Under The Hood

A Peek Inside the 'Black Box' of Machine Learning

Jack Fraser-Govil
The Wellcome Sanger Institute, Hinxton, UK

18th May 2023

wellcome sanger institute

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

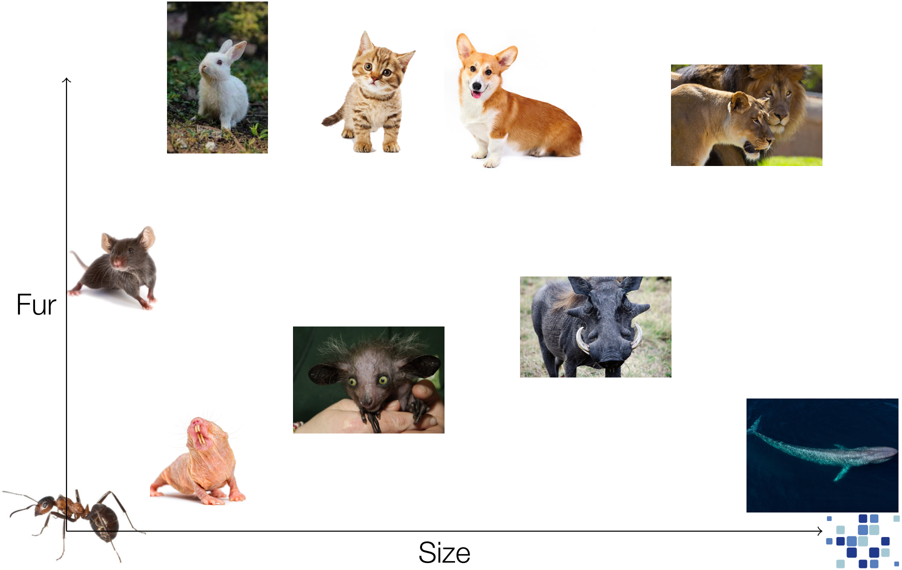
- ADAM vs AdaGrad?
- Softplus vs ReLu vs Leaky ReLu vs Sigmoid?
- Cross-Entropy vs Least Squares?

wellcome sanger institute

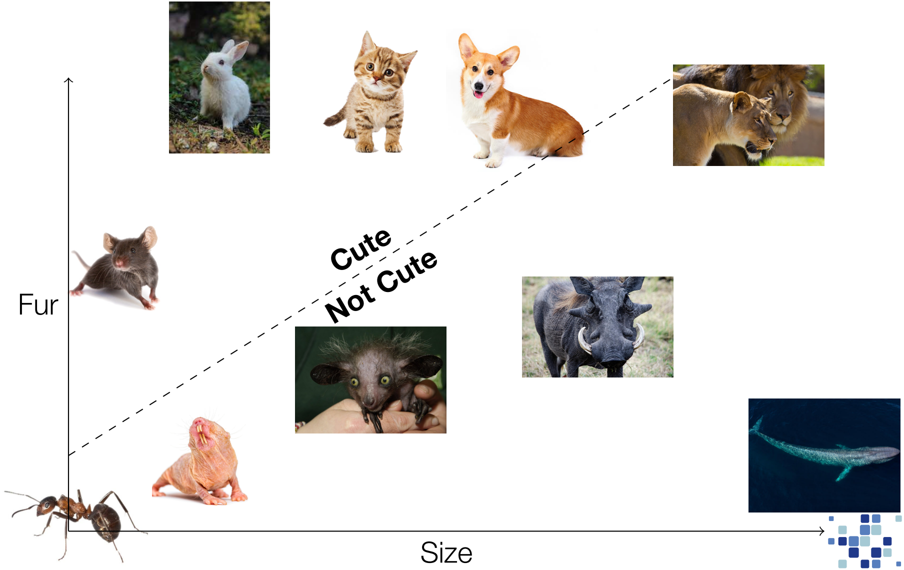
ML Folklore

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

Basic Decision Making: Defining Cuteness



Basic Decision Making: Defining Cuteness



wellcome sanger institute

In maths...

Let $\vec{x}$ be the 'feature vector', $\vec{x} = \begin{pmatrix} 1 \\ \text{size} \\ \text{fur} \end{pmatrix}$, and $\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ be our 'weights':

In maths...

