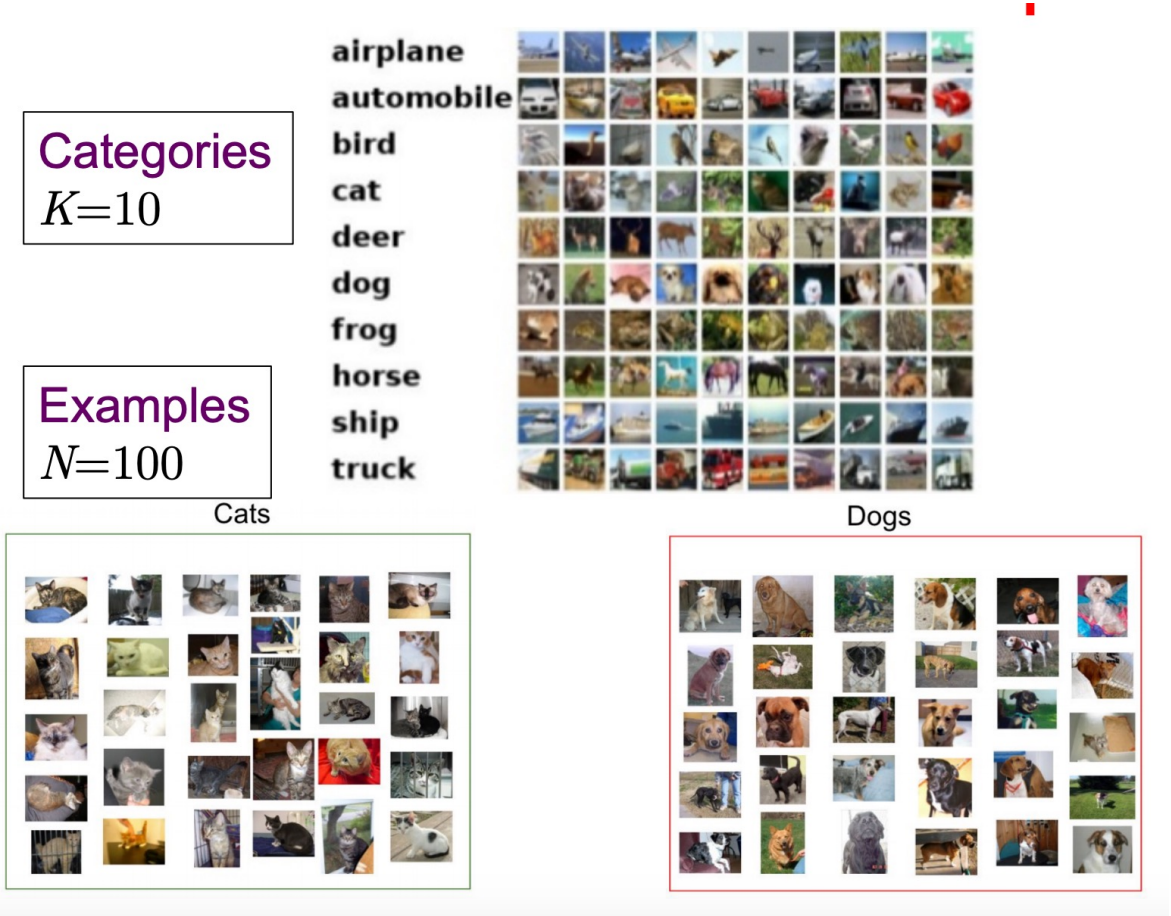
COMP [56]630– Machine Learning

Lecture 8 – Multiclass LR, Evaluation Metrics

Multi-class problems

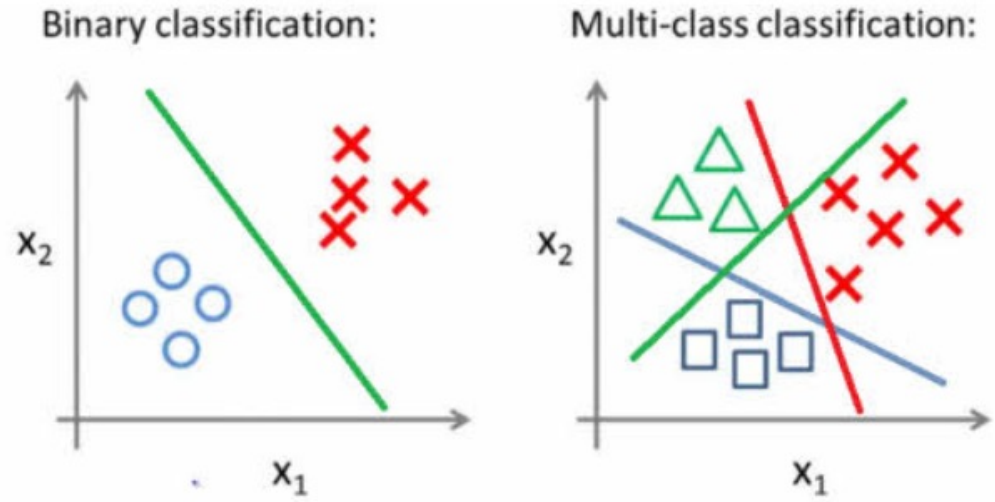
Multi-class classification



Credits to Dr. Srihari @ Buffalo

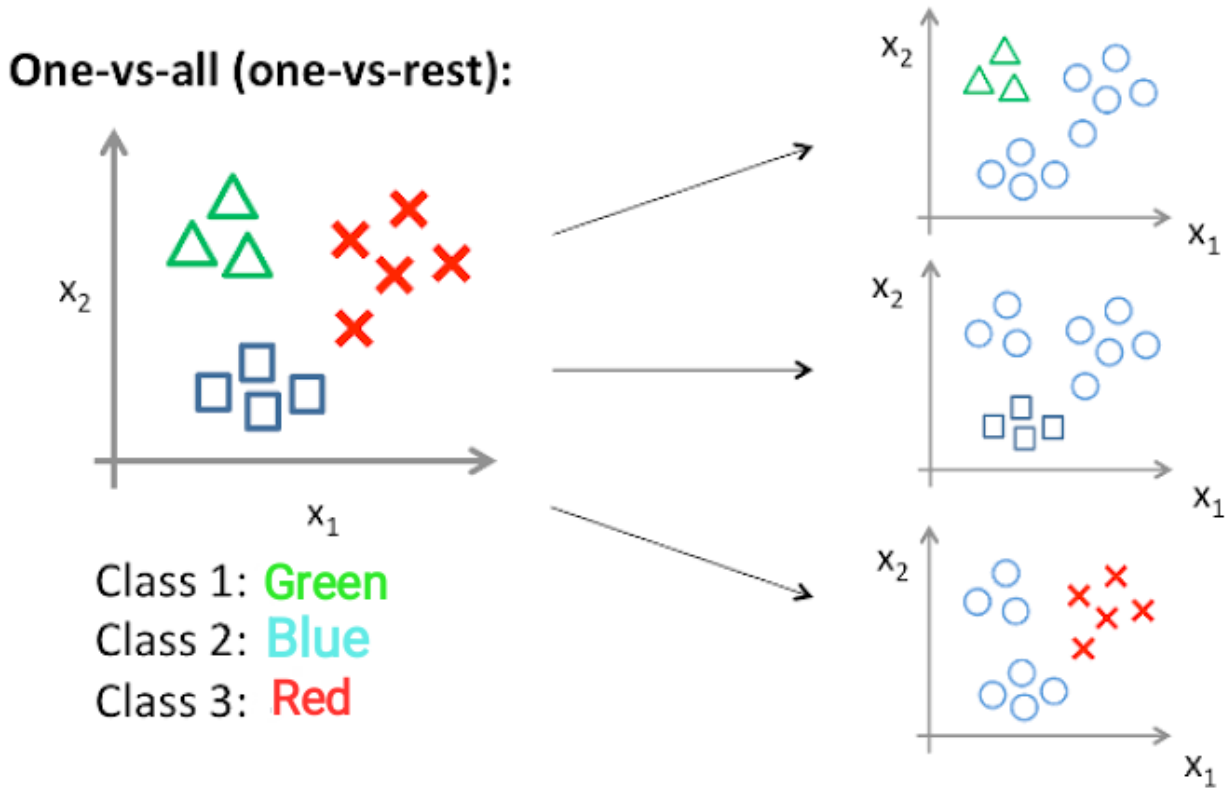
Binary vs Multi-class Classification

- More than two classes (binary) in our problem setting.
