

Natural Language Processing with Deep Learning

RUHR
UNIVERSITÄT
BOCHUM

RUB

Lecture 4 — Text classification and feed-forward networks

Prof. Dr. Ivan Habernal

November 6, 2025

www.trusthlt.org

Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security



This lecture

- Recap: Binary text classification
- Log-linear models, Cross-entropy loss, Stochastic gradient descent
- Multi-class classification and softmax
- Deep neural networks

Addressing your feedback

Overlap of exercise with 2 other classes in my Applied Informatics master

no bonus points and it's not entirely clear how the exam is set up / mock exam not available

exercise 3 felt a bit disconnected from the lecture

slow pace of lectures

Recap: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\mathbf{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \quad f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b$$

What we need:

$$\mathbf{x} \in \mathbb{R}^{d_{in}}$$

What we have:

*One of my favorite movies ever, The Shawshank Redemption
is a modern day classic ...*

Simple solution:

- Bag-of-words (tokenized), $d_{in} = |V|$

Binary text classification

- 1 Binary text classification
- 2 From binary to multi-class task
- 3 Loss function for softmax
- 4 Stacking transformations and non-linearity

Binary text classification

Binary classification as a function

Linear function and its derivatives

We have this linear function

$$f(\mathbf{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \quad f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b = x_{[1]} w_{[1]} + \dots + x_{[d_{in}]} w_{[d_{in}]} + b$$

Linear function and its derivatives

We have this linear function

$$f(\mathbf{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \quad f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b = x_{[1]} w_{[1]} + \dots + x_{[d_{in}]} w_{[d_{in}]} + b$$

Derivatives wrt. parameters w and b

$$\frac{df}{d\mathbf{w}_{[i]}} = \mathbf{x}_{[i]} \quad \frac{df}{db} = 1$$

Non-linear mapping to $[0, 1]$

We have this linear function

$$f(\mathbf{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \quad f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b$$

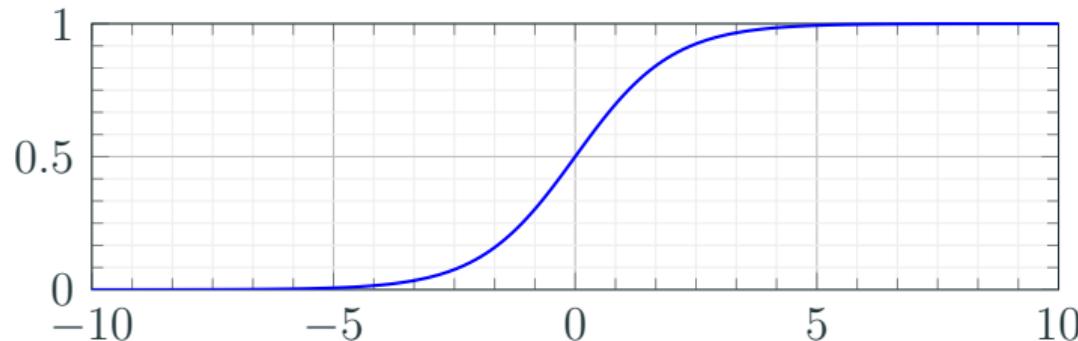
which has an unbounded range $(-\infty, +\infty)$

However, each example's label is $y \in \{0, 1\}$

Sigmoid (logistic) function

Sigmoid function $\sigma(t) : \mathbb{R} \rightarrow \mathbb{R}$

$$\sigma(t) = \frac{\exp(t)}{\exp(t) + 1} = \frac{1}{1 + \exp(-t)}$$



Symmetric function, range of $\sigma(t) \in [0, 1]$,

$$\textbf{Sigmoid } \sigma(t) = \frac{1}{1+\exp(-t)}$$

Derivative of sigmoid wrt. its input

$$\begin{aligned}\frac{d\sigma}{dt} &= \frac{\exp(t) \cdot (1 + \exp(t)) - \exp(t) \cdot \exp(t)}{(1 + \exp(t))^2} \\&= \dots \\&= \sigma(t) \cdot (1 - \sigma(t))\end{aligned}$$

