Natural Language Processing with Deep Learning

Lecture 5 — Feed-forward network and language modeling

Prof. Dr. Ivan Habernal

November 14, 2024

`www.trusthlt.org`
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security

From binary to multi-class task

Our binary text classification function

Linear function through sigmoid — log-linear model

$$\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))} \qquad \hat{y} \in (0, 1), y \in \{0, 1\}$$

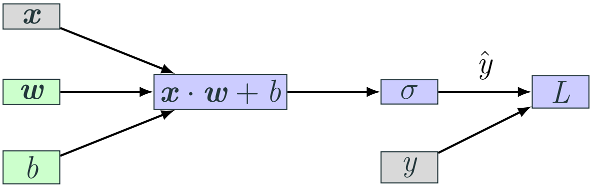


Figure 1: Computational graph; green nodes are trainable parameters, gray are constant inputs

From binary to multi-class labels

So far we mapped our gold label $y \in \{0, 1\}$

- Categorical: There is no 'ordering'
- Example: Classify the language of a document into 6 languages (En, Fr, De, It, Es, Other)

From binary to multi-class labels

So far we mapped our gold label $y \in \{0, 1\}$

- Categorical: There is no 'ordering'
- Example: Classify the language of a document into 6 languages (En, Fr, De, It, Es, Other)

One-hot encoding of labels

$$\text{En} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \text{Fr} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{De} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad \ldots$$

$\boldsymbol{y} \in \mathbb{R}^{d_{out}}$ where d_{out} is the number of classes

Possible solution: Six weight vectors and biases

Consider for each language $\ell \in \{\text{En}, \text{Fr}, \text{De}, \text{It}, \text{Es}, \text{Other}\}$

- Weight vector $\boldsymbol{w}^{\ell}$ (e.g., $\boldsymbol{w}^{\text{Fr}}$)
- Bias b^{ℓ} (e.g., b^{Fr})

Possible solution: Six weight vectors and biases

Consider for each language $\ell \in \{\text{En}, \text{Fr}, \text{De}, \text{It}, \text{Es}, \text{Other}\}$

- Weight vector $\boldsymbol{w}^{\ell}$ (e.g., $\boldsymbol{w}^{\text{Fr}}$)
- Bias b^{ℓ} (e.g., b^{Fr})

We can predict the language resulting in the highest score

$$\hat{y} = f(\boldsymbol{x}) = \operatorname*{argmax}_{\ell \in \{\text{En},\text{Fr},\text{De},\text{It},\text{Es},\text{Other}\}} \boldsymbol{x} \cdot \boldsymbol{w}^{\ell} + b^{\ell}$$

Possible solution: Six weight vectors and biases

Consider for each language $\ell \in \{\text{En}, \text{Fr}, \text{De}, \text{It}, \text{Es}, \text{Other}\}$

- Weight vector $\boldsymbol{w}^{\ell}$ (e.g., $\boldsymbol{w}^{\text{Fr}}$)
- Bias b^{ℓ} (e.g., b^{Fr})

We can predict the language resulting in the highest score

$$\hat{y} = f(\boldsymbol{x}) = \underset{\ell \in \{\text{En},\text{Fr},\text{De},\text{It},\text{Es},\text{Other}\}}{\operatorname{argmax}} \quad \boldsymbol{x} \cdot \boldsymbol{w}^{\ell} + b^{\ell}$$

But we can re-arrange the $\boldsymbol{w} \in \mathbb{R}^{d_{in}}$ vectors into columns of a matrix $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times 6}$ and $\boldsymbol{b} \in \mathbb{R}^{6}$, to get

$$f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

Projecting input vector to output vector $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$

Projecting input vector to output vector $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$

Recall from lecture 3: High-dimensional linear functions

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\quad \boldsymbol{b} \in \mathbb{R}^{d_{out}}$

Prediction of multi-class classifier

Project the input $\boldsymbol{x}$ to an output $\boldsymbol{y}$

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

and pick the element of $\hat{\boldsymbol{y}}$ with the highest value

$$\text{prediction} = \hat{y} = \operatorname*{argmax}_i \hat{\boldsymbol{y}}_{[i]}$$

Sanity check

What is $\hat{y}$?

Prediction of multi-class classifier

Project the input $\boldsymbol{x}$ to an output $\boldsymbol{y}$

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

and pick the element of $\hat{\boldsymbol{y}}$ with the highest value

$$\text{prediction} = \hat{y} = \operatorname*{argmax}_{i} \hat{\boldsymbol{y}}_{[i]}$$

Sanity check

What is $\hat{y}$?