- Logistic regression returns a confidence score that scores the probability of whether an example is positive class or negative class.
- Binary classification requires only one classifier model.



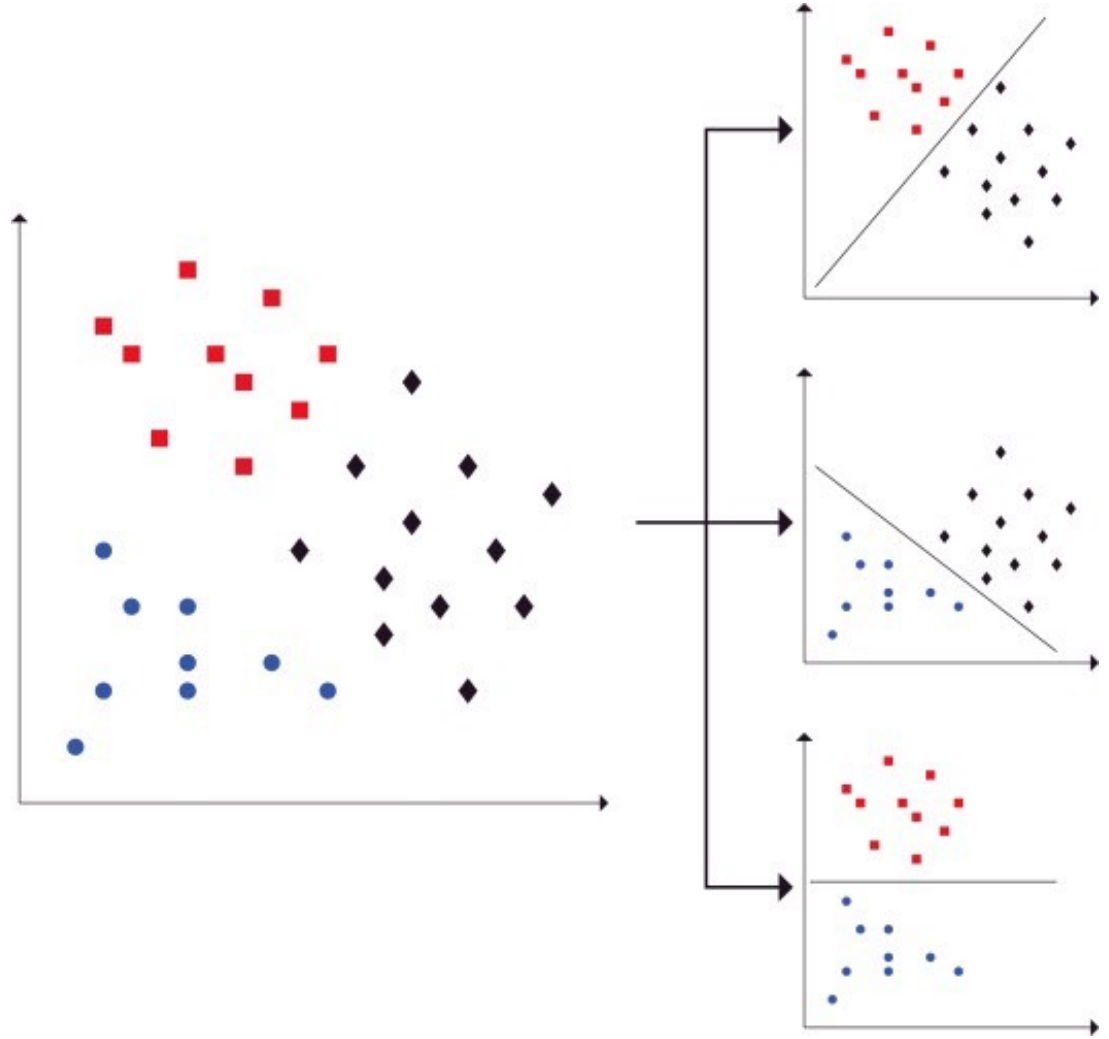
One-vs-all

- Create the N-binary classifier models, one for each class in the data.
- The number of class labels present in the dataset and the number of generated binary classifiers must be the same.
- Also called one-vs-rest classification



