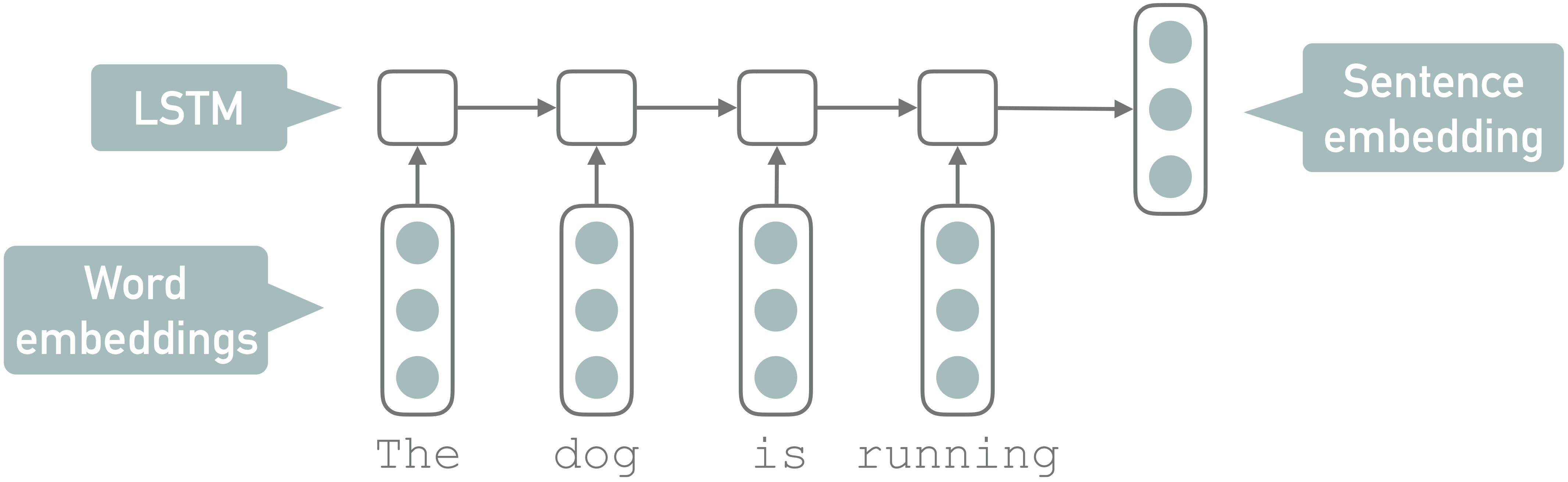
DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 4: Word Representation

Caio Corro

NEURAL NETWORK AND TEXTUAL INPUT



But…

- Most words rarely appear in the training data:
  is a few update enough to tune the word embedding?
- Data annotation is expensive, and therefore limited:
  How to generalize to words unseen in the training data?

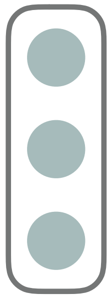
Solutions

- Special unknown word embedding
- Sub-word models (e.g. character based representation)
- Pre-trained word representation (i.e. use large unlabeled dataset to train word embeddings)

UNKNOWN WORD EMBEDDING 1/2

Main idea

- Add a special <UNK> word your embedding table
- At test time, map word unseen in the training data to <UNK>



Training the <UNK> word embedding

- Replace all rare words in the training data with <UNK>
  (e.g. all words occurring less the 2 times)
- Replace words with <UNK> with a given probability at each update: **word dropout**

$$p(w) = \frac{1}{1 + n.\ occurences\ of\ w\ in\ train\ data}$$

Frequent words are replaced less often

UNKNOWN WORD EMBEDDING 2/2

```
def forward(self, inputs):
  batch_words = [t["words"].to(self.word_embs.weight.device) for t in inputs]

  if self.training and self.word_dropout:
    batch_words = [words.clone() for words in batch_words]

    for b in range(len(batch_words)):
      for i in range(batch_words[b].size()[0]):

        if np.random.rand() < 1 / (1 + self.word_counts[batch_words[b][i].item()]):
          batch_words[b][i] = self.unk_word_index

  padded_inputs = torch.nn.utils.rnn.pad_sequence(
    batch_words,
    batch_first=True,
    padding_value=self.word_embs.padding_idx
  )
```

Move input tensors to the same device as the embedding table

Word dropout if module is in training

Copy because it will be updated

For each sentence in the batch

For each word in the sentence

Replace with given probability

Build batched input

SUB-WORD MODELS 1/3

Morphological rich languages

- **Analytic languages** (e.g. English): morphology plays plays a relatively modest role.

Plural in English: *« dog/dogs »*

- **Synthetic languages** (e.g. German): Morphology plays an important role, therefore the number of words can huge.

In Sumerian the following differences are encoded
via morphological inflection: *« I went/he went/he went to him »*

SUB-WORD MODELS 1/3

Morphological rich languages

- **Analytic languages** (e.g. English): morphology plays plays a relatively modest role.

Plural in English: *« dog/dogs »*

- **Synthetic languages** (e.g. German): Morphology plays an important role, therefore the number of words can huge.

In Sumerian the following differences are encoded
via morphological inflection: *« I went/he went/he went to him »*

Preprocessing step

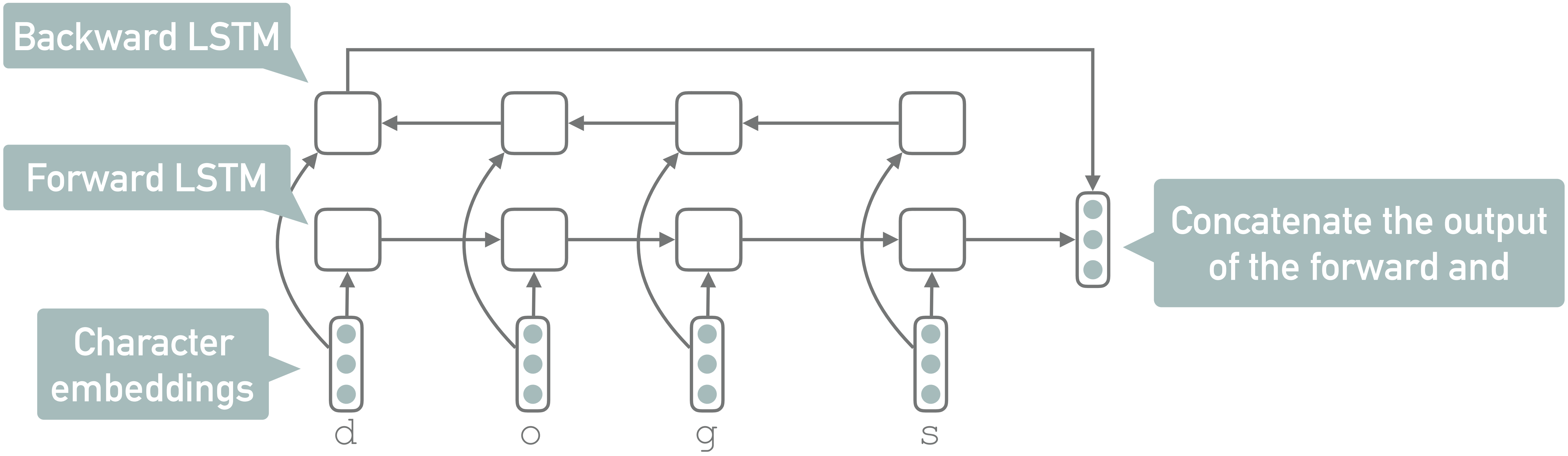
We don't like preprocessing steps
in deep learning!

Extract features from word before building word embeddings:

- Lemma/stem
- Inflectional morphemes (e.g. « s » indicating plural, « ed » indicating past tense, etc)

and combine such information.

SUB-WORD MODELS 2/3



Character LSTM

Or character CNN

- Character embeddings as input
- Concatenation of outputs of forward and backward LSTM as output

In practice

- Add <BOS>and <EOW> embeddings to each input (usually done during preprocessing)
- Concatenate word + character LSTM output

SUB-WORD MODELS 3/3

```
class CharacterEncoder(nn.Module):
    def __init__(self, voc_size):
        super(CharacterEncoder, self).__init__()

        self.embeddings = nn.Embedding(voc_size+1, 100, padding_idx=voc_size)
        self.lstm = nn.LSTM(100, 125, bidirectional=True, batch_first=True)
```

Add a padding word to the vocabulary so we can batch words of different size

Dim of input embeddings

LSTM hidden rear dim

BiLSTM

First dimension is batch

SUB-WORD MODELS 3/3

```
class CharacterEncoder(nn.Module):
    def __init__(self, voc_size):
        super(CharacterEncoder, self).__init__()

        self.embeddings = nn.Embedding(voc_size+1, 100, padding_idx=voc_size)
        self.lstm = nn.LSTM(100, 125, bidirecti

    def forward(self, input):
        padded_inputs = torch.nn.utils.rnn.pad_sequence(
            input,
            batch_first=True,
            padding_value=self.embeddings.padding_idx
        )
```

List of tensors

Input words can be of different lengths so we need to pad them

SUB-WORD MODELS 3/3

```
class CharacterEncoder(nn.Module):
    def __init__(self, voc_size):
        super(CharacterEncoder, self).__init__()

        self.embeddings = nn.Embedding(voc_size+1, 100, padding_idx=voc_size)
        self.lstm = nn.LSTM(100, 125, bidirectional=True, batch_first=True)


    def forward(self, input):
        padded_inputs = torch.nn.utils.rnn.pad_sequence(
            input,
            batch_first=True,
            padding_value=self.
        )

        lengths = [len(i) for i in input]
        packed_embs = torch.nn.utils.rnn.pack_padded_sequence(
            emb_inputs,
            lengths,
            batch_first=True,
            enforce_sorted=False
        )
```

Retrieve embeddings
+ store batching information (length of each word)
+ other stuff under the hood

Input is not sorted by length

SUB-WORD MODELS 3/3

```
class CharacterEncoder(nn.Module):
    def __init__(self, voc_size):
        super(CharacterEncoder, self).__init__()

        self.embeddings = nn.Embedding(voc_size+1, 100, padding_idx=voc_size)
        self.lstm = nn.LSTM(100, 125, bidirectional=True, batch_first=True)


    def forward(self, input):
        padded_inputs = torch.nn.utils.rnn.pad_sequence(
            input,
            batch_first=True,
            padding_value=self.
        )

        lengths = [len(i) for i in input]
        packed_embs = torch.nn.utils.rnn.pack_padded_sequence(
            emb_inputs,
            lengths,
            batch_first=True,
            enforce_sorted=False
        )
        _, (endpoints, _) = self.lstm(packed_embs)
```

Retrieve embeddings
+ store batching information (length of each word)
+ other stuff under the hood

Run the LSTM over batched inputs

SUB-WORD MODELS 3/3

```
class CharacterEncoder(nn.Module):
    def __init__(self, voc_size):
        super(CharacterEncoder, self).__init__()

        self.embeddings = nn.Embedding(voc_size+1, 100, padding_idx=voc_size)
        self.lstm = nn.LSTM(100, 125, bidirectional=True, batch_first=True)


    def forward(self, input):
        padded_inputs = torch.nn.utils.rnn.pad_sequence(
            input,
            batch_first=True,
            padding_value=self.embeddings.padding_idx
        )

        lengths = [len(i) for i in input]
        packed_embs = torch.nn.utils.rnn.pack_padded_sequence(
            emb_inputs,
            lengths,
            batch_first=True,
            enforce_sorted=False
        )
        _, (endpoints, _) = self.lstm(packed_embs)

        token_repr = torch.cat([endpoints[0], endpoints[1]], dim=1)
        return token_repr
```

Concatenate forward and backward outputs

PRE-TRAINED WORD EMBEDDINGS

Semi-supervised learning

- Can we use large corpus of unlabeled text to improve to performance of a model?
- Task specific

Pre-trained word embeddings

Task agnostic word representations that can be used to « bootstrap » a neural network

- Type of models: count and predict
- Context: non-contextual and contextual embeddings

Word embeddings only

Also include RNN/Attention layers

Evaluation

- Intrinsic evaluation: what does the learned representation says about words?
- Extrinsic evaluation: does this representation improve results on a downstream task?

DISTRIBUTIONAL SEMANTICS AND COUNT MODELS

DISTRIBUTIONAL SEMANTICS

Meaning

- Signifier: how it is represented using symbols (word, sentences)
- Signified: what does it express

Use theory of meaning

The meaning of a word is defined by the context where it is used,
i.e. similar words are used in similar contexts.

Geometric approach to word meaning

- Word meanings (i.e. context) encoded in vectors
- Semantic relatedness given by distance metrics
- Word composition via vector operations,
  i.e. build a vector for « red car » by combining word vectors « red » and « car »

Problematic for non-compositional expressions, e.g. « rock and roll »

WORD CO-OCCURENCE MATRICES 1/2

- Each line correspond to the « vector » of each word in the vocabulary
- Each column track to co-occurence count with other words

Construction

- Initialize the matrix to 0

The matrix will be very sparse!