Let $\vec{x}$ be the 'feature vector', $\vec{x} = \begin{pmatrix} 1 \\ \text{size} \\ \text{fur} \end{pmatrix}$, and $\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ be our 'weights':

The Perceptron algorithm is:

$$\mathcal{P}_{\vec{w}}(\vec{x}) = \begin{cases} 1 \text{ (cute)} & \text{when } \vec{w} \cdot \vec{x} > 0 \\ 0 \text{ (not cute)} & \text{else} \end{cases} \tag{1}$$

In maths...

Let $\vec{x}$ be the 'feature vector', $\vec{x} = \begin{pmatrix} 1 \\ \text{size} \\ \text{fur} \end{pmatrix}$, and $\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ be our 'weights':

The Perceptron algorithm is:

$$\mathcal{P}_{\vec{w}}(\vec{x}) = \begin{cases} 1 \text{ (cute)} & \text{when } \vec{w} \cdot \vec{x} > 0 \\ 0 \text{ (not cute)} & \text{else} \end{cases} \tag{1}$$

Therefore $\vec{w}$ defines a *line* which divides our 2D space – Perceptron simply splits the region into two $\rightarrow$ though can pull some fancy tricks.

Training Perceptron

Training = finding the ***best*** $\vec{w}$

Training Perceptron

Training = finding the ***best*** $\vec{w}$

Perceptron algorithm loops over a labelled 'training set', where the known cuteness of j is C_j

$$\vec{w} \rightarrow \vec{w} + r\left(C_j - \mathcal{P}_{\vec{w}}(\vec{x}_j)\right)\vec{x}_j \tag{2}$$

Training Perceptron

Training = finding the ***best*** $\vec{w}$

Perceptron algorithm loops over a labelled ‘training set’, where the known cuteness of j is C_j

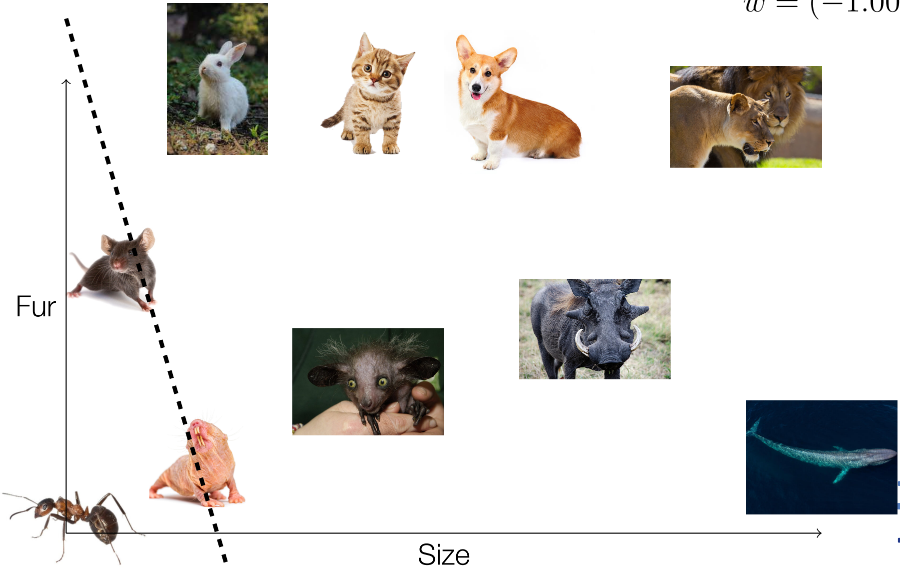
$$\vec{w} \rightarrow \vec{w} + r\left(C_j - \mathcal{P}_{\vec{w}}(\vec{x}_j)\right)\vec{x}_j \tag{2}$$

In words

For each element in the set, if I guess wrong ($C_j \neq \mathcal{P}_j$), move $\vec{w}$ by a small amount (r) to make me less wrong.

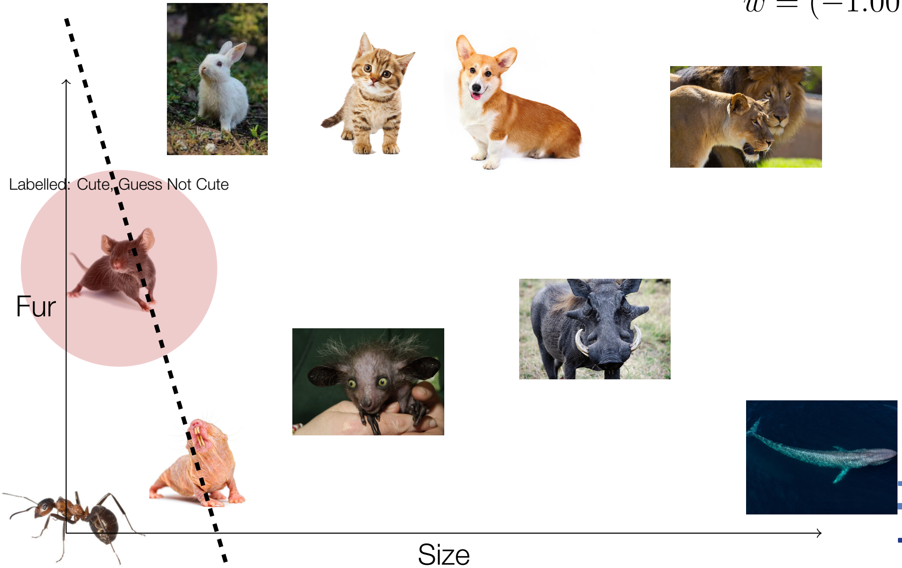
Training Perceptron

$$\vec{w} = (-1.00, 0.50, 0.15)$$



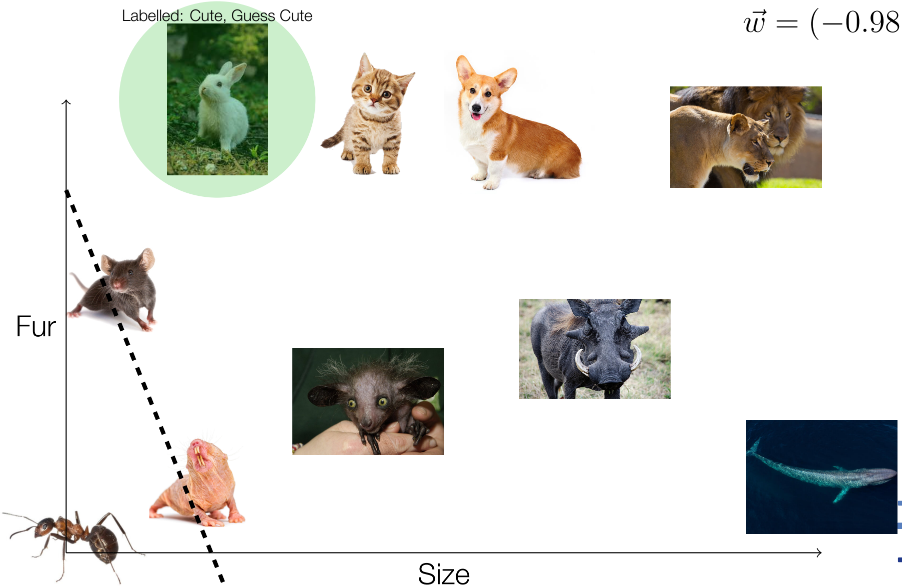
Training Perceptron

$$\vec{w} = (-1.00, 0.50, 0.15)$$



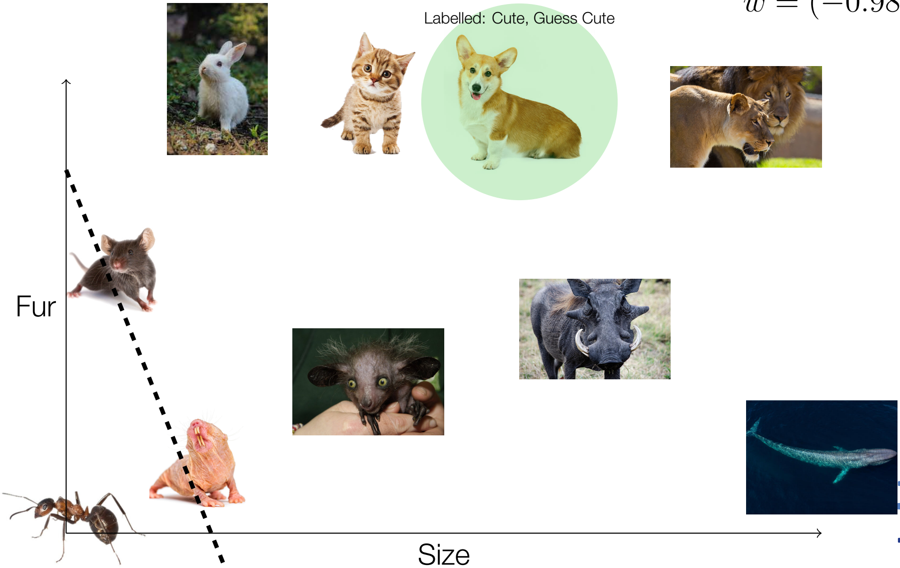
Training Perceptron

$$\vec{w} = (-0.98, 0.51, 0.20)$$



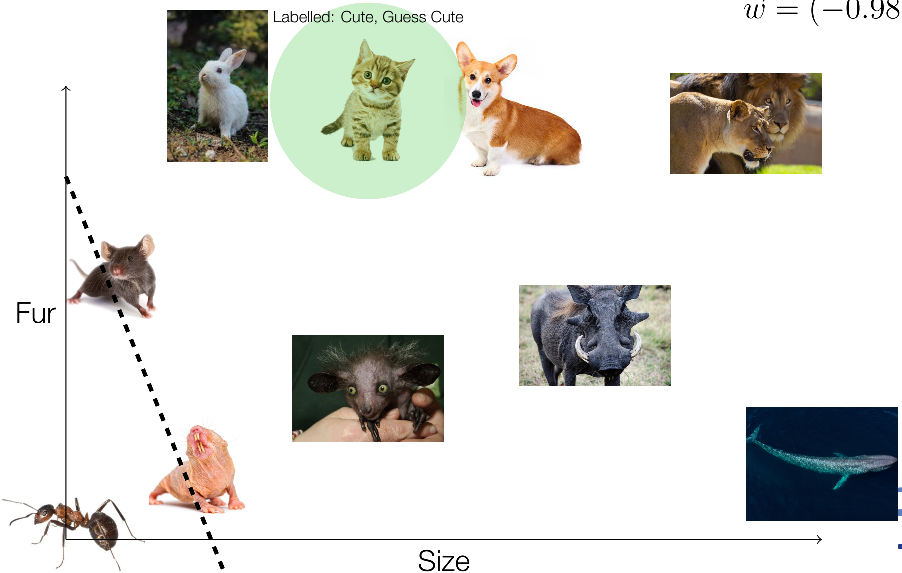
Training Perceptron

$$\vec{w} = (-0.98, 0.51, 0.20)$$



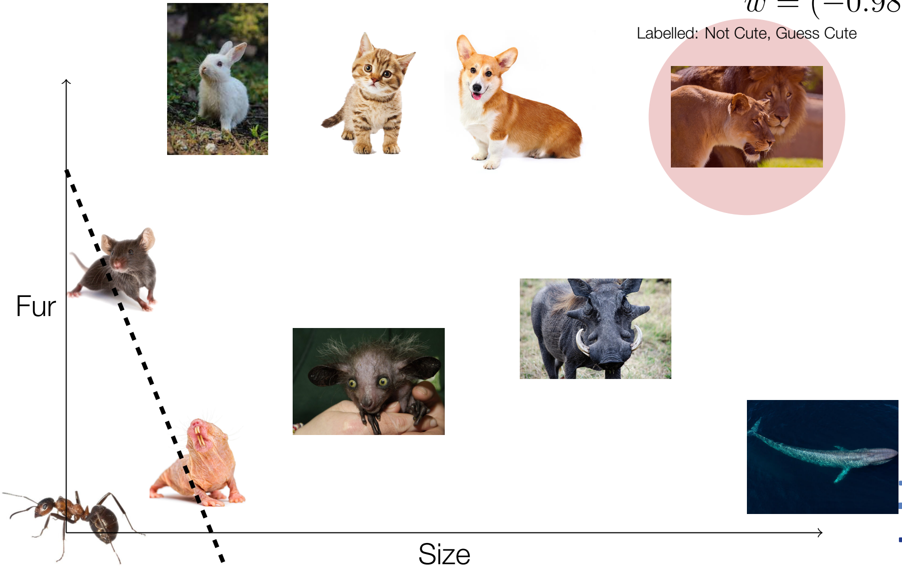
Training Perceptron

$$\vec{w} = (-0.98, 0.51, 0.20)$$



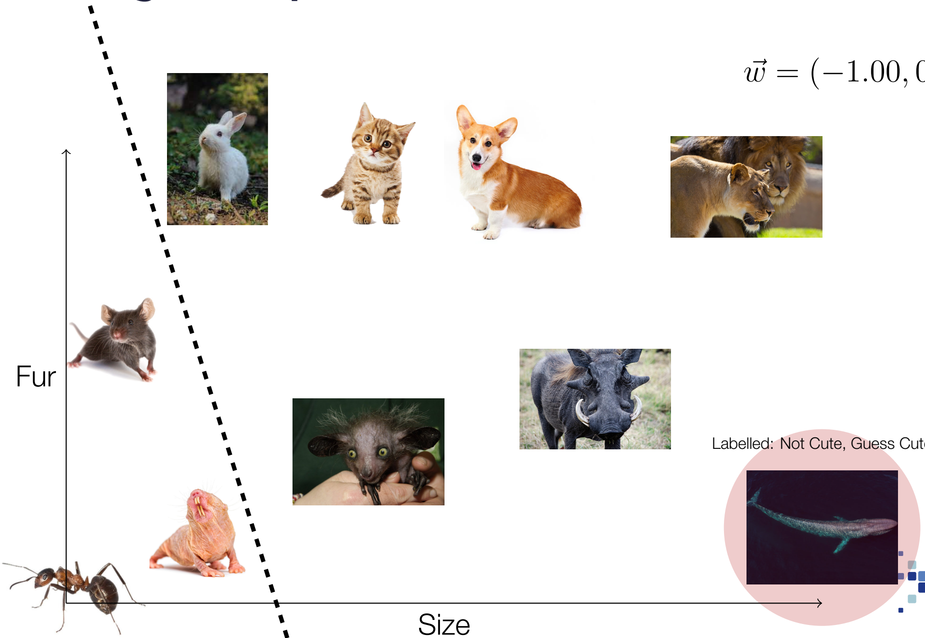
Training Perceptron

$$\vec{w} = (-0.98, 0.51, 0.20)$$



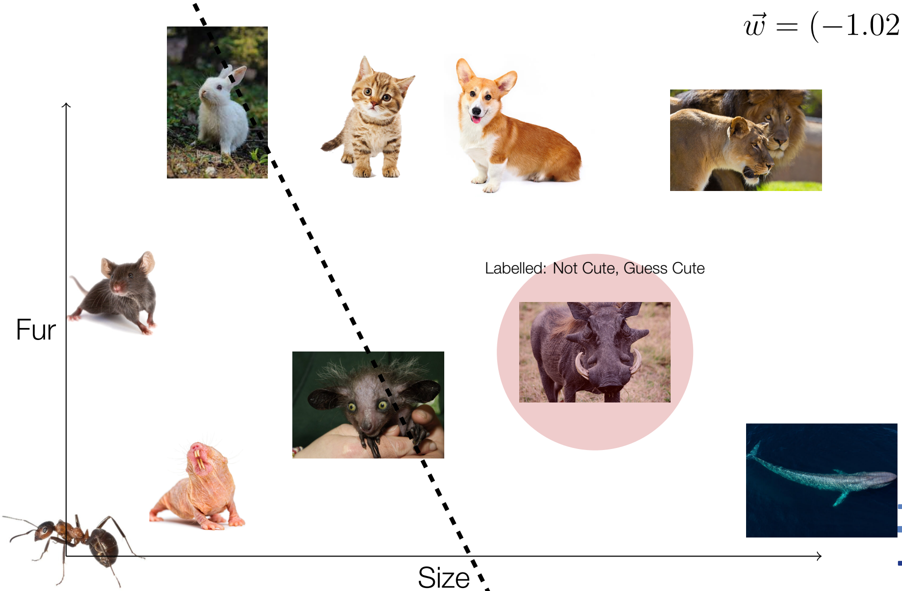
Training Perceptron

$$\vec{w} = (-1.00, 0.36, 0.11)$$