Our binary text classification function

Linear function through sigmoid — log-linear model

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

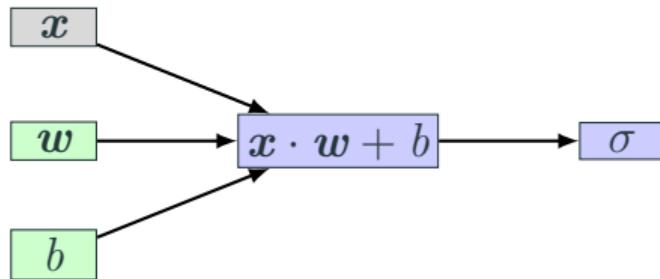


Figure 1: Computational graph; green nodes are trainable parameters, gray are inputs

Decision rule of log-linear model

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

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

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

$$\sigma(f(\mathbf{x})) = \Pr(\text{prediction} = 1 | \mathbf{x})$$

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

Binary text classification

Finding the best model's parameters

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}

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

Full computational graph

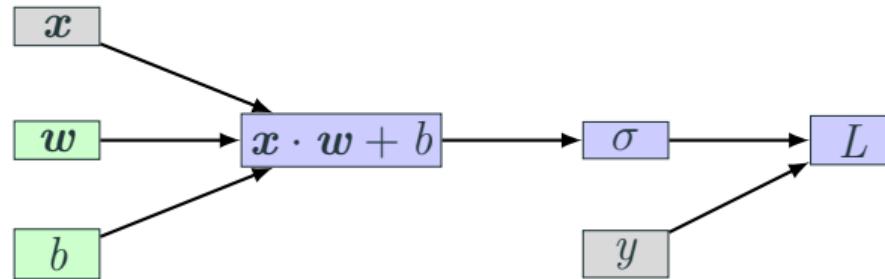


Figure 2: Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. w and b ?

Full computational graph

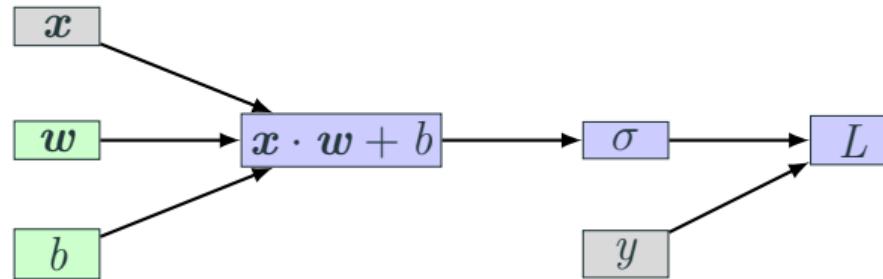


Figure 2: Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. w and b ?

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

(Online) Stochastic Gradient Descent

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

Loss in line 4 is based on a **single training example** → 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

```
1: function mbSGD( $f(\mathbf{x}; \Theta)$ ,  $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ ,  $(\mathbf{y}_1, \dots, \mathbf{y}_n)$ ,  $L$ )
2:   while stopping criteria not met do
3:     Sample  $m$  examples  $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_m, \mathbf{y}_m)\}$ 
4:      $\hat{\mathbf{g}} \leftarrow 0$ 
5:     for  $i = 1$  to  $m$  do
6:       Compute the loss  $L(f(\mathbf{x}_i; \Theta), \mathbf{y}_i)$ 
7:        $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \text{gradient of } \frac{1}{m} L(f(\mathbf{x}_i; \Theta), \mathbf{y}_i) \text{ wrt. } \Theta$ 
8:      $\Theta \leftarrow \Theta - \eta_t \hat{\mathbf{g}}$ 
9:   return  $\Theta$ 
```

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

From binary to multi-class task

- 1 Binary text classification
- 2 From binary to multi-class task
- 3 Loss function for softmax
- 4 Stacking transformations and non-linearity

Our binary text classification function

Linear function through sigmoid — log-linear model

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

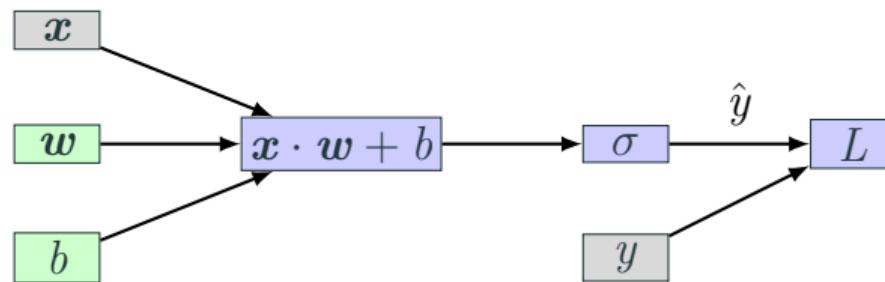


Figure 3: Computational graph; green nodes are trainable parameters, gray are constant inputs

