Advanced Methods in Text Analytics

Exercise 3: Language Models - Part 1

Daniel Ruffinelli

University of Mannheim

FSS 2025

Language Modeling

Context

- Language models are able to predict the next word in a given sequence of n words.
- Similarly, language models can give the probability of a given sequence of n words.

Language Modeling

Question a)

- **Question:** Give a formal expression for the probability of word $n+1$ given the previous n words in a sequence.

Language Modeling

Question a)

- **Question:** Give a formal expression for the probability of word $n+1$ given the previous n words in a sequence.
- **Answer:** Let x_i be word in position i in a sequence of n words. Then:

$$p(x_{n+1}|x_1, x_2, \ldots, x_{n-1}, x_n)$$

  is the probability of predicting word $n+1$ given previous n words.

Language Modeling

Question b)

- **Question:** Give a formal expression for the probability of a sequence of *n* words and specify how this probability is computed using the marginal probabilities of each word in the sequence.

Language Modeling

Question b)

- **Question:** Give a formal expression for the probability of a sequence of *n* words and specify how this probability is computed using the marginal probabilities of each word in the sequence.
- **Answer:** We use the chain rule to compute the joint distribution of the words in a sequence. That is,

$$p(x_1, x_2, \ldots, x_n) = p(x_1)p(x_2|x_1)\ldots p(x_n|x_1, x_2, \ldots, x_{n-1}).$$

Language Modeling

Question c)

- What is the main assumption that n-gram language models make?
- What is the name of this assumption?
- What is a bigram model? And a trigram model?
- Give the expressions for computing the probability of a sequence of words (i.e. your answer to b)) using a bigram model, and a trigram model.

Language Modeling

Answer c) (1)

- N-gram models assume that predicting a given word x_n only depends on the $n-1$ words that precede it.
- This is known as a Markov assumption, which often refers to the assumption that we can predict future probabilities without looking too much into the past.
- A bigram model predicts a word given only the single previous word.
- So using the chain rule, we can compute the probability of a given sequence as follows:

$$p(x_n|x_1,\ldots,x_n) = p(w_1)p(w_2|w_1)p(w_3|w_2)\ldots p(w_k|w_{k-1}).$$

- Similarly, for trigram models, we have:

$$p(x_n|x_1,\ldots,x_n) = p(w_1)p(w_2|w_1)p(w_3|w_1,w_2)\ldots p(w_k|w_{k-1},w_{k-2})$$

Language Modeling

Answer c) (2)

- In practice, special tags ⟨s⟩ and ⟨/s⟩ are used to mark the beginning and end of sentences.
- Thus, a bigram model's prediction would be as follows:

$$p(x_n|x_1, \ldots, x_n) = p(w_1|\langle s \rangle)p(w_2|w_1)p(w_3|w_2)\ldots p(w_k|w_{k-1}).$$

- Similarly, if a model predicts ⟨/s⟩, then we know the generated sentence is over.

Language Modeling

Question d)

- How is the probability of predicting the next word given previous words *estimated* in n-gram models?

Language Modeling

Answer d)

- The probability for predicting word x_n is given by comparing the probability of the sequence of n words, i.e. the $n-1$ words followed by x_n, against the probability of the sequence of previous $n-1$ words.
- These probabilities are estimated by counting these sequences over a large corpus.
- That is:

$$p(x_n | x_1, \ldots, x_{n-1}) = \frac{\text{count}(x_1, x_2, \ldots, x_n)}{\text{count}(x_1, x_2, \ldots, x_{n-1})}.$$

- It can be shown that this corresponds to performing MLE.

Language Modeling

Question e)

- **Question:** What are some of the problems that n-gram language models have?

