WestlakeNLP

Natural Language Processing

Yue Zhang
Westlake University

西湖大學
WESTLAKE UNIVERSITY

Chapter 17

Pre-training and Transfer Learning

Contents

Contents

Neural n-gram language modelling

WestlakeNLP

- **n-gram language models**

  In Chapter 2, n-gram LM are parameterized by conditional probability.

$$P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

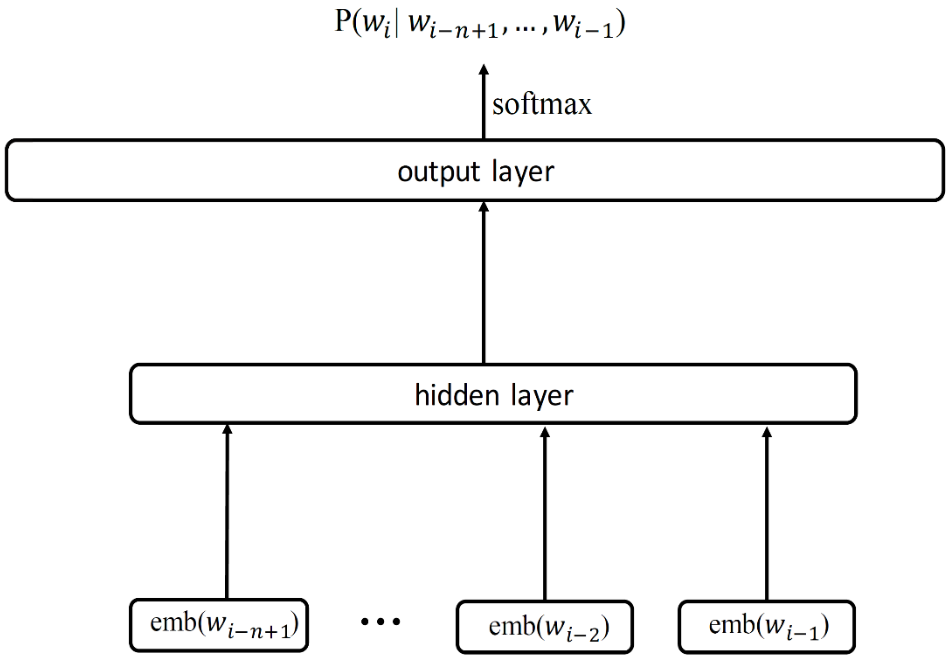
- **Neural n-gram language models**

  parameterized by neural network

Neural n-gram language modelling

- **Base model**

$$\mathbf{x} = \left(emb(w_{i-n+1}) \oplus \ldots \oplus emb(w_{i-1})\right)$$
$$\mathbf{h} = tanh(\mathbf{W}^h \mathbf{x} + \mathbf{b}^h)$$
$$\mathbf{o} = \mathbf{W}^o \mathbf{h} + \mathbf{b}^o$$
$$\mathbf{p} = softmax(\mathbf{o})$$



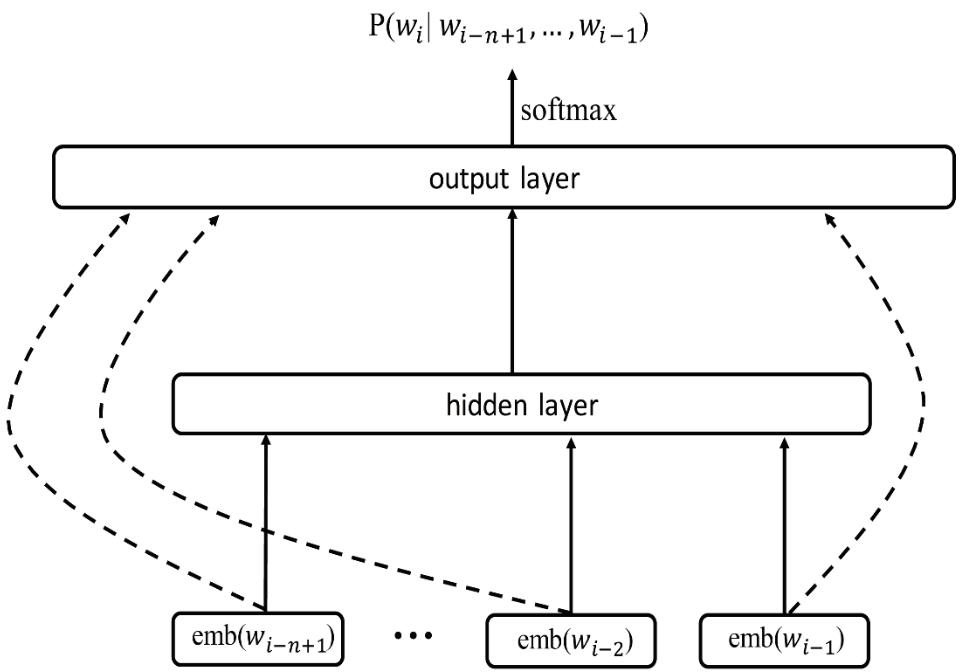
Here *emb* denotes word embeddings.
$\mathbf{W}^h$,$\mathbf{W}^o$,$\mathbf{b}^h$,$\mathbf{b}^o$ are model parameters.
$\mathbf{p}$ stands for the probability of word w_i.

Neural n-gram language modelling

WestlakeNLP

- **Adding shortcut connection**

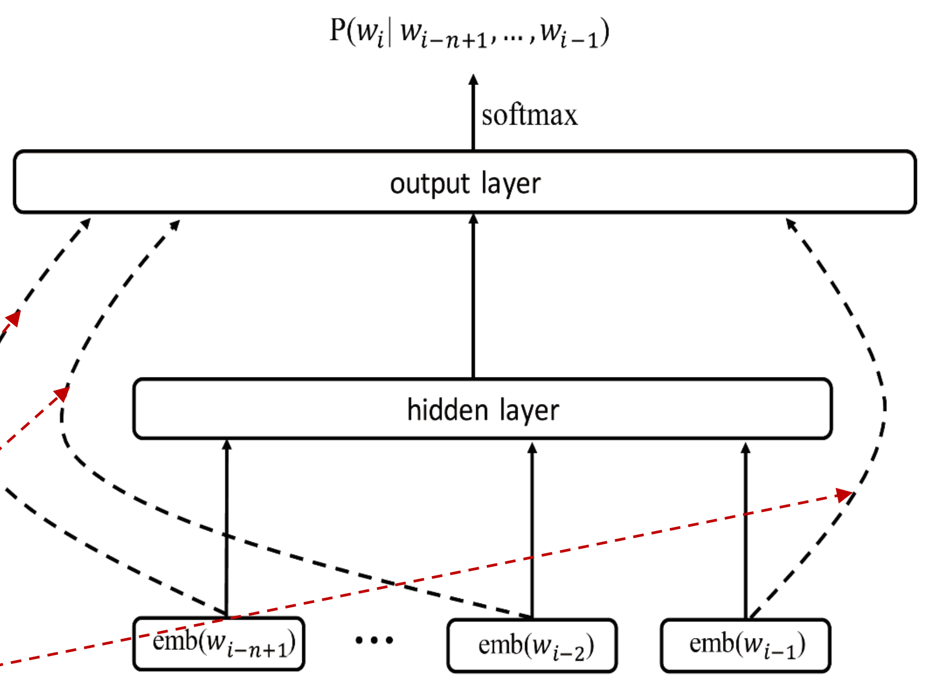
$$\mathbf{x} = \left(emb(w_{i-n+1}) \oplus \ldots \oplus emb(w_{i-1})\right)$$
$$\mathbf{h} = tanh(\mathbf{W}^n\mathbf{x} + \mathbf{b}^n)$$
$$\mathbf{o} = \mathbf{U}\mathbf{x} + \mathbf{W}^o\mathbf{h} + \mathbf{b}^o$$
$$\mathbf{p} = softmax(\mathbf{o}),$$



Neural n-gram language modelling

- **Adding shortcut connection**

$$\mathbf{x} = \left(emb(w_{i-n+1}) \oplus \ldots \oplus emb(w_{i-1})\right)$$
$$\mathbf{h} = tanh(\mathbf{W}^n\mathbf{x} + \mathbf{b}^n)$$
$$\mathbf{o} = \mathbf{U}\mathbf{x} + \mathbf{W}^o\mathbf{h} + \mathbf{b}^o$$
$$\mathbf{p} = softmax(\mathbf{o}),$$