Index of 1 in the one-hot. For example, if $\hat{y} = 3$, then the document is in German $\text{De} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$

From binary to multi-class task

Representations

Two representations of the input document

$$\hat{\boldsymbol{y}} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

Vector $\boldsymbol{x}$ is a document representation

- Bag of words, for example ($d_{in} = |V|$ dimensions, sparse)

Vector $\hat{\boldsymbol{y}}$ is **also** a document representation

- More compact (only 6 dimensions)
- More specialized for the language prediction task

Learned representations — central to deep learning

Representations are central to deep learning

One could argue that the main power of deep-learning is the ability to learn good representations

From binary to multi-class task

From multi-dimensional linear transformation to probabilities

Turning output vector into probabilities of classes

Recap: Categorical probability distribution

Categorical random variable X is defined over K categories, typically mapped to natural numbers $1, 2, \ldots, K$, for example En = 1, De = 2, $\ldots$

Turning output vector into probabilities of classes

Recap: Categorical probability distribution

Categorical random variable X is defined over K categories, typically mapped to natural numbers $1, 2, \ldots, K$, for example En = 1, De = 2, . . .

Each category parametrized with probability $\Pr(X = k) = p_k$

Turning output vector into probabilities of classes

Recap: Categorical probability distribution

Categorical random variable X is defined over K categories, typically mapped to natural numbers $1, 2, \ldots, K$, for example En = 1, De = 2, ...

Each category parametrized with probability $\Pr(X = k) = p_k$

Must be valid probability distribution: $\sum_{i=1}^{K} \Pr(X = i) = 1$

Turning output vector into probabilities of classes

Recap: Categorical probability distribution

Categorical random variable X is defined over K categories, typically mapped to natural numbers $1, 2, \ldots, K$, for example En = 1, De = 2, ...

Each category parametrized with probability $\Pr(X = k) = p_k$

Must be valid probability distribution: $\sum_{i=1}^{K} \Pr(X = i) = 1$

How to turn an **unbounded** vector in $\mathbb{R}^K$ into a categorical probability distribution?

The softmax function $\mathrm{softmax}(\boldsymbol{x}) : \mathbb{R}^K \to \mathbb{R}^K$

Softmax

Applied element-wise, for each element $\boldsymbol{x}_{[i]}$ we have

$$\mathrm{softmax}(\boldsymbol{x}_{[i]}) = \frac{\exp\left(\boldsymbol{x}_{[i]}\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{x}_{[k]}\right)}$$

The softmax function $\mathrm{softmax}(\boldsymbol{x}) : \mathbb{R}^K \to \mathbb{R}^K$

Softmax

Applied element-wise, for each element $\boldsymbol{x}_{[i]}$ we have

$$\mathrm{softmax}(\boldsymbol{x}_{[i]}) = \frac{\exp\left(\boldsymbol{x}_{[i]}\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{x}_{[k]}\right)}$$

- Nominator: Non-linear bijection from $\mathbb{R}$ to $(0; \infty)$
- Denominator: Normalizing constant to ensure $\sum_{j=1}^{K} \mathrm{softmax}(\boldsymbol{x}_{[j]}) = 1$

The softmax function $\mathrm{softmax}(\boldsymbol{x}) : \mathbb{R}^K \to \mathbb{R}^K$

Softmax

Applied element-wise, for each element $\boldsymbol{x}_{[i]}$ we have

$$\mathrm{softmax}(\boldsymbol{x}_{[i]}) = \frac{\exp\left(\boldsymbol{x}_{[i]}\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{x}_{[k]}\right)}$$

- Nominator: Non-linear bijection from $\mathbb{R}$ to $(0; \infty)$
- Denominator: Normalizing constant to ensure $\sum_{j=1}^{K} \mathrm{softmax}(\boldsymbol{x}_{[j]}) = 1$

We also need to know how to compute the partial derivative of $\mathrm{softmax}(\boldsymbol{x}_{[i]})$ wrt. each argument $\boldsymbol{x}_{[k]}$: $\frac{\partial \, \mathrm{softmax}(\boldsymbol{x}_{[i]})}{\partial \boldsymbol{x}_{[k]}}$

