LZ~~1~~1a

Support Vector Machines (SVM)

SVM is one of the most powerful supervised machine learning algorithm. One don't have to do a lot of tweaks to obtain good results with SVM.

Basically, SVM is a binary learning machine with some highly elegant properties.

Main idea of SVM

"Given a training sample, SVM constructs a hyperplane as the decision surface in such a way that the margin of separation b/w two classes is maximized"

Hyperplane → geometric entity whose dimension is one less than that of its ambient space

Hyperplane for 2D space are 1D lines
3D space are 2D planes
and soon

2 dimensional linearly separable data can be separated by a line.

$$y = mx + c$$

$$y = ax + b \quad \text{same right?}$$

Now $y = x_2$ & $x = x_1$ $\Rightarrow$ $x_2 = ax_1 + b$

One then get

$$ax_1 - x_2 + b = 0 \quad -①$$

$X = (x_1, x_2)$ and say $w = (a, -1)$.

① is written as

$$w^T \cdot x + b = 0$$

This eqn is derived from 2-dimensional we close let us try to extend it to higher dimension.

$$a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n + b = 0$$

$X = (x_1, x_2, \ldots x_n)$ $\quad w = (a_1, a_2, a_3, \ldots a_n)$

So we can explain hyperplane in same way.

$$w^T \cdot x + b = 0$$

Once we have the hyperplane, we can use it for making prediction.

Consider training sample $\{(x_i, d_i)\}_{i=1}^{N}$, where x_i is input pattern for the i^{th} example and d_i is the corresponding desired response (target output).

Assume classes represented by subset $d_i = +1$ and other subset $d_i = -1$ are "linearly separable".

The equation of decision surface in the form of

hyperplane that does the separation is

$$w^T x + b = 0 \quad \cdots (1)$$

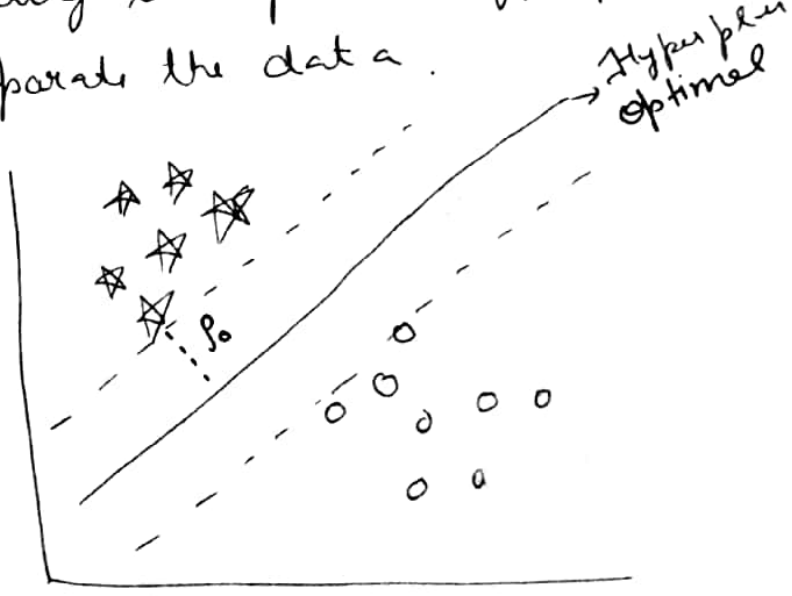
where x is input vector, w is an adjustable weight vector & b is a bias.

One then write

$$w^T x_i + b \geq 0 \quad \text{for } d_i = +1 \quad \cdots (2)$$
$$w^T x_i + b < 0 \quad \text{for } d_i = -1$$

The idea of assumption of linearly separable classes is made here to explain basic idea behind a SVM in simple setting.

→ SVM works by finding the optimal hyperplane which could best separate the data.