- For each word in each sentence in a large corpus of text, increment the cell value of observed context word

« The **dog** is eating. »

Context word

Vocabulary

	dog	cat	eating	sleeping	the	is
dog			+1		+1	+1
cat						
eating						
sleeping						
the						
is						

WORD CO-OCCURENCE MATRICES 2/2

Counting methods

- Window restriction: don't look at the whole sentence but only a limited number of surrounding words
- Syntactic restriction: use syntactic relations instead of fixed size window

\+ count transformation, e.g. apply (positive) Pointwise Mutual Information (PMI)

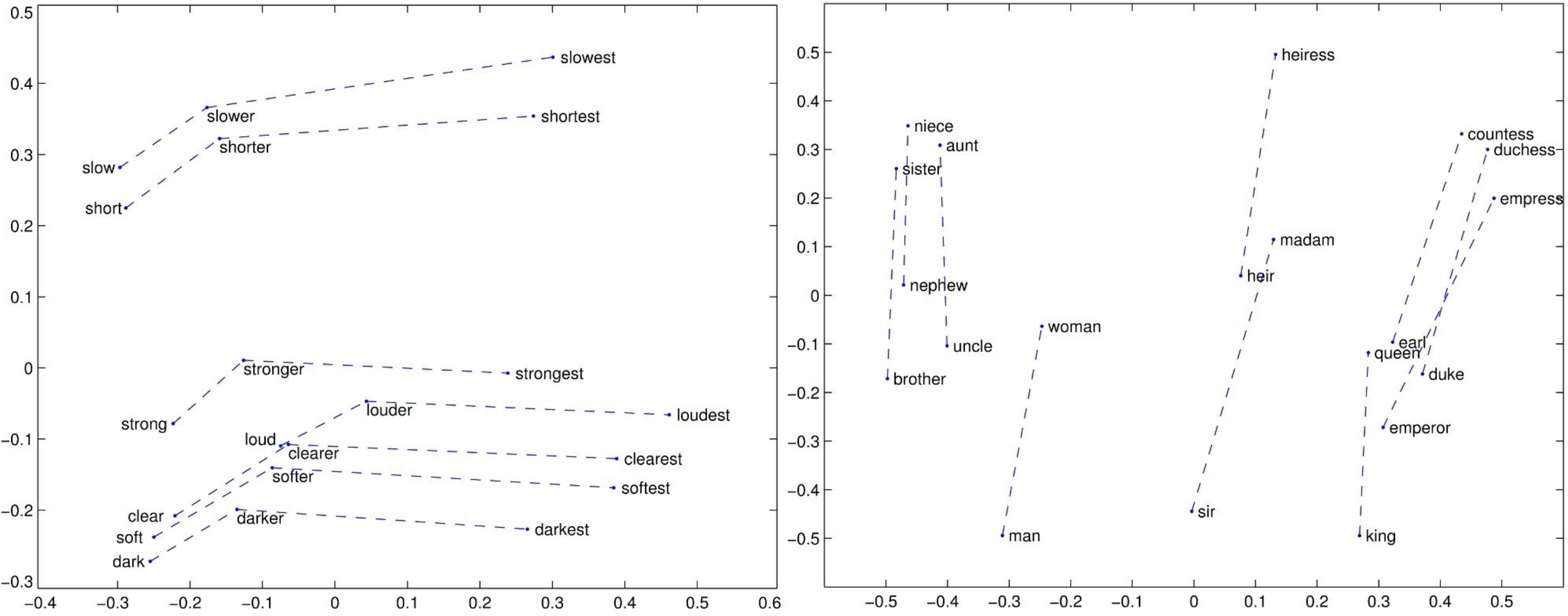
Many words co-occur frequently and do not carry useful semantic information, e.g. « the », « a », « an »,…

Singular Value Decomposition (SVD)

- Co-occurence matrices are very big, sparse and noisy
- Use (truncated) SVD to keep only the « main directions » and reduce dimensionality => the resulting matrix is dense

EXAMPLE

[Sebastian Ruder]



MULTI-LINGUAL WORD EMBEDDING

[Ruder et al., 2019]

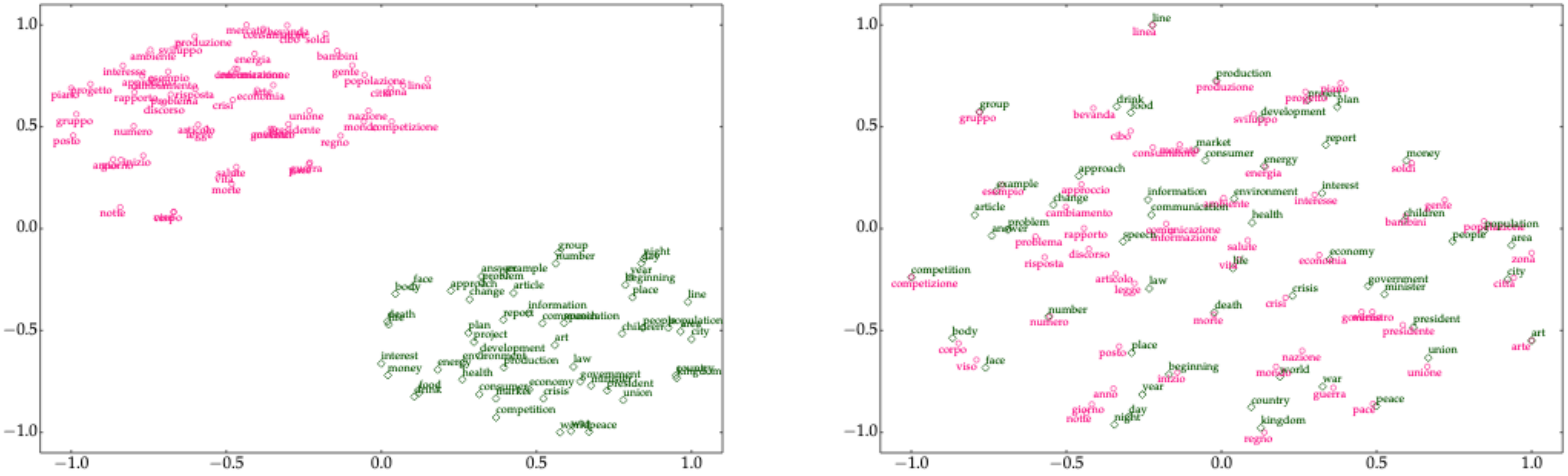


Figure 1: Unaligned monolingual word embeddings (left) and word embeddings projected into a joint cross-lingual embedding space (right). Embeddings are visualized with t-SNE.

PREDICT MODELS

PREDICT MODELS

[Mikolov et al., 2017]

Main idea

- Train a (shallow) neural network for a simple word prediction task
- Use the learned input embeddings as pre-trained embeddings

Models

- Continuous Bag of Words (CBOW): predict a word given its context
- Skip-grams: predict the context of a word

Relation with language models

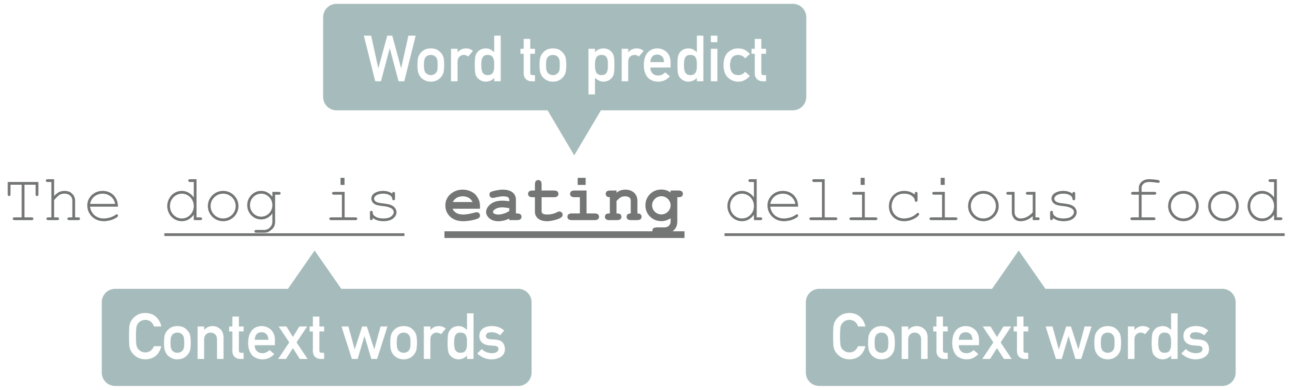
- We could use MLP or RNN language model to pre-train embeddings too
- But CBOW and Skip-grams are way faster!

CONTINUOUS BAG-OF-WORDS (CBOW) 1/2

[Mikolov et al., 2017]

Operation principle

- Predict a word given its context
- The context is a limited window around (left and right) the word to predict
- There is no word order information (i.e. bag-of-word)



Word to predict

`The dog is` **`eating`** `delicious food`

Context words

Context words

Objective

- Maximize the probability of the probability of the word to predict
  => Negative log likelihood loss

CONTINUOUS BAG-OF-WORDS (CBOW) 2/2

[Mikolov et al., 2017]

Word to predict

`The dog is` **`eating`** `delicious food`

Context words

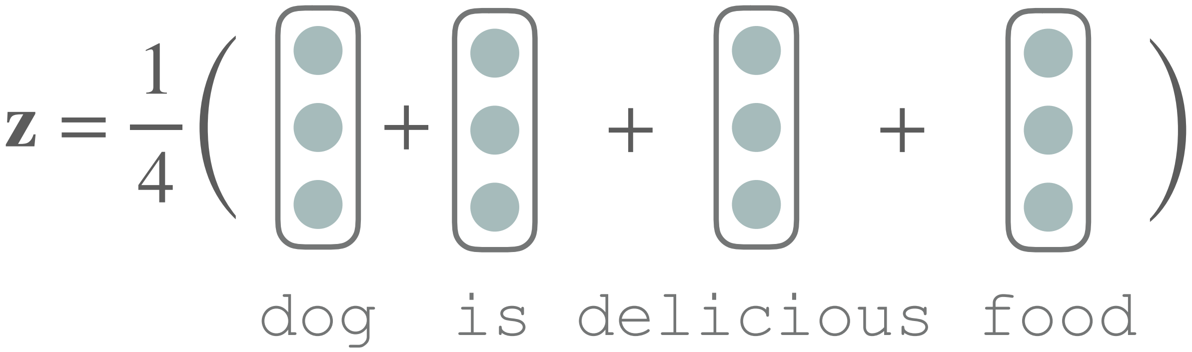
Context words

$|V|$: vocabulary size

h : embedding size

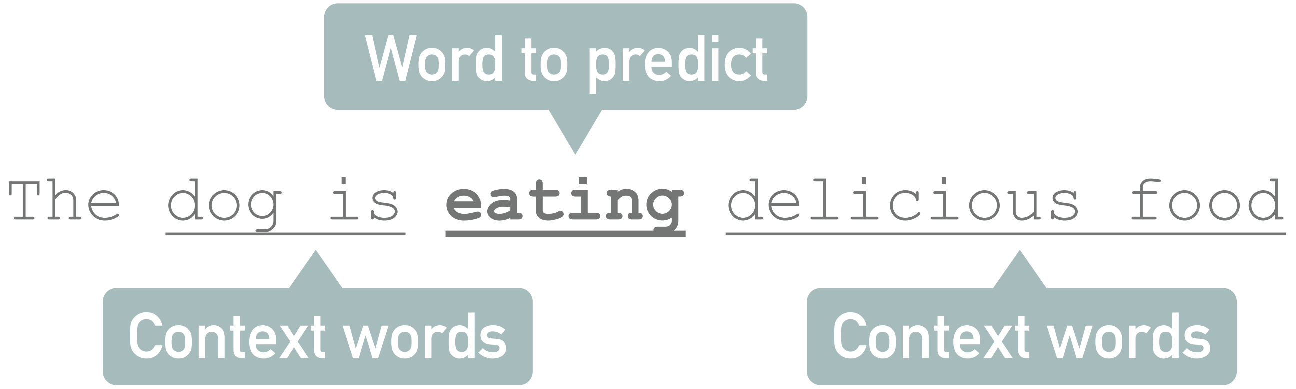
$\mathbf{E} \in \mathbb{R}^{h \times |V|}$: embedding table

$\mathbf{z} \in \mathbb{R}^{h}$: hidden representation



CONTINUOUS BAG-OF-WORDS (CBOW) 2/2

[Mikolov et al., 2017]