Softmax can be smoothed with a 'temperature' T

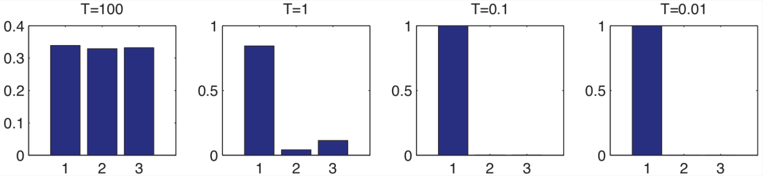
$$\text{softmax}(\boldsymbol{x}_{[i]}; T) = \frac{\exp\left(\frac{\boldsymbol{x}_{[i]}}{T}\right)}{\sum_{k=1}^{K} \exp\left(\frac{\boldsymbol{x}_{[k]}}{T}\right)}$$

Softmax can be smoothed with a ‘temperature’ T

$$\text{softmax}(\boldsymbol{x}_{[i]}; T) = \frac{\exp\left(\frac{\boldsymbol{x}_{[i]}}{T}\right)}{\sum_{k=1}^{K} \exp\left(\frac{\boldsymbol{x}_{[k]}}{T}\right)}$$

Figure from K. Murphy (2012). **Machine Learning: a Probabilistic Perspective.** MIT Press

Example: Softmax of $x = (3, 0, 1)$ at different T



High temperature $\rightarrow$ uniform distribution

Low temperature $\rightarrow$ ‘spiky’ distribution, all mass on the largest element

Loss function for softmax

Categorical cross-entropy loss (aka. negative log likelihood)

Vector representing the gold-standard categorical distribution over the classes/labels $1, \ldots, K$:

$$\boldsymbol{y} = (\boldsymbol{y}_{[1]}, \boldsymbol{y}_{[2]}, \ldots, \boldsymbol{y}_{[K]})$$

Output from softmax:

$$\hat{\boldsymbol{y}} = (\hat{\boldsymbol{y}}_{[1]}, \hat{\boldsymbol{y}}_{[2]}, \ldots, \hat{\boldsymbol{y}}_{[K]})$$

which is in fact $\hat{\boldsymbol{y}}_{[\boldsymbol{i}]} = \Pr(y = i | \boldsymbol{x})$

Cross entropy loss

$$L_{\text{cross-entropy}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = - \sum_{k=1}^{K} \boldsymbol{y}_{[k]} \log \left(\hat{\boldsymbol{y}}_{[k]} \right)$$

