

$$\Pr(w_{i+1}|w_{1:i}) \approx \Pr(w_{i+1}|w_{i-k:i})$$

- Estimating the probability of a sentence is

$$\Pr(w_{1:n}) \approx \prod_{i=1}^{n} \Pr(w_i|w_{i-k:i-1})$$

  where $w_{-k+1}, \ldots, w_0$ are replaced with the padding symbol.
- The task is to learn $\Pr(w_{i+1}|w_{1:i})$ given a large corpus.

Perplexity

- Perplexity is an information-theoretic measurement of how well a probability model predicts a sample.
- Given a text corpus of n words $\{w_1, \ldots w_n\}$ and a language model function $\Pr(\cdot)$, the perplexity is:
$$2^{-\frac{1}{n}\sum_{i=1}^{n}\log \widehat{\Pr}(w_i|w_{1:i-1})}$$
- Good language models (i.e., reflective of real language usage) assign high probabilities to the observed words in the corpus, resulting in lower (better) perplexity values.

N-Gram Approach to Language Modeling

- Let $\#(w_{i:j})$ be the count of the sequence of words $w_{i:j}$ in the corpus.
- The MLE estimate for the probability of a word given the previous k words is

$$\widehat{\Pr}(w_{i+1}|w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$$

N-Gram Approach to Language Modeling

- Let $\#(w_{i:j})$ be the count of the sequence of words $w_{i:j}$ in the corpus.
- The MLE estimate for the probability of a word given the previous k words is

$$\widehat{\Pr}(w_{i+1}|w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$$

- The obvious problem:
  - if $w_{i-k:i+1}$ was never observed in the corpus, $\widehat{\Pr}$ is zero, which gives infinite perplexity.
  - zero events are quite common because many phrases are unique.
  - smoothing (adding a small constant to the numerator and denominator) helps.

Neural Language Model Baseline (Goldberg 2017)

- Input:
  - preceding sequence (context words) $w_{1:k}$.
  - V is a finite vocabulary, including special symbols for unknown words, start of sentence, and end of sentence.
  - Each context word is associated with an embedding vector $v(w) \in \mathbb{R}^{n_w}$, the wth row of $\boldsymbol{E}$.
  - The input vector $\boldsymbol{x}$ is a concatenation of the word vectors

Neural Language Model Baseline (Goldberg 2017)

- Input:
  - preceding sequence (context words) $w_{1:k}$.
  - V is a finite vocabulary, including special symbols for unknown words, start of sentence, and end of sentence.
  - Each context word is associated with an embedding vector $v(w) \in \mathbb{R}^{n_w}$, the wth row of $\boldsymbol{E}$.
  - The input vector $\boldsymbol{x}$ is a concatenation of the word vectors
- Output:
  - probability distribution over the next word.

Neural Language Model Baseline (Goldberg 2017)

- Input:
  - preceding sequence (context words) $w_{1:k}$.
  - V is a finite vocabulary, including special symbols for unknown words, start of sentence, and end of sentence.
  - Each context word is associated with an embedding vector $v(w) \in \mathbb{R}^{n_w}$, the wth row of $\boldsymbol{E}$.
  - The input vector $\boldsymbol{x}$ is a concatenation of the word vectors
- Output:
  - probability distribution over the next word.
- The model:

$$
\begin{aligned}
\boldsymbol{x} =& [v(w_1), ..., v(w_k)] \\
\boldsymbol{h} =& \boldsymbol{g}(\boldsymbol{x}\boldsymbol{W}_h) \\
\boldsymbol{y} =& \text{softmax}(\boldsymbol{h}\boldsymbol{W}_y)
\end{aligned}
$$

Training Neural Language Models (Goldberg 2017)

$$
\begin{aligned}
\boldsymbol{x} =& [v(w_1), ..., v(w_k)] \\
\boldsymbol{h} =& \boldsymbol{g}(\boldsymbol{x}\boldsymbol{W}_h) \\
\boldsymbol{y} =& \text{softmax}(\boldsymbol{h}\boldsymbol{W}_y)
\end{aligned}
$$

- Training examples are simply each word in the corpus, with the associated k preceding words as the inputs.
- Each word is associated with an n_w-dimensional embedding vector from a row of E, as well as an n_y-dimensional vector from a column of W_y.
  - These are both informative word representations where words that appear in similar contexts will have similar vector representations.
- The computational cost of these language models is the softmax in the final layer, which becomes slower with an increase in vocabulary size.