One-vs-one

- Create the **N* (N-1)/2** binary classifier models, one for each combination of classes in the data.
- Let us say there are three classes, A, B and C.
  - Create classifier for A-vs-B, A-vs-C, B-vs-C.
  - Each binary classifier predicts one class label.
  - Take the class with majority votes as final prediction



In the two-class case $p(C_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi}+b)$

where $\boldsymbol{\phi}=[\phi_1,..,\phi_M]^{\mathrm{T}}$, $\boldsymbol{w}=[w_1,..,w_M]^{\mathrm{T}}$ and $a=\boldsymbol{w}^T\boldsymbol{\phi}+b$ is the "activation"

For K classes, we work with soft-max function instead of logistic sigmoid (*Softmax regression*)

$$p(C_k \mid \boldsymbol{\phi}) = y_k(\boldsymbol{\phi}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

where $a_k=\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{\phi}+b_k, \quad k=1,..,K$

$\boldsymbol{w_k}=[w_{k1},..,w_{kM}]^{\mathrm{T}}$ and $\boldsymbol{a}=\{a_1,..a_K\}$

– We learn a set of K weight vectors $\{\boldsymbol{w}_1,..,\boldsymbol{w}_K\}$ and biases $\boldsymbol{b}$

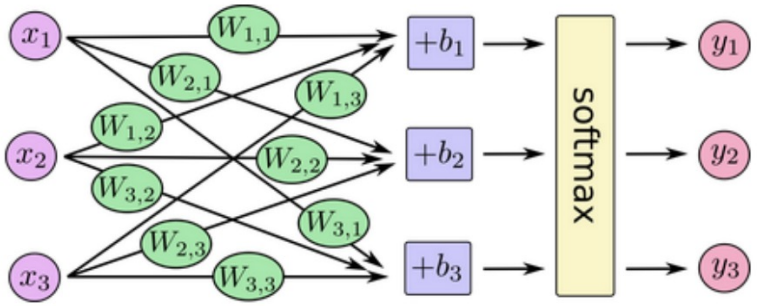
Arranging weight vectors as a matrix W

$$\boldsymbol{a} = W^T\boldsymbol{\phi}+\boldsymbol{b}$$

$$W = \begin{bmatrix} \boldsymbol{w}_1 \\ \cdot \\ \cdot \\ \boldsymbol{w}_K \end{bmatrix} = \begin{bmatrix} w_{11} & & w_{1M} \\ \cdot & & \\ \cdot & & \\ w_{K1} & & w_{KM} \end{bmatrix}$$

$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{a})$$
$$y_i = \frac{\exp(a_i)}{\sum_{j=1}^{3}\exp(a_j)}$$

Multi-class Logistic Regression



$$\boldsymbol{a} = W^T\boldsymbol{x} + \boldsymbol{b}$$

$$W = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix}$$

$$\boldsymbol{y} = \text{softmax}(\boldsymbol{a})$$

$$y_i = \frac{\exp(a_i)}{\sum_{j=1}^{3} \exp(a_j)}$$

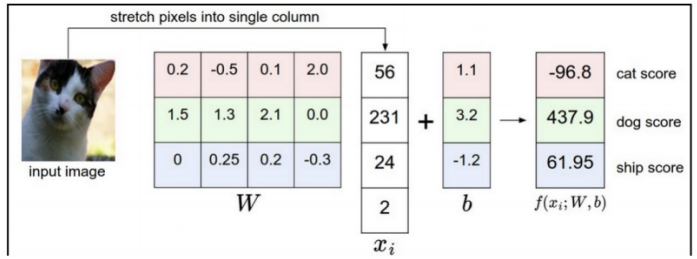
Network Computes

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

In matrix multiplication notation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

An example



We use maximum likelihood to determine the parameters $\{\boldsymbol{w}_k\}$, $k=1,..K$

The exp within softmax $\boxed{\text{softmax}(\boldsymbol{a})_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}}$ works very well when training using log-likelihood

- Log-likelihood can undo the exp of softmax

$$\boxed{\log \text{softmax}(\boldsymbol{a})_i = a_i - \log \sum_j \exp(a_j)}$$

- Input a_i always has a direct contribution to cost
  - Because this term cannot saturate, learning can proceed even if second term becomes very small
- First term encourages a_i to be pushed up
- Second term encourages all $\boldsymbol{a}$ to be pushed down

Derivatives

The multiclass logistic regression model is

$$p(C_k \mid \phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

For maximum likelihood we will need the derivatives of y_k wrt all of the activations a_j

These are given by

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

– where I_{kj} are the elements of the identity matrix

One-hot targets

Classes $C_1,..C_K$ represented by 1-of-K scheme

- One-hot vector:
  - class C_k is a K-dim vector or $[t_1,..,t_K]^{\mathrm{T}}$, $t_i \varepsilon \{0,1\}$
    - With K=6, class C_3 is $(0,0,1,0,0,0)^T$ with t_1=t_2=t_4=t_5=t_6=0, & t_3=1
- The class probabilities obey

$$\boxed{\sum_{k=1}^{K} p(C_k) = \sum_{k=1}^{K} t_k = 1}$$

- If $p(t_k{=}1) = \mu_k$ then

$$p(C_k) = \prod_{k=1}^{K} \mu_k^{t_k} \text{ where } \boldsymbol{\mu} = (\mu_1,..,\mu_K)^T$$

e.g., probability of C_3 is

$$p([0,0,1,0,0,0]) = \mu_3$$

Why use one-hot representation?
If we used numerical categories $1,2,3\ldots$we would impute ordinality. We can now use simpler Bernoulli instead of multinoulli