Stacking transformations and non-linearity

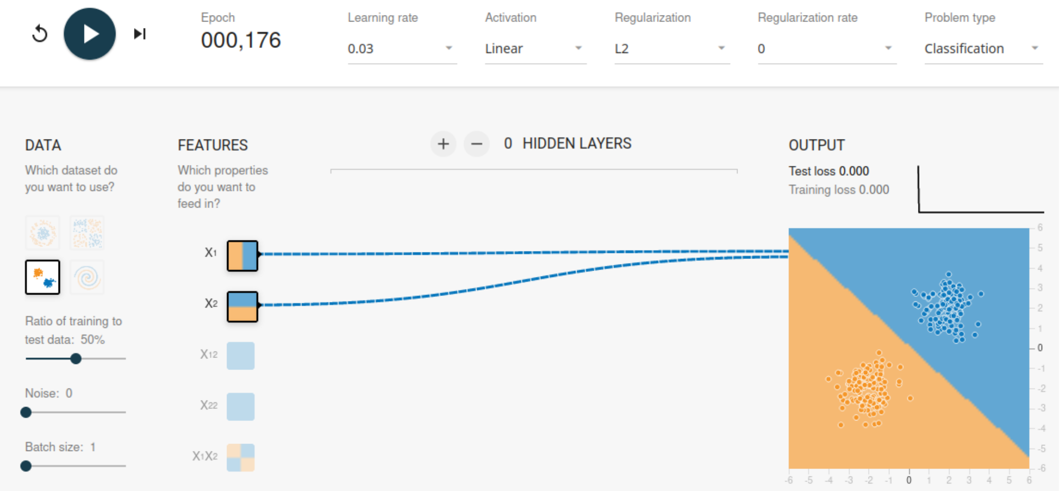


Figure 2: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

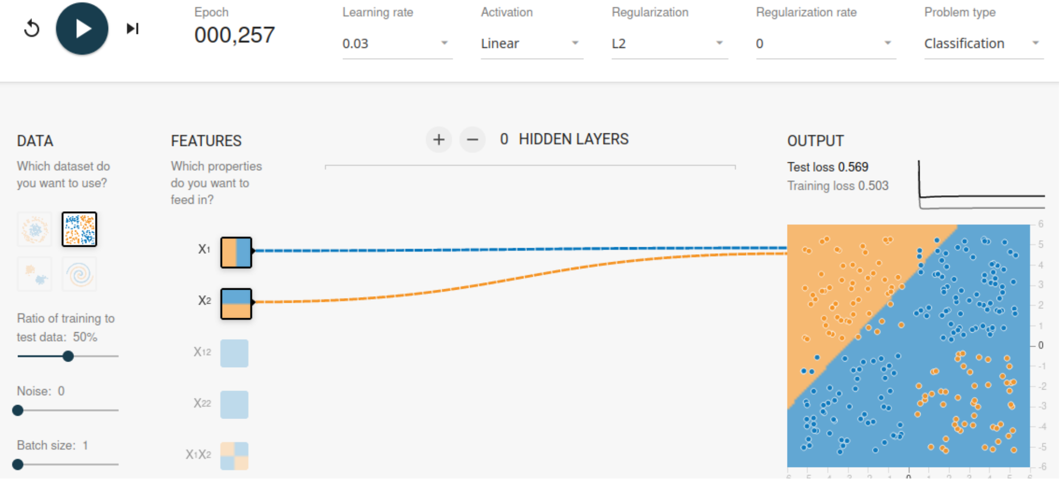


Figure 3: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

Stacking linear layers on top of each other — still linear!

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad$ $\boldsymbol{W}^1 \in \mathbb{R}^{d_{in} \times d_1}$ $\quad$ $\boldsymbol{b}^1 \in \mathbb{R}^{d_1}$ $\quad$ $\boldsymbol{W}^2 \in \mathbb{R}^{d_1 \times d_{out}}$ $\quad$ $\boldsymbol{b}^2 \in \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) = \left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right) \boldsymbol{W}^2 + \boldsymbol{b}^2$$

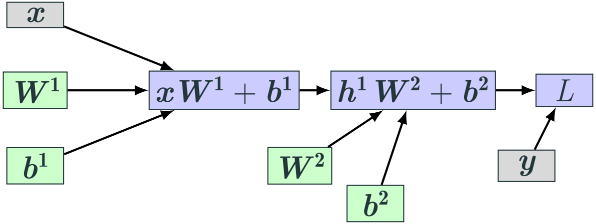


Figure 4: Computational graph; green circles are trainable parameters, gray are constant inputs

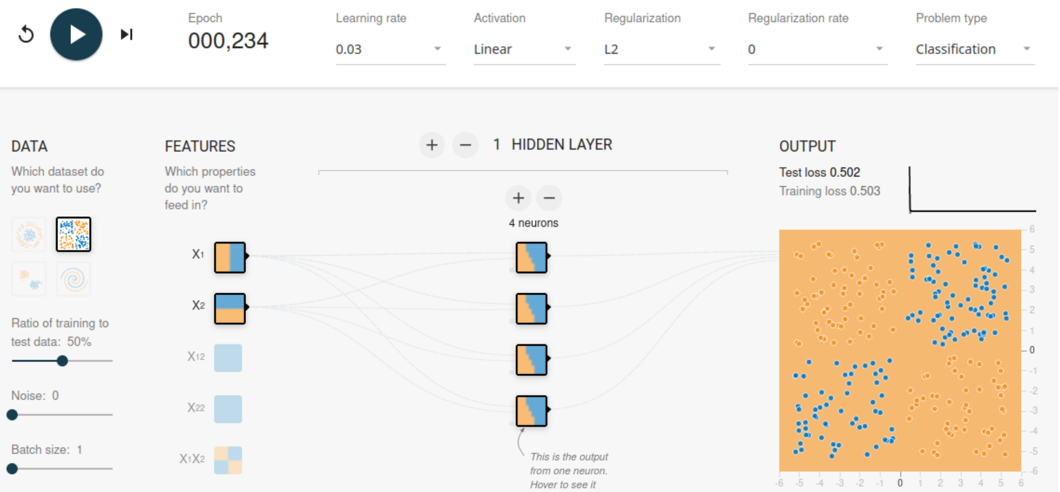


Figure 5: Linear hidden layers do not help (`http://playground.tensorflow.org`)

Adding non-linear function $g : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_1}$

$$f(\boldsymbol{x}) = g\left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right) \boldsymbol{W}^2 + \boldsymbol{b}^2$$

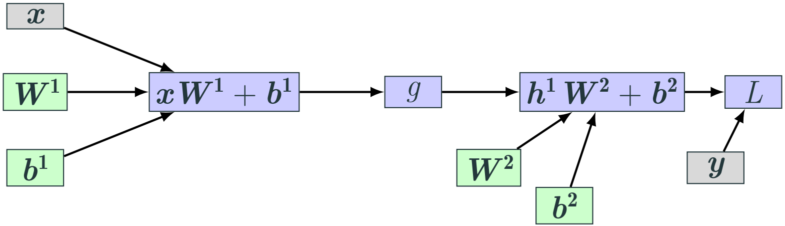


Figure 6: Computational graph; green circles are trainable parameters, gray are constant inputs

