Examples II

Eugene Morozov

(Eugene@HiEugene.com)

Contents

Part II
Natural Language Processing / Natural Language Understanding

6 NLP/NLU

6.1 Entropy

Let $p(x)$ be the probability mass function of a random variable X, over a discrete set of symbols (or alphabet) $\mathcal{X}$:

$$p(x) = P(X = x),\, x \in \mathcal{X}$$

The entropy (or self-information) is the average uncertainty of a single random variable:

$$H(p) = H(X) = -\sum_{x\in\mathcal{X}} p(x)\log_2 p(x) = \sum_{x\in\mathcal{X}} p(x)\log_2\frac{1}{p(x)} \tag{6.1}$$

Entropy measures the amount of information in a random variable. It is normally measured in bits (hence the log to the base 2), but using any other base yields only a linear scaling of results. Also, for this definition to make sense, we define $0\log 0 = 0$. It makes more sense to interpret (6.1) as

$$H(X) = \mathbb{E}\left[\log_2\frac{1}{p(x)}\right]$$

One can also think of entropy in terms of the *Twenty Questions* game. For example, for a dice if you ask yes/no questions like "Is it over 3?" or "Is it an even number?" then on average you will need to ask 2.6 questions to identify each number with total certainty (assuming that you ask good questions!). In other words, entropy can be interpreted as a measure of the size of the "search space" consisting of the possible values of a random variable and its associated probabilities.

The joint entropy of a pair of discrete random variables $X, Y \sim p(x,y)$ is the amount of information needed on average to specify both their values. It is defined as:

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(x,y)$$

The **conditional entropy** of a discrete random variable Y given another X, for $X, Y \sim p(x,y)$, expresses how much extra information you still need to supply on average to communicate Y given that the other party knows X:

$$\begin{aligned}
H(Y|X) &= \sum_{x\in\mathcal{X}} p(x) H(Y|X=x) \\
&= \sum_{x\in\mathcal{X}} p(x) \left[-\sum_{y\in\mathcal{Y}} p(y|x)\log_2 p(y|x) \right] \\
&= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log_2 p(y|x)
\end{aligned} \tag{6.2}$$

By the chain rule for entropy,

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

Therefore,

$$H(X) - H(X|Y) = H(Y) - H(Y|X)$$

This difference is called the **mutual information** between X and Y. It is the reduction in uncertainty of one random variable due to knowing about another, or in other words, the amount of information one random variable contains about another.

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) \\
&= H(X) + H(Y) - H(X,Y) \\
&= \sum_{x,y} p(x,y)\log_2 \frac{p(x,y)}{p(x)p(y)} \\
&= \mathbb{E}_{p(x,y)}\left[\log_2 \frac{p(x,y)}{p(x)p(y)}\right] \\
&= KL\left(p(x,y)\|p(x)p(y)\right)
\end{aligned} \tag{6.3}$$

Consider Simplified Polynesian language. This language has 6 letters. The simplest code is to use 3 bits for each letter of the language. This is equivalent to assuming that a good model of the language (where our ‘model’ is simply a probability distribution) is a uniform model. However, we noticed that not all the letters occurred equally often, and, noting these frequencies, produced a zeroth order model of the language. This had a lower entropy of 2.5 bits per letter (and we showed how this observation could be used to produce a more efficient code for transmitting the language). Thereafter, we noticed the syllable structure of the language, and developed an even better model that incorporated that syllable structure into it. The resulting model had an even lower entropy of 1.22 bits per letter. The essential point here is that if a model captures more of the structure of a language, then the entropy of the model should be lower. In other words, we can use entropy as a measure of the quality of our models.

Statistical NLP problems as decoding problems:

Application	Input	Output	$p(i)$	$p(o\|i)$
Machine Translation	L_1 word sequences	L_2 word sequences	$p(L_1)$ in a language model	translation model
Optical Character Recognition (OCR)	actual text	text with mistakes	prob of language text	model of OCR errors
Part Of Speech (POS) tagging	POS tag sequences	English words	prob of POS sequences	$p(w\|t)$
Speech recognition	word sequences	speech signal	prob of word sequences	acoustic model

n-gram models represent k^{th} order Markov chain approximation:

$$\begin{gathered} P(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) \\ \approx P(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_{n-k} = x_{n-k}) \end{gathered}$$

For the probability of occurrence of particular words w_1 and w_2 we can define

$$\begin{aligned} I(w_1; w_2) &= \log_2 \frac{P(w_1 w_2)}{P(w_1)P(w_2)} \\ &= \log_2 \frac{P(w_1|w_2)}{P(w_1)} \\ &= \log_2 \frac{P(w_2|w_1)}{P(w_2)} \end{aligned} \tag{6.4}$$

where $P(w_1) = \mathit{count\,of\,word\,w_1\,/\,total\,number\,of\,words}$, and for $P(w_1 w_2)$ we use the collocation frequency, i.e. bigram. In information theory, mutual information is more often defined as holding between random variables (cf (6.3)), not values of random variables as we have defined it here. The 2nd equation in (6.4) gives us the amount of information provided by the occurrence of the event represented by $[w_2]$ about the occurrence of the event represented by $[w_1]$. For example, the mutual information measure tells us that the amount of information we have about the occurrence of "Prime" at position i in the corpus increases by certain number of bits if we are told that "Minister" occurs at position $i+1$. We could also say that our uncertainty is reduced by that number of bits.

Information radius

$$IRad = KL\left(p \| \frac{p+q}{2}\right) + KL\left(q \| \frac{p+q}{2}\right)$$

(or total divergence to the average) is symmetric ($IRad(p,q) = IRad(q,p)$) and there is no problem with infinite values since $\frac{p_i+q_i}{2} \neq 0$ if either $p_i \neq 0$ or $q_i \neq 0$.

The intuitive interpretation of $IRad$ is that it answers the question: How much information is lost if we describe the two words (or random variables in the general case) that correspond to p and q with their average distribution? $IRad$ ranges from 0 for identical distributions to $2\log 2$ for maximally different distributions. As usual we assume $0 \log 0 = 0$.

Word (or term) occurrence frequency can be used to represent a document as a vector in n-dimensional word (phrase) space. In this space "close" often means small angle between vectors (without magnitude). To search for collocations the tuple: (word occurrence frequency, position), where position is simply an offset from beginning, can be used.

Cosine measure or normalized correlation coefficient

$$cos(\overrightarrow{q}, \overrightarrow{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}$$

where $\overrightarrow{q}$ is the query vector, e.g. "car insurance", $\overrightarrow{d}$ is a document vector.

An interesting property of the cosine is that, if applied to normalized vectors, it will give the same ranking of similarities as Euclidean distance does.

$$\begin{aligned}
|\overrightarrow{x} - \overrightarrow{y}|^2 &= \sum_{i=1}^{n}(x_i - y_i)^2 \\
&= \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} y_i^2 \\
&= 1 - 2\sum_{i=1}^{n} x_i y_i + 1 \\
&= 2(1 - \sum_{i=1}^{n} x_i y_i)
\end{aligned}$$

So for a particular query $\overrightarrow{q}$ and any 2 documents $\overrightarrow{d}_1$ and $\overrightarrow{d}_2$ we have:

$$cos(\overrightarrow{q}, \overrightarrow{d}_1) > cos(\overrightarrow{q}, \overrightarrow{d}_2) \Leftrightarrow |\overrightarrow{q} - \overrightarrow{d}_1| < |\overrightarrow{q} - \overrightarrow{d}_2|$$

If the vectors are normalized (i.e. $\sqrt{\sum_{i=1}^{n} d_i^2} = 1$), we can compute the cosine as a simple dot product. Normalization is generally seen as a good thing – otherwise longer vectors (corresponding to longer documents) would have an unfair advantage and get ranked higher than shorter ones.

Term weighting:

quantity	symbol	definition
term frequency	$tf_{i,j}$	number of occurrences of w_i in d_j
document frequency	df_i	number of documents in the collection that w_i occurs in
collection frequency	cf_i	total number of occurrences of w_i in the collection

Term frequency is usually dampened by a function like $f(tf) = \sqrt{tf}$ or $f(tf) = 1 + \log(tf)$, $tf > 0$ because more occurrences of a word indicate higher importance, but not as much importance as the undampened count would suggest.

One way to combine a word's term frequency $tf_{i,j}$ and document frequency df_i into a single weight is as follows:

$$weight(i,j) = \begin{cases} (1+\log tf_{i,j}) \log \frac{N}{df_i} & if\, tf_{i,j} \geq 1 \\ 0 & if\, tf_{i,j} = 0 \end{cases}$$

where N is the total number of documents. This is so-called inverse document frequency (idf) weighting. Thus, it produces *term frequency - inverse document frequency* (tf.idf). There are several weighting configurations.