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

$$\begin{aligned} \text{En} &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & \text{Fr} &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \\ \text{De} &= \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} & \dots & \end{aligned}$$

$\mathbf{y} \in \mathbb{R}^{d_{out}}$ 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 w^ℓ (e.g., w^{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 \mathbf{w}^ℓ (e.g., \mathbf{w}^{Fr})
- Bias b^ℓ (e.g., b^{Fr})

We can predict the language resulting in the highest score

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

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 \mathbf{w}^ℓ (e.g., \mathbf{w}^{Fr})
- Bias b^ℓ (e.g., b^{Fr})

We can predict the language resulting in the highest score

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

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

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

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

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

Recall from lecture 3: High-dimensional linear functions

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

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

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

Prediction of multi-class classifier

Project the input \mathbf{x} to an output \mathbf{y}

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

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

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

Sanity check

What is \hat{y} ?

Prediction of multi-class classifier

Project the input \mathbf{x} to an output \mathbf{y}

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

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

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

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 $\mathbf{De} = (0 \ 0 \ 1 \ 0 \ 0 \ 0)$

From binary to multi-class task

Representations

Two representations of the input document

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

Vector \mathbf{x} is a document representation

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

Vector \hat{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 $1, 2, \dots, K$, for example En = 1, De = 2, ...

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 $1, 2, \dots, K$, 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 $1, 2, \dots, K$, 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 $1, 2, \dots, K$, 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 $\text{softmax}(\boldsymbol{x}) : \mathbb{R}^K \rightarrow \mathbb{R}^K$

Softmax

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

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

The softmax function $\text{softmax}(\mathbf{x}) : \mathbb{R}^K \rightarrow \mathbb{R}^K$

Softmax

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

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

- Nominator: Non-linear bijection from \mathbb{R} to $(0; \infty)$
- Denominator: Normalizing constant to ensure

$$\sum_{j=1}^K \text{softmax}(\mathbf{x}_{[j]}) = 1$$

The softmax function $\text{softmax}(\mathbf{x}) : \mathbb{R}^K \rightarrow \mathbb{R}^K$

Softmax

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

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

- Nominator: Non-linear bijection from \mathbb{R} to $(0; \infty)$
- Denominator: Normalizing constant to ensure

$$\sum_{j=1}^K \text{softmax}(\mathbf{x}_{[j]}) = 1$$

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

Softmax can be smoothed with a ‘temperature’ T

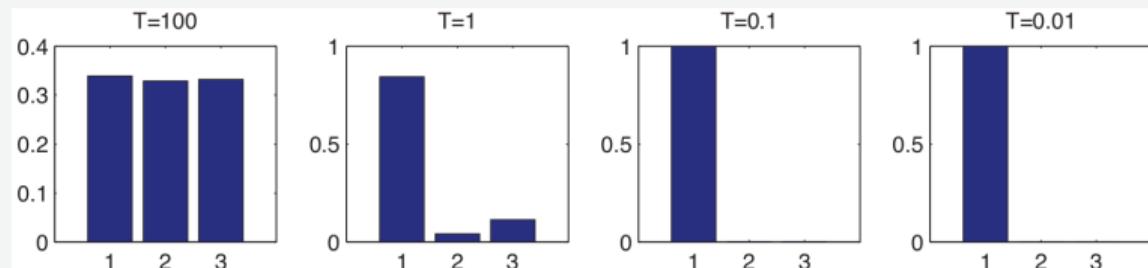
$$\text{softmax}(\mathbf{x}_{[i]}; T) = \frac{\exp\left(\frac{\mathbf{x}_{[i]}}{T}\right)}{\sum_{k=1}^K \exp\left(\frac{\mathbf{x}_{[k]}}{T}\right)}$$