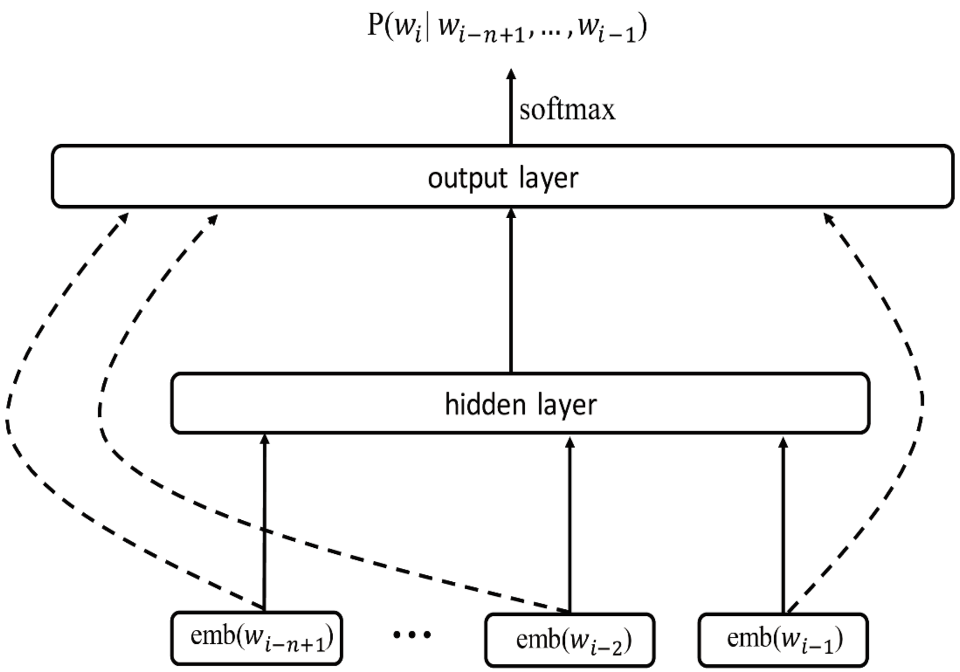


Short connection can empirically give better results.

Neural n-gram language modelling

- **Adding shortcut connection**

$$\mathbf{x} = \left(emb(w_{i-n+1}) \oplus \ldots \oplus emb(w_{i-1})\right)$$
$$\mathbf{h} = tanh(\mathbf{W}^n\mathbf{x} + \mathbf{b}^n)$$
$$\mathbf{o} = \mathbf{U}\mathbf{x} + \mathbf{W}^o\mathbf{h} + \mathbf{b}^o$$
$$\mathbf{p} = softmax(\mathbf{o}),$$



- **Training:**

Given training corpus $T=\{(w_1^i, w_2^i \ldots w_n^i)\}|_i$
We minimise the loss function:

$$L = -\frac{1}{|T|}\sum_{i=1}^{|T|} \log\left(P(w_n^i | w_1^i, \ldots, w_{n-1}^i)\right)$$

Contents

- 17.1 Neural language models and word embedding
  - 17.1.1 Neural n-gram language modelling
  - 17.1.2 Noise contrastive estimation
  - 17.1.3 Word Embeddings
- 17.2 Contextualized word representations
  - 17.2.1 Recurrent neural language models
  - 17.2.2 ELMo, GPT and BERT
  - 17.2.3 Using contextualized embeddings
- 17.3 Transfer learning
  - 17.3.1 Multi-task learning
  - 17.3.2 Shared-private network structure

WestlakeNLP

Noise contrastive estimation

WestlakeNLP

- The training objective in the last section can also ve viewed as maxmising:

$$J = \frac{1}{|T|}\sum_{i=1}^{|T|}\log\left(P(w_n^i|w_1^i, ..., w_{n-1}^i)\right)$$

- Given the training instance (w, c), the local derivative with respect to a model parameter θ is :

$$\begin{aligned}
\frac{\partial}{\partial\theta}J &= \frac{\partial}{\partial\theta}\log P(w|c) = \frac{\partial}{\partial\theta}\log\frac{\exp(\mathbf{o}[w])}{\sum_{w'\in V}\exp(\mathbf{o}[w'])} \\
&= \frac{\partial}{\partial\theta}\mathbf{o}[w] - \sum_{w'\in V}\frac{\exp(\mathbf{o}[w'])}{\sum_{w''\in V}\exp(\mathbf{o}[w''])}\frac{\partial}{\partial\theta}\mathbf{o}[w'] \\
&= \frac{\partial}{\partial\theta}\mathbf{o}[w] - \sum_{w'\in V}P(w'|c)\frac{\partial}{\partial\theta}\mathbf{o}[w']
\end{aligned}$$

Noise contrastive estimation

- However, to compute the probability cen be expensive due to enumeration of all words in the vocabulary where

$$P(w_n^i \mid w_{n-1}^i, \ldots, w_1^i) = o[w_n^i] = \frac{\exp(o[w_n^i])}{Z} = \frac{\exp(o[w_n^i])}{\sum_{w' \in V} \exp(p[w'])}$$

Noise contrastive estimation

WestlakeNLP

- **Noise contrastive estimation (NCE)**
- NCE approximates MLE by drawing positive (real) samples and negative (out of data) sample.

- In particular, we sample positive and negative samples from different distributions:

$$P(d, w|c) = \begin{cases} \frac{k}{1+k} & \times & Q(w) & \text{if } d = 0 \\ \frac{1}{1+k} & \times & \tilde{P}(w|c) & \text{if } d = 1 \end{cases}$$

d=1: positive samples; d=0:denotes negative samples
$\tilde{p}(w \mid c)$: empirical (data) distribution
Q: uniform or empirical unigram distribution

Noise contrastive estimation

WestlakeNLP

- According to :

$$P(d|c,w) = \frac{P(d,w|c)}{P(w|c)} = \frac{P(d,w|c)}{\sum_{d'\in\{0,1\}} P(d',w|c)}$$

$$P(d=0|c,w) = \frac{\frac{k}{1+k} \times Q(w)}{\frac{1}{1+k} \times \tilde{P}(w|c) + \frac{k}{1+k} \times Q(w)}$$

$$= \frac{k \times Q(w)}{\tilde{P}(w|c) + k \times Q(w)}$$

$$P(d=1|c,w) = \frac{\tilde{P}(w|c)}{\tilde{P}(w|c) + k \times Q(w)}$$

Noise contrastive estimation

WestlakeNLP

- To assume Z=1 in $P_\theta(w|c)=\dfrac{\exp(o[w])}{Z}$, We further have:

$$P(d=0|c,w)=\frac{k\times Q(w)}{\exp(\mathbf{o}[w])+k\times Q(w)}$$
$$P(d=1|c,w)=\frac{\exp(\mathbf{o}[w])}{\exp(\mathbf{o}[w])+k\times Q(w)}$$

And we get the final training objective using NCE:

$$J_{NCE}=\frac{1}{|T|}\sum_{i=1}^{|T|}\Big(\log P(d_i=1|c_i,w_i)+\sum_{j=1,w_j\sim Q}^{k}\log P(d_i=0|c_i,w_j)\Big)$$

Optimizing neural language models

WestlakeNLP

- **Speed the model:**

  A big computational bottleneck is the softmax function (output layer) over the whole vocabulary. NCE does not change the model itself.

