# Machine Learning for Natural Language Processing

## Language Modeling

Lecture 5

**Benjamin Muller**

INRIA Paris - ALMANACH

benjamin.muller@inria.fr

# Course Outline

1. The Why and What of Natural Language Processing
2. Representing text with vectors
3. Task specific Modeling of Text
4. Neural Natural Language Processing
5. Language Modeling
6. Transfer Learning with Neural Modeling for NLP

# Lecture Outline

- Language Model
- Conditioned Language Model : focus on Sequence to Sequence

# Language Model

- What is a Language Model ?
- Modeling language with n-grams
- Modeling language with a LSTM
- The Transformer Architecture

# Language modeling

# What is language modeling?

- **Language modeling** corresponds to assigning a probability to a text
- A text is a sequence of tokens, or characters
- Tokens can be words, sub-words,
- For example:

$$\{\text{a cat}\} \quad = \quad \{\text{a}, \text{cat}\},$$

# What is language modeling?

- **Language modeling** corresponds to assigning a probability to a text
- A text is a sequence of tokens, or characters
- Tokens can be words, sub-words,
- For example:

$$\begin{aligned}\{\text{a cat}\} &= \{\text{a}, \text{cat}\}, \\ &= \{\text{a}, \ , \text{c}, \text{a}, \text{t}\},\end{aligned}$$

# What is language modeling?

- **Language modeling** corresponds to assigning a probability to a text
- A text is a sequence of tokens, or characters
- Tokens can be words, sub-words,
- For example:

$$
\begin{aligned}
\{\text{a cat}\} &= \{\text{a}, \text{cat}\}, \\
&= \{\text{a}, \ , \text{c}, \text{a}, \text{t}\}, \\
&= \{\text{a}, \ , \text{ca}, \text{t}\}.
\end{aligned}
$$

# What is language modeling?

- Given a sequence $\{w_1, \ldots, w_T\}$ of tokens, a language model estimates its probability:

$$P(w_1, \ldots, w_T)$$

- $P$ depends on a **vocabulary**, i.e., the set of unique tokens.

- Question: How to estimate $P$ ?

# What is language modeling?

- Given a sequence $\{w_1, \ldots, w_T\}$ of tokens, a language model estimates its probability:

$$P(w_1, \ldots, w_T)$$

- $P$ depends on a **vocabulary**, i.e., the set of unique tokens.

- Question: How to estimate $P$ ?

# Language Models

- Causal Language Model
- Mask Language Model

# Applications of language modeling

Language models are applied in several fields:

- Speech recognition:

$$P(\text{"Vanilla, I scream"}) < P(\text{"Vanilla ice cream"}).$$

- Machine translation:

$$\begin{aligned} P(\text{"Déçu en bien"} \mid \text{"Pleasantly surprised"}) &< \\ P(\text{"Agréablement surpris"} \mid \text{"Pleasantly surprised"}) & \end{aligned}$$

- Optical Character Recognition:

$$P(\text{"m0ve fast"}) < P(\text{"move fast"})$$

# Probabilistic *Causal* language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

# Probabilistic *Causal* language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

# Probabilistic *Causal* language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

- Recursively applied to a sequence:

$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_1)P(w_2, w_3 \mid w_1) \\ &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2, w_1). \end{aligned}$$

# Probabilistic *Causal* language model

- Sequence probability as a product of token probabilities:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Indeed we have:

$$P(a, b) = P(a)P(b \mid a)$$

- Recursively applied to a sequence:

$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_1)P(w_2, w_3 \mid w_1) \\ &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2, w_1). \end{aligned}$$

- Causal Language models estimate probability of upcoming token given past:

$$P(w_t \mid w_{t-1}, \ldots, w_1).$$

# Estimating Language Models

- Causal Language Model
- Mask Language Model

# Mask Language Model [1,2]

**Sentence** The cat is drinking milk in the kitchen

[1] Devlin et al. (2018)
[2] also referred as Cloze Task

# Mask Language Model [1,2]

**Sentence** The cat is drinking milk in the kitchen

**input** The cat <MASK> drinking <MASK> in the kitchen

- Randomly replace 15% of words in sentence with a <MASK> token

[1] Devlin et al. (2018)

