Natural Language Processing with Deep Learning

Lecture 4 — Text classification 2: Deep neural networks

Prof. Dr. Ivan Habernal

November 3, 2023

Natural Language Processing Group
Paderborn University
We focus on Trustworthy Human Language Technologies

PADERBORN UNIVERSITY

www.trusthlt.org

Where we finished last time

Where we finished last time

Log-linear multi-class classification

- Representations
- From multi-dimensional linear transformation to probabilities

Loss function for softmax

Stacking transformations and non-linearity

Our binary text classification function

Linear function through sigmoid — log-linear model

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

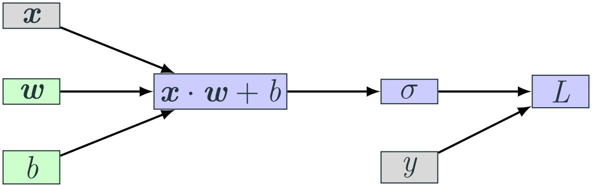


Figure 1: Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this function?

(Online) Stochastic Gradient Descent

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

(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

(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($f(\boldsymbol{x};\Theta)$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, L)

  while stopping criteria not met **do**

    Sample m examples $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$

    $\hat{\boldsymbol{g}} \leftarrow 0$

    for $i = 1$ to m **do**

      Compute the loss $L(f(\boldsymbol{x}_i;\Theta), \boldsymbol{y}_i)$

      $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ gradient of $\frac{1}{m} L(f(\boldsymbol{x}_i;\Theta), \boldsymbol{y}_i)$ wrt. Θ

    $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$

  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

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)

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})

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

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

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

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

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

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

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 $\hat{y}$?

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

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

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 W as learned representation — columns

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

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

Matrix 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 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

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

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

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

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

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

$$=$$

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

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

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

Two equivalent views; $W^{D_{[i]}}$ 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

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

Two equivalent views; $W^{D_{[i]}}$ 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

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

Two equivalent views; $W^{D_{[i]}}$ 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, $\ldots$

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, $\ldots$

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, $\ldots$

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, $\ldots$

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

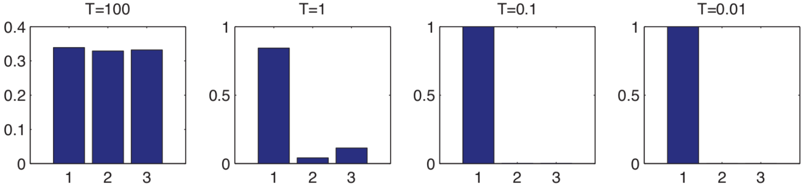
$$\text{softmax}(\boldsymbol{x}_{[i]}; T) = \frac{\exp\left(\frac{\boldsymbol{x}_{[i]}}{T}\right)}{\sum_{k=1}^{K} \exp\left(\frac{\boldsymbol{x}_{[k]}}{T}\right)}$$

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

- Where we finished last time
- 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 $1, \ldots, K$:

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

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

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

Background: K-L divergence (also 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
- Log-linear multi-class classification
  - Representations
  - From multi-dimensional linear transformation to probabilities
- Loss function for softmax
- Stacking transformations and non-linearity