Generating Text with a Keras RNN (Geron Ch. 16)

```
model = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=[None, max_id],
                     dropout=0.2, recurrent_dropout=0.2),
    keras.layers.GRU(128, return_sequences=True,
                     dropout=0.2, recurrent_dropout=0.2),
    keras.layers.TimeDistributed(keras.layers.Dense(max_id,
                                                    activation="softmax"))
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam")
history = model.fit(dataset, epochs=20)
```

```
>>> print(complete_text("t", temperature=0.2))
the belly the great and who shall be the belly the
>>> print(complete_text("w", temperature=1))
thing? or why you gremio.
who make which the first
>>> print(complete_text("w", temperature=2))
th no cce:
yeolg-hormer firi. a play asks.
fol rusb
```

- Keras has some really useful text pre-processing tools.
- “Temperature” is the level of randomness in the generator.

Beam Search

- Text decoding generally works word by word.
  - but a generated word at any given point might create a low-probability sequence of words.

Beam Search

- Text decoding generally works word by word.
  - but a generated word at any given point might create a low-probability sequence of words.
- Beam search generates multiple words at any given point, and follows those "beams" to generate several branching sequences.
  - after computing the sequences, e.g., 3-4 words, evaluate their probability and only choose beams with relatively high probability.

Conditioned Generation

- Text generators can use metadata, for example on the speaker.
  - e.g., Li et al (2016) learn a categorical embedding for each user who wrote a response, in order to produce automated responses in the style of each user.

Conditioned Generation

- Text generators can use metadata, for example on the speaker.
  - e.g., Li et al (2016) learn a categorical embedding for each user who wrote a response, in order to produce automated responses in the style of each user.
- As a side effect of training the generator, the network learns user embeddings, producing similar vectors to users who have similar communication styles.
  - At test time, one can influence the style of the generated response by feeding in a particular user (or average user vector) as a conditioning context.

Outline

Autoencoders can memorize complex data

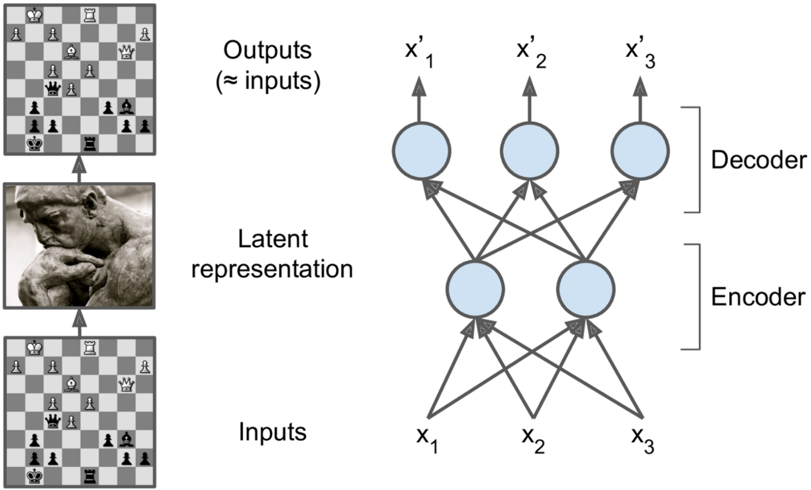


Figure 17-1. The chess memory experiment (left) and a simple autoencoder (right)

- can they memorize low-dimensional encodings and then reproduce text?

