Text Classification and Neural Networks

Giovanni Colavizza

Text Mining
Amsterdam University College

March 10, 2021

Overview

1. Text Classification
2. Logistic Regression
3. Evaluation
4. Neural Networks
5. Extras

Text Classification

Task definition

- We are given a **training set** $\{X, Y\}$ of data pairs (x, y), where x is a text document and y is the class the document belongs to.
- Each $y \in \mathcal{Y}$, where $\mathcal{Y} = \{c_1, c_2, \ldots, c_k\}$ are the distinct (finite and enumerable) classes we have. If $|\mathcal{Y}| = k = 2$, we have a binary classification task.
- Using a *learning method*, our goal is to learn a **classifier**, or a classification function γ that maps documents to classes:

$$\gamma : \mathcal{X} \to \mathcal{Y}$$

- The fact that we use annotated data to learn makes this a form of *supervised learning*. Note that a “document” can be anything really: words, text sequences, longer texts.

Examples

task	x	$\mathcal{Y}$
language ID	text	{english, mandarin, greek, ...}
spam classification	email	{spam, not spam}
authorship attribution	text	{jk rowling, james joyce, ...}
genre classification	novel	{detective, romance, gothic, ...}
sentiment analysis	text	{postive, negative, neutral, mixed}

Credit: David Bamman (UC Berkeley).

Text representation

Our text documents X can be **represented** in many ways:

- Pre-computed features (e.g., the length of the document or the average length of the words it contains).
- A selection of words (e.g., only stopwords for language detection).
- Words in isolation (so called “bag of words”, or unigram model).
- Conjunctions of words (e.g., bigrams).
- Higher-order features (e.g., PoS).
- Word embeddings.

Logistic Regression

Logistic regression

- Our goal is, given a document represented with a feature vector $\boldsymbol{x}$ and classes $c \in \mathcal{Y}$, to learn a classifier discriminating the right class for $\boldsymbol{x}$:

$$\hat{p}(y = c|\boldsymbol{x})$$

- Let us start with a binary classifier and two classes, thus $\mathcal{Y} = \{0, 1\}$.
- We need to estimate $\hat{p}(y = 1|\boldsymbol{x})$, and $\hat{p}(y = 0|\boldsymbol{x}) = 1 - \hat{p}(y = 1|\boldsymbol{x})$ will follow suit.
- Logistic regression uses two components for this: a **linear model** of the inputs and the **Sigmoid (or logistic) function**. So, it is like the perceptron but with a different classification function.

Sigmoid (or logistic) function

- Let us consider the set of features $x_1, x_2, \ldots, x_d$ we used to represent our input document $\boldsymbol{x}$. We add $x_0 = 1$ to model the intercept, and create a linear model with them:

$$z = \sum_{j=0}^{d} w_j x_j = \boldsymbol{w} \cdot \boldsymbol{x}$$

- To create a probability distribution, we pass z through the Sigmoid $\sigma(z)$:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- The Sigmoid squeezes z within 0 and 1 and is always positive.

Sigmoid (or logistic) function

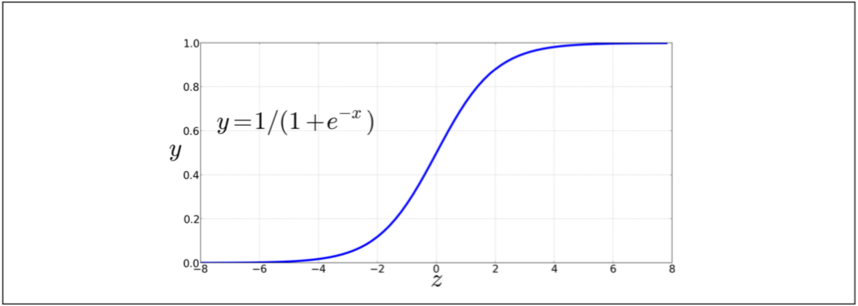


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1.

¿¿

Credit: M&J, Ch. 5.