Non-linear function g: Rectified linear unit (ReLU) activation

$$\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

or $\mathrm{ReLU}(z) = \max(0, z)$

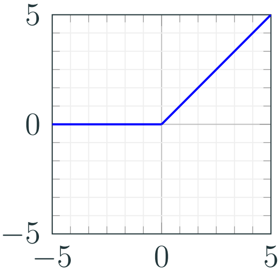


Figure 7: ReLU function

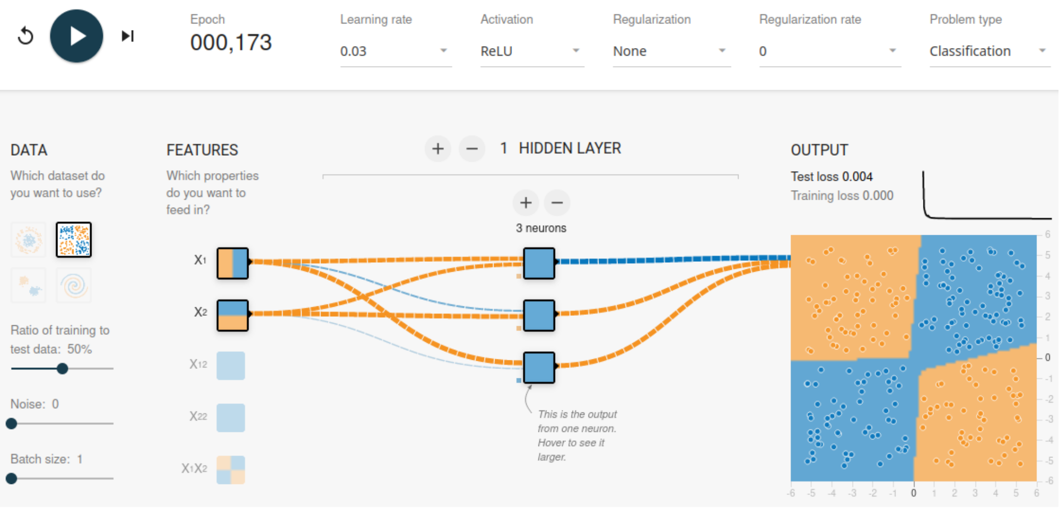


Figure 8: XOR solvable with, e.g., ReLU (`http://playground.tensorflow.org`)

XOR example in super-simplified sentiment classification

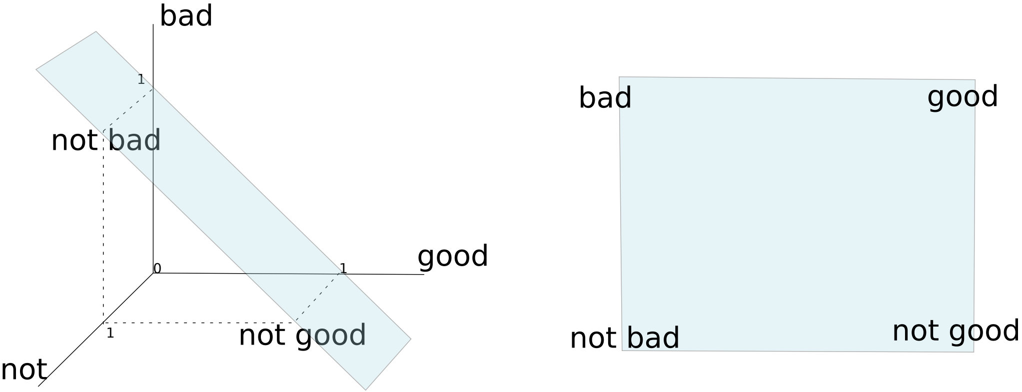


Figure 9: $V = \{\text{not}, \text{bad}, \text{good}\}$, binary features $\in \{0, 1\}$

Multi-layer perceptron (MLP) aka. feed-forward network

$$f(\boldsymbol{x}) = \sigma\left(g\left(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1\right)\boldsymbol{W}^2 + \boldsymbol{b}^2\right)$$