$|V|$: vocabulary size

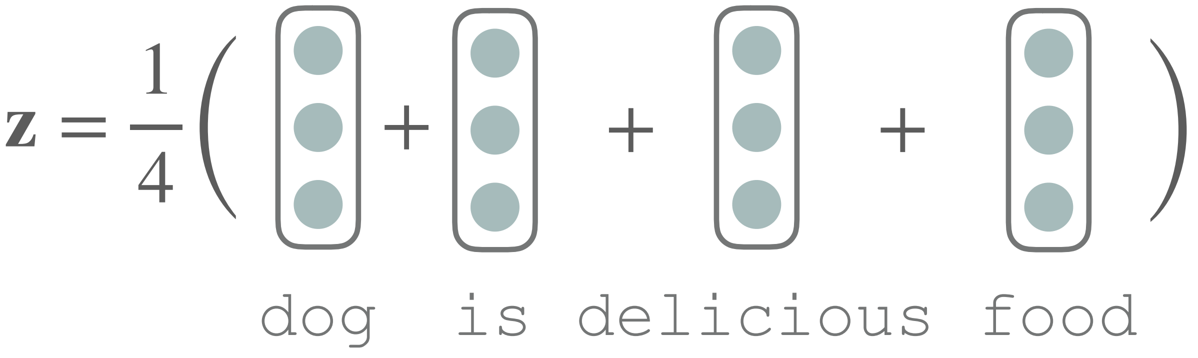
h : embedding size

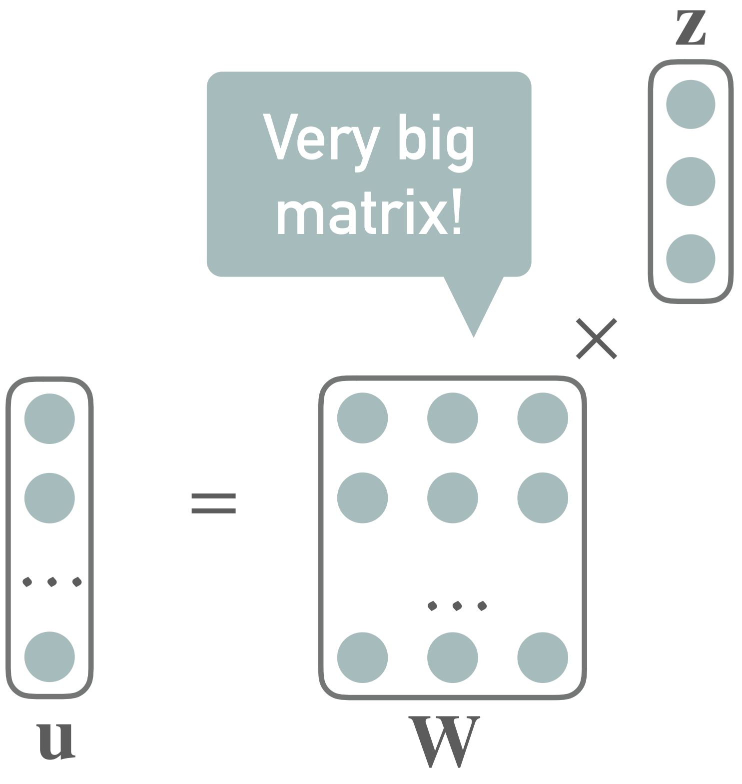
$\mathbf{E} \in \mathbb{R}^{h \times |V|}$: embedding table

$\mathbf{z} \in \mathbb{R}^{h}$: hidden representation

$\mathbf{W} \in \mathbb{R}^{|V| \times h}$: output projection (also called output embedding table)

$\mathbf{u} \in \mathbb{R}^{|V|}$: output weights



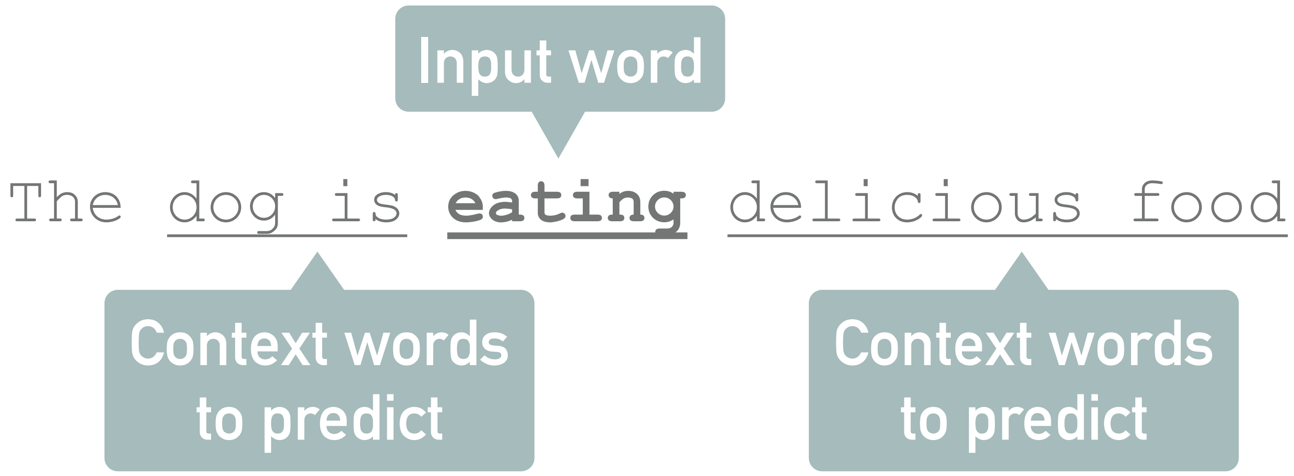


SKIP-GRAMS 1/2

[Mikolov et al., 2017]

Operation principle

- Predict the context given a word
- The context is a limited window around (left and right) the word to predict



Objective

- Maximize the probability of the probability of context words
  => Negative log likelihood loss for each context word

SKIP-GRAMS 2/2

[Mikolov et al., 2017]

Input word

`The dog is eating delicious food`

Context words to predict

Context words to predict

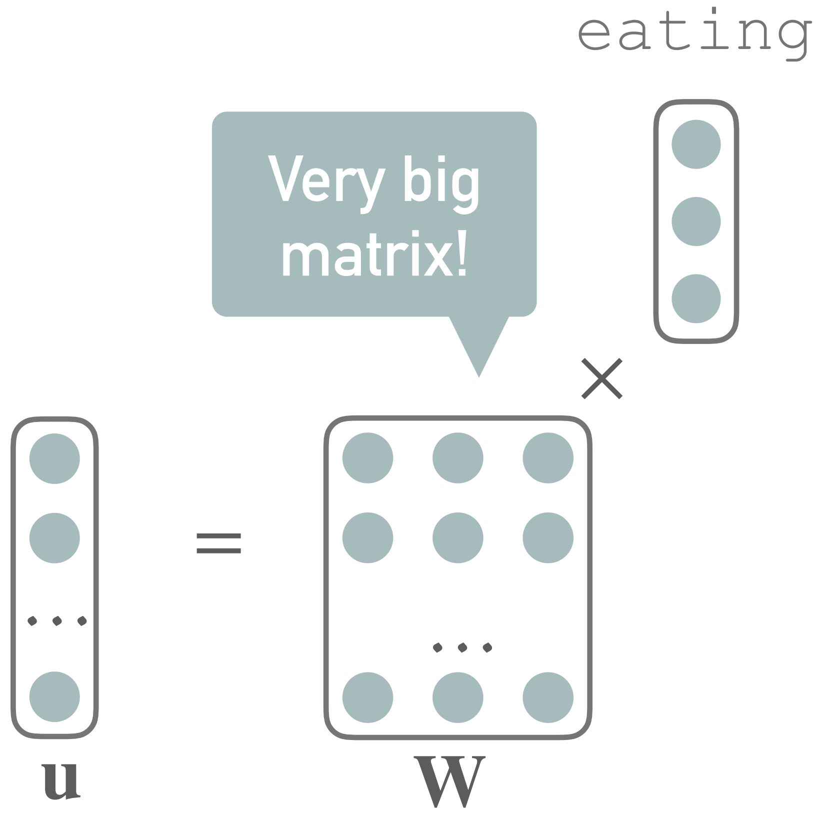
$|V|$: vocabulary size

h : embedding size

$\mathbf{E} \in \mathbb{R}^{h \times |V|}$: embedding table

$\mathbf{W} \in \mathbb{R}^{|V| \times h}$: output projection
(also called output embedding table)

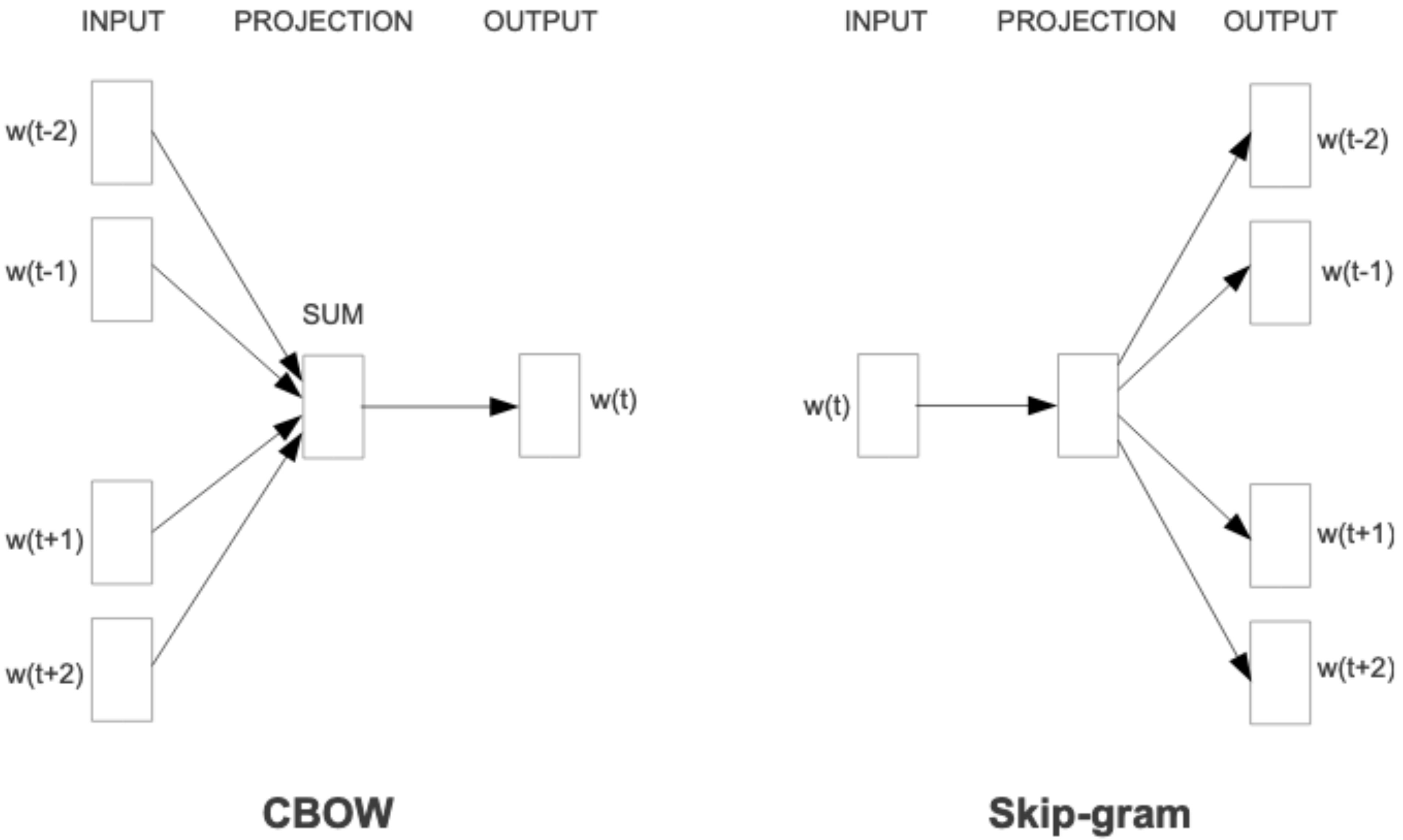
$\mathbf{u} \in \mathbb{R}^{|V|}$: output weights



Same weight vector for all context words

ARCHITECTURE COMPARISON

[Mikolov et al., 2017]



Issue: computing the loss function is very expensive!

ON NEGATIVE LOG LIKELIHOOD LOSS WITH LARGE VOCABULARIES

NEGATIVE LOG LIKELIHOOD LOSS

Given a context y, we want to maximize to probability of word x

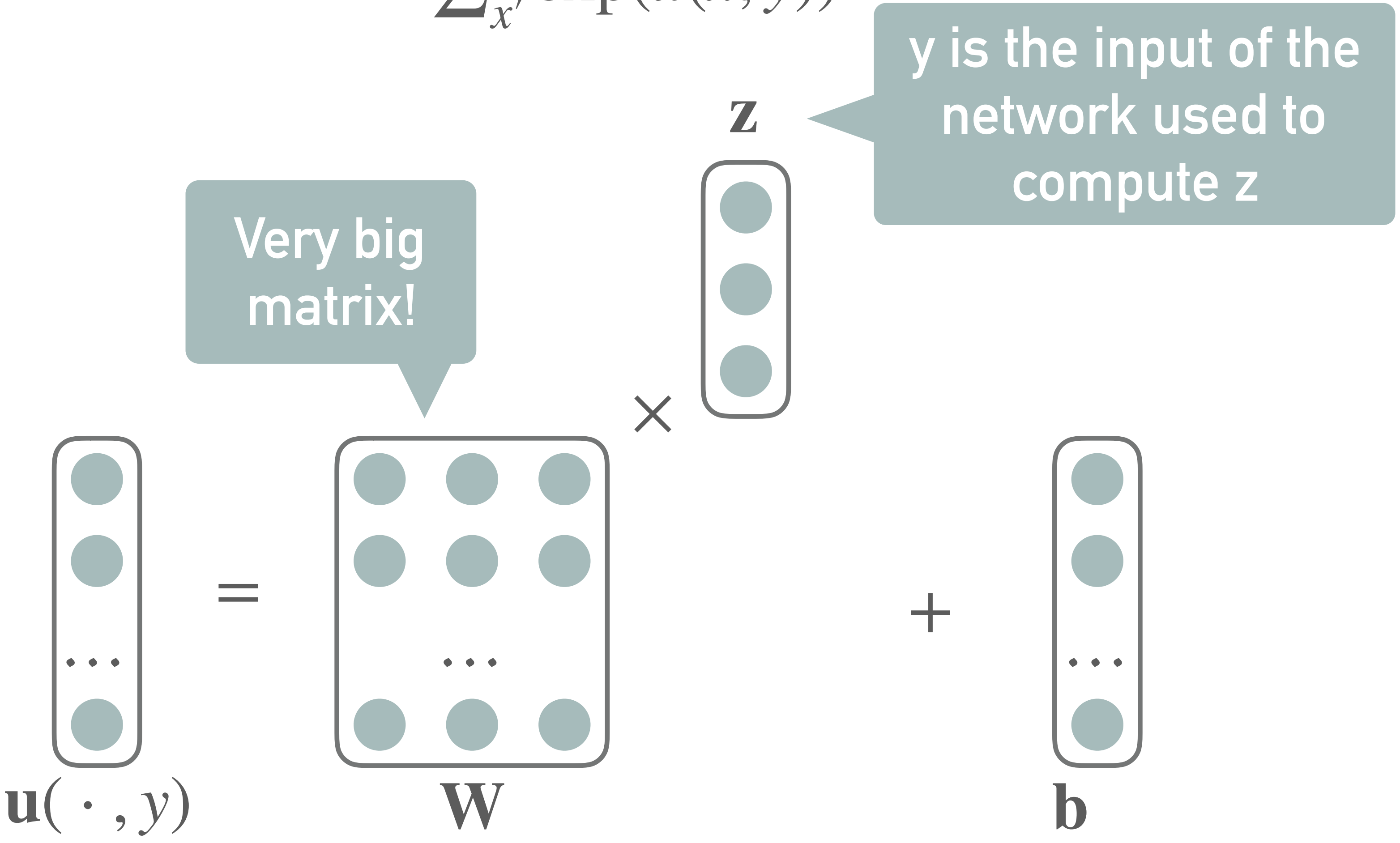
- Language modeling: y can be the previous word, or the 2 previous words, etc.
- Skip-gram: y is the current word and x is a context word
- CBOW: y is the set of context word and x is the current word

$$\mathscr{L}(x, y) = -\log p(x \mid y) = -\log \frac{\exp(u(x, y))}{\sum_{x'} \exp(u(x, y))}$$