It can be shown that mutual information between random variables $\mathcal{T}$ and $\mathcal{D}$ corresponding to respectively drawing a document or a term

$$I(\mathcal{T};\mathcal{D}) = \frac{1}{|D|} \sum_{t,d} tf(t,d) \cdot idf(t)$$

This expression shows that summing the tf.idf of all possible terms and documents recovers the mutual information between documents and term taking into account all the specificities of their joint distribution. Each tf.idf hence carries the "bit of information" attached to a term w document pair.

The probability that a word w_i occurs a particular number of times k in a document can be modeled (although sometimes inaccurately) by the Poisson distribution:

$$p(k;\lambda_i) = e^{-\lambda_i} \frac{\lambda_i^k}{k!}$$

where $\lambda_i > 0$ is the average number of occurrences of w_i per document, that is, $\lambda_i = cf_i/N$ where N is the total number of documents in the collection.

A simpler distribution that fits empirical word distributions about as well as the negative binomial is Katz's K mixture:

$$P_i(k) = (1-\alpha)\delta_{k,0} + \frac{\alpha}{\beta+1}\left(\frac{\beta}{\beta+1}\right)^k$$

where $\delta_{k,0} = 1$ iff $k = 0$ and $\delta_{k,0} = 0$ otherwise and α and β are parameters that can be fit using the observed mean λ and the observed inverse document frequency idf as follows.

$$\begin{aligned} \lambda &= \frac{cf}{N} \\ idf &= \log_2 \frac{N}{df} \\ \beta &= \lambda \times 2^{idf} - 1 = \frac{cf - df}{df} \\ \alpha &= \frac{\lambda}{\beta} \end{aligned}$$

The parameter β is the number of "extra terms" per document in which the term occurs (compared to the case where a term has only one occurrence per document). The decay factor $\frac{\beta}{\beta+1} = \frac{cf-df}{cf}$ (extra terms per term occurrence) determines the ratio $\frac{P_i(k)}{P_i(k-1)}$. For example, if there are 1/10 as many extra terms as term occurrences, then there will be ten times as many documents with 1 occurrence as with 2 occurrences and ten times as many with 2 occurrences as with 3 occurrences. If there are no extra terms ($cf = df \Rightarrow \frac{\beta}{\beta+1} = 0$), then we predict that there are no documents with more than 1 occurrence.

The parameter α captures the absolute frequency of the term. Two terms with the same β have identical ratios of collection frequency to document frequency, but different values for α if their collection frequencies are different.

6.2 Latent Semantic Indexing

LSI is a technique that projects queries and documents into a space with "latent" semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms – as long as their terms are semantically similar according to the co-occurrence analysis. We can look at LSI as a similarity metric that is an alternative to word overlap measures like tf.idf.

Latent Semantic Indexing is closely related to Principal Component Analysis (PCA), another technique for dimensionality reduction. Take a term-by-document matrix (where rows are words and columns are documents; values are weighted occurrences) and perform an SVD. SVD (and hence LSI) is a least-squares method. The projection into the latent semantic space is chosen such that the representations in the original space are changed as little as possible when measured by the sum of the squares of the differences.

One objection to SVD is that, along with all other least-squares methods, it is really designed for normally-distributed data. But often other distributions like Poisson or negative binomial are more appropriate for term counts. One problematic feature of SVD is that, since the reconstruction $\hat{A}$ of the term-by-document matrix A is based on a normal distribution, it can have negative entries, clearly an inappropriate approximation for counts. A dimensionality reduction based on Poisson would not predict such impossible negative counts.

$\triangle$ **Wikipedia**.

Out of 10,460,720 English Wikipedia articles I've selected 288 articles in 2 areas: statistics and topology. They contain 14,427 terms. After performing SVD $K = 20$ dimensions were retained, i.e. the term-by-documents matrix A is represented as

$$A = TSD^T \tag{6.5}$$

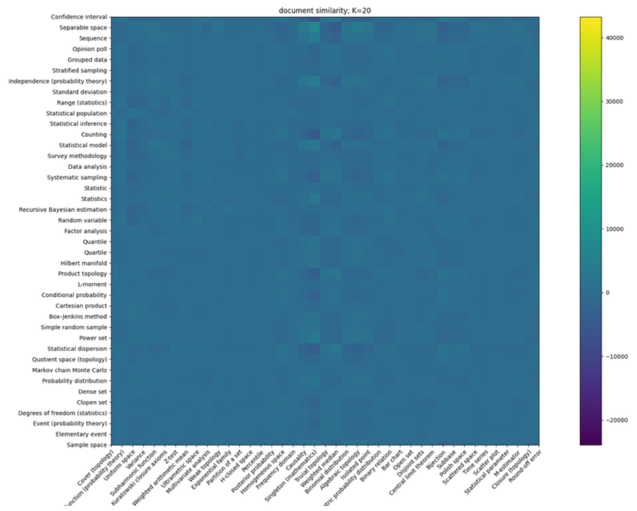
where T is the term matrix and D is the document matrix.

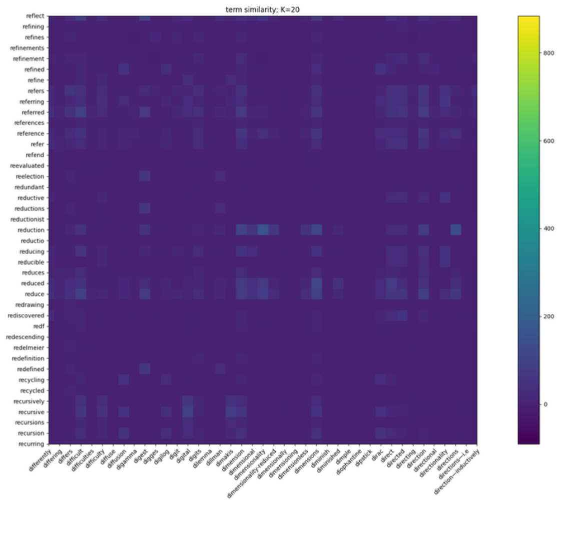
A query can be thought as a mini document. The search is based on (6.5) being equivalent to

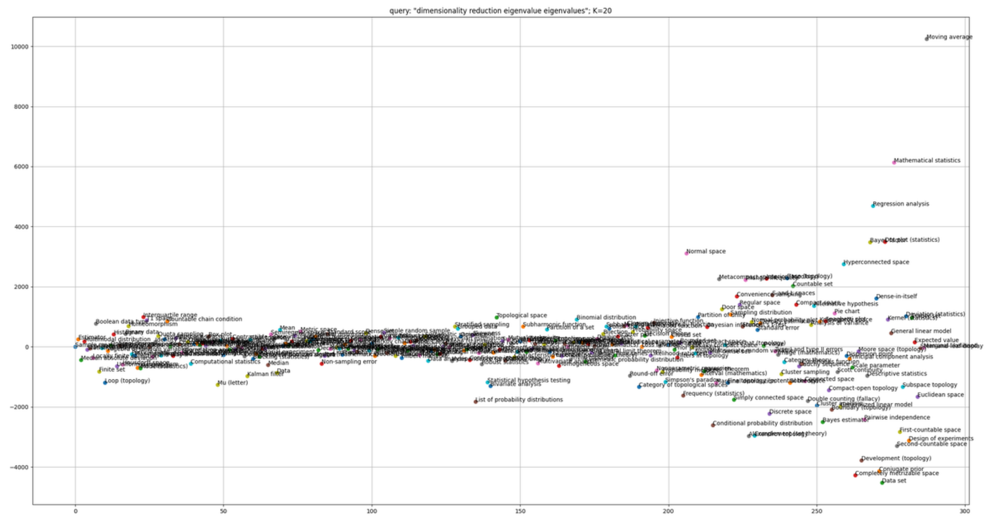
$$T^T A = SD^T$$

So we just multiply the query or document vector with the transpose of the term matrix T (after it has been truncated to the desired dimensionality). For example, for a query vector $\overrightarrow{q}$ and a reduction to dimensionality k, the query representation in the reduced space is $T^T_{m \times k}\overrightarrow{q}$. In other words, we project the query to the reduced dimensionality space and find the closest (by dot product) documents.

$(DS)(DS)^T$ gives us the correlation or document similarity; the same is for term similarity: $(TS)(TS)^T$.







▲

6.3 Classification Models

Contingency table for evaluating a binary classifier:

	YES is correct	NO is correct
YES was assigned	a	b
NO was assigned	c	d

then

accuracy	$\frac{a+d}{a+b+c+d}$
precision	$\frac{a}{a+b}$
recall	$\frac{a}{a+c}$
fallout	$\frac{b}{b+d}$

6.3.1 Decision Trees

For text categorization, the information retrieval vector space model is frequently used as the data representation. That is, each document is represented as a vector of (possibly weighted) word counts.

