Natural Language Processing with Deep Learning

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

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

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

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

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

Binary text classification

Binary classification as a function

Linear function and its derivatives

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = \boldsymbol{x}_{[1]} \boldsymbol{w}_{[1]} + \ldots + \boldsymbol{x}_{[d_{in}]} \boldsymbol{w}_{[d_{in}]} + b$$

Linear function and its derivatives

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = \boldsymbol{x}_{[1]} \boldsymbol{w}_{[1]} + \ldots + \boldsymbol{x}_{[d_{in}]} \boldsymbol{w}_{[d_{in}]} + b$$

Derivatives wrt. parameters w and b

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{w}_{[i]}} = \boldsymbol{x}_{[i]} \qquad \frac{\mathrm{d}f}{\mathrm{d}b} = 1$$

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

We have this linear function

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

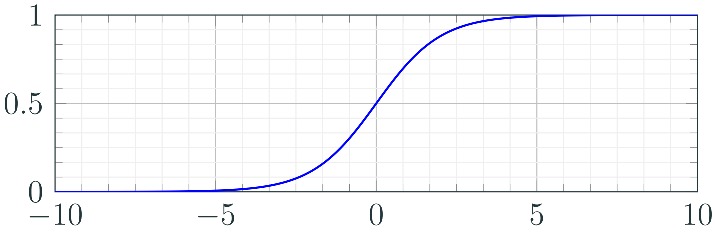
which has an unbounded range $(-\infty, +\infty)$

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

Sigmoid (logistic) function

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

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



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

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

Derivative of sigmoid wrt. its input

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

Our binary text classification function

Linear function through sigmoid — log-linear model

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

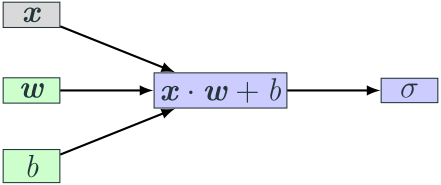


Figure 1: Computational graph; green nodes are trainable parameters, gray are inputs

Decision rule of log-linear model

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

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

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

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

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 $\hat{y}$

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

Full computational graph

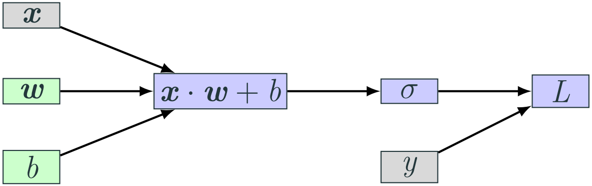


Figure 2: Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. $\boldsymbol{w}$ and b?

Full computational graph

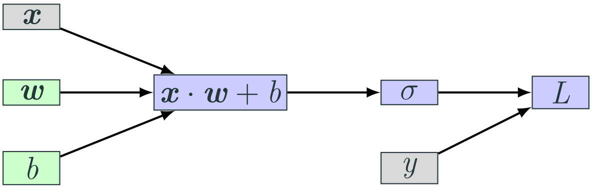


Figure 2: Computational graph; green nodes are trainable parameters, gray are constant inputs

How can we minimize this loss function wrt. $\boldsymbol{w}$ and b?

- Recall: (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

From binary to multi-class task

Our binary text classification function

Linear function through sigmoid — log-linear model

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

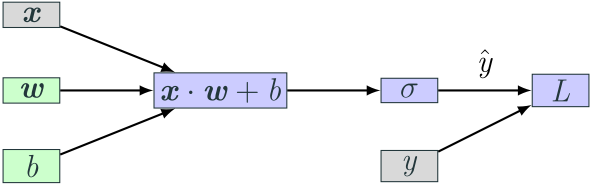


Figure 3: Computational graph; green nodes are trainable parameters, gray are constant inputs

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

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

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

$\boldsymbol{y} \in \mathbb{R}^{d_{out}}$ 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

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

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

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

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

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

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 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, ...