Softmax can be smoothed with a ‘temperature’ T

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

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

- 1 Binary text classification
- 2 From binary to multi-class task
- 3 Loss function for softmax
- 4 Stacking transformations and non-linearity

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

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

$$\mathbf{y} = (\mathbf{y}_{[1]}, \mathbf{y}_{[2]}, \dots, \mathbf{y}_{[K]})$$

Output from softmax:

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

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

Cross entropy loss

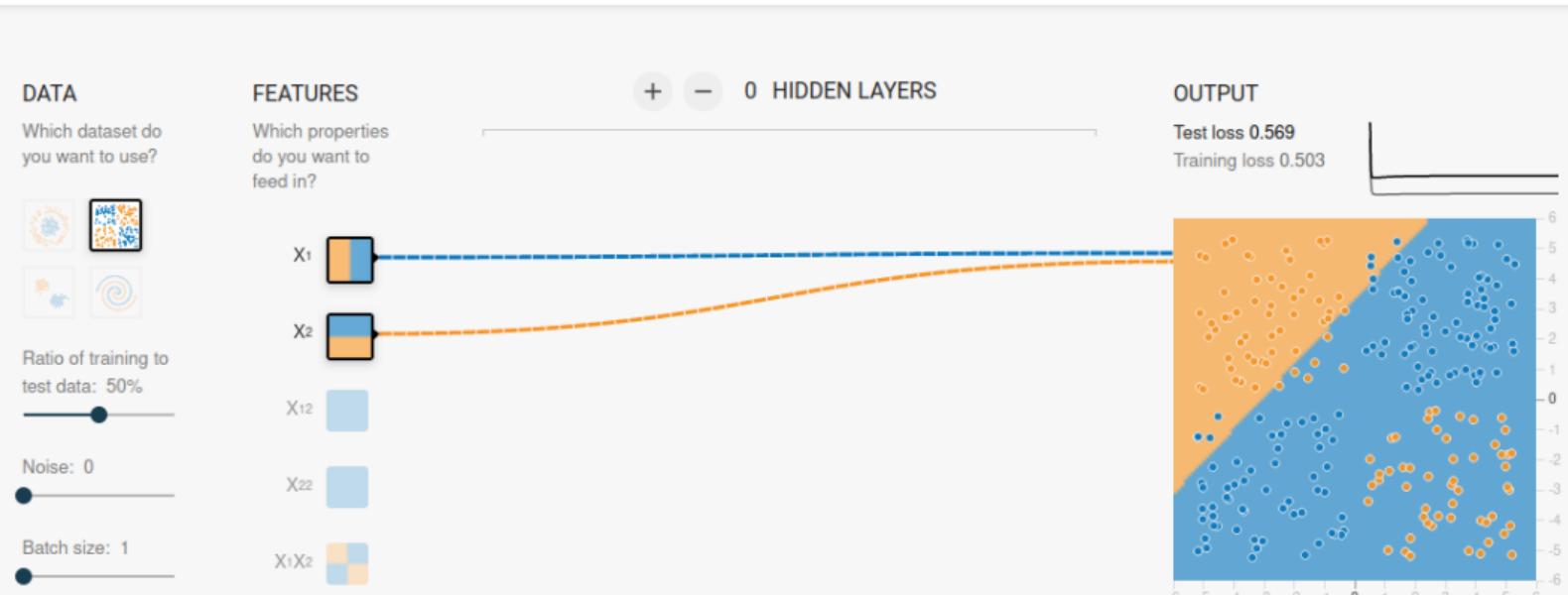
$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K \mathbf{y}_{[k]} \log (\hat{\mathbf{y}}_{[k]})$$

Stacking transformations and non-linearity

- 1 Binary text classification
- 2 From binary to multi-class task
- 3 Loss function for softmax
- 4 Stacking transformations and non-linearity

Epoch
000,176Learning rate
0.03Activation
LinearRegularization
L2Regularization rate
0Problem type
Classification

Figure 4: Linear model can tackle only linearly-separable problems (<http://playground.tensorflow.org>)

Epoch
000,257Learning rate
0.03Activation
LinearRegularization
L2Regularization rate
0Problem type
Classification