- **Methods:**

  Two techniques can be introduced to make the model smaller:

  Hierarchical softmax and *log-bilinear model.*

Optimizing neural language models

WestlakeNLP

- **Hierarchical softmax**
  - We can arrange the vocabulary into a hierarchy of two layers, with the first layer containing M categories, and the second layer containing |V|/M words in each category.

$$\mathbf{p}^c = softmax(\mathbf{W}^c\mathbf{h} + \mathbf{b}^c)$$
$$\mathbf{p} = softmax(\mathbf{W}\mathbf{p}^c + \mathbf{b})$$

Size of the output layer:

$$|h| \times |V| \Rightarrow |\mathbf{h}| \times M + M \times |V|$$

Optimizing neural language models

WestlakeNLP

- **Log-bilinear model**
  - *emb*(w_i) is used directly for computing the probability through a bi-linear similarity function, which reduce the model size effectively.

$$\mathbf{c} = \sum_{j=i-n+1}^{i-1} s_j \cdot emb(w_j)$$

$$sim\Big(\mathbf{c}, emb(w_i)\Big) = \mathbf{c}^T \cdot emb(w_i)$$

$$\mathbf{p} = softmax\bigg(sim\Big(\mathbf{c}, emb(w)\Big)\bigg)$$

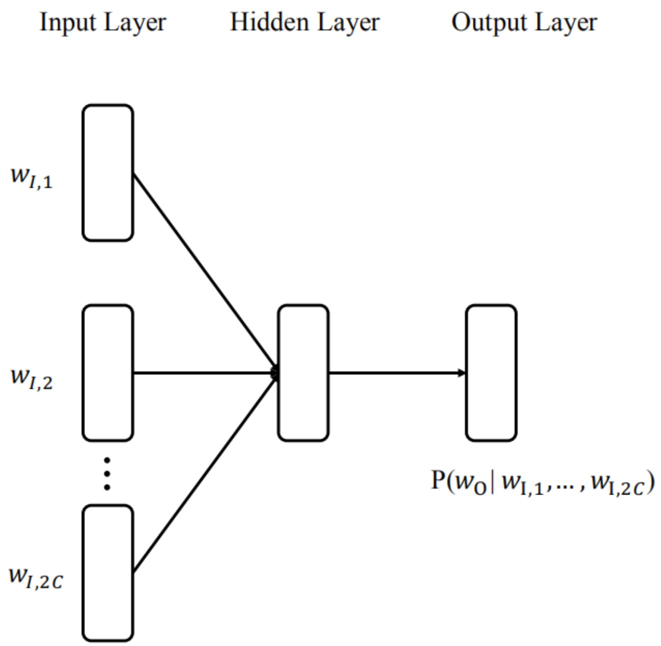
Contents

Distributed word representations

- **Continuous bag of words (CBOW)**

$$\mathbf{h} = \frac{1}{2C}\Big(emb(w_{I,1}) + ... + emb(w_{I,2C})\Big)$$
$$\mathbf{u}^o = emb'(w_O) \cdot \mathbf{h}$$
$$\mathbf{p} = softmax(\mathbf{u}^o),$$

where C is the window size for target word w_o, $w_{I,1}$, $w_{I,2}$, ..., $w_{I,2C}$ can be a surrounding window of w_o, *emb* and emb' represents context and target embeddings, respectively.



Distributed word representations

WestlakeNLP

- **Continuous bag of words (CBOW)**

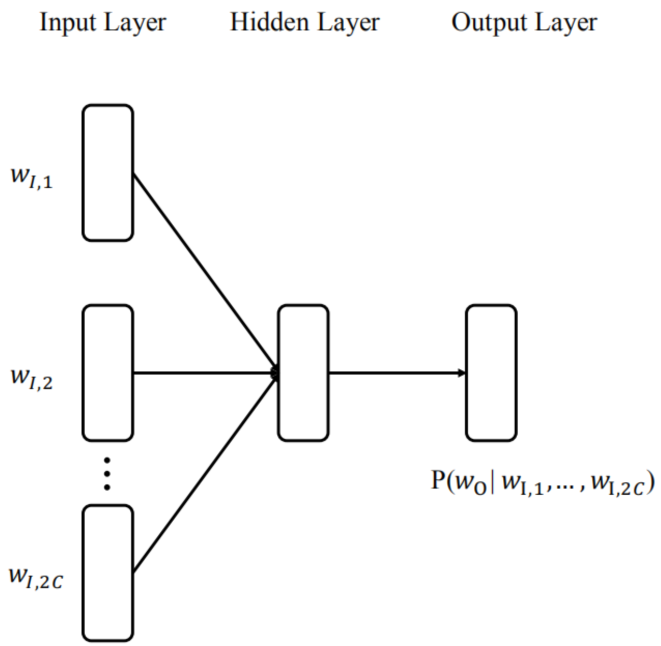
Training:

$$J = \frac{1}{T}\sum_{i=1}^{|T|}\left(logP(d_i = 1|c_i, w_O^i) + \sum_{j=1, w_j \sim Q}^{k} logP(d_i = 0|c_i, w_j)\right)$$

where

$$P(d = 1|c, w) = \frac{\exp(\mathbf{u}_o)}{\exp(\mathbf{u}_o) + k \times Q(w)}$$

$$P(d = 0|c, w) = \frac{k \times q(w)}{\exp(\mathbf{u}_o) + k \times Q(w)}$$

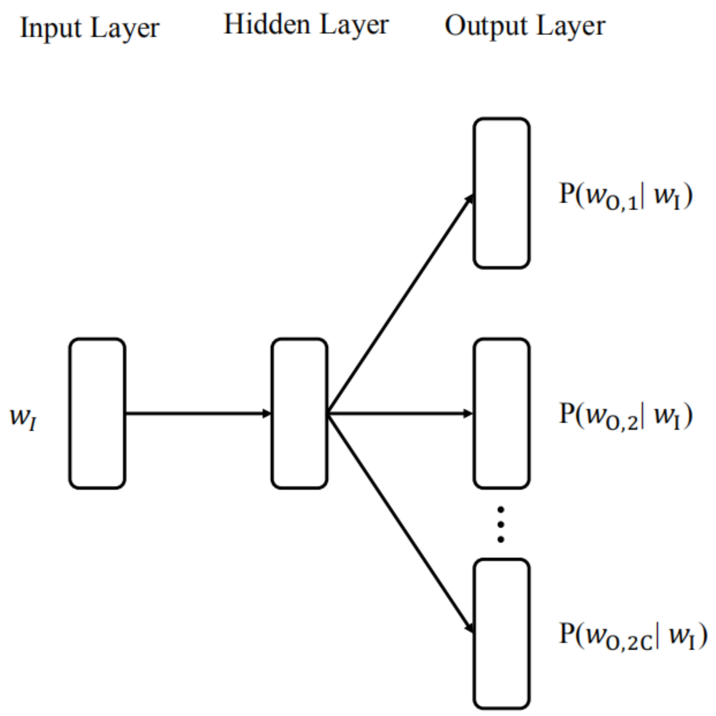


Here k is the number of negative samples, d = 1 represents a positive sample and d = 0 represents a negative sample.

Distributed word representations

- **Skip-gram**

$$\mathbf{u}_j^o = emb'(w_{O,j}) \cdot emb(w_I)$$
$$\mathbf{p} = softmax(\mathbf{u}_j^o),$$



where C is the window size for inout wor $w_{O,1}$, $w_{O,2}$, ..., $w_{O,2C}$ denote the context wor and emb' represents target and context word embeddings, respectively.

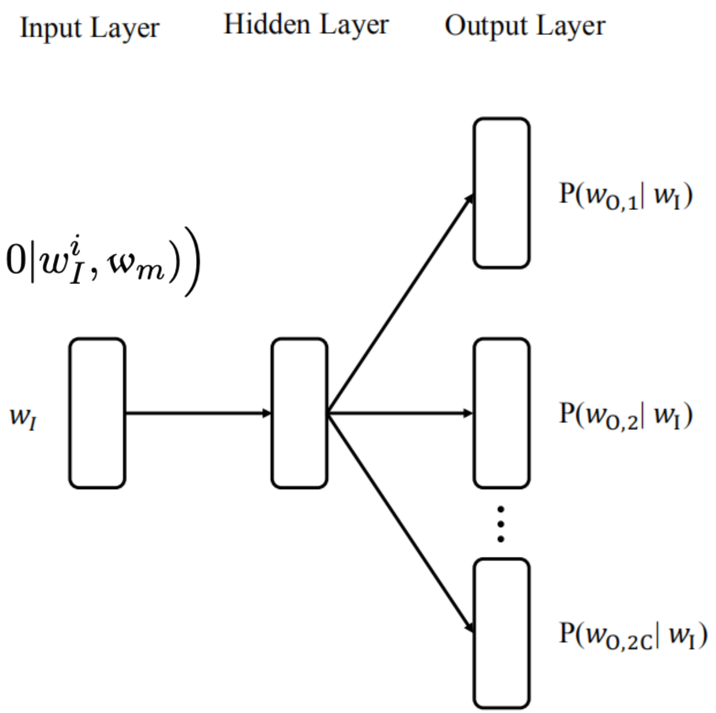
Distributed word representations

- **Skip-gram**

Training:

$$J = \frac{1}{|T|}\sum_{i=1}^{|T|}\sum_{j=1}^{2C}\Big(logP(d_i = 1|w_I^i, w_{O,j}^i) + \sum_{m=1, w_m \sim Q}^{k} logP(d_i = 0|w_I^i, w_m)\Big)$$

where
$$P(d = 1|w_I, w_{O,j}) = \frac{\exp(\mathbf{u}_{O,j})}{\exp(\mathbf{u}_{O,j}) + k \times Q(w_{O,j})}$$
$$P(d = 0|w_I, w_{O,j}) = \frac{k \times Q(w)}{\exp(\mathbf{u}_{O,j}) + k \times Q(w_{O,j})}$$



Here k is the number of negative samples, d = 1 represents a positive sample and d = 0 represents a negative sample.