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

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

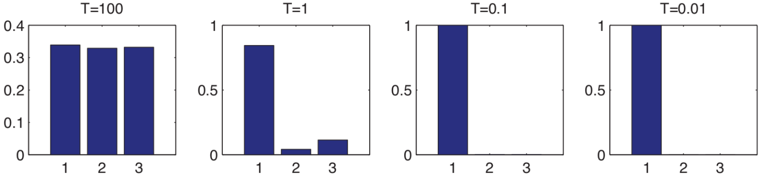
$$\operatorname{softmax}(\boldsymbol{x}_{[i]}; T) = \frac{\exp\left(\frac{\boldsymbol{x}_{[i]}}{T}\right)}{\sum_{k=1}^{K} \exp\left(\frac{\boldsymbol{x}_{[k]}}{T}\right)}$$

Softmax can be smoothed with a ‘temperature’ T

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

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

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



High temperature $\rightarrow$ uniform distribution

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

Loss function for softmax

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

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

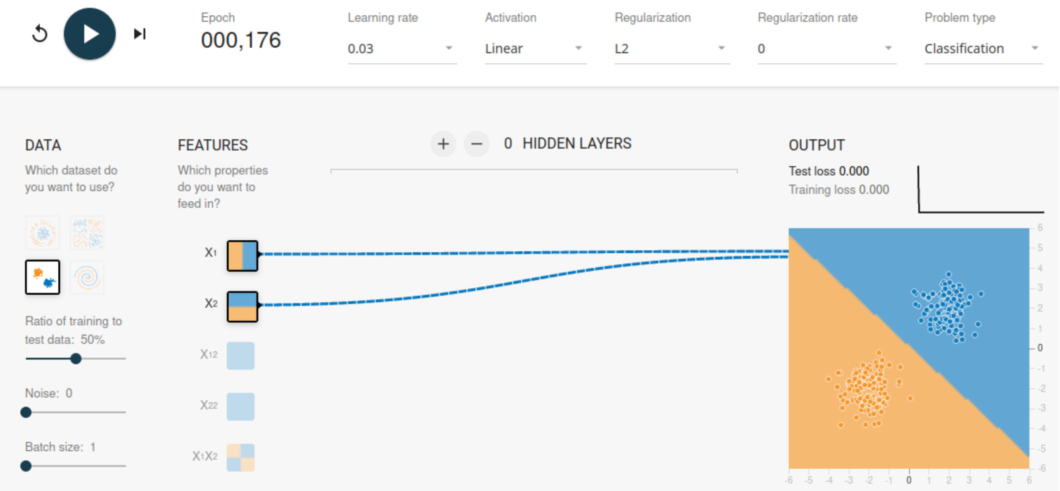


Figure 4: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

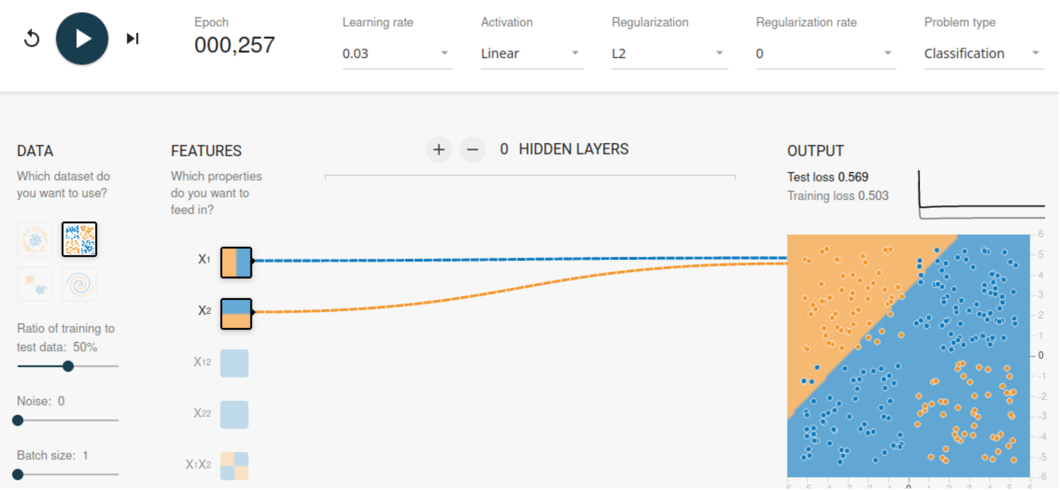


Figure 5: Linear model can tackle only linearly-separable problems (`http://playground.tensorflow.org`)