Logistic regression

- Applied to our binary classification task, we have that:

$$\hat{p}(y=1|\boldsymbol{x}) = \sigma(z_{\boldsymbol{x}}) = \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$$

$$\hat{p}(y=0|\boldsymbol{x}) = 1-\sigma(z_{\boldsymbol{x}}) = \frac{e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$$

- Then, we just need to use a **decision boundary** to assign the class given the estimated probabilities:

$$\hat{y} = \begin{cases} 1 \text{ if } \hat{p}(y=1|\boldsymbol{x}) > 0.5 \\ 0 \text{ otherwise} \end{cases}$$

- **So, we have defined out data and task, and have a model. What do we miss?**

Logistic regression: Cross-entropy

- We need a loss function. Let us use MLE to find one.
- We have that $p(y|\boldsymbol{x})$ follows a Bernoulli distribution given that we only have two discrete outcomes $(0, 1)$, hence:

$$p(y|\boldsymbol{x}) = \hat{y}^y(1-\hat{y})^{1-y}$$

- As usual, let us move to log space and add a minus to switch to a minimization problem (note we work with a single data point $(\boldsymbol{x}, y)$ for now):

$$\begin{aligned} -logp(y|\boldsymbol{x}) &= -log\left[\hat{y}^y(1-\hat{y})^{1-y}\right] \\ &= -\left[ylog\hat{y} + (1-y)log(1-\hat{y})\right] \end{aligned}$$

- Let us now plug-in the Sigmoid and call it the loss:

$$\mathcal{L}_{\boldsymbol{x}}(\boldsymbol{w}) = -\left[ylog\sigma(\boldsymbol{wx}) + (1-y)log(1-\sigma(\boldsymbol{wx}))\right]$$

Logistic regression: Cross-entropy

- Let us now plug-in the Sigmoid and call it the loss:

$$\mathcal{L}_{\boldsymbol{x}}(\boldsymbol{w}) = -\big[y log \sigma(\boldsymbol{wx}) + (1-y) log(1-\sigma(\boldsymbol{wx}))\big]$$

- The loss on the whole dataset is going to be (note we are already in log space thus we can sum):

$$\mathcal{L}(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\big[y_i log \sigma(\boldsymbol{wx}_i) + (1-y_i) log(1-\sigma(\boldsymbol{wx}_i))\big]$$

- To this we can, as usual, attach regularization:

$$\mathcal{L}_{L_2}(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

Logistic regression: Optimization via SGD

- The last missing bit is how to find good parameters $\boldsymbol{w}$: we can use SGD.
- It turns out that the derivative for one data point $\boldsymbol{x}$ is (w.o. regularization):

$$\frac{\partial \mathcal{L}_{\boldsymbol{x}}(\boldsymbol{w})}{\partial \boldsymbol{w}_j} = \big[\sigma(\boldsymbol{w}\boldsymbol{x}) - y\big]\boldsymbol{x}_j$$

- For multiple data points, we just sum (w.o. regularization), and with this we are good to go for SGD:

$$\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}_j} = \sum_{i=1}^{N} \big[\sigma(\boldsymbol{w}\boldsymbol{x}_i) - y_i\big]\boldsymbol{x}_{ij}$$

- *Full derivation as an extra, below.*

Notes

Notes

Evaluation

Data splitting

	training	development	testing
size	80%	10%	10%
purpose	training models	model selection; hyperparameter tuning	evaluation; never look at it until the very end

Credit: David Bamman (UC Berkeley).

Accuracy and baselines

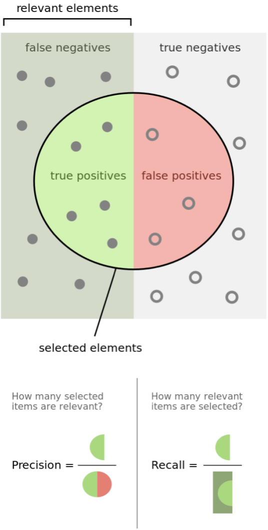
- **Accuracy** is the fraction of correctly predicted data points over the total. It can be calculated on any dataset split: train, development and test. Very good starting point.
- **Baseline**: important to have one. It can be a random classifier (i.e., flip a coin for a binary classifier), or a fast and reasonable model (e.g., logistic regression with TF-IDF features).

Precision and recall

Given a binary classifier:

- **True positive**: a data point correctly predicted to be 1.
- **True negative**: a data point correctly predicted to be 0.
- **False positive**: a data point incorrectly predicted to be 1.
- **False negative**: a data point incorrectly predicted to be 0.