Distributed word representation

WestlakeNLP

- **Conparison between CBOW and Skip-gram**

  - In CBOW, each target word is predicted once conditioned on the context window. The time complexity is O(|V|).

  - In skip-gram, in contrast, each target word is used to predict 2C context words, respectively, and therefore the time complexity is O(2C|V|).

  - During training, CBOW is faster than skip-gram. Skip-gram has been shown comparable or sightly more accurate empirically compared to CBOW.

Word embeddings using Global Statistics (GloVe)

WestlakeNLP

- CBOW and skip-gram make weak use global corpus-level information since they train word vectors based on local context windows.
- **GloVe** is a different approach to train embeddings on global word-word co-occurrence counts in a corpus.

Word embeddings using Global Statistics (GloVe)

- Use a matrix X denote word-word co-occurrence counts; X_{ij} represents the number of times word j occurs in the context of word i; emb and emb' to denote the target embedding and the context embedding.
- The word vectors are learned with so that the dot-product scales with the probabilities:

$$emb(w_i)^T emb'(w_j) + b_i + b_j = log(X_{ij})$$

Word embeddings using Global Statistics (GloVe)

- The training objective is to minimise:

$$L = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij}) \left(emb(w_i)^T emb'(w_j) + b_i + b_j - \log X_{ij} \right)^2$$

where V denotes the vocabulary, $f(X_{ij})$ is a weighting function.

Word embedding evaluation

- **Word similarities**
  - There is a corpora that contain words and their related words, each with a similarity score given by human expert.

word1	word2	similarity
computer	keyboard	7.62
computer	internet	7.58
plane	car	5.77
train	car	6.31

Word embedding evaluation

WestlakeNLP

- **Word similarities**
  - Finding the correlation between embedding based similarity scores and human-given scores using *Pearson correlation co-efficient*:

$$\rho_{(X,Y)} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

	model (X)	human (Y)
banking-investment	0.5	0.6
telephone-mobile	0.8	0.9
beer-drink	0.7	0.5
animal-human	0.7	0.8
horse-house	0.4	0.2
leg-finger	0.8	0.8

Word embedding evaluation

WestlakeNLP

- For the scores in the right table:

$$\mu_X = E[X] = \frac{0.5 + 0.8 + 0.7 + 0.7 + 0.4 + 0.8}{6} = 0.65$$

$$\mu_Y = E[Y] = \frac{0.6 + 0.9 + 0.5 + 0.8 + 0.2 + 0.8}{6} = 0.6333$$

$$\sigma_X = \sqrt{E[X^2] - E[X]^2} = 0.15$$

$$\sigma_Y = \sqrt{E[Y^2] - E[Y]^2} = 0.2357$$

$$\rho_{(X,Y)} = \frac{E[XY] - E[X]E[Y]}{\sqrt{E[X^2] - E[X]^2}\sqrt{E[Y^2] - E[Y]^2}} = 0.849$$

	model (X)	human (Y)
banking-investment	0.5	0.6
telephone-mobile	0.8	0.9
beer-drink	0.7	0.5
animal-human	0.7	0.8
horse-house	0.4	0.2
leg-finger	0.8	0.8

The Pearson correlation value of 0.849 shows that word embeddings well capture word relations.

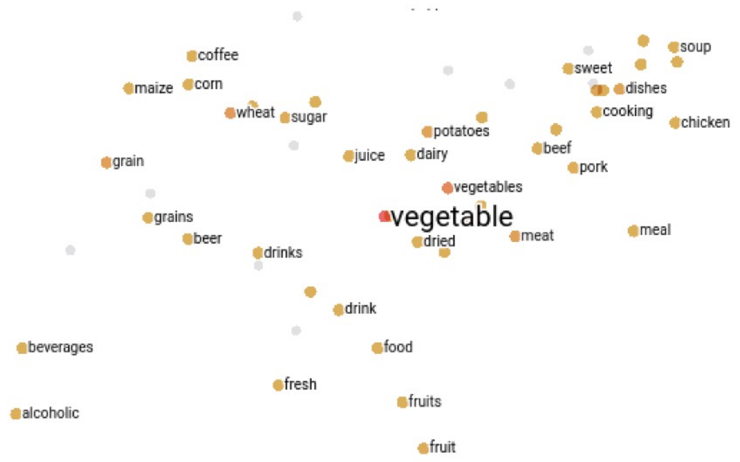
Word embedding evaluation

- **Word analogy**
  - Given a word pair “king - queen” and a third word “man”, and a fourth word is asked for making an analogy:
    - (king - queen) v.s. (man - ?)
  - The correct answer is “woman”.
  - Evaluate word embeddings, which allows the word vectors to follow:

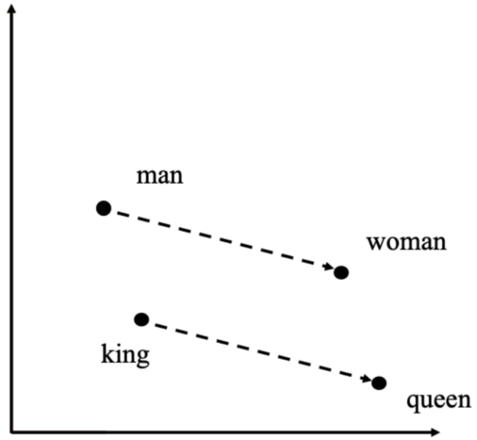
$$emb(\text{king}) - emb(\text{queen}) \approx emb(\text{man}) - emb(\text{woman})$$

Word embedding evaluation

- **Visualization**
  - A non-linear dimensionality reduction technique ***t-SNE***(t-Distributed Stochastic Neighbor Embedding) is used to preserve the distance correlation between points in the high-dimensional space and the projected two-dimensional space.



Similarity



Analogy

Embeddings and unknown words

- The vocabulary V_p for pre-trained embeddings and the vocabulary V_t for a dataset of the end task can be different.

- 1. $|V_p|$ is much larger.
- 2. There can also be words in V_t that do not exist in V_p.

Embeddings and unknown words

- V_p - V_t: *words in* V_p *but not* V_t
- 1) if pre-trained embeddings are not fine-tuned:
- consistent on the word embedding level for both in-domain and cross-domain test sets.
- 2) if pre-trained embeddings are fine-tuned:

  increase in-domain performance;
  decrease the generalisation power of the model across domains.

Embeddings and unknown words

- V_t - V_p : *words in* V_t *but not* V_p

- How to represent them?

- 1) assigned as <*UNK*> and set to **0** vector or a random vector.
- 2) learn the embedding during training.
- 3) derived from their characters and sub-words.

