Linear Classifiers, etc.

Linear Classifiers

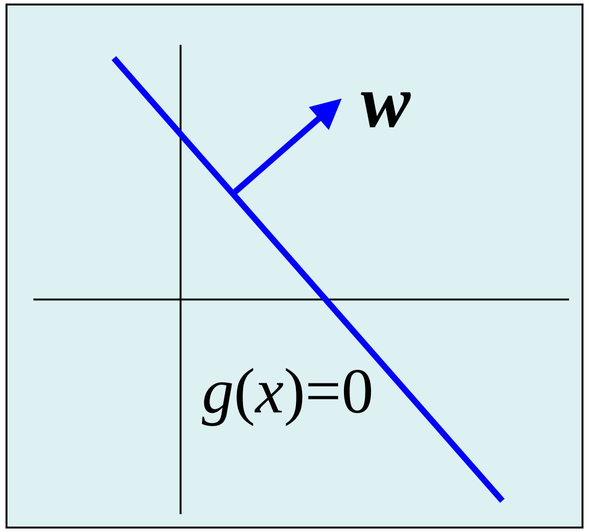
A linear classifier is one that has linear decision boundaries.

Consider 2-class problems. A linear discriminant function has the form of

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

The decision boundary (line, plane, hyperplane) is given by

$$g(\boldsymbol{x}) = 0$$

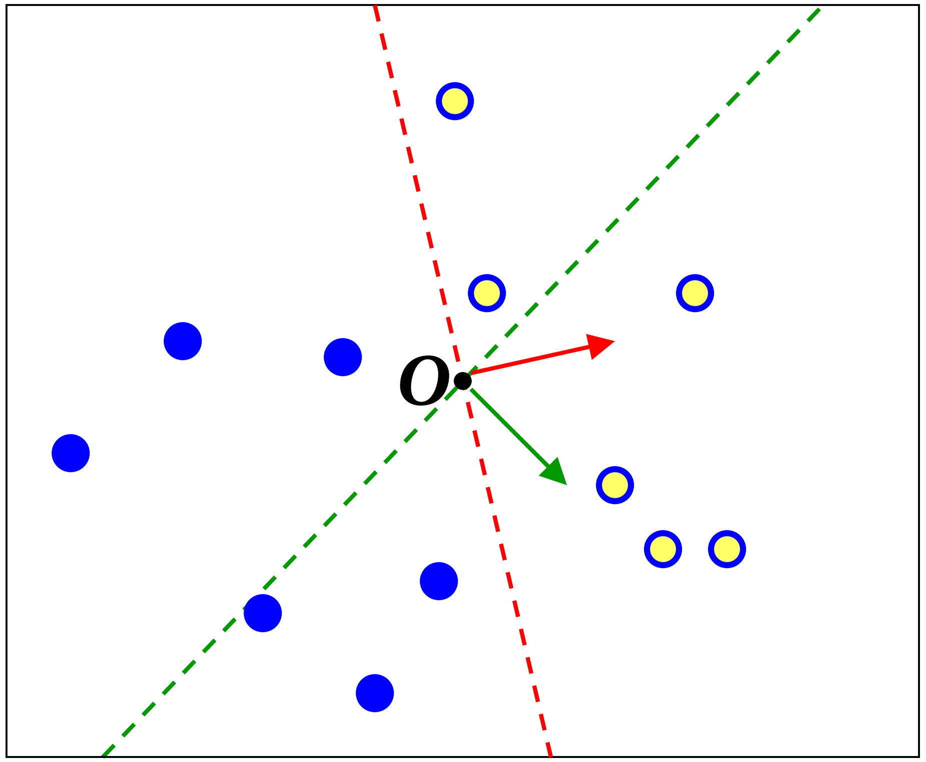


The vector $\boldsymbol{w}$ is normal to the decision hyperplane.

Augmented Vectors

Augmented vectors: To simplify notation, we apply notation changes to incorporate the constant term in $g(\boldsymbol{x})$:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \quad \text{with} \quad \boldsymbol{w} \leftarrow \begin{bmatrix} w_0 \\ \boldsymbol{w} \end{bmatrix} \quad \boldsymbol{x} \leftarrow \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$$



○ ω_1 ● ω_2

"Augmented" $\boldsymbol{w}$ in $(l+1)$-dimensional space; the decision boundary always passes through the origin.

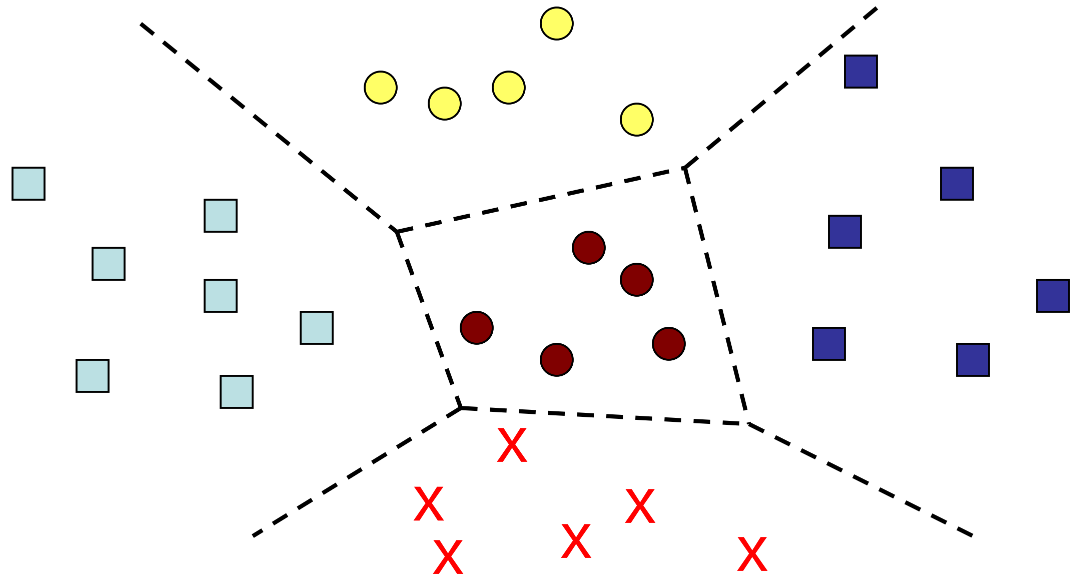
Multi-Class Cases

Let there be M classes. The idea is to use a set of M linear discriminant functions,

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x}$$

with the goal being that $\mathbf{w}_i^T \mathbf{x} > \mathbf{w}_j^T \mathbf{x} \quad \forall j \neq i$ if $\mathbf{x} \in \omega_i$

Example with M=5:



Decision regions formed by linear classifiers are convex.

Linear Regression for Binary Classification

$$\begin{tikzcd}
\boldsymbol{x}_1 \arrow[r] & y_1 \\
\boldsymbol{x}_2 \arrow[r] & y_2 \\
\vdots &
\end{tikzcd}$$

Now assume that we have a set of training data, in the form of inputs $\boldsymbol{x}_i$ and corresponding correct outputs y_i (for two-class classification, we can set y_i to $+1$ and -1 for the two classes):

$$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T$$

$$\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^T$$

Linear regression with min-squared error (MSE) loss function:

$$E(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2 = \| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w} \|^2$$

To solve for $\boldsymbol{w}$: $$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = 0$$

Solution: $$\boldsymbol{w} = \boldsymbol{X}^+ \boldsymbol{y} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

$\boldsymbol{X}^+$: the pseudo-inverse of $\boldsymbol{X}$

Widrow-Hoff Procedure for MSE

A gradient-descent approach to minimize the MSE loss function:

Gradient: $$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = 2\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

Update equation (batch version) for $\boldsymbol{w}$: $$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \underbrace{\eta_t}_{\text{Learning Rate}}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

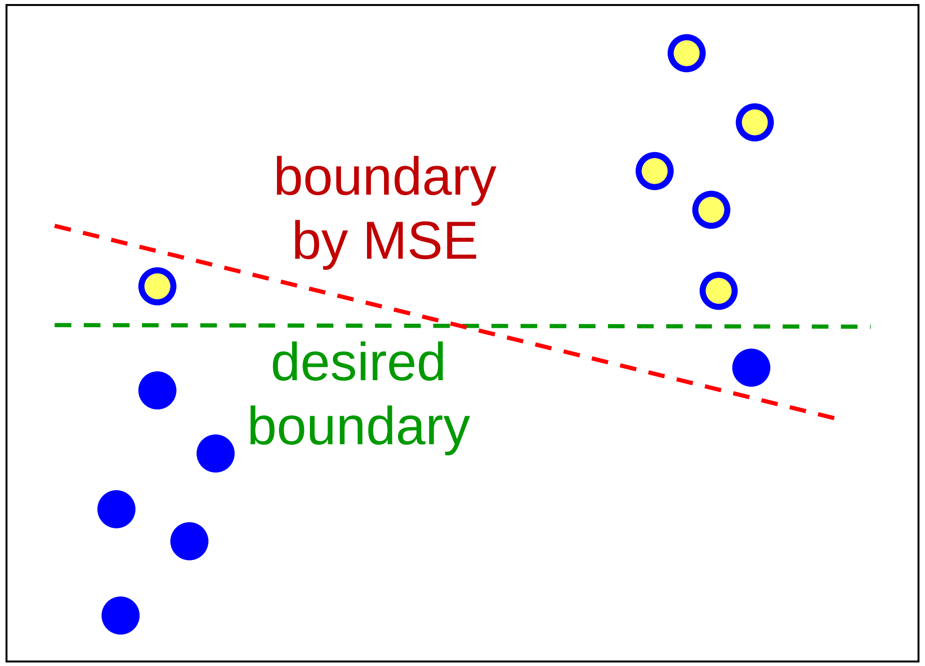
An on-line version (update $\boldsymbol{w}$ for each sample) is also possible: $$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta_t \frac{\partial E}{\partial \boldsymbol{w}} = \boldsymbol{w}(t) + \eta_t e(t)\boldsymbol{x}(t)$$

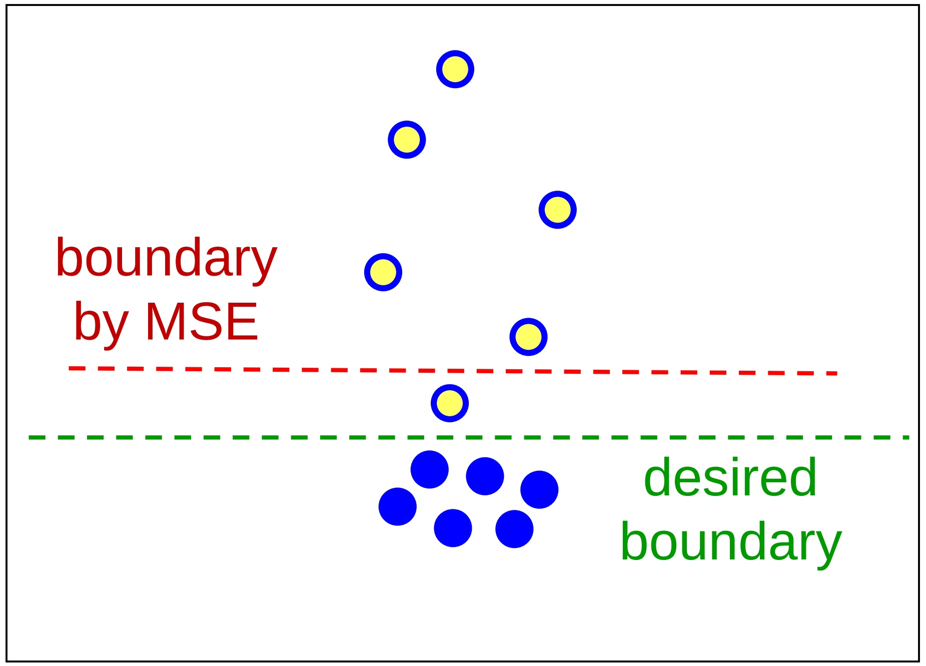
- Works even if $\boldsymbol{X}^T\boldsymbol{X}$ is singular.
- Reduced time/memory complexity.
- Convergence proof for this learning rate: $\eta_t = \eta_1 / t$ and $\eta_1 > 0$
- Applicable to dynamic (time-varying) systems using the on-line version.

Problem of Classification with MSE

"Strange" cases to illustrate that

minimizing MSE ≠ minimizing classification error





The issue of MSE loss is that it attempts to make the samples have similar distances to the boundary. This also leads to sensitivity to outliers.

Perceptron Algorithm

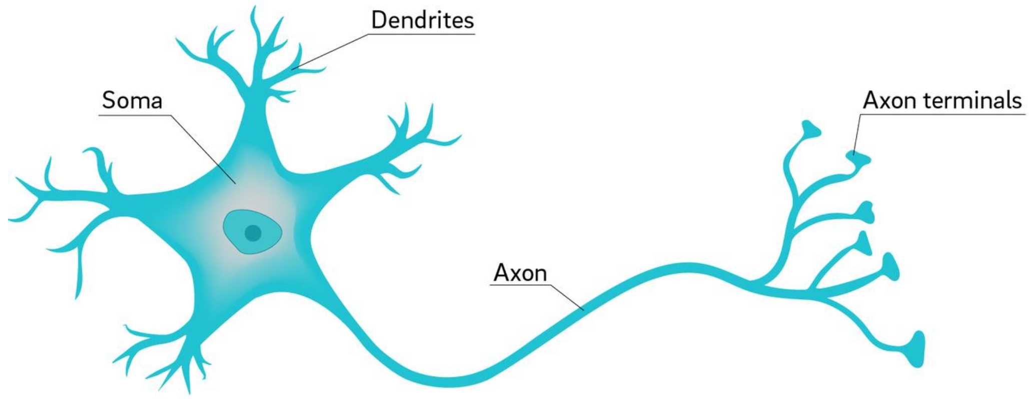
- Instead of treating classification as regression, one can try to optimize correct classification rate directly.
- The idea: Iterative minimization of a loss function consisting of contributions only from misclassified samples:

$$J(\mathbf{w}) = \sum \left(-\mathbf{w}^T \mathbf{x} \delta_{\mathbf{x}} \right) \qquad \delta_{\mathbf{x}} = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is misclassified and } \mathbf{x} \in \omega_1 \\ -1 & \text{if } \mathbf{x} \text{ is misclassified and } \mathbf{x} \in \omega_2 \\ 0 & \text{if } \mathbf{x} \text{ is classified correctly} \end{cases}$$

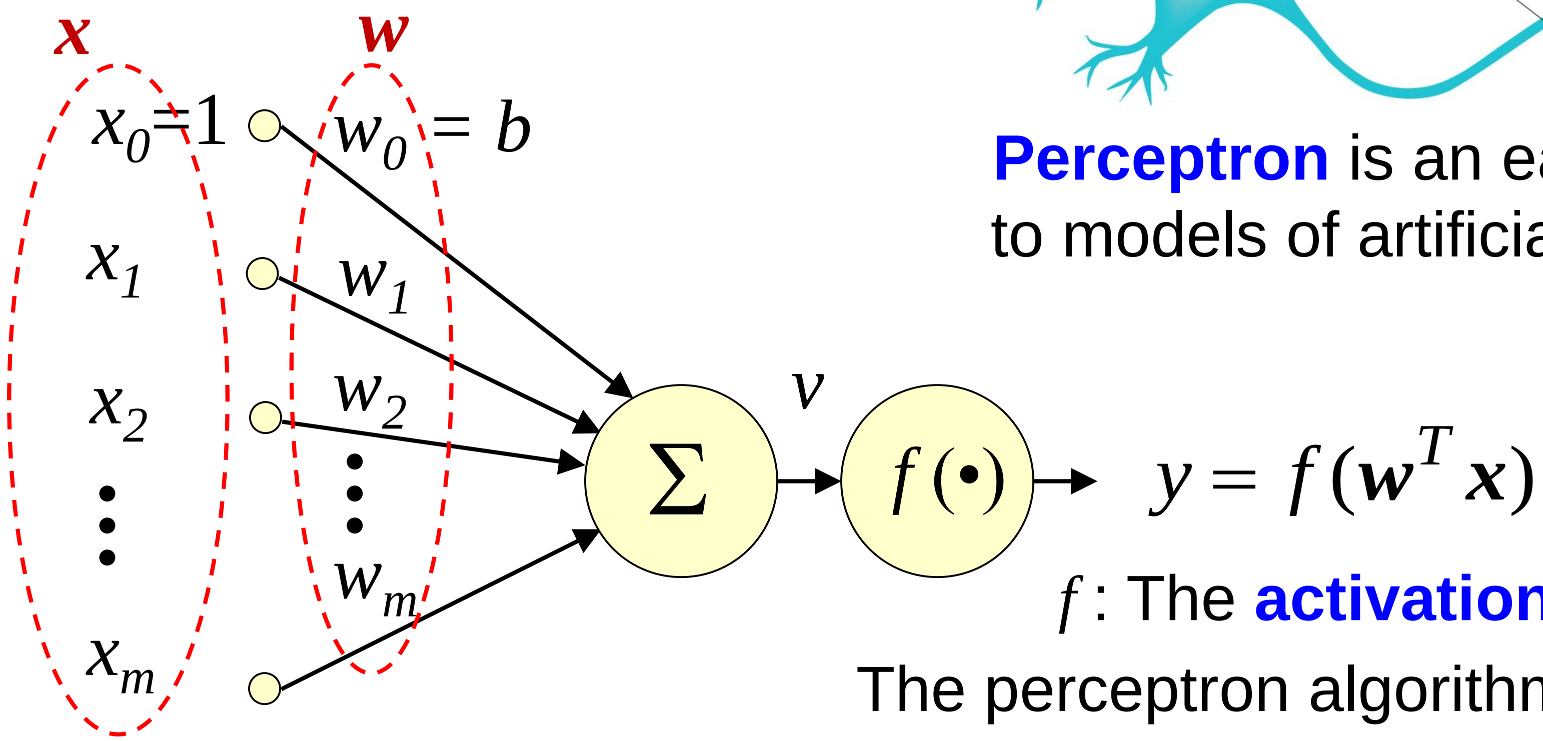
- Update equations (gradient-descent of $J(\mathbf{w})$):
  - On-line form: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta_t \mathbf{x}(t) \delta_{\mathbf{x}(t)}$
  - Batch form: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta_t \sum_{\mathbf{x} \in Y} \mathbf{x} \delta_{\mathbf{x}}$

with learning rate $\eta_t > 0$

What is "Perceptron" Anyway?



Perceptron is an early name given to models of artificial neurons.



$$y = f(\boldsymbol{w}^T \boldsymbol{x})$$

f : The **activation function**

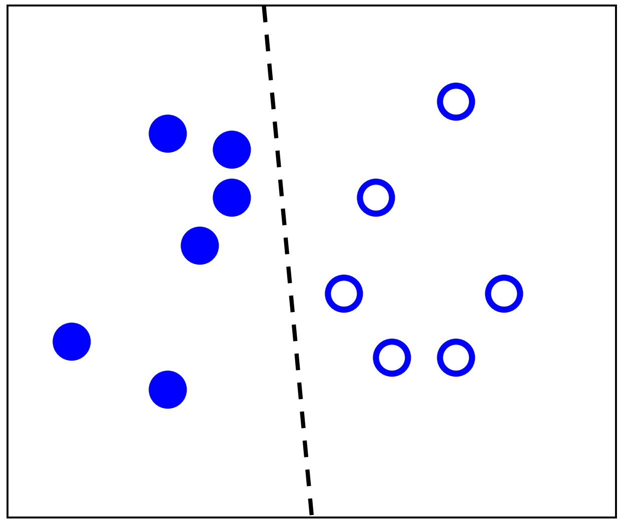
The perceptron algorithm just uses $f(v) = v$.

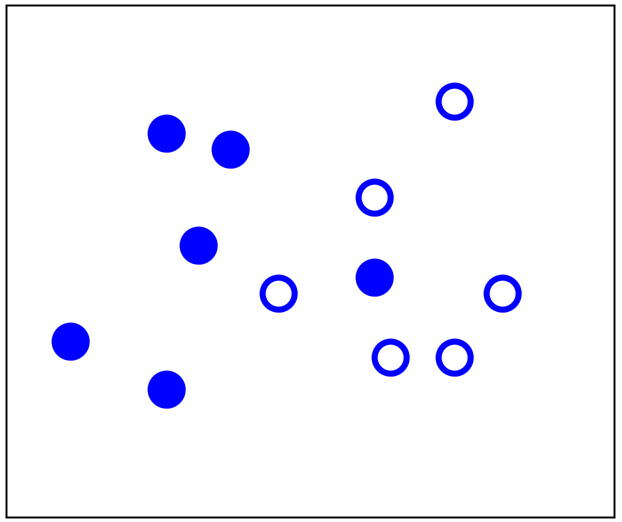
Some Thoughts About the Perceptron Algorithm

- Provable convergence for suitable learning rates (such as a constant learning rate or one proportional to $1/t$) AND linearly separable data.
- Solutions (if existent) are not unique.
- Data that are not linearly separable result in infinite updates. (Think about the loss function: the gradient never approach zero.)
- Not used much today, but historically important:
  - The first real "machine learning" algorithm designed for classification problems. ➔ Over-optimism.
  - The limitation on data that are not linearly separable led people to give up on neural computing altogether. ➔ Over-pessimism. (This issue was eventually solved by the invention of backpropagation almost 10 years later.)

Linearly Separable Classes

A 2-class problem is linearly separable iff a decision hyperplane exists such that all the data points in ω_1 are on one side of the hyperplane and all the data points in ω_2 are on the other side of the hyperplane.


Linearly separable


Not linearly separable

Perceptron Algorithm When Not Linearly Separable ...

Some simple modifications:

- The **Pocket Algorithm**: Remember the currently best solution, and keep going until you decide to stop.
- The **Fuzzy Perceptron Algorithm**:
  - Idea: Reduce the influence of the more confusing samples.
  - Give each sample "memberships" in the two classes (sum to one), computed according to the sample's distances to the two class means.
  - Scale the weight update by the difference between a sample's two memberships.

Fuzzy Perceptron Algorithm

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta_t \, \boldsymbol{x}(t)\delta_{\boldsymbol{x}(t)}$$

$$\Rightarrow \quad \boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta_t \left|u_1 - u_2\right|^m \boldsymbol{x}(t)\delta_{\boldsymbol{x}(t)}$$

$$u_1 = \frac{1}{2} + \frac{e^{\alpha(d_2-d_1)/d} - e^{-\alpha}}{2(e^{\alpha} - e^{-\alpha})}, \quad u_2 = 1 - u_1$$

- Parameters:
  - m: fuzzification factor (>1)
  - d_1 and d_2: distances to the two class means
  - d: distance between the two class means
  - α: constant
- Confusing samples: Those with $|u_1 - 0.5| \le \beta$.
- Termination: When all the non-confusing samples are classified correctly.

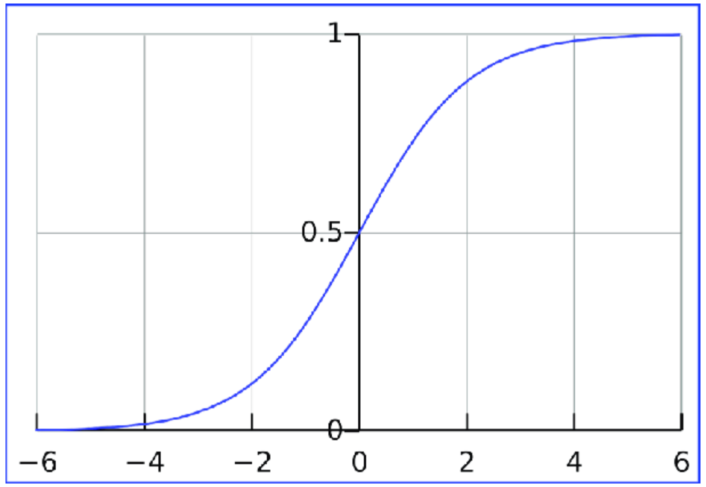
Cross-Entropy Loss

- Instead of optimizing for correct classification directly, another idea is to make the classifier predict the class probabilities of the samples, and then optimize the likelihood of the samples being correctly classified.:
  - Let the target outputs y for the two classes be 1 or 0.
  - Let the predicted outputs (probabilities in the two classes) of a sample $\boldsymbol{x}$ be p_1 and $p_0 = 1 - p_1$.
  - Likelihood: $\prod_{\boldsymbol{x}\in\omega_1} p_1(\boldsymbol{x}) \prod_{\boldsymbol{x}\in\omega_0} p_0(\boldsymbol{x})$
  - Loglikelihood: $\sum_{\boldsymbol{x}\in\omega_1} \ln\big(p_1(\boldsymbol{x})\big) + \sum_{\boldsymbol{x}\in\omega_0} \ln\big(p_0(\boldsymbol{x})\big) = \sum_{\boldsymbol{x}_i} \big[y_i \ln\big(p_1(\boldsymbol{x}_i)\big) + (1 - y_i)\ln\big(p_0(\boldsymbol{x}_i)\big)\big]$
- The negative loglikelihood is the **cross-entropy** between two distributions (target and predicted), each consisting of probabilities in two classes.

Logistic Regression

- To use cross-entropy loss in classification, we need a parameterized representation of the predicted class probabilities.
- Logistic function is the most commonly used form:

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



- This is a form of sigmoid functions.
- Parameterized for linear classification:

$$p(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

- We can substitute this into cross-entropy and solve for the parameters (***w***). This is considered maximum likelihood estimation. (No analytical solution exists, and iterative optimization methods are required.)

Logistic Regression

- Connection with Bayes rule:
  - In Bayesian classifiers, we make classification decision on $\boldsymbol{x}$ based on the posteriori probabilities (two-class case):

$$P(\omega_1 \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_1)P(\omega_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x} \mid \omega_1)P(\omega_1)}{p(\mathbf{x} \mid \omega_1)P(\omega_1) + p(\mathbf{x} \mid \omega_2)P(\omega_2)}$$

  - We can express it as a logistic function:

$$\frac{1}{1+e^{-z}} \qquad \text{where} \qquad z = \ln \frac{p(\mathbf{x} \mid \omega_1)P(\omega_1)}{p(\mathbf{x} \mid \omega_2)P(\omega_2)}$$

- This is considered "probabilistic discriminative model" of classification. Unlike Bayesian classifiers, the actual class-conditional pdfs do not have to be estimated. ➔ Fewer parameters, faster, and not dependent on the assumed functional form of the pdfs.

Multinominal (Multi-Class) Logistic Regression

- Cross entropy can be easily generalized to more than two classes.
- The predicted class probabilities of logistic regression becomes

$$\frac{e^{-z_i}}{\sum_k e^{-z_k}}$$

- This is commonly called the softmax function.