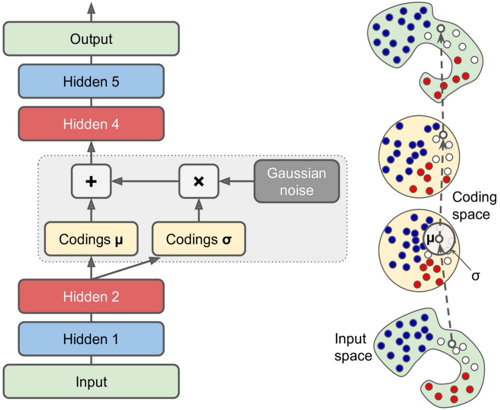


Figure 17-12. Variational autoencoder (left) and an instance going through it (right)

- Variational Autoencoder transforms low-dimensional encodings to the parameters of a gaussian (mean μ and variances σ^2), then draws from the distribution to produce first layer for the decoder.

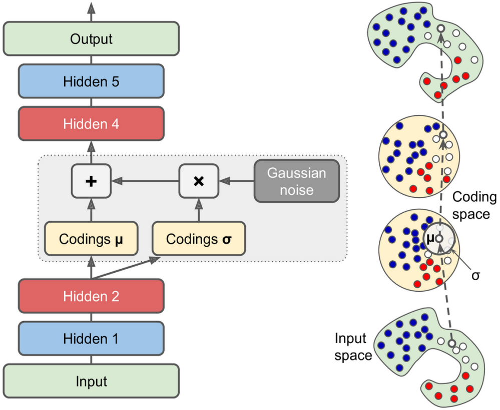


Figure 17-12. Variational autoencoder (left) and an instance going through it (right)

- Variational Autoencoder transforms low-dimensional encodings to the parameters of a gaussian (mean μ and variances σ^2), then draws from the distribution to produce first layer for the decoder.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.

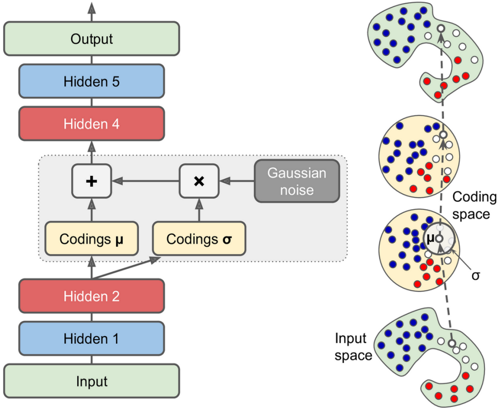


Figure 17-12. Variational autoencoder (left) and an instance going through it (right)

- Variational Autoencoder transforms low-dimensional encodings to the parameters of a gaussian (mean μ and variances σ^2), then draws from the distribution to produce first layer for the decoder.

- Can then sample from the normal distribution (or just choose numbers) and generate reconstructions.
- The amazing thing about VAE's is *semantic interpolation*:
  picking an encoding vector between two encodings will produce a reconstruction that is in between the associated images/documents:

- doesn't seem to work will with text (but a nice replication exercise).

Outline

Language Models

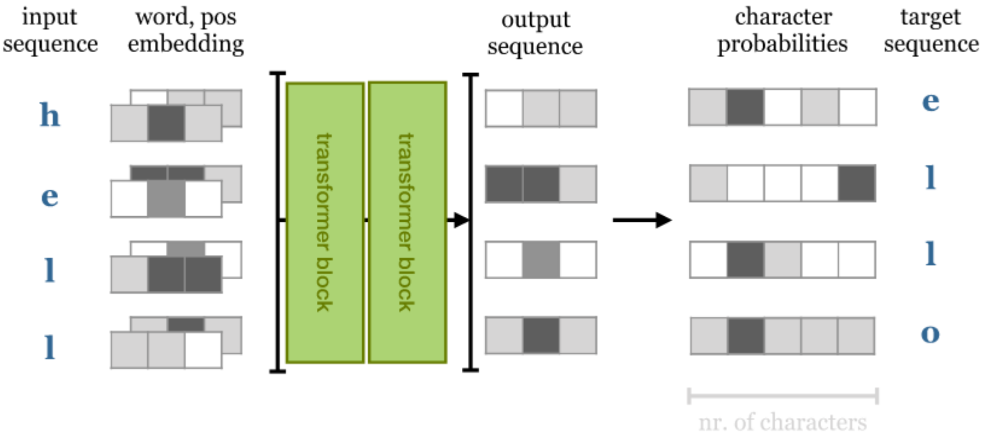
Variational Autoencoders

Text Generation with Transformers

- GPT-2
- Kreps et al (2019): All the News that's Fit to Fabricate
- Ash and Peric (2020): Legal Text Generation using Transformer