$\mathbf{z} \in \mathbb{R}^m$ Last hidden representation

$\mathbf{u}(\cdot, y) \in \mathbb{R}^{|V|}$ Output logits

$\mathbf{W} \in \mathbb{R}^{|V| \times m}$
$\mathbf{b} \in \mathbb{R}^{|V| \times m}$ Output projection



NOISE DISTRIBUTION

Main idea

- We cannot manipulate p(x|y) directly (neither evaluate or sample from it)
- We can manipule the unnormalized distribution exp(u(x, y)) for a given x
- We can manipulate « simpler » noise distribution q(x) or q(x|y)

NOISE DISTRIBUTION

Main idea

- We cannot manipulate p(x|y) directly (neither evaluate or sample from it)
- We can manipule the unnormalized distribution exp(u(x, y)) for a given x
- We can manipulate « simpler » noise distribution q(x) or q(x|y)

Example of noise distribution

- Uniform distribution: $q(x) = \frac{1}{|V|}$
- Unigram distribution: $q(x) = \frac{n.\ occurence\ of\ x\ in\ data}{num\ words\ in\ data}$
- Bigram, trigram… distributions

Note

There exists other methods that replace the output softmax qui with a different layer, like hierarchical softmax => we won't cover this in the lecture.

NLL GRADIENT

$$\nabla - \log p(y\,|\,x) = \nabla - \log \frac{\exp(u(x, y))}{Z}$$

Partition function

NLL GRADIENT

$$\nabla - \log p(y \mid x) = \nabla - \log \frac{\exp(u(x, y))}{Z} \quad = \nabla - \log \frac{\exp(u(x, y))}{\sum_{x}^{\prime} \exp(u(x', y))}$$

Partition function

NLL GRADIENT

$$\nabla - \log p(y \mid x) = \nabla - \log \frac{\exp(u(x, y))}{Z} \quad = \nabla - \log \frac{\exp(u(x, y))}{\sum_{x}^{'} \exp(u(x', y))}$$

$$= \nabla \big(- u(x, y) + \log \sum_{x'} \exp(u(x', y)) \big)$$

NLL GRADIENT

$$\nabla - \log p(y \mid x) = \nabla - \log \frac{\exp(u(x,y))}{Z} \quad = \nabla - \log \frac{\exp(u(x,y))}{\sum_{x}^{'} \exp(u(x',y))}$$

$$= \nabla \big(- u(x,y) + \log \sum_{x'} \exp(u(x',y)) \big)$$

$$= - \nabla u(x,y) + \nabla \log \sum_{x'} \exp(u(x',y))$$

NLL GRADIENT

$$\nabla - \log p(y \mid x) = \nabla - \log \frac{\exp(u(x, y))}{Z} \quad = \nabla - \log \frac{\exp(u(x, y))}{\sum_{x}^{\prime} \exp(u(x', y))}$$

$$= \nabla \big(- u(x, y) + \log \sum_{x'} \exp(u(x', y)) \big)$$

$$= - \nabla u(x, y) + \nabla \log \sum_{x'} \exp(u(x', y))$$

$$= - \nabla u(x, y) + \frac{1}{\sum_{x'} \exp(u(x', y))} \nabla \sum_{x'} \exp(u(x', y))$$

Chain rule

NLL GRADIENT

$$\nabla - \log p(y\,|\,x) = \nabla - \log \frac{\exp(u(x,y))}{Z} \quad = \nabla - \log \frac{\exp(u(x,y))}{\sum_{x}^{'} \exp(u(x',y))}$$

$$= \nabla\big(-u(x,y) + \log \sum_{x'} \exp(u(x',y))\big)$$

$$= -\nabla u(x,y) + \nabla \log \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{\sum_{x'} \exp(u(x',y))} \nabla \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{Z} \sum_{x'} \exp(u(x'))\,\nabla u(x',y)$$

NLL GRADIENT

$$\nabla - \log p(y\,|\,x) = \nabla - \log \frac{\exp(u(x,y))}{Z} \quad = \nabla - \log \frac{\exp(u(x,y))}{\sum_x^{'} \exp(u(x',y))}$$

$$= \nabla\Big(-u(x,y) + \log \sum_{x'} \exp(u(x',y))\Big)$$

$$= -\nabla u(x,y) + \nabla \log \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{\sum_{x'} \exp(u(x',y))} \nabla \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{Z} \sum_{x'} \exp(u(x'))\, \nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} \frac{\exp(u(x'))}{Z} \nabla u(x',y)$$

NLL GRADIENT

$$\nabla - \log p(y\,|\,x) = \nabla - \log \frac{\exp(u(x,y))}{Z} \quad = \nabla - \log \frac{\exp(u(x,y))}{\sum_{x}^{'} \exp(u(x',y))}$$

$$= \nabla\big(-u(x,y) + \log \sum_{x'} \exp(u(x',y))\big)$$

$$= -\nabla u(x,y) + \nabla \log \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{\sum_{x'} \exp(u(x',y))} \nabla \sum_{x'} \exp(u(x',y))$$

Chain rule

$$= -\nabla u(x,y) + \frac{1}{Z} \sum_{x'} \exp(u(x')) \nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} \frac{\exp(u(x'))}{Z} \nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} p(x') \nabla u(x',y)$$

MONTE-CARLO ESTIMATION

NLL gradient

$$\nabla - \log p(x \mid y) = - \nabla u(x) + \sum_{x'} p(x' \mid y) \nabla u(x', y)$$

MONTE-CARLO ESTIMATION

NLL gradient

$$\nabla - \log p(x \mid y) = -\nabla u(x) + \sum_{x'} p(x' \mid y) \nabla u(x', y) \;=\; - \underbrace{\nabla u(x, y)}_{1} + \underbrace{\mathbb{E}_{p(x' \mid y)}\left[\nabla u(x', y)\right]}_{2}$$

1. Increase the score of the gold output

   Easy to compute!

2. Decrease the score of every other output

   Expensive for large output spaces!

MONTE-CARLO ESTIMATION

NLL gradient

$$\nabla - \log p(x \mid y) = -\nabla u(x) + \sum_{x'} p(x' \mid y) \nabla u(x', y) \quad = -\underbrace{\nabla u(x, y)}_{1} + \underbrace{\mathbb{E}_{p(x'|y)}\big[\nabla u(x', y)\big]}_{2}$$

1. Increase the score of the gold output

Easy to compute!

2. Decrease the score of every other output

Expensive for large output spaces!

Monte-Carlo estimation of the second term

We can approximate the expectation in the second term with k samples:

$$x^{(1)}, \ldots, x^{(k)} \sim p(x \mid y)$$

$$\mathbb{E}_{p(x', y)}\big[\nabla u(x')\big] \simeq \frac{1}{n} \sum_{i=1}^{k} \nabla u(x^{(i)}, y)$$

MONTE-CARLO ESTIMATION

NLL gradient

$$\nabla - \log p(x \mid y) = -\nabla u(x) + \sum_{x'} p(x' \mid y) \nabla u(x', y) \;= -\underbrace{\nabla u(x, y)}_{1} + \underbrace{\mathbb{E}_{p(x'|y)}\big[\nabla u(x', y)\big]}_{2}$$

1. Increase the score of the gold output
2. Decrease the score of every other output

Easy to compute!

Expensive for large output spaces!

Monte-Carlo estimation of the second term

We can approximate the expectation in the second term with k samples:

$$x^{(1)}, \ldots, x^{(k)} \sim p(x \mid y)$$

To sample, we need to compute $u(x, y)$:(

$$\mathbb{E}_{p(x',y)}\big[\nabla u(x')\big] \simeq \frac{1}{n} \sum_{i=1}^{k} \nabla u(x^{(i)}, y)$$

IMPORTANCE SAMPLING

IMPORTANCE SAMPLING

Intuition

Instead of sampling from $p(x \mid y)$, we sample from a « simpler » distribution $q(x)$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x \mid y) = - \nabla u(x, y) + \sum_{x'} p(x') \nabla u(x', y)$$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x \mid y) = -\nabla u(x, y) + \sum_{x'} p(x') \nabla u(x', y)$$

$$= -\nabla u(x, y) + \sum_{x'} q(x') \frac{p(x' \mid y)}{q(x')} \nabla u(x', y)$$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x \,|\, y) = -\nabla u(x, y) + \sum_{x'} p(x') \nabla u(x', y)$$

$$= -\nabla u(x, y) + \sum_{x'} q(x') \frac{p(x' \,|\, y)}{q(x')} \nabla u(x', y)$$

$$= -\nabla u(x, y) + \mathbb{E}_{q(x')} [\frac{p(x')}{q(x')} \nabla u(x', y)]$$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x\,|\,y) = -\nabla u(x, y) + \sum_{x'} p(x')\,\nabla u(x', y)$$

$$= -\nabla u(x, y) + \sum_{x'} q(x')\frac{p(x'\,|\,y)}{q(x')}\nabla u(x', y)$$

$$= -\nabla u(x, y) + \mathbb{E}_{q(x')}\Big[\frac{p(x')}{q(x')}\nabla u(x', y)\Big]$$

The expectation here is over q(x)

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla -\log p(x|y) = -\nabla u(x,y) + \sum_{x'} p(x')\nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)$$

$$= -\nabla u(x,y) + \mathbb{E}_{q(x')}\left[\frac{p(x')}{q(x')}\nabla u(x',y)\right]$$

The expectation here is over q(x)

Monte-Carlo estimation

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x\,|\,y) = -\nabla u(x,y) + \sum_{x'} p(x')\,\nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} q(x')\frac{p(x'\,|\,y)}{q(x')}\nabla u(x',y)$$

$$= -\nabla u(x,y) + \mathbb{E}_{q(x')}[\frac{p(x')}{q(x')}\nabla u(x',y)]$$

The expectation here is over q(x)

Monte-Carlo estimation

$$x^{(1)},\ldots,x^{(k)} \sim q(x)$$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x \mid y) = -\nabla u(x,y) + \sum_{x'} p(x')\,\nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} q(x') \frac{p(x' \mid y)}{q(x')} \nabla u(x',y)$$

$$= -\nabla u(x,y) + \mathbb{E}_{q(x')}\Big[\frac{p(x')}{q(x')} \nabla u(x',y)\Big]$$

The expectation here is over q(x)

Monte-Carlo estimation

$$x^{(1)}, \ldots, x^{(k)} \sim q(x)$$

$$\mathbb{E}_{q(x')}\Big[\frac{p(x')}{q(x')} \nabla u(x')\Big] \simeq \frac{1}{n} \sum_{i=1}^{k} \frac{p(x')}{q(x')} \nabla u(x^{(i)})$$

IMPORTANCE SAMPLING

Intuition

Instead of sampling from p(x | y), we sample from a « simpler » distribution q(x)

$$\nabla - \log p(x\,|\,y) = -\nabla u(x,y) + \sum_{x'} p(x')\,\nabla u(x',y)$$

$$= -\nabla u(x,y) + \sum_{x'} q(x')\frac{p(x'\,|\,y)}{q(x')}\nabla u(x',y)$$

$$= -\nabla u(x,y) + \mathbb{E}_{q(x')}[\frac{p(x')}{q(x')}\nabla u(x',y)]$$

The expectation here is over q(x)

Monte-Carlo estimation

$$x^{(1)},\ldots,x^{(k)} \sim q(x)$$

Easy because of the premise! :)

We still need to compute the partition Z for the numerator :(

$$\mathbb{E}_{q(x')}[\frac{p(x')}{q(x')}\nabla u(x')] \simeq \frac{1}{n}\sum_{i=1}^{k}\frac{p(x')}{q(x')}\nabla u(x^{(i)})$$

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'\,|\,y)\,\nabla u(x',y)\right] = \sum_{x'} p(x'\,|\,y)\,\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'\,|\,y)}{q(x')}\,\nabla u(x',y)$$

Importance sampling

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'|y)\nabla u(x',y)\right] = \sum_{x'} p(x'|y)\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)$$

Importance sampling

$$= \frac{\sum_{x'} q(x')\frac{p(x')}{q(x')}\nabla u(x',y)}{\sum_{x''} p(x''|y)}$$

=1 (because p is a probability distribution)

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'|y)\nabla u(x',y)\right] = \sum_{x'} p(x'|y)\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)$$

Importance sampling

$$= \frac{\sum_{x'} q(x')\frac{p(x')}{q(x')}\nabla u(x',y)}{\sum_{x''} p(x''|y)}$$

=1 (because p is a probability distribution)

$$= \frac{\sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{p(x''|y)}{q(x'')}}$$

Same trick as in the numerator

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'|y)\nabla u(x',y)\right] = \sum_{x'} p(x'|y)\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)$$

Importance sampling

$$= \frac{\sum_{x'} q(x')\frac{p(x')}{q(x')}\nabla u(x',y)}{\sum_{x''} p(x''|y)}$$

=1 (because p is a probability distribution)

$$= \frac{\sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{p(x''|y)}{q(x'')}}$$

Same trick as in the numerator

$$= \frac{\sum_{x'} q(x')\frac{Z^{-1}\exp(u(x',y))}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{Z^{-1}\exp(u(x''))}{q(x'')}}$$

The partition function appear as constant wrt summations!

