Multiclass Logistic Regression, Multilayer Perceptron

Milan Straka

October 24, 2022

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

BY SA
unless otherwise stated

Logistic Regression

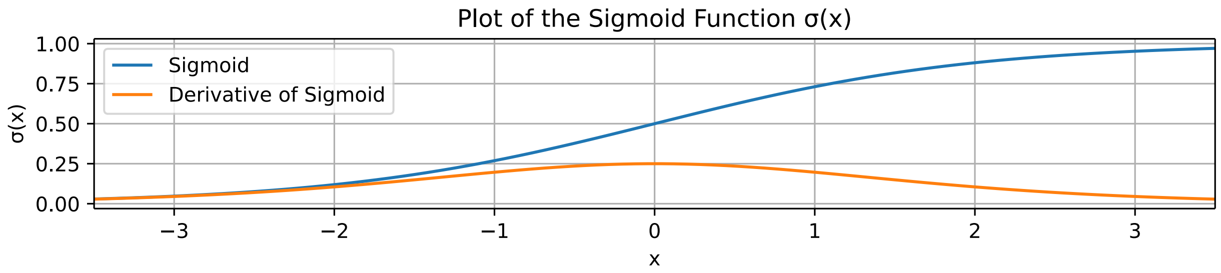
An extension of perceptron, which models the conditional probabilities of $p(C_0|\boldsymbol{x})$ and of $p(C_1|\boldsymbol{x})$. Logistic regression can in fact handle also more than two classes, which we will see shortly.

Logistic regression employs the following parametrization of the conditional class probabilities:

$$p(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{w} + b)$$
$$p(C_0|\boldsymbol{x}) = 1 - p(C_1|\boldsymbol{x}),$$

where σ is a **sigmoid function**

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



It can be trained using the SGD algorithm.

Logistic Regression

We denote the output of the "linear part" of the logistic regression as $\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$ and the overall prediction as $y(\boldsymbol{x}; \boldsymbol{w}) = \sigma(\bar{y}(\boldsymbol{x}; \boldsymbol{w})) = \sigma(\boldsymbol{x}^T \boldsymbol{w})$.

The logistic regression output $y(\boldsymbol{x}; \boldsymbol{w})$ models the probability of class C_1, $p(C_1|\boldsymbol{x})$.

To give some meaning to the output of the linear part $\bar{y}(\boldsymbol{x}; \boldsymbol{w})$, starting with

$$p(C_1|\boldsymbol{x}) = \sigma(\bar{y}(\boldsymbol{x}; \boldsymbol{w})) = \frac{1}{1 + e^{-\bar{y}(\boldsymbol{x}; \boldsymbol{w})}},$$

we arrive at

$$\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \log\left(\frac{p(C_1|\boldsymbol{x})}{1 - p(C_1|\boldsymbol{x})}\right) = \log\left(\frac{p(C_1|\boldsymbol{x})}{p(C_0|\boldsymbol{x})}\right),$$

which is called a **logit** and it is a logarithm of odds of the probabilities of the two classes.

Logistic Regression

To train the logistic regression, we use MLE (the maximum likelihood estimation). Its application is straightforward, given that $p(C_1|\boldsymbol{x}; \boldsymbol{w})$ is directly the model output $y(\boldsymbol{x}; \boldsymbol{w})$.

Therefore, the loss for a minibatch $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ is

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_i -\log(p(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})).$$

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \{0, +1\}^N$), learning rate $\alpha \in \mathbb{R}^+$.

- $\boldsymbol{w} \leftarrow \boldsymbol{0}$ or we initialize $\boldsymbol{w}$ randomly
- until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
    - $\boldsymbol{g} \leftarrow \frac{1}{|\mathbb{B}|} \sum_{i\in\mathbb{B}} \nabla_{\boldsymbol{w}} \Big(-\log\big(p(C_{t_i}|\boldsymbol{x}_i; \boldsymbol{w})\big)\Big)$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}$

Linearity in Logistic Regression

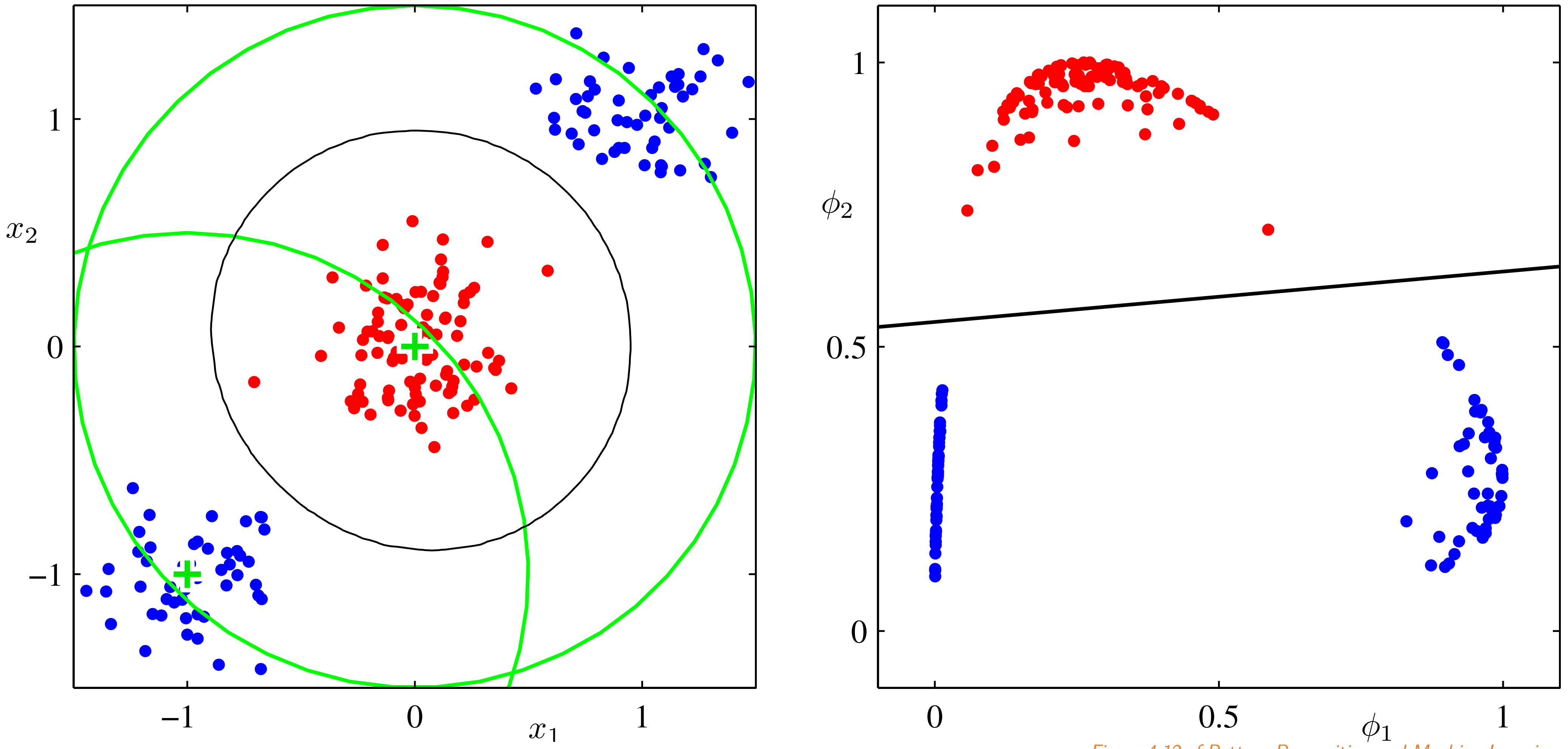


Figure 4.12 of Pattern Recognition and Machine Learning.