[2] also referred as Cloze Task

# Mask Language Model [1,2]

| | |
|---|---|
| **Sentence** | The cat is drinking milk in the kitchen |
| **input** | The cat <MASK> drinking <MASK> in the kitchen |
| **targets** | {“is”, “milk”} |

- Randomly replace 15% of words in sentence with a <MASK> token
- Take the masked words as targets for the model to predict

[1] Devlin et al. (2018)
[2] also referred as Cloze Task

# Mask Language Model [1,2]

**Sentence** The cat is drinking milk in the kitchen

**input** The cat mushroom drinking shoes in the kitchen

**targets** {“is”, “milk”}

- Randomly replace 15% of words in sentence with a <MASK> token
- Take the masked words as targets for the model to predict
- Extension: use random words from vocabulary instead of <MASK>

[1] Devlin et al. (2018)

[2] also referred as Cloze Task

# Mask language model

Masked Language Modeling estimates the probability of sequence tokens of length T with:

$$P(w_i | w_1, .., w_{i-1}, w_i, .., w_T)$$

# Language Models in a nutshell

- a Language Model is a model that predicts a **token** based on its surrounding linguistic **context**
- Tokens can be words, sub-words or characters
- Context can be the *left sequence*, *left and right sequence*, the sentence, a window around the words, the paragraph...
- We saw two way of defining langue models: Causal Language Model and Mask Language Model

# Estimating language models

# Estimating language models

- Statistical approach: N-Gram model
- Neural Language Models
-
    - Recurrent Neural Networks (LSTM)
    - The Transformer Architecture

# Count based language model

- Example:

$$
\begin{aligned}
P(\text{English} \mid \text{The moment one learns}) &= \frac{c\,(\text{The moment one learns English})}{c\,(\text{The moment one learns})} \\
&= \frac{35}{73} = 0.48
\end{aligned}
$$

Sentence "The moment one learns English" appears 35 in dataset

Sentence "The moment one learns" appears 75 in dataset

# Limitiations of count based language model

- Number of unique sentences increases with dataset size,
- Long sentences are rare: no good statistics for them

$\rightarrow$ Too many sentences with not enough statistics
(Sparsity due to combinatorial structure of language)

# Count based language model

- **Solution** truncate past to a fixed size window
- For example:

$$P(\text{English} \mid \text{The moment one learns}) \approx P(\text{English} \mid \text{one learns})$$

- Implicit assumption: most important information about a word is in its recent history
- **Beware!** In general:

$$P(w_1, \dots, w_T) \neq \prod_{t=1}^{T} P(w_t \mid w_{t-1}, \dots, w_{t-n+1})$$

# Count based language model

- **Truncated count based models = *n*-gram models**
- "n" refers to the size of past
- Examples:
  - Unigram:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t)$$

  - Bigram:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1})$$

# Count based language model: unigram

- Probability of a sentence with a unigram model:

$$P_U(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t) = \prod_{t=1}^{T} \frac{c(w_t)}{N}$$

$N$ = total number of tokens in dataset
$c(w_t)$ = number of occurences of $w_t$ in dataset

- Unigram only uses word frequency

- Example of text generation with this model:

*the or is ball then car*

# Count based language model: bigram

- Probability of a sentence with a bigram model:

$$P_U(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}) = \prod_{t=1}^{T} \frac{c(w_{t-1} w_t)}{c(w_{t-1})}$$

$c(w_{t-1} w_t) =$ number of occurences of sequence $w_{t-1} w_t$

- Predict a word just with the previous word

# Count based language model: bigram

- Example of text generation with bigram model:

*new car parking lot of the*

- "car" is generated from "new", "parking" from "car"...
- But "new" has no influence on "parking"

# Count based language model

- Simple to extend to longer dependencies: trigrams, 4-grams...
- *n*-grams can be "good enough" in some cases
- But *n*-grams cannot capture long term dependencies required to truely model language

# Estimating $n$-gram probabilites: an example

- bigram:

$$P(w_t \mid w_{t-1}) = \frac{c(w_{t-1}w_t)}{c(w_{t-1})}$$

- Dataset:

$$<s>\textit{we sat in the house}$$
$$<s>\textit{we sat here we two and we said}$$
$$<s>\textit{how we wish we had something to do}$$