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'|y)\nabla u(x',y)\right] = \sum_{x'} p(x'|y)\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)$$

Importance sampling

$$= \frac{\sum_{x'} q(x')\frac{p(x')}{q(x')}\nabla u(x',y)}{\sum_{x''} p(x''|y)}$$

=1 (because p is a probability distribution)

$$= \frac{\sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{p(x''|y)}{q(x'')}}$$

Same trick as in the numerator

$$= \frac{\sum_{x'} q(x')\frac{Z^{-1}\exp(u(x',y))}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{Z^{-1}\exp(u(x''))}{q(x'')}}$$

The partition function appear as constant wrt summations!

$$= \frac{Z^{-1}\sum_{x'} q(x')\frac{\exp(u(x',y))}{q(x')}\nabla u(x',y)}{Z^{-1}\sum_{x''} q(x'')\frac{\exp(u(x''))}{q(x'')}}$$

SELF-NORMALIZED IMPORTANCE SAMPLING 1/2

$$\mathbb{E}_{p(x'|y)}\left[p(x'|y)\,\nabla u(x',y)\right] = \sum_{x'} p(x'|y)\,\nabla u(x',y) = \sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\,\nabla u(x',y)$$

Importance sampling

$$= \frac{\sum_{x'} q(x')\frac{p(x')}{q(x')}\,\nabla u(x',y)}{\sum_{x''} p(x''|y)}$$

=1 (because p is a probability distribution)

$$= \frac{\sum_{x'} q(x')\frac{p(x'|y)}{q(x')}\,\nabla u(x',y)}{\sum_{x''} q(x'')\frac{p(x''|y)}{q(x'')}}$$

Same trick as in the numerator

$$= \frac{\sum_{x'} q(x')\frac{Z^{-1}\exp(u(x',y))}{q(x')}\,\nabla u(x',y)}{\sum_{x''} q(x'')\frac{Z^{-1}\exp(u(x''))}{q(x'')}}$$

The partition function appear as constant wrt summations!

$$= \frac{\cancel{Z}^{-1}\sum_{x'} q(x')\frac{\exp(u(x',y))}{q(x')}\,\nabla u(x',y)}{\cancel{Z}^{-1}\sum_{x''} q(x'')\frac{\exp(u(x''))}{q(x'')}}$$

No partition function anymore!

SELF-NORMALIZED IMPORTANCE SAMPLING 2/2

$$\mathbb{E}_{p(x')}\big[\nabla u(x', y)\big] = \frac{\sum_{x'} q(x') \frac{\exp(u(x', y))}{q(x')} \nabla u(x', y)}{\sum_{x''} q(x'') \frac{\exp(u(x''))}{q(x'')}}$$

No partition, but still an expensive sum! :(

SELF-NORMALIZED IMPORTANCE SAMPLING 2/2

$$\mathbb{E}_{p(x')}\big[\nabla u(x', y)\big] = \frac{\sum_{x'} q(x') \frac{\exp(u(x', y))}{q(x')} \nabla u(x', y)}{\sum_{x''} q(x'') \frac{\exp(u(x''))}{q(x'')}}$$

No partition, but still an expensive sum! :(

$$= \frac{\mathbb{E}_{q(x')}\big[\frac{\exp(u(x', y))}{q(x')} \nabla u(x')\big]}{\mathbb{E}_{q(x'')}\big[\frac{\exp(u(x'', y))}{q(x'')}\big]}$$

Expectations over q

SELF-NORMALIZED IMPORTANCE SAMPLING 2/2

$$\mathbb{E}_{p(x')}\left[\nabla u(x',y)\right] = \frac{\sum_{x'} q(x')\frac{\exp(u(x',y))}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{\exp(u(x''))}{q(x'')}}$$

No partition, but still an expensive sum! :(

$$= \frac{\mathbb{E}_{q(x')}\left[\frac{\exp(u(x',y))}{q(x')}\nabla u(x')\right]}{\mathbb{E}_{q(x'')}\left[\frac{\exp(u(x'',y))}{q(x'')}\right]}$$

Expectations over q

Monte-Carlo estimation

$$x^{(1)},\ldots,x^{(n)} \sim q(x)$$

Easy because of the premise! :)

$$\mathbb{E}_{p(x')}\left[\nabla u(x')\right] \simeq \frac{\frac{1}{n}\sum_{i=1}^{n}\frac{\exp(u(x^{(i)}))}{q(x^{(i)})}\nabla u(x^{(i)}))]}{\frac{1}{n}\sum_{i=1}^{n}\frac{\exp(u(x^{(i)}))}{q(x^{(i)})}}$$

No partition! :)
We say that the target distribution
is unnormalized.

SELF-NORMALIZED IMPORTANCE SAMPLING 2/2

$$\mathbb{E}_{p(x')}\big[\nabla u(x',y)\big] = \frac{\sum_{x'} q(x')\frac{\exp(u(x',y))}{q(x')}\nabla u(x',y)}{\sum_{x''} q(x'')\frac{\exp(u(x''))}{q(x'')}}$$

No partition, but still an expensive sum! :(

$$= \frac{\mathbb{E}_{q(x')}\big[\frac{\exp(u(x',y))}{q(x')}\nabla u(x')\big]}{\mathbb{E}_{q(x'')}\big[\frac{\exp(u(x'',y))}{q(x'')}\big]}$$

Expectations over q

Monte-Carlo estimation

$$x^{(1)},\ldots,x^{(n)} \sim q(x)$$

Easy because of the premise! :)

$$\mathbb{E}_{p(x')}\big[\nabla u(x')\big] \simeq \frac{\frac{1}{n}\sum_{i=1}^{n}\frac{\exp(u(x^{(i)}))}{q(x^{(i)})}\nabla u(x^{(i)}))]}{\frac{1}{n}\sum_{i=1}^{n}\frac{\exp(u(x^{(i)}))}{q(x^{(i)})}}$$

No partition! :)
We say that the target distribution
is unnormalized.

Weighted importance sampling (same thing, different notation)

$$\mathbb{E}_{p(x')}\big[\nabla u(x')\big] \simeq \frac{1}{W}\sum_{i=1}^{n} w(x^{(i)})\,\nabla u(x^{(i)}) \quad \text{where} \quad w(x^{(i)}) = \frac{\exp(u(x^{(i)}))}{q(x^{(i)})} \quad \text{and} \quad W = \sum_{i=1}^{n} w(x^{(i)})$$

Weight instead of probability

NOISE CONTRASTIVE ESTIMATION (NCE) 1/5

Motivation

Can be arbitrary large! :(

- Self-normalized importance sampling is unbounded: $\mathbb{E}_{p(x')}\left[\nabla u(x')\right] \simeq \frac{1}{W}\sum_{i=1}^{n} w(x^{(i)})\,\nabla u(x^{(i)})$
- NCE change the problem into a binary classification task

Intuition

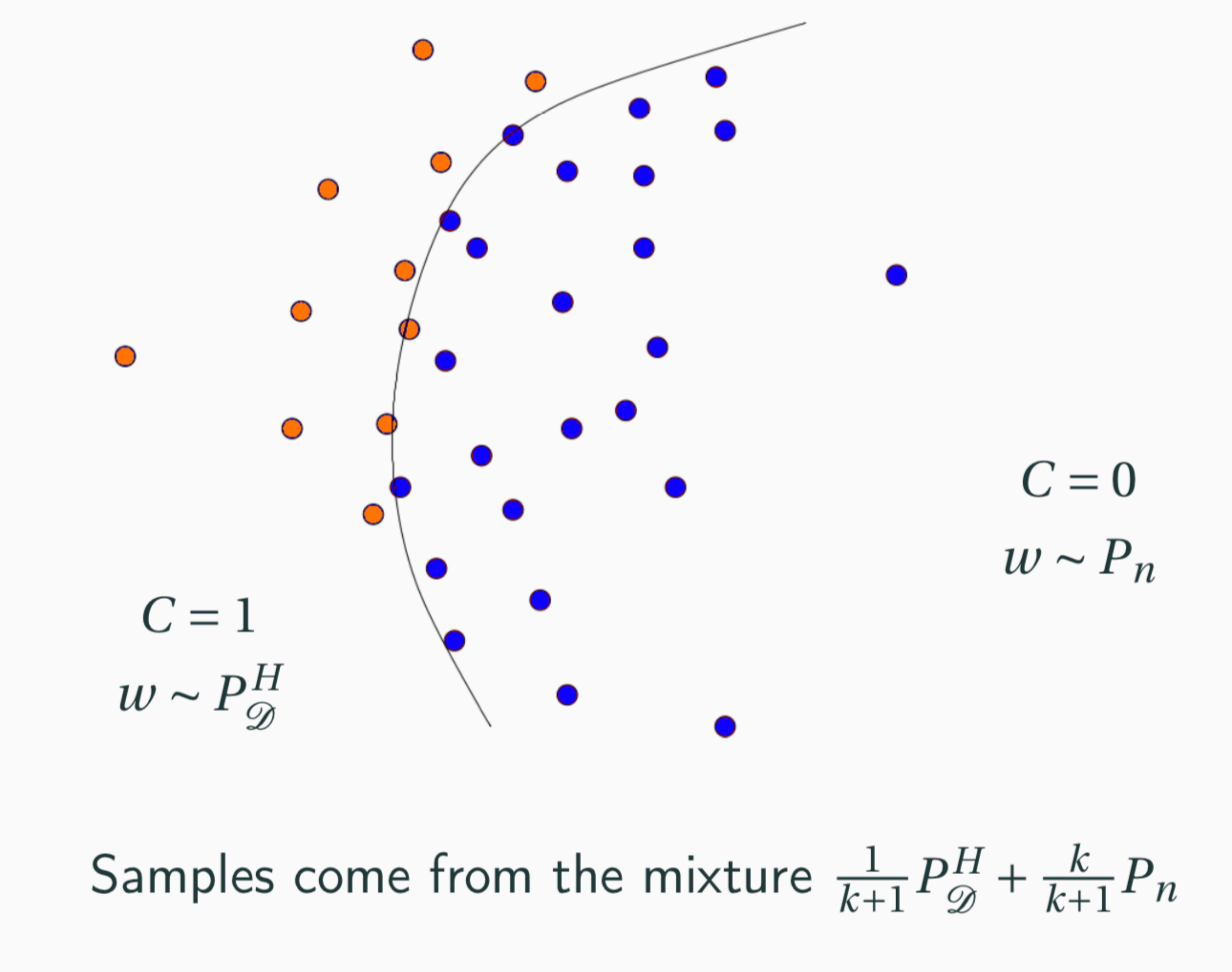
- We want to use the gold word positive reinforcement but **only a few other words** as negative reinforcement
- We want to keep theoretical guarantee (i.e. convergence guarantee)
- We cannot sample from p(x | y) so we use a **noise distribution** q(x)

Joint distribution over observed and noisy words

Let D be a random variable indicating if a given word is from the data distribution (D=1) or from the noise distribution (D=0)

$$p(d, x \,|\, y) = \begin{cases} p(D=0) \times q(x) & , \mathit{if}\ D = 0 \\ p(D=1) \times p(x \,|\, y) & , \mathit{if}\ D = 1 \end{cases}$$

NOISE CONTRASTIVE ESTIMATION ILLUSTRATION 1



Samples come from the mixture $\frac{1}{k+1} P_{\mathscr{D}}^{H} + \frac{k}{k+1} P_n$

(sorry, different notations!)