FEATURES + - 0 HIDDEN LAYERS

OUTPUT

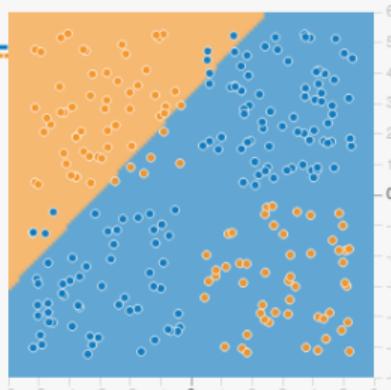
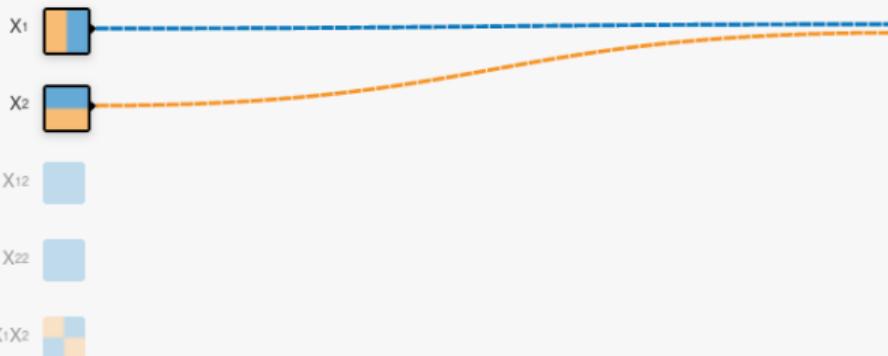
Test loss 0.569
Training loss 0.503

Figure 5: Linear model can tackle only linearly-separable problems (<http://playground.tensorflow.org>)

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

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

$$\boldsymbol{W}^1 \in \mathbb{R}^{d_{in} \times d_1}$$

$$\boldsymbol{b}^1 \in \mathbb{R}^{d_1}$$

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

$$\boldsymbol{b}^2 \in \mathbb{R}^{d_{out}}$$

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

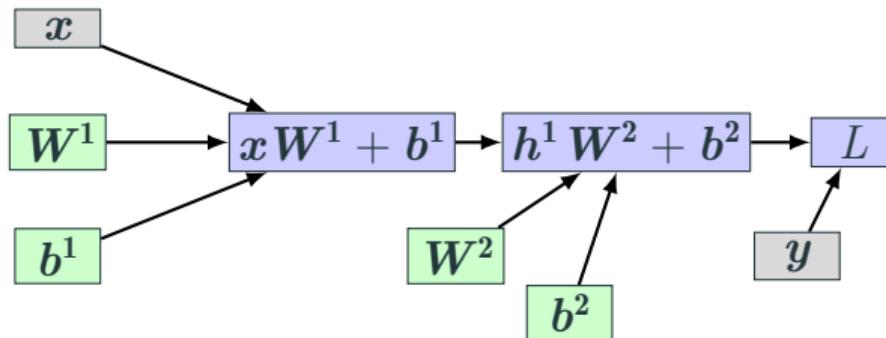


Figure 6: Computational graph; green circles are trainable parameters, gray are constant inputs

