First Principles of ML

A Peek Inside the 'Black Box' of Machine Learning

Jack Fraser-Govil
The Wellcome Sanger Institute, Hinxton, UK
15th October 2024

Why bother?

In a world with dozens of pre-built ML tools....why bother studying the fundamentals?

Why bother?

In a world with dozens of pre-built ML tools....why bother studying the fundamentals?
In a word...

Why bother?

In a world with dozens of pre-built ML tools....why bother studying the fundamentals?
In a word...

FOLKLORE

ML Folklore

ML Folklore

- ADAM vs AdaGrad?

ML Folklore

- ADAM vs AdaGrad?
- Softplus vs ReLu vs Leaky ReLu vs Sigmoid?

ML Folklore

- ADAM vs AdaGrad?
- Softplus vs ReLu vs Leaky ReLu vs Sigmoid?
- Cross-Entropy vs Least Squares?

ML Folklore

- ADAM vs AdaGrad?
- Softplus vs ReLu vs Leaky ReLu vs Sigmoid?
- Cross-Entropy vs Least Squares?
- Validation set magic numbers

ML Folklore

- ADAM vs AdaGrad?
- Softplus vs ReLu vs Leaky ReLu vs Sigmoid?
- Cross-Entropy vs Least Squares?
- Validation set magic numbers
- (Explainability!)

Today's Agenda

The aim for today is:

Today's Agenda

The aim for today is:

- Classic Perceptron

Today's Agenda

The aim for today is:

- Classic Perceptron
- Feedforward Networks

Today's Agenda

The aim for today is:

- Classic Perceptron
- Feedforward Networks
- Non-Linearity

Today's Agenda

The aim for today is:

- Classic Perceptron
- Feedforward Networks
- Non-Linearity
- Optimisation & Backpropagation

Today's Agenda

The aim for today is:

- Classic Perceptron
- Feedforward Networks
- Non-Linearity
- Optimisation & Backpropagation
- Convolutional Networks*

(* Time dependent!)

Today's Agenda

The aim for today is:

- Classic Perceptron
- Feedforward Networks
- Non-Linearity
- Optimisation & Backpropagation
- Convolutional Networks*

(* Time dependent!)
As we progress you will slowly build up your own ML toolkit, built entirely from scratch!

A Warning

There will be equations.

A Warning

There will be equations.
You will need to know what they mean!

A Warning

There will be equations.
You will need to know what they mean!

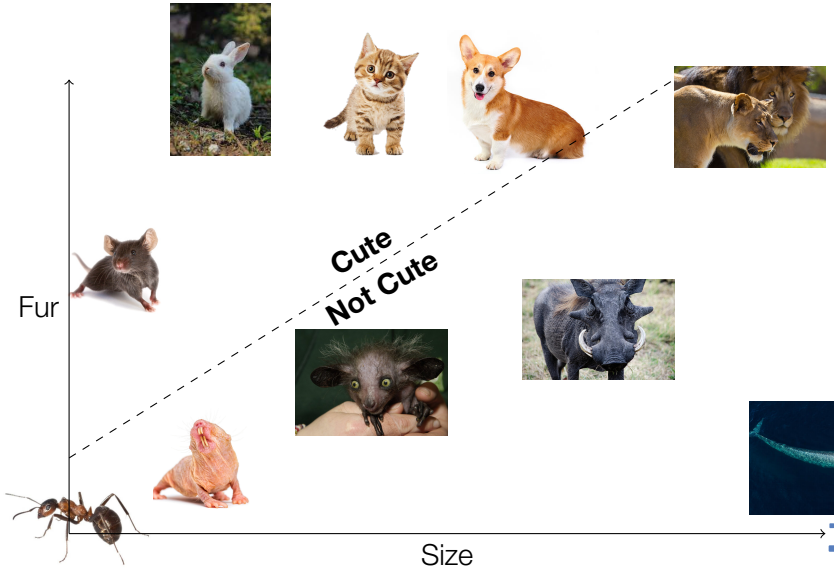
Please, please, please, ask if you want clarification on the underlying mathematics and theory! That's why you're here today!

Part 1
The Perceptron

Basic Decision Making: Defining Cuteness

wellcome sanger institute

Basic Decision Making: Defining Cuteness



wellcome sanger institute

Splitting The Plane

In order to split the plane into two parts, we merely need to define a ***line***.

Splitting The Plane

In order to split the plane into two parts, we merely need to define a ***line***.
Question: If we have N dimensions ($N = 2$), how many parameters do we need to define a line?

Splitting The Plane

In order to split the plane into two parts, we merely need to define a ***line***.

Question: If we have N dimensions ($N = 2$), how many parameters do we need to define a line?

Answer: The answer is N - in 2 dimensions, this is friendly $y = mx + c \rightarrow (m, c)$

Splitting The Plane

In order to split the plane into two parts, we merely need to define a ***line***.

Question: If we have N dimensions ($N = 2$), how many parameters do we need to define a line?

Answer: The answer is N - in 2 dimensions, this is friendly $y = mx + c \rightarrow (m, c)$

Question: Why then do we need $N + 1$ dimensions?