1. Pick $K = 20$ words whose χ^2-test score is the highest for a category.
2. Each document is then represented as a vector of K integers, $\vec{x}_j = (s_{1j}, \ldots, s_{Kj})$, where s_{ij} is computed as the following quantity:

$$s_{ij} = 10\frac{1 + \log tf_{ij}}{1 + \log l_j}$$

where tf_{ij} is the number of occurrences of term i in document j and l_j is the length of document j. The score s_{ij} is set to 0 for no occurrences of the term.

3. Use *grow* and *prune* technique to train the model. The *splitting criterion* which we can use is to split the objects at a node into two piles in the way that gives us maximum information gain. **Information gain** is an information-theoretic measure defined as the difference of the entropy of the mother node and the weighted sum of the entropies of the child nodes:

$$G(a, y) = H(t) - H(t|a) = H(t) - (p_L H(t_L) + p_R H(t_R))$$

where a is the attribute we split on, y is the value of a we split on, t is the distribution of the node that we split, p_L and p_R are the proportion of elements that are passed on to the left and right nodes, and t_L and t_R are the distributions of the left and right nodes. Information gain is intuitively appealing because it can be interpreted as measuring the reduction of uncertainty.

6.3.2 Maximum Entropy Modeling

This term may initially seem perverse, since we have spent most of the book trying to minimize the (cross) entropy of models, but the idea is that we do not want to go beyond the data. If we chose a model with less entropy, we would add “information” constraints to the model that are not justified by the empirical evidence available to us. Choosing the maximum entropy model is motivated by the desire to preserve as much uncertainty as possible. Somewhat simplified scheme:

1. Select features. We first compute the expectation of each feature based on the training set. Each feature then defines the constraint that this empirical expectation be the same as the expectation the feature has in our final maximum entropy model.

2. Select models (family of probability distributions). For example, log-linear models

$$p(\overrightarrow{x}, c) = \frac{1}{Z} \prod_{i=1}^{K} \alpha_i^{f_i(\overrightarrow{x}, c)}$$

where K is the number of features, α_i is the weight for feature f_i, and Z is a normalizing constant, used to ensure that a probability distribution results. The features f_i may be binary functions that can be used to characterize any property of a pair $(\overrightarrow{x}, c)$, where $\overrightarrow{x}$ is a vector representing an input element (e.g. the same 20-dimensional vector of words as in the previous subsection), and c is the class label (1 if the article is in the “earnings” category, 0 otherwise). For text categorization, we can define features as follows:

$$f_i(\overrightarrow{x}_j, c) = \begin{cases} 1 & if\, s_{ij} > 0\, and\, c = 1 \\ 0 & otherwise \end{cases}$$

3. Of all probability distributions that obey these constraints, we attempt to find the maximum entropy distribution p^*, the one with the highest entropy. One can show that there is a unique such maximum entropy distribution and there exists an algorithm, generalized iterative scaling, which is guaranteed to converge to it.

   (a) p^* obeys the following set of constrains:

   $$E_{p^*} f_i = E_{\bar{p}} f_i \tag{6.6}$$

   In other words, the expected value of f_i for p^* is the same as the expected value for the empirical distribution (in other words, the training set).

   (b) The algorithm requires that the sum of the features for each possible $(\overrightarrow{x}, c)$ be equal to a constant C:

   $$\forall\ \overrightarrow{x}, c\ \sum_i f_i\ (\overrightarrow{x}, c) = C$$

   In order to fulfill this requirement, we define C as the greatest possible feature sum:

   $$C := \max_{\overrightarrow{x}, c} \sum_{i=1}^{K} f_i(\overrightarrow{x}, c)$$

   and add a feature f_{K+1} that is defined as follows:

   $$f_{K+1}(\overrightarrow{x}, c) = C - \sum_{i=1}^{K} f_i(\overrightarrow{x}, c)$$

   Note that this feature is not binary, in contrast to the others.

   (c) $E_p f_i$ is defined as follows:

   $$E_p f_i = \sum_{\overrightarrow{x}, c} p(\overrightarrow{x}, c) f_i(\overrightarrow{x}, c)$$

   where the sum is over the event space, that is, all possible vectors $\overrightarrow{x}$ class labels c.

   (d) The empirical expectation is easy to compute:

   $$E_{\bar{p}} f_i = \sum_{\overrightarrow{x}, c} \bar{p}(\overrightarrow{x}, c) f_i(\overrightarrow{x}, c) = \frac{1}{N} \sum_{j=1}^{K} f_i(\overrightarrow{x}_j, c)$$

   where N is the number of elements in the training set and we use the fact that the empirical probability for a pair that doesn't occur in the training set is 0.

(e) In general, the maximum entropy distribution $E_p f_i$ cannot be computed efficiently since it would involve summing over all possible combinations of $\overrightarrow{x}$ and c, a potentially infinite set. Instead, we use the following approximation:

$$E_p f_i = \frac{1}{N} \sum_{j=1}^{N} \sum_{c} p(c|\overrightarrow{x}_j) f_i(\overrightarrow{x}_j, c)$$

where c ranges over all possible classes, in our case $c \in 0, 1$.

(f) Now we have all the pieces to state the generalized **iterative scaling algorithm**:

i. Initialize $\alpha_i^{(1)}$. Any initialization will do, but usually we choose $\alpha_i^{(1)} = 1, \forall 1 \leq j \leq K+1$. Compute $E_{\tilde{p}} f_i$ as shown above. Set $n = 1$.

ii. Compute $p^{(n)}(\overrightarrow{x}, c)$ for the distribution $p^{(n)}$ given by the $\{\alpha_i^{(n)}\}$ for each element $(\overrightarrow{x}, c)$ in the training set:

$$p^{(n)}(\overrightarrow{x}, c) = \frac{1}{Z} \prod_{i=1}^{K+1} \left(\alpha_i^{(n)}\right)^{f_i(x,c)}$$

iii. Compute $E_{p^{(n)}} f_i$ for all $1 \leq i \leq K+1$.

iv. Update the parameters α_i:

$$\alpha_i^{(n+1)} = \alpha_i^{(n)} \left(\frac{E_{\tilde{p}} f_i}{E_{p^{(n)}} f_i} \right)^{\frac{1}{c}}$$

v. If the parameters of the procedure have converged, stop, otherwise increment n and go to 2.

vi. Note: in an actual implementation, it is more convenient to do the computations using logarithms. One can show that this procedure converges to a distribution p^* that obeys the constraints (6.6), and that of all such distributions it is the one that maximizes the entropy $H(p)$ and the likelihood of the data.

4. We can use $P(\text{“}earnings\text{”}|\overrightarrow{x}) > P(\neg\text{“}earnings\text{”}|\overrightarrow{x})$ as our decision rule.

6.3.3 Perceptrons

As before, text documents are represented as term vectors. Our goal is to learn a weight vector $\overrightarrow{w}$ and a threshold θ, such that comparing the dot product of the weight vector and the term vector against the threshold provides the categorization decision. We decide “yes” (the article is in the “earnings” category) if the inner product of weight vector and document vector is greater than the threshold and “no” otherwise:

$$decide\,\text{"}yes\text{"}\ iff \quad \overrightarrow{w} \cdot \overrightarrow{x} = \sum_{i=1}^{K} w_i x_{ij} > 0$$

where K is the number of features ($K = 20$ for our example as before) and x_{ij} is component i of vector $\overrightarrow{x}_j$.

The basic idea of the perceptron learning algorithm is simple. If the weight vector makes a mistake, we move it (and θ) in the direction of greatest change for our optimality criterion $\sum_{i=1}^{K} w_i x_{ij} - \theta$. To see that the changes to $\overrightarrow{w}$ and θ in figure 16.8 are made in the direction of greatest change, we first define an optimality criterion φ that incorporates θ into the weight vector:

$$\varphi(\overrightarrow{w}') = \varphi \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \\ \theta \end{pmatrix} = \overrightarrow{w}' \cdot \overrightarrow{x}' = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \\ \theta \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \\ -1 \end{pmatrix}$$

The gradient of φ (which is the direction of greatest change) is the vector $\overrightarrow{x}'$:

$$\nabla \varphi(\overrightarrow{w}') = \overrightarrow{x}'$$

Of all vectors of a given length that we can add to $\overrightarrow{w}'$, $\overrightarrow{x}'$ is the one that will change the most.