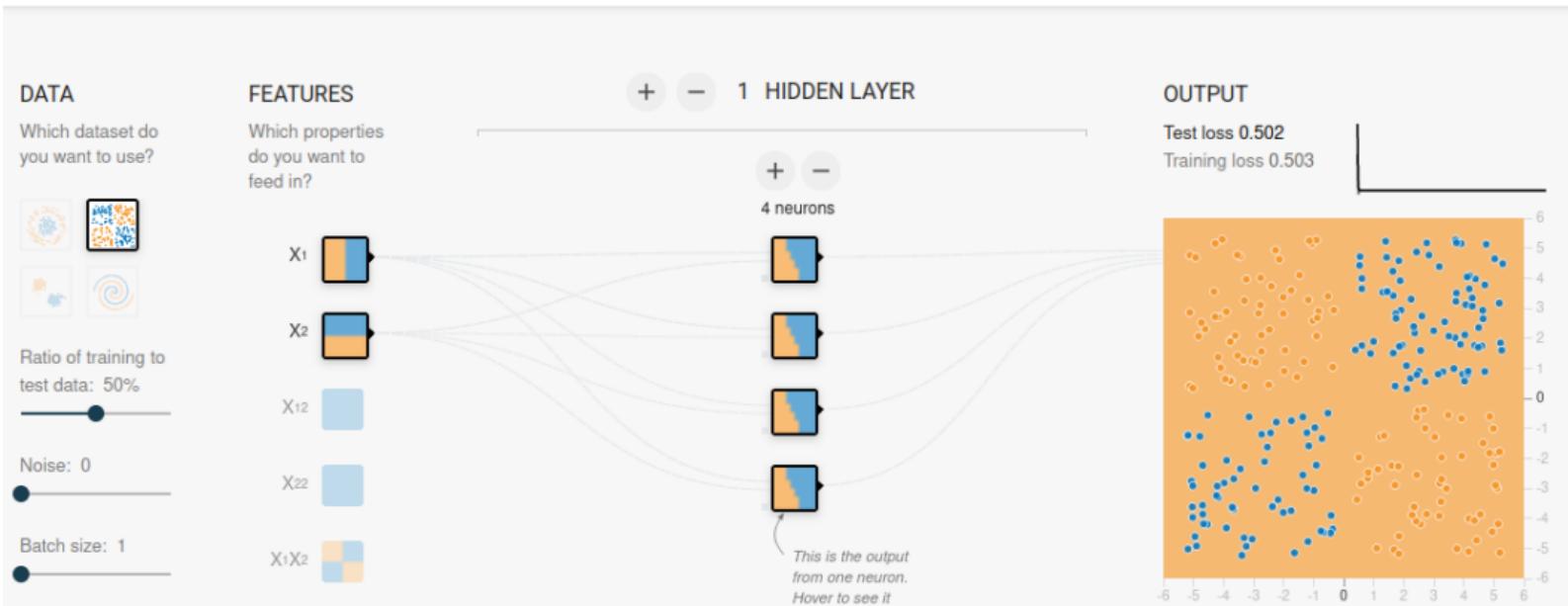
Epoch
000,234Learning rate
0.03Activation
LinearRegularization
L2Regularization rate
0Problem type
Classification

Figure 7: Linear hidden layers do not help
(<http://playground.tensorflow.org>)

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

$$f(x) = g(xW^1 + b^1)W^2 + b^2$$

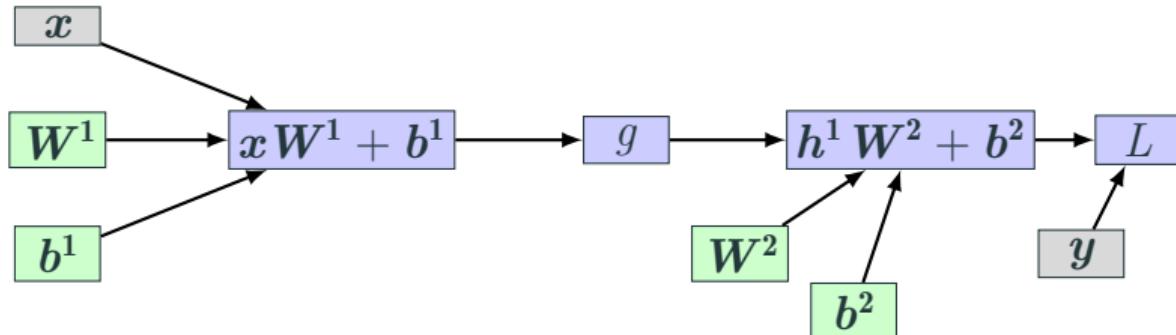


Figure 8: Computational graph; green circles are trainable parameters, gray are constant inputs