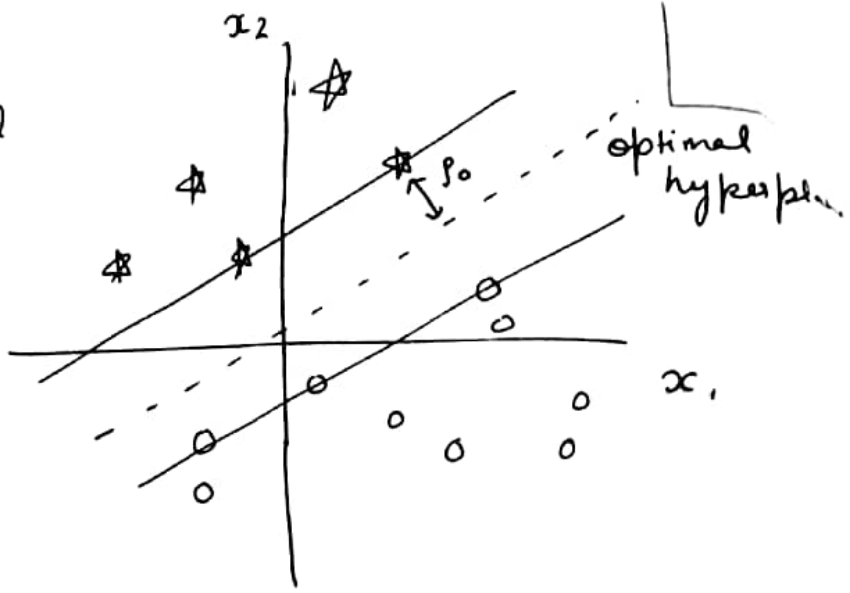


For a given weight vector w and bias b, separation b/w the hyperplane defined in (1) and the closest data point is called the "margin of separation" denoted by ρ.

* Goal of SVM is to find the particular hyperplane for which the margin of separation ρ is maximized.

Decision surface referred to as optimal hyperplane

let
w_o & b_o denote optimal values of weight vector and bias, respectively



Correspondingly

optimal hyperplane is defined as

$$w_o^T x + b_o = 0 \quad -(3)$$

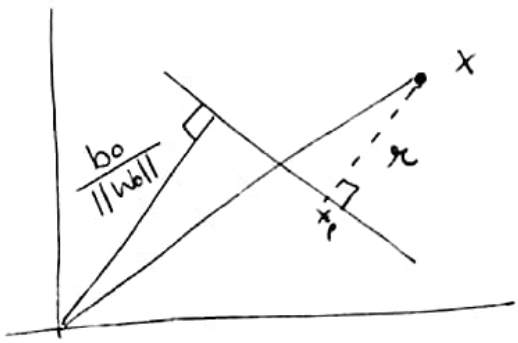
The discriminant function

$$g(x) = w_o^T x + b_o \quad -(4)$$

gives an algebraic measure of the distance from x to the optimal hyperplane.

$$x = x_p + r\frac{w_o}{\|w_o\|} \quad -(5)$$

x_p is normal projection of x onto the optimal hyperplane & r is the desired algebraic distance; r is +ve if x is on the positive side of the optimal hyperplane & -ve if x is on the negative side.



by definition

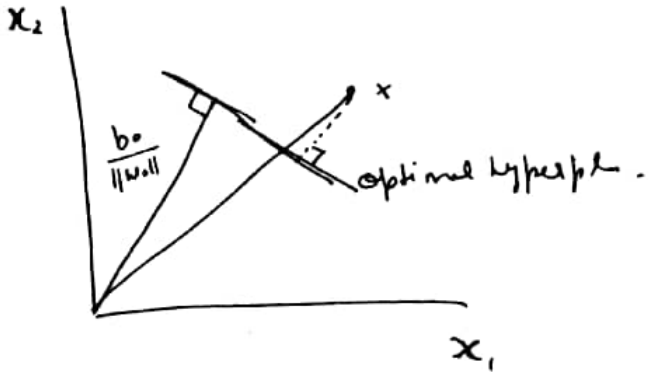
$$g(x_p) = 0$$

$$g(x) = w_o^T x + b_o = r\|w_o\|$$

$$r = \frac{g(x)}{\|w_o\|}$$

distance from origin (i.e; $x=0$) to optimal hyperplane is given by $b_o/\|w_o\|$. If $b_o > 0$; origin is on the side of optimal hyperplane; if $b_o < 0$; it is on -ve side.

If $b_o = 0$; optimal hyperplane passes through origin



find parameters w_o & b_o for optimal hyperplane given the training set $\mathcal{T} = \{(x_i, d_i)\}$.

(w_o, b_o) must satisfy

$$w_o^T x_i + b_o \geq 1 \quad \text{for } d_i = +1$$
$$w_o^T x_i + b_o \leq -1 \quad \text{for } d_i = -1 \qquad \text{— (6)}$$

The data points (x_i, d_i) for which first or second line of (6) is satisfied with equality sign are called support vectors. Hence "SVM"

Support Vector Machine

All remaining examples in training samples are completely irrelevant. ∵ of their distinct property. Support vectors play a prominent role in the operation of this class of learning machines.