The Perceptron

We need 3 parameters to define a **directional line**. The Perceptron classifier algorithm is:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{if } \tilde{\mathbf{x}} \cdot \mathbf{w}^1 > 0 \\ 0 & \text{else} \end{cases} \tag{1}$$

[1] The dot/inner product is covered in Section 3.1 in the notes

The Perceptron

We need 3 parameters to define a **directional line**. The Perceptron classifier algorithm is:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{if } \tilde{\mathbf{x}} \cdot \mathbf{w}^1 > 0 \\ 0 & \text{else} \end{cases} \tag{1}$$

Where

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$$

[1] The dot/inner product is covered in Section 3.1 in the notes

The Perceptron

We need 3 parameters to define a **directional line**. The Perceptron classifier algorithm is:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{if } \tilde{\mathbf{x}} \cdot \mathbf{w}^1 > 0 \\ 0 & \text{else} \end{cases} \tag{1}$$

Where

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$$

$\mathbf{w}$ are the **weights**.

[1] The dot/inner product is covered in Section 3.1 in the notes

Exercise 1: Perceptron Classifier

Using the provided class structure & weights, write a perceptron classifier for the cuteness data.
The provided weights classify only one animal correctly. Which is it?

Training A Perceptron

Training a perceptron is easy. Loop over the data, make a prediction (P), compare it to the truth (T) and if it is wrong:

Training A Perceptron

Training a perceptron is easy. Loop over the data, make a prediction (P), compare it to the truth (T) and if it is wrong:

$$\mathbf{w} \rightarrow \mathbf{w} + r \times (T - P)\, \tilde{\mathbf{x}} \tag{2}$$

Training A Perceptron

Training a perceptron is easy. Loop over the data, make a prediction (P), compare it to the truth (T) and if it is wrong:

$$\mathbf{w} \rightarrow \mathbf{w} + r \times (T - P)\, \tilde{\mathbf{x}} \tag{2}$$

This works because it always makes $\mathbf{w} \cdot \tilde{\mathbf{x}}$ move closer towards zero (and hence to the tipping point of altering its decision).

Exercise 2: Train Your Perceptron

Write the Train() method of your Perceptron. Run it on the cuteness data.

Exercise 2: Train Your Perceptron

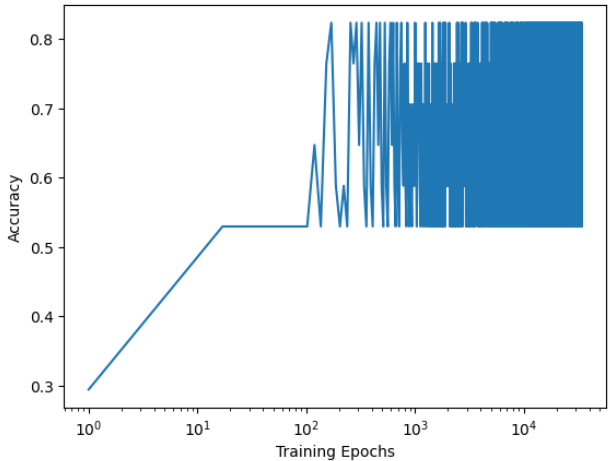
Write the Train() method of your Perceptron. Run it on the cuteness data.
How long does it take to train to 100% accuracy?

Exercise 3: The Limits of Linearity

Add in more data – what goes wrong?

Exercise 3: The Limits of Linearity

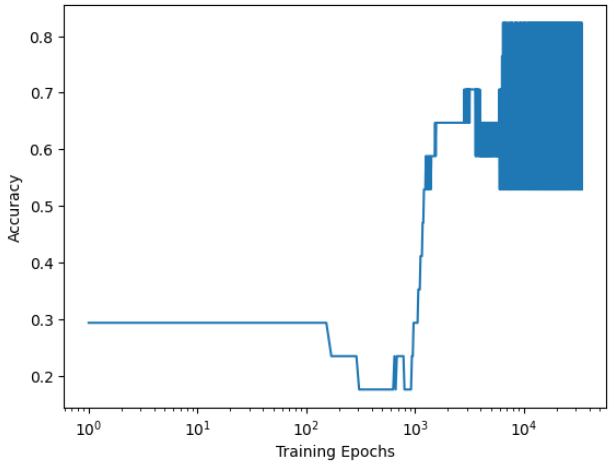
Add in more data – what goes wrong?



wellcome sanger institute

Exercise 3: The Limits of Linearity

Add in more data – what goes wrong?



wellcome sanger institute

The Nonlinear Perceptron

Eminently possible to have a non-linear perceptron.

The Nonlinear Perceptron

Eminently possible to have a non-linear perceptron.
Pass $\mathbf{x}$ through a non-linear transform to make it:

$$\mathbf{x}' = \begin{pmatrix} 1 \\ x \\ y \\ x^2 \\ \sin(x) \\ x^2 \exp(y) \\ \cos(\sin(\exp(\sin(\log(x^2 + 9xy))))) \\ \vdots \end{pmatrix} \tag{3}$$