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

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

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

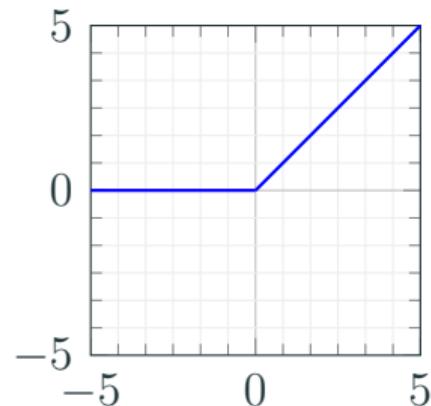


Figure 9: ReLU function

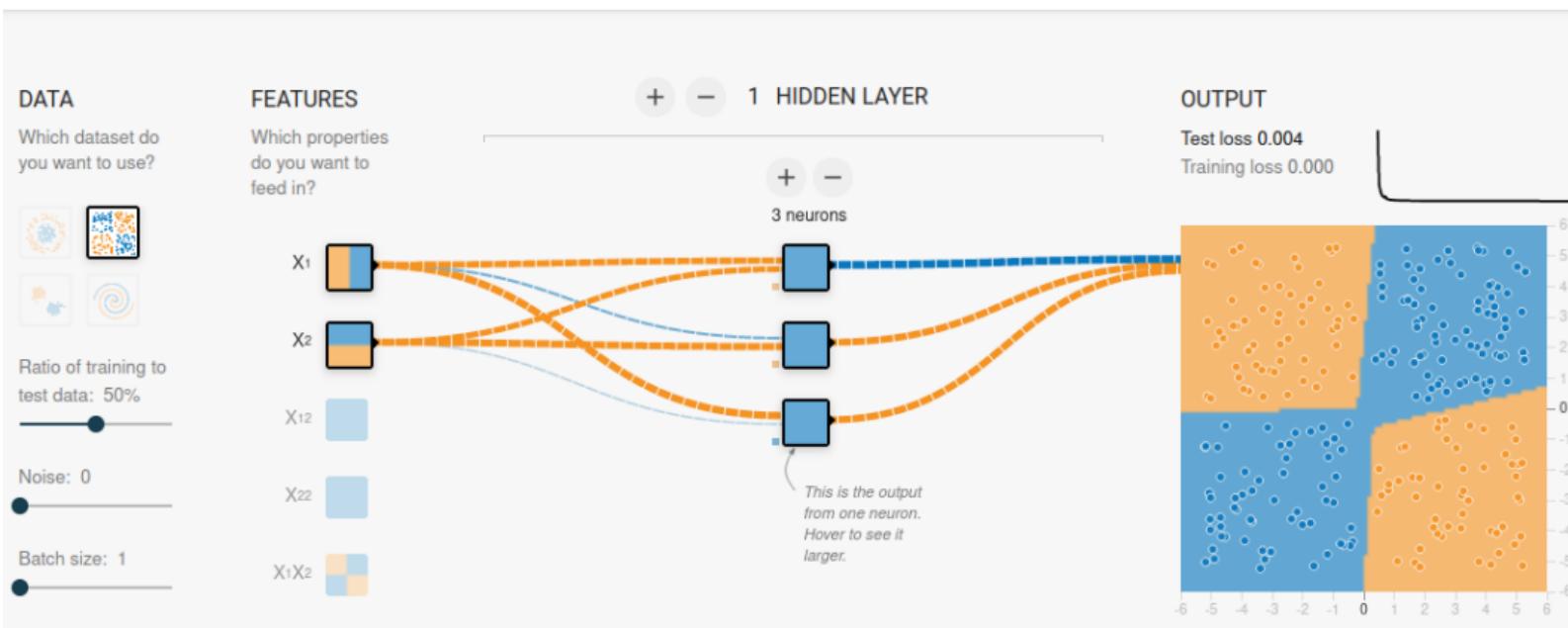
Epoch
000,173Learning rate
0.03Activation
ReLURegularization
NoneRegularization rate
0Problem type
Classification

Figure 10: XOR solvable with, e.g., ReLU
(<http://playground.tensorflow.org>)

