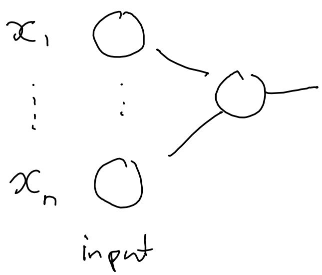
# Classification

## 1.1) Logistic regression

The perceptron we introduced before represents a linear classifier




$$= \sigma(\vec{x} \cdot \vec{\omega} + b)$$

where $\sigma(z) = \Theta(z)$ for a hard classifier and (e.g.) $\sigma(z) = \frac{1}{1+e^{-z}}$ for a soft classifier

hard $\rightarrow$ either 0 or 1

soft $\rightarrow$ probability the result is 0 or 1 it's convenient for notation to write

$$\vec{X} = (1, \vec{x}) \quad \text{and}$$

$$\vec{W} = (b, \vec{w})$$

such that the output is simply

$$\sigma(\vec{X} \cdot \vec{W})$$

Concretely we may write the probability that a result y (also called <u>score</u>) is 1 for a point $\vec{x}$ with the model $\vec{W}$ as

$$P(y=1 \mid \vec{x}, \vec{W}) = \frac{1}{1 + e^{-\vec{X} \cdot \vec{W}}}$$

and

$$P(y=0 \mid \vec{x}, \vec{W}) = 1 - P(y=1 \mid \vec{x}, \vec{W})$$

This can be seen clearly by analogy with a Boltzman distribution of a two state (non-interacting) system

$$\underbrace{p_i}_{\text{probability for state } i=0,1} = Q^{-1} \exp\left(-E_i/(kT)\right)$$

energy of state $i$

Boltzman constant times Temp.

where $$Q = \sum_{i=0}^{1} \exp\left(-E_i/(kT)\right)$$

$$p_1 = \frac{e^{-E_1/(kT)}}{e^{-E_0/(kT)} + e^{-E_1/(kT)}}$$

$$= \frac{1}{1 + \exp\left(-\underbrace{(E_0 - E_1)}_{\Delta E}/(kT)\right)}$$

and $p_0 = 1 - p_1$. So we see that

$\vec{x} \cdot \vec{w}$ plays the rôle of

$\Delta E/(kT)$.

We can write the likelihood that a model $\vec{w}$ matches a dataset

$$\mathcal{D} = \{ \vec{x}_i , y_i \} \qquad i = 1, \ldots, n$$

as

$$P(\{\vec{x}_i, y_i\} \mid \vec{w}) =$$

$$\prod_{i=1}^{n} \left( \sigma(\vec{x}\cdot\vec{w}) \right)^{y_i} \left( 1 - \sigma(\vec{x}\cdot\vec{w}) \right)^{1-y_i}$$

the loss function for this model is then $-\log(\text{likelihood})$

$$\mathcal{L}(\vec{w}) = -\sum_{i=1}^{n} y_i \log(\sigma(\vec{x}\cdot\vec{w})) + (1-y_i)\log(\sigma(\vec{x}\cdot\vec{w}))$$

which is known as the cross entropy.

As always the minimum of the loss function gives the optimal set of weights for our model. It is easy to compute

$$\frac{\partial \mathcal{L}}{\partial \vec{w}} = \sum_i \vec{x}_i \left( \sigma(\vec{x}_i \cdot \vec{w}) - y_i \right)$$

$$= \sum_i \vec{x}_i \left( \hat{y}_i - y_i \right)$$

using $\sigma(\vec{x}_i \cdot \vec{w}) = \hat{y}_i$.

From where we may now turn to gradient descent methods. $\boxed{\text{NB}}$

$\frac{\partial \mathcal{L}}{\partial \vec{w}} = 0$ is a complicated ~~set~~ set of transcendental equations so we must turn to numerical methods.

## 1.2) Soft Max regression

An obvious generalisation of the binary classifier is a discrete set of possible outcomes, $y = 0, 1, \ldots M$ (often refered to as <u>labels</u>)

The generalisation can be seen obtained by considering a Boltzman distribution for a multi-state system.

$$P_I = Q^{-1} \exp\left(-E_I / (kT)\right)$$

probability of a state $I$

$$Q = \sum_{I=1}^{M} \exp\left(-E_I / (kT)\right)$$

$$\Rightarrow \sum_{I=1}^{M} p_I = 1$$

translating to a model of weights $\vec{w}_J$ we write the probability of a state $I$ as

$$P\left(y = I \mid \vec{x}, \{\vec{w}_J\}\right) = \frac{e^{-\vec{x}\cdot\vec{w}_I}}{\sum_J e^{-\vec{x}\cdot w_J}}$$

$$= \frac{1}{1 + \sum_{J \neq I} \exp\left(-\vec{x}\cdot(\vec{w}_I - \vec{w}_J)\right)}$$

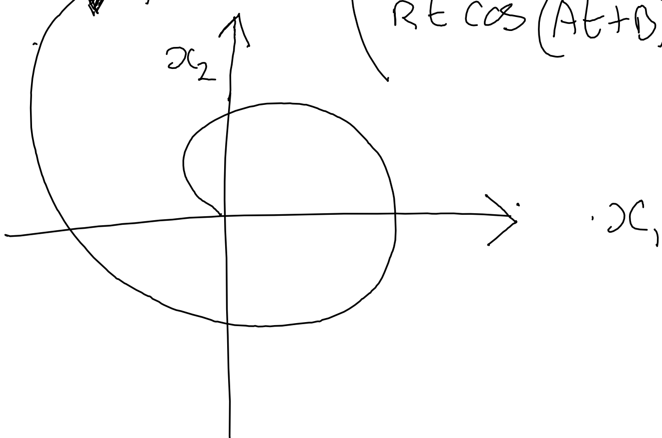
we may think of this as a vector of probabilities $\vec{P}\left(\vec{x}, \{\vec{w}_J\}\right)$.