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

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

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

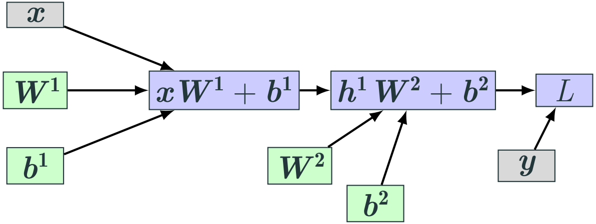


Figure 6: Computational graph; green circles are trainable parameters, gray are constant inputs

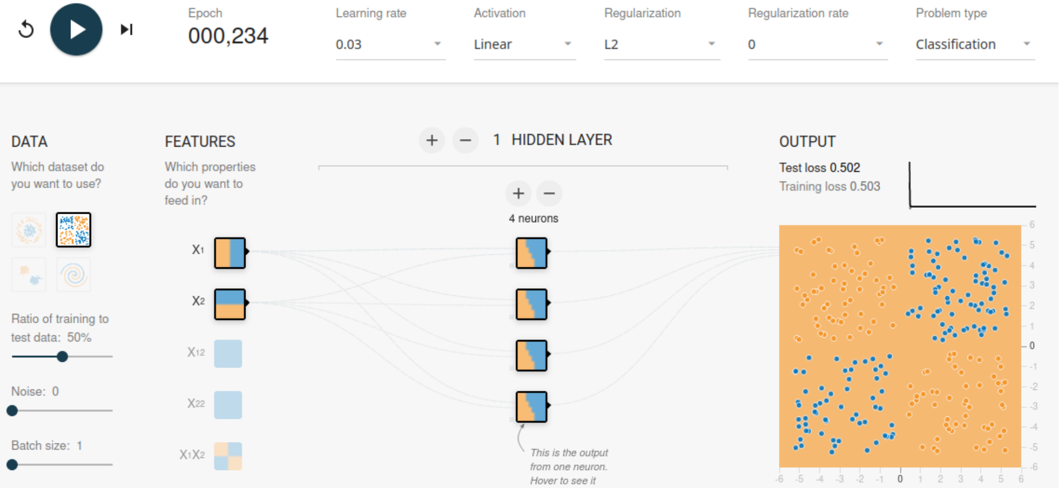


Figure 7: Linear hidden layers do not help (`http://playground.tensorflow.org`)

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

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

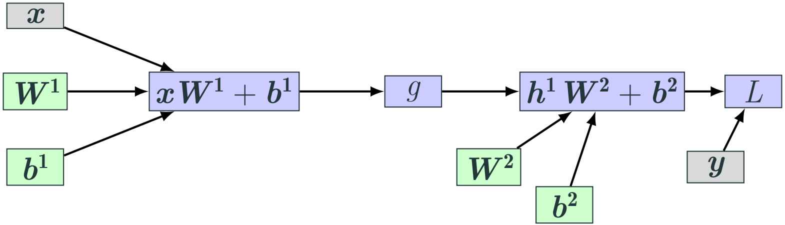


Figure 8: Computational graph; green circles are trainable parameters, gray are constant inputs