Target Matrix

Classes have values $1, .., K$

Each represented as a K-dimensional binary vector

We have N labeled samples

- So instead of target vector $\boldsymbol{t}$ we have a target matrix T

Classes →

$$T = \begin{bmatrix} t_{11} & \cdot & \cdot & t_{1K} \\ \cdot & & & \\ \cdot & & & \\ t_{N1} & & & t_{NK} \end{bmatrix}$$

↓ Samples

Note that t_{nk} corresponds to sample n and class k

Objective Function & Gradient

Likelihood of observations $T = \begin{bmatrix} t_{11} & \cdot & \cdot & t_{1K} \\ \cdot & & & \\ \cdot & & & \\ t_{N1} & & & t_{NK} \end{bmatrix}$ is

$$p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(C_k \mid \boldsymbol{\phi}_n)^{t_{n,k}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

- Where, for feature vector $\boldsymbol{\phi}_n$ $y_{nk} = y_k(\boldsymbol{\phi}_n) = \dfrac{\exp(\boldsymbol{w}_k^T \boldsymbol{\phi}_n)}{\sum_j \exp(\boldsymbol{w}_j^T \boldsymbol{\phi}_n)}$

Objective Function: negative log-likelihood

$$E(\boldsymbol{w}_1,...,\boldsymbol{w}_K) = -\ln p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

– Known as cross-entropy error for multi-class

Gradient of error function wrt parameter $\boldsymbol{w}_j$

$$\nabla_{\mathrm{w}_j} E(\boldsymbol{w}_1,...,\boldsymbol{w}_K) = -\sum_{n=1}^{N} (y_{nj} - t_{nj})\phi_n$$

using $\sum_k t_{nk} = 1$

Error x Feature Vector

$$y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \qquad a_k = \boldsymbol{w}_k^{\mathrm{T}} \phi$$

$\dfrac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$ where I_{kj} are elements of the identity matrix

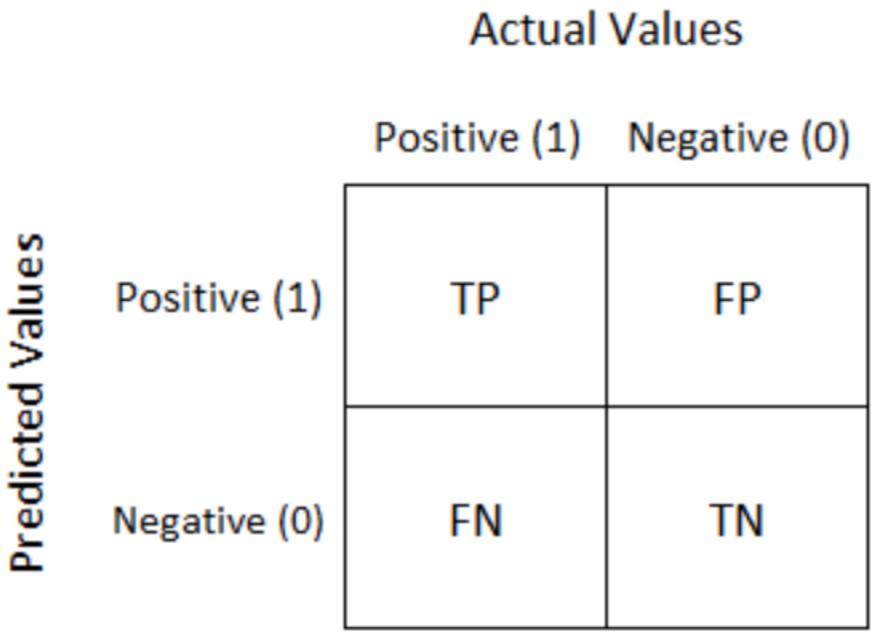
Performance of Classifiers

Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.

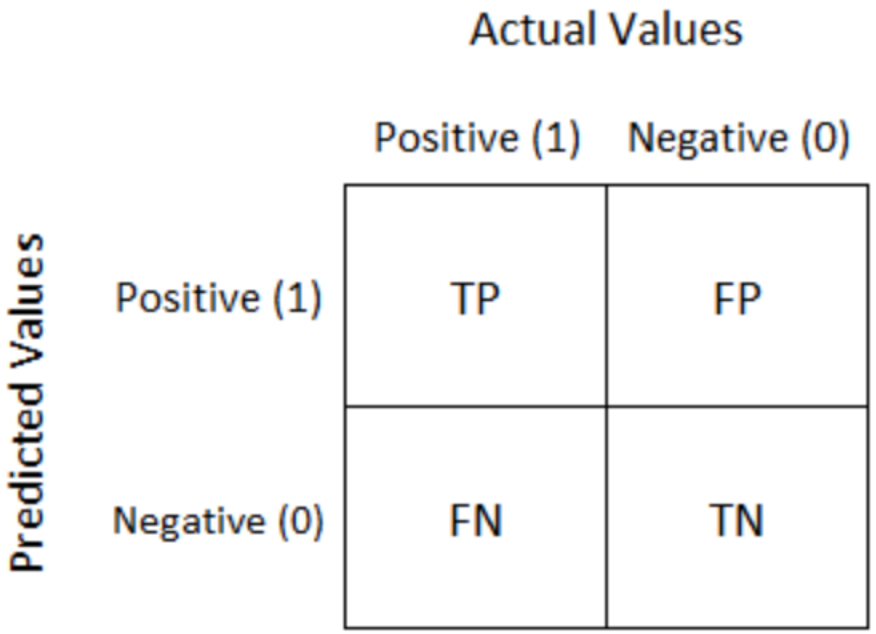
Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.



Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.
- Models Four major characteristics
- **true positives (TP)**
- **true negatives (TN)**
- **false positives (FP)**
- **false negatives (FN)**



Perf. Measures from Confusion Matrix

- **Accuracy:** Overall, how often is the classifier correct?
  - (TP+TN)/total
- **Misclassification Rate:** Overall, how often is it wrong?
  - (FP+FN)/total
- **Recall:** When it's actually yes, how often does it predict yes?
  - also known as "Sensitivity"

$$Recall = \frac{TP}{TP + FN}$$

- **Precision:** When it predicts yes, how often is

$$Precision = \frac{TP}{TP + FP}$$

Perceptron Algorithm

The Perceptron

- Consider modifying the logistic regression method to “force” it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of g to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- If we then let h(x) = g(θT x) as before but using this modified definition of g, and if we use the update rule

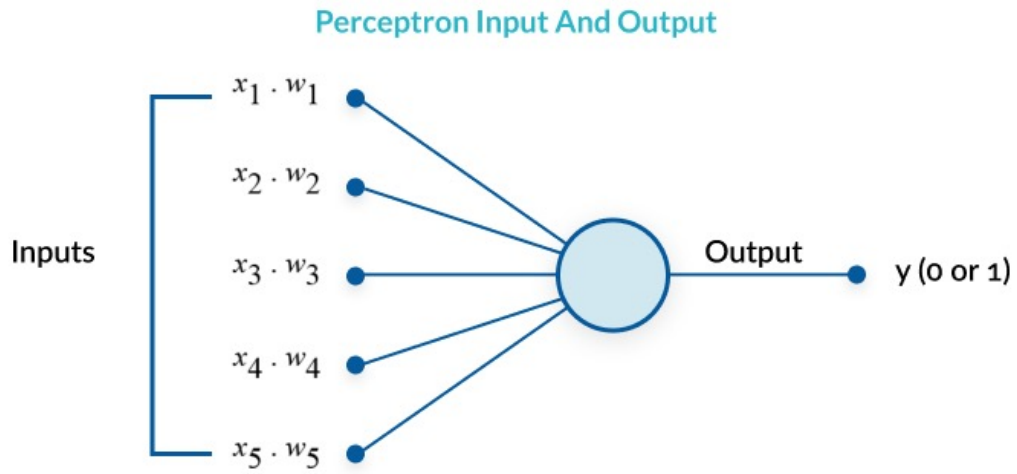
$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Perceptron

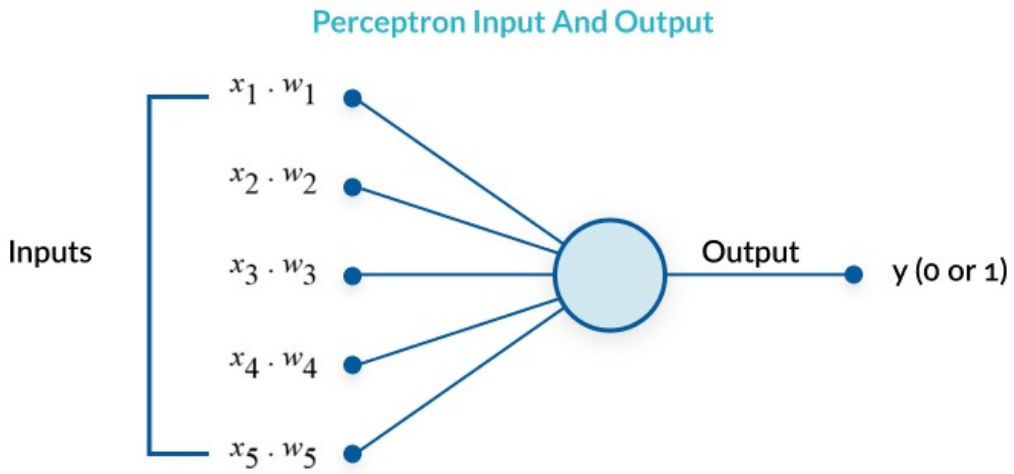
- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a step function can of course not be interpreted as any kind of probability.
  - Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.

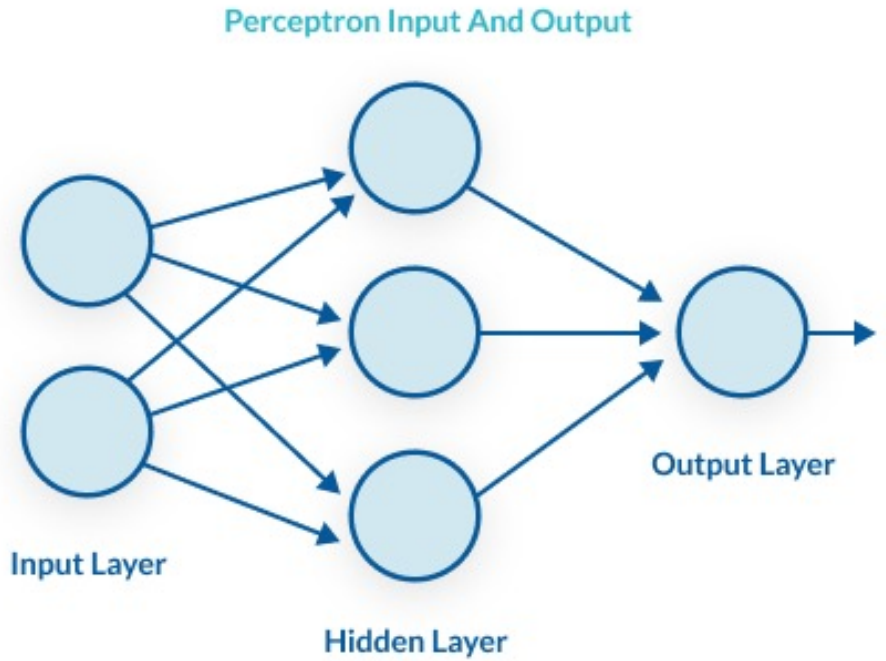
Multi-layer perceptrion

Perceptron vs Multilayer Perceptron (MLP)