- Extract some probabilities:

$$P(sat \mid we) = 0.33, \; P(wish \mid we) = 0.17, \; P(in \mid sat) = 0.5$$

- $<$s$>$= token for beginning of sentence; $P(<\text{s}>) = 1$.
- Compute sentence probability with them

# Estimating $n$-gram probabilites: an example

- Extract count from Berkeley Restaurant dataset (9222 sentences)
- Unigram counts:

| i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|
| 2533 | 927 | 2417 | 746 | 158 | 1093 | 341 | 278 |

- Bigram counts:

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 5 | 827 | 0 | 9 | 0 | 0 | 0 | 2 |
| want | 2 | 0 | 608 | 1 | 6 | 6 | 5 | 1 |
| to | 2 | 0 | 4 | 686 | 2 | 0 | 6 | 211 |
| eat | 0 | 0 | 2 | 0 | 16 | 2 | 42 | 0 |
| chinese | 1 | 0 | 0 | 0 | 0 | 82 | 1 | 0 |
| food | 15 | 0 | 15 | 0 | 1 | 4 | 0 | 0 |
| lunch | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| spend | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

# Estimating $n$-gram probabilites: an example

- The bigram probabilities are obtained by dividing the bigram counts with the unigram counts:

$$P(w_2 \mid w_1) = \frac{c(w_1 w_2)}{c(w_1)}$$

- Resulting bigram probabilities:

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 0.022 | 0.33 | 0 | 0.036 | 0 | 0 | 0 | 0.00079 |
| want | 0.0022 | 0 | 0.66 | 0.0011 | 0.0065 | 0.0065 | 0.0054 | 0.0011 |
| to | 0.00083 | 0 | 0.0017 | 0.28 | 0.00083 | 0 | 0.0025 | 0.087 |
| eat | 0 | 0 | 0.0027 | 0 | 0.021 | 0.0027 | 0.056 | 0 |
| chinese | 0.0063 | 0 | 0 | 0 | 0 | 0.52 | 0.0063 | 0 |
| food | 0.014 | 0 | 0.014 | 0 | 0.00092 | 0.0037 | 0 | 0 |
| lunch | 0.0059 | 0 | 0 | 0 | 0 | 0.0029 | 0 | 0 |
| spend | 0.0036 | 0 | 0.0036 | 0 | 0 | 0 | 0 | 0 |

# Estimating $n$-gram probabilites: an example

- Example:

$$P(\text{<s> i want chinese food})?$$

$\text{<s>}=$ token for beginning of sentence; $P(\text{<s>}) = 1$.

- Result:

$$\begin{aligned} P(\text{<s> i want chinese food}) &= P(\text{<s>})P(\text{i}|\text{<s>})P(\text{want}|\text{i})P(\text{chinese}|\text{want})P(\text{food}|\text{chinese}) \\ &= 1 \times .25 \times 0.33 \times 0.0065 \times 0.52 \\ &= 0.00027885 \end{aligned}$$

# Estimating $n$-gram probabilites: an example

| | i | want | to | eat | chinese | food | lunch | spend |
|---|---|---|---|---|---|---|---|---|
| i | 0.022 | 0.33 | 0 | 0.036 | 0 | 0 | 0 | 0.00079 |
| ... | | | | | | | | |
| food | 0.014 | 0 | 0.014 | 0 | 0.00092 | 0.0037 | 0 | 0 |
| lunch | 0.0059 | 0 | 0 | 0 | 0 | 0.0029 | 0 | 0 |
| spend | 0.0036 | 0 | 0.0036 | 0 | 0 | 0 | 0 | 0 |

- Example:

$$P(\text{<s> i bring my lunch to work})?$$

- Result:

$$\begin{aligned} P(\text{<s> i bring my lunch to work}) &= P(\text{<s>}) \dots P(\text{to}|\text{lunch}) \dots \\ &= 1 \times \cdots \times 0 \times \dots \\ &= \mathbf{0} \end{aligned}$$

- **Does not generalize well!**

# Good-Turing estimation

- **Idea** reallocate probability mass of $n$-grams that occur exactly $c + 1$ times to $n$-grams that occur exactly $c$ times
- reallocate mass of $n$-grams appearing once to unseen $n$-grams