(S.V.) are those data points that lies closest to the optimal hyperplane and are therfor most difficult to classify.

Consider (S.V.) $x^{(s)}$ for which $d^{(s)} = +1$.

Then $$g(x^{(s)}) = w_o^T x^{(s)} + b_o = \mp 1 \quad (7)$$
for $d^{(s)} = \mp 1$

(5); algebraic distance from support vector $x^{(s)}$ to the optimal hyperplane is

$$r = \frac{g(x^{(s)})}{\|w_o\|}$$

$$= \begin{cases} \frac{1}{\|w_o\|} & \text{if } d^{(s)} = +1 \quad - \text{+ve side of hyperplane} \\ -\frac{1}{\|w_o\|} & \text{if } d^{(s)} = -1 \quad - \text{-ve side of hyperplane} \end{cases}$$

$\rho \rightarrow$ optimum value of margin of separation b/w 2 classes that constitute the training sample of. Then $\rho = 2r = \frac{2}{\|w_o\|}$

Maximizing the margin of separation b/w binary classes is equivalent to minimizing the Euclidean norm of the weight vector W.

The training sample $\mathcal{T} = \{x_i, d_i\}_{i=1}^{N}$

expressed as

$$w_o^T x_i + b_o \geq 1 \quad \text{for } d_i = +1$$

$$w_o^T x_i + b_o \leq -1 \quad \text{for } d_i = -1$$

written as

$$d_i (w^T x_i + b) \geq 1 \rightarrow \text{constraint.}$$

What we want to do.

Maximize $\rho = \frac{2}{\|w\|}$

minimize $\|w\|$

One can formulate as Convex optimization problem.

$$\min \frac{1}{2}\|w\|^2$$

$$\phi(w) = \frac{1}{2} w^T w$$

$\frac{1}{2}\|w\|^2$ is used instead of $\|w\|$ ∵ it simplifies the derivative in optimization process

$$\text{S.t.} \quad d_i (w^T x_i + b) \geq 1$$

As we have seen earlier, one can solve this using method of Lagrange multipliers

$$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i \left[d_i (w^T x_i + b) - 1 \right]$$

$\alpha_i \rightarrow$ Lagrange multiplier

some place written λ_i

Solution to this constrained-optimization problem is determined by the saddle point of Lagrangian fn. $L(w, b, \alpha)$.

Saddle point has to be minimized w.r.t. w and b.
it also has to be maximized w.r.t. α

$$\frac{\partial L}{\partial w} = 0$$

$$w - \sum_{i=1}^{N} \alpha_i d_i x_i = 0$$

$$w = \sum_{i}^{N} \alpha_i d_i x_i \quad \text{--- (a)}$$

$$\frac{\partial L}{\partial b} = 0$$

$$-\sum_{i=1}^{N} \alpha_i d_i = 0$$

$$\sum_{i=1}^{N} \alpha_i d_i = 0 \quad \text{--- (b)}$$

Important to note that for all constraints that are not satisfied as equalities; correspondiy α_i must be zero.

Only those multipliers that exactly satisfy condition

$$\alpha_i \left[d_i (w^T x_i + b) - 1 \right] = 0$$

can assume nonzero values

If you check this property is statement of Karush-Kuhn-Tucker condition

$$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum \alpha_i \left[d_i (w^T x_i + b) - 1 \right]$$

(4) $\alpha = n$

→ put (a) & (b)

→ Possible to construct another problem called the dual problem.

→ Second problem has same optimal value as primal problem, but with Lagrange multipliers providing optimal soln..

→ In order for w_o to be an optimal primal solution and α_o to be optimal dual soln; it is necessary & sufficient that w_o is feasible for primal problem, and

$$\phi(w_o) = \cancel{L}(w_o, b_o, \alpha_o) = \min_w \cancel{L}(w, b, \alpha)$$

$$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_i^N \alpha_i d_i w^T x_i - b \sum_i^N \alpha_i d_i + \sum_i^N \alpha_i$$

$$w^T w = \sum_i^N \alpha_i d_i w^T x_i = \sum_i^N \sum_j^N \alpha_i \alpha_j d_i d_j x_i^T x_j$$

$$= \frac{1}{2} \sum_{i=1}^N \alpha_i d_i w^T x_i - \sum_{i=1}^N \alpha_i d_i w^T x_i + \sum_{i=1}^N \alpha_i$$