Language Modeling
Question e)

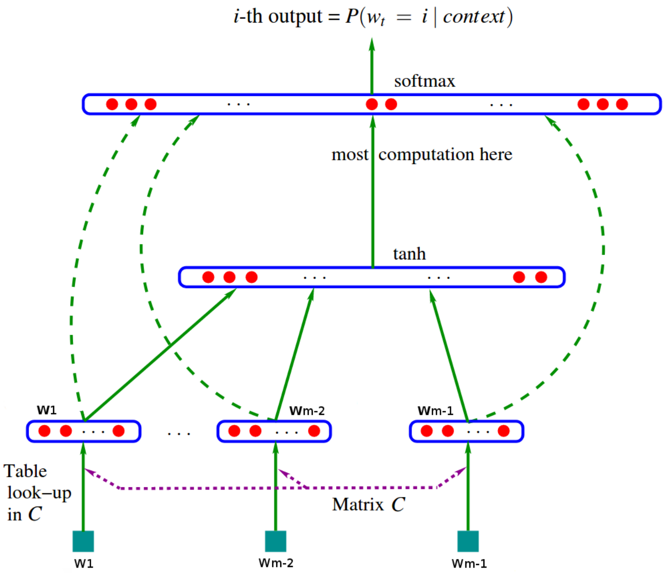
- **Question:** What are some of the problems that n-gram language models have?
- **Answer:**
  - Counting becomes expensive as n grows larger, because there are many more possible sequences.
  - In addition, storing all counts is expensive, and this count matrix is often sparse, as most possible sequences are not semantically relevant and thus never observed in a corpus.

Language Modeling
Question f)

- Neural language models can address some of the issues with n-gram models.
- Describe the neural language model designed by Bengio et al., which was discussed in the lecture.
- What kind of neural network was it? How many hidden layers did it have? How did they represent the input sequence? How did it compute the relevant probabilities for language modeling?
- How was it parameterized? And how many parameters does it have with a vocabulary V and word embedding of size d?
- For simplicity, you may ignore the optional linear layer and all biases.

Language Modeling

Answer f) (1)



Language Modeling

Answer f) (2)

- The proposed architecture was a feed-forward neural network, with an embedding layer as input, a single hidden layer with a *tanh* activation, and a softmax layer as output.
- The input sequence was represented by concatenating all embeddings of the given sequence.
- The probabilities were computed using a softmax layer, where the target was the correct word to predict.
- As parameters, let $\boldsymbol{W}_w \in \mathbb{R}^{|V| \times d}$ be the word embedding matrix in the input embedding layer, $\boldsymbol{W}_h \in \mathbb{R}^{(n \cdot d) \times h}$ be the weight matrix for the hidden layer with n being the (max) number of input tokens, and $\boldsymbol{W}_s \in \mathbb{R}^{h \times |V|}$ be the weight matrix for the softmax layer.
- Then, the model was parameterized by $\boldsymbol{\theta} = \{\boldsymbol{W}_w, \boldsymbol{W}_h, \boldsymbol{W}_s\}$.

Language Modeling

Question g)

- What is *self-supervision*? Describe the self-supervised training approach used by Bengio et al. to train their neural language model.

Language Modeling

Answer g)

- Self-supervision refers to the process of constructing labeled examples by hiding part of an input example and asking the model to predict the hidden part given the rest of it.
- For images, we can hide part of it and ask the model to predict it.
- For video, we can ask a model to predict a frame given previous $n - 1$ frames.
- In the context of language models, any sequence of length n in a corpus can be used to construct an example for predicting the n-th word given previous $n - 1$ words.
- This is what Bengio et al. did.

Language Modeling

Question h)

- What problems in n-gram models were no longer an issue with neural language models? And what problems still persisted with such models?
- Explain why the existing limitations are indeed limitations.

Language Modeling

Answer h)

- Neural language models such as the one described above did not need to store counts of n-grams, thus reducing the memory costs significantly.
- In addition, the sparsity problem is no longer relevant, as the model focuses on observed sequences only.
- However, a persisting problem is that the context window is still too small, and it can't be enlarged easily in such models.
- Why? Note that the input sequence is represented with a vector of size $d \cdot n$, where d is the size of the word embeddings and n the size of the input sequence.
- The hidden layer that takes this vector as input is parameterized by $\boldsymbol{W}_h \in \mathcal{R}^{(d \cdot n) \times h}$, for input sequence of size n.
- Thus, by increasing the window size, we increase the size of $\boldsymbol{W}_h$, thus making the model more difficult to train as the window grows larger.
- And even so, the problem of a fixed window persists.

Language Model Evaluation

Question a)

- **Question:**
  - LMs are often used as part of various *downstream tasks*.
  - Give two examples of downstream tasks that may use LMs in their pipeline, and describe how each may use LMs.