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x | y) = \begin{cases} p(D = 0) \times q(x) & , if\ D = 0 \\ p(D = 1) \times p(x | y) & , if\ D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D = 0) = \frac{k}{k + 1} \qquad p(D = 1) = \frac{1}{k + 1}$$

For each gold word we sample k noisy words

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D=0) \times q(x) & , \textit{if } D = 0 \\ p(D=1) \times p(x \mid y) & , \textit{if } D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D=0) = \frac{k}{k+1} \qquad p(D=1) = \frac{1}{k+1}$$

For each gold word we sample k noisy words

Binary classification problem

$p(d = 0 \mid x, y)$

$p(d = 1 \mid x)$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D=0) \times q(x) & , \textit{if } D = 0 \\ p(D=1) \times p(x \mid y) & , \textit{if } D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D=0) = \frac{k}{k+1} \qquad p(D=1) = \frac{1}{k+1}$$

For each gold word we sample k noisy words

Binary classification problem

$$p(d=0 \mid x, y) = \frac{p(d=0, x \mid y)}{p(d=0, x \mid y) + p(d=1, x \mid y)} \qquad p(d=1 \mid x)$$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D=0) \times q(x) & , if\ D = 0 \\ p(D=1) \times p(x \mid y) & , if\ D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D=0) = \frac{k}{k+1} \qquad p(D=1) = \frac{1}{k+1}$$

For each gold word we sample k noisy words

Binary classification problem

$$p(d = 0 \mid x, y) = \frac{p(d=0, x \mid y)}{p(d=0, x \mid y) + p(d=1, x \mid y)} \qquad p(d = 1 \mid x)$$

$$= \frac{\frac{k}{k+1} q(x)}{\frac{k}{k+1} q(x) + \frac{1}{k+1} p(x \mid y)}$$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D = 0) \times q(x) & , \mathit{if}\ D = 0 \\ p(D = 1) \times p(x \mid y) & , \mathit{if}\ D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D = 0) = \frac{k}{k + 1} \qquad\qquad p(D = 1) = \frac{1}{k + 1}$$

For each gold word we sample k noisy words

Binary classification problem

$$p(d = 0 \mid x, y) = \frac{p(d = 0,x \mid y)}{p(d = 0,x \mid y) + p(d = 1,x \mid y)} \qquad\qquad p(d = 1 \mid x)$$

$$= \frac{\frac{k}{\cancel{k+1}}q(x)}{\frac{k}{\cancel{k+1}}q(x) + \frac{1}{\cancel{k+1}}p(x \mid y)}$$

$$= \frac{kq(x)}{kq(x) + p(x \mid y)}$$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D = 0) \times q(x) & , \textit{if } D = 0 \\ p(D = 1) \times p(x \mid y) & , \textit{if } D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D = 0) = \frac{k}{k + 1} \qquad p(D = 1) = \frac{1}{k + 1}$$

For each gold word we sample k noisy words

Binary classification problem

$$p(d = 0 \mid x, y) = \frac{p(d = 0, x \mid y)}{p(d = 0, x \mid y) + p(d = 1, x \mid y)} \qquad p(d = 1 \mid x) = \frac{p(d = 1, x \mid y)}{p(d = 0, x \mid y) + p(d = 1, x \mid y)}$$

$$= \frac{\frac{k}{\cancel{k + 1}} q(x)}{\frac{k}{\cancel{k + 1}} q(x) + \frac{1}{\cancel{k + 1}} p(x \mid y)}$$

$$= \frac{kq(x)}{kq(x) + p(x \mid y)}$$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \mid y) = \begin{cases} p(D=0) \times q(x) & , \textit{if } D = 0 \\ p(D=1) \times p(x \mid y) & , \textit{if } D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D=0) = \frac{k}{k+1} \qquad\qquad p(D=1) = \frac{1}{k+1}$$

For each gold word we sample k noisy words

Binary classification problem

$$\begin{aligned} p(d=0 \mid x, y) &= \frac{p(d=0, x \mid y)}{p(d=0, x \mid y) + p(d=1, x \mid y)} \\ &= \frac{\frac{k}{\cancel{k+1}} q(x)}{\frac{k}{\cancel{k+1}} q(x) + \frac{1}{\cancel{k+1}} p(x \mid y)} \\ &= \frac{kq(x)}{kq(x) + p(x \mid y)} \end{aligned}$$

$$\begin{aligned} p(d=1 \mid x) &= \frac{p(d=1, x \mid y)}{p(d=0, x \mid y) + p(d=1, x \mid y)} \\ &= \frac{\frac{1}{k+1} p(x \mid y)}{\frac{k}{k+1} q(x) + \frac{1}{k+1} p(x \mid y)} \end{aligned}$$

NOISE CONTRASTIVE ESTIMATION 2/5

$$p(d, x \,|\, y) = \begin{cases} p(D=0) \times q(x) & , \mathit{if}\ D = 0 \\ p(D=1) \times p(x \,|\, y) & , \mathit{if}\ D = 1 \end{cases}$$

Let assume that we want to use k words for negative reinforcement, then we have:

$$p(D=0) = \frac{k}{k+1} \qquad p(D=1) = \frac{1}{k+1}$$

For each gold word we sample k noisy words

Binary classification problem

$$p(d=0 \,|\, x, y) = \frac{p(d=0, x \,|\, y)}{p(d=0, x \,|\, y) + p(d=1, x \,|\, y)}$$

$$= \frac{\frac{k}{\cancel{k+1}} q(x)}{\frac{k}{\cancel{k+1}} q(x) + \frac{1}{\cancel{k+1}} p(x \,|\, y)}$$

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

$$p(d=1 \,|\, x) = \frac{p(d=1, x \,|\, y)}{p(d=0, x \,|\, y) + p(d=1, x \,|\, y)}$$

$$= \frac{\frac{1}{\cancel{k+1}} p(x \,|\, y)}{\frac{k}{\cancel{k+1}} q(x) + \frac{1}{\cancel{k+1}} p(x \,|\, y)}$$

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

NOISE CONTRASTIVE ESTIMATION 3/5

Partition function

- NCE assume the partition function is a learned parameter
  i.e. we don't compute it, so the p(x|y) may not be a valid distribution
- There is one partition parameter per datapoint in the dataset
- An usual trick is to fix all of them to 1 when the dataset is large

$$p(D=0\,|\,x,y) = \frac{kq(x)}{kq(x)+p(x\,|\,y)}$$

$$= \frac{kq(x)}{kq(x)+\frac{\exp(u(x,y))}{Z(x,y)}}$$

$$p(D=0\,|\,x,y) = \frac{p(x\,|\,y)}{kq(x)+p(x\,|\,y)}$$

$$= \frac{\frac{\exp(u(x,y))}{Z(x,y)}}{kq(x)+\frac{\exp(u(x,y))}{Z(x,y)}}$$

Z(x, y) is a learned parameter here

NOISE CONTRASTIVE ESTIMATION 3/5

Partition function

- NCE assume the partition function is a learned parameter
  i.e. we don't compute it, so the p(x|y) may not be a valid distribution
- There is one partition parameter per datapoint in the dataset
- An usual trick is to fix all of them to 1 when the dataset is large

$$p(D = 0 | x, y) = \frac{kq(x)}{kq(x) + p(x|y)}$$

$$= \frac{kq(x)}{kq(x) + \frac{\exp(u(x,y))}{Z(x,y)}}$$

$$= \frac{kq(x)}{kq(x) + \exp(u(x,y))}$$

$$p(D = 0 | x, y) = \frac{p(x|y)}{kq(x) + p(x|y)}$$

$$= \frac{\frac{\exp(u(x,y))}{Z(x,y)}}{kq(x) + \frac{\exp(u(x,y))}{Z(x,y)}}$$

$$= \frac{\exp(u(x,y))}{kq(x) + \exp(u(x,y))}$$

Z(x, y) is a learned parameter here

Assume the partition function is always equal to 1

NOISE CONTRASTIVE ESTIMATION 3/5

Partition function

- NCE assume the partition function is a learned parameter
  i.e. we don't compute it, so the p(x|y) may not be a valid distribution
- There is one partition parameter per datapoint in the dataset
- An usual trick is to fix all of them to 1 when the dataset is large

$$
\begin{aligned}
p(D=0|x,y) &= \frac{kq(x)}{kq(x)+p(x|y)} \\
&= \frac{kq(x)}{kq(x)+\frac{\exp(u(x,y))}{Z(x,y)}} \\
&= \frac{kq(x)}{kq(x)+\exp(u(x,y))}
\end{aligned}
$$

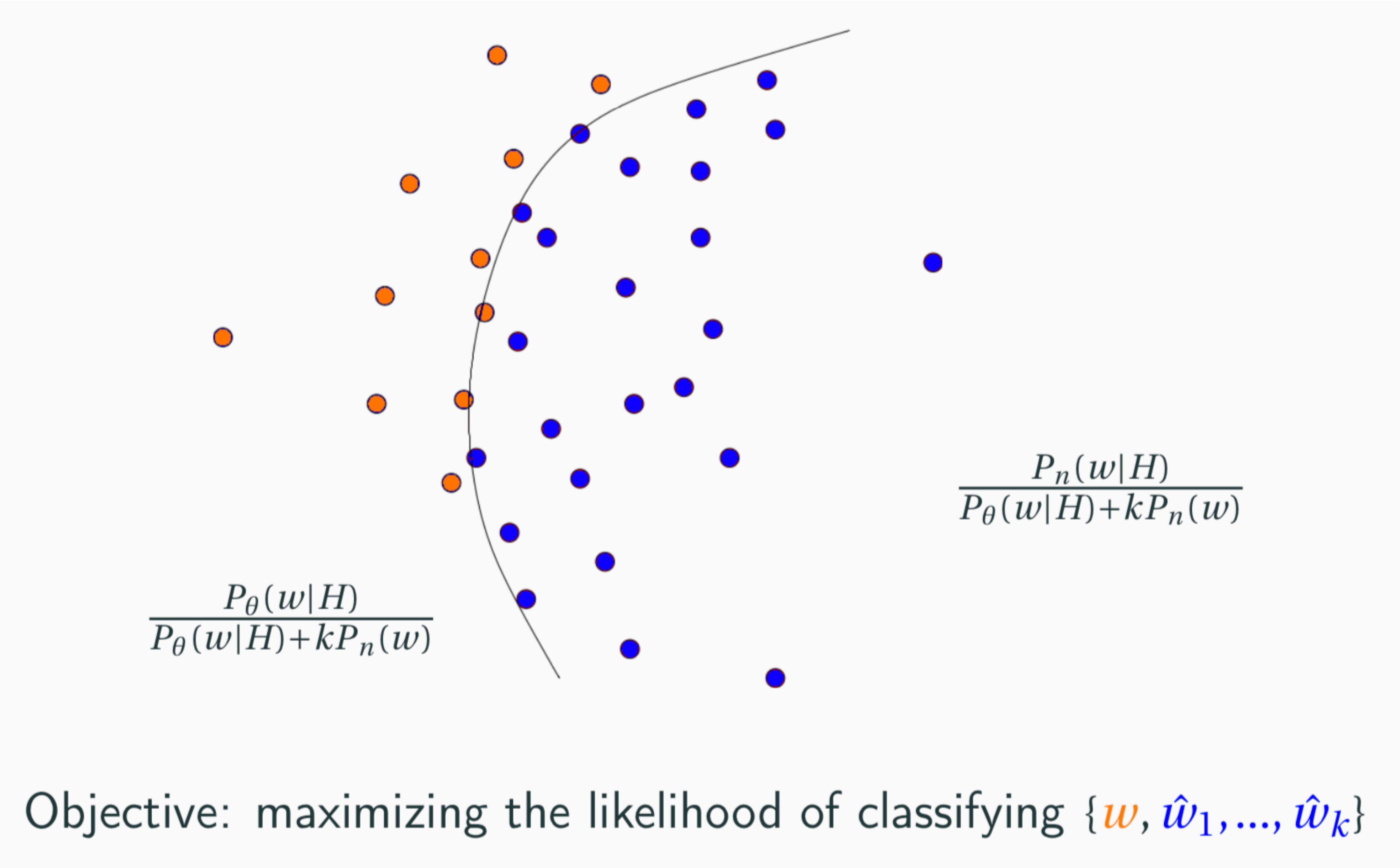
$$
\begin{aligned}
p(D=0|x,y) &= \frac{p(x|y)}{kq(x)+p(x|y)} \\
&= \frac{\frac{\exp(u(x,y))}{Z(x,y)}}{kq(x)+\frac{\exp(u(x,y))}{Z(x,y)}} \\
&= \frac{\exp(u(x,y))}{kq(x)+\exp(u(x,y))}
\end{aligned}
$$

Z(x, y) is a learned parameter here

Assume the partition function is always equal to 1

- Even if we use an un-normalized probability distributions,
  p(d|x, y) is always between 0 and 1 and is a valid distribution
- When the number of negative samples goes to infinity, **it is equivalent to NLL**

NOISE CONTRASTIVE ESTIMATION ILLUSTRATION 2



Objective: maximizing the likelihood of classifying $\{w, \hat{w}_1, ..., \hat{w}_k\}$

(sorry, different notations!)