$\rightarrow$ **alternative to Add-1**

# Good-Turing estimation

- **Idea** reallocate probability mass of $n$-grams that occur exactly $c+1$ times to $n$-grams that occur exactly $c$ times
- reallocate mass of $n$-grams appearing once to unseen $n$-grams

→ **alternative to Add-1**

- the adjusted count:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

where $N_c$ is the number of $n$-grams that appears exactly $c$ times

# Good-Turing estimation

- **Idea** reallocate probability mass of $n$-grams that occur exactly $c+1$ times to $n$-grams that occur exactly $c$ times
- reallocate mass of $n$-grams appearing once to unseen $n$-grams

$\rightarrow$ **alternative to Add-1**

- the adjusted count:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

  where $N_c$ is the number of $n$-grams that appears exactly $c$ times
- $n$-gram probability depends on $c^*$ instead of $c$

# Good-Turing estimation

- **Idea** reallocate probability mass of $n$-grams that occur exactly $c+1$ times to $n$-grams that occur exactly $c$ times
- reallocate mass of $n$-grams appearing once to unseen $n$-grams

$\rightarrow$ **alternative to Add-1**

- the adjusted count:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

  where $N_c$ is the number of $n$-grams that appears exactly $c$ times

- $n$-gram probability depends on $c^*$ instead of $c$
- **Problem** What if $N_{c+1} = 0$ (but $N_c > 0$)?

# Backoff and Interpolation

- If no good statistics on long context: use shorter context
- **Backoff**: use trigram if enough data, else backoff to bigram.
- **Interpolation**: mix statistics of trigram, bigram and unigram.

# Pros and Cons of N-Gram Language Models

**Pros**

- Fast at training and inference
- Can reach good accuracy if lots of data

**Cons**

- Impossible to model very long dependencies (simplistic assumptions done)
- Generalization limited
- Not Deep Learning compatible

# Estimating language models

- Statistical approach: N-Gram model
- Neural Language Models
-
    - Recurrent Neural Networks (LSTM)
    - The Transformer Architecture

# Neural Language Models

How to frame language modeling in a deep learning compatible way ?
What neural architecture/objective ?

- Neural Language Model objective and Training
- Architectures
  - Recurrent Network
  - Transformer

# Neural Language Model

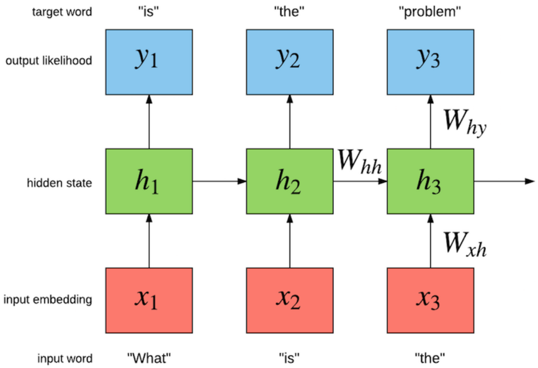



Figure: Neural Language modeling schema view [3]

[3] http://torch.ch/blog/2016/07/25/nce.html

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) $\{$f,W$\}$ dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \; \forall t <= T \; \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \; \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) {f,W} dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \ \forall t <= T \ \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \ \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

**Train time**

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) {f,W} dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \; \forall t <= T \; \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \; \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

**Train time**

$$\hat{p}_t = softmax_V(o_t) = (\frac{e^{o_{tv}}}{\sum_k e^{o_{tk}}})_{v \in 1.V}$$

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) $\{$f,W$\}$ dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \; \forall t <= T \; \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \; \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

**Train time**

$$\hat{p}_t = softmax_V(o_t) = (\frac{e^{o_{tv}}}{\sum_k e^{o_{tk}}})_{v \in 1.V}$$

$$loss = CE(\hat{p}_t, p_t) = log(\hat{p}_{t_{x^{t+1}}})$$

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) {f,W} dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \; \forall t <= T \; \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \; \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

**Train time**

$$\hat{p}_t = softmax_V(o_t) = (\frac{e^{o_{tv}}}{\sum_k e^{o_{tk}}})_{v \in 1.V}$$

$$loss = CE(\hat{p}_t, p_t) = log(\hat{p}_{t_{x^{t+1}}})$$

Compute $\nabla loss$ backprop
(update E, $\theta$, $W$)

