Deep Learning for Natural Language Processing

Lecture 4 — Text classification 2: Deep neural networks

Dr. Ivan Habernal

May 2, 2023

Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

www.trusthlt.org

Where we finished last time

Our binary text classification function

Linear function through sigmoid — log-linear model

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

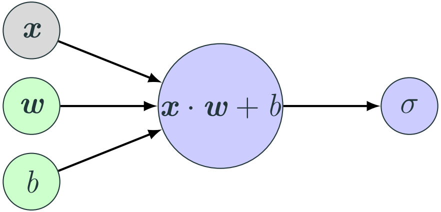


Figure 1: Computational graph; green circles are trainable parameters, gray are inputs

Decision rule of log-linear model

Log-linear model $\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1+\exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))}$

- Prediction = 1 if $\hat{y} > 0.5$
- Prediction = 0 if $\hat{y} < 0.5$

Natural interpretation: Conditional probability of prediction = 1 given the input $\boldsymbol{x}$

$$\sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 1 | \boldsymbol{x})$$
$$1 - \sigma(f(\boldsymbol{x})) = \Pr(\text{prediction} = 0 | \boldsymbol{x})$$

Finding the best model's parameters

Where we finished last time

Finding the best model's parameters

Log-linear multi-class classification

- Representations
- From multi-dimensional linear transformation to probabilities

Loss function for softmax

Stacking transformations and non-linearity

The loss function

Loss function: Quantifies the loss suffered when predicting $\hat{y}$ while the true label is y for a single example. In binary classification:

$$L(\hat{y}, y) : \mathbb{R}^2 \to \mathbb{R}$$

Given a labeled training set $(\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n})$, a per-instance loss function L and a parameterized function $f(\boldsymbol{x}; \Theta)$ we define the corpus-wide loss with respect to the parameters Θ as the average loss over all training examples

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

Training as optimization

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

The training examples are fixed, and the values of the parameters determine the loss

The goal of the training algorithm is to set the values of the parameters Θ, such that the value of $\mathcal{L}$ is minimized

$$\hat{\Theta} = \operatorname*{argmin}_{\Theta} \mathcal{L}(\Theta) = \operatorname*{argmin}_{\Theta} \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

Binary cross-entropy loss (logistic loss)

$$L_{\text{logistic}} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Partial derivative wrt. input $\hat{y}$

$$\frac{\mathrm{d}L_{\text{Logistic}}}{\mathrm{d}\hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) = -\frac{y - \hat{y}}{\hat{y}(1-\hat{y})}$$

Full computational graph

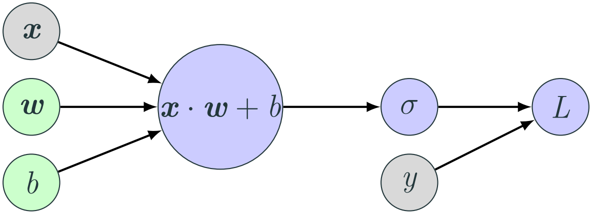


Figure 2: Computational graph; green circles are trainable parameters, gray are constant inputs

How can we minimize this function?

- Recall Lecture 2: (a) Gradient descent and (b) backpropagation

(Online) Stochastic Gradient Descent

1: **function** SGD($f(\boldsymbol{x};\Theta)$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, L)
2:     **while** stopping criteria not met **do**
3:         Sample a training example $\boldsymbol{x}_i, \boldsymbol{y}_i$
4:         Compute the loss $L(f(\boldsymbol{x}_i;\Theta), \boldsymbol{y}_i)$
5:         $\hat{\boldsymbol{g}} \leftarrow$ gradient of $L(f(\boldsymbol{x}_i;\Theta), \boldsymbol{y}_i)$ wrt. Θ
6:         $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
7:     **return** Θ

Loss in line 4 is based on a **single training example** $\rightarrow$ a rough estimate of the corpus loss $\mathcal{L}$ we aim to minimize