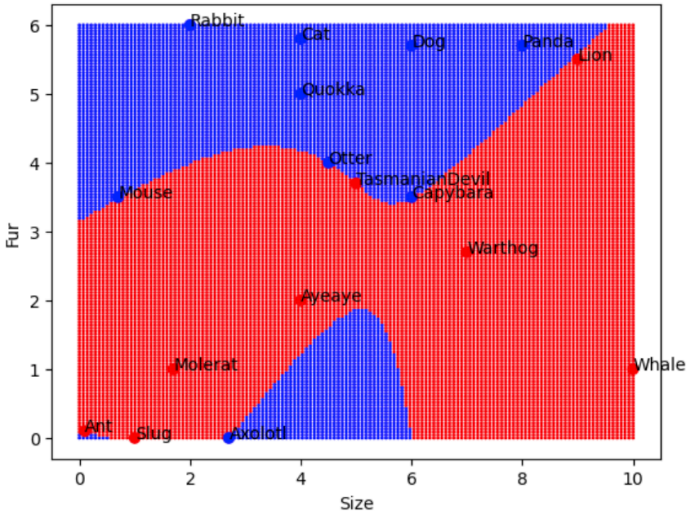
The Nonlinear Perceptron

Here is the results of a perceptron where:

$$\mathbf{x}'(x,y) = \begin{pmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 & x^4 & x^3y & x^2y^2 & xy^3 & y^4 \end{pmatrix}^\intercal \tag{4}$$

Bonus Question: Why did I choose this?

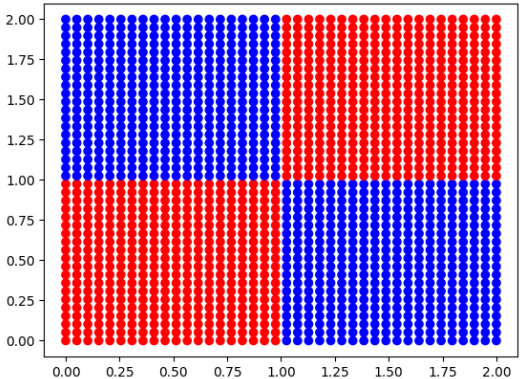
The Nonlinear Perceptron



wellcome sanger institute

The Limits of NonLinearity

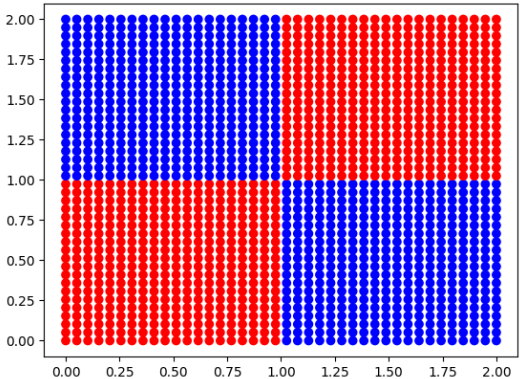
Target



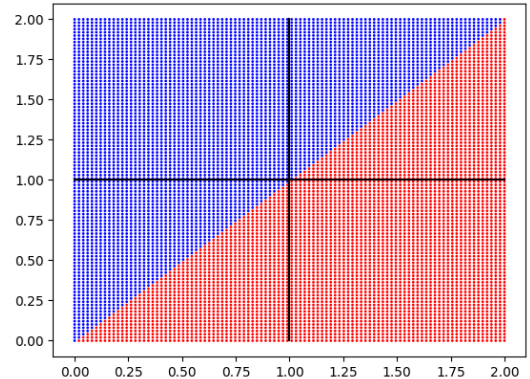
wellcome sanger institute

The Limits of NonLinearity

Target



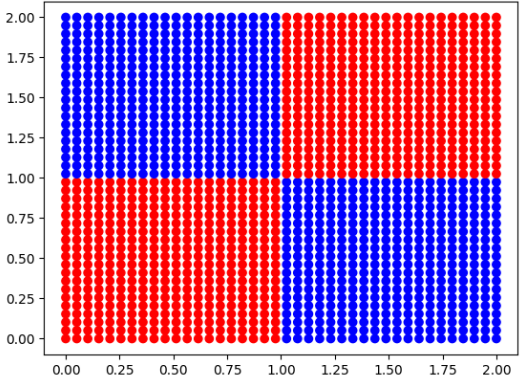
Reality (1st Order - 3 parameters)



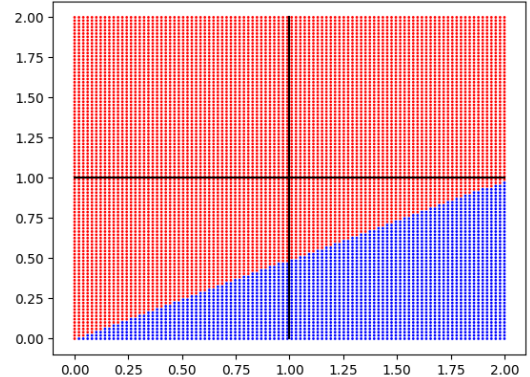
wellcome sanger institute

The Limits of NonLinearity

Target



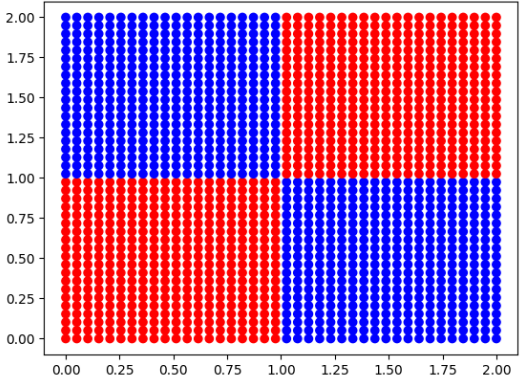
Reality (2nd Order - 6 parameters)