Generalized Linear Models

The logistic regression is in fact an extended linear regression. A linear regression model, which is followed by some **activation function** a, is called **generalized linear model**:

$$p(t|\boldsymbol{x};\boldsymbol{w},b) = y(\boldsymbol{x};\boldsymbol{w},b) = a\big(\bar{y}(\boldsymbol{x};\boldsymbol{w},b)\big) = a(\boldsymbol{x}^T\boldsymbol{w}+b).$$

Name	**Activation**	**Distribution**	**Loss**	**Gradient**
linear regression	identity	?	$\mathrm{MSE} \propto \mathbb{E}(y(\boldsymbol{x})-t)^2$	$\big(y(\boldsymbol{x})-t\big)\boldsymbol{x}$
logistic regression	$\sigma(\bar{y})$	Bernoulli	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t|\boldsymbol{x}))$	?

Logistic Regression Gradient

We start by computing the gradient of the $\sigma(x)$.

$$\begin{aligned}
\frac{\partial}{\partial x}\sigma(x) &= \frac{\partial}{\partial x}\frac{1}{1+e^{-x}} \\
&= \frac{\frac{\partial}{\partial x} - (1+e^{-x})}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \sigma(x) \cdot \frac{e^{-x}+1-1}{1+e^{-x}} \\
&= \sigma(x) \cdot \big(1 - \sigma(x)\big)
\end{aligned}$$

$$\frac{\partial}{\partial x}\frac{1}{g(x)} = -\frac{\frac{\partial}{\partial x}g(x)}{g(x)^2}$$

$$\frac{\partial}{\partial x}e^{g(x)} = e^{g(x)} \cdot \frac{\partial}{\partial x}g(x)$$

Logistic Regression Gradient

Consider the log-likelihood of logistic regression $\log p(t|\boldsymbol{x};\boldsymbol{w})$. For brevity, we denote $\bar{y}(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{x}^T\boldsymbol{w}$ just as $\bar{y}$ in the following computation.

Remembering that for $t \sim \operatorname{Ber}(\varphi)$ we have $p(t) = \varphi^t(1-\varphi)^{1-t}$, we can rewrite the log-likelihood to:

$$\begin{aligned}\log p(t|\boldsymbol{x};\boldsymbol{w}) &= \log \sigma(\bar{y})^t\big(1-\sigma(\bar{y})\big)^{1-t} \\ &= t\cdot\log\big(\sigma(\bar{y})\big) + (1-t)\cdot\log\big(1-\sigma(\bar{y})\big)\end{aligned}$$

Logistic Regression Gradient

$$\nabla_{\boldsymbol w} -\log p(t|\boldsymbol x; \boldsymbol w) =$$

$$= \nabla_{\boldsymbol w}\Big(-t\cdot\log\big(\sigma(\bar y)\big) - (1-t)\cdot\log\big(1-\sigma(\bar y)\big)\Big)$$

$$\frac{\partial}{\partial x}\log g(x) = \frac{1}{g(x)}\cdot\frac{\partial}{\partial x}g(x)$$

$$= -t\cdot\frac{1}{\sigma(\bar y)}\cdot\nabla_{\boldsymbol w}\sigma(\bar y) - (1-t)\cdot\frac{1}{1-\sigma(\bar y)}\cdot\nabla_{\boldsymbol w}\big(1-\sigma(\bar y)\big)$$

$$\frac{\partial}{\partial x}f\big(g(x)\big) = \frac{\partial}{\partial g(x)}f\big(g(x)\big)\cdot\frac{\partial}{\partial x}g(x) = \frac{\partial}{\partial z}f(z)\cdot\frac{\partial}{\partial x}g(x)$$

$$\nabla_{\boldsymbol w}\sigma(\bar y) = \tfrac{\partial}{\partial \bar y}\sigma(\bar y)\cdot\nabla_{\boldsymbol w}\bar y$$

$$= -t\cdot\frac{1}{\sigma(\bar y)}\cdot\sigma(\bar y)\cdot\big(1-\sigma(\bar y)\big)\cdot\nabla_{\boldsymbol w}\bar y + (1-t)\cdot\frac{1}{1-\sigma(\bar y)}\cdot\sigma(\bar y)\cdot\big(1-\sigma(\bar y)\big)\cdot\nabla_{\boldsymbol w}\bar y$$

$$= \big(-t + t\sigma(\bar y) + \sigma(\bar y) - t\sigma(\bar y)\big)\boldsymbol x$$

$$= \big(y(\boldsymbol x; \boldsymbol w) - t\big)\boldsymbol x$$

Generalized Linear Models

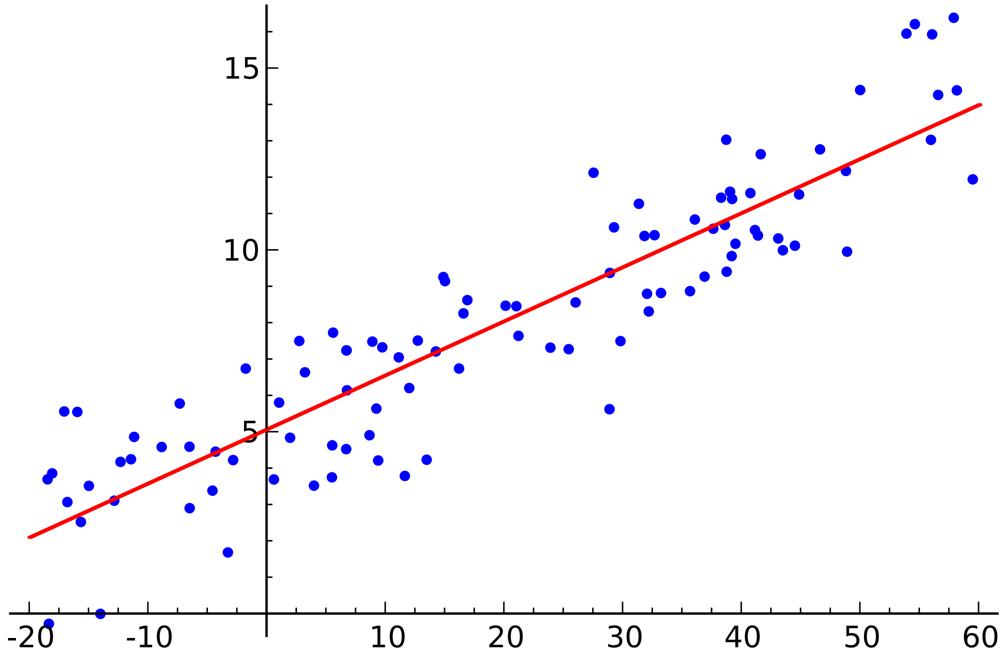
The logistic regression is in fact an extended linear regression. A linear regression model, which is followed by some **activation function** a, is called **generalized linear model**:

$$p(t|\boldsymbol{x}; \boldsymbol{w}, b) = y(\boldsymbol{x}; \boldsymbol{w}, b) = a\big(\bar{y}(\boldsymbol{x}; \boldsymbol{w}, b)\big) = a(\boldsymbol{x}^T\boldsymbol{w} + b).$$

Name	**Activation**	**Distribution**	**Loss**	**Gradient**
linear regression	identity	?	$\mathrm{MSE} \propto \mathbb{E}(y(\boldsymbol{x}) - t)^2$	$\big(y(\boldsymbol{x}) - t\big)\boldsymbol{x}$
logistic regression	$\sigma(\bar{y})$	Bernoulli	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\|\boldsymbol{x}))$	$\big(y(\boldsymbol{x}) - t\big)\boldsymbol{x}$

Mean Square Error as MLE

During regression, we predict a number, not a real probability distribution. In order to generate a distribution, we might consider a distribution with the mean of the predicted value and a fixed variance σ^2 – the most general such a distribution is the normal distribution.



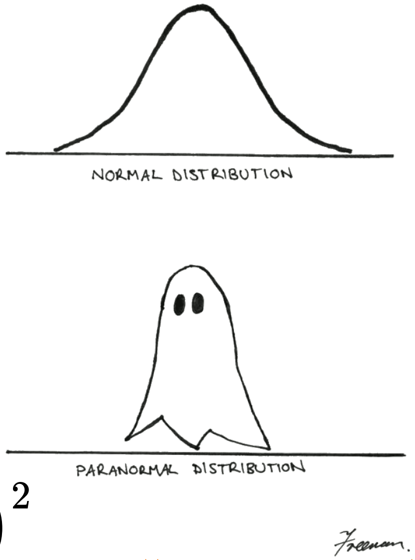
https://upload.wikimedia.org/wikipedia/commons/3/3a/Linear_regression.svg

Mean Square Error as MLE

Therefore, assume our model generates a distribution $p(t|\boldsymbol{x};\boldsymbol{w}) = \mathcal{N}(t; y(\boldsymbol{x};\boldsymbol{w}), \sigma^2)$.

Now we can apply the maximum likelihood estimation and get

$$\begin{aligned}
\arg\max_{\boldsymbol{w}} p(\boldsymbol{t}|\boldsymbol{X};\boldsymbol{w}) &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^N -\log p(t_i|\boldsymbol{x}_i;\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} -\sum_{i=1}^N \log \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(t_i - y(\boldsymbol{x}_i;\boldsymbol{w}))^2}{2\sigma^2}} \\
&= \arg\min_{\boldsymbol{w}} -N\log(2\pi\sigma^2)^{-1/2} - \sum_{i=1}^N -\frac{\big(t_i - y(\boldsymbol{x}_i;\boldsymbol{w})\big)^2}{2\sigma^2} \\
&= \arg\min_{\boldsymbol{w}} \sum_{i=1}^N \frac{\big(t_i - y(\boldsymbol{x}_i;\boldsymbol{w})\big)^2}{2\sigma^2} = \arg\min_{\boldsymbol{w}} \frac{1}{N}\sum_{i=1}^N \big(y(\boldsymbol{x}_i;\boldsymbol{w}) - t_i\big)^2.
\end{aligned}$$



https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2465539/

Generalized Linear Models

We have therefore extended the GLM table to

Name	**Activation**	**Distribution**	**Loss**	**Gradient**
linear regression	identity	Normal	$\mathrm{NLL} \propto \mathrm{MSE}$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$
logistic regression	$\sigma(\bar y)$	Bernoulli	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t \vert \boldsymbol x))$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$

Multiclass Logistic Regression

To extend the binary logistic regression to a multiclass case with K classes, we:

- generate K outputs, each with its own set of weights, so that for $\boldsymbol{W} \in \mathbb{R}^{D\times K}$,

$$\bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{x}^T \boldsymbol{W}, \quad \text{or in other words,} \quad \bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W})_i = \boldsymbol{x}^T (\boldsymbol{W}_{*,i})$$

- generalize the sigmoid function to a $\operatorname{softmax}$ function, such that

$$\operatorname{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

  Note that the original sigmoid function can be written as

$$\sigma(x) = \operatorname{softmax}\left(\begin{bmatrix}x & 0\end{bmatrix}\right)_0 = \frac{e^x}{e^x + e^0} = \frac{1}{1 + e^{-x}}.$$

The resulting classifier is also known as **multinomial logistic regression**, **maximum entropy classifier** or **softmax regression**.

Multiclass Logistic Regression

Using the $\operatorname{softmax}$ function, we naturally define that

$$p(C_i|\boldsymbol{x};\boldsymbol{W}) = \boldsymbol{y}(\boldsymbol{x};\boldsymbol{W})_i = \operatorname{softmax}\big(\bar{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{W})\big)_i = \operatorname{softmax}(\boldsymbol{x}^T\boldsymbol{W})_i = \frac{e^{(\boldsymbol{x}^T\boldsymbol{W})_i}}{\sum_j e^{(\boldsymbol{x}^T\boldsymbol{W})_j}}.$$

Considering the definition of the $\operatorname{softmax}$ function, it is natural to obtain the interpretation of the linear part of the model $\bar{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{W})$ as **logits** by computing a logarithm of the above:

$$\bar{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{W})_i = \log(p(C_i|\boldsymbol{x};\boldsymbol{W})) + c.$$

The constant c is present, because the output of the model is *overparametrized* (for example, the probability of the last class could be computed from the remaining ones). This is connected to the fact that softmax is invariant to addition of a constant:

$$\operatorname{softmax}(\boldsymbol{z}+c)_i = \frac{e^{z_i+c}}{\sum_j e^{z_j+c}} = \frac{e^{z_i}}{\sum_j e^{z_j}} \cdot \frac{e^c}{e^c} = \operatorname{softmax}(\boldsymbol{z})_i.$$

Multiclass Logistic Regression

The difference between softmax and sigmoid output can be compared on the binary case, where the binary logistic regression outputs of the linear part of the model are

$$\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \log\left(\frac{p(C_1|\boldsymbol{x}; \boldsymbol{w})}{p(C_0|\boldsymbol{x}; \boldsymbol{w})}\right),$$

while the outputs of the softmax variant with two outputs can be interpreted as $\bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W})_0 = \log(p(C_0|\boldsymbol{x}; \boldsymbol{W})) + c$ and $\bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W})_1 = \log(p(C_1|\boldsymbol{x}; \boldsymbol{W})) + c$.