Language Model Evaluation

Question a)

- **Question:**
  - LMs are often used as part of various *downstream tasks*.
  - Give two examples of downstream tasks that may use LMs in their pipeline, and describe how each may use LMs.
- **Answer:**
  - Machine translation: a LM may determine which of n predicted phrases is more likely.
  - Speech recognition: given $n-1$ recognized spoken words, a LM may determine which of a small set of possible words is mostly likely to follow.
  - As we can see, LMs are very general tools in NLP.

Language Model Evaluation

Question b)

- LMs can be evaluated extrinsically and intrinsically. Describe the difference between intrinsic and extrinsic evaluation.
- Discuss some advantages and disadvantages of each.

Language Model Evaluation

Answer b)

- **Extrinsic:** evaluating model based on performance on a specific task, e.g. machine translation.
  - This downstream task must use a *downstream* model that uses a language model as part of it, i.e. a machine translation model.
  - PROs: we rely on meaningful performance on a task.
  - CONs: reported performance dependent on details of task-specific downstream model and evaluation, may not rely so much on language model performance.
  - Also, performance may not translate to different tasks.
- **Intrinsic:** task-neutral evaluation of a model.
  - PROs: no need for possibly expensive downstream pipeline.
  - CONs: performance on intrinsic task may not translate to some/all downstream tasks.

Language Model Evaluation

Question c)

- **Question:** Say you are tasked with designing the simplest intrinsic evaluation for language models using the basic machine learning principle of *held-out data* and a given text corpus. What sort of evaluation would you propose?

Language Model Evaluation

Question c)

- **Question:** Say you are tasked with designing the simplest intrinsic evaluation for language models using the basic machine learning principle of *held-out data* and a given text corpus. What sort of evaluation would you propose?
- **Answer:** We could split our data into training and test, use training data to learn our LM, then evaluate it by computing the probability of entire test corpus, i.e. computing the loss over by taking the entire test split as a single sequence.

Language Model Evaluation

Question d)

- **Question:**
  - The most common form of intrinsic evaluation of language models is computing its *perplexity* on held-out data.
  - Perplexity is defined as follows:

    $$ppl(w_1, w_2, \ldots, w_n) = p(w_1, w2, \ldots, w_n)^{-\frac{1}{n}},$$

    where p is the *likelihood* of held-out sequence $x_1, x_2, \ldots, x_n$.
  - Describe the intuition of perplexity from the perspective of the likelihood of the data. How does one change when the other changes?

Language Model Evaluation

Question d)

- **Question:**
  - The most common form of intrinsic evaluation of language models is computing its *perplexity* on held-out data.
  - Perplexity is defined as follows:

$$ppl(w_1, w_2, \ldots, w_n) = p(w_1, w2, \ldots, w_n)^{-\frac{1}{n}},$$

    where p is the *likelihood* of held-out sequence $x_1, x_2, \ldots, x_n$.
  - Describe the intuition of perplexity from the perspective of the likelihood of the data. How does one change when the other changes?
- **Answer:** The higher the likelihood, the lower the perplexity. Thus, we want to minimize perplexity.

Language Model Evaluation

Question e)

- Assume a uniform unigram model over some vocabulary V.
- What probability does such a model assign to each word in the vocabulary?
- Compute the likelihood of this model and use it to compute its perplexity over a held-out sequence of n words.

Language Model Evaluation

Answer e) (1)

- For simplicity, let $V = |V|$. Being unigram-based and uniform, this model assigns a probability of $\frac{1}{V}$ to each word.
- It's likelihood is given by:

$$\begin{aligned} l(w_1, w_2, \dots, w_n) &= p(w_1, w_2, \dots, w_n) \\ &= p(w_1)p(w_2)\dots p(w_n) \\ &= \left(\frac{1}{V}\right)^n \end{aligned}$$

It's perplexity is then:

$$\begin{aligned} ppl(w_1, w_2, \dots, w_n) &= l(w_1, w_2, \dots, w_n)^{-\frac{1}{n}} \\ &= \left(\left(\frac{1}{V}\right)^n\right)^{-\frac{1}{n}} \\ &= \left(\frac{1}{V}\right)^{-1} \\ &= V. \end{aligned}$$

Language Model Evaluation

Answer e) (2)

- Thus, the perplexity of LMs is often seen as proportional to the size of the vocabulary, where V (the size of the vocabulary) is informally seen as an lower-bound for the performance of a model, similar to comparing a classifier to a model tnat predicts a random class.
- In other words, for some vocabulary V, if the perplexity of a model is greater than $|V|$, we say it performs worse than a unigram model that assigns $1/|V|$ probability to each word.
- For example, on the *Penn Treebank* dataset, a 5-gram model achieves a perplexity of 141, but this can be improved to 80 using an RNNs with LSTMs.
- In both cases, this is a much lower number that the size of the vocabulary.