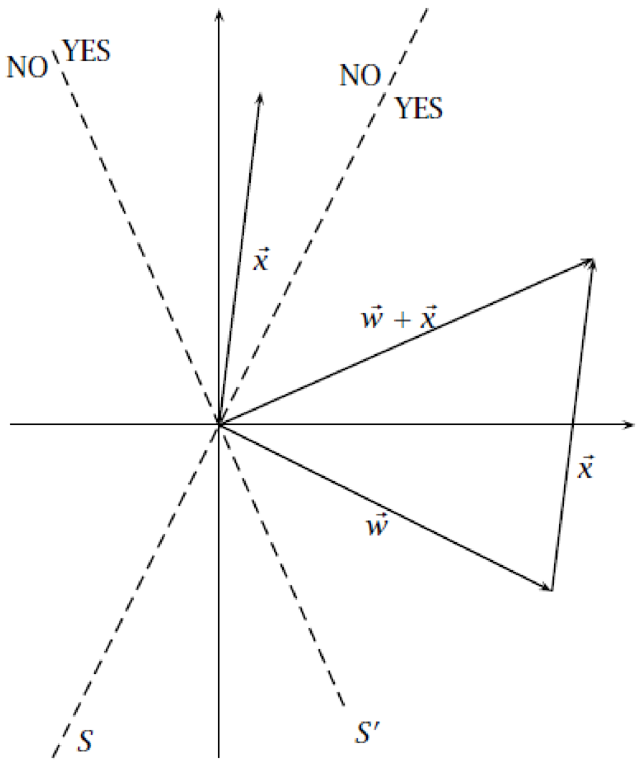


Figure 16.8 One error-correcting step of the perceptron learning algorithm. Data vector $\vec{x}$ is misclassified by the current weight vector $\vec{w}$ since it lies on the "no"-side of the decision boundary S. The correction step adds $\vec{x}$ to $\vec{w}$ and (in this case) corrects the decision since $\vec{x}$ now lies on the "yes"-side of S', the decision boundary of the new weight vector $\vec{w} + \vec{x}$.

The figure also illustrates the class of models that can be learned by perceptrons: linear separators.

Algorithm:

```
// Categorization Decision
fun decision($\overrightarrow{x}$,$\overrightarrow{w}$,$\theta$) =
  if $\overrightarrow{w} \cdot \overrightarrow{x} > 0$ then
    return “yes”
  else
    return “no”
// Initialization
$\overrightarrow{w} = 0$
$\theta = 0$
// Perceptron Learning Algorithm
while not converged yet do
  for all elements $\overrightarrow{x}_j$ in the training set do
    d = decision($\overrightarrow{x}_j$,$\overrightarrow{w}$,$\theta$)
    if class($\overrightarrow{x}_j$) = d then
      continue
    else if class($\overrightarrow{x}_j$) = ”yes” and d = ”no” then
      $\theta = \theta - 1$
      $\overrightarrow{w} = \overrightarrow{w} + \overrightarrow{x}_j$
    else if class($\overrightarrow{x}_j$) = ”no” and d = ”yes” then
      $\theta = \theta + 1$
      $\overrightarrow{w} = \overrightarrow{w} - \overrightarrow{x}_j$
    fi
  end
end
```

6.3.4 k Nearest Neighbor Classification

The cosine similarity metric is often used.

6.4 Conditional Random Field

Assuming we have a sequence of input words $X = x_1 \ldots x_n$ and want to compute a sequence of output tags $Y = y_1 \ldots y_n$. In an HMM to compute the best tag sequence that maximizes $P(Y|X)$ we rely on Bayes’ rule and the likelihood $P(X|Y)$:

$$\begin{aligned}\hat{Y} &= \arg\max_Y p(Y|X) \\ &= \arg\max_Y p(X|Y)p(Y) \\ &= \arg\max_Y \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1})\end{aligned}$$

In a CRF, by contrast, we compute the posterior $P(Y|X)$ directly, training the CRF to discriminate among the possible tag sequences:

$$\hat{Y} = \arg\max_{Y \in \mathcal{Y}} P(Y|X)$$

However, the CRF does not compute a probability for each tag at each time step. Instead, at each time step the CRF computes log-linear functions over a set of relevant features, and these local features are aggregated and normalized to produce a global probability for the whole sequence.

A CRF is a log-linear model that assigns a probability to an entire output (tag) sequence Y, out of all possible sequences $\mathcal{Y}$, given the entire input (word) sequence X. We can think of a CRF as like a giant version of what multinomial logistic regression does for a single token. Recall that the feature function f in regular multinomial logistic regression maps a tuple of a token x and a label y into a feature vector. In a CRF, the function F maps an entire input sequence X and an entire output sequence Y to a feature vector. Let's assume we have K features, with a weight w_k for each feature F_k:

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)}$$

It's common to also describe the same equation by pulling out the denominator into a function $Z(X)$:

$$p(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)$$

We'll call these K functions $F_k(X,Y)$ global features, since each one is a property of the entire input sequence X and output sequence Y. We compute them by decomposing into a sum of local features for each position i in Y:

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

Each of these local features f_k in a linear-chain CRF is allowed to make use of the current output token y_i, the previous output token y_{i-1}, the entire input string X (or any subpart of it), and the current position i. This constraint to only depend on the current and previous output tokens y_i and y_{i-1} are what characterizes a linear chain CRF. This limitation makes it possible to use versions of the efficient Viterbi and Forward-Backwards algorithms from the HMM. A general CRF, by contrast, allows a feature to make use of any output token, and are thus necessary for tasks in which the decision depend on distant output tokens, like y_{i-4}. General CRFs require more complex inference, and are less commonly used for language processing.

Examples for some features include:

$$\mathbb{1}\{x_i = \mathit{the},\ y_i = \mathit{DET}\}$$
$$\mathbb{1}\{y_i = \mathit{PROPN},\ x_{i+1} = \mathit{Street},\ y_{i-1} = \mathit{NUM}\}$$

To decode (finding the most likely tag sequence) we use he Viterbi algorithm. This involves filling an $N \times T$ array with the appropriate values, maintaining backpointers as we proceed. As with HMM Viterbi, when the table is filled, we simply follow pointers back from the maximum value in the final column to retrieve the desired set of labels. A cell value of time t for state j

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1}, y_t, X, t) \quad 1 \leq j \leq N, 1 < t \leq T$$

Learning in CRFs relies on the same supervised learning algorithms as for logistic regression. Given a sequence of observations, feature functions, and corresponding outputs, we use stochastic gradient descent to train the weights to maximize the log-likelihood of the training corpus. The local nature of linear-chain CRFs means that a CRF version of the forward-backward algorithm can be used to efficiently compute the necessary derivatives. As with logistic regression, L1 or L2 regularization is important.

6.5 Naive Bayes for words disambiguation

Bayes decision rule:

decide $s' \; if \; P(s'|c) > P(s_k|c) \; for \; s_k \neq s'$

where $s_1, \ldots, s_K$ senses of an ambiguous word w, $c_1, \ldots, c_I$ contexts of w in a corpus, $v_1, \ldots, v_J$ words used as contextual features for disambiguation.

We usually do not know the value of $P(s_k|c)$, but we can compute it using Bayes' rule:

$$P(s_k|c) = \frac{P(c|s_k)}{P(c)} P(s_k)$$

$P(s_k)$ is the prior probability of sense s_k, the probability that we have an instance of s_k if we do not know anything about the context. $P(s_k)$ is updated with the factor $\frac{P(c|s_k)}{P(c)}$ which incorporates the evidence which we have about the context, and results in the posterior probability $P(s_k|c)$.

If all we want to do is choose the correct class then

$$\begin{aligned} s' &= \arg\max_{s_k} P(s_k|c) \\ &= \arg\max_{s_k} \frac{P(c|s_k)}{P(c)} P(s_k) \\ &= \arg\max_{s_k} P(c|s_k) P(s_k) \\ &= \arg\max_{s_k} [\log P(c|s_k) + \log P(s_k)] \end{aligned}$$

The Naive Bayes assumption is that the attributes used for description are all conditionally independent:

$$P(c|s_k) = P\left(\{v_j|v_j \, in \, c\}|s_k\right) = \prod_{v_j \, in \, c} P(v_j|s_k)$$

In our case, the Naive Bayes assumption has two consequences. The first is that all the structure and linear ordering of words within the context is ignored. This is often referred to as a bag of words model. The other is that the presence of one word in the bag is independent of another. This is clearly not true.

Decision rule for Naive Bayes:

decide $s' \; if \; s' = \arg\max_{s_k} \left[\log P(s_k) + \sum_{v_j \, in \, c} \log P(v_j|s_k)\right]$

$P(v_j|s_k)$ and $P(s_k)$ are computed via Maximum-Likelihood estimation, perhaps with appropriate smoothing, from the labeled training corpus:

$$P(v_j|s_k) = \frac{C(v_j, s_k)}{C(s_k)}$$

$$P(s_k) = \frac{C(s_k)}{C(w)}$$

where $C(v_j, s_k)$ is the number of occurrences of v_j in a context of sense s_k in training corpus, $C(s_k)$ is the number of occurrences of s_k in the training corpus, and $C(w)$ is the total number of occurrences of the ambiguous word w.