If we consider $\bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W})_0$ to be zero, the model can then predict only the probability $p(C_1|\boldsymbol{x})$, and the constant c is fixed to $-\log(p(C_0|\boldsymbol{x}; \boldsymbol{W}))$, recovering the original interpretation.

Generalizing to a K-class classification, we could produce only $K-1$ outputs and define $\bar{\boldsymbol{y}}_0 = 0$, resulting in the interpretation of the linear part outputs analogous to the binary case:

$$\bar{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{W})_i = \log\left(\frac{p(C_i|\boldsymbol{x}; \boldsymbol{W})}{p(C_0|\boldsymbol{x}; \boldsymbol{W})}\right).$$

Multiclass Logistic Regression

To train K-class classification, analogously to the binary logistic regression we can use MLE and train the model using minibatch stochastic gradient descent:

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{0, 1, \ldots, K-1\}^N$), learning rate $\alpha \in \mathbb{R}^+$.
Model: Let $\boldsymbol{w}$ denote all parameters of the model (in our case, the parameters are a weight matrix $\boldsymbol{W}$ and maybe a bias vector $\boldsymbol{b}$).

- $\boldsymbol{w} \leftarrow \boldsymbol{0}$ or we initialize $\boldsymbol{w}$ randomly
- until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
  - $\boldsymbol{g} \leftarrow \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_{\boldsymbol{w}} \Big(-\log \big(p(C_{t_i} | \boldsymbol{x}_i; \boldsymbol{w})\big)\Big)$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}$

Multiclass Logistic Regression

Note that the decision regions of the binary/multiclass logistic regression are convex (and therefore connected).

To see this, consider $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$ in the same decision region R_k.

Any point $\boldsymbol{x}$ lying on the line connecting them is their convex combination, $\boldsymbol{x} = \lambda \boldsymbol{x}_A + (1-\lambda)\boldsymbol{x}_B$, and from the linearity of $\bar{\boldsymbol{y}}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{W}$ it follows that

$$\bar{\boldsymbol{y}}(\boldsymbol{x}) = \lambda \bar{\boldsymbol{y}}(\boldsymbol{x}_A) + (1-\lambda)\bar{\boldsymbol{y}}(\boldsymbol{x}_B).$$

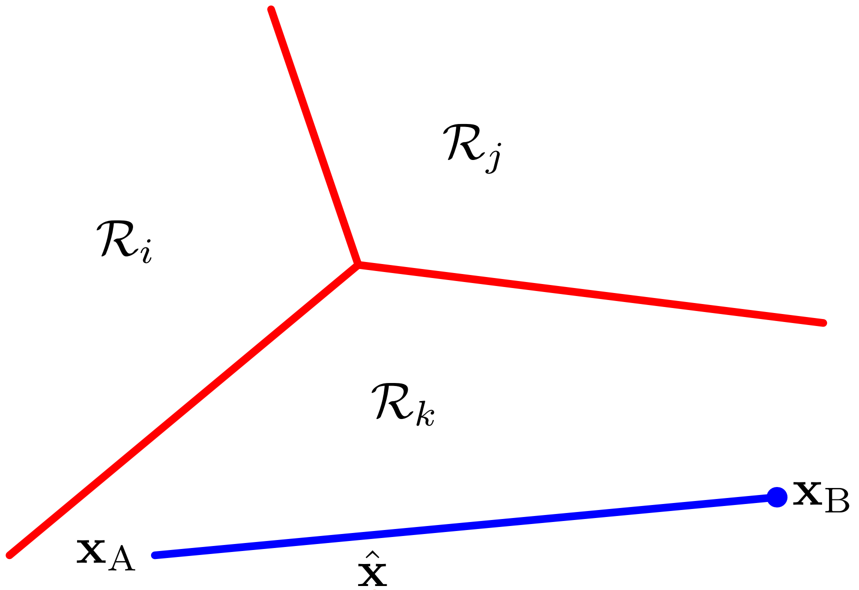


Figure 4.3 of Pattern Recognition and Machine Learning.

Given that $\bar{\boldsymbol{y}}(\boldsymbol{x}_A)_k$ was the largest among $\bar{\boldsymbol{y}}(\boldsymbol{x}_A)$ and also given that $\bar{\boldsymbol{y}}(\boldsymbol{x}_B)_k$ was the largest among $\bar{\boldsymbol{y}}(\boldsymbol{x}_B)$, it must be the case that $\bar{\boldsymbol{y}}(\boldsymbol{x})_k$ is the largest among all $\bar{\boldsymbol{y}}(\boldsymbol{x})$.

Generalized Linear Models

The multiclass logistic regression can now be added to the GLM table:

Name	Activation	Distribution	Loss	Gradient
linear regression	identity	Normal	$\mathrm{NLL} \propto \mathrm{MSE}$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$
logistic regression	$\sigma(\bar y)$	Bernoulli	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\vert\boldsymbol x))$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$
multiclass logistic regression	$\operatorname{softmax}(\bar{\boldsymbol y})$	categorical	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\vert\boldsymbol x))$	$\big((\boldsymbol y(\boldsymbol x) - \mathbf{1}_t)\boldsymbol x^T\big)^T$

Recall that $\mathbf{1}_t = \big([i = t]\big)_{i=0}^{K-1}$ is one-hot representation of target $t \in \{0, 1, \ldots, K-1\}$.

The gradient $\big((\boldsymbol y(\boldsymbol x) - \mathbf{1}_t)\boldsymbol x^T\big)^T$ can be of course also computed as $\boldsymbol x\big(\boldsymbol y(\boldsymbol x) - \mathbf{1}_t\big)^T$.

Poisson Regression

Several other GLMs exist, we now describe a final one, this time for regression and not for classification. Compared to regular linear regression, where we assume the output distribution is normal, we turn our attention to **Poisson distribution**.