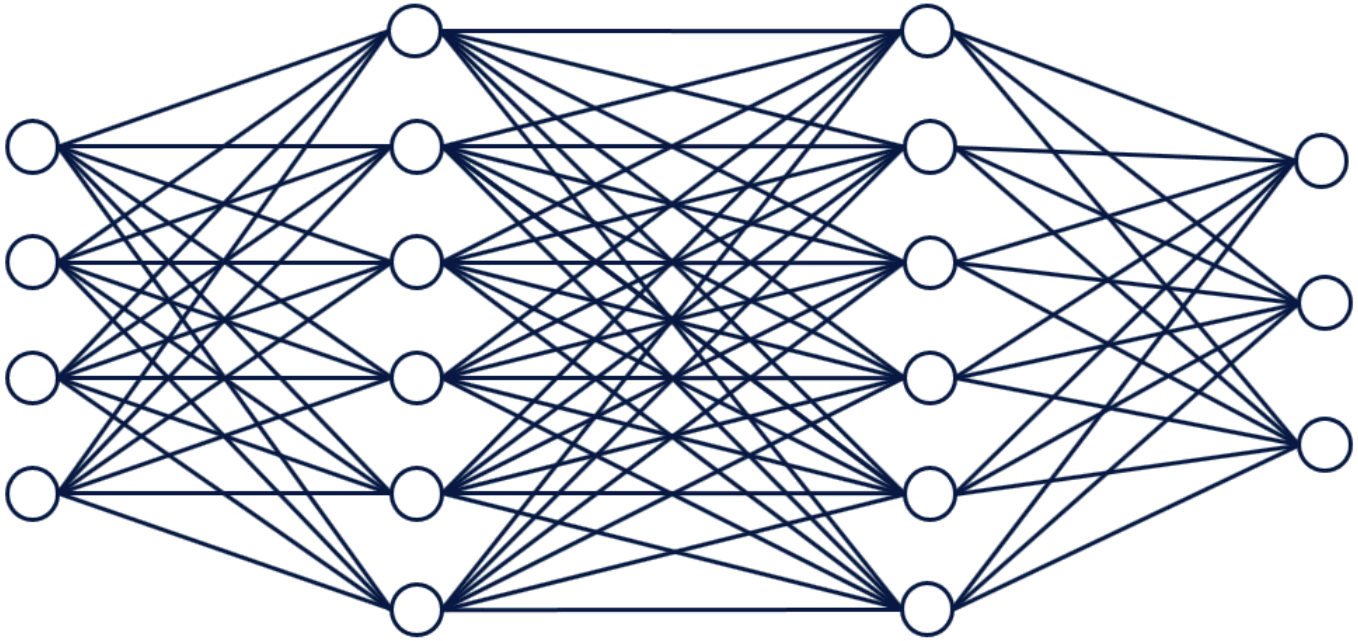


Perceptron vs Multilayer Perceptron (MLP)

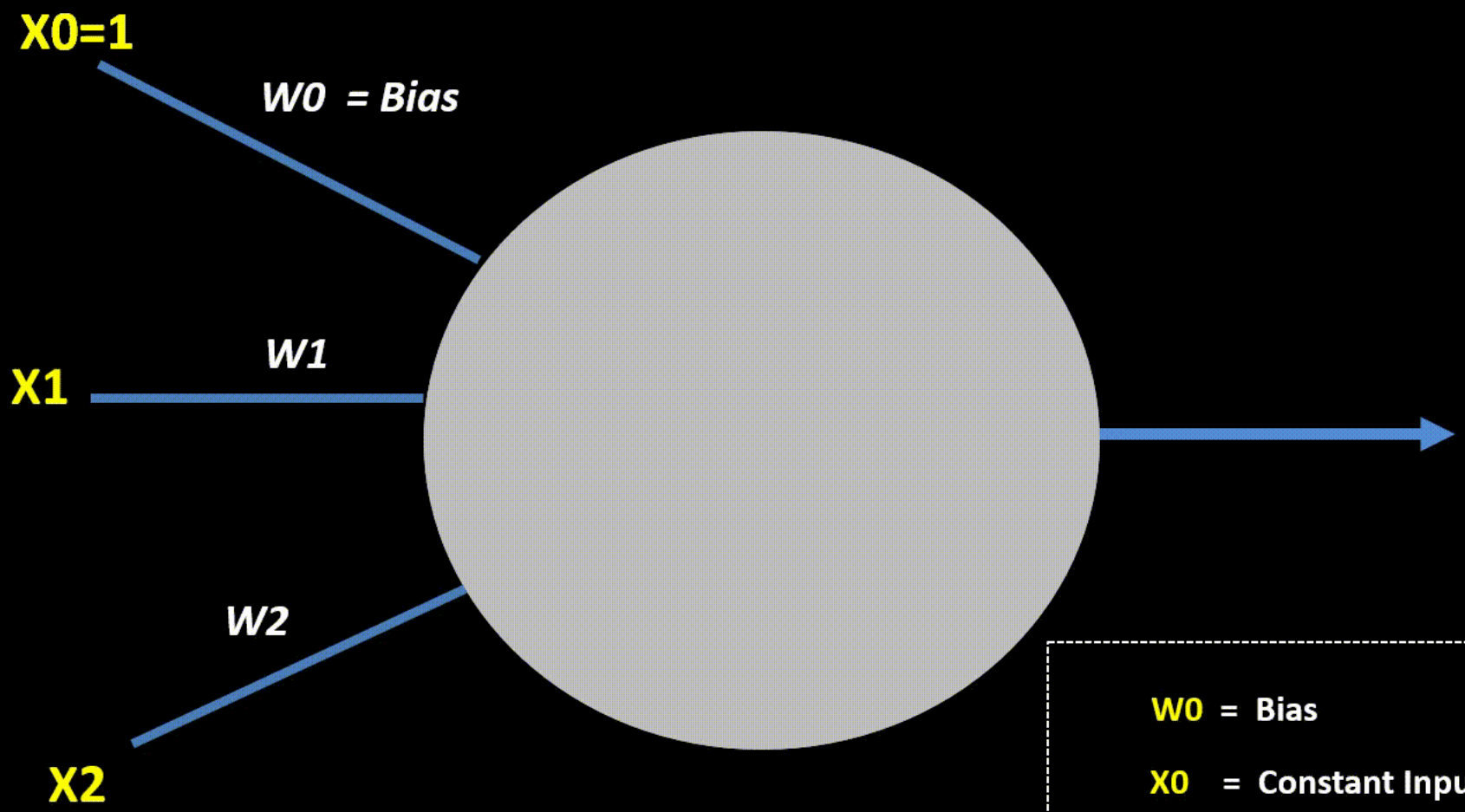




Why do we need more layers?