Generative Adversarial Networks

Text generation transformer



Outline

OPENAI'S NEW MULTITALENTED AI WRITES, TRANSLATES, AND SLANDERS

A step forward in AI text-generation that also spells trouble

By James Vincent | Feb 14, 2019, 12:00pm EST

Howard, co-founder of Fast.AI agrees. "I've been trying to warn people about this for a while," he says. "We have the technology to totally fill Twitter, email, and the web up with reasonable-sounding, context-appropriate prose, which would drown out all other speech and be impossible to filter."

https://transformer.huggingface.co/doc/distil-gpt2

Outline

News Generation Experiment

- ...

Outline

Last Year's Projects (1)

Lazar Peric: GPT Text Generator for Legal Text

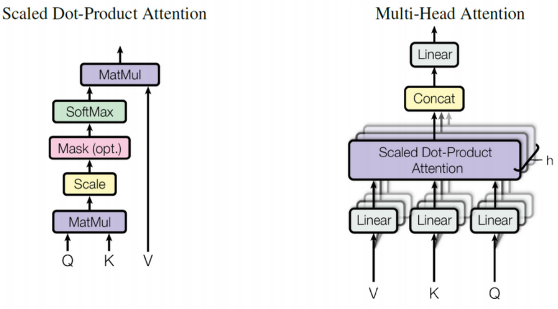


Figure 4.1: Scaled Dot-Product Attention (left) and Multi-Head Attention (right) block. Figure taken from [6]

Last Year's Projects (1)

Lazar Peric: GPT Text Generator for Legal Text

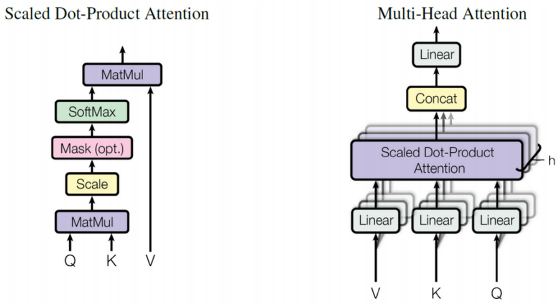


Figure 4.1: Scaled Dot-Product Attention (left) and Multi-Head Attention (right) block. Figure taken from [6]

```
======================================== Start text ========================================
[ * 1209 ] JERRE S. WILLIAMS , Circuit Judge : <eos> This appeal is lodged by H. Roger Lawler
, a debtor in bankruptcy . He claims that an award in bankruptcy of $ <unk> in attorneys
' fees to Richard W. Horton and Vernon O. <unk> is too high . Horton and <unk> are cross -
appealing the amount of the award
======================================= Generated text =======================================
in question , which was based on a $ 1.84 judgment . The district judge found , and the court
concluded that there were sufficient facts in support thereof . We find that there is no
evidence that they are not supported in any way . The judgment is reversed . The judgments
appealed therefrom will stand and will stand and will bear in all other parts of this
judgment , except as they will bear their respective portions of their judgments , together
therewith , with costs of this opinion , with directions that they be reversed and the case
is REMANDED to that portion thereof , and will bear its part with instructions for the new
```

Outline

Generative Adversarial Networks

- The model has two players, generator and discriminator:
  - discriminator says, given an input, what should the label be.
  - generator tries to generate an input that fools the discriminator

Generative Adversarial Networks

- The model has two players, generator and discriminator:
  - discriminator says, given an input, what should the label be.
  - generator tries to generate an input that fools the discriminator
- this has been good for image classification but again, not much in the way of social science applications.