6.6 Word2vec

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. The vectors are chosen carefully such that the cosine similarity between the vectors indicates the level of semantic similarity between the words represented by those vectors.

For a sentence of n words $w_1, \ldots, w_n$, contexts of a word w_i comes from a window of size k around the word: $C(w) = w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}$, where k is a parameter. [6] used a dynamic window size $k' \leq k$ with k' sampled uniformly for each word in the corpus.

The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier; a word c that occurs near the target word *apricot* acts as gold "correct answer" to the question "Is word c likely to show up near *apricot*?" This method, often called self-supervision.

The continuous skip-gram model is similar to the continuous bag-of-words model, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. In the skip-gram and ivLBL models, the objective is to predict a word's context given the word itself, whereas the objective in the CBOW and vLBL models is to predict a word given its context.

In the skip-gram model we are given a corpus of words w and their contexts c. We consider the conditional probabilities $p(c|w)$, and given a corpus T (Text), the goal is to set the parameters θ of $p(c|w;\theta)$ so as to maximize the corpus probability (with the assumption that all context words are independent):

$$\arg\max_{\theta} \prod_{w \in T} \left[\prod_{c \in C(w)} p(c|w;\theta) \right]$$

in this equation, $C(w)$ is the set of contexts of word w. Alternatively:

$$\arg\max_{\theta} \prod_{(w,c) \in D} p(c|w;\theta) \tag{6.7}$$

here D is the set of all word and context pairs we extract from the text.

So, skip-gram actually stores two embeddings for each word, one for the word as a target, and one for the word considered as context. Thus the parameters we need to learn are two matrices W and C, each containing an embedding for every one of the $|V|$ words in the vocabulary V.

One approach for parameterizing the skip-gram model follows the neural-network language models literature, and models the conditional probability $p(c|w;\theta)$ using soft-max:

$$p(c|w;\theta) = \frac{e^{v_c v_w}}{\sum_{c' \in C} e^{v_{c'} v_w}}$$

where v_c and $v_w \in \mathbb{R}^d$ are vector representations for c and w respectively, and C is the set of all available contexts. The parameters θ are v_{c_i}, v_{w_i} for $w \in V$, $c \in C$, $i \in 1, \ldots, d$ (a total of $|C| \times |V| \times d$ parameters). We would like to set the parameters such that the product (6.7) is maximized.

Taking the log switches from product to sum:

$$\arg\max_{\theta} \prod_{(w,c) \in D} \log p(c|w;\theta) = \sum_{(w,c) \in D} \left(\log e^{v_c v_w} - \log \sum_{c' \in C} e^{v_{c'} v_w} \right) \tag{6.8}$$

An assumption is that maximizing objective (6.8) will result in good embeddings $v_w \; \forall w \in V$, in the sense that similar words will have similar vectors. It is not clear to us at this point why this assumption holds. But we try to increase the quantity $v_w \cdot v_c$ for good word-context pairs (after all, cosine is just a normalized dot product), and decrease it for bad ones. Intuitively, this means that words that share many contexts will be similar to each other (note also that contexts sharing many words will also be similar to each other).

While objective (6.8) can be computed, it is computationally expensive to do so, because the term $p(c|w;\theta)$ is very expensive to compute due to the summation $\sum_{c' \in C} e^{v_{c'} v_w}$ over all the contexts $c\prime$ (there can be hundreds of thousands of them). One way of making the computation more tractable is to replace the softmax with a hierarchical softmax.

[6] presents the negative-sampling approach as a more efficient way of deriving word embeddings. While negative-sampling is based on the skip-gram model, it is in fact optimizing a different objective.

Consider a pair (w, c) of word and context. Did this pair come from the training data? Let's denote by $p(D = 1|w, c)$ the probability that (w, c) came from the corpus data. We don't actually care about this binary prediction task; instead we'll take the learned classifier weights as the word embeddings. Correspondingly, $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ will be the probability that (w, c) did not come from the corpus data. As before, assume there are parameters θ controlling the distribution: $p(D = 1|w, c; \theta)$. Our goal is now to find parameters to maximize the probabilities that all of the observations indeed came from the data:

$$\begin{aligned}
&\arg\max_{\theta} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \\
&= \arg\max_{\theta} \log \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log p(D = 1|w, c; \theta)
\end{aligned}$$

The quantity $p(D = 1|c, w; \theta)$ can be defined using softmax:

$$p(D = 1|c, w; \theta) = \frac{1}{1 + e^{-v_c v_w}}$$

Leading to the objective:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c v_w}}$$

This objective has a trivial solution if we set θ such that $p(D = 1|w, c; \theta) = 1$ for every pair (w, c). This can be easily achieved by setting θ such that $v_c = v_w$ and $v_c \cdot v_w = K$ for all v_c, v_w, where K is large enough number (practically, we get a probability of 1 as soon as $K \approx 40$).

We need a mechanism that prevents all the vectors from having the same value, by disallowing some (w, c) combinations. One way to do so, is to present the model with some (w, c) pairs for which $p(D = 1|w, c; \theta)$ must be low, i.e. pairs which are not in the data. This is achieved by generating the set D' of random (w, c) pairs, assuming they are all incorrect (the name "negative-sampling" stems from the set D' of randomly sampled negative examples). The optimization objective now becomes:

$$
\begin{aligned}
&\arg\max_\theta \prod_{(w,c)\in D} p(D=1|w,c;\theta) \prod_{(w,c)\in D'} p(D=0|w,c;\theta) \\
&= \arg\max_\theta \prod_{(w,c)\in D} p(D=1|w,c;\theta) \prod_{(w,c)\in D'} (1-p(D=1|w,c;\theta)) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log p(D=1|w,c;\theta) + \sum_{(w,c)\in D'} \log (1-p(D=1|w,c;\theta)) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c v_w}} + \sum_{(w,c)\in D'} \log \left(1-\frac{1}{1+e^{-v_c v_w}}\right) \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c v_w}} + \sum_{(w,c)\in D'} \log \frac{1}{1+e^{v_c v_w}}
\end{aligned}
$$

If we let $\sigma(x) = \frac{1}{1+e^{-x}}$ (to turn the dot product into a probability) we get:

$$
\begin{aligned}
&\arg\max_\theta \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c v_w}} + \sum_{(w,c)\in D'} \log \frac{1}{1+e^{v_c v_w}} \\
&= \arg\max_\theta \sum_{(w,c)\in D} \log \sigma(v_c v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c v_w)
\end{aligned}
\tag{6.9}
$$

With negative sampling of k, [6] constructed D' is k times larger than D, and for each $(w,c) \in D$ we construct k samples $(w,c_1),\dots,(w,c_k)$, where each c_j is drawn according to its unigram distribution (frequency in T) raised to the 3/4 power. In [6] each context is a word (and all words appear as contexts), and so $p_{context}(x) = p_{words}(x) = \frac{count(x)^\alpha}{|T|^\alpha}$. Setting $\alpha = 0.75$ gives rare noise words slightly higher probability (it dampens the probability for high occurrence words, e.g. *the*, and boosts one for rare words, e.g. *aardvark*).

So, after (6.9) our loss function is

$$
L = -\left[\sum_{(w,c)\in D} \log \sigma(v_c v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c v_w)\right]
$$

We minimize this loss function using stochastic gradient descent. Taking the derivative with respect to the different embeddings we obtain

$$
\begin{aligned}
\frac{\partial L}{\partial v_{c+}} &= -\sum_{(w,c)\in D} \frac{v_w}{1+e^{v_{c+} v_w}} \\
\frac{\partial L}{\partial v_{c-}} &= \sum_{(w,c)\in D'} \frac{v_w}{1+e^{-v_{c-} v_w}} \\
\frac{\partial L}{\partial v_w} &= -\sum_{(w,c+)\in D} \frac{v_{c+}}{1+e^{v_{c+} v_w}} + \sum_{(w,c-)\in D'} \frac{v_{c-}}{1+e^{-v_{c-} v_w}}
\end{aligned}
$$

The update equations going from time step t to $t+1$ in stochastic gradient descent are thus:

$$v_{c+}(t+1) = v_{c+}(t) - \eta \frac{\partial L}{\partial v_{c+}}(t)$$
$$v_{c-}(t+1) = v_{c-}(t) - \eta \frac{\partial L}{\partial v_{c-}}(t)$$
$$v_w(t+1) = v_w(t) - \eta \frac{\partial L}{\partial v_w}$$

Word2vec embeddings are static embeddings, meaning that the method learns one fixed embedding for each word in the vocabulary. This is in contrast with dynamic contextual embeddings like the popular BERT or ELMO representations, in which the vector for each word is different in different contexts.