XOR example in super-simplified sentiment classification

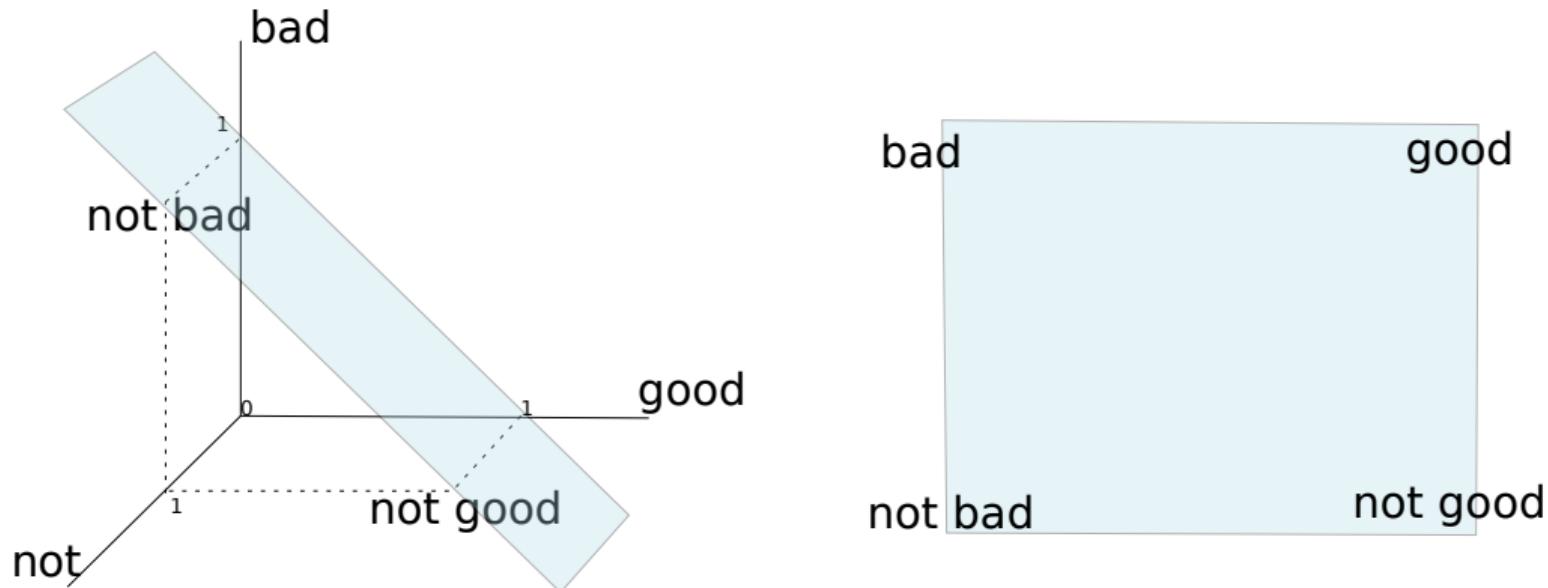


Figure 11: $V = \{\text{not}, \text{bad}, \text{good}\}$, binary features $\in \{0, 1\}$

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

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

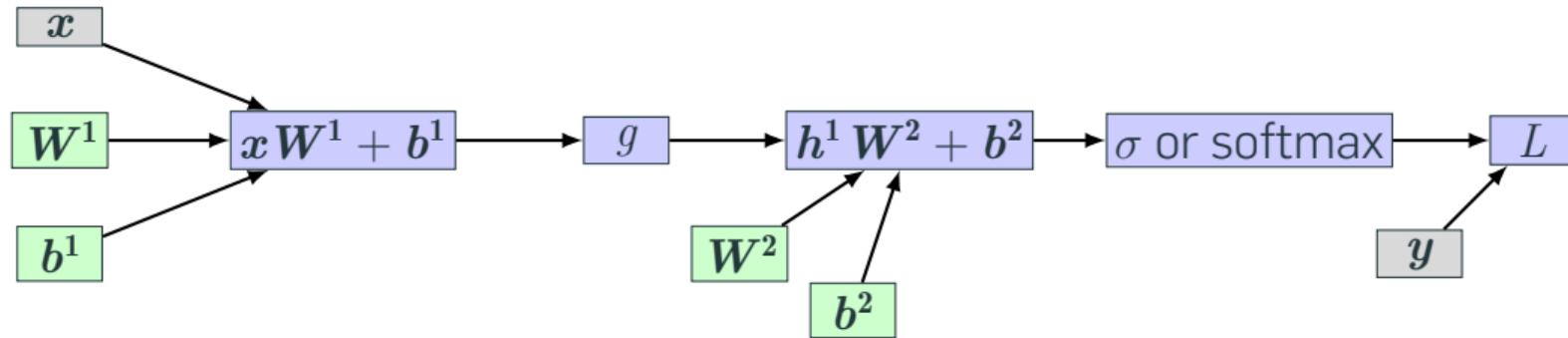


Figure 12: Computational graph; green boxes are trainable parameters, gray are constant inputs

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>