Recurrent Neural Networks

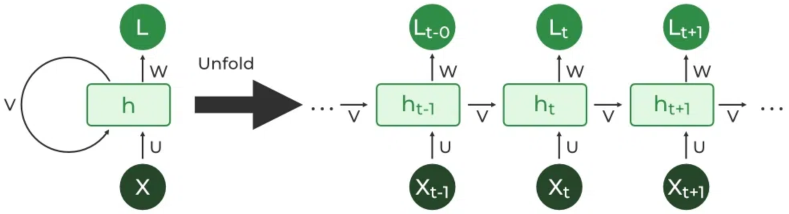
Question a)

- Describe the components of an RNN and discuss how it differs from fully-connected neural networks.
- What are the parameters of an RNN?
- Are RNNs deep? In what way(s)?

Recurrent Neural Networks

Answer a) (1)

- RNNs are composed of (possibly parameterized) units that are shared across time-steps, i.e. across elements of a sequence.
- Let's separate these components down into input units, hidden units, and output units.



Recurrent Neural Networks

Answer a) (2)

- **Input units:** At time-step t, a RNN takes as input $\boldsymbol{x}_t$, which is a representation of the t-th element of a given sequence, e.g. a word embedding. It then transforms it via matrix $\boldsymbol{U}$, and passes this transformed input to hidden state $\boldsymbol{h}_t$.

Recurrent Neural Networks

Answer a) (3)

- **Hidden units:** In addition to the transformed input $\boldsymbol{x}_t$, a hidden state $\boldsymbol{h}_t$ takes as additional input $\boldsymbol{h}_{t-1}$, i.e. the hidden state at previous time-step $t-1$, but transformed by matrix $\boldsymbol{V}$.
- This connection between the same hidden state across time-steps is the recurrent connection that gives the network its name (which comes from recurrent functions).
- Thus, $\boldsymbol{h}_t$ takes as input $\boldsymbol{U}\boldsymbol{x}_t$ and $\boldsymbol{V}\boldsymbol{h}_{t-1}$, combines them via some operation (usually addition), and then applies a (possibly non-linear) activation function f, before passing this result to the hidden state in time-step $t+1$, i.e. $\boldsymbol{h}_{t+1}$.
- In other words, $\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1})$.
- By sharing parameters $\boldsymbol{U}, \boldsymbol{V}$ and passing hidden state $\boldsymbol{h}_i$ forward, the intuition is that, at any given time-step t, $\boldsymbol{h}_t$ encodes what the model has "seen" in the sequence so far, i.e. all elements up until t.

Recurrent Neural Networks

Answer a) (4)

- **Output units:** Optionally, the hidden state may also be passed to an output unit $\boldsymbol{L}_t$, but transformed via some matrix $\boldsymbol{W}$ and a (potentially non-linear) activation function g.
- That is, $\boldsymbol{L}_t = g(\boldsymbol{W}\boldsymbol{h}_t)$.
- Whether we have an output unit per time-step or not depends on the task at hand (more in the next question).

Recurrent Neural Networks

Answer a) (5)

- RNNs are different from FNNs in that they share parameters across time-steps, and are thus deep in time.
- This is different from fully-connected networks in that those typically use different weight matrices in each hidden layer.
- However, RNNs can also be stacked to become deep in the same sense as fully-connected neural networks.
- That is, an RNN layer may produce an output for each input element, thus producing an output sequence, which can in turn be used as input for another RNN layer.
- In such cases, parameters are not shared across layers.