6.7 GloVe

GloVe, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. As log-bilinear regression model for unsupervised learning of word representations, it combines the features of two model families, namely the global matrix factorization and local context window methods.

The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations. Let's consider the ratio of co-occurrence probabilities of 2 words i and j with various probe words k: P_{ik}/P_{jk}. For words k that are either related to both or to neither the ratio should be close to one. This argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. Noting that the ratio P_{ik}/P_{jk} depends on three words i, j, and k, the most general model takes the form,

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{6.10}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate context word vectors. In this equation, the right-hand side is extracted from the corpus, and F may depend on some as-of-yet unspecified parameters. The number of possibilities for F is vast, but by enforcing a few desiderata we can select a unique choice. First, we would like F to encode the information which presents the ratio P_{ik}/P_{jk} in the word vector space. Since vector spaces are inherently linear structures, the most natural way to do this is with vector differences.

With this aim, we can restrict our consideration to those functions F that depend only on the difference of the two target words, modifying Eqn. (6.10) to,

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{6.11}$$

Next, we note that the arguments of F in Eqn. (6.11) are vectors while the right-hand side is a scalar. While F could be taken to be a complicated function parameterized by, e.g., a neural network, doing so would obfuscate the linear structure we are trying to capture. To avoid this issue, we can first take the dot product of the arguments,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{6.12}$$

which prevents F from mixing the vector dimensions in undesirable ways. Next, note that for word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary and that we are free to exchange the two roles. To do so consistently, we must not only exchange $w \leftrightarrow \tilde{w}$ but also $X \leftrightarrow X^T$, where X is the matrix of word-word co-occurrence counts, whose entries X_{ij} tabulate the number of times word j occurs in the context of word i. Our final model should be invariant under this relabeling, but Eqn. (6.12) is not. However, the symmetry can be restored in two steps. First, we require that F be a homomorphism between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$, i.e.,

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} \tag{6.13}$$

which, by Eqn. (6.12), is solved by,

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i} \tag{6.14}$$

where $X_i = \sum_k X_{ik}$ is the number of times any word appears in the context of word i.

The solution to Eqn. (6.13) is $F = \exp$, or,

$$w_i^T \tilde{w}_k = \ln P_{ik} = \ln X_{ik} - \ln X_i \tag{6.15}$$

Next, we note that Eqn. (6.15) would exhibit the exchange symmetry if not for the $\ln X_i$ on the right-hand side. However, this term is independent of k so it can be absorbed into a bias b_i for w_i. Finally, adding an additional bias $\tilde{b}_k$ for $\tilde{w}_k$ restores the symmetry,

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \ln X_{ik} \tag{6.16}$$

Eqn. (6.16) is a drastic simplification over Eqn. (6.10), but it is actually ill-defined since the logarithm diverges whenever its argument is zero. One resolution to this issue is to include an additive shift in the logarithm, $\ln X_{ik} \rightarrow$

$\ln(X_{ik}+1)$, which maintains the sparsity of X while avoiding the divergences. The idea of factorizing the log of the co-occurrence matrix is closely related to LSA. A main drawback to this model is that it weighs all co-occurrences equally, even those that happen rarely or never. Such rare co-occurrences are noisy and carry less information than the more frequent ones - yet even just the zero entries account for 75-95% of the data in X, depending on the vocabulary size and corpus.

The following weighted least squares regression model addresses these problems. Casting Eqn. (6.16) as a least squares problem and introducing a weighting function $f(X_{ij})$ into the cost function gives us the model

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \ln X_{ij})^2 \tag{6.17}$$

where V is the size of the vocabulary. The weighting function should obey the following properties:

1. $f(0) = 0$. If f is viewed as a continuous function, it should vanish as $x \to 0$ fast enough that the $\lim_{x\to 0} f(x)\ln^2 x$ is finite.

2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.

3. $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighted.

Of course a large number of functions satisfy these properties, but one class of functions that we found to work well can be parameterized as,

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & if\, x < x_{max} \\ 1 & otherwise \end{cases} \tag{6.18}$$

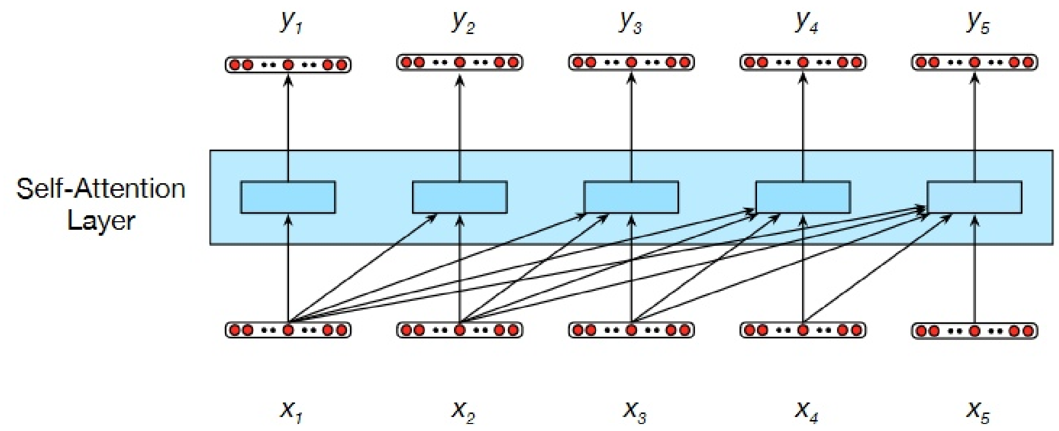
The performance of the model depends weakly on the cutoff, which we fix to $x_{max} = 100$ for all our experiments. We found that $\alpha = 3/4$ gives a modest improvement over a linear version with $\alpha = 1$.

6.8 Bidirectional Encoder Representations from Transformers (BERT)

Despite the ability of LSTMs to mitigate the loss of distant information due to the recurrence in RNNs, the underlying problem remains. Passing information forward through an extended series of recurrent connections leads to a loss of relevant information and to difficulties in training. Moreover, the inherently sequential nature of recurrent networks inhibits the use of parallel computational resources. These considerations led to the development of Transformers - an approach to sequence processing that eliminates recurrent connections and returns to architectures reminiscent of the fully connected networks.

Transformers map sequences of input vectors $(x_1, \ldots, x_n)$ to sequences of output vectors $(y_1, \ldots, y_n)$ of the same length. Transformers are made up of stacks of network layers consisting of simple linear layers, feedforward networks, and custom connections around them. In addition to these standard components, the key innovation of transformers is the use of self-attention layers. Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs.

A causal (or masked) self-attention model:



At the core of an attention-based approach is the ability to compare an item of interest to a collection of other items in way that reveals their relevance in the current context. It's essentially an auto-regression. Usually the dot product is used, but to allow for other possible comparisons, let's use a score function

$$score(x_i, x_j) = x_i x_j$$

then we normalize (to provide a probability distribution) to create weights

$$\begin{aligned}\alpha_{ij} &= softmax\left(score(x_i, x_j)\right)\ \forall j \leq i \\ &= \frac{exp\left(score(x_i, x_j)\right)}{\sum_{k=1}^{i} exp\left(score(x_i, x_k)\right)}\ \forall j \leq i\end{aligned} \tag{6.19}$$

and

$$y_i = \sum_{j \leq i} \alpha_{ij} x_j$$

Unfortunately, this simple mechanism provides no opportunity for learning, everything is directly based on the original input values x. To allow for this, Transformers include additional parameters in the form of a set of weight matrices that operate over the input embeddings, called *query*, *key* and *value:*

$$q_i = W^Q x_i;\ k_i = W^K x_i;\ v_i = W^V x_i$$

Given input embeddings of size d_m, the dimensionality of these matrices are $d_q \times d_m$, $d_k \times d_m\, and\, d_v \times d_m$, respectively. In the original Transformer work ([11]), d_m was 1024 and 64 for d_k, d_q and d_v. Given these projections,

$$score(x_i, x_j) = q_i k_j$$

and

$$y_i = \sum_{j \leq i} \alpha_{ij} v_j$$

To avoid overflow in exponentiation in (6.19) the dot product needs to be scaled, typically

$$score(x_i, x_j) = \frac{q_i k_j}{\sqrt{d_k}}$$

Since each output, y_i, is computed independently this entire process can be parallelized by taking advantage of efficient matrix multiplication routines by packing the input embeddings into a single matrix and multiplying it by the key, query and value matrices to produce matrices containing all the key, query and value vectors.