# Neural Language Model training and inference

Let $(x^1, .., x^T)_i$ sequence of tokens (1-hot encoded), E embedding layer, $NN_\theta$ a sequential model (e.g. LSTM) {f,W} dense layer
We present forward/backward step to predict token $x^{t+1}$ with $x^1, .., x^t$

$$e_t = Ex^t \ \forall t <= T \ \textit{Embedding layer}$$

$$h_t = NN_\theta(e_1, ..e_{t-1}, e_t) \ \textit{sequential layers with weights } \theta$$

$$s_t = f(Wh_t) \text{ Dense Layer}$$

**Train time**

$$\hat{p}_t = softmax_V(o_t) = (\frac{e^{o_{tv}}}{\sum_k e^{o_{tk}}})_{v \in 1.V}$$

$$loss = CE(\hat{p}_t, p_t) = log(\hat{p}_{t_{x^{t+1}}})$$

Compute $\nabla loss$ backprop
(update E, $\theta$, $W$)

**Inference/Prediction Time**

$$x_{t+1} = argmax_{v \in 1,..,V}(s_{tv})$$

# Neural Language Model with LSTM cell

In this case, $NN_\theta$ is defined as (seen in lecture 4):
($\theta$ is equal to $W_{p \in C,f,i,o}$)

Based on $e_1, .., e_t$ we compute iteratively $h_1, .., h_t$

$$\widetilde{C}^t = tanh(W_C[e_t, h_{t-1}] + b_c) \quad \text{candidate cell}$$

$$f^t = \sigma(W_f[x_t, h_{t-1}] + b_f) \quad \text{forget gate}$$
$$i^t = \sigma(W_i[x_t, h_{t-1}] + b_i) \quad \text{input gate}$$
$$o^t = \sigma(W_o[x_t, h_{t-1}] + b_o) \quad \text{ouput gate}$$

$$C^t = i^t \star \widetilde{C}^t + f^t \star C^{t-1} \quad \text{new cell state}$$
$$h_t = o^t \star tanh(C_t) \quad \text{new hidden vector}$$

4

[4] $\star$ *elementwiseproduct*

# Evaluating Language Models

- Language Models are evaluated with perplexity

$$perplexity = 2^{-p_i log(\hat{p}_i)}$$

- It is a measure of "surprise" of the model

# Comparing various language Models

| Model | Perplexity |
| --- | --- |
| Kneser-Ney 5-gram | 141 |
| Neural *n*-gram | 140 |
| RNN | 125 |
| LSTM | **115** |

- Penn Treebank dataset
- LSTM outperforms RNN

# Limits of LSTM-based architectures

- LSTM models are widely used in NLP for their ability to model sequential data
- In theory, they are able to model sequences of infinite length (Siegelmann and Sontag, 1992)
- In practice, until recently LSTM based models were State-of-the-Art (SOTA) for language modeling (Rae et al., 2018)

- In practice, the recurrent nature of LSTM limits the possibility to scale the training process to more data (we cannot parallelize LSTM easily!)
- $\rightarrow$ Transformer were recently shown to work better in practice for a great variety of tasks including Language Model (Radford et al., 2019)

# The Transformer Architecture[5]

[5] Vaswani et al. (2017)

# Combining vectors with attention

- Use (self) attention mechanism
- Given a set of vectors $\mathbf{w}_1$, ..., $\mathbf{w}_T \in \mathbb{R}^d$ representing words

$$\mathbf{h}_t = \sum_{i=1}^{T} a_{it} \mathbf{V} \mathbf{w}_i$$

  where $\sum_{i=1}^{T} a_{it} = 1$.
- We could use $a_{it} = \frac{1}{T}$ and get bag of words
- We can also learn $a_{it}$ based on the input and output as we did for the standard attention mechanism

# Combining vectors with attention

- Introducing matrix $\mathbf{W} \in \mathbb{R}^{d \times T}$ where columns correspond to $\mathbf{w}_i$,

$$\mathbf{h}_t = \mathbf{V}\mathbf{W}\mathbf{a}_t$$

- And finally

$$\mathbf{H} = \mathbf{V}\mathbf{W}\mathbf{A}$$

# Combining vectors with attention

- How to compute the matrix $\mathbf{A}$?

$$\mathbf{A} = \text{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q} \mathbf{W})$$

  where the softmax is applied column-wise.