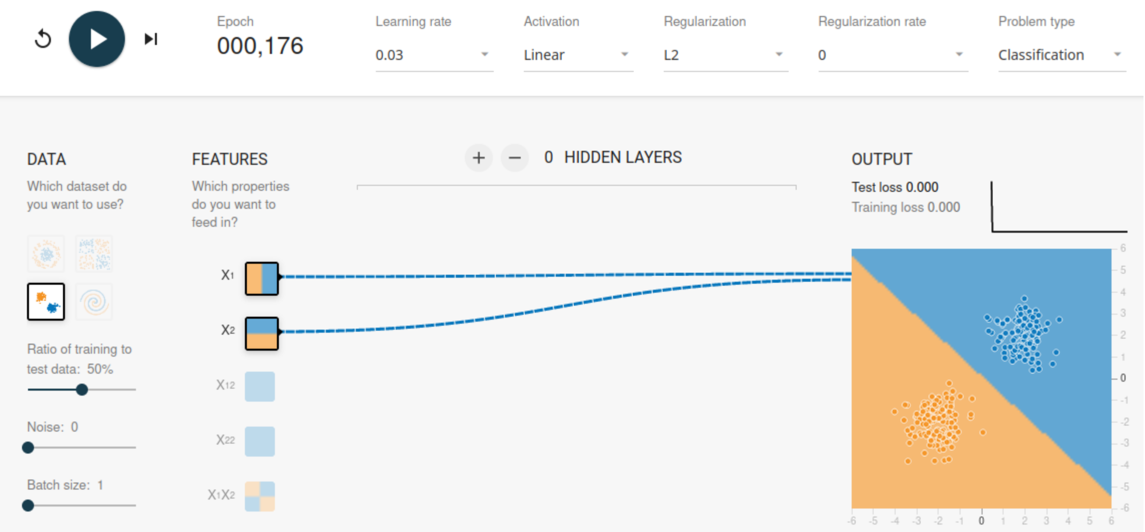


Figure 2: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

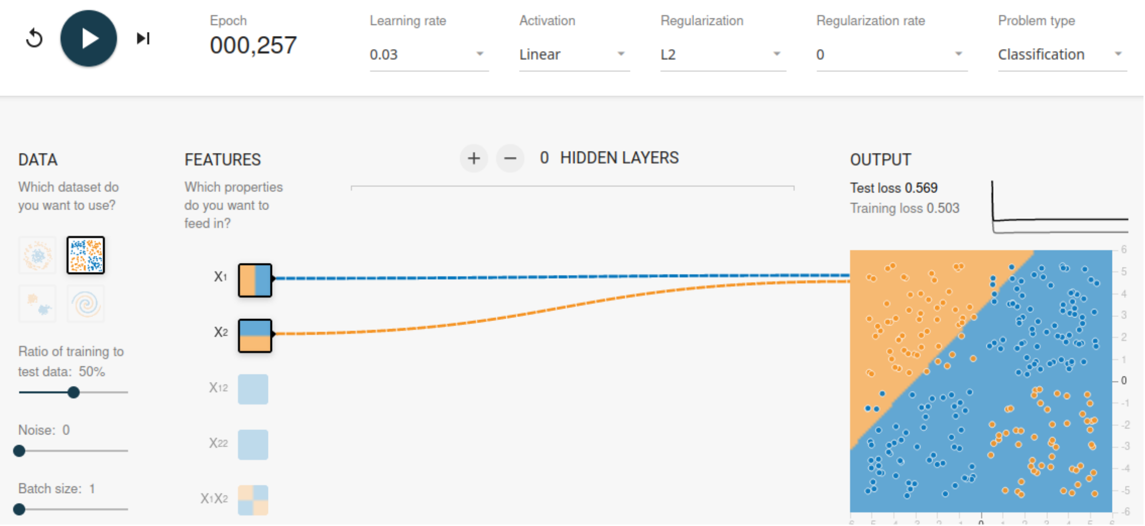


Figure 3: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

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

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

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

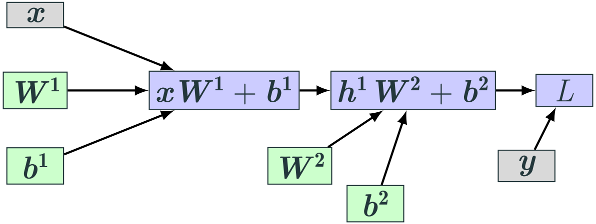


Figure 4: Computational graph; green circles are trainable parameters, gray are constant inputs

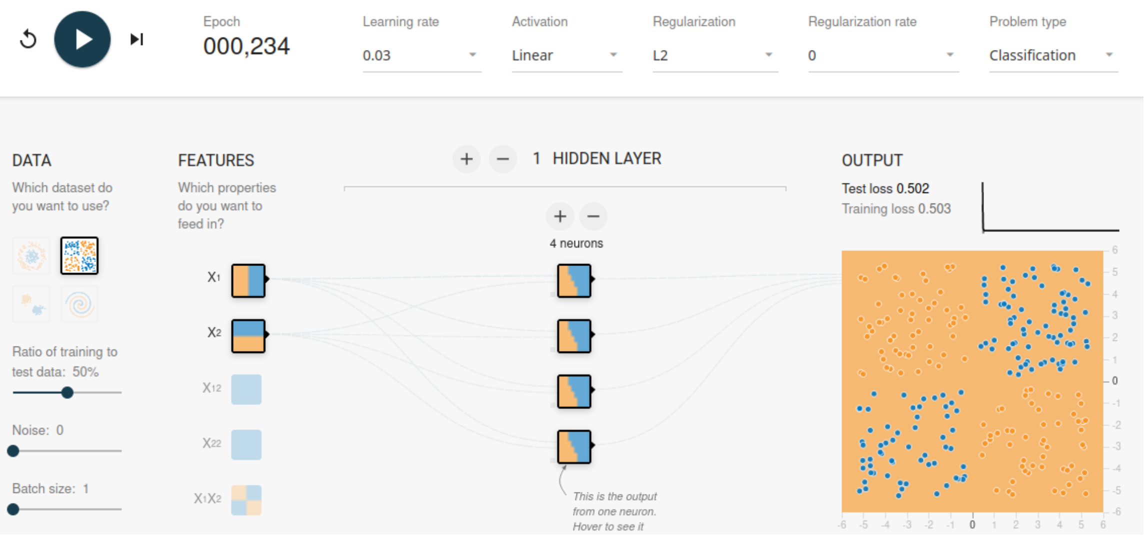


Figure 5: Linear hidden layers do not help (`http://playground.tensorflow.org`)