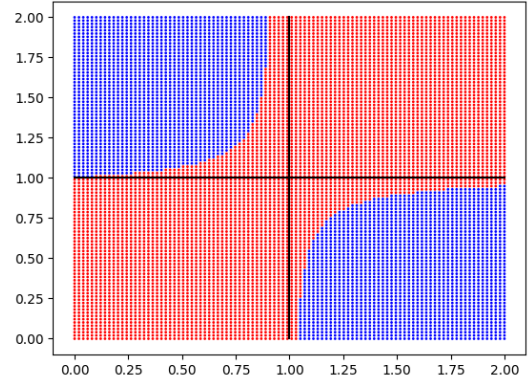
wellcome sanger institute

The Limits of NonLinearity

Target



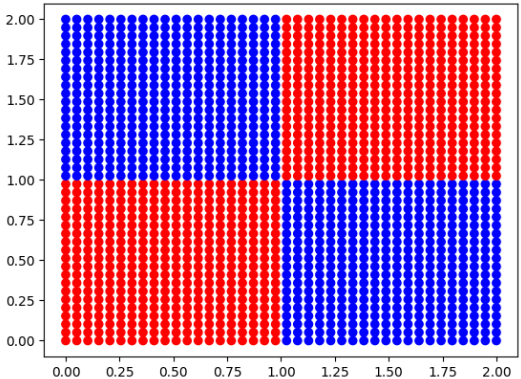
Reality (3rd Order - 10 parameters)



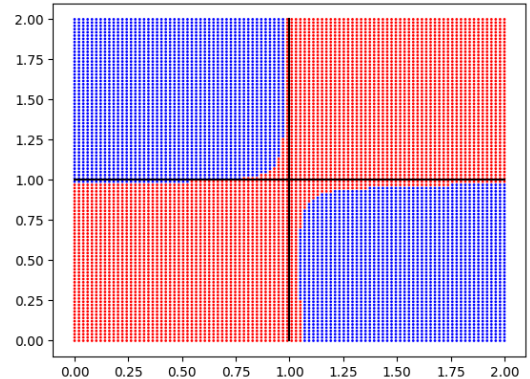
wellcome sanger institute

The Limits of NonLinearity

Target



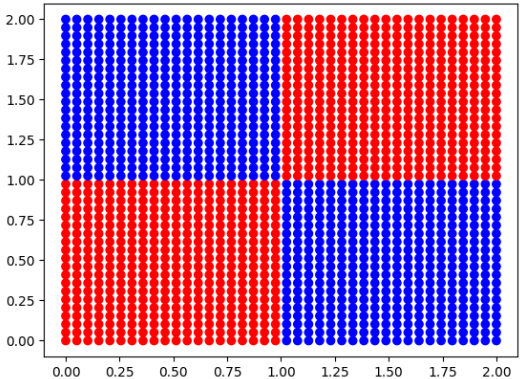
Reality (4th Order - 15 parameters)



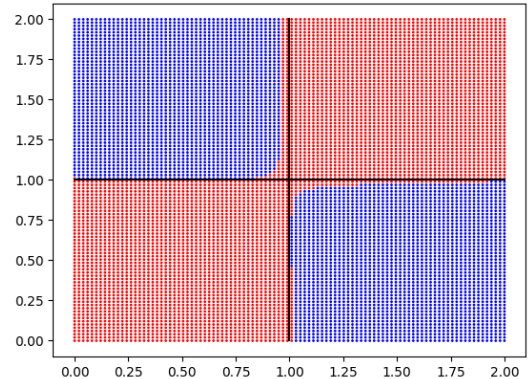
wellcome sanger institute

The Limits of NonLinearity

Target



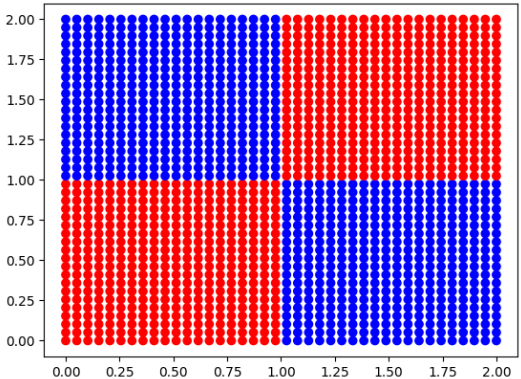
Reality (5th Order - 21 parameters)



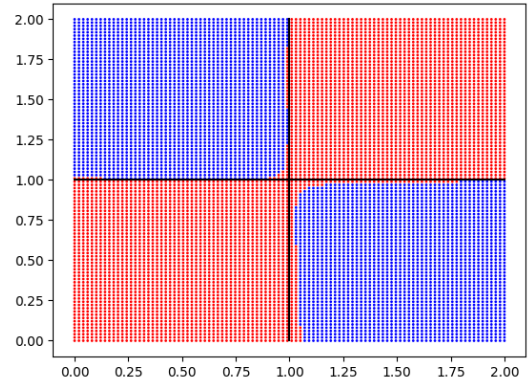
wellcome sanger institute

The Limits of NonLinearity

Target



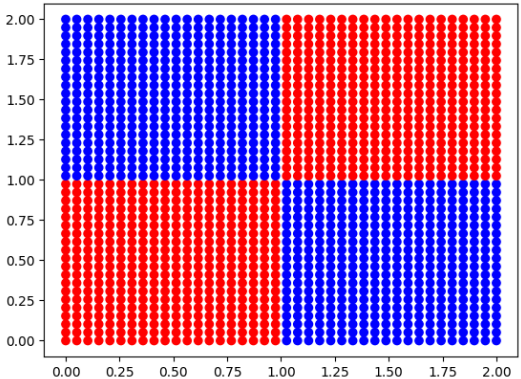
Reality (5th Order - 21 parameters)



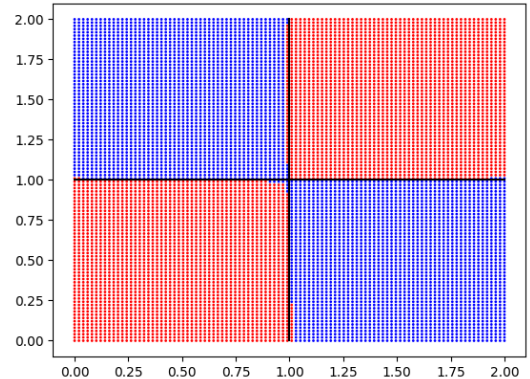
wellcome sanger institute

The Limits of NonLinearity

Target



Reality (20th Order - 231 parameters)



wellcome sanger institute

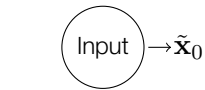
What do to?

What do to?

MORE!

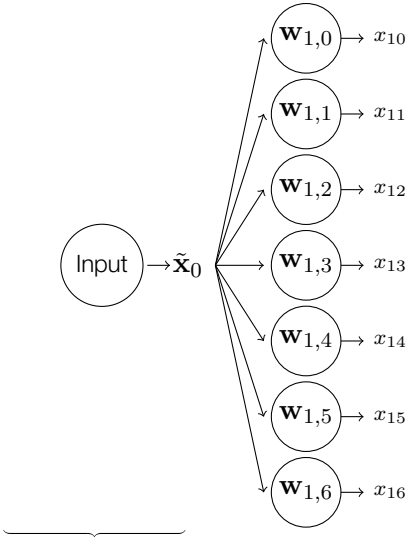
The Multilayered Perceptron

The Multilayered Perceptron

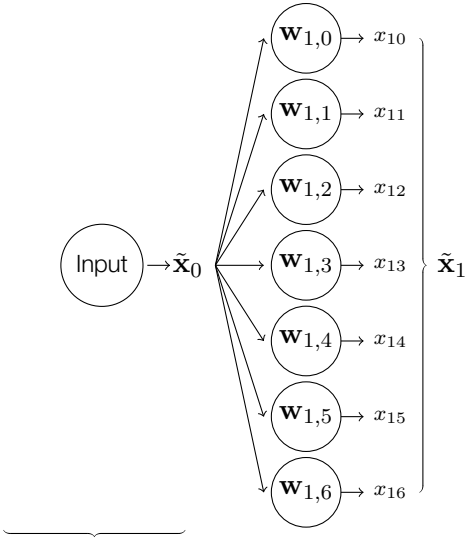


Input layer

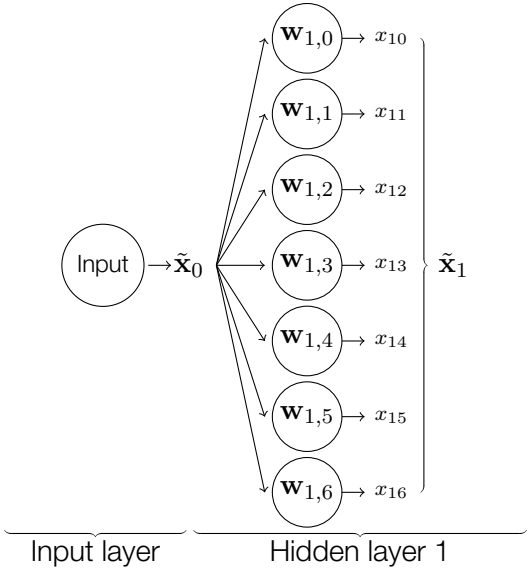
The Multilayered Perceptron



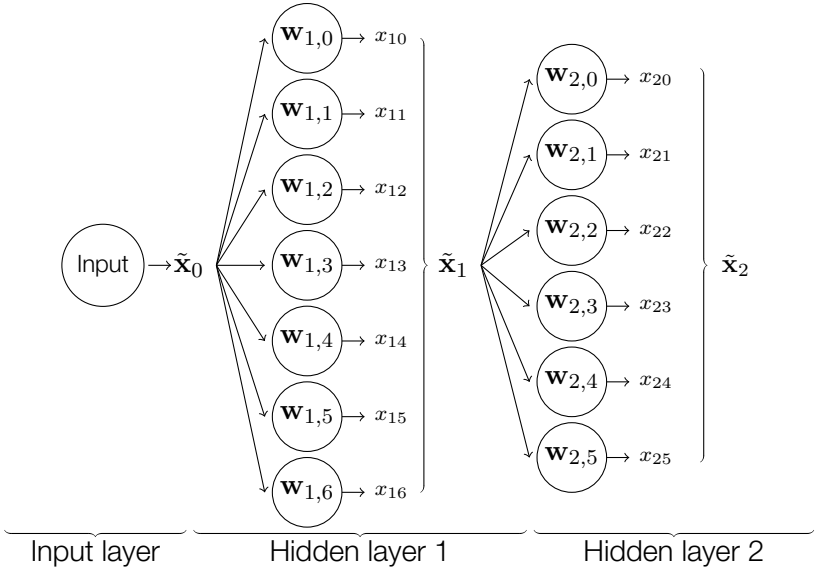
The Multilayered Perceptron



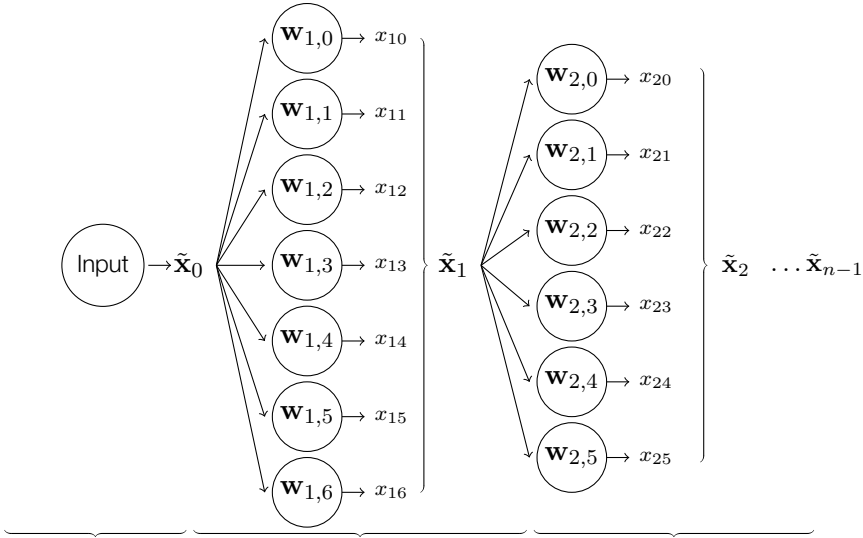
The Multilayered Perceptron



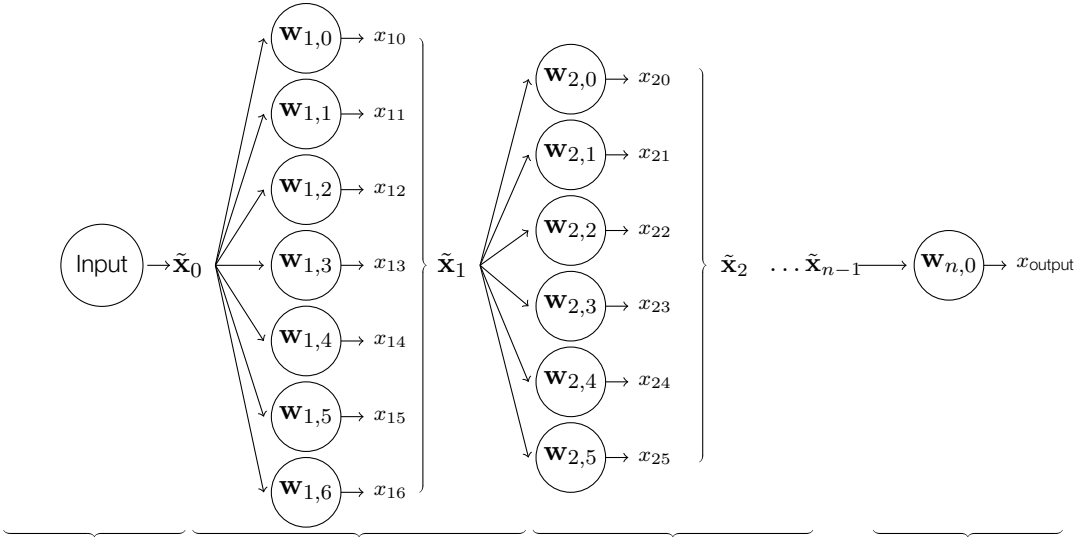
The Multilayered Perceptron



The Multilayered Perceptron



The Multilayered Perceptron



Exercise 4: The Multilayered Perceptron

I have provided the skeleton of a Node, Layer and Network class.

Exercise 4: The Multilayered Perceptron

I have provided the skeleton of a Node, Layer and Network class.

Fill them in so that you can construct and operate a network.

Exercise 4: The Multilayered Perceptron

I have provided the skeleton of a Node, Layer and Network class.

Fill them in so that you can construct and operate a network.

Do you notice anything odd about how this predictor behaves?

Exercise 4: The Multilayered Perceptron

I have provided the skeleton of a Node, Layer and Network class.

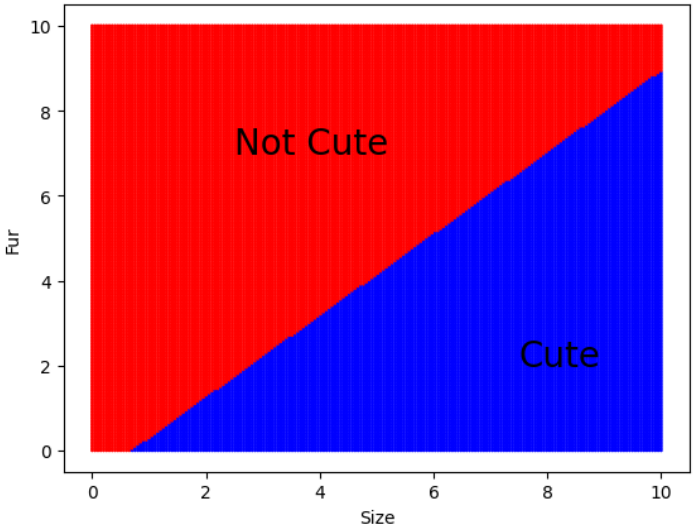
Fill them in so that you can construct and operate a network.

Do you notice anything odd about how this predictor behaves?

(For now: randomly initialise your weights)

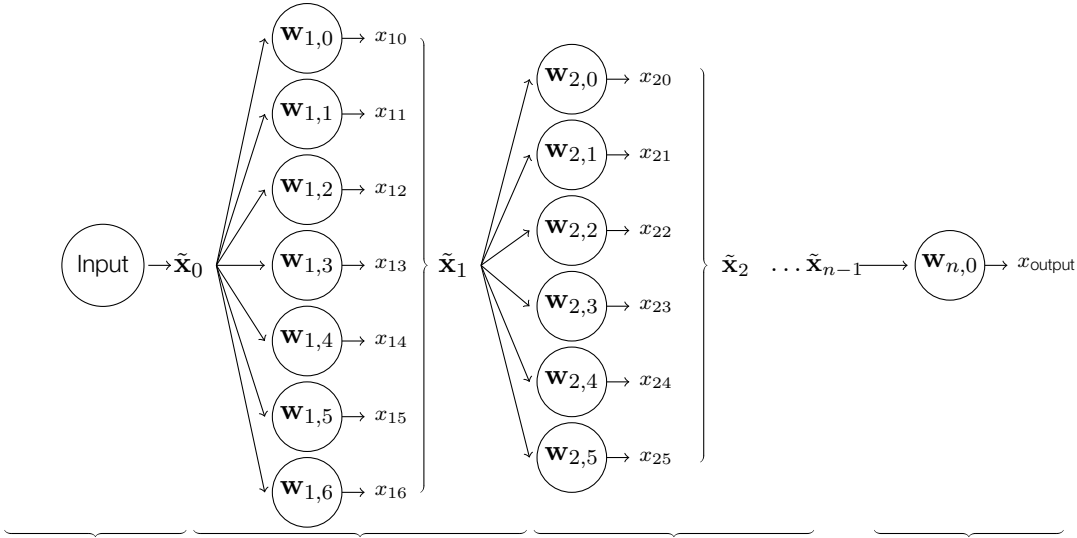
All that for nothing?

If everything went well, you should see something like this:

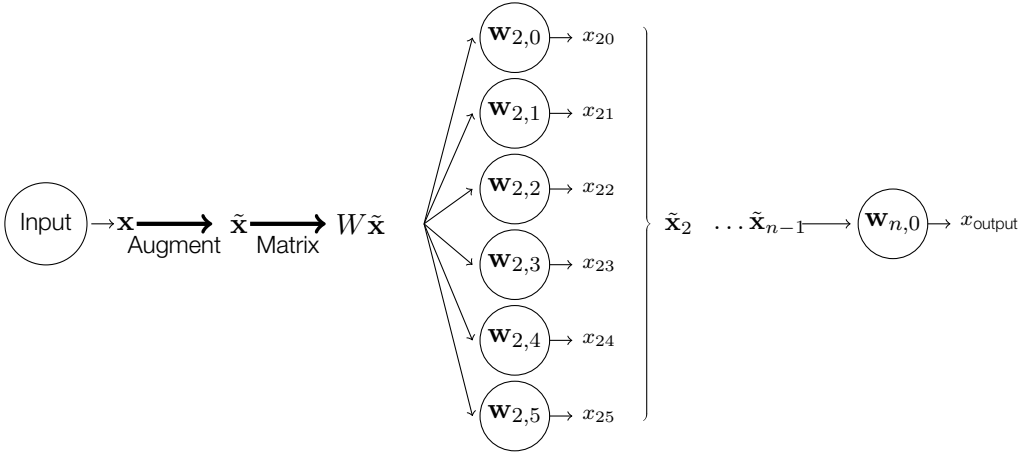


wellcome sanger institute

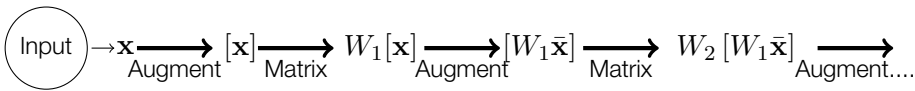
Affine Mess We Find Ourselves in



Affine Mess We Find Ourselves in



Affine Mess We Find Ourselves in



Affine Mess We Find Ourselves in

Or, in terms of **Affine Operators** (Chapter 3.4 in the notes):

$$
\begin{aligned}
\mathbf{x}_1 &= \hat{\mathcal{W}}_1 \mathbf{x}_{\text{input}} \\
\mathbf{x}_2 &= \hat{\mathcal{W}}_2 \mathbf{x}_1 = \hat{\mathcal{W}}_2 \hat{\mathcal{W}}_1 \mathbf{x}_{\text{input}} \\
\mathbf{x}_3 &= \hat{\mathcal{W}}_3 \mathbf{x}_2 = \hat{\mathcal{W}}_3 \hat{\mathcal{W}}_2 \hat{\mathcal{W}}_1 \mathbf{x}_{\text{input}} \\
&\vdots \\
x_{\text{output}} &= \begin{cases} 1 & \hat{\mathcal{W}}_N \mathbf{x}_{N-1} > 0 \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{5}
$$

Affine Mess We Find Ourselves in

But...affine operators obey:

$$\hat{\mathcal{W}}_2\hat{\mathcal{W}}_1 = \hat{\mathcal{V}} \tag{6}$$

(Affine + Affine = Affine)
Which means:

$$\hat{\mathcal{W}}_N\hat{\mathcal{W}}_{N-1}\hat{\mathcal{W}}_{N-2}\ldots\hat{\mathcal{W}}_1\mathbf{x}_{\text{input}} = \hat{\mathcal{V}}\mathbf{x}_{\text{input}} \tag{7}$$

Affine Mess We Find Ourselves in

But...affine operators obey:

$$\hat{\mathcal{W}}_2\hat{\mathcal{W}}_1 = \hat{\mathcal{V}} \tag{6}$$

(Affine + Affine = Affine)
Which means:

$$\hat{\mathcal{W}}_N\hat{\mathcal{W}}_{N-1}\hat{\mathcal{W}}_{N-2}\ldots\hat{\mathcal{W}}_1\mathbf{x}_{\text{input}} = \hat{\mathcal{V}}\mathbf{x}_{\text{input}} \tag{7}$$

If you do the maths:

$$\hat{\mathcal{V}}\mathbf{x} = \mathbf{v}\cdot\mathbf{x} + \mathbf{b} \tag{8}$$

Affine Mess We Find Ourselves in

But...affine operators obey:

$$\hat{\mathcal{W}}_2\hat{\mathcal{W}}_1 = \hat{\mathcal{V}} \tag{6}$$

(Affine + Affine = Affine)
Which means:

$$\hat{\mathcal{W}}_N\hat{\mathcal{W}}_{N-1}\hat{\mathcal{W}}_{N-2}\ldots\hat{\mathcal{W}}_1\mathbf{x}_{\text{input}} = \hat{\mathcal{V}}\mathbf{x}_{\text{input}} \tag{7}$$

If you do the maths:

$$\hat{\mathcal{V}}\mathbf{x} = \mathbf{v}\cdot\mathbf{x} + \mathbf{b} \tag{8}$$

Which is just....a single perceptron layer

Affine Mess We Find Ourselves in

If all you do is multiply by matrices & add vectors, you will *never* be doing anything other than an (extremely wasteful) perceptron algorithm

NonLinearity to the Rescue!

Need to break this affine relationship: enter **activation functions**.
Let:

$$\mathbf{x}_n = \text{activate}\left(W\mathbf{x}_{n-1} + \mathbf{b}\right) \tag{9}$$

Where activate (x) is a non-linear function, the activation function.
One of the simples ways to break linearity is to add a *elementwise function*:

$$\begin{aligned} \mathbf{v} &= W\mathbf{x}_{n-1} + \mathbf{b} \\ [\mathbf{x}_n]_i &= \sigma(v_i) \end{aligned} \tag{10}$$

Elementwise operations on vectors are – except for very specific cases – great for making vectors forget that they're vectors.

Elementwise Activation Functions

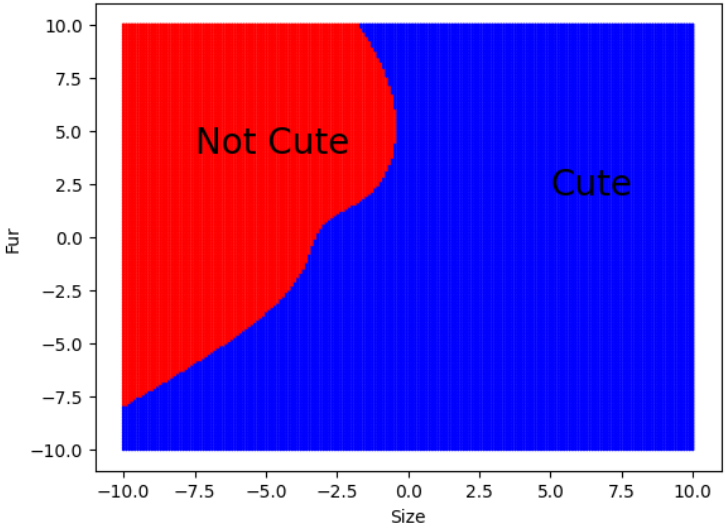
You can probably all name dozens of these functions in common use.

Sigmoid /Logit	$\sigma(x)$	$= \frac{1}{1+\exp(-x)}$
Rectified Linear Unit (ReLU)	$\text{Relu}(x)$	$= \begin{cases} x & x > 0 \\ 0 & \text{else} \end{cases}$
Leaky ReLU	$\text{Lelu}(x)$	$= \begin{cases} x & x > 0 \\ 0.01x & \text{else} \end{cases}$
tanh	$\tanh(x)$	$= \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
Softplus	$\text{sft}(x)$	$= \log\left(1+\exp(-x)\right)$

*(Bonus Question: Can you name any commonly used functions which **don't** follow this pattern?)*

NonLinearity Ahoy!

By replacing my (randomly initialised) hidden layers with Sigmoid nodes, I get:



wellcome sanger institute