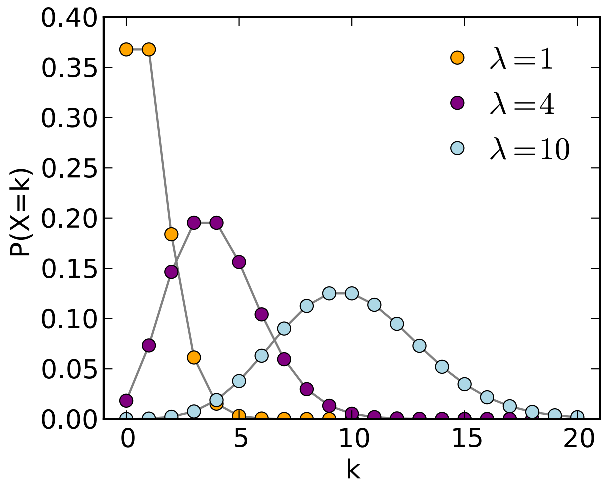
Poisson Distribution

Poisson distribution is a discrete distribution suitable for modeling the probability of a given number of events occurring in a fixed time interval, if these events occur at a known rate and independently of each other.

$$P(\mathrm{x} = k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

It is easy to show that if x has Poisson distribution,

$$\mathbb{E}[x] = \lambda$$
$$\mathrm{Var}(x) = \lambda$$



https://upload.wikimedia.org/wikipedia/commons/1/16/Poisson_pmf.svg

Poisson Distribution Derivation

The Poisson distribution can be obtained as a limit of the binomial distribution.

Assume we are considering n independent events, each with probability p_n, and that np_n converges to λ. Then

$$\lim_{n\to\infty} \binom{n}{k} p_n^k (1-p_n)^{n-k} = e^{-\lambda}\frac{\lambda^k}{k!}.$$

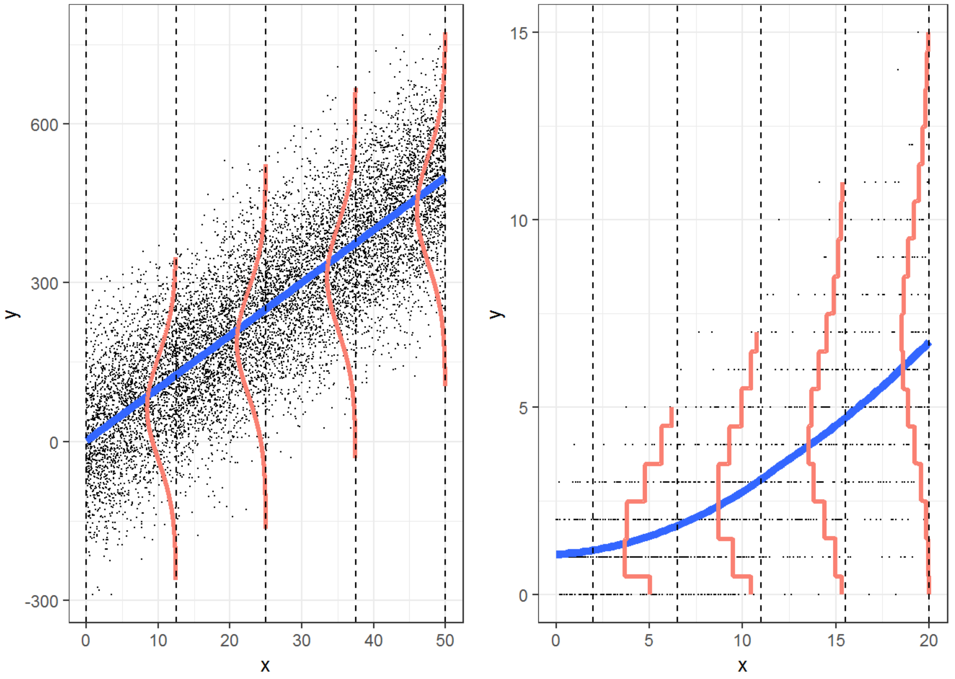
$$\begin{aligned}\lim_{n\to\infty} \binom{n}{k} p_n^k (1-p_n)^{n-k} &= \lim_{n\to\infty} \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!}\left(\frac{\lambda}{n}\right)^k\left(1-\frac{\lambda}{n}\right)^{n-k} \\ &= \lim_{n\to\infty} \frac{n^k + \mathcal{O}(n^{k-1})}{k!}\frac{\lambda^k}{n^k}\left(1-\frac{\lambda}{n}\right)^{n-k} = \lim_{n\to\infty}\frac{\lambda^k}{k!}\left(1-\frac{\lambda}{n}\right)^{n-k}\end{aligned}$$

and the result follows, since $\lim_{n\to\infty}\left(1-\frac{\lambda}{n}\right)^n = e^{-\lambda}$ and $\lim_{n\to\infty}\left(1-\frac{\lambda}{n}\right)^{-k} = 1$.

Poisson Distribution

An important difference compared to the normal distribution is that the latter assumes that the variance does not depend on the mean, i.e., that the model "makes errors of the same magnitude everywhere".

On the other hand, the variance of a Poisson distribution increases with the mean. It is useful if we want to measure error relatively, not as an absolute difference.



https://bookdown.org/roback/bookdown-bysh/bookdown-bysh_files/figure-html/OLSpois-1.png

Poisson Regression

Poisson regression is a generalized linear model producing a Poisson distribution (i.e., the mean rate λ).

Again, we use NLL as the loss. To choose a suitable activation, we might be interested in obtaining the same gradient as for other GLMs – solving for an activation function while requiring the gradient to be $\big(a(\bar y(\boldsymbol x)) - t\big) \cdot \boldsymbol x$ yields $a(\bar y) = \exp(\bar y)$, which means the linear part of the model is predicting $\log(\lambda)$.