- Why softmax? to get positive entries, and columns summing to 1.
- Why $\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q} \mathbf{W}$? Bilinear form over the input

# Combining vectors with attention

- Putting everything together:

$$\mathbf{H} = \mathbf{V}\mathbf{W}\text{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q}\mathbf{W})$$

  where $\mathbf{H}, \mathbf{W} \in \mathbb{R}^{d \times T}$ and $\mathbf{V}, \mathbf{K}, \mathbf{Q} \in \mathbb{R}^{d \times d}$
- $\mathbf{V}, \mathbf{K}, \mathbf{Q}$ are parameters to be learned.
- This operation is called self-attention

# Combining vectors with attention

- Putting everything together:

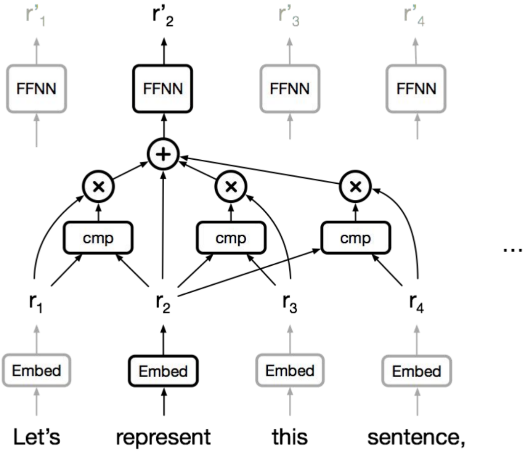
$$\mathbf{H} = \mathbf{V}\mathbf{W}\text{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q}\mathbf{W})$$

where $\mathbf{H}, \mathbf{W} \in \mathbb{R}^{d \times T}$ and $\mathbf{V}, \mathbf{K}, \mathbf{Q} \in \mathbb{R}^{d \times d}$

- $\mathbf{V}, \mathbf{K}, \mathbf{Q}$ are parameters to be learned.
- This operation is called self-attention

- It can be generalized to multiple heads:
  - Split input vectors into $n$ subvectors of dimension $d/n$,
  - Apply self attention (with different $\mathbf{V}, \mathbf{K}, \mathbf{Q}$) over these smaller vectors
  - Concatenate the results to get back $d$ dimensional vectors

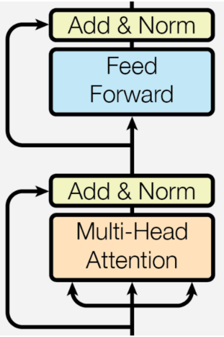
# Combining vectors with attention




from Vaswani and Huang:
`http://web.stanford.edu/class/cs224n/slides/`

Transformer block:

- Multi-head attention layer with skip connection and normalization
- Followed by feed forward with skip connection and normalization




Vaswani et al. (2017)

Skip connection+normalization:

- Given a network block **nn** and input **x**
- The output **y** is computed as

$$\mathbf{y} = \mathbf{norm}(\mathbf{x} + \mathbf{nn}(\mathbf{x}))$$

where **norm** normalize the input

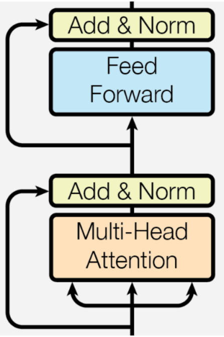
# Transformer network

Feed forward block

- Two layer network, with ReLU activation

$$\mathbf{y} = \mathbf{W}_2 \texttt{ReLU}(\mathbf{W}_1 \mathbf{x})$$

- Usually, $\mathbf{W}_1 \in \mathbb{R}^{4d \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times 4d}$
- i.e. hidden layer of dimension $4d$.