Recurrent Neural Networks

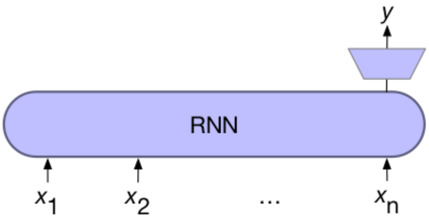
Question b)

- How can a RNN be used for a sequence classification task?
- And what about for a task where each token in the sequence requires a predicted label, e.g. part-of-speech (POS) tagging?
- Describe the input, output and general architecture of the model.

Recurrent Neural Networks

Answer b) (1)

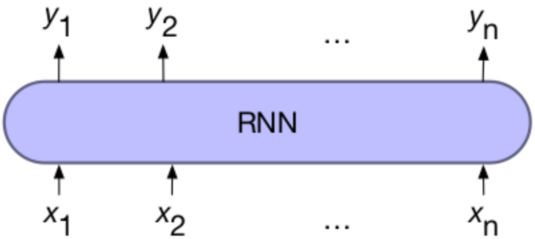
- In *sequence classification*: single prediction per sequence.
- E.g. in sentiment classification, we may predict whether a given sentence describes a positive or negative sentiment, i.e. binary classification.
- Thus, we do not need an output unit per element in the input unit. Instead, we may produce a single output input based on the hidden state of the final time-step.
- This output may be passed by a softmax function to produce a relevant probability vector, or for binary classification, a logistic function.



Recurrent Neural Networks

Answer b) (2)

- Conversely, requiring a label for each time-step is known as *sequence labeling*.
- For example, in POS tagging, we want to determine whether each word in a sentence is a verb, noun, etc.
- Here, the RNN should produce an output at each time-step, as illustrated by the image below.



- Each output unit includes softmax function for classification.
- In both cases, the inputs and hidden states are as described in answer to question (a) above.

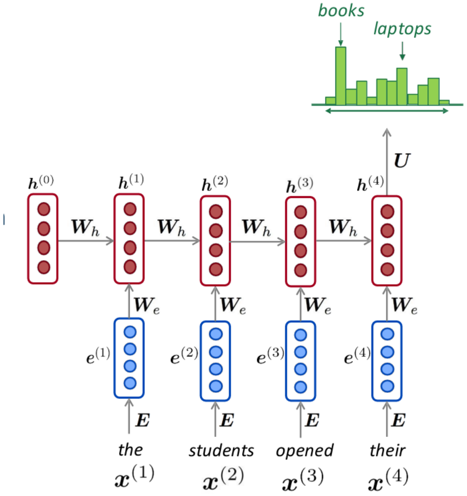
Recurrent Neural Networks

Question c)

- How can RNNs be used for language modeling? Describe the model's architecture.
- What is the fundamental problem that RNNs solve over fully-connected neural networks for language modeling?

Recurrent Neural Networks

Answer c) (1)



Recurrent Neural Networks

Answer c) (2)

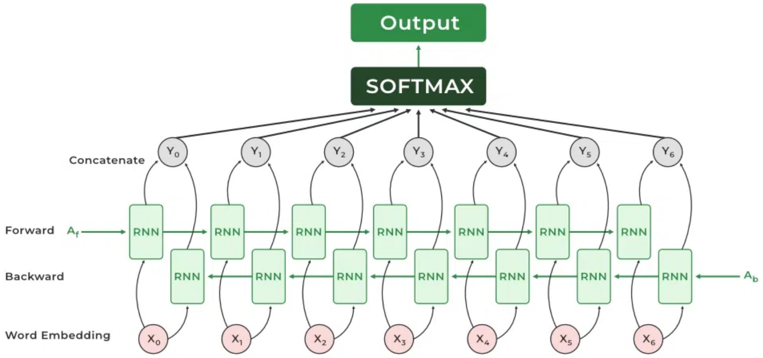
- Language models predict the next word given a sequence of n previous words.
- This is done by producing an output at each time-step given the corresponding input element and the hidden state at the previous time-step.
- Thus, unlike LMs implemented with fully-connected networks, such as the model from Bengio et al. (2003), here we do not have a maximum length of the input sequence that we accept, but can take arbitrarily long input sequence.
- In addition, unlike n-gram models, predictions made by RNN-based language models are not limited by any fixed number of previous words, since the hidden state can *in principle* encode all the sequence seen so far.