Character n-gram based word embedding

WestlakeNLP

- Suppose $w = c_1, \ldots, c_m$, $C_{ij} = c_i, \ldots, c_j$ denotes the subsequence of characters from position i to position j. The embedding of w can be written as:

$$
\begin{aligned}
emb(w) &= \mathbf{E}\mathbf{v}^{ng} \\
&= \mathbf{E}\Big(\sum_{i=1}^{m-n+1} \mathbf{1}[\text{IDX}(C_i^{i+n-1})]\Big) \\
&= \sum_{i=1}^{m-n+1} \mathbf{E} \cdot \mathbf{1}[\text{IDX}(C_i^{i+n-1})] \\
&= \sum_{i=1}^{m-n+1} emb^c(C_i^{i+n-1})
\end{aligned}
$$

Tri-grams for word “embedding” :

emb('embedding')
= *emb*('emb')+*emb*('mbe')+
emb('bed')+*emb*('edd')+
emb('ddi')+*emb*('din')+
emb('ing')

Character n-gram based word embedding

WestlakeNLP

- Summing up n-gram embeddings of different length:

$$emb(w) = \sum_{n=1}^{4} \sum_{i=1}^{m-n+1} emb^c(C_i^{i+n-1})$$

Contents

Recurrent neural language models

- RNN LM use a history from beginning of the sentence until the previous word to predict the target word.

- First, CNN is used to represent each word.

$$\begin{aligned}
\mathbf{x}_i &= [emb^c(c_1^i); \ldots; emb^c(c_{||w_i||}^i)] \\
\mathbf{H}_i^c &= \textsc{Conv}(\mathbf{x}_i, k, d_c) \\
\mathbf{h}_i^c &= \textsc{MaxPooling}(\mathbf{H}_i^c) \\
emb(w_i) &= \textsc{Highway}(\mathbf{h}_i^c),
\end{aligned}$$

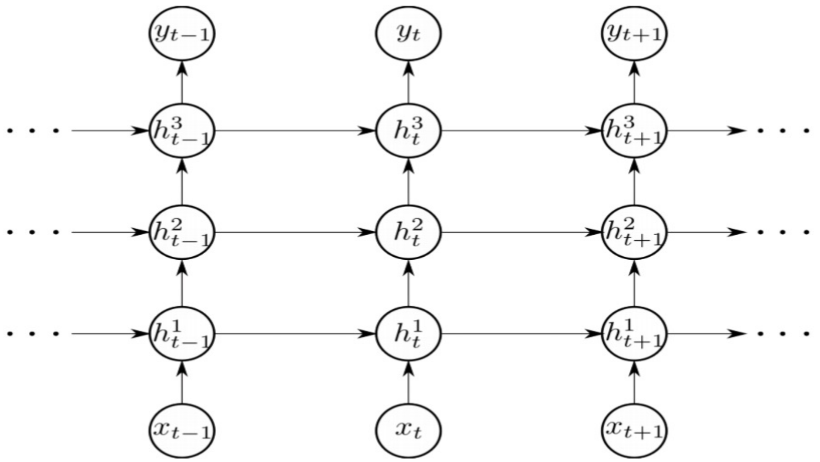
Recurrent neural language models

- On the sequence level, $P(w_i \mid w_1, w_2, \ldots, w_{i-1})$ is calculated by:

$$\mathbf{h}_i = \text{RNN\_STEP}(emb(w_{i-1}), \mathbf{h}_{i-1})$$
$$\mathbf{p} = softmax(\mathbf{W}^o \mathbf{h}_i),$$

- Stacking multiple layers:

$$\mathbf{h}_i^0 = \text{RNN\_STEP}(emb(w_{i-1}), \mathbf{h}_{i-1}^0)$$
$$\mathbf{h}_i^j = \text{RNN\_STEP}(\mathbf{h}_i^{j-1}, \mathbf{h}_{i-1}^j) \quad j \in [1, \ldots, k]$$
$$\mathbf{p} = softmax(\mathbf{W}^o \mathbf{h}_i^k),$$



Conventional Stacked RNN

Figure Credit: https://cs224d.stanford.edu/reports/Lambert.pdf

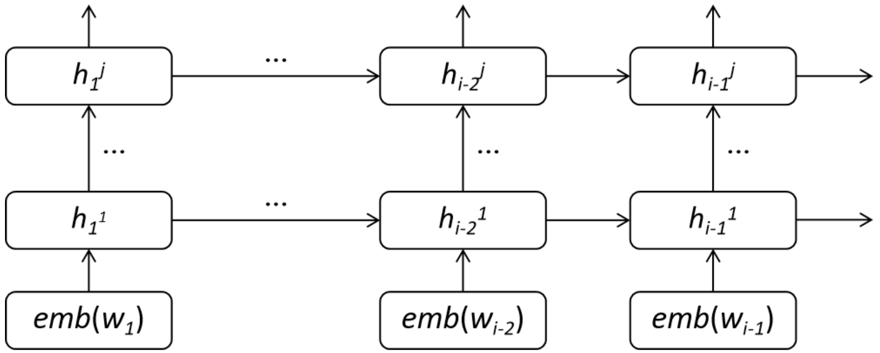
Contents

ELMo

- The contextualized word embeddings can be computed by integrating multiple hidden layers:

$$\mathbf{H} = \sum_{j=1}^{k} s^j \mathbf{H}^j$$

Considering bi-directional information:



$$\mathbf{H}^j = [\overrightarrow{\mathbf{h}}_1^j \oplus \overleftarrow{\mathbf{h}}_1^j; \overrightarrow{\mathbf{h}}_2^j \oplus \overleftarrow{\mathbf{h}}_2^j; \ldots; \overrightarrow{\mathbf{h}}_n^j \oplus \overleftarrow{\mathbf{h}}_n^j]$$

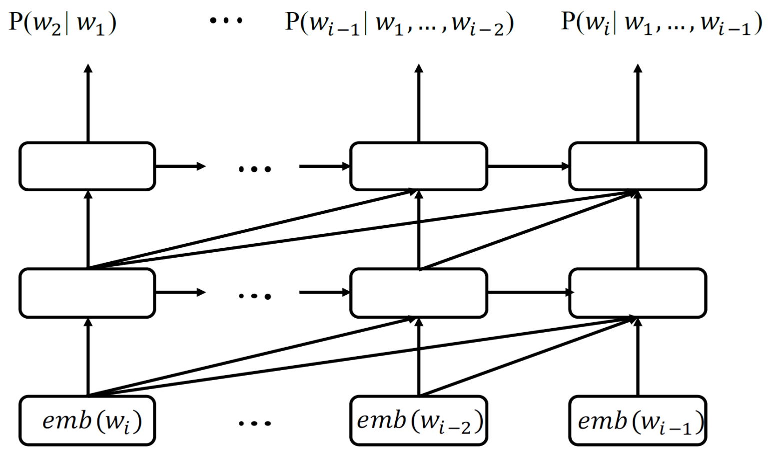
The embeddings above is also referred to as Embeddings from Language Models (**ELMo**).

GPT

- SANs being an alternative sequence encoder choice to RNNsThis method has been used as a principle behind the Generative Pre-Training (**GPT**) model.

- Given a sequence of words $W_1^{i-1} = w_1, \ldots, w_{i-1}$, a k-layers SAN LM predicts the next word w_i by:

$$\mathbf{H}^0 = [emb(w_1); \ldots; emb(w_{i-1})] + \mathbf{V}^p$$
$$\mathbf{H}^j = \textsc{San\_Encoder}(\mathbf{H}^{j-1}) \quad j \in [1, \ldots, k]$$
$$\mathbf{p} = softmax(\mathbf{W}\mathbf{h}_i^k),$$



GPT

- In the last equation, V^p is a *position embedding* matrix.

$$\mathbf{V}^p[i][2j] = sin(i/10000^{2j/|\mathbf{V}^p|})$$
$$\mathbf{V}^p[i][2j+1] = cos(i/10000^{2j/|\mathbf{V}^p|})$$
$$j \in [1, ..., |\mathbf{V}^p|/2],$$

where $2j$ and $2j + 1$ denotes an element in V^p[i].

BERT

WestlakeNLP

- **Masked langauge modeling (MLM).**
- predicting the mask word using left and right context.

e.g. I went to the <mask> to get some food.