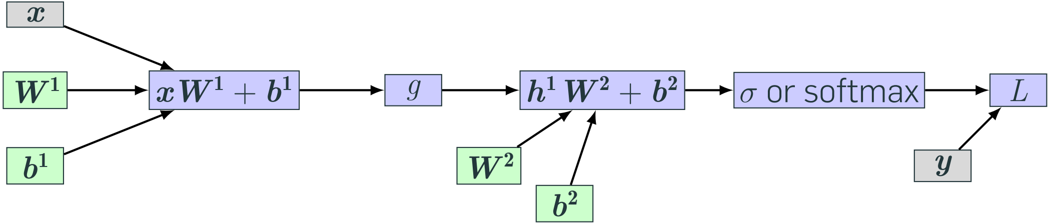


Figure 10: Computational graph; green boxes are trainable parameters, gray are constant inputs

Language modeling

Language modeling

‘Classical’ language models

Goal of language modeling

Assign a probability to sentences in a language

Example

"What is the probability of seeing the sentence *the lazy dog barked loudly*?"

Assigns a probability for the likelihood of given word (or a sequence of words) to follow a sequence of words

Example

"What is the probability of seeing the word *barked* after the seeing sequence *the lazy dog*?

Language models formally

Sequence of words $w_{1:n} = w_1 w_2 w_3 \dots w_n$ estimate

$$\Pr(w_{1:n}) = \Pr(w_1, w_2, \dots, w_n)$$

Note: We misuse notation and usually omit the RVs

$\Pr(W_1 = w_1, W_1 = w_2, \dots, W_n = w_n)$

We *factorize* the joint probability into a product

- One factorization is very useful: left-to-right

$$\Pr(w_{1:n}) = \Pr(w_1 | \text{<s>}) \Pr(w_2 | \text{<s>}, w_1) \Pr(w_3 | \text{<s>}, w_1, w_2) \cdots$$
$$\cdots \Pr(w_n | \text{<s>}, w_1, w_2, \dots, w_{n-1})$$

Simplifications in ‘classical’ language models

Despite factorization, the last term of $\Pr(w_{1:n}) =$
$\Pr(w_1 | \text{<s>}) \Pr(w_2 | \text{<s>}, w_1) \Pr(w_3 | \text{<s>}, w_1, w_2) \cdots \Pr(w_n | \text{<s>}, w_1, w_2, \ldots, w_{n-1})$
still depends on all the previous words of the sequence

k-th order markov-assumption

The next word depends only on the last k words

$$\Pr(w_i | w_{1:i-1}) \approx \Pr(w_i | w_{i-k:i-1}) \qquad \text{(inclusive indexing!)}$$

Estimating probabilities in ‘classical’ language models

Maximum Likelihood Estimation (aka. counting and dividing)

$$\hat{P}_{\mathsf{MLE}}(W_i = w | w_{i-k:i-1}) = \frac{\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1} \quad w)}{\#(w_{i-k} \quad w_{i-k+1} \quad \ldots \quad w_{i-1})}$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of 'sentence' s: $\Pr(s)$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log \left(\frac{1}{\Pr(s_i)} \right) =$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log \left(\frac{1}{\Pr(s_i)} \right) = \frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{1}{\Pr(s_i)} \right) =$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of ‘sentence’ s: $\Pr(s)$

Let’s have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log \left(\frac{1}{\Pr(s_i)} \right) = \frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{1}{\Pr(s_i)} \right) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)$$

Evaluating language models: Perplexity

Recall: Trained LM tells us probability of 'sentence' s: $\Pr(s)$

Let's have n sentences in a test corpus, each of them has a uniform probability of appearing: $\frac{1}{n}$

Then the **cross-entropy** of our model is

$$\sum_{i=1}^{n} \frac{1}{n} \log \left(\frac{1}{\Pr(s_i)} \right) = \frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{1}{\Pr(s_i)} \right) = -\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)$$

Perplexity of LM

$$2^{\text{cross-entropy}} = 2^{\left(-\frac{1}{n} \sum_{i=1}^{n} \log \Pr(s_i)\right)}$$

Shortcomings of n-gram language models

Shortcomings of n-gram language models

Long-range dependencies

- To capture a dependency between the next word and the word 10 positions in the past, we need to see a relevant 11-gram in the text

Shortcomings of n-gram language models

Y. Goldberg (2017). **Neural Network Methods for Natural Language Processing.** Morgan & Claypool, p. 108

Long-range dependencies

- To capture a dependency between the next word and the word 10 positions in the past, we need to see a relevant 11-gram in the text

Lack of generalization across contexts

- Having observed *black car* and *blue car* does not influence our estimates of the event *red car* if we haven't see it before

Language modeling

Neural language models

Neural LMs

Let's build a neural network

- Input: a k-gram of words $w_{1:k}$
- Desired output: a probability distribution over the vocabulary V for the next word w_{k+1}