The noise in the loss computation may result in inaccurate gradients

Minibatch Stochastic Gradient Descent

```
function MBSGD(f(x; Θ), (x_1, ..., x_n), (y_1, ..., y_n), L)
    while stopping criteria not met do
        Sample m examples {(x_1, y_1), ... (x_m, y_m)}
        ĝ ← 0
        for i = 1 to m do
            Compute the loss L(f(x_i; Θ), y_i)
            ĝ ← ĝ + gradient of (1/m) L(f(x_i; Θ), y_i) wrt. Θ
        Θ ← Θ − η_t ĝ
    return Θ
```

Properties of Minibatch Stochastic Gradient Descent

The minibatch size can vary in size from $m = 1$ to $m = n$

Higher values provide better estimates of the corpus-wide gradients, while smaller values allow more updates and in turn faster convergence

Lines 6+7: May be easily parallelized

Log-linear multi-class classification

- Where we finished last time
- Finding the best model's parameters
- Log-linear multi-class classification
  - Representations
  - From multi-dimensional linear transformation to probabilities
- Loss function for softmax
- Stacking transformations and non-linearity

From binary to multi-class labels

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

What if we classify into distinct categorical classes?

- 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})

We can predict the language resulting in the highest score

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

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

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

The simplest neural network — a perceptron (simply a linear model)

- How to find the prediction $\hat{y}$?

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

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

Log-linear multi-class classification

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

Matrix $\boldsymbol{W}$ as learned representation — columns

$\hat{\boldsymbol{y}} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$ $\quad \rightarrow$ two views of $\boldsymbol{W}$, as rows or as columns

	En	Fr	De	It	Es	Ot
a	•	•	•	•	•	•
at	•	•	•	•	•	•
...						
zoo	•	•	•	•	•	•

Each of the 6 columns (corresponding to a language) is a d_{in}-dimensional vector representation of this language in terms of its characteristic word unigram patterns (e.g., we can then cluster the 6 language vectors according to their similarity)

Matrix $\boldsymbol{W}$ as learned representation — rows

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

	En	Fr	De	It	Es	Ot
a	•	•	•	•	•	•
at	•	•	•	•	•	•
...						
zoo	•	•	•	•	•	•

Each of the d_{in} rows corresponds to a particular unigram, and provides a 6-dimensional vector representation of that unigram in terms of the languages it prompts

From bag-of-words to continuous bag-of-words

Recall from lecture 3 — Averaged bag of words

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}}$$

$D_{[i]}$ — word in doc D at position i, $\boldsymbol{x}^{D_{[i]}}$ — one-hot vector

$$\hat{\boldsymbol{y}} = \boldsymbol{x}\boldsymbol{W} = \left(\frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}} \right) \boldsymbol{W} = \frac{1}{|D|} \sum_{i=1}^{|D|} \left(\boldsymbol{x}^{D_{[i]}} \boldsymbol{W} \right)$$

$$= \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{W}^{D_{[i]}}$$

(we ignore the bias $\boldsymbol{b}$ here)

From bag-of-words to continuous bag-of-words (CBOW)

Two equivalent views; $\boldsymbol{W}^{D_{[i]}}$ is the $D_{[i]}$-th row of matrix $\boldsymbol{W}$

$$\hat{\boldsymbol{y}} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{W}^{D_{[i]}} \qquad \hat{\boldsymbol{y}} = \left(\frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}} \right) \boldsymbol{W}$$

The continuous-bag-of-words (CBOW) representation

- Either by summing word-representation vectors
- Or by multiplying a bag-of-words vector by a matrix in which each row corresponds to a dense word representation (also called **embedding matrix**)

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

Log-linear multi-class classification

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, ...

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 $\text{softmax}(\boldsymbol{x}) : \mathbb{R}^K \to \mathbb{R}^K$

Softmax

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

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

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

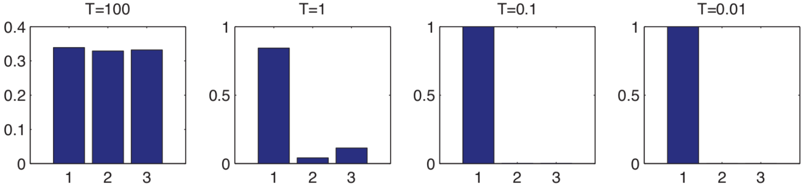
We also need to know how to compute the partial derivative of $\text{softmax}(\boldsymbol{x}_{[i]})$ wrt. each argument $\boldsymbol{x}_{[k]}$: $\frac{\partial \, \text{softmax}(\boldsymbol{x}_{[i]})}{\partial \boldsymbol{x}_{[k]}}$