“restaurant”, “pizzeria”,“store”, “cafe” or “pub”…

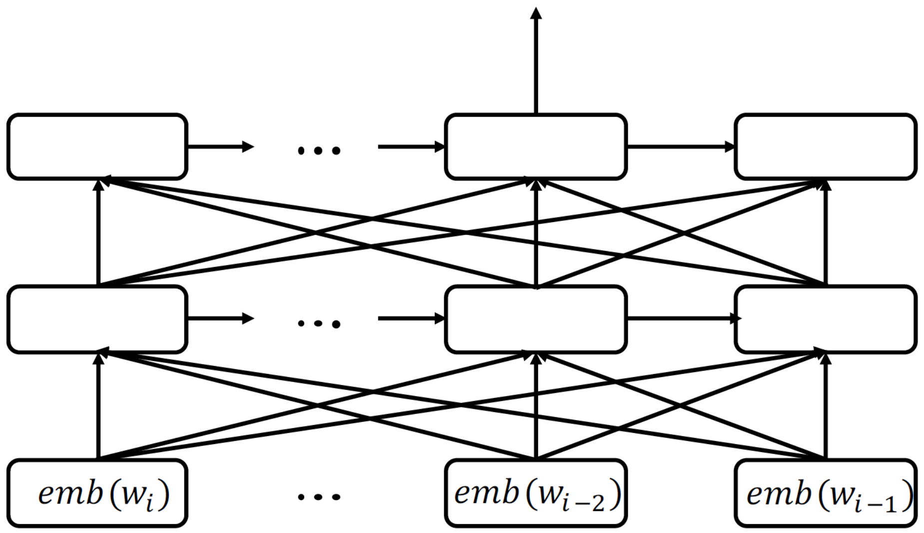
BERT

- Instead of using the history information, we predict w_i using MLM by:

$$\mathbf{H}^0 = [emb(w_1); ...; emb(w_{i-1}); emb(\langle\text{MASK}\rangle); emb(w_{i+1}); ...; emb(w_n)] + \mathbf{V}^p$$

$$\mathbf{H}^j = \text{SAN\_ENCODER}(\mathbf{H}^{j-1}) \quad j \in [1, ..., k]$$

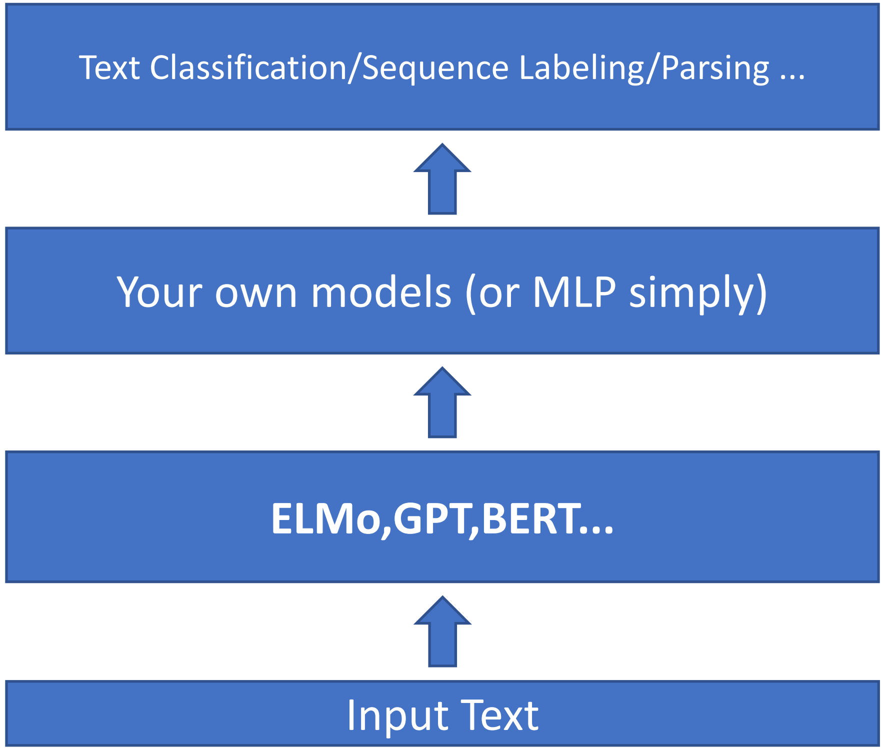
$$\mathbf{p} = softmax(\mathbf{W}\mathbf{h}_i^k),$$



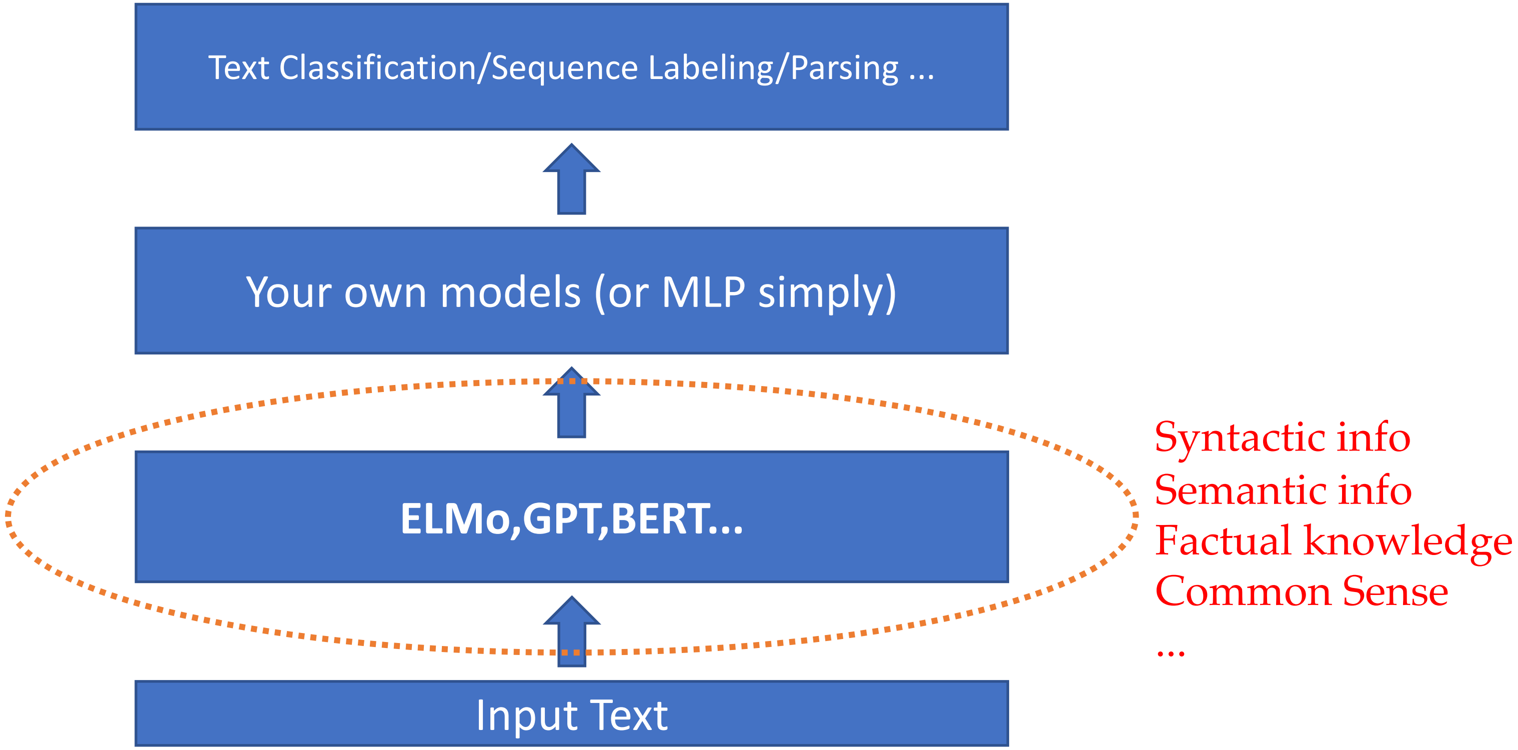
The above contextualised embedding method has been used as a principle behind the Bidirectional Encoder Representations from Transformers (**BERT**) model.

Contents

Using contextualized embeddings



Using contextualized embeddings



Contents

Transfer learning

- **Pretraining from LM**

  Skip Gram, CBOW, ELMo, GPT, BERT

- **Pretraining beyond LM**

  Transferring knowledge from resource-rich tasks, domains, languages and annotation standards to low-resource ones.

  e.g

  syntactic resources pretraining for semantic role labeling.
  corpora in news domain pretraining for low-resources social media.
  English-French machine translation pretraining for English-Uzbek.