We can understand the notation best by following an example.

## 1.2.1) Soft-Max regression example

[see jupyter notebook] We consider the case of a 2d distribution of points, each of which may have a colour red, blue or yellow. We generate data points that live on a spiral starting at the origin and generate 3 spiral "arms" of different colours,

$$\vec{f}(t) = \begin{pmatrix} Rt\sin(At+B) \\ Rt\cos(At+B) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$



data input $-$ $x_{\alpha i}$ ,

data output $-$ $y_\alpha$ ,

where $\alpha = 1, \dots, N$

$i = 0, 1$

$y_\alpha \in \{0, 1, 2\}$

Let's first try a <u>linear model</u>

$$Z_I = x \cdot W_I + b_I$$

where $I = (0, 1, 2)$

$$\sigma_I(Z_I(x)) = \frac{e^{-Z_I}}{\sum_{J=0,1,2} e^{-Z_J}}$$

The probabilities of an input value

$x$ corresponding to an output value

$I = (0, 1, 2)$ is just

$$P_I(x) = \sigma_I(z_I(x))$$

For our dataset for training, $x_i$,

the probabilities for the model $\vec{w}_I$, $b_I$

are

$$p_{\alpha I} = \sigma_I(z_I(x_\alpha))$$

where the <u>scores</u> $z$ are

$$z_{\alpha I} = x_{\alpha i} w_{iI} + b_I$$

$$\Rightarrow p_{\alpha I} = \frac{\exp(z_{\alpha I})}{\sum_J \exp(z_{\alpha J})}$$

There's a quick shortcut to the loss function, $\mathcal{L}$, from here

$$\mathcal{L}_\alpha = -\log\left( P_{\alpha(I=y_\alpha)} \right)$$

we may also add a <u>regularisation</u> loss coming from the model weights $W_{iI}$ to prevent preferring $\underbrace{\text{large weights}}$.

(therefore reducing overfitting)

$$\mathcal{L}^{reg} = \frac{\Gamma}{2} \sum_{i,I} W_{iI}^2$$

such that

$$\langle \mathcal{L} \rangle = \frac{1}{N} \sum_{\alpha=1}^{N} \mathcal{L}_\alpha + \mathcal{L}^{reg}$$

The gradient is computed from

$$d\mathcal{L}_{\alpha I} = \left( P_{\alpha I} - \delta_{\alpha(I=y_\alpha)} \right) \frac{1}{N}$$

where the updated parameters are computed from

$$dW_{iI} = x_{\alpha i}\, d\mathcal{Y}_{\alpha I}$$

$$dB_I = \sum_\alpha \mathcal{Y}_{\alpha I}$$

and
$$dW^{reg}_{iI} = r\, W_{iI} \longleftarrow \boxed{\frac{d}{dw_{iI}}\left(\frac{r}{2} w^2_{iI}\right)}$$

Now update via

$$W^{new}_{iI} = w_{iI} - \eta\left(dW_{iI} + dW^{reg}_{iI}\right)$$

$$B^{new}_I = B_I - \eta\, dB_I$$

See the Jupyter notebook for the implementation with a hidden layer.

## Application: Supersymmetry and Searches at Colliders

Supersymmetry (SUSY) was (is) a popular model for physics <u>beyond the Standard Model</u> (BSM).

This proposal introduces partners for every particle in the SM through symmetry between bosons (integer spin) and fermions ($\frac{1}{2}$-integer spin).

It may seem a fairly innocuous extension but the underlying principle is a <u>unique</u> extension of the

## Poincaré group for spacetime.

## Poincare algebra in SO(1,3)

$P^\mu$ - translations

$M^{\mu\nu}$ - Lorentz boosts + rotations

satisfy:

$$[P^\mu, P^\nu] = 0$$

$$-i\,[M_{\mu\nu}, P_\rho] = g_{\mu\nu} P_\nu - g_{\nu\rho} P_\mu$$

$$-i\,[M_{\mu\nu}, M_{\rho\sigma}] = g_{\mu\rho} M_{\nu\sigma} - g_{\nu\rho} M_{\mu\sigma} + g_{\nu\sigma} M_{\mu\rho} - g_{\mu\sigma} M_{\nu\rho}$$

Supersymmetry is represented by an operators $Q, \bar{Q}$ that relate particles differing by ½ integer spin. $Q$ and $\bar{Q}$ are Weyl spinors and the central objects in the anti-commutation relations involving the (extended) Pauli matrices $\sigma^{\mu}_{\alpha\dot{\alpha}}$

$$\{ Q_\alpha , \bar{Q}_{\dot{\alpha}} \} = 2 \sigma^{\mu}_{\alpha\dot{\alpha}} P_\mu$$

In addition

$$[ M^{\mu\nu}, Q_\alpha ] = \sigma_\alpha^{\mu\nu\ \beta} Q_\beta$$

$$[ M^{\mu\nu}, \bar{Q}^{\dot{\alpha}} ] = (\bar{\sigma}^{\mu\nu})^{\dot{\alpha}}{}_{\dot{\beta}} \, \bar{Q}^{\dot{\beta}}$$

where $\sigma_\alpha^{\mu\nu\ \beta} = \frac{i}{4} \left( \sigma^\mu \bar{\sigma}^\nu - \sigma^\nu \bar{\sigma}^\mu \right)_\alpha{}^\beta$

and

$$[ Q_\alpha , P^\mu ] = 0$$

$$\{ Q_\alpha , Q_\beta \} = 0$$