Derivation of Softmax, k-NN

Milan Straka

November 01, 2021
Where Are We

We have seen the gradual development of machine learning systems to neural networks.

- linear regression → Perceptron → (multinomial) logistic regression → MLP

Figure 1.5, page 10 of Deep Learning Book, http://deeplearningbook.org.
Constrained Optimization

Given a function \( f(\mathbf{x}) \), we can find a minimum/maximum with respect to a vector \( \mathbf{x} \in \mathbb{R}^D \), by investigating the critical points \( \nabla_x f(\mathbf{x}) = 0 \).

Consider now finding a minimum subject to a constraint \( g(\mathbf{x}) = 0 \).

On the left, there is an example with \( f(x, y) = x + y \) and the constraint \( x^2 + y^2 = 1 \), which can be represented as \( g(x, y) = x^2 + y^2 - 1 \).

https://upload.wikimedia.org/wikipedia/commons/e/ed/Lagrange_very_simple.svg
Lagrange Multipliers – Equality Constraints

Let \( f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R} \) be a function. We seek its minimum subject to an equality constraint \( g(\mathbf{x}) = 0 \) for \( g(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R} \).

- Note that \( \nabla_x g(\mathbf{x}) \) is orthogonal to the surface of the constraint, because if \( \mathbf{x} \) and a nearby point \( \mathbf{x} + \mathbf{\varepsilon} \) lie on the surface, from the Taylor expansion \( g(\mathbf{x} + \mathbf{\varepsilon}) \approx g(\mathbf{x}) + \mathbf{\varepsilon}^T \nabla_x g(\mathbf{x}) \) we get \( \mathbf{\varepsilon}^T \nabla_x g(\mathbf{x}) \approx 0 \).

- In the sought minimum, \( \nabla_x f(\mathbf{x}) \) must also be orthogonal to the constraint surface (or else moving in the direction of the derivative would increase the value).

- Therefore, there must exist \( \lambda \) such that \( \nabla_x f = \lambda \nabla_x g \).

Consequently, the sought minimum either fulfills \( \nabla_x f - \lambda \nabla_x g = 0 \) for some \( \lambda \), or it is an unconstrained minimum – in that case, the equation also holds with \( \lambda = 0 \).
Minimization – Equality Constraint

Let \( f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R} \) be a function, which has a minimum (or a maximum) in \( \mathbf{x} \) subject to equality constraint \( g(\mathbf{x}) = 0 \). Assume that both \( f \) and \( g \) have continuous partial derivatives and that \( \frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}) \neq 0 \).

Then there exists a \( \lambda \in \mathbb{R} \), such that the Lagrangian function

\[
\mathcal{L}(\mathbf{x}, \lambda) \overset{\text{def}}{=} f(\mathbf{x}) - \lambda g(\mathbf{x})
\]

has a zero gradient in both \( \mathbf{x} \) and \( \lambda \).

In detail,

- \( \frac{\partial \mathcal{L}}{\partial \lambda} = 0 \) leads to \( g(\mathbf{x}) = 0 \);
- \( \frac{\partial \mathcal{L}}{\partial \mathbf{x}} = 0 \) is the previously derived \( \nabla_\mathbf{x}f - \lambda \nabla_\mathbf{x}g = 0 \).
Minimization – Multiple Equality Constraints

We can use induction if there are multiple equality constraints, resulting in the following generalization.

Let $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}$ be a function, which has a minimum (or a maximum) in $x$ subject to equality constraints $g_1(x) = 0, \ldots, g_m(x) = 0$. Assume that $f, g_1, \ldots, g_m$ have continuous partial derivatives and that the gradients $\nabla_x g_1(x), \ldots, \nabla_x g_m(x)$ are linearly independent.

Then there exist $\lambda_1 \in \mathbb{R}, \ldots, \lambda_m \in \mathbb{R}$, such that the Lagrangian function

$$
L(x, \lambda) \overset{\text{def}}{=} f(x) - \sum_{i=1}^{m} \lambda_i g_i(x)
$$

has a zero gradient in both $x$ and $\lambda$.

This strategy of finding constrained minima is known as the method of Lagrange multipliers.
Assume we want to find a categorical distribution $p_1, \ldots, p_n$ with maximum entropy.

Then we want to minimize $-H(p)$ under the constraints

- $p_i \geq 0$ for all $i$,
- $\sum_{i=1}^{n} p_i = 1$.

Ignoring the first constraint for the time being, we form a Lagrangian

$$\mathcal{L} = \left( \sum_i p_i \log p_i \right) - \lambda \left( \sum_i p_i - 1 \right).$$

Computing the derivative with respect to $p_i$ and setting it equal to zero, we get

$$0 = \frac{\partial \mathcal{L}}{\partial p_i} = 1 \cdot \log(p_i) + p_i \cdot \frac{1}{p_i} - \lambda = \log(p_i) + 1 - \lambda.$$

Therefore, all $p_i = e^{\lambda-1}$ must be the same, and the constraint $\sum_{i=1}^{n} p_i = 1$ yields $p_i = \frac{1}{n}$. 
So far, we minimized a function with respect to a finite number of variables.

A function of a function, \( J[f] \), is known as a **functional**, for example the entropy \( H[\cdot] \).

To minimize a functional with respect to a function, we can turn to the **calculus of variations**.

Consider a functional

\[
J[f] = \int_a^b g(x, f(x)) \, dx,
\]

where \( f(x) \) and \( g(x, y = f(x)) \) are twice continuously differentiable with respect to all arguments.

If \( J \) has a minimum (or a maximum) in function \( f \), then for all \( x \)

\[
\frac{\partial g(x, y = f(x))}{\partial y} = 0.
\]
Function with Maximum Entropy

What distribution over \( \mathbb{R} \) maximizes entropy \( H[p] = -\mathbb{E}_x[\log p(x)] \)?

For continuous values, the entropy is an integral \( H[p] = -\int p(x) \log p(x) \, dx \).

We cannot just maximize \( H \) with respect to a function \( p \), because:

- the result might not be a probability distribution – we need to add a constraint that \( \int p(x) \, dx = 1 \);
- the problem is underspecified because a distribution can be shifted without changing entropy – we add a constraint \( \mathbb{E}[x] = \mu \);
- because entropy increases as variance increases, we ask which distribution with a fixed variance \( \sigma^2 \) has maximum entropy – adding a constraint \( \text{Var}(x) = \sigma^2 \).
Lagrangian $\mathcal{L}(p(x), x, \lambda; \mu, \sigma^2)$ of all the constraints and the entropy function is

$$\mathcal{L} = -H[p] - \lambda_1 \left( \int p(x) \, dx - 1 \right) - \lambda_2 (\mathbb{E}[x] - \mu) - \lambda_3 \left( \text{Var}(x) - \sigma^2 \right).$$

By expanding all definitions to integrals, we get

$$\mathcal{L}(p(x), x, \lambda; \mu, \sigma^2) = \int \left( p(x) \log p(x) - \lambda_1 p(x) - \lambda_2 p(x)x - \lambda_3 p(x)(x - \mu)^2 \right) \, dx$$

$$+ \lambda_1 + \mu \lambda_2 + \sigma^2 \lambda_3.$$

We now set the derivative of the inner of the integral with respect to $p(x)$ equal to zero:

$$0 = \log p(x) + 1 - \lambda_1 - \lambda_2 x - \lambda_3 (x - \mu)^2,$$

obtaining $p(x) = \exp \left( \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 \right).$
We can verify that setting \( \lambda_1 = 1 - \log \sqrt{2\pi\sigma^2} \), \( \lambda_2 = 0 \) and \( \lambda_3 = -1/(2\sigma^2) \) fulfills all the constraints, arriving at

\[
p(x) = \exp \left( \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 \right)
\]

\[
= \exp \left( 1 - \log \sqrt{2\pi\sigma^2} + -1/(2\sigma^2)(x - \mu)^2 - 1 \right)
\]

\[
= \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( - \frac{(x - \mu)^2}{2\sigma^2} \right)
\]

\[= \mathcal{N}(x; \mu, \sigma^2).\]
Derivation of Softmax using Maximum Entropy

Let $X = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}$ be training data of a $K$-class classification, with $x_i \in \mathbb{R}^D$ and $t_i \in \{1, 2, \ldots, K\}$.

We want to model it using a function $\pi : \mathbb{R}^D \to \mathbb{R}^K$ so that $\pi(x)$ gives a distribution of classes for input $x$.

We impose the following conditions on $\pi$:

1. $\forall 1 \leq k \leq K : \pi(x)_k \geq 0$,

2. $\sum_{k=1}^{K} \pi(x)_k = 1$,

3. $\forall 1 \leq k \leq K : \sum_{i=1}^{N} \pi(x_i)_k x_i = \sum_{i=1}^{N} [t_i == k] x_i$. 
Derivation of Softmax using Maximum Entropy

There are many such $\pi$, one particularly bad is

$$
\pi(\boldsymbol{x}) = \begin{cases} 
1_{t_i} & \text{if there exists } i : \boldsymbol{x}_i = \boldsymbol{x}, \\
1_0 & \text{otherwise,}
\end{cases}
$$

where $1_i$ is a one-hot encoding of $i$ (vector of zeros, except for position $i$, which is equal to 1).

Therefore, we want to find a more general $\pi$ – consequently, we turn to the principle of maximum entropy and search for $\pi$ with maximum entropy.
Derivation of Softmax using Maximum Entropy

We want to minimize \(- \sum_{i=1}^{N} H(\pi(x_i))\) given

- \(\forall 1 \leq i \leq N, \forall 1 \leq k \leq K : \pi(x_i)_k \geq 0\),
- \(\forall 1 \leq i \leq N : \sum_{k=1}^{K} \pi(x_i)_k = 1\),
- \(\forall 1 \leq j \leq D, \forall 1 \leq k \leq K : \sum_{i=1}^{N} \pi(x_i)_k x_{i,j} = \sum_{i=1}^{N} [t_i == k] x_{i,j}\).

We therefore form a Lagrangian (ignoring the first inequality constraint):

\[
\mathcal{L} = \sum_{i=1}^{N} \sum_{k=1}^{K} \pi(x_i)_k \log(\pi(x_i)_k) \\
- \sum_{j=1}^{D} \sum_{k=1}^{K} \lambda_{j,k} \left( \sum_{i=1}^{N} \pi(x_i)_k x_{i,j} - [t_i == k] x_{i,j} \right) \\
- \sum_{i=1}^{N} \beta_i \left( \sum_{k=1}^{K} \pi(x_i)_k - 1 \right).
\]
We now compute partial derivatives of the Lagrangian, notably the values

$$\frac{\partial}{\partial \pi(x)_k} \mathcal{L}.$$  

We arrive at

$$\frac{\partial}{\partial \pi(x)_k} \mathcal{L} = \log(\pi(x)_k) + 1 - x_i^T \lambda_{*,k} - \beta_i.$$  

Setting the Lagrangian to zero, we obtain

$$\pi(x)_k = e^{x_i^T \lambda_{*,k} + \beta_i - 1}.$$  

Such a form guarantees $\pi(x)_k > 0$, which we did not include in the conditions.
In order to find out the $\beta_i$ values, we turn to the constraint

$$\sum_k \pi(\mathbf{x}_i)_k = \sum_k e^{\mathbf{x}_i^T \mathbf{\lambda}_{*,k} + \beta_i - 1} = 1,$$

from which we get

$$e^{\beta_i} = \frac{1}{\sum_k e^{\mathbf{x}_i^T \mathbf{\lambda}_{*,k} - 1}},$$

yielding

$$\pi(\mathbf{x}_i)_k = e^{\mathbf{x}_i^T \mathbf{\lambda}_{*,k} + \beta_i - 1} = \frac{e^{\mathbf{x}_i^T \mathbf{\lambda}_{*,k}}}{\sum_{k'} e^{\mathbf{x}_i^T \mathbf{\lambda}_{*,k'}}} = \text{softmax}(\mathbf{x}_i \mathbf{\lambda})_k.$$
When evaluating binary classification, we have used accuracy so far. However, there are other metrics we might want to consider. One of them is $F_1$-score.

Consider the following confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>Target positive</th>
<th>Target negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted positive</td>
<td><strong>True Positive (TP)</strong></td>
<td><strong>False Positive (FP)</strong></td>
</tr>
<tr>
<td>Predicted negative</td>
<td><strong>False Negative (FN)</strong></td>
<td><strong>True Negative (TN)</strong></td>
</tr>
</tbody>
</table>

Accuracy can be computed as

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$
In some cases, we are mostly interested in positive examples.

We define **precision** (percentage of correct predictions in predicted examples) and **recall** (percentage of correct predictions in the gold examples) as

\[
\text{precision} = \frac{TP}{TP + FP},
\]

\[
\text{recall} = \frac{TP}{TP + FN}.
\]

### F1-score

<table>
<thead>
<tr>
<th></th>
<th>Target positive</th>
<th>Target negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted positive</td>
<td><strong>True Positive (TP)</strong></td>
<td><strong>False Positive (FP)</strong></td>
</tr>
<tr>
<td>Predicted negative</td>
<td><strong>False Negative (FN)</strong></td>
<td><strong>True Negative (TN)</strong></td>
</tr>
</tbody>
</table>

![Precision recall diagram](https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg)

**Precision** = \(\frac{TP}{TP + FP}\)

**Recall** = \(\frac{TP}{TP + FN}\)
The precision and recall go “against each other”: decreasing the classifier threshold usually increases recall and decreases precision, and vice versa.

We therefore define a single $F_1$-score as a harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + FP + TP + FN}.$$
Arithmetic mean of precision recall is

\[ AM(p, r) \stackrel{\text{def}}{=} \frac{p + r}{2}. \]

As any mean, it is "between" the input values

\[ \min(p, r) \leq AM(p, r), \]
\[ AM(p, r) \leq \max(p, r). \]

However,

\[ AM(1\%, 100\%) = 50.5\%. \]
Geometric mean of precision recall is

$$GM(p, r) \overset{\text{def}}{=} \sqrt{p \cdot r}.$$  

It is better than arithmetic mean, but still

$$GM(1\%, 100\%) = 10\%.$$
Harmonic mean of precision recall is

$$HM(p, r) \overset{\text{def}}{=} \frac{2}{\frac{1}{p} + \frac{1}{r}}.$$ 

In addition to being bounded by the input values, it is also dominated by the minimum of its input values:

$$\min(p, r) \leq HM(p, r),$$

$$HM(p, r) \leq \max(p, r),$$

$$HM(p, r) \leq 2 \min(p, r).$$

For example,

$$HM(1\%, 100\%) \approx 2\%.$$
The $F_1$ score can be generalized to $F_\beta$ score, which can be used as a metric when recall is $\beta$ times more important than precision; $F_2$ favoring recall and $F_{0.5}$ favoring precision are commonly used.

The formula for $F_\beta$ is

\[
F_\beta = \frac{1 + \beta^2}{\text{precision}^{-1} + \beta^2 \cdot \text{recall}^{-1}} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} = \frac{TP + \beta^2 \cdot TP}{TP + FP + \beta^2 \cdot (TP + FN)}.
\]
General $F_\beta$-score

You may wonder why $\beta^2$ is used in the formula

$$F_\beta = \frac{1 + \beta^2}{\text{precision}^{-1} + \beta^2 \cdot \text{recall}^{-1}}$$

instead of just $\beta$.

Quoting C. J. van Rijsbergen from his book *Information Retrieval*, 1979:

*What we want is therefore a parameter $\beta$ to characterise the measurement function in such a way that we can say: it measures the effectiveness of retrieval with respect to a user who attaches $\beta$ times as much importance to recall as precision. The simplest way I know of quantifying this is to specify the recall/precision ratio at which the user is willing to trade an increment in precision for an equal loss in recall.*

It is straightforward to verify that indeed $\frac{\partial F_\beta}{\partial \text{precision}} = \frac{\partial F_\beta}{\partial \text{recall}}$ implies $\frac{\text{recall}}{\text{precision}} = \beta$. 
$F_1$-score and Other Means of Precision and Recall

- Harmonic mean, $\beta = 1.0$
- Harmonic mean, $\beta = 2.0$
- Harmonic mean, $\beta = 0.5$
Changing the threshold in logistic regression allows us to trade off precision for recall, and vice versa. Therefore, we can tune it on the development set to achieve the highest possible $F_1$ score, if required.

Also, if we want to evaluate $F_1$-score without considering a specific threshold, the area under curve (AUC) is sometimes used as a metric.
To extend $F_1$-score to multiclass classification, we expect one of the classes to be *negative* and the others *different kinds of positive*. For each of the positive classes, we compute the same confusion matrix as in the binary case (considering all other labels as negative ones), and then combine the results in one of the following ways:

- **micro-averaged $F_1$ (or just micro $F_1$):** we first sum all the TP, FP and FN of the individual binary classifications and compute the final $F_1$-score (this way, the frequency of the individual classes is taken into account);
- **macro-averaged $F_1$ (or just macro $F_1$):** we first compute the $F_1$-scores of the individual binary classifications and then compute an unweighted average (therefore, the frequency of the classes is more or less ignored).
The precision-recall curve is useful when we are interested in the positive examples (i.e., we are ignoring true negative instances). In case we want to consider also the true negatives, we might instead use the **Receiver Operating Characteristic (ROC)** curve.

In the ROC curve, we consider two measures of a binary classifier under changing threshold:

- **true positive rate** or **sensitivity** (probability of detection): \[ \frac{TP}{TP+FN}; \]
- **false positive rate** or **1-specificity** (probability of false alarm): \[ \frac{FP}{FP+TN}; \]

![ROC Curve](https://modtools.files.wordpress.com/2020/01/roc_pr-1.png)

![Precision-recall curve](https://upload.wikimedia.org/wikipedia/commons/4/4f/ROC_curves.svg)
# Binary Confusion Metric Measures Overview

<table>
<thead>
<tr>
<th>Target positive</th>
<th>Target negative</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Predicted positive</strong></td>
<td><strong>Predicted negative</strong></td>
</tr>
<tr>
<td><strong>True Positive (TP)</strong></td>
<td><strong>False Positive (FP)</strong></td>
</tr>
<tr>
<td><strong>False Negative (FN)</strong></td>
<td><strong>True Negative (TN)</strong></td>
</tr>
</tbody>
</table>

- **Type I Error** (False Positive)
- **Type II Error** (False Negative)
- **Precision** = \( \frac{TP}{TP + FP} \)
- **Recall**, **Sensitivity** = \( \frac{TP}{TP + FN} \)
- **Specificity** = \( \frac{TN}{TN + FP} \)
- **Accuracy** = \( \frac{TP + TN}{TP + FP + FN + TN} \)
- **F\(_1\)-score** = \( \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + FN} + \frac{TP}{TP + FP} \)

### Examples:

\[
\begin{align*}
F\(_1\)-score & = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + FN} + \frac{TP}{TP + FP} \\
\text{accuracy} & = \frac{TP + TN}{TP + FP + FN + TN}
\end{align*}
\]
All machine learning models which we have discussed so far are parametric, because they use a fixed number of parameters (usually depending on the number of features, $K$ for multiclass classification, hidden layer in MLPs, ...).

However, there also exist nonparametric models. Even if the name seems to suggest they do not have any parameters, they have a non-fixed number of parameters, because the number of parameters usually depend on the size of the training data – therefore, the model size usually grows with the size of the training data.
A simple but sometimes effective nonparametric method for both classification and regression is **k-nearest neighbors** algorithm.

The training phase of the $k$-nearest neighbors algorithm is trivial, it consists of only storing the whole train set (the so-called **lazy learning**).

For a given test example, the main idea is to use the targets of the most similar training data to perform the prediction.

[Image: https://upload.wikimedia.org/wikipedia/commons/e/e7/KnnClassification.svg]
k-Nearest Neighbors

Several hyperparameters influence the behavior of the prediction phase:

- **k**: consider \( k \) most similar training examples (higher \( k \) usually decreases variance, but increases bias);
- **metric**: a function used to find the nearest neighbors; common choices are metrics based on \( L_p \) norms (with usual values of \( p \) being 1, 2, 3, \( \infty \)). For \( \mathbf{x}, \mathbf{y} \in \mathbb{R}^D \), the distance is measured as \( \| \mathbf{x} - \mathbf{y} \|_p \), where

\[
\| \mathbf{x} \|_p = \left( \sum_i |x_i|^p \right)^{1/p};
\]

- **weights**: optionally, more similar examples can be considered with bigger weights:
  - **uniform**: all \( k \) nearest neighbors are considered equally;
  - **inverse**: the weight of an example is proportional to the inverse of distance;
  - **softmax**: the weights are proportional to \( \text{softmax} \) of negative distances.
**k-Nearest Neighbors**

**Regression**
To perform regression when $k$ nearest neighbors have values $t_i$ and weights $w_i$, we predict

$$t = \sum_i \frac{w_i}{\sum_j w_j} \cdot t_i.$$

**Classification**
For uniform weights, we can use voting during prediction – the most frequent class is predicted (with ties broken arbitrarily).

Otherwise, we weight the categorical distributions $t_i \in \mathbb{R}^K$ (with the training target classes represented using one-hot encoding), predicting a distribution

$$t = \sum_i \frac{w_i}{\sum_j w_j} \cdot t_i.$$

The predicted class is then the one with the largest probability, i.e., $\arg \max_k \sum_i w_i t_{i,k}$. 
A trivial implementation of the $k$-nearest neighbors algorithm is extremely demanding during the inference, requiring to measure distances of a given example to all training instances. However, several data structures capable of speeding up the $k$-nearest neighbor search exist, like

- $k$-$d$ trees, which allow both a static or dynamic construction and can perform nearest neighbor queries of uniformly random points in logarithmic time on average, but which become inefficient for high-dimensional data;
- ball trees, R-trees, …