Softmax can be smoothed with a 'temperature' T

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

Figure: K. P. Murphy (2012). *Machine Learning: A Probabilistic Perspective*. Cambridge, Massachusetts: The MIT Press, p. 103

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



High temperature $\rightarrow$ uniform distribution

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

Loss function for softmax

- Where we finished last time
- Finding the best model's parameters
- Log-linear multi-class classification
  - Representations
  - From multi-dimensional linear transformation to probabilities
- Loss function for softmax
- Stacking transformations and non-linearity

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

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

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

Output from softmax:

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

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

Cross entropy loss

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

Background: K-L divergence (lso known as *relative entropy*)

Let Y and $\hat{Y}$ be categorical random variables over same categories, with probability distributions $P(Y)$ and $Q(\hat{Y})$

$$
\begin{aligned}
\mathbb{D}(P(Y)||Q(\hat{Y})) &= \mathbb{E}_{P(Y)}\left[\log \frac{P(Y)}{Q(\hat{Y})}\right] \\
&= \mathbb{E}_{P(Y)}\left[\log P(Y) - \log Q(\hat{Y})\right] \\
&= \mathbb{E}_{P(Y)}\left[\log P(Y)\right] - \mathbb{E}_{P(Y)}\left[\log Q(\hat{Y})\right] \\
&= -\mathbb{E}_{P(Y)}\left[\log \frac{1}{P(Y)}\right] - \mathbb{E}_{P(Y)}\left[\log Q(\hat{Y})\right] \\
&= -\mathbb{H}_P(Y) - \mathbb{E}_{P(Y)}\left[\log Q(\hat{Y})\right]
\end{aligned}
$$

Stacking transformations and non-linearity

Where we finished last time
Finding the best model's parameters
Log-linear multi-class classification
- Representations
- From multi-dimensional linear transformation to probabilities

Loss function for softmax
Stacking transformations and non-linearity

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

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

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

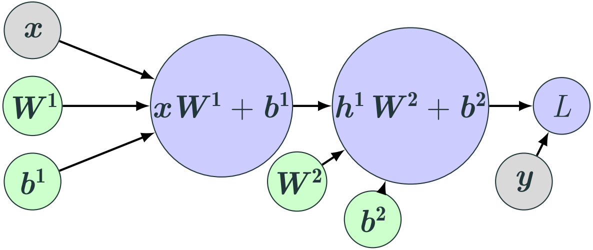


Figure 3: Computational graph; green circles are trainable parameters, gray are constant inputs

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

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

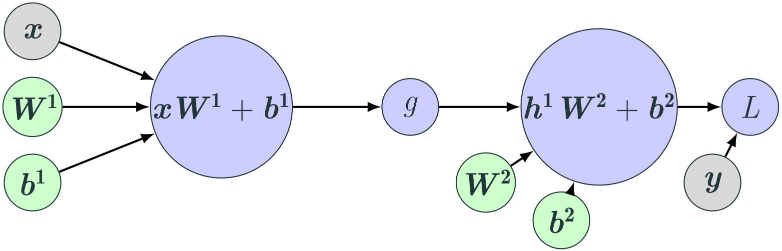


Figure 4: Computational graph; green circles are trainable parameters, gray are constant inputs

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

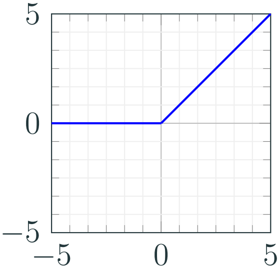


Figure 5: ReLU function

Recap

Where we finished last time

Finding the best model's parameters

Log-linear multi-class classification

- Representations
- From multi-dimensional linear transformation to probabilities

Loss function for softmax

Stacking transformations and non-linearity

Take aways

- Binary classification as a linear function of words and a sigmoid
- Binary cross-entropy (logistic) loss
- Training as minimizing the loss using minibatch SGD and backpropagation
- Stacking layers and non-linear functions: MLP
- ReLU as a go-to activation function in NLP

License and credits

Licensed under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`