Precision and recall



Credit: Wikipedia.

F-measure and accuracy reloaded

- F-measure (harmonic mean of precision and recall):

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- Accuracy:

$$A = \frac{tp + tn}{tp + tn + fp + fn}$$

Parameters and hyperparameters

Parameters whose values are *learned*

Hyperparameters whose values are *chosen*

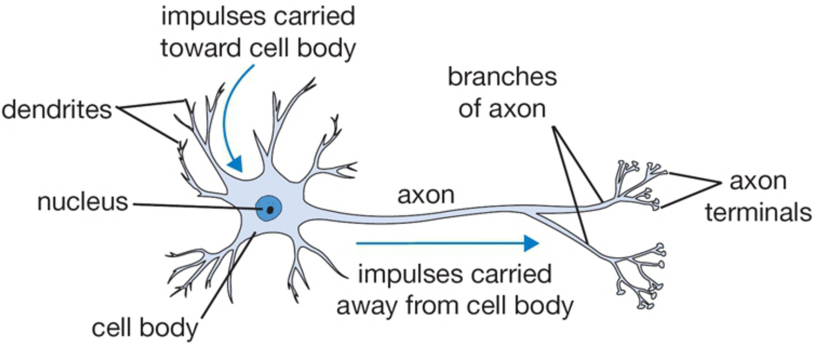
Feature	β
the	0.01
and	0.03
bravest	1.4
love	3.1
loved	1.2
genius	0.5
BIAS	-0.1

Hyperparameter	value
minimum word frequency	5
max vocab size	10000
lowercase	TRUE
regularization strength	1.0

Credit: David Bamman (UC Berkeley).

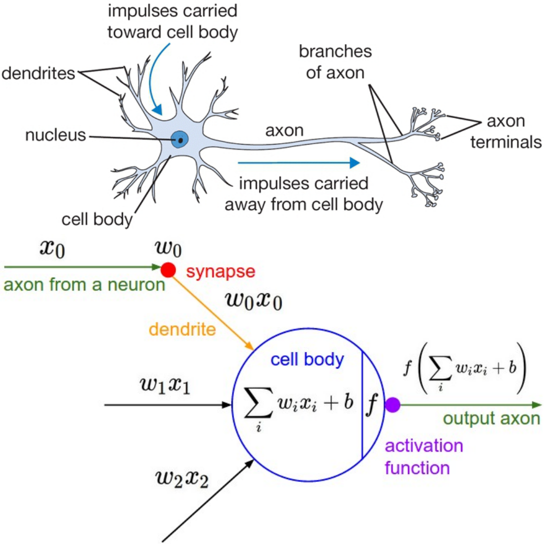
Neural Networks

A single neuron



Credit: Andrej Karpathy via Stanford's CS231N.

A single neuron



Credit: Andrej Karpathy via Stanford's CS231N.

Logistic regression as a neural network

Following the notation in the previous slide, we have:

- $\boldsymbol{x} = \langle x_0, x_1, x_2, \ldots, x_d \rangle$ is our input representation.
- We aggregate the features $\boldsymbol{x}$ into a linear combination using weights $\boldsymbol{w}$. We also include the bias term b into the matrix by adding an appropriate dimension fixed at 1 to $\boldsymbol{x}$, so that we can use matrix notation:

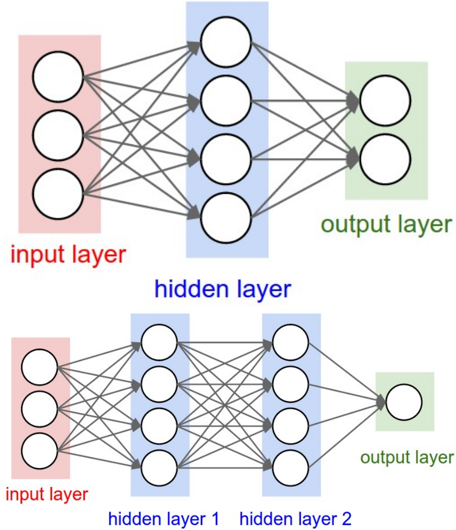
$$z = \sum_{i=0}^{d} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$$

- We pass z through an activation function, in this case the sigmoid:

$$f = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Linear models are a single neuron**. *Question: what is the activation function for linear regression?*

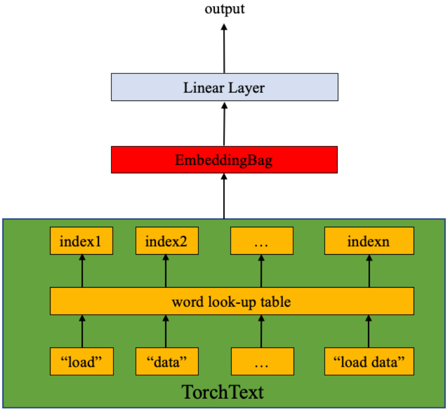
From single layer to multi-layer



Credit: Andrej Karpathy via Stanford's CS231N.

Using embeddings as features

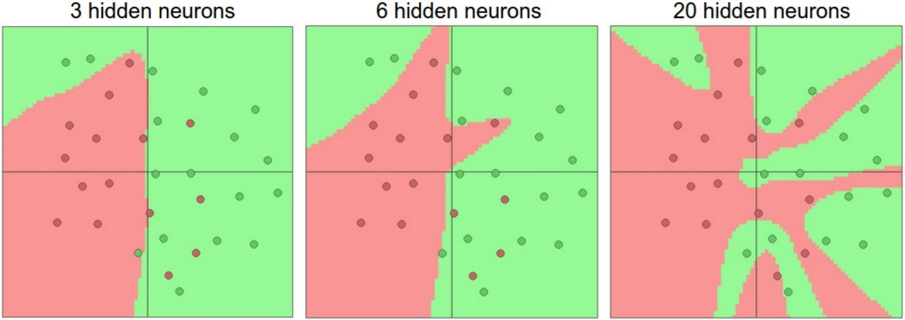
Neural networks are **modular**: we can piece them together into advanced architectures. For example, we can use embeddings to represent our input (either training them or using pre-trained ones). *More on this in the lab.*



Credit: TorchText.

Why do we need non-linearities?

Multiple layers and **non-linear functions** (such as the sigmoid) allow us to fit complex decision boundaries.



Credit: Andrej Karpathy via Stanford's CS231N.

How do we train neural networks?

- Key idea: use a smart way to apply SGD, called **backpropagation**.
- Backpropagation combines using the chain rule to calculate local derivatives (called gradients) with the re-use of pre-computed operations to speed the computation up.
- *More on this in the external materials for the course.*

Neural networks practicalities

Training neural networks entails a lot more than stacking up layers. Several topics require practical and theoretical knowledge beyond this course:

- Weight initialization
- Regularization (e.g., via dropout)
- Which non-linearities to use
- Which loss functions to use
- How to monitor and adjust the learning process (e.g., optimizers and learning rates) to avoid dying neurons and overfitting

Notes

Notes

Extras

Full derivation for logistic regression

- First, we need some notable derivatives:

$$
\begin{aligned}
\frac{\partial log(x)}{\partial x} &= \frac{1}{x} \\
\frac{\partial \sigma(x)}{\partial x} &= \sigma(x)(1-\sigma(x)) \\
\frac{\partial f(g(x))}{\partial x} &= \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} \rightarrow \text{chain rule}
\end{aligned}
$$

Full derivation for logistic regression

- Then:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\boldsymbol{x}}(\boldsymbol{w})}{\partial w_j} &= -\partial\Big[y log \sigma(\boldsymbol{wx}) + (1-y) log(1-\sigma(\boldsymbol{wx}))\Big] \\
&= -\Big[\partial y log \sigma(\boldsymbol{wx}) + \partial(1-y) log(1-\sigma(\boldsymbol{wx}))\Big] \\
&= -\frac{y}{\sigma(\boldsymbol{wx})}\partial\sigma(\boldsymbol{wx}) - \frac{1-y}{1-\sigma(\boldsymbol{wx})}\partial(1-\sigma(\boldsymbol{wx})) \rightarrow \text{chain rule} \\
&= -\Big[\frac{y}{\sigma(\boldsymbol{wx})} - \frac{1-y}{1-\sigma(\boldsymbol{wx})}\Big]\partial\sigma(\boldsymbol{wx}) \rightarrow \text{re-arrange}
\end{aligned}$$

- *Exercise: plug-in the derivative of the Sigmoid and re-arrange yourself to reach:*

$$\ldots = \big[\sigma(\boldsymbol{wx} - y)\big] x_j$$

Full derivation for logistic regression

- In case you were wondering:

$$
\begin{aligned}
\frac{\partial \sigma(x)}{\partial x} &= \partial \frac{1}{1+e^{-x}} \\
&= \partial [1+e^{-x}]^{-1} \\
&= \frac{e^{-x}}{1+e^{-x}} \frac{1}{1+e^{-x}} \\
&= \frac{(1+e^{-x})-1}{1+e^{-x}} \sigma(x) \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}
$$

- *Exercise, derive*:

$$
\frac{\partial log \sigma(x)}{\partial x} = \sigma(-x)
$$

Why MSE and cross-entropy?

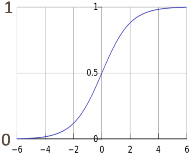
- It turns out that, given some standard assumptions on our models, using those two losses corresponds to doing Maximum Likelihood Estimation. See `https://www.expunctis.com/2019/01/27/Loss-functions.html`.
- If you are curious about the information theory underpinning cross-entropy, read this: `http://colah.github.io/posts/2015-09-Visual-Information`.

NN activation functions

Several non-linear activation functions have been proposed. A good default options is ReLU.

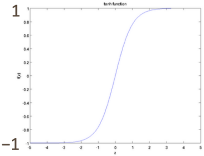
logistic ("sigmoid")

$$f(z) = \frac{1}{1+\exp(-z)}.$$



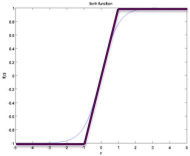
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



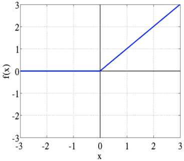
hard tanh

$$\mathrm{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



ReLU (Rectified Linear Unit)

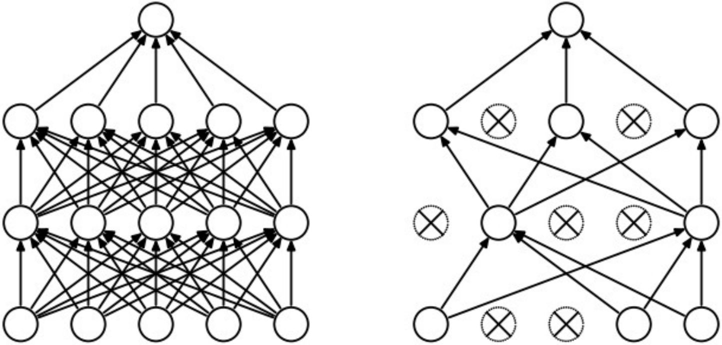
$$\mathrm{rect}(z) = \max(z, 0)$$



Credit: Stanford CS224N.

NN regularization via dropout

Dropout's idea is to mask a random set of neuron connections at training time, in order to compel the network to learn redundant paths and avoid overfitting.

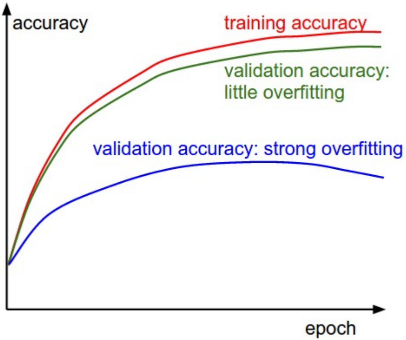


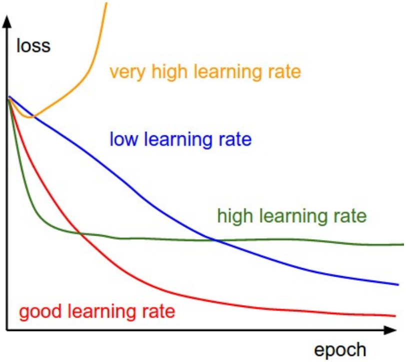
(a) Standard Neural Net (b) After applying dropout.

Credit: Srivastava et al.
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

NN under/overfitting and learning rates

Two illustrations on how to spot correct learning behaviour.





Credit: Andrej Karpathy via Stanford's CS231N.