NOISE CONTRASTIVE ESTIMATION 4/5

[Mnih and Teh, 2012]

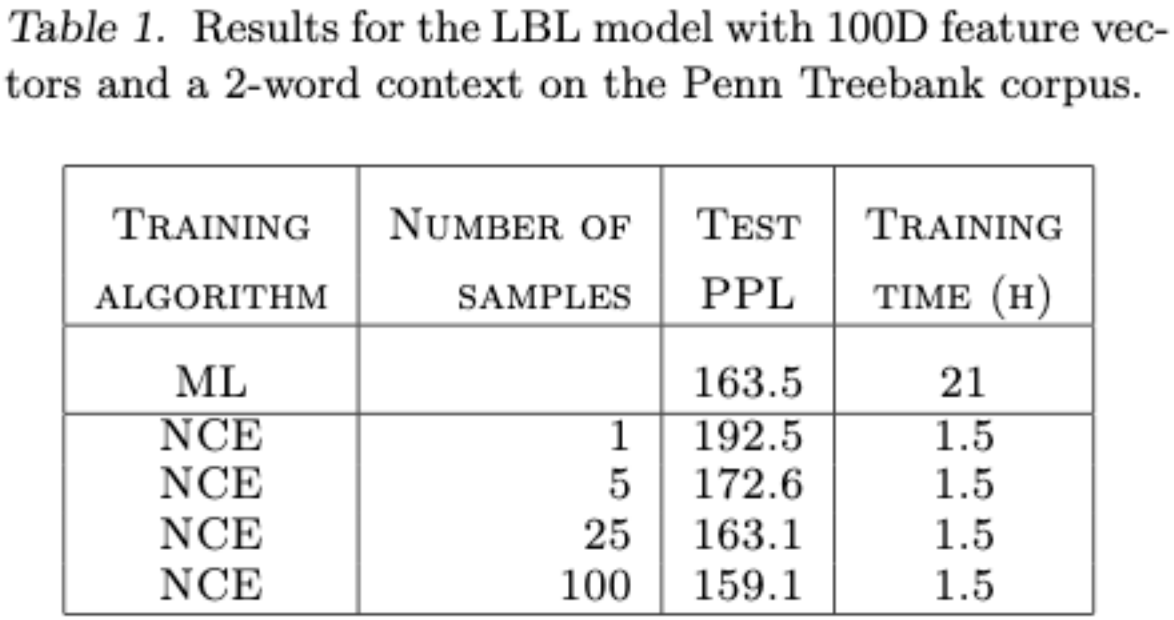


Table 1. Results for the LBL model with 100D feature vectors and a 2-word context on the Penn Treebank corpus.

TRAINING ALGORITHM	NUMBER OF SAMPLES	TEST PPL	TRAINING TIME (H)
ML		163.5	21
NCE	1	192.5	1.5
NCE	5	172.6	1.5
NCE	25	163.1	1.5
NCE	100	159.1	1.5

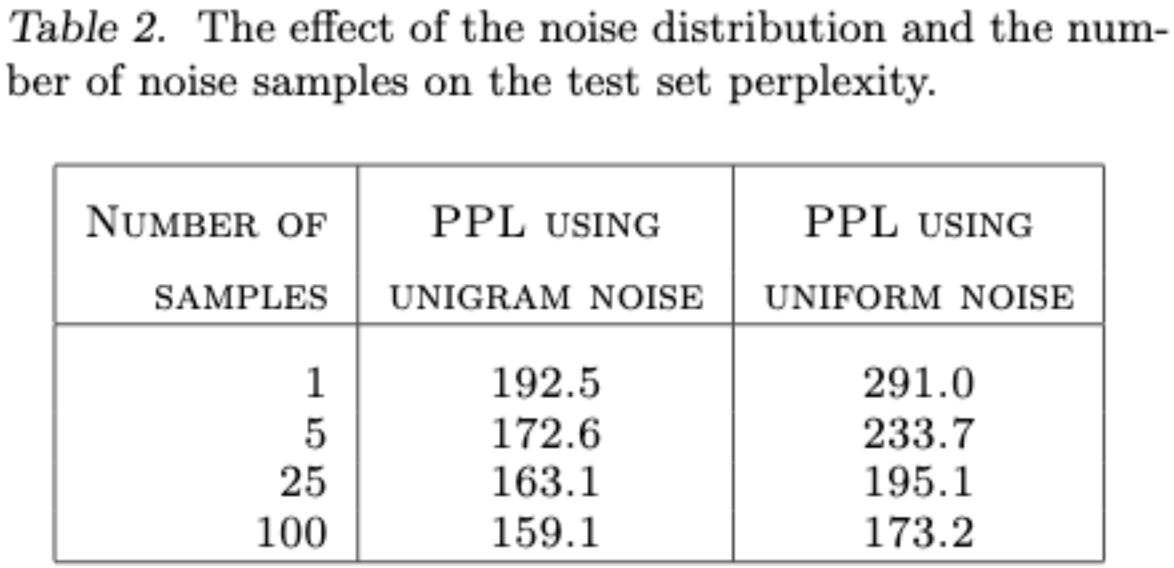


Table 2. The effect of the noise distribution and the number of noise samples on the test set perplexity.

NUMBER OF SAMPLES	PPL USING UNIGRAM NOISE	PPL USING UNIFORM NOISE
1	192.5	291.0
5	172.6	233.7
25	163.1	195.1
100	159.1	173.2

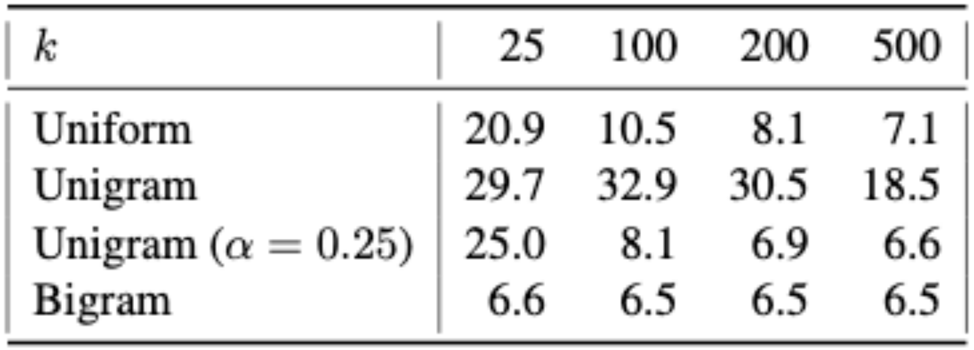
NOISE CONTRASTIVE ESTIMATION 5/5

[Labeau and Allauzen, 2017]

Unigram distribution smoothing

A distortion coefficient $0 < \alpha < 1$ is used to smooth the noise distribution increase the probability to sample rare words

$$q(x) = \frac{(n.\ occurence\ of\ x\ in\ data)^{\alpha}}{Z}$$



k	25	100	200	500
Uniform	20.9	10.5	8.1	7.1
Unigram	29.7	32.9	30.5	18.5
Unigram ($\alpha = 0.25$)	25.0	8.1	6.9	6.6
Bigram	6.6	6.5	6.5	6.5

Table 1: Negative log-likelihood after one epoch of training with a full vocabulary, for various noise distributions and a varying number of noise samples k

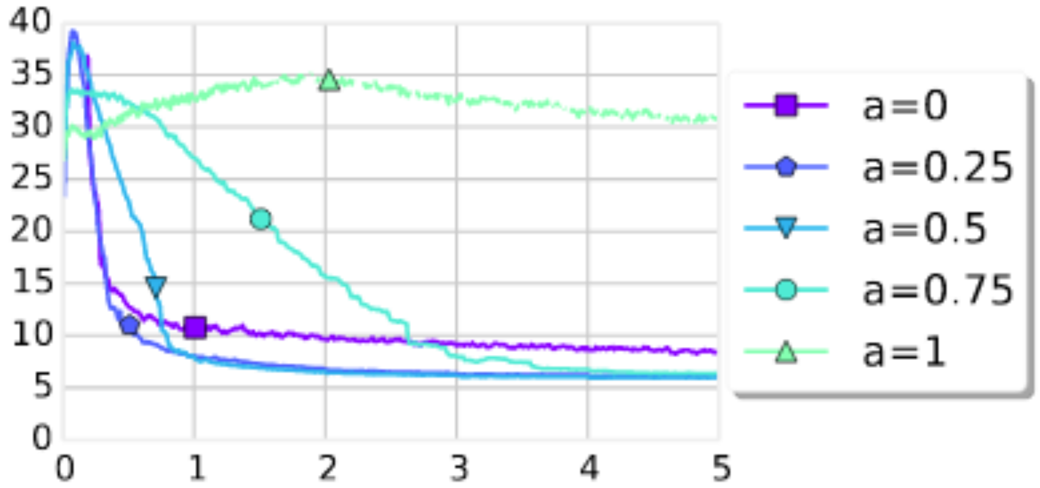


Figure 2: Comparative training of full vocabulary models with $k = 100$ noise samples for a varying distortion, on 5 epochs.

NEGATIVE SAMPLING

[Mikolov et al., 2013]

Idea

- Replace the NLL with a binary classification task
- Positive examples come from the data, negative example from the noise distribution

$$p(D = 0 \,|\, x, y) = \frac{1}{1 + \exp(u(x, y))} \qquad p(D = 1 \,|\, x, y) = \frac{\exp(u(x, y))}{1 + \exp(u(x, y))}$$

Warning

- It is a surrogate loss, it is not consistent with negative log likelihood
- It can be used to train word embeddings
- It is <u>**inadequate**</u> to train langage model (i.e. optimize the wrong objective)

CONTEXT SENSITIVE WORD EMBEDDINGS

NEURAL NETWORK PRE-TRAINING

Main idea

- Word embeddings may be useful on their own, but they are widely use to pre-train embedding table to increase accuracy, especially regarding out-of-vocabulary word
- Can we pre-train « more » than just word embeddings?

Yes! :)

Most popular pre-trained architectures

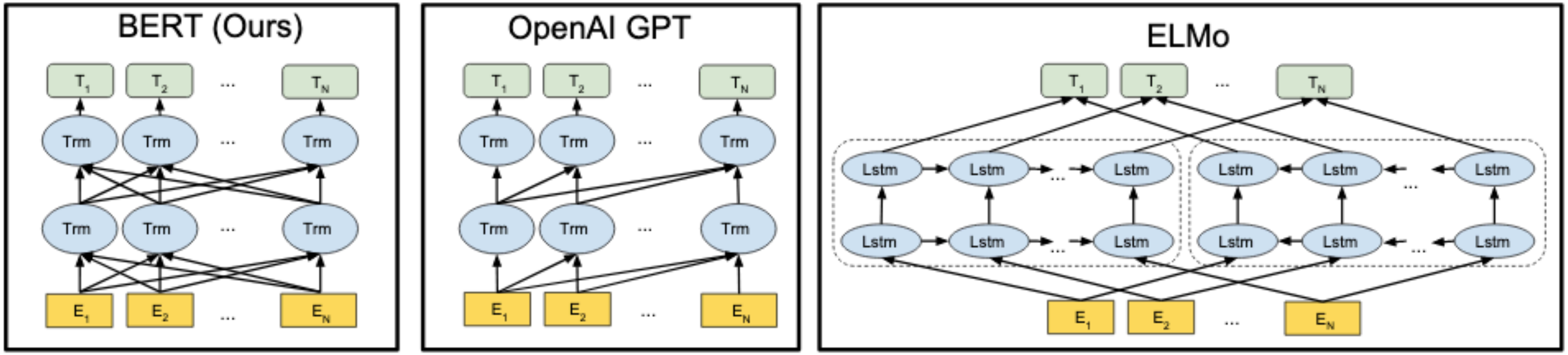
- ELMO
- GPT

We will not cover this one

- BERT

ARCHITECTURE COMPARISON

[Devlin et al., 2019]



EMBEDDINGS FROM LANGUAGE MODELS (ELMO)

[Peters et al., 2018]

Main idea

- ➤ Train a bi-directional language model based on LSTMs
- ➤ Each LSTM as several layer

Probability of a sentence

Forward LSTM

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

Backward LSTM

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

Training loss function

$$\sum_{k=1}^{N} \big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \big)$$

Links

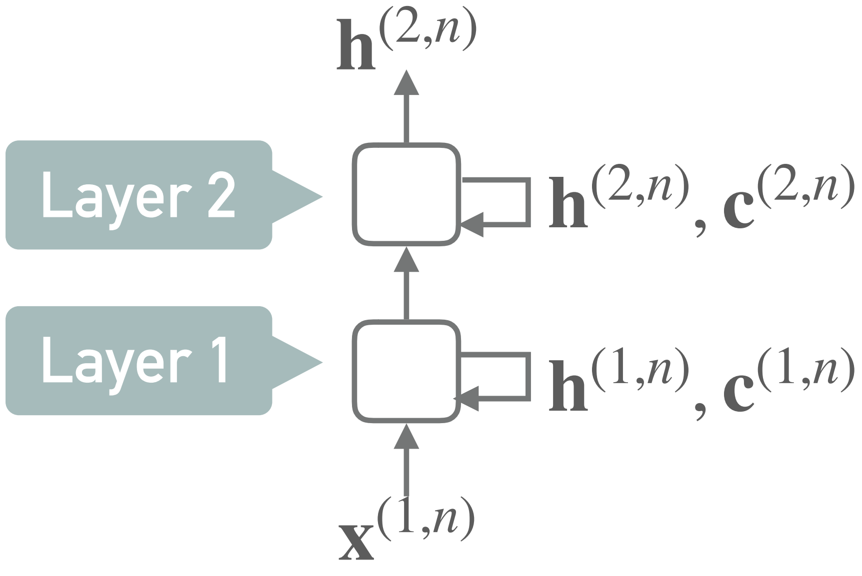
- ➤ https://allennlp.org/elmo
- ➤ https://allenai.github.io/allennlp-docs/api/allennlp.modules.elmo.html

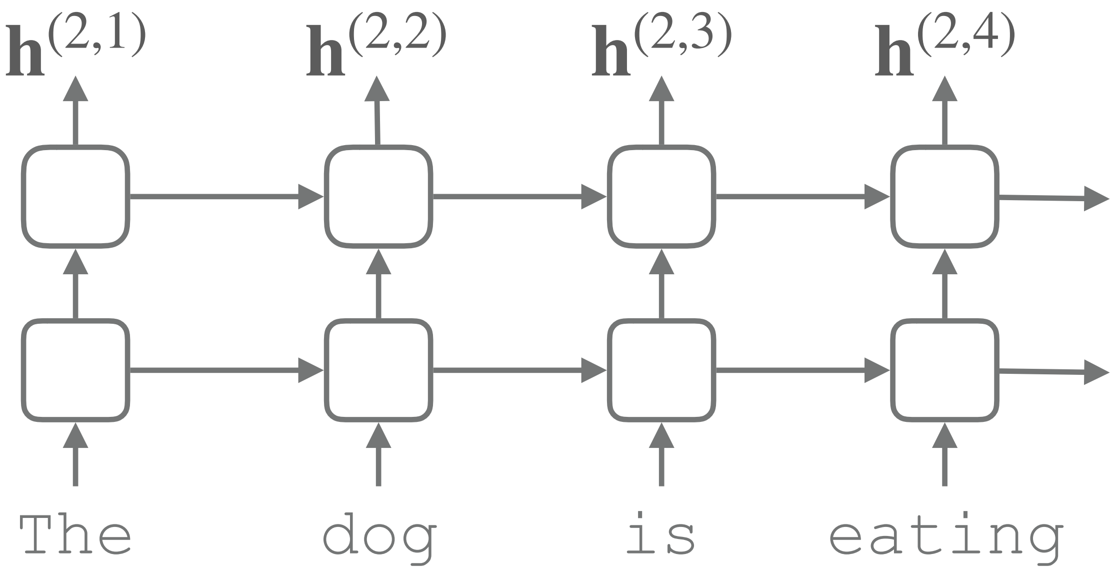
EMBEDDINGS FROM LANGUAGE MODELS (ELMO)

[Peters et al., 2018]

Multi-layer RNN

- Each layer as its own set of learn parameters
- Each direction as its own set of parameters
- In practice, the model is trained with 3 layers in each direction





Word input

- No word embeddings
- Character embeddings combined via a character CNN

EMBEDDINGS FROM LANGUAGE MODELS (ELMO)

[Peters et al., 2018]

How to use it in practice

- Concatenate hidden layers of the forward and backward LSTMs
- Do a parameterized convex combination of layers
- The ELMo LSTMs can either be fixed or fine-tuned

$\overrightarrow{\mathbf{h}}^{(l,i)}$: hidden layer of the forward LSTM at layer l for word at position i

$\overleftarrow{\mathbf{h}}^{(l,i)}$: hidden layer of the forward LSTM at layer l for word at position i

$$\mathbf{h}^{(l,i)} = [\overrightarrow{\mathbf{h}}^{(l,i)}, \overleftarrow{\mathbf{h}}^{(l,i)}]$$

Concatenate

$$\mathbf{h}^{(i)} = \sum_{l=1}^{L} \alpha^{(l)} \times \mathbf{h}^{(l,i)}$$

$$\alpha \in \left\{\alpha \in \mathbb{R}^{L} | \sum_{l=1}^{L} \alpha^{l} = 1 \; and \; \forall l : \alpha^{l} \geq 0\right\}$$

Context sensitive embedding for word i

Simplex constraint

```
import allennlp.modules.elmo as elmo

# init method of a module
def __init__(self)
   self.elmo = elmo.Elmo(
      options_file=path_to_options,
      weight_file=path_to_weights,
      # how many convex combination do you want?
      num_output_representations=1,
      # set to true at training time
      # if you want to fine-tune Elmo weights
      requires_grad=False,
      do_layer_norm=False,
      keep_sentence_boundaries=False,
      dropout=0.
   )
   # to get the output hidden dimension:
   #self.elmo.get_output_dim()
```

You need to install the AllenNLP lib

The convex combination parameters will have a gradient

```
import allennlp.modules.elmo as elmo
```

You need to install the AllenNLP lib

```
# init method of a module
def __init__(self)
    self.elmo = elmo.Elmo(
        options_file=path_to_options,
        weight_file=path_to_weights,
        # how many convex combination do you want?
        num_output_representations=1,
        # set to true at training time
        # if you want to fine-tune Elmo weights
        requires_grad=False,
        do_layer_norm=False,
        keep_sentence_boundaries=False,
        dropout=0.
    )
    # to get the output hidden dimension:
    #self.elmo.get_output_dim()
```

The convex combination parameters will have a gradient

```
def forward(elmo_inputs):
    # elmo_inputs should be a list of list of string
    # e.g. [["Sentence", "n.", "1"], ["Sentence", "n.", "2"]]
    elmo_inputs = elmo.batch_to_ids(elmo_inputs)
    # move to GPU if needed
    elmo_inputs = elmo_inputs.to(self.elmo.scalar_mix_0.gamma.device)

    #Â¬àcompute representation!
    elmo_output = self.elmo(elmo_inputs)['elmo_representations'][0])
```

BERT: DEEP BIDIRECTIONAL TRANSFORMERS

[Devlin et al., 2019]

Main idea

- ➤ Use a (big) transformer instead of a LSTM
- ➤ Use (trained) subword segmentation instead of word or char embeddings
- ➤ Use learned position embeddings
- ➤ Use two objective function:
  1. Masked language model
  2. Next sentence prediction (some variants don't)

Many variants

- ➤ Cased/uncased English BERT
- ➤ Multilingual BERT
- ➤ French Bert: CamemBERT and FlauBERT

INRIA (+ Facebook)

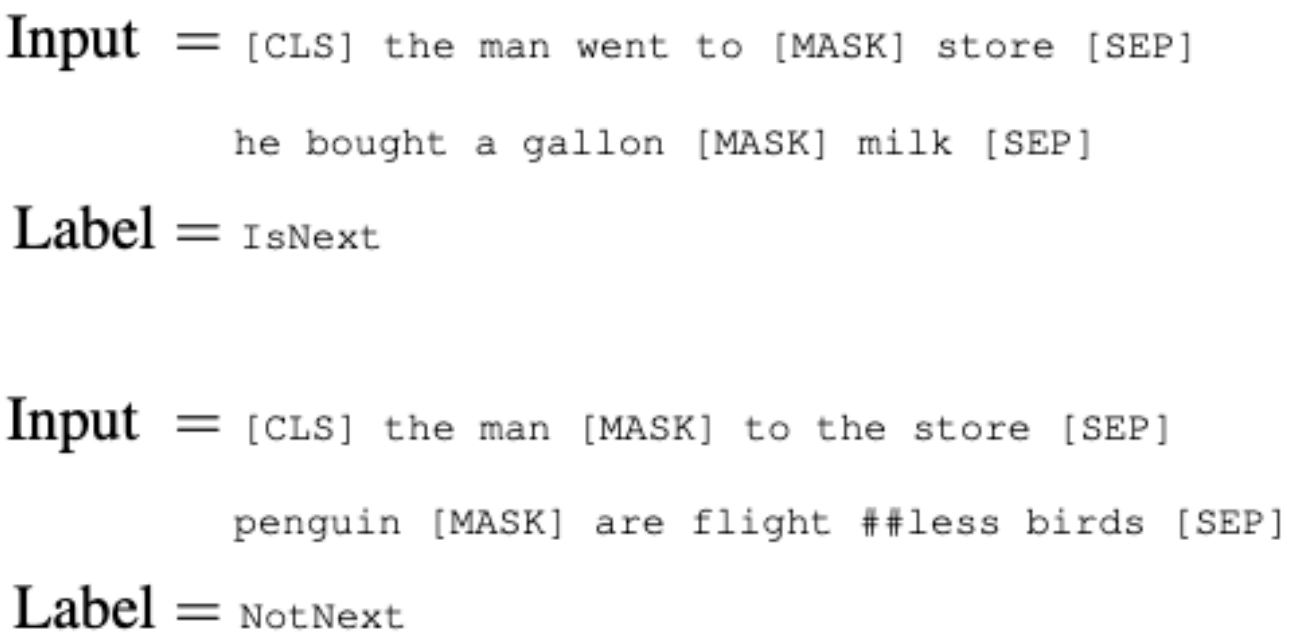
CNRS

BERT: DEEP BIDIRECTIONAL TRANSFORMERS

[Devlin et al., 2019]

Training data

- Introduce [CLS] token at the beginning of each sentence
- End sentence with the [SEP] token
- Randomly replace a subset of tokens with the [MASK] token
- Add the correct next sentence or a randomly sampled sentence from the corpus



Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`
Label = `IsNext`

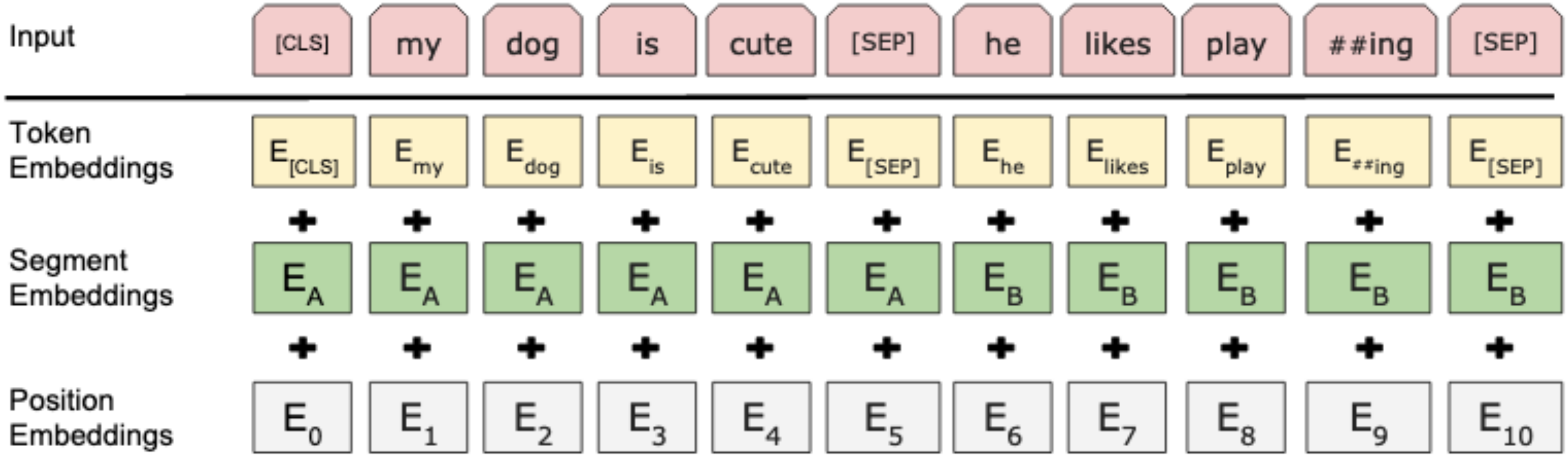
Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`
Label = `NotNext`

Training objective

- Predict masked tokens, i.e. word that have randomly been replaced with [MASK]
- Predict from the context sensitive embedding of [CLS] if the second sentence is correct

BERT: DEEP BIDIRECTIONAL TRANSFORMERS

[Devlin et al., 2019]

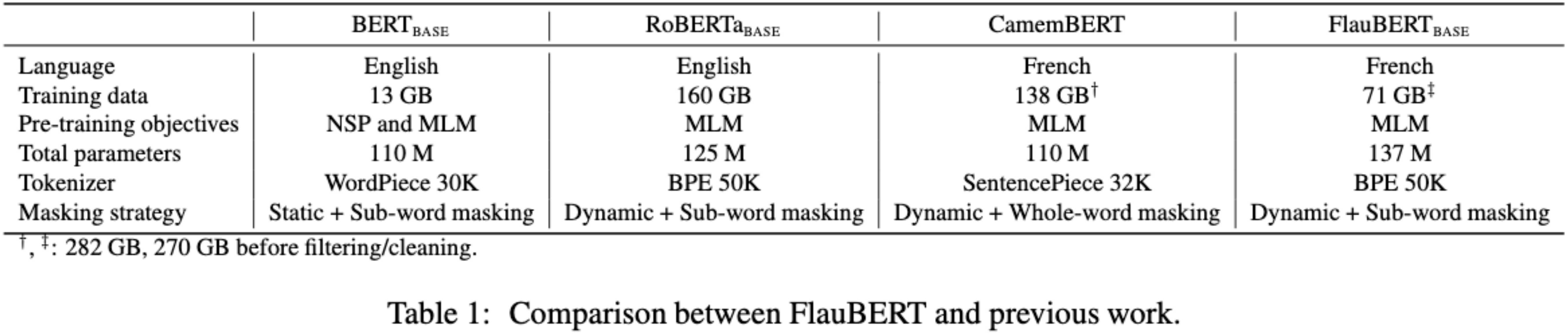


Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

- You may have more than one embedding per word!
  Either use the first or last token embedding for each word
- For sentence classification, you can use the [CLS] embedding
- Similar to ELMO, you can learn a convex combination of several layers instead of using the laster layer

BERT: DEEP BIDIRECTIONAL TRANSFORMERS

[Le et al., 2019]



	$\text{BERT}_{\text{BASE}}$	$\text{RoBERTa}_{\text{BASE}}$	CamemBERT	$\text{FlauBERT}_{\text{BASE}}$
Language	English	English	French	French
Training data	13 GB	160 GB	138 GB†	71 GB‡
Pre-training objectives	NSP and MLM	MLM	MLM	MLM
Total parameters	110 M	125 M	110 M	137 M
Tokenizer	WordPiece 30K	BPE 50K	SentencePiece 32K	BPE 50K
Masking strategy	Static + Sub-word masking	Dynamic + Sub-word masking	Dynamic + Whole-word masking	Dynamic + Sub-word masking

†, ‡: 282 GB, 270 GB before filtering/cleaning.

Table 1: Comparison between FlauBERT and previous work.