Recurrent Neural Networks

Question d)

- What are bidirectional RNNs? What are their advantages and disadvantages compared to standard (unidimensional) RNNs?

Recurrent Neural Networks
Answer d) (1)


Output
SOFTMAX
Concatenate
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6
Forward
A_f
RNN
Backward
A_b
Word Embedding
X_0 X_1 X_2 X_3 X_4 X_5 X_6

- Image source

Recurrent Neural Networks

Answer d) (2)

- Bidirectional RNNs use an additional hidden state to process a given sequence from right to left.
- These hidden states are passed "backward" from the end of the sequence to the beginning, thus encoding it in the opposite direction.
- This allows the model to, at time-step t, encode information about both previous and subsequent elements in the input sequence.
- Thus, the model has more context to make predictions at any given time-step.
- A disadvantage is that this is more costly to train, and more importantly, such a model cannot be used to make real-time predictions, since we do not have information about future time-steps.

Recurrent Neural Networks

Answer d) (3)

- Perhaps more importantly, isn't it *cheating* to use information about future time-steps?
- Well, that depends on the application.
- If the information is available at inference time, then why not use it?
- For example, in machine translation, we are usually given an entire sequence to translate, so a model can be trained to read the entire sequence in both directions before producing a sequence in the target language.
- This intuition extends to the information we give to any model during training, independent of its architecture.
- For example, when the goal is to learn generally useful word representations, then why not use information in both directions to make better use of the context of words in the training set?
- This is what the transformer-based model BERT does.

Recurrent Neural Networks

Answer d) (3)

- Conversely, when predicting the next word in a sequence, models typically have access only to previous words.
- Thus, it would not make sense to train models to use information about future time-steps to make predictions, as this information is not available at inference time.

Recurrent Neural Networks

Question e)

- Describe the main problem with RNNs and how LSTMs were designed to address this issue.
- Give a high-level intuition for each of the gates in an LSTM unit.

Recurrent Neural Networks

Answer e) (1)

- The main problem with RNN-based models is that the hiddens state needs to encode the entirety of a sequence.
- In practice, it's often the case the the hidden states "remembers" information mostly aobut the most recently seen elements, and not those seen far into the past.
- This is mostly due to the vanishing gradient problem, which describes the impact that operators have over gradient information over a long period of time.
- Specifically, if an operator makes a gradient smaller, applying the same operator multiple times would continuously reduce this gradient.

Recurrent Neural Networks

Answer e) (2)

- LSTM units were designed to address this issue by allowing the model to control what to remember and what to forget about the information it has seen.
- Thus, models may realize during training that, in order to reduce the loss, it's more convenient to retain information further in the past instead of more recently seen information.
- This is not possible in vanilla RNNs.
- LSTM units use parameterized gated mechanisms to allow the model to control the flow of information across time-steps.
- Specifically, it uses the following gates:
  - **Forget Gate:** allows model to delete information that it has seen but no longer needs.
  - **Input/Add Gate:** allows model to select what information from the current input and hidden state to use.
  - **Output Gate:** allows the model to separate information needed to produce current output from what is needed to pass to future time-steps.

Recurrent Neural Networks

Question f)

- Describe an encoder-decoder architecture and explain how it can be used for machine translation, i.e. the task of predicting a sequence of words in a target language given a sequence of words in a source language.
- When implementing such an architecture with RNNs, what is the input and output of the hidden state in the encoder at each time step? And in the decoder?

Recurrent Neural Networks

Answer f) (1)

- An encoder-decoder architecture is made up of three components: (i) an encoder which takes as input a given source sequence, (ii) a context vector produced by the encoder that represents the input sequence, and (iii) a decoder, which takes as input the context vector and produces an output sequence.
- Let's describe the encoder and decoder in more detail.

Recurrent Neural Networks

Answer f) (2)

- **Encoder:** In machine translation, the encoder takes as input the source sequence, and produces a context vector (usually the hidden state of the last time-step).
- This encoder produces no other outputs and in general can be any architecture that makes as much use of the input data as possible.
- RNN-based encoders are typically stacked bidirectional RNNs with LSTM units (biLSTMs for short), but this can also be done with other types of architectures, e.g. transformers.