$$= -\frac{1}{2} \sum_{i=1}^N \alpha_i d_i w^T x_i + \sum_{i=1}^N \alpha_i$$

$$Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j d_i d_j x_i^T x_j$$

Dual problem

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j x_i^T x_j \quad - (*)$$

Subject to

1) $\sum_{i=1}^{N} \alpha_i d_i = 0 \quad - (*)1$

2) $\alpha_i \geq 0$ for $i = 1, 2, \ldots, N \quad - (*)2$

The Dual problem (*) is cast entirely in terms of training data

Moreover, function $Q(\alpha)$ to be maximized depends only on input patters in form of a set of dot products

$$\{x_i^T x_j\}_{i,j=1}^{N}$$

Constraint (2) of dual problem is satisfied with inequality sign for all SV support vectors for which α are nonzero & equality sign for all other training data points

(*) is Quadratic programming problem.

Solve numerically.

- find values of α_i;

Then compute $w = \sum_{i=1}^{N} \alpha_i d_i x_i$

Determine the b.

For any support vector x_k (where $\alpha_k > 0$).

bias b can be found.

$$b = [redacted] - w \cdot x_k + d_k$$

$$= d_k - w \cdot x_k$$

→ After we know w & b;
one can simply classify a new point using

$$f(x) = \text{sign}(w \cdot x + b)$$

→ If data is not linearly separable, one introduces slack variables $\xi_i \geq 0$ to allow some misclassification.

One then modify objective fn to

$$\phi(w, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

C is regularization parameter controlling the trade-off b/w maximizing the margin & minimize the classification error.

C is viewed as reciprocal of "regularization"

When C is assigned larger value; designer of SVM has high confidence in quality of T (training sample)

If C is assigned lower value; T is considered to be noisy

Given the to

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

s.t. $d_i(w^T x_i + b) \geq 1 - \xi_i$ for $i = 1, 2, \ldots, r$

$\xi_i \geq 0$ for all i

One then formulate into $Q(\alpha)$ using dual problem

$$\max \; Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{r}\sum_{j=1}^{}\alpha_i\alpha_j d_i d_j X_i^T X_j$$

s.t. $$\sum_{i=1}^{r}\alpha_i d_i = 0$$

$$0 \leq \alpha_i \leq C \quad \text{for } i = 1, 2, \ldots, N$$

C is user-specified the parameter

Note that neither slack variable ξ_i nor their own Lagrange multipliers appear in dual problem.

Kernel trick

Nonlinear SVM uses Kernel trick to handle data which is not linearly separable.

One map the original ~~data~~ input space into a higher dimensional feature space.

This help in finding a hyperplane that will separate data in this new space

Certain datasets → linear decision / hyperplanes are not able to effectively separate classes.

To address this, one map input data points $x \in \mathbb{R}^m$ into higher-dimensional feature space using transformation

$$\phi \; \mathbb{R}^m \longrightarrow \mathbb{R}^M \qquad M > m.$$

In the dimensional feature

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

$$d_i(\phi(x_i) \cdot w + b) \geq 1$$

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[d_i(\phi(x_i) \cdot w + b) - 1 \right]$$

one can get dual problem.

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \phi(x_i) \cdot \phi(x_j)$$

$$s.t. \quad \alpha_i \geq 0 \; ; \quad \sum_{i=1}^{N} \alpha_i d_i = 0$$

→ Introducing kernel function

Feature mapping $\phi(x)$ may be complex or high-dimensional making $\phi(x_i) \cdot \phi(x_j)$ difficult to compute.

Instead one use Kernel function

$K(x_i, x_j)$ → computes dot product in feature space without explicitly calculating $\phi(x)$

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j K(x_i, x_j)$$

$$\alpha_i \geq 0 \; ; \; \sum_{i=1}^{N} \alpha_i d_i = 0.$$

Decision function

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i d_i K(x_i, x) + b \right)$$

$K(x_i, x)$ computes inner product in higher dimensional feature space

→ Some of common Kernel function are

1) Linear Kernel $K(x_i, x_j) = x_i \cdot x_j$ (for linear SVM)

2) Polynomial Kernel

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^p \; ; \; p \rightarrow \text{degree of polynomial}$$

3) Gaussian (RBF) Kernel. Radial Basis function

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2} \right)$$, σ is parameter specified apriori by user

4) Sigmoid Kernel

$$K(x_i, x_j) = \tanh(K \cdot x_i \cdot x_j + C) \; ;$$

mimics behaviour of neural network

Neuron

fundamental building block of artificial neural network., inspired by biology.