Embedding layer

If the input are symbolic **categorical features**

- e.g., words from a closed vocabulary

it is common to associate each possible feature value

- i.e., each word in the vocabulary

with a d-dimensional vector for some d

These vectors are also *parameters* of the model, and are trained jointly with the other parameters

Embedding layer: Lookup operation

The mapping from a symbolic feature values such as `word-number-48` to d-dimensional vectors is performed by an embedding layer (a lookup layer)

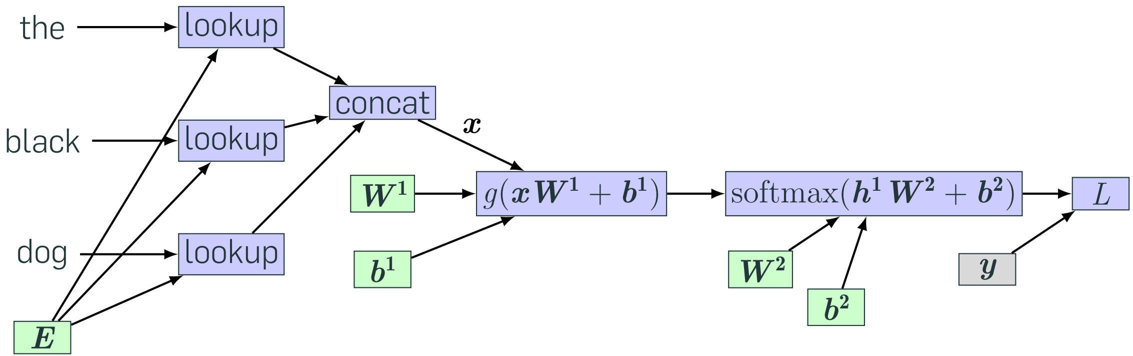
The parameters in an embedding layer is a matrix $\boldsymbol{E}^{|V| \times d}$, each row corresponds to a different word in the vocabulary

The lookup operation is then indexing $v(w)$, e.g.,

$$v(w) = v_{48} = \boldsymbol{E}_{[48,:]}$$

If the symbolic feature is encoded as a one-hot vector $\boldsymbol{x}$, the lookup operation can be implemented as the multiplication $\boldsymbol{xE}$

Network concatenating 3 words as embeddings ($d_w = 50$)



Each word $\in \mathbb{R}^{|V|}$ (one hot), $\boldsymbol{E} \in \mathbb{R}^{|V| \times 50}$, each lookup output $\in \mathbb{R}^{50}$, concat output $\boldsymbol{x} \in \mathbb{R}^{150}$

Neural LMs

Let's build a neural network

- Input: a k-gram of words $w_{1:k}$
- Desired output: a probability distribution over the vocabulary V for the next word w_{k+1}

Each input word w_k is associated with an embedding vector $v(w) \in \mathbb{R}^{d_w}$ (d_w — word embedding dimensionality)

Input vector $\boldsymbol{x}$ is a concatenation of k words

$$\boldsymbol{x} = [v(w_1); v(w_2); \ldots; v(w_k)]$$

Neural LMs

MLP with one (or more) hidden layers

$$\begin{aligned} v(w) &= \boldsymbol{E}_{w,:} \\ \boldsymbol{x} &= [v(w_1); v(w_2); \ldots; v(w_k)] \\ \boldsymbol{h} &= g(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}) \\ \hat{\boldsymbol{y}} = \Pr(W_i | w_{1:k}) &= \mathrm{softmax}(\boldsymbol{h}\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}}) \end{aligned}$$

Output dimension: $\hat{\boldsymbol{y}} \in \mathbb{R}^{|V|}$

Training neural LMs

Where to get training examples?

Training examples are simply word k-grams from an unlabeled corpus

- Identities of the first $k-1$ words are used as features
- The last word is used as the target label for the classification

The model is trained using cross-entropy loss

Some advantages and limitations of neural LMs

$\approx$ linear increase in parameters with $k+1$ (better than 'classical' LMs) but

- The size of the output vocabulary affects the computation time
- The softmax at the output layer requires an expensive matrix-vector multiplication with the matrix $\boldsymbol{W^2} \in \mathbb{R}^{d_{\text{hid}} \times |V|}$, followed by $|V|$ exponentiations