Recurrent Neural Networks

Answer f) (3)

- **Decoder:** The decoder takes as input the context vector and a “beginning of sentence” tag (in machine translation, this tag is used to separate source from target sentences pairs shown to the model).
- Using this, it produces a hidden state and an output word, *both of which* are then used as input for the hidden state of the decoder in the next time step.
- In addition, it’s common to use the context vector produced by the encoder as input in each time-step of the decoder, to preserve this information.

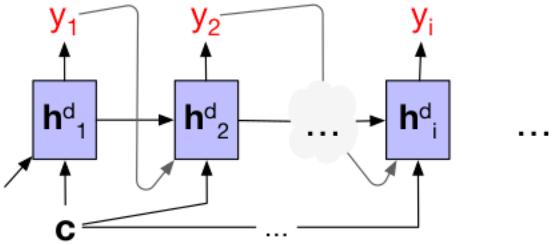
Recurrent Neural Networks

Answer f) (4)

- Formally, the hidden state of the decoder at time-step t is given by:

$$\boldsymbol{h}_t^d = g(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-1}^d, \boldsymbol{c}),$$

  where $\boldsymbol{y}_{t-1}$ is the embedding of word produce by decoder in the previous time-step, and $\boldsymbol{c}$ is the context vector produced by the encoder.
- This decoder architecture is illustrated in the image below.



Sampling for Text Generation

Context

- An important aspect of using language models to (autoregressively) generate text is the step of *sampling* the next word from the distribution over words in our vocabulary that is produced by a language model.
- In this task, we discuss some basic methods for sampling from this distribution to generate text.

Sampling for Text Generation

Question a)

- What is autoregressive text generation?
- What are its advantages compared non-autoregressive text generation?
- And its disadvantages?

Sampling for Text Generation

Answer a)

- **Autoregressive generation:** generate sequence of word/tokens by repeatedly sampling the next word/token conditioned on previously chosen words/tokens.

Sampling for Text Generation

Answer a)

- **Autoregressive generation:** generate sequence of word/tokens by repeatedly sampling the next word/token conditioned on previously chosen words/tokens.
- **Advantages:** can capture dependencies between words/tokens in a sequence.

Sampling for Text Generation

Answer a)

- **Autoregressive generation:** generate sequence of word/tokens by repeatedly sampling the next word/token conditioned on previously chosen words/tokens.
- **Advantages:** can capture dependencies between words/tokens in a sequence.
- **Disadvantages:** can be slow to generate long sequences, the longer the range of dependencies that need to be captured, the higher the cost of sampling. In other words, non-autoregressive generation is cheaper/faster.

Sampling for Text Generation

Question b)

- In a machine translation setting, let $X = \{x_1, x_2, \ldots, x_n\}$ be the input sequence in some source language, $Y = \{y_1, y_2, \ldots, y_m\}$ the output sequence in some target language, and $p(y_t | X, \boldsymbol{\theta})$ the distribution for the t-th word in a non-autoregressive setting using a model parameterized by $\boldsymbol{\theta}$.
- Give the corresponding expression for p in an autoregressive model.

Sampling for Text Generation

Question b)

- In a machine translation setting, let $X = \{x_1, x_2, \dots, x_n\}$ be the input sequence in some source language, $Y = \{y_1, y_2, \dots, y_m\}$ the output sequence in some target language, and $p(y_t|X, \boldsymbol{\theta})$ the distribution for the t-th word in a non-autoregressive setting using a model parameterized by $\boldsymbol{\theta}$.
- Give the corresponding expression for p in an autoregressive model.
- **Answer:**$p(y_t|y_{<t}, X, \theta)$

Sampling for Text Generation
Question c)

- Let sample(p) be a function to sample a word from the autoregressive expression for p you provided in the previous question, and $< s >, < /s >$ be the start-of-sentence and end-of-sequence tokens, respectively.
- Write an algorithm in pseudocode to autoregressively generate a sequence of words of arbitrary length.