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

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

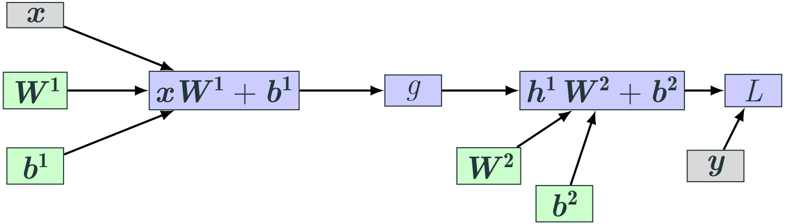


Figure 6: Computational graph; green circles are trainable parameters, gray are constant inputs

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

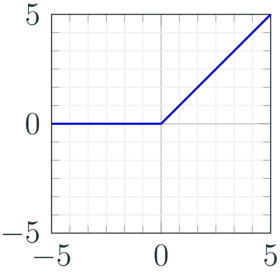


Figure 7: ReLU function

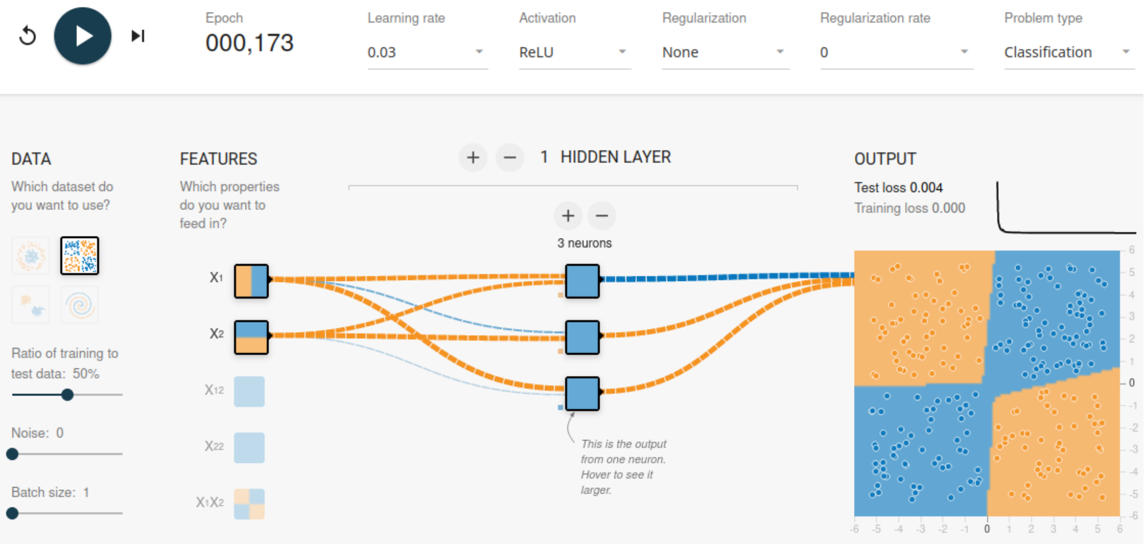


Figure 8: XOR solvable with, e.g., ReLU (`http://playground.tensorflow.org`)

XOR example in super-simplified sentiment classification

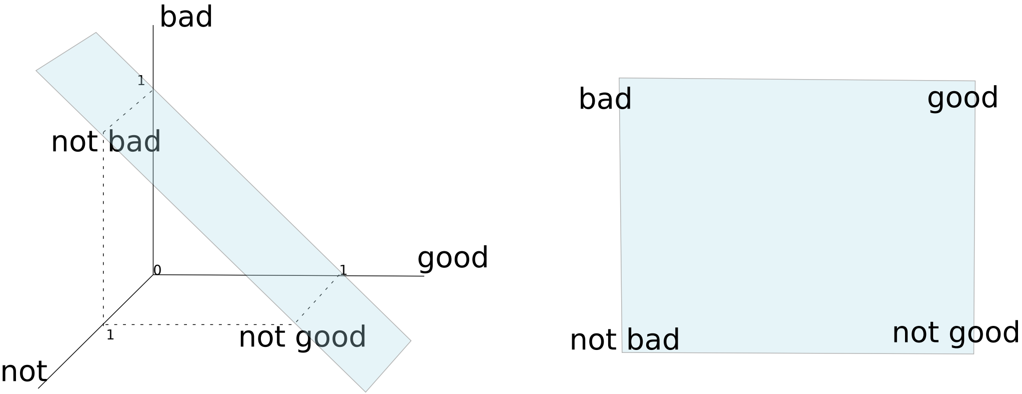


Figure 9: $V = \{\text{not}, \text{bad}, \text{good}\}$, binary features $\in \{0, 1\}$

Recap

Where we finished last time

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 (Multi-Layer Perceptron)
- ReLU as a go-to activation function in NLP

License and credits

Credits

Ivan Habernal