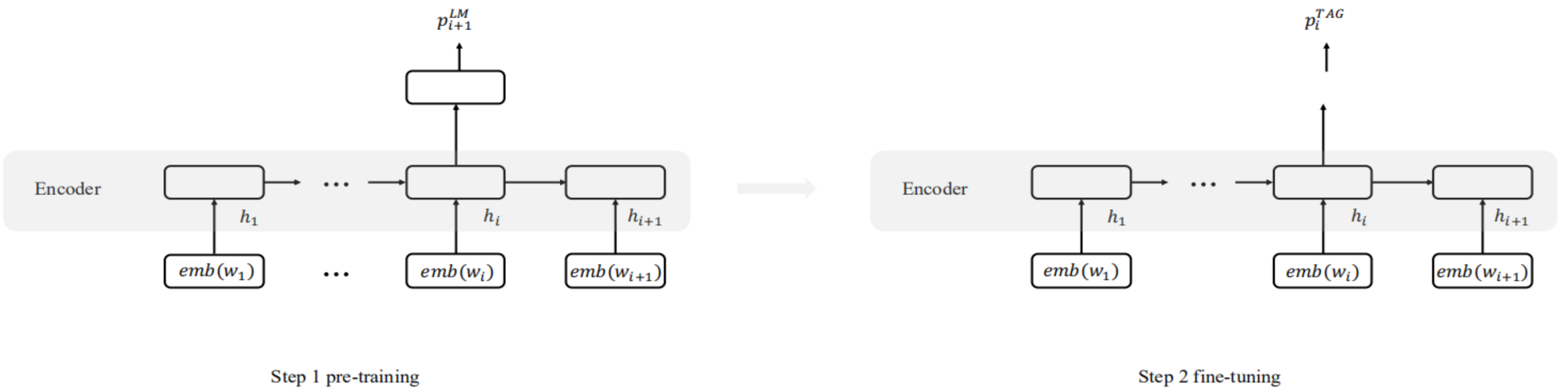
Multi-task learning

- Multi-task learning (**MTL**) exploits parameter sharing for seeking mutual benefits between tasks.

  e.g
- knowledge from language modeling is transferred to sequence labeling.

$$\mathbf{p}_{i+1}^{\mathrm{LM}} = softmax(\mathbf{W}^{\mathrm{LM}}\mathbf{h}_i + \mathbf{b}^{\mathrm{LM}})$$

$$\mathbf{p}_{i}^{\mathrm{TAG}} = softmax(\mathbf{W}^{\mathrm{TAG}}\mathbf{h}_i + \mathbf{b}^{\mathrm{TAG}})$$



(a) language modeling pre-training for better sequence labeling.

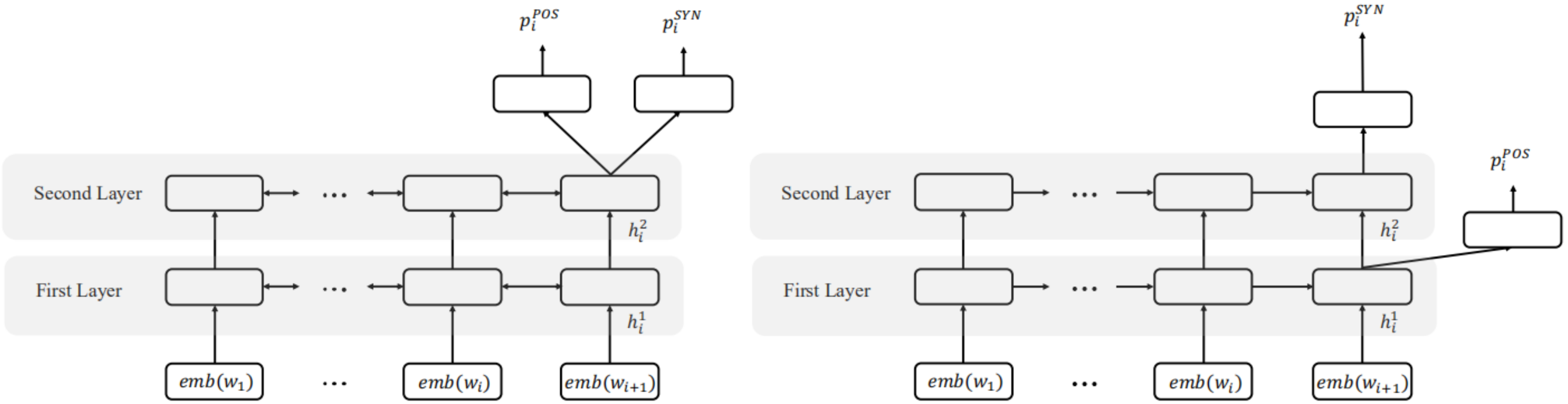
1)We do the LM pre-training.

2)Sequence labeling can benefit from LM with more informed starting parameters.

Selecting parameters for sharing

- **Selecting shared layers.**

Different infomation can be encoded in different layers.



(a) standard sharing. (b) flexible sharing.

Sequence labeling benefits from more shallow contextual information, which is better captured in lower LSTM layers.

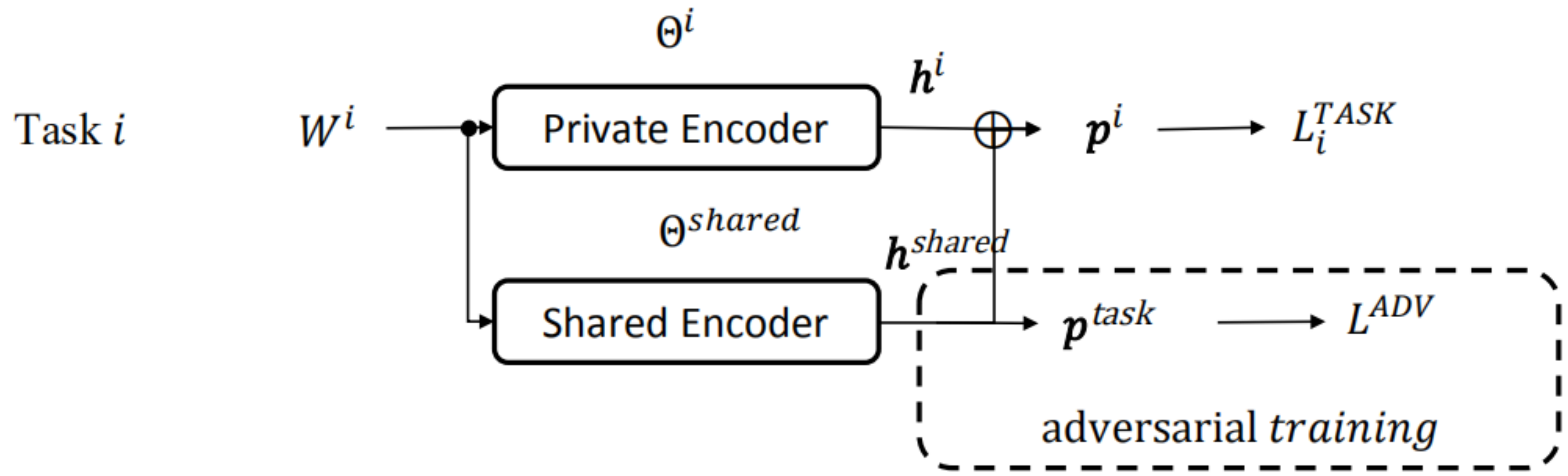
Shared-private network structure

WestlakeNLP

- Sharing model parameters across tasks can potentially suffer from **information conflict**.
- We want to minimise such conflict while maximising the mutual benefit between tasks via common information.
- Solution:
  - learn a set of **shared** parameters across tasks,
  - while keeping a **separate** copy of parameters for each task.