Artificial Neuron



W0 = Bias

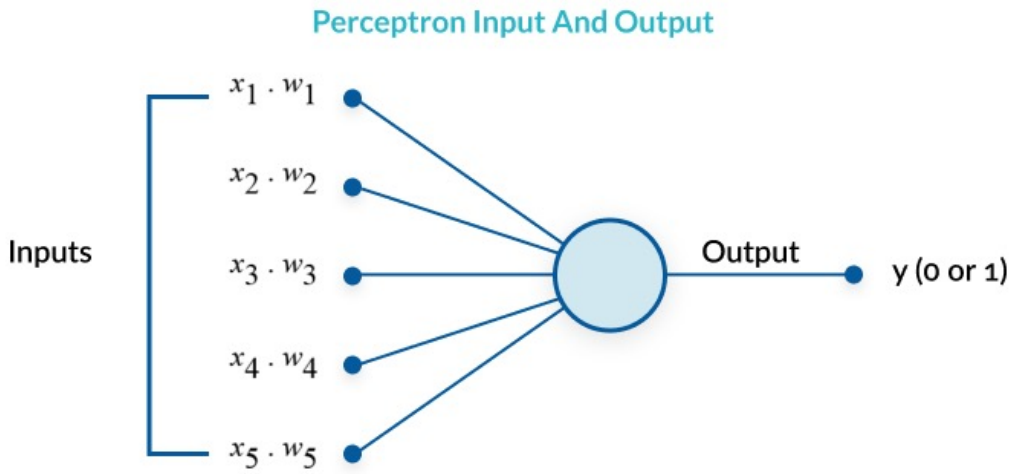
X0 = Constant Input 1 for Bias

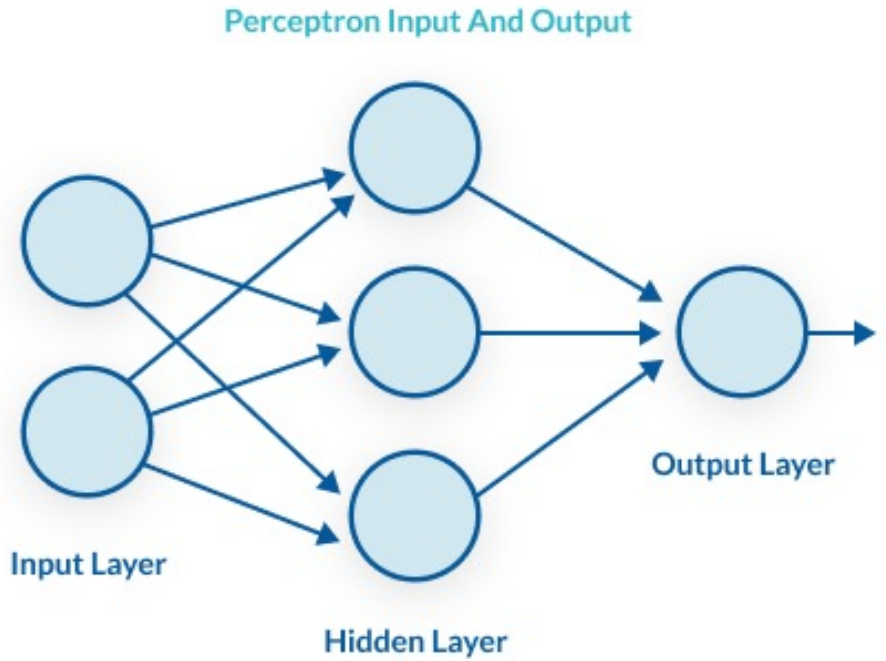
X1,X2 = Attribute Inputs

W1,W2 = Weights of Inputs X1,X2

machinelearningknowledge.ai

Perceptron vs Multilayer Perceptron (MLP)





Feed forward Neural Networks

A neural network can also be represented similar to linear models but basis functions are generalized

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

activation function

For regression: identity function

For classification: a non-linear function

Coefficients w_j adjusted during training

There can be several activation functions

Basis functions

$\phi_j(\boldsymbol{x})$ a nonlinear function of a linear combination of D inputs

its parameters are adjusted during training

Activation Functions

- Each activation a_j is transformed using differentiable nonlinear activation functions $z_j = h(a_j)$
- The z_j correspond to outputs of basis functions $\varphi_j(x)$
  - or first layer of network or hidden units
- Nonlinear functions h
- Three examples of activation functions:

1. Logistic sigmoid

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

2. Hyperbolic tangent

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

3. Rectified Linear unit

$$f(x) = \max(0, x)$$

Activation Function

- Determined by the nature of the data and the assumed distribution of the target variables
- For standard regression problems the activation function is the identity function so that $y_k=a_k$
- For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that $y_k=\sigma(a_k)$
- For multiclass problems, a softmax activation function of the form:

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Visualization of activation functions

Forward Pass

- Output of a layer l is given by

$$z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$$

- Where W^l is the weights of the layer, b^l is the bias and a^l is the activation