Vaswani et al. (2017)

# Position embeddings

- **Limitation:** self attention does not take position into account!
- Indeed, shuffling the input gives the same results

- **Solution:** add position encodings.
- Replace the matrix $\mathbf{W}$ by $\mathbf{W} + \mathbf{E}$, where $\mathbf{E} \in \mathbb{R}^{d \times T}$

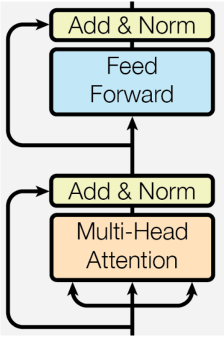
- $\mathbf{E}$ can be learned, or defined using sin and cos:

$$e_{2i,j} = \sin\left(\frac{j}{10000^{2i/d}}\right)$$
$$e_{2i+1,j} = \cos\left(\frac{j}{10000^{2i/d}}\right)$$

# Transformer network

Transformer network:

- Word embeddings + Position embeddings
- Then $N$ transformer blocks (e.g. $N = 12$)
- Softmax classifier (e.g. for language modeling)




Vaswani et al. (2017)

# Masking for Transformer Language Models

- In transformer, $\mathbf{h}_t$ depends on all inputs
- Could not be used as is for language modeling
- Solution: use mask in attention, to only use past

- Reminder:

$$\begin{aligned}\mathbf{H} &= \mathbf{V}\mathbf{W}\mathrm{softmax}(\mathbf{W}^\top \mathbf{K}^\top \mathbf{Q}\mathbf{W}) \\ &= \mathbf{V}\mathbf{W}\mathbf{A}\end{aligned}$$

  Hence, $\mathbf{a}_{it}$ is weight of input $i$ in representation of position $t$
- We want representation at time $t$ to only depends on $i \leq t$
- We could enforce $\mathbf{a}_{it} = 0$ for $i \geq t$

# Masked softmax

- We introduce the masked softmax operator
- Given an input $\mathbf{x}$ and a binary mask $\mathbf{m}$,

$$[\text{masked\_softmax}(\mathbf{x}, \mathbf{m})]_i = \frac{m_i \exp(x_i)}{\sum_{i=1}^{d} m_i \exp(x_i)}$$

- Still sums to one, $m_i = 0$ implies $[\text{masked\_softmax}(\mathbf{x}, \mathbf{m})]_i = 0$
- Sometimes implemented as:

$$\text{softmax}(\mathbf{x} + \log(\mathbf{m}))$$

- **Beware:** do not learn the mask (e.g. PyTorch: `register_buffer`)

# Training of a Transformer

- In practice, transformers are very unstable during training
- If the learning rate is too large, it diverges
- However if the learning rate is too small, it does not learn well

# Transformer network for Language Modeling: Results

| Model | bpc |
|---|---|
| LSTM | 1.25 |
| Transformer | **1.07** |

- Text8
- Character level language modeling
- bpc = bit per character.

# Why are language model useful?

- Standard Language Models are not that useful as such
- For specific-tasks we will see that they can be useful in Lecture 6
- For controlled generation (Machine Translation, Speech to Text, Question Answering...) we need more.

- **How to build a "controllable" text generation system using a language model ?**

# Lecture Outline

- Language Model
- Conditioned Language Model : focus on Sequence to Sequence

# Conditioned Language Models

- Problematically, controllable text generation can be seen as estimating:

$P(w_t|w_1, .., w_{t-1}, C)$ where C is a conditioning variable

# Sequence to Sequence Architecture

# Direct modeling of translation

We have:

a sentence $S = (x_1, \dots, x_m)$ in a Source language (e.g. French)

its translation $T = (y_1, \dots, y_n)$ in a Target language (e.g. English)

We directly work on the probability of a translation given a source sentence by expressing translation as conditional language modeling:

$$P(T \mid S) = \prod_{t=1}^{n} P(y_t \mid y_{t-1}, \dots, y_1, S)$$

Goal Learn a translation model where $T$ is the most probable sentence given $S$:

$$T = \underset{T' \text{ in Target language}}{\operatorname{argmax}} P(T' \mid S)$$

Challenge How to encode the source sentence S ?