Structure of Neuron

1) Inputs $(x_1, x_2, x_3, \ldots x_n)$

Each input is associated with a weight.

$(w_1, w_2, w_3, \ldots, w_n)$

weight influence how much importance neuron apply to or assign to the input.

During training, weights are adjusted to minimize error of network.

2) Bias (b) → additional parameter added to input; which allow activation function to be shifted to left or right.

helps in model to fit data

3) Weighted Sum.

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

z is passed to activation function

Activation function

applied to weighted sum to introduce non linearity to model.

helps network learn complex patterns in data

$$\text{Sigmoid}(z_i) = \frac{1}{1+e^{-z}}$$

output 0 to 1; useful for binary classification

ReLU (Rectified Linear Unit)

$$\text{ReLU}(z) = \max(0, z)$$

outputs the input directly if it is positive otherwise ~~output~~ output zero.

widely ~~hidden~~ hidden in layers of neural network.

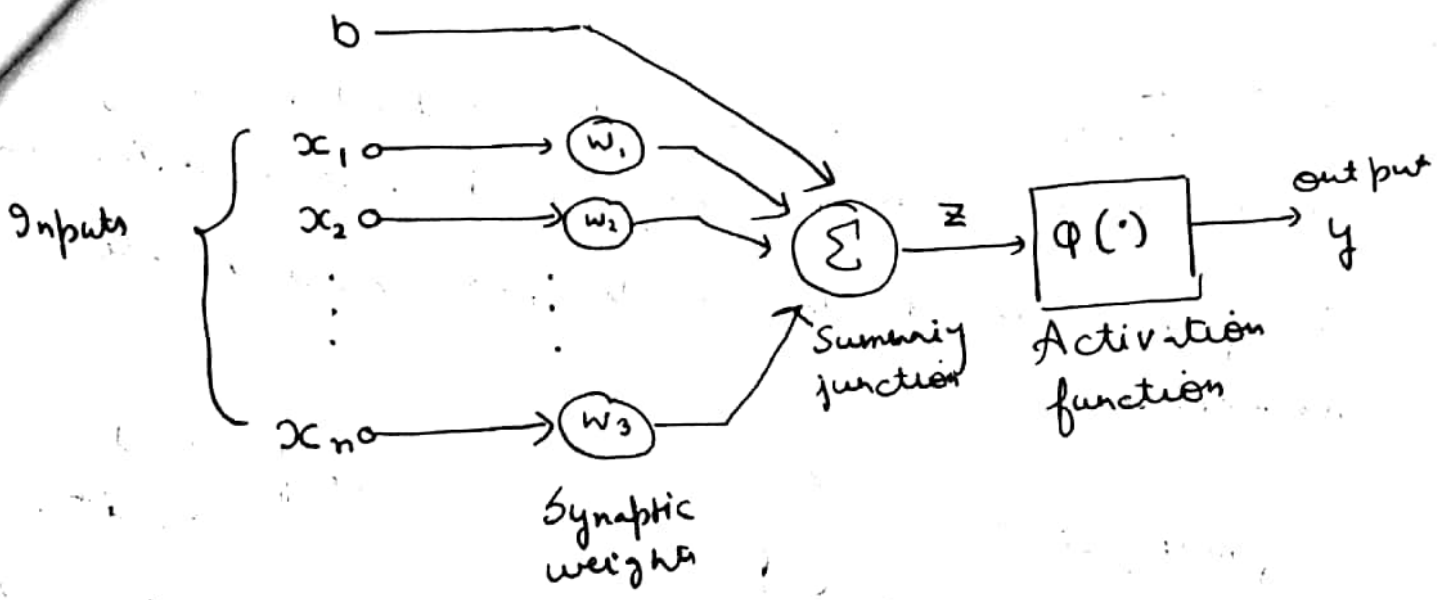
$[0, \infty]$ →

Tanh →

$$\text{Tanh}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

-1 to 1

Stronger gradient than Sigmoid



ReLU (Rectified Linear Unit)

$$\text{ReLU}(z) = \max(0, z)$$

Introduce non-linearity into model

→ output zero for -ve input + input itself for +ve

Leaky ReLU

$$\text{Leaky ReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$$

allow small, non-zero gradient for -ve inputs to prevent neuron from dying (ie. giving zero for any input)

α is small constant

often 0.01

Softmax

→ activation function used in output layer of neural network when performing multi-class classification

Transforms raw outputs (logits) from network into probabilities.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

e → base of natural logarithm.

z_i → i^{th} raw score (logit)

n → no. of classes.