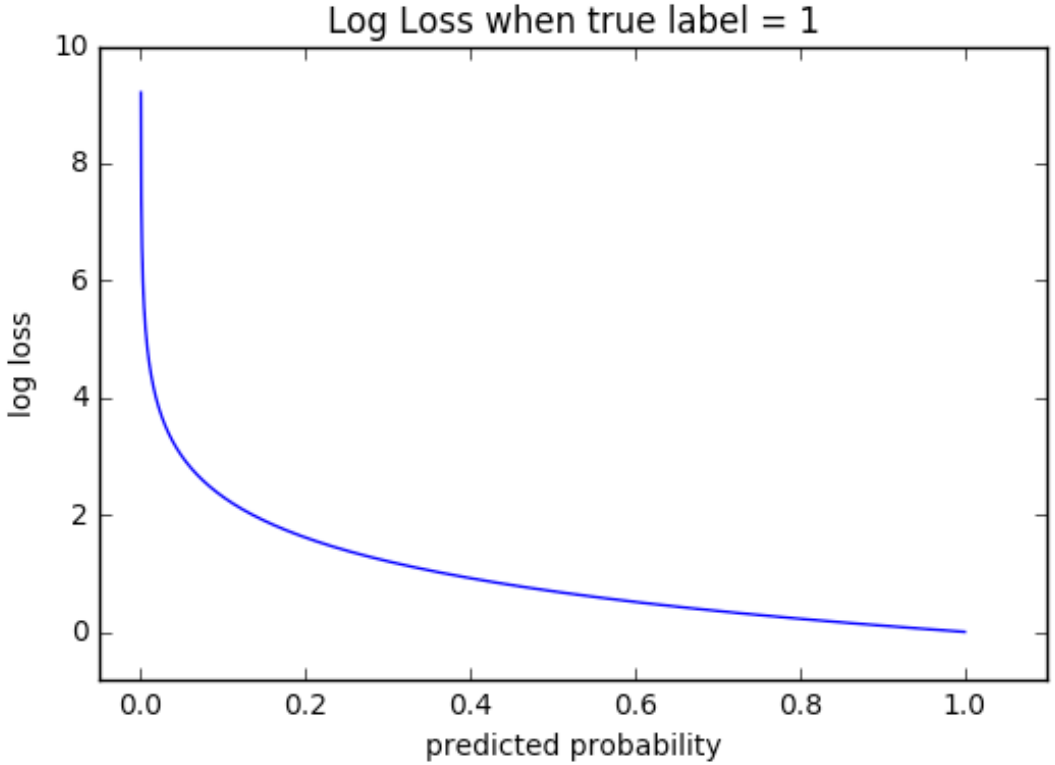
$$a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$$

Loss Function: Log loss

- Measures the performance of a classification model whose output is a probability value between 0 and 1.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

- Cross-entropy loss increases as the predicted probability diverges from the actual label.



How to train this network?

- Gradient Descent + Backprop!
- What is Backprop?
- Backprop or Backpropagation is a way to train multilayer neural networks using gradients.

Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero; N(0; 0:1))
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss L, in our case the log loss

$$\mathcal{L}(\hat{y}, y) = -\Big[(1-y)\log(1-\hat{y}) + y\log\hat{y}\Big]$$

Defining Variables

Let X be the input features . [n x f]
y be the output (target) labels [n x 1]

Define the weights [parameters] of the neural network

First layer → W_1 , bias → b_1
Second layer → W_2 , bias → b_2
Third layer → W_3 , bias → b_3

The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$
$$a_1 = g(z_1)$$ where "g" is the activation function

The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$
$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$
$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$

Defining the loss

Task : Binary classification

$\Rightarrow$ loss is log loss or binary cross entropy

$$L = -\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]$$

Learning with gradient descent

To learn the parameters

(1) Provide random values for weights w_1, w_2, w_3 & biases b_1, b_2, b_3

(2) Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{dL}{dw_i}$$

$$b_i := b_i - \alpha \cdot \frac{dL}{db_i}$$

where i refers to the ith layer.

(3) repeat until convergence

Finding dL/dW_3

To find $\frac{\partial L}{\partial w_3}$ to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3}\left[-\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]\right]$$

$$= -\left[(1-y)\cdot\frac{\partial}{\partial w_3}\left(\log(1-\hat{y})\right) + y\cdot\frac{\partial}{\partial w_3}\left(\log(\hat{y})\right)\right]$$

Finding dL/dW_3

Remember: $\frac{d}{dz}(\log(z)) = \frac{1}{z}$

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = -\left[\frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial}{\partial w_3}(1-\hat{y}) + \frac{y}{\hat{y}} \frac{\partial}{\partial w_3}(\hat{y})\right]$$

$$= -\left[\frac{-(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} + \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}\right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} - \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \left[\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}\right] \cdot \frac{\partial \hat{y}}{\partial w_3}$$

Finding dL/dW_3

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

Finding dL/dW_3

We know that $\hat{y} = a_3 = g(z_3)$

Finding dL/dW_3

if $g(z)$ is a sigmoid function,

$$g'(z) = g(z) \cdot (1 - g(z))$$

$$\therefore \frac{\partial \hat{y}}{\partial w_3} = \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3}$$

$$= g(z_3) \cdot (1 - g(z_3)) \cdot \frac{\partial z_3}{\partial w_3}$$

$$= a_3 (1 - a_3) \cdot \frac{\partial}{\partial w_3} [w_3 a_2 + b_3]$$

$$= a_3 (1 - a_3) \cdot a_2$$

$$= \hat{y} (1 - \hat{y}) \cdot a_2$$

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$

Finding dL/dW_2

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights W_2 and the loss L.
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite

Finding dL/dW_2

We know that Loss is dependent on $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{?} \cdot \frac{?}{\partial w_2}$$

we know that $a_3 = g(z_3)$

$\Rightarrow a_3$ is dependent on z_3

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

We know that $z_3 = w_3 a_2 + b_3$

$\Rightarrow z_3$ is dependent on a_2

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

But $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{?} \cdot \frac{?}{\partial w_2}$$

But $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$

Finding dL/dW_2

If we rewrite $\frac{dL}{dw_3}$ using chain rule,

$$\frac{dL}{dw_3} = \frac{dL}{da_3} \cdot \frac{da_3}{dz_3} \cdot \frac{dz_3}{dw_3}$$

$\frac{dL}{da_3} \cdot \frac{da_3}{dz_3} = a_3 - y$

$\frac{dz_3}{dw_3} = a_2$

Since $z_3 = w_3 a_2 + b_3$

Finding dL/dW_2

$$\Rightarrow \frac{dL}{dW_2} = (a_3 - y) \cdot \frac{dz_3}{da_2} \cdot \frac{da_2}{dz_2} \cdot \frac{dz_2}{dW_2}$$

If $z_3 = w_3 \cdot a_2 + b_3$

Then $\frac{dz_3}{da_2} = w_3$

if $a_2 = g(z_2)$

then $\frac{da_2}{d(z_2)} = g'(z_2)$

If $z_2 = w_2 \cdot a_1 + b_2$

then $\frac{dz_2}{dw_2} = a_1$

Finding dL/dW_2

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorization,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)$$

Finding dL/dW_1

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x$$

Ill-ly

$$\frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$