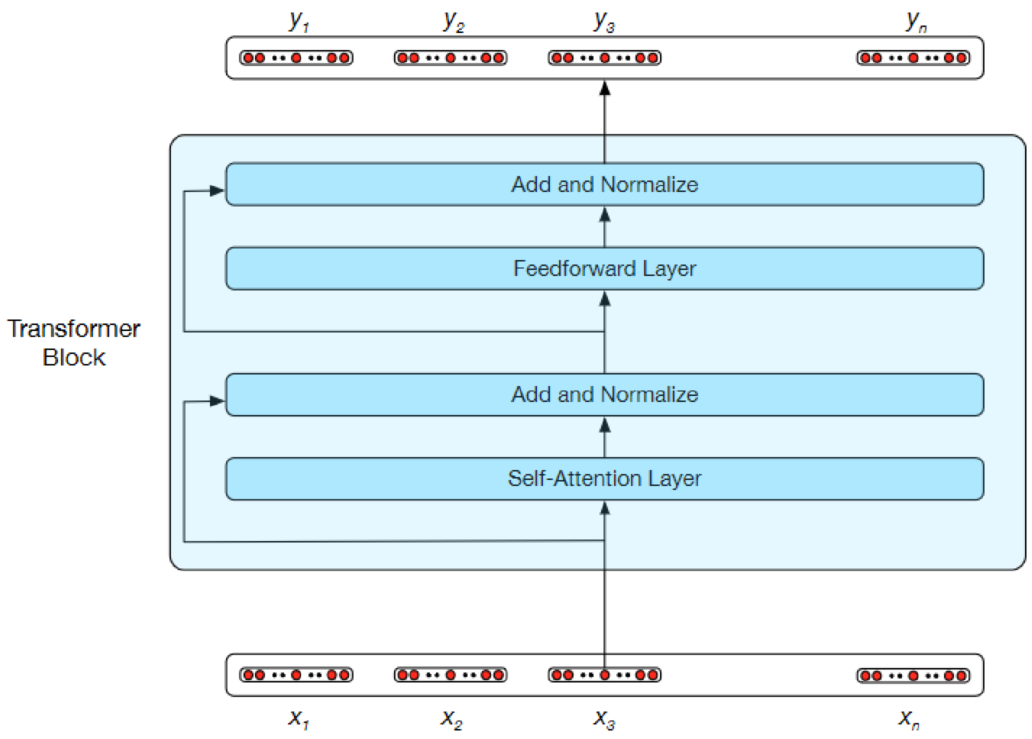
$$Q = W^Q X;\; K = W^K X;\; V = W^V X$$

and

$$SelfAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

To exclude values following the query, set the upper triangle to $-\infty$.

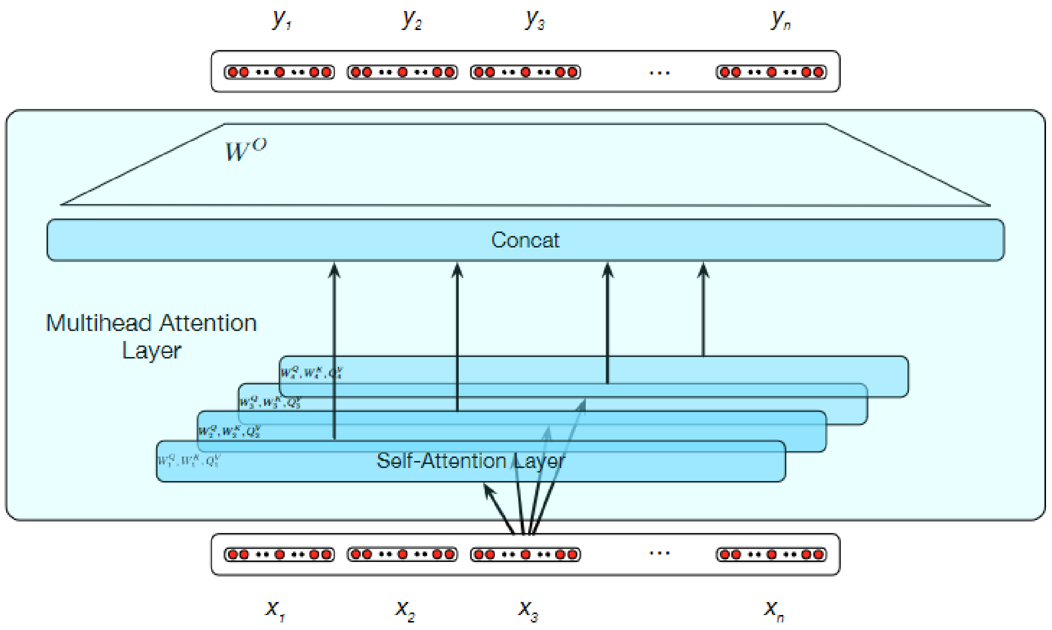
A typical transformer block:



Multihead self-attention layers are sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters.

$$\begin{aligned} head_i &= SelfAttention(W_i^Q X, W_i^K X, W_i^V X) \\ MultiHeadAttn(Q, K, V) &= W^O(head_1 \oplus head_2 \ldots \oplus head_h) \end{aligned}$$

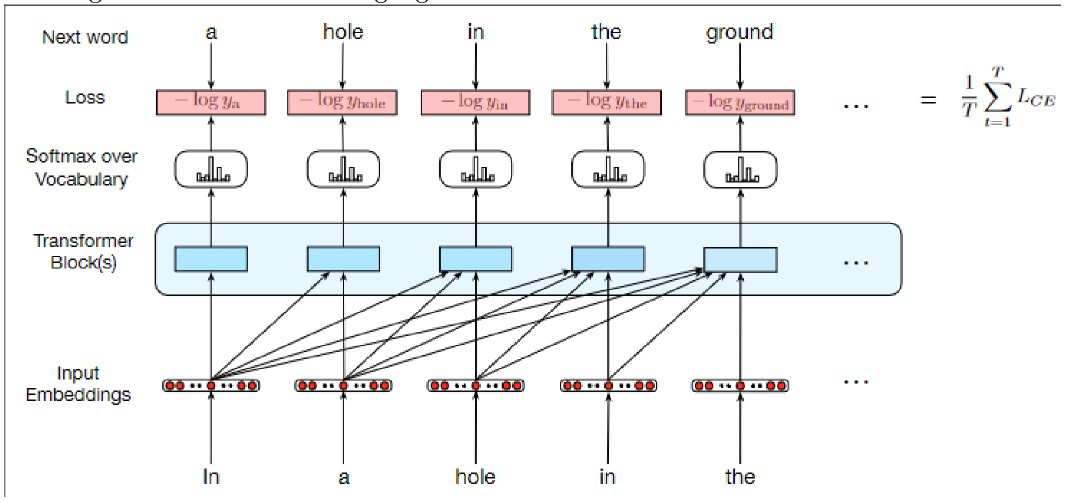
where $\oplus$ is concatenation.



The rest of the Transformer block with its feedforward layer, residual connections, and layer norms remains the same.

Unlike in RNN the positional information has to be encoded explicitly here.

Training a Transformer as a language model:



BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. There are two existing strategies for applying pre-trained language representations to down-stream tasks: feature-based and fine-tuning. The feature-based approach, such as ELMo, uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning all pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

A "masked language model" randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context.