# Sequence to Sequence: Machine Translation

We want to condition a language model of the target language (e.g English) on a source sentence

1. Encode source sentences
2. Generate the target sentence based on the encoded source and a language target language model

# Sequence to Sequence: Machine Translation

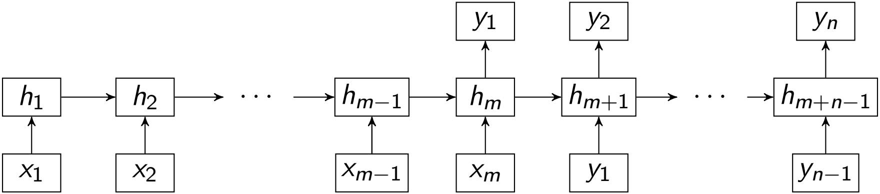
We want to condition a language model of the target language (e.g English) on a source sentence

1. Encode source sentences
2. Generate the target sentence based on the encoded source and a language target language model

- We have seen that Neural Networks are good Language Models (i.e. can generate proper sentences)
- We have seen that Neural Networks are good at modeling sequence.
  → We are going to combine two network
  - An encoder for encoding source sentences
  - A decoder for conditioned language modeling

→ This new architecture is referred to as an *encoder-decoder* or *sequence to sequence model*

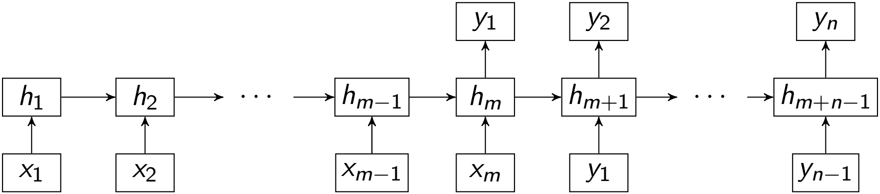
# Simple approach: sequence-to-sequence (seq2seq)




Equivalent to:

1. Build a representation of the source sentence by taking last hidden layer $\mathbf{h}_T$ of LSTM applied to the source sentence
2. Use this representation as initialization of the hidden variables of LSTM applied to the target sentence

# Simple approach: sequence-to-sequence (seq2seq)




- Pro:
    - Very simple to implement:
    - input: Concatenation of source and target sentence.
    - ouptut: target sentence
- Cons:
    - Needs large hidden layer to store everything about source sentence
    - Does not work on very long sentences
    - Same conditioning for the whole target sentence

# Seq2seq

Architecture

- Encoder and Decoder can be any NN architecture seen so far
- In practice, LSTMs and Transformer are the most efficient (in most cases)

Decoding

- At inference, we can improve performance by using *beam-search* instead of *greedy* decoding

# Training Seq2Seq

- As any Neural Networks, we train a seq2seq architecture with backpropagation
- Using pairs of source-target aligned sentences we train the model to generate the target language based on source language

  Source: *This week we'll continue to try to close a deal to purchase a dairy farm.*

  Target: *Cette semaine, nous allons continuer d'essayer de signer un contrat d'achat d'une exploitation laitière.*

# Attention Mechanism for sequence to sequence

- To overcome the main encoding issue, the sequential attention on the source sentence improve importantly the performance

# Evaluation

- How good is a given machine translation system?
- Hard problem, since many different translations acceptable
- Evaluation metrics:
  - subjective judgments by human evaluators
  - automatic evaluation metrics
  - task-based evaluation (how much post-editing effort? does information come across?)

NB: Evaluating sequence generation model is never easy (subjectivity!)
[6]

[6] from Philipp Koehn: `http://mt-class.org/jhu/`

# BLEU

Measure $n$-gram overlap between machine translation output and reference translation

Compute precision for $n$-grams of size 1 to 4

Add brevity penalty to avoid too short translations

$$\text{BLEU} = \min\left(1, \frac{output\_length}{reference\_length}\right)\left(\prod_{i=1}^{4}\frac{C_i}{N_i}\right)^{\frac{1}{4}}$$

where $C_i$ is the number of correct $n$-gram of size $i$ and $N_i$ is the total number of $n$-grams in the output of the system

Computed over full corpora, not just a sentence from Philipp Koehn: `http://mt-class.org/jhu/`

# Other Sequence to Sequence Tasks

- (Abstractive) Summarization

  **Input:** Document
  **Output:** Summary

- Text Simplification

  **Input:** Complex sentence
  **Output:** Simplified sentence

- Multi-Modal tasks
  - Speech To Text
  - Caption Generation (Image to Text)

# References I


Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Rae, J. W., Dyer, C., Dayan, P., and Lillicrap, T. P. (2018). Fast parametric learning with activation memorization. *arXiv preprint arXiv:1803.10049*.

Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.