Sampling for Text Generation

Answer c)

```
$w \leftarrow \text{sample}(p(w_i | < s >))$
out $\leftarrow [w]$
**while** $w \neq < /s >$ **do**
    $w \leftarrow \text{sample}(p(w_i | out))$
    out $\leftarrow$ out $+[w]$
**end while**
**return** out
```

Sampling for Text Generation
Question c)

- The most straightforward way to implement the *sample* function used in the previous question is by choosing the word with the highest probability.
- This is known as **greedy sampling**.
- Can you think of advantages and disadvantages to such an approach?
- Use the following two incomplete sentences to reason about this.
  1. The capital of Germany is ...
  2. The title of my talk is ...

Sampling for Text Generation

Question c)

- **Advantages:** straightforward to understand/implement.
- **Important disadvantage:** can lead to generation of text that is predictable and repetitive.
- In fact, given a model and the same input sequence, greedy sampling should deterministically produce the same output sequence.
- This is convenient when we want to predict facts that do not change, e.g. the answer to "What is the capital of Germany"?
- But this is an undesired property when we want to generate text that is more original/creative, such as what should follow after "The title of my talk is"

Sampling for Text Generation

Question d)

- Another way to implement the *sample* function is by **randomly sampling** from p.
- This means we sample a word w with probability $p(w)$, so that words with higher probability get sampled more often, while words with lower probability get sampled less often.
- Can you think of advantages and disadvantages to such an approach?
- **Hint:** the size of the vocabulary is often very large, and for a given input sequence, the distribution over words is very skewed.

Sampling for Text Generation

Answer d)

- Random sampling can lead to more diverse and creative text, as it allows the model to sample from the entire vocabulary.
- However, it can also lead to less coherent text, as the model may sample words that are less likely to follow the previous words.
- In addition, the size of the vocabulary is often very large, and for a given input sequence, the distribution over words is very skewed.
- This means that the model may sample from words that are very rare, and thus produce text that is less coherent.

Sampling for Text Generation

Question e)

- Can you think of an approach that can be used to sample from the distribution over words in a more balanced way compared to both *greedy* and *random* sampling?
- What are its advantages and disadvantages compared to the approaches described in the previous questions?

Sampling for Text Generation

Answer e) (1)

- The answer is **top-k sampling**.
- This is a generalization of *greedy* sampling, where we now sample from the set of k words with the most probability.
- When $k = 1$, we have greedy sampling.
- When $k = |V|$, we have random sampling.
- Generally, for values of $k > 1$, we allow the model to generate words that are not so predictible, but still probable enough that the resulting text can be of high-quality.
- **Important disadvantage:** k is a fixed value, but the distributions over words produced by a model change for given inputs.
- E.g. when the input sequence is "The capital of Germany is", most of the probability mass in the resulting distribution would ideally be in the Berlin token (or combination of tokens), which is a (proper) noun.

Sampling for Text Generation

Answer e) (2)

- But when the input sequence is “The title of my talk”, the probability mass would most likely be in the token “is” or similar verbs.
- To address this issue, **top-p sampling** was proposed, where we sample from the words that add up to the top *p* probability mass in the distribution.
- That way, we can sample from a variable number of words given different input sequences.
- A good model may indeed learn to put most of the mass in the answer to a factual question, e.g. Berlin is the capital of Germany, or more generally distribute it more evenly among several probable words in other settings.

Sampling for Text Generation

Question f)

- *Temperature* sampling is another approach, where instead of truncating a model distribution, we reshape it.
- It is defined in the context of softmax distributions as follows:

$$\boldsymbol{y} = \text{softmax}\left(\frac{\boldsymbol{u}}{\tau}\right),$$

  where $\boldsymbol{u}$ is the unnormalized distribution over words, i.e. a vector of logits, τ is the temperature parameter, and we do an element-wise division with it.
- Describe the impact that different values of τ have over the resulting model distribution that we will then use to sample the next word from.
- Specifically, how does the distribution change as τ is less than 1 and approaches zero? And as it approaches infinity?

Sampling for Text Generation

Answer f)

- As *tau* approaches zero and is less than 1, it will increase the values of all logits, e.g. $12 < 12/0.5 < 12/0.3$.
- This will result in increased probability mass on words that already have high mass, and decreased probability mass on words that had low mass in the first place.
- Thus, as we decrease the temperature, most of the probability mass in the distribution concentrates around the most probable words.
- Conversely, if we increase the temperature to values greater than 1 and beyond, the probability mass will be more evenly distributed across all words, as the operation will decrease values of all logits.
- See some animations here.