Gaussian Error Linear Unit (GELU) is used which is

$$\frac{1}{2}x\left(1+erf\left(\frac{x}{\sqrt{2}}\right)\right)=x\Phi(x)$$

6.9 Transformer Embedding Dialogue (TED)

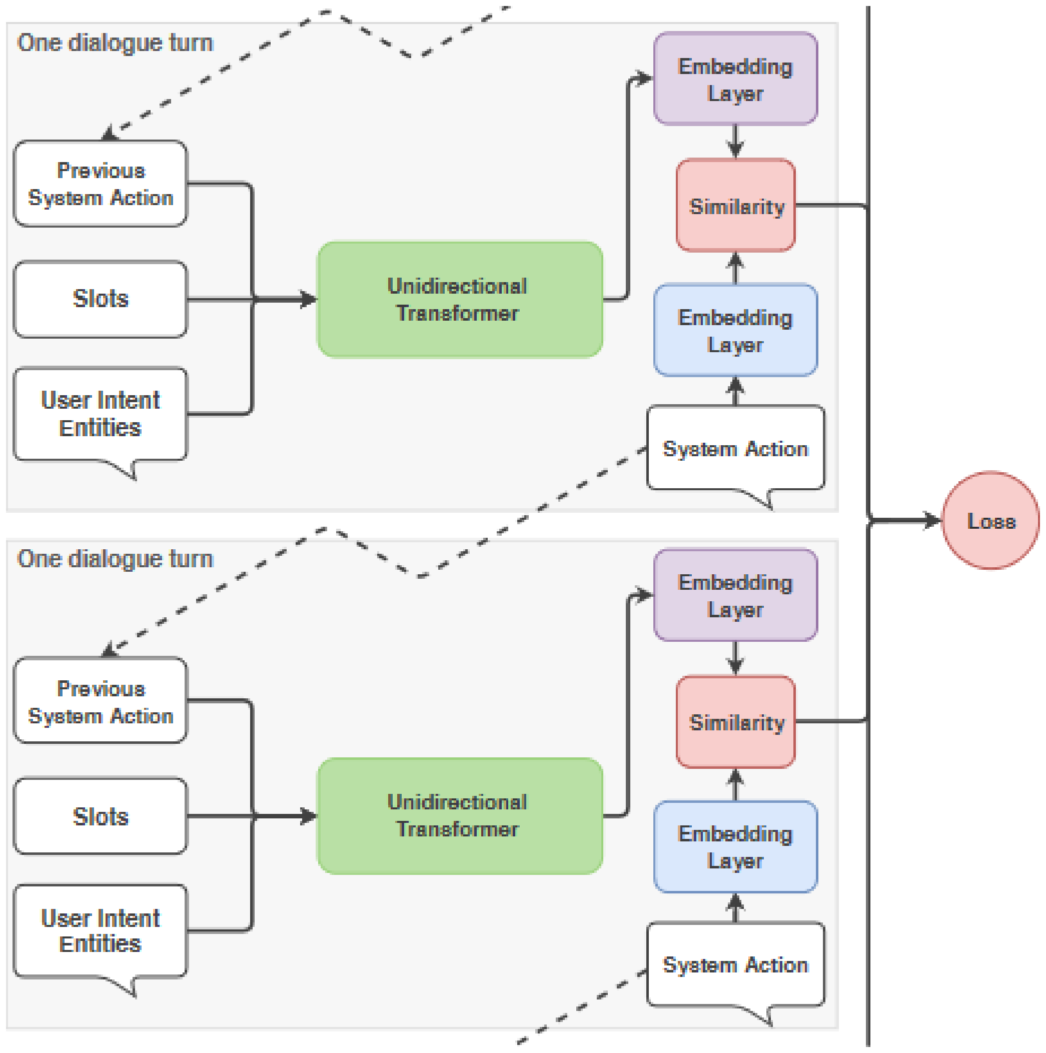


FIG. 1. A schematic representation of two time steps of the transformer embedding dialogue policy.

The transformer output a_{dialog} and system actions y_{action} are embedded into a single semantic vector space $h_{dialog} = E(a_{dialog})$, $h_{action} = E(y_{action})$, where $h \in \mathbb{R}^{20}$. The dot product loss is used to maximize the similarity $S^{+} = h^{T}_{dialog} h^{T}_{action}$ with the target label y^{+}_{action} and minimize similarities $S^{-} = h^{T}_{dialog} h^{-}_{action}$ with negative samples y^{-}_{action}. Thus, the loss function for one dialogue reads

$$L_{dialog} = -\frac{1}{N}\sum^{N}\left[S^{+} - \log\left(e^{S^{+}} + \sum_{\Omega^{-}} e^{S^{-}}\right)\right]$$

where the second sum is taken over the set of negative samples Ω^{-} and the average is taken over time steps inside one dialogue.

The global loss is an average of all loss functions from all dialogues.

At inference time, the dot-product similarity serves as a ranker for the next utterance retrieval problem.

During modular training, we use a balanced batching strategy to mitigate class imbalance, as some system actions are far more frequent than others.

6.10 Dual Intent and Entity Transformer (DIET)

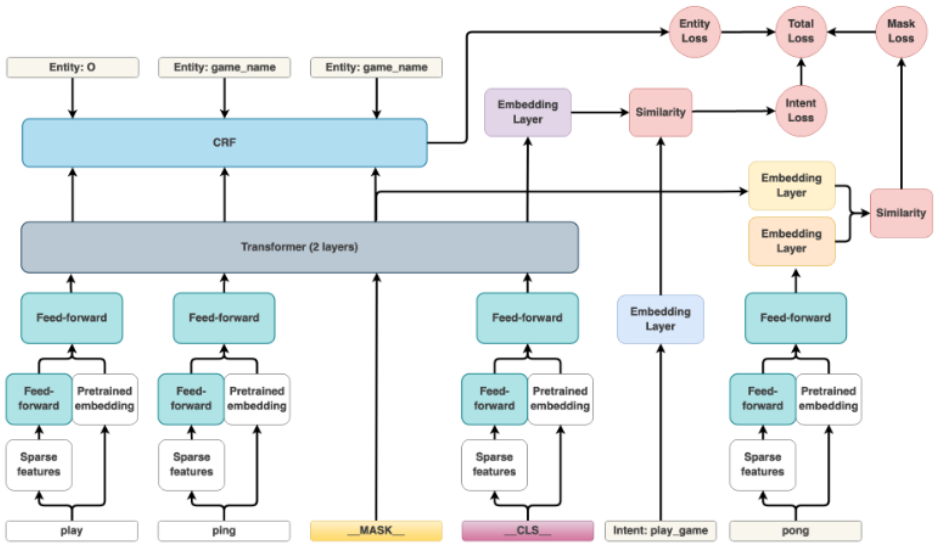


Figure 1: A schematic representation of the DIET architecture. The phrase "play ping pong" has the intent `play_game` and entity `game_name` with value "ping pong". Weights of the feed-forward layers are shared across tokens.

Input sentences are treated as a sequence of tokens, which can be either words or sub-words depending on the featurization pipeline. A special classification token__CLS__ is added to the end of each sentence. Each input token is featurized with what we call sparse features and/or dense features. Sparse features are token level one-hot encodings and multi-hot encodings of character n-grams ($n \leq 5$). Character n-grams contain a lot of redundant information, so to avoid overfitting we apply dropout to these sparse features. Dense features can be any pre-trained word embeddings: ConveRT, BERT or GloVe. Sparse features are passed through a fully connected layer with shared weights across all sequence steps to match the dimension of the dense features. The output of the fully connected layer is concatenated with the dense features from pre-trained models.

For intent classification the transformer output for__CLS__ token a_{CLS} and intent labels y_{intent} are embedded into a single semantic vector space $h_{CLS} = E(a_{CLS})$, $h_{intent} = E(y_{intent})$, where $h \in \mathbb{R}^{20}$. The dot product loss is used to maximize the similarity $S^+ = h_{CLS}^T h_{intent}^+$ with the target label y_{intent}^+ and minimize similarities $S^- = h_{CLS}^T h_{intent}^-$ with negative samples y_{intent}^-. Thus, the loss function

$$L = -\frac{1}{N}\sum^{N}\left[S^+ - \log\left(e^{S^+} + \sum_{\Omega^-} e^{S^-}\right)\right]$$

where the second sum is taken over the set of negative samples Ω^- and the average is taken over all N examples.

At inference time, the dot-product similarity serves as a ranker over all possible intent labels.

Masking is added as an additional training objective to predict randomly masked input tokens.

The total loss is

$$L_{total} = L_{intent} + L_{entity} + L_{mask}$$

6.11 Intent / Entity identification

	intent	entity
Bag-of-words model with tf.idf	✓	
Naive Bayes	✓	
CRF		✓
GloVe	✓	✓
BERT	✓	✓
TED	✓	✓
DIET	✓	✓

7 ...

References

[1] S. Bird, E. Klein, E. Loper "Natural Language Processing with Python", 2009

[2] C. Manning, H. Schütze "Foundations of Statistical Natural Language Processing", 1999

[3] T.Bunk, D.Varshneya, V.Vlasov, A.Nichol “DIET: Lightweight Language Understanding for Dialogue Systems”, 2020

[4] D.Jurafsky, J.Martin “Speech and Language Processing”, 2020

[5] T.Mikolov, K.Chen, G.Corrado, J.Dean “Efficient Estimation of Word Representations in Vector Space”, 2013

[6] T.Mikolov, I.Sutskever, K.Chen, G.Corrado, J.Dean “Distributed Representations of Words and Phrases and their Compositionality”, 2013

[7] Y.Goldberg, O.Levy “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method”, 2014

[8] D.Blei, A.Ng, M.Jordan “Latent Dirichlet allocation”, 2003

[9] J.Pennington, R.Socher, C.Manning “Glove: Global vectors for word representation”, 2014

[10] J.Devlin, M.Chang, K.Lee, K.Toutanova “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2018

[11] A.Vaswani, N.Shazeer, N.Parmar, J.Uszkoreit, L.Jones, A.Gomez, L.Kaiser, I.Polosukhin “Attention Is All You Need”, 2017

[12] L.Wu, A.Fisch, S.Chopra, K.Adams, A.Bordes, J.Weston “StarSpace: Embed All The Things!”, 2017

[13] “The Annotated Transformer”, 2018

[14] V.Vlasov, J.Mosig, A.Nichol “Dialogue Transformers”, 2020