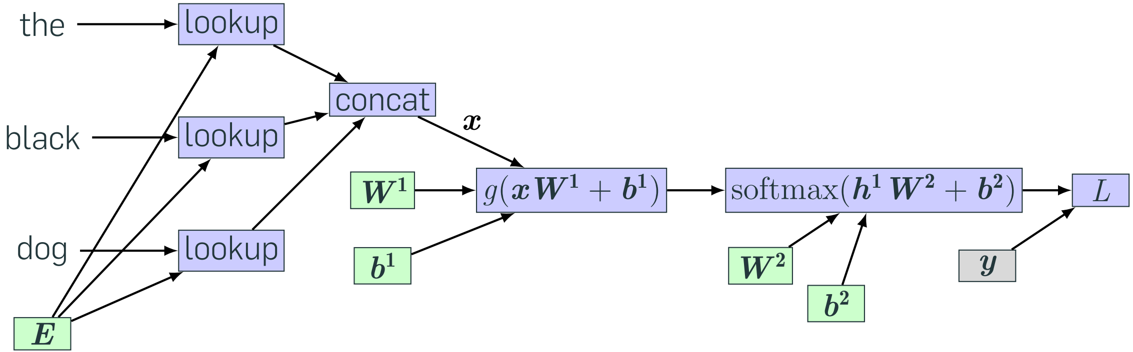
Solutions: Hierarchical softmax, noise-contrastive estimation

Generating text with language models

We can generate ("sample") random sentences from the model according to their probability

1. Predict a probability distribution over the vocabulary conditioned on the start symbol <s>
2. Draw the first word from the predicted distribution
3. Predict a probability distribution over the vocabulary conditioned on the start symbol and the first word
4. Draw the second word from the predicted distribution
5. Repeat until generated *end-of-sentence* symbol </s> (or <EOS>)

Learned word representations as a by-product



Each row of $\boldsymbol{E}$ learns a word representation

Word embeddings

Word embeddings as pre-trained word representation

Option A: We can initialize the embeddings matrix $\boldsymbol{E}$ randomly and learn during our supervised task

Option B: Use pre-trained word embeddings from task for which we have a lot of data

- Use self-supervised learning (create labeled data 'for free' using the next word prediction objective)
- Learned word embedding matrix plugged into our supervised task

Desired word embeddings properties: 'Similar' words have similar embeddings vectors

Take aways

- Language modeling is an essential part of contemporary NLP
- Self-supervised models, unlabeled data, next word prediction
- Neural language models learn embedding of words

License and credits

Credits

Ivan Habernal

Appendix: Probability refresher

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the}, \text{be}, \text{to}, \text{of}, \text{and}, \ldots\}$, we assume a fixed vocabulary

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the}, \text{be}, \text{to}, \text{of}, \text{and}, \ldots\}$, we assume a fixed vocabulary

Probability distribution over random variables

For example, probability of *'the'* at position 1

$$\Pr(W_1 = w_1) = \Pr(W_1 = \text{the}) = 0.00024$$

Probability refresher 1

Categorical random variables

For example, the first word in a sentence

$W_1 \in \{\text{the}, \text{be}, \text{to}, \text{of}, \text{and}, \ldots\}$, we assume a fixed vocabulary

Probability distribution over random variables

For example, probability of *'the'* at position 1

$\Pr(W_1 = w_1) = \Pr(W_1 = \text{the}) = 0.00024$

Notation shortcuts: $\Pr(W_1 = w_1) \rightarrow P(W_1)$, $P(\text{the})$, etc.

Probability refresher 2

Joint probability

For example, probability of *'the'* at position 1 and *'cat'* at position 2

$$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$$

Probability refresher 2

Joint probability

For example, probability of *'the'* at position 1 and *'cat'* at position 2

$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$

Notation shortcuts: $P(W_1, W_2) = P(W_2, W_1)$

Probability refresher 2

Joint probability

For example, probability of *'the'* at position 1 and *'cat'* at position 2

$$\Pr(W_1 = \text{the} \cap W_2 = \text{cat}) = 0.0000074$$

Notation shortcuts: $P(W_1, W_2) = P(W_2, W_1)$

Conditional probability

For example, probability of *'cat'* at position 2, **given** *'the'* at position 1

$$\Pr(W_2 = \text{cat} | W_1 = \text{the}) = \frac{P(W_1, W_2)}{P(W_1)}$$

Probability refresher 3

Independence

Two random variables X, Y are **independent** if and only if

$P(X, Y) = P(X) \cdot P(Y)$

Probability refresher 3

Independence

Two random variables X, Y are **independent** if and only if

$$P(X, Y) = P(X) \cdot P(Y)$$

Conditional independence

Two random variables X, Y are **conditionally independent** given Z if and only if

$$P(X, Y|Z) = P(X|Z) \cdot P(Y|Z)$$