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

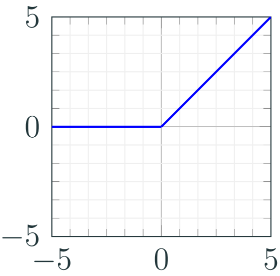


Figure 9: ReLU function

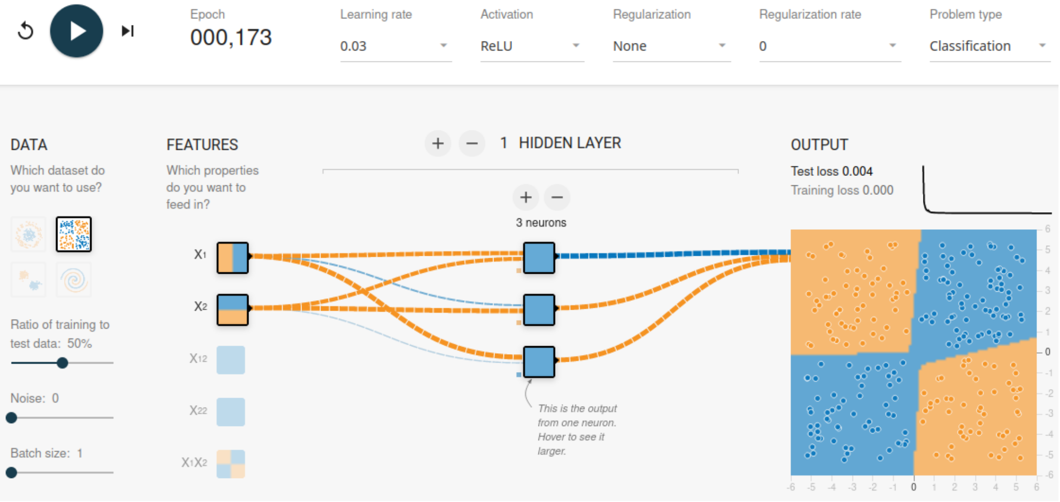


Figure 10: XOR solvable with, e.g., ReLU (`http://playground.tensorflow.org`)

XOR example in super-simplified sentiment classification

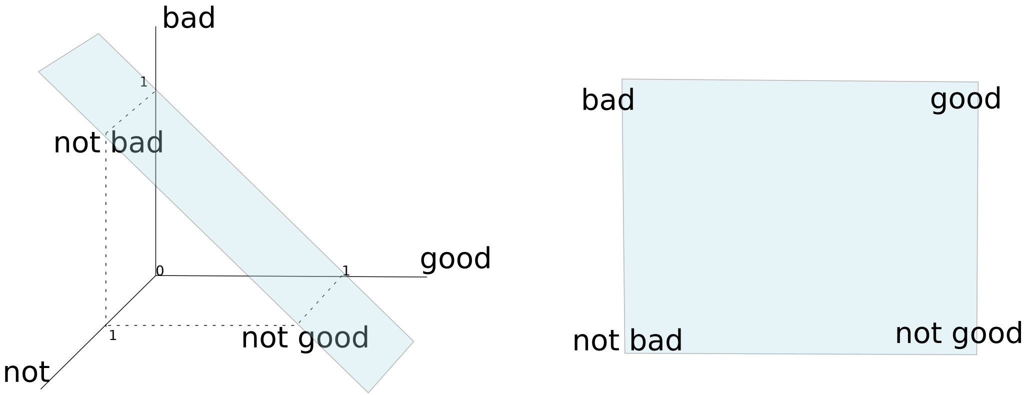


Figure 11: $V = \{\text{not}, \text{bad}, \text{good}\}$, binary features $\in \{0, 1\}$

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

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

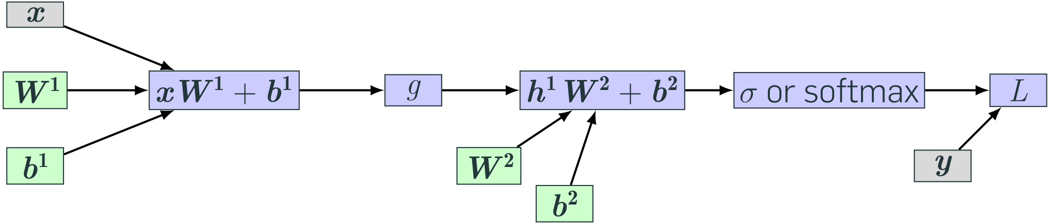


Figure 12: Computational graph; green boxes are trainable parameters, gray are constant inputs

License and credits

Credits

Ivan Habernal