Shared-private network structure

- **Shared/Separated representation for each tasks**



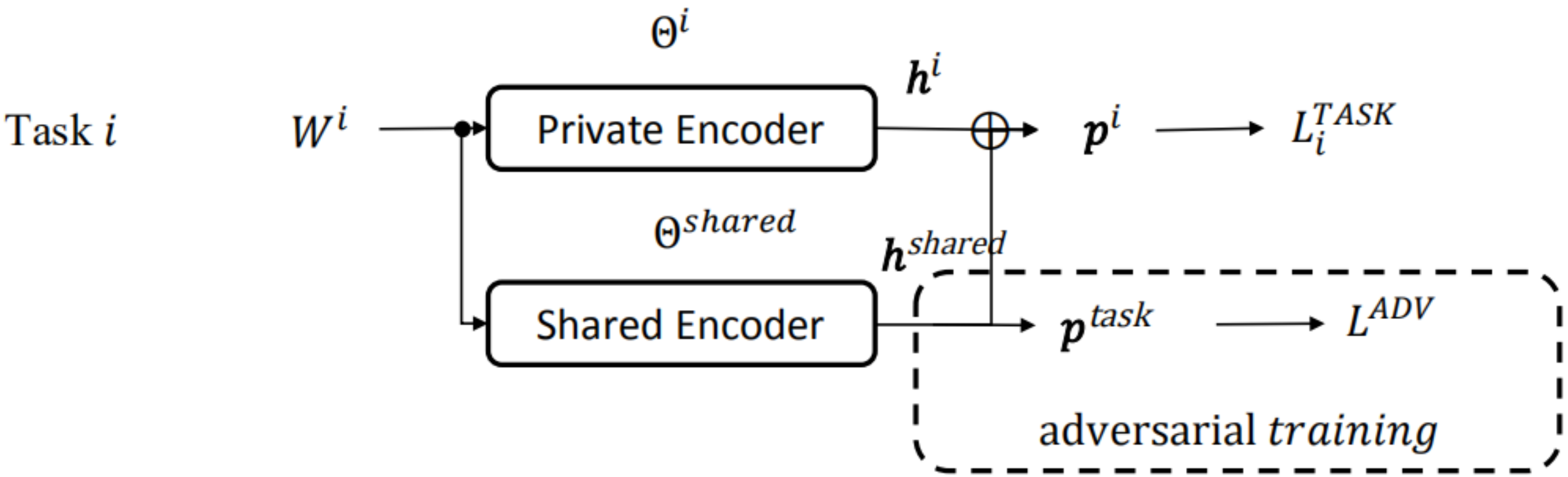
$$\mathbf{h}^{shared} = \text{ENCODER}(W^i, \Theta^{shared})$$

$$\mathbf{h}^i = \text{ENCODER}(W^i, \Theta^i)$$

$$\mathbf{p}^i = g(\mathbf{h}^{shared} \oplus \mathbf{h}^i)$$

Shared-private network structure

- **Loss for each tasks**

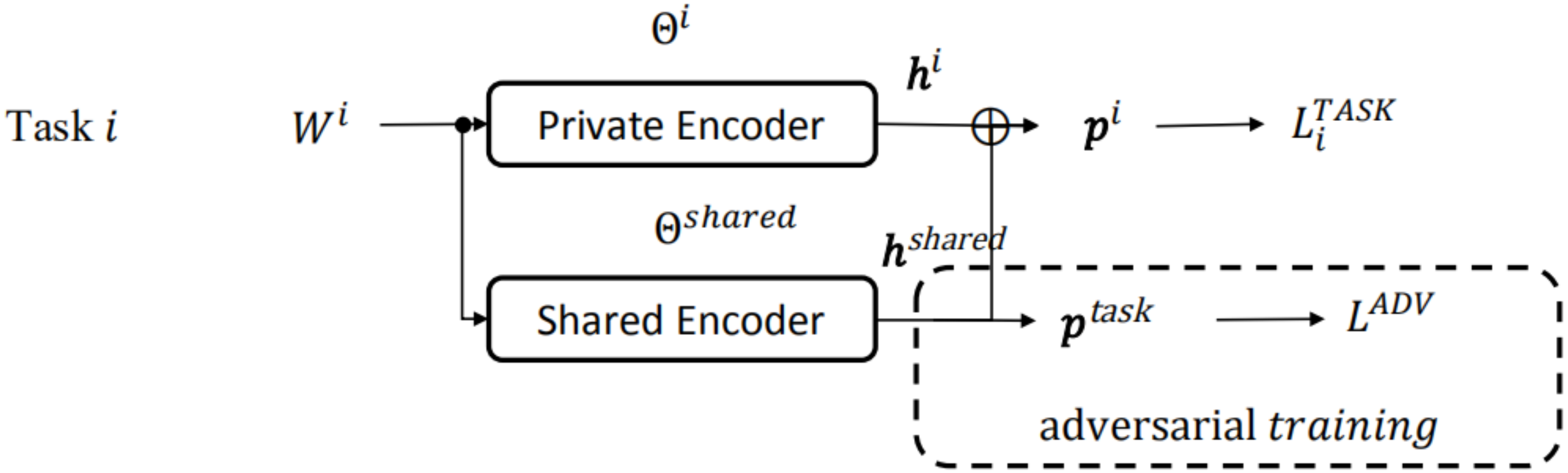


$$L^{\text{TASK}} = \sum_{i=1}^{M} L_i^{\text{TASK}} = \sum_{i=1}^{M} \sum_{j=1}^{|D_i|} - \log \mathbf{p}_j^i [y_j^i]$$

Shared-private network structure

- **Adversarial Training**

To ensure that h^{shared} contains no task-specific information through p^{task}

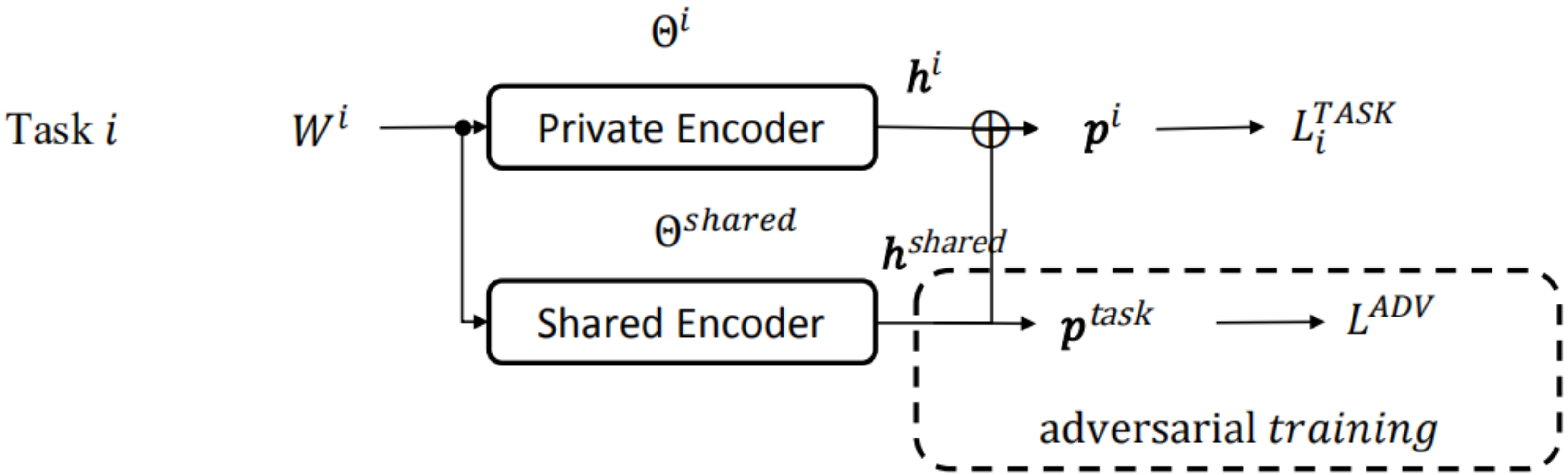


$$L^{\text{ADV}} = \sum_{i=1}^{M} \sum_{j=1}^{|D_i|} \log(\mathbf{p}^{task})_j^i[i]$$

Shared-private network structure

WestlakeNLP

- **Total Loss**



$$L = \sum_{i=1}^{M} L_i^{\text{TASK}} + L^{\text{ADV}}$$

Summary

WestlakeNLP

- Neural n-gram language models and recurrent neural language models;
- Noise contrastive estimation;
- Word embeddings as distributed word representations;
- Contextualized word embeddings;
- Pre-training and transfer learning.