Suppose raw output of (logits) from 3 neurons in output layer is

$$z = [2.0, 1.0, 0.1]$$

→ $$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{3} e^{z_j}}$$

① $$\text{Softmax}(z_1) = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = 0.659$$

② $$\text{softmax}(z_2) = \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = 0.242$$

③ $$\text{Softmax}(z_3) = \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} = 0.099$$

Choose the highest probability from Softmax output

0.659 ① class

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

useful for binary classification

Example

Input

$x = [1.5, -2.0, 3.0, -4.5, 2.0]$

$w = [0.5, -0.3, 0.8, -0.6, 0.2)$

$b = 1.0$

$z = \sum_{i=1}^{5} w_i x_i + b$

$= 7.85$

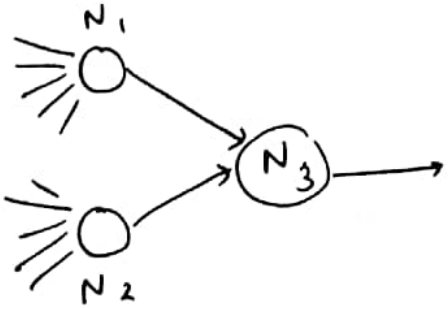
Apply activation

$$\text{Leaky Relu}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$$

$= 7.85$

$$\text{Sigmoid}(7.85) = \frac{1}{1+e^{-7.85}} = \frac{1}{1+0.000388}$$

≈ 0.9996



Apply same input 2 two neurons & then combine with by another neuron.

$$x = [1.5, -2.0, 3.0, -4.5, 2.0]$$

N_1

$w_1 = [0.5, -0.3, 0.8, -0.6, 0.2)$ $\quad \alpha = -0.01$

$b_1 = 1.0$

N_2

$w_1 = [-0.4, 0.7, -0.5, 0.9, -0.2)$

$b_2 = -0.5)$

$N_1 \rightarrow z_1 = 7.85$, Leaky ReLU$(z_1) = 7.851$

$N_2 = z_2 = -8.45$, LReLU$(z_2) = -0.08$

$N_3 \rightarrow w_3 [0.6, -0.9)$

$b_3 = 0.5$

$z_3 = w_{31} \cdot$ output of N1 $+ w_{32} \cdot$ output of $N_2 + b_3$

$= 5.28605$

Sigmoid $z_3 = \frac{1}{1+e^{-z_3}} \approx 0.99496$

if we apply Softmax.

Softmax(Z_1) =

Subtract max. value from each output before taking exponential to prevent numerical overflow

$Z_1 = \max(7.85, -0.0845) = 7.85$

for N1 = $7.85 - 7.85 = 0$

N2 = $-0.0845 - 7.85 = -7.9345$

$e^0 = 1$ $\quad e^{-7.9345} \approx 0.000357$

$$\text{Softmax}(Z_1) = \frac{e^0}{e^0 + e^{-7.93}} = 0.9996$$

$$\text{Softmax}(Z_2) = \frac{e^{-7.9345}}{1.00035} \approx 0.00036$$

Output from N_1 → higher than N2.

higher probability for N_1

Backpropogation

① help in updating weights & bias.

x → input, weight → w

bias b

$y_{pred} = xw + b = z$

y_{true}.

We use Mean Squared Error (MSE) as loss function

$$Loss = \frac{1}{2}(y_{pred} - y_{true})^2$$

② Backward pass (compute gradient)

① $\frac{\partial Loss}{\partial y_{pred}} = y_{pred} - y_{true}$

② $\frac{\partial y_{pred}}{\partial z}$

as $y_{pred} = z \Rightarrow \frac{\partial y_{pred}}{\partial z} = 1$

③ $\frac{\partial z}{\partial w} = x$

④ $\frac{\partial z}{\partial b} = 1$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial z} * \frac{\partial z}{\partial w}$$

$$\frac{\partial Loss}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial z} * x$$

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial z} * \frac{\partial z}{\partial b} = \frac{\partial Loss}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial z}$$

(3) update weight & bias using gradient descent

(1) Learning rate η.

(2) Update weight

$$W_{new} = W - \eta * \frac{\partial Loss}{\partial W}$$

(3) update bias

$$b_{new} = b - \eta * \frac{\partial Loss}{\partial b}$$

One simply repeat the process in subsequent training iteration to continue minimize error.