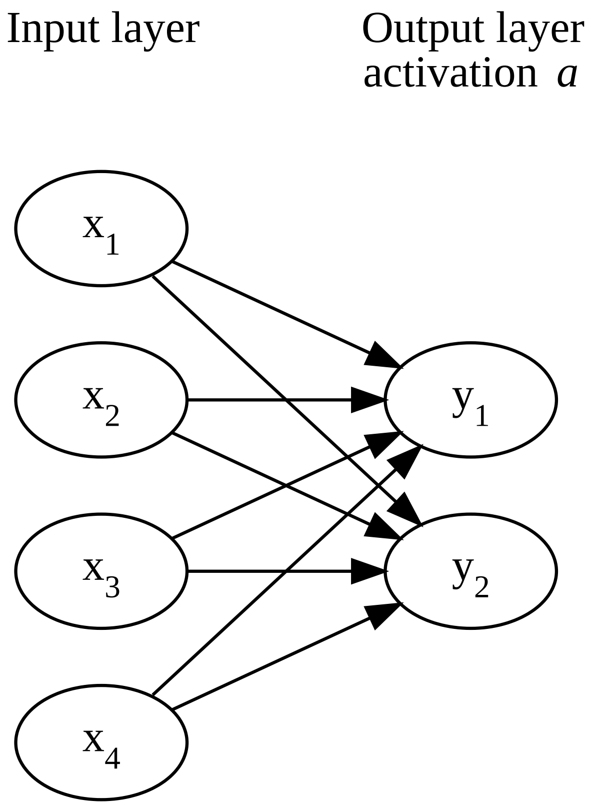
Name	Activation	Distribution	Loss	Gradient
linear regression	identity	Normal	$\mathrm{NLL} \propto \mathrm{MSE}$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$
logistic regression	$\sigma(\bar y)$	Bernoulli	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\vert\boldsymbol x))$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$
multiclass logistic regression	$\operatorname{softmax}(\bar{\boldsymbol y})$	categorical	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\vert\boldsymbol x))$	$\big((\boldsymbol y(\boldsymbol x) - \mathbf{1}_t)\boldsymbol x^T\big)^T$
Poisson regression	$\exp(\bar y)$	Poisson	$\mathrm{NLL} \propto \mathbb{E} - \log(p(t\vert\boldsymbol x))$	$\big(y(\boldsymbol x) - t\big)\boldsymbol x$

Multilayer Perceptron

We can reformulate the generalized linear models in the following framework.

- Assume we have an input node for every input feature.
- Additionally, we have an output node for every model output (one for linear regression or binary classification, K for classification in K classes).
- Every input node and output node are connected with a directed edge, and every edge has an associated weight.
- Value of every (output) node is computed by summing the values of predecessors multiplied by the corresponding weights, added to a bias of this node, and finally passed through an activation function a:

$$y_i = a\left(\sum\nolimits_j x_j w_{j,i} + b_i\right)$$



or in matrix form $\boldsymbol{y} = a(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b})$, or for a batch of examples $\boldsymbol{X}$, $\boldsymbol{Y} = a(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b})$.

Multilayer Perceptron

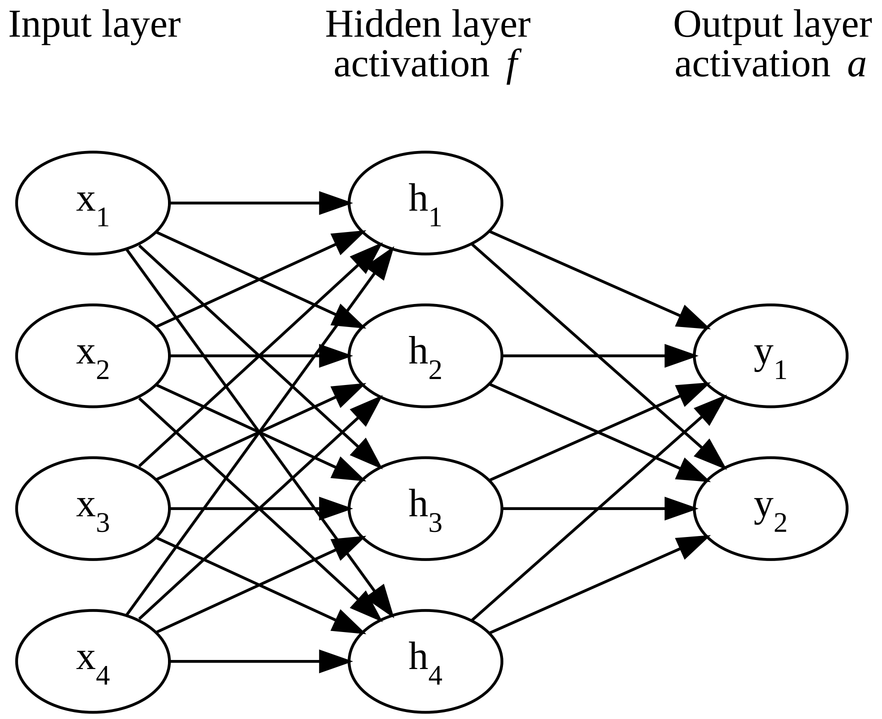
We now extend the model by adding a **hidden layer** with activation f.

- The computation is performed analogously:

$$h_i = f\left(\sum\nolimits_j x_j w_{j,i}^{(h)} + b_i^{(h)}\right),$$
$$y_i = a\left(\sum\nolimits_j h_j w_{j,i}^{(y)} + b_i^{(y)}\right),$$

or in matrix form

$$\boldsymbol h = f\Big(\boldsymbol x^T \boldsymbol W^{(h)} + \boldsymbol b^{(h)}\Big),$$
$$\boldsymbol y = a\Big(\boldsymbol h^T \boldsymbol W^{(y)} + \boldsymbol b^{(y)}\Big),$$



and for batch of inputs $\boldsymbol H = f\Big(\boldsymbol X \boldsymbol W^{(h)} + \boldsymbol b^{(h)}\Big)$ and $\boldsymbol Y = a\Big(\boldsymbol H \boldsymbol W^{(y)} + \boldsymbol b^{(y)}\Big)$.

Multilayer Perceptron

Note that:

- the structure of the *input* layer depends on the input features;
- the structure and the *activation* function of the *output* layer depends on the target data;
- however, the *hidden* layer has no pre-image in the data and is completely arbitrary – which is the reason why it is called a *hidden* layer.

Also note that we can absorb biases into weights analogously to the generalized linear models.

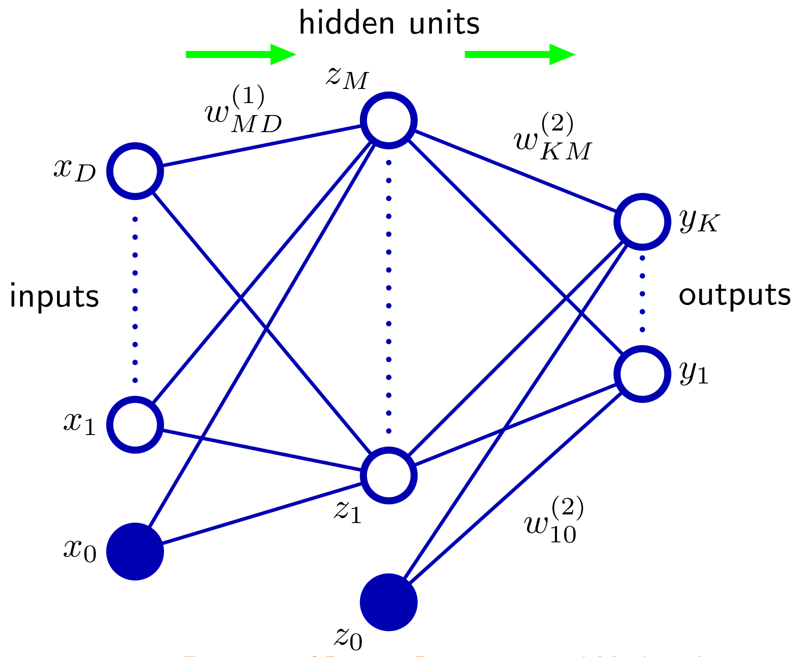


Figure 5.1 of Pattern Recognition and Machine Learning.

Output Layer Activation Functions

- regression:
  - identity activation: we model normal distribution on output (linear regression)
  - $\exp(x)$: we model Poisson distribution on output (Poisson regression)

- binary classification:
  - $\sigma(x)$: we model the Bernoulli distribution (the model predicts a probability)

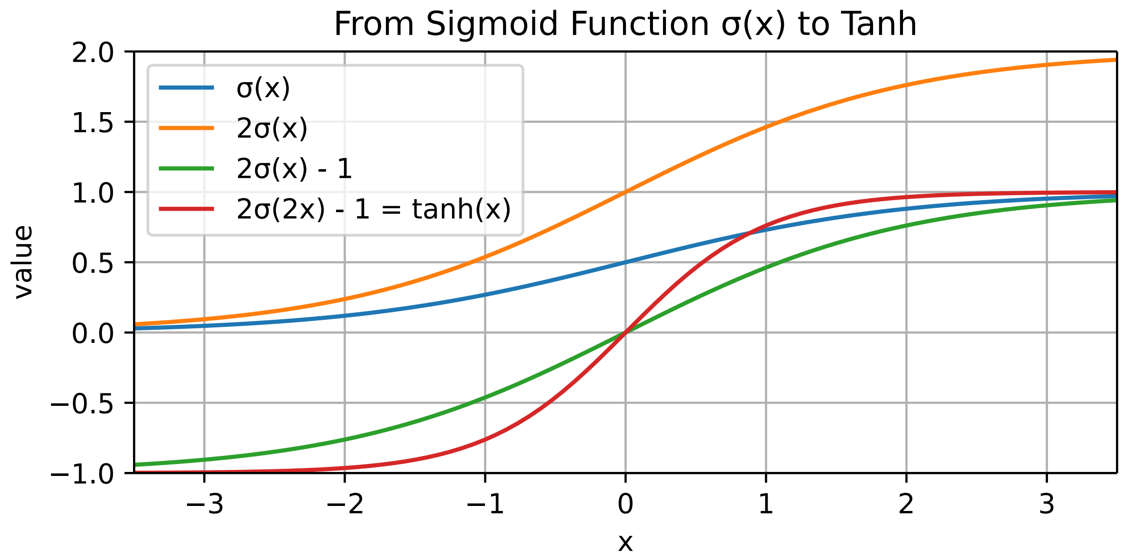
$$\sigma(x) \stackrel{\text{def}}{=} \frac{1}{1+e^{-x}}$$

- K-class classification:
  - $\operatorname{softmax}(\boldsymbol{x})$: we model the (usually overparametrized) categorical distribution

$$\operatorname{softmax}(\boldsymbol{x}) \propto e^{\boldsymbol{x}}, \quad \operatorname{softmax}(\boldsymbol{x})_i \stackrel{\text{def}}{=} \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}$$

Hidden Layer Activation Functions

- no activation (identity): does not help, composition of linear mapping is a linear mapping
- σ (but works suboptimally – nonsymmetrical, $\frac{d\sigma}{dx}(0) = 1/4$)
- $\tanh$
  - result of making σ symmetrical and making derivation in zero 1
  - $\tanh(x) = 2\sigma(2x) - 1$
- ReLU
  - $\max(0, x)$
  - the most common nonlinear activation used nowadays



Training MLP

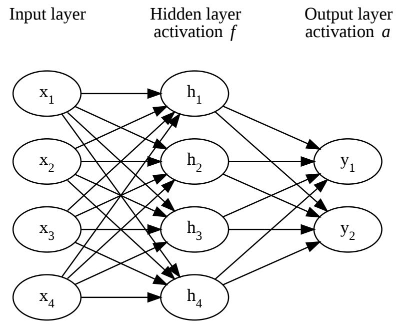
The multilayer perceptron can be trained using again a minibatch SGD algorithm:

Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t}$ targets), learning rate $\alpha \in \mathbb{R}^+$.
Model: Let $\boldsymbol{w}$ denote all parameters of the model (all weight matrices and bias vectors).

- initialize $\boldsymbol{w}$
  - set weights randomly
    - for a weight matrix processing a layer of size M to a layer of size O, we can sample its elements uniformly for example from the $\left[-\frac{1}{\sqrt{M}}, \frac{1}{\sqrt{M}}\right]$ range
    - the exact range becomes more important for networks with many hidden layers
  - set biases to 0
- until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
  - $\boldsymbol{g} \leftarrow \frac{1}{|\mathbb{B}|}\sum_{i\in\mathbb{B}} \nabla_{\boldsymbol{w}}\Big(-\log\big(p(t_i|\boldsymbol{x}_i;\boldsymbol{w})\big)\Big)$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{g}$

Training MLP – Computing the Derivatives

Assume we have an MLP with input of size D, weights $\boldsymbol W^{(h)} \in \mathbb{R}^{D\times H}$, $\boldsymbol b^{(h)} \in \mathbb{R}^H$, hidden layer of size H and activation f with weights $\boldsymbol W^{(y)} \in \mathbb{R}^{H\times K}$, $\boldsymbol b^{(y)} \in \mathbb{R}^K$, and finally an output layer of size K with activation a.



In order to compute the gradient of the loss L with respect to all weights, you should proceed gradually:

- first compute $\frac{\partial L}{\partial \boldsymbol y}$,
- then compute $\frac{\partial \boldsymbol y}{\partial \boldsymbol y^{(in)}}$, where $\boldsymbol y^{(in)}$ are the inputs to the output layer (i.e., before applying activation function a; in other words, $\boldsymbol y = a(\boldsymbol y^{(in)})$),
- then compute $\frac{\partial \boldsymbol y^{(in)}}{\partial \boldsymbol W^{(y)}}$ and $\frac{\partial \boldsymbol y^{(in)}}{\partial \boldsymbol b^{(y)}}$, which allows us to obtain $\frac{\partial L}{\partial \boldsymbol W^{(y)}} = \frac{\partial L}{\partial \boldsymbol y} \cdot \frac{\partial \boldsymbol y}{\partial \boldsymbol y^{(in)}} \cdot \frac{\partial \boldsymbol y^{(in)}}{\partial \boldsymbol W^{(y)}}$ and analogously $\frac{\partial L}{\partial \boldsymbol b^{(y)}}$,
- followed by $\frac{\partial \boldsymbol y^{(in)}}{\partial \boldsymbol h}$ and $\frac{\partial \boldsymbol h}{\partial \boldsymbol h^{(in)}}$,
- and finally using $\frac{\partial \boldsymbol h^{(in)}}{\partial \boldsymbol W^{(h)}}$ and $\frac{\partial \boldsymbol h^{(in)}}{\partial \boldsymbol b^{(h)}}$ to compute $\frac{\partial L}{\partial \boldsymbol W^{(h)}}$ and $\frac{\partial L}{\partial \boldsymbol b^{(h)}}$.

Hidden Layer Interpretation and Initialization

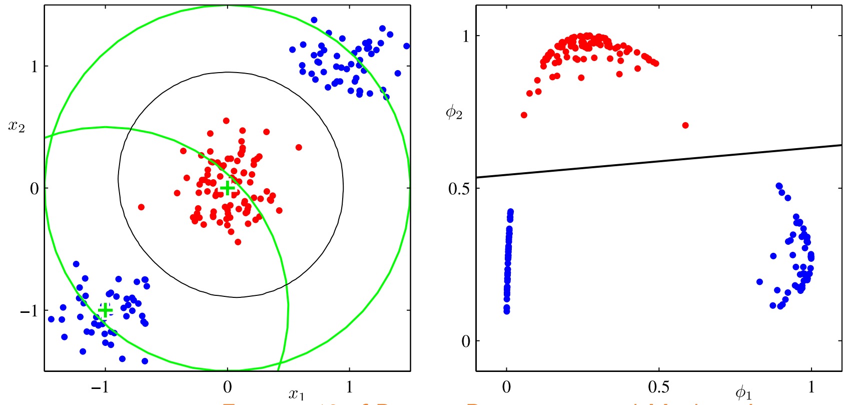


Figure 4.12 of Pattern Recognition and Machine Learning.

One way how to interpret the hidden layer is:

- the part from the hidden layer to the output layer is the previously used generalized linear model (linear regression, logistic regression, …);
- the part from the inputs to the hidden layer can be considered automatically constructed features. The features are a linear mapping of the input values followed by a nonlinearity, and the theorem on the next slide proves they can always be constructed to achieve as good a fit of the training data as is required.

Note that the weights in an MLP must be initialized randomly. If we used just zeros, all the constructed features (hidden layer nodes) would behave identically and we would never distinguish them.

Using random weights corresponds to starting with random features, which allows the SGD to make progress (improve the individual features).

Universal Approximation Theorem '89

Let $\varphi(x) : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded and nondecreasing continuous function.
(Later a proof was given also for $\varphi = \mathrm{ReLU}$ and even for any nonpolynomial function.)

For any $\varepsilon > 0$ and any continuous function $f : [0,1]^D \to \mathbb{R}$, there exists $H \in \mathbb{N}$, $\boldsymbol{v} \in \mathbb{R}^H$, $\boldsymbol{b} \in \mathbb{R}^H$ and $\boldsymbol{W} \in \mathbb{R}^{D \times H}$, such that if we denote

$$F(\boldsymbol{x}) = \boldsymbol{v}^T \varphi(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}) = \sum_{i=1}^{H} v_i \varphi(\boldsymbol{x}^T \boldsymbol{W}_{*,i} + b_i),$$

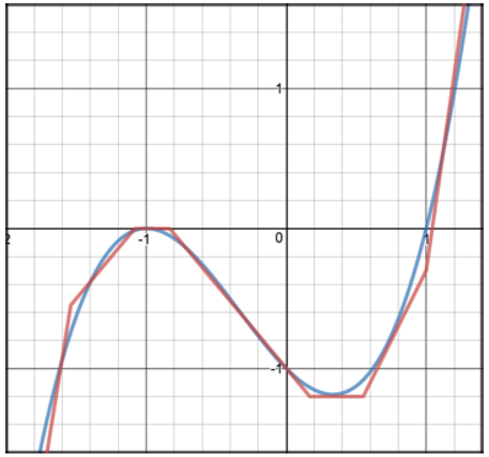
where φ is applied elementwise, then for all $\boldsymbol{x} \in [0,1]^D$:

$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon.$$

Universal Approximation Theorem for ReLUs

Sketch of the proof:

- If a function is continuous on a closed interval, it can be approximated by a sequence of lines to arbitrary precision.



$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$\begin{aligned}Z(x) = &-n_1(x) - n_2(x) - n_3(x)\\ &+ n_4(x) + n_5(x) + n_6(x)\end{aligned}$$

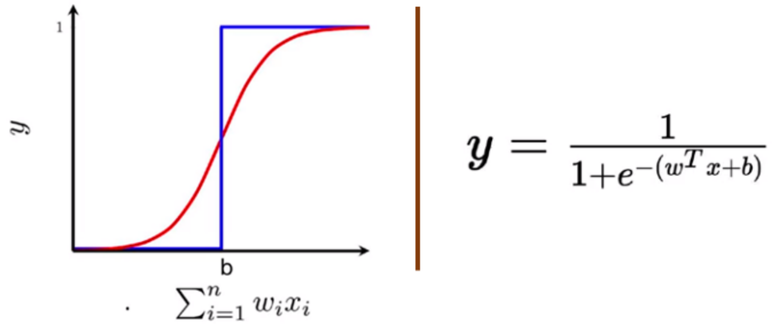
*https://miro.medium.com/max/844/1*lihbPNQgl7oKjpCsmzPDKw.png*

- However, we can create a sequence of k linear segments as a sum of k ReLU units – on every endpoint a new ReLU starts (i.e., the input ReLU value is zero at the endpoint), with a tangent which is the difference between the target tangent and the tangent of the approximation until this point.

Universal Approximation Theorem for Squashes

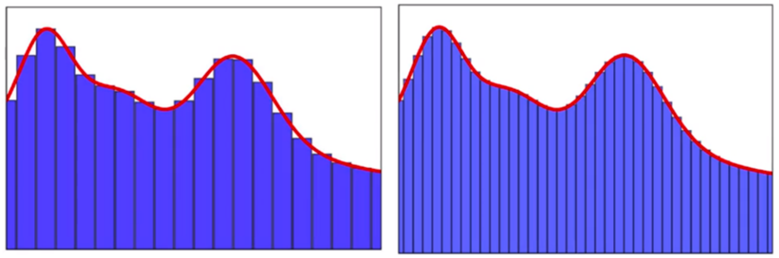
Sketch of the proof for a squashing function $\varphi(x)$ (i.e., nonconstant, bounded and nondecreasing continuous function like sigmoid):

- We can prove φ can be arbitrarily close to a hard threshold by compressing it horizontally.



*https://hackernoon.com/hn-images/1*N7dfPwbiXC-Kk4TCbfRerA.png*

- Then we approximate the original function using a series of straight line segments



*https://hackernoon.com/hn-images/1*hVuJgUTLUFWTMmJhl_fomg.png*