Softmax function

The **softmax function,** also known as **softargmax**[1]:184 or **normalized exponential function**,[2]:198 is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes, based on Luce's choice axiom.

The softmax function takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval $[0, 1]$, and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

The standard (unit) softmax function $\sigma : \mathbb{R}^K \to [0, 1]^K$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K.$$

In simple words, it applies the standard exponential function to each element z_i of the input vector $\mathbf{z}$ and normalizes these values by dividing by the sum of all these exponentials; this normalization ensures that the sum of the components of the output vector $\sigma(\mathbf{z})$ is 1.

Instead of e, a different base $b > 0$ can be used. If $0 < b < 1$, smaller input components will result in larger output probabilities, and decreasing the value of b will create probability distributions that are more concentrated around the positions of the smallest input values. Conversely, if $b > 1$, larger input components will result in larger output probabilities, and increasing the value of b will create probability distributions that are more concentrated around the positions of the largest input values. Writing $b = e^{\beta}$ or $b = e^{-\beta}$[a] (for real β)[b] yields the expressions:[c]

$$\sigma(\mathbf{z})_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}} \text{ or } \sigma(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}} \text{ for } i = 1, \dots, K.$$

In some fields, the base is fixed, corresponding to a fixed scale,[d] while in others the parameter β is varied.

Contents

Interpretations

Smooth arg max

The name "softmax" is misleading; the function is not a smooth maximum (a smooth approximation to the maximum function), but is rather a smooth approximation to the arg max function: the function whose value is *which index* has the maximum. In fact, the term "softmax" is also used for the closely related LogSumExp function, which is a smooth maximum. For this reason, some prefer the more accurate term "softargmax", but the term "softmax" is conventional in machine learning.[3][4] This section uses the term "softargmax" to emphasize this interpretation.

Formally, instead of considering the arg max as a function with categorical output $1, \dots, n$ (corresponding to the index), consider the arg max function with one-hot representation of the output (assuming there is a unique maximum arg):

$$\arg\max(z_1, \dots, z_n) = (y_1, \dots, y_n) = (0, \dots, 0, 1, 0, \dots, 0),$$

where the output coordinate $y_i = 1$ if and only if i is the arg max of $(z_1, \dots, z_n)$, meaning z_i is the unique maximum value of $(z_1, \dots, z_n)$. For example, in this encoding $\arg\max(1, 5, 10) = (0, 0, 1)$, since the third argument is the maximum.

This can be generalized to multiple arg max values (multiple equal z_i being the maximum) by dividing the 1 between all max args; formally $1/k$ where k is the number of arguments assuming the maximum. For example, $\arg\max(1, 5, 5) = (0, 1/2, 1/2)$, since the second and third argument are both the maximum. In case all arguments are equal, this is simply $\arg\max(z, \dots, z) = (1/n, \dots, 1/n)$. Points z with multiple arg max values are singular points (or singularities, and form the singular set) – these are the points where arg max is discontinuous (with a jump discontinuity) – while points with a single arg max are known as non-singular or regular points.

With the last expression given in the introduction, softargmax is now a smooth approximation of arg max: as $\beta \to \infty$, softargmax converges to arg max. There are various notions of convergence of a function; softargmax converges to arg max pointwise, meaning for each fixed input $\mathbf{z}$ as $\beta \to \infty$, $\sigma_\beta(\mathbf{z}) \to \arg\max(\mathbf{z})$. However, softargmax does not converge uniformly to arg max, meaning intuitively that different points converge at different rates, and may converge arbitrarily slowly. In fact, softargmax is continuous, but arg max is not continuous at the singular set where two coordinates are equal, while the uniform limit of continuous functions is continuous. The reason it fails to converge uniformly is that for inputs where two coordinates are almost equal (and one is the maximum), the arg max is the index of one or the other, so a small change in input yields a large change in output. For example, $\sigma_\beta(1, 1.0001) \to (0, 1)$, but $\sigma_\beta(1, 0.9999) \to (1, 0)$, and $\sigma_\beta(1, 1) = 1/2$ for all inputs: the closer the points are to the singular set (x, x), the slower they converge. However, softargmax does converge compactly on the non-singular set.

Conversely, as $\beta \to -\infty$, softargmax converges to arg min in the same way, where here the singular set is points with two arg *min* values. In the language of tropical analysis, the softmax is a deformation or "quantization" of arg max and arg min, corresponding to using the log semiring instead of the max-plus semiring (respectively min-plus semiring), and recovering the arg max or arg min by taking the limit is called "tropicalization" or "dequantization".

It is also the case that, for any fixed β, if one input z_i is much larger than the others *relative* to the temperature, $T = 1/\beta$, the output is approximately the arg max. For example, a difference of 10 is large relative to a temperature of 1:

$$\sigma(0, 10) := \sigma_1(0, 10) = \left(1/(1+e^{10}), e^{10}/(1+e^{10})\right) \approx (0.00005, 0.99995)$$

However, if the difference is small relative to the temperature, the value is not close to the arg max. For example, a difference of 10 is small relative to a temperature of 100:

$$\sigma_{1/100}(0, 10) = \left(1/(1+e^{1/10}), e^{1/10}/(1+e^{1/10})\right) \approx (0.475, 0.525).$$

As $\beta \to \infty$, temperature goes to zero, $T = 1/\beta \to 0$, so eventually all differences become large (relative to a shrinking temperature), which gives another interpretation for the limit behavior.

Probability theory

In probability theory, the output of the softargmax function can be used to represent a categorical distribution – that is, a probability distribution over K different possible outcomes.

Statistical mechanics

In statistical mechanics, the softargmax function is known as the Boltzmann distribution (or Gibbs distribution):[5]:7 the index set $1, \ldots, k$ are the microstates of the system; the inputs z_i are the energies of that state; the denominator is known as the partition function, often denoted by Z; and the factor β is called the coldness (or thermodynamic beta, or inverse temperature).

Applications

The softmax function is used in various multiclass classification methods, such as multinomial logistic regression (also known as softmax regression)[2]:206–209 [1] (http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/), multiclass linear discriminant analysis, naive Bayes classifiers, and artificial neural networks.[6] Specifically, in multinomial logistic regression and linear discriminant analysis, the input to the function is the result of K distinct linear functions, and the predicted probability for the j'th class given a sample vector $\mathbf{x}$ and a weighting vector $\mathbf{w}$ is:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

This can be seen as the composition of K linear functions $\mathbf{x} \mapsto \mathbf{x}^\mathsf{T}\mathbf{w}_1, \ldots, \mathbf{x} \mapsto \mathbf{x}^\mathsf{T}\mathbf{w}_K$ and the softmax function (where $\mathbf{x}^\mathsf{T}\mathbf{w}$ denotes the inner product of $\mathbf{x}$ and $\mathbf{w}$). The operation is equivalent to applying a linear operator defined by $\mathbf{w}$ to vectors $\mathbf{x}$, thus transforming the original, probably highly-dimensional, input to vectors in a K-dimensional space $\mathbb{R}^K$.

Neural networks

The softmax function is often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

Since the function maps a vector and a specific index i to a real value, the derivative needs to take the index into account:

$$\frac{\partial}{\partial q_k}\sigma(\mathbf{q}, i) = \sigma(\mathbf{q}, i)(\delta_{ik} - \sigma(\mathbf{q}, k)).$$

This expression is symmetrical in the indexes i, k and thus may also be expressed as

$$\frac{\partial}{\partial q_k}\sigma(\mathbf{q}, i) = \sigma(\mathbf{q}, k)(\delta_{ik} - \sigma(\mathbf{q}, i)).$$

Here, the Kronecker delta is used for simplicity (cf. the derivative of a sigmoid function, being expressed via the function itself).

If the function is scaled with the parameter β, then these expressions must be multiplied by β.

See Multinomial logit for a probability model which uses the softmax activation function.

Reinforcement learning

In the field of reinforcement learning, a softmax function can be used to convert values into action probabilities. The function commonly used is:[7]

$$P_t(a) = \frac{\exp(q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(q_t(i)/\tau)},$$

where the action value $q_t(a)$ corresponds to the expected reward of following action a and τ is called a temperature parameter (in allusion to statistical mechanics). For high temperatures ($\tau \to \infty$), all actions have nearly the same probability and the lower the temperature, the more expected rewards affect the probability. For a low temperature ($\tau \to 0^+$), the probability of the action with the highest expected reward tends to 1.

Properties

Geometrically the softmax function maps the vector space $\mathbb{R}^K$ to the boundary of the standard $(K-1)$-simplex, cutting the dimension by one (the range is a $(K-1)$-dimensional simplex in K-dimensional space), due to the linear constraint that all output sum to 1 meaning it lies on a hyperplane.

Along the main diagonal $(x, x, \dots, x)$, softmax is just the uniform distribution on outputs, $(1/n, \dots, 1/n)$: equal scores yield equal probabilities.

More generally, softmax is invariant under translation by the same value in each coordinate: adding $\mathbf{c} = (c, \dots, c)$ to the inputs $\mathbf{z}$ yields $\sigma(\mathbf{z} + \mathbf{c}) = \sigma(\mathbf{z})$, because it multiplies each exponent by the same factor, e^c (because $e^{z_i + c} = e^{z_i} \cdot e^c$), so the ratios do not change:

$$\sigma(\mathbf{z}+\mathbf{c})_j = \frac{e^{z_j+c}}{\sum_{k=1}^{K} e^{z_k+c}} = \frac{e^{z_j}\cdot e^{c}}{\sum_{k=1}^{K} e^{z_k}\cdot e^{c}} = \sigma(\mathbf{z})_j.$$

Geometrically, softmax is constant along diagonals: this is the dimension that is eliminated, and corresponds to the softmax output being independent of a translation in the input scores (a choice of 0 score). One can normalize input scores by assuming that the sum is zero (subtract the average: $\mathbf{c}$ where $c = \frac{1}{n}\sum z_i$), and then the softmax takes the hyperplane of points that sum to zero, $\sum z_i = 0$, to the open simplex of positive values that sum to 1$\sum \sigma(\mathbf{z})_i = 1$, analogously to how the exponent takes 0 to 1, $e^0 = 1$ and is positive.

By contrast, softmax is not invariant under scaling. For instance, $\sigma\big((0,1)\big) = \big(1/(1+e), e/(1+e)\big)$ but $\sigma\big((0,2)\big) = \big(1/(1+e^2), e^2/(1+e^2)\big)$.

The standard logistic function is the special case for a 1-dimensional axis in 2-dimensional space, say the *x*-axis in the $(\mathrm{x},\ \mathrm{y})$ plane. One variable is fixed at 0 (say $z_2 = 0$), so $e^0 = 1$, and the other variable can vary, denote it $z_1 = x$, so $e^{z_1}/\sum_{k=1}^{2} e^{z_k} = e^x/(e^x+1)$, the standard logistic function, and $e^{z_2}/\sum_{k=1}^{2} e^{z_k} = 1/(e^x+1)$, its complement (meaning they add up to 1). The 1-dimensional input could alternatively be expressed as the line $(x/2, -x/2)$, with outputs $e^{x/2}/(e^{x/2}+e^{-x/2}) = e^x/(e^x+1)$ and $e^{-x/2}/(e^{x/2}+e^{-x/2}) = 1/(e^x+1)$.

The softmax function is also the gradient of the LogSumExp function, a smooth maximum:

$$\frac{\partial}{\partial z_i}\operatorname{LSE}(\mathbf{z}) = \frac{\exp z_i}{\sum_{j=1}^{K}\exp z_j} = \sigma(\mathbf{z})_i, \quad \text{for } i = 1,\dots,K, \quad \mathbf{z} = (z_1,\dots,z_K) \in \mathbb{R}^K,$$

where the LogSumExp function is defined as $\operatorname{LSE}(z_1,\dots,z_n) = \log(\exp(z_1)+\cdots+\exp(z_n))$.

History

The softmax function was used in statistical mechanics as the Boltzmann distribution in the foundational paper Boltzmann (1868),[8] formalized and popularized in the influential textbook Gibbs (1902).[9]

The use of the softmax in decision theory is credited to Luce (1959),[10]:1 who used the axiom of independence of irrelevant alternatives in rational choice theory to deduce the softmax in Luce's choice axiom for relative preferences.

In machine learning, the term "softmax" is credited to John S. Bridle in two 1989 conference papers, Bridle (1990a):[10]:1 and Bridle (1990b):[3]

> We are concerned with feed-forward non-linear networks (multi-layer perceptrons, or MLPs) with multiple outputs. We wish to treat the outputs of the network as probabilities of alternatives (*e.g.* pattern classes), conditioned on the inputs. We look for appropriate output non-linearities and for appropriate criteria for adaptation of the parameters of the network (*e.g.* weights). We explain two modifications: probability scoring, which is an alternative to squared error minimisation, and a normalised exponential (**softmax**) multi-input generalisation of the logistic non-linearity.[11]:227

> For any input, the outputs must all be positive and they must sum to unity. ...

Given a set of unconstrained values, $V_j(x)$, we can ensure both conditions by using a Normalised Exponential transformation:

$$Q_j(x) = e^{V_j(x)} \Big/ \sum_k e^{V_k(x)}$$

This transformation can be considered a multi-input generalisation of the logistic, operating on the whole output layer. It preserves the rank order of its input values, and is a differentiable generalisation of the 'winner-take-all' operation of picking the maximum value. For this reason we like to refer to it as **softmax**.[12]:213

Example

If we take an input of [1, 2, 3, 4, 1, 2, 3], the softmax of that is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]. The output has most of its weight where the '4' was in the original input. This is what the function is normally used for: to highlight the largest values and suppress values which are significantly below the maximum value. But note: softmax is not scale invariant, so if the input were [0.1, 0.2, 0.3, 0.4, 0.1, 0.2, 0.3] (which sums to 1.6) the softmax would be [0.125, 0.138, 0.153, 0.169, 0.125, 0.138, 0.153]. This shows that for values between 0 and 1 softmax, in fact, de-emphasizes the maximum value (note that 0.169 is not only less than 0.475, it is also less than the initial proportion of 0.4/1.6=0.25).

Computation of this example using Python code:

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813, 0.474833, 0.02364054,
       0.06426166, 0.1746813])
```

Here is an example of Julia code:

```
julia> A = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0];  # semicolon to suppress interactive output

julia> exp.(A) ./ sum(exp.(A))
7-element Array{Float64,1}:
 0.0236405
 0.0642617
 0.174681
 0.474833
 0.0236405
 0.0642617
 0.174681
```

Here is an example of R code:

```
> z <- c(1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0)
> softmax <- exp(z)/sum(exp(z))
> softmax
[1] 0.02364054 0.06426166 0.17468130 0.47483300 0.02364054 0.06426166 0.17468130
```

Here is an example of Elixir code:[13]

```
iex> t = Nx.tensor([[1, 2], [3, 4]])
iex> Nx.divide(Nx.exp(t), Nx.sum(Nx.exp(t)))
```

```
#Nx.Tensor<
  f64[2][2]
  [
    [0.03205860328008499, 0.08714431874203257],
    [0.23688281808991013, 0.6439142598879722]
  ]
>
```

See also

- Softplus
- Multinomial logistic regression
- Dirichlet distribution – an alternative way to sample categorical distributions
- Partition function

Notes

a. Positive β corresponds to the maximum convention, and is usual in machine learning, corresponding to the highest score having highest probability. The negative $-\beta$ corresponds to the minimum convention, and is conventional in thermodynamics, corresponding to the lowest energy state having the highest probability; this matches the convention in the Gibbs distribution, interpreting β as coldness.

b. The notation β is for the thermodynamic beta, which is inverse temperature: $\beta = 1/T$, $T = 1/\beta$.

c. For $\beta = 0$ (coldness zero, infinite temperature), $\boldsymbol{b} = e^{\beta} = e^0 = \mathbf{1}$, and this becomes the constant function $(1/n, \dots, 1/n)$, corresponding to the discrete uniform distribution.

d. In statistical mechanics, fixing β is interpreted as having coldness and temperature of 1.

References

1. Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.2.2.3 Softmax Units for Multinoulli Output Distributions" (https://www.deeplearningbook.org/contents/mlp.html). *Deep Learning* (http://www.deeplearningbook.org). MIT Press. pp. 180–184. ISBN 978-0-26203561-3.
2. Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer. ISBN 0-387-31073-8.
3. Sako, Yusaku (2018-06-02). "Is the term "softmax" driving you nuts?" (https://medium.com/@u39kun/is-the-term-softmax-driving-you-nuts-ee232ab4f6bd). Medium.
4. Goodfellow, Bengio & Courville 2016, pp. 183–184: The name "softmax" can be somewhat confusing. The function is more closely related to the arg max function than the max function. The term "soft" derives from the fact that the softmax function is continuous and differentiable. The arg max function, with its result represented as a one-hot vector, is not continuous or differentiable. The softmax function thus provides a "softened" version of the arg max. The corresponding soft version of the maximum function is $\mathbf{softmax}(\mathbf{z})^\top \mathbf{z}$. It would perhaps be better to call the softmax function "softargmax," but the current name is an entrenched convention.
5. LeCun, Yann; Chopra, Sumit; Hadsell, Raia; Ranzato, Marc'Aurelio; Huang, Fu Jie (2006). "A Tutorial on Energy-Based Learning" (http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf) (PDF). In Gökhan Bakır; Thomas Hofmann; Bernhard Schölkopf; Alexander J. Smola; Ben Taskar; S.V.N Vishwanathan (eds.). *Predicting Structured Data* (https://mitpress.mit.edu/books/predicting-structured-data). Neural Information Processing series. MIT Press. ISBN 978-0-26202617-8.

6. ai-faq What is a softmax activation function? (http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html)
7. Sutton, R. S. and Barto A. G. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998. Softmax Action Selection (http://incompleteideas.net/book/ebook/node17.html)
8. Boltzmann, Ludwig (1868). "Studien über das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten" [Studies on the balance of living force between moving material points]. *Wiener Berichte*. **58**: 517–560.
9. Gibbs, Josiah Willard (1902). *Elementary Principles in Statistical Mechanics*.
10. Gao, Bolin; Pavel, Lacra (2017). "On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning". arXiv:1704.00805 (https://arxiv.org/abs/1704.00805) [math.OC (https://arxiv.org/archive/math.OC)].
11. Bridle, John S. (1990a). Soulié F.F.; Hérault J. (eds.). *Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition*. Neurocomputing: Algorithms, Architectures and Applications (1989). NATO ASI Series (Series F: Computer and Systems Sciences). **68**. Berlin, Heidelberg: Springer. pp. 227–236. doi:10.1007/978-3-642-76153-9_28 (https://doi.org/10.1007%2F978-3-642-76153-9_28).
12. Bridle, John S. (1990b). D. S. Touretzky (ed.). *Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters* (https://papers.nips.cc/paper/195-training-stochastic-model-recognition-algorithms-as-networks-can-lead-to-maximum-mutual-information-estimation-of-parameters). Advances in Neural Information Processing Systems 2 (1989). Morgan-Kaufmann.
13. https://github.com/elixir-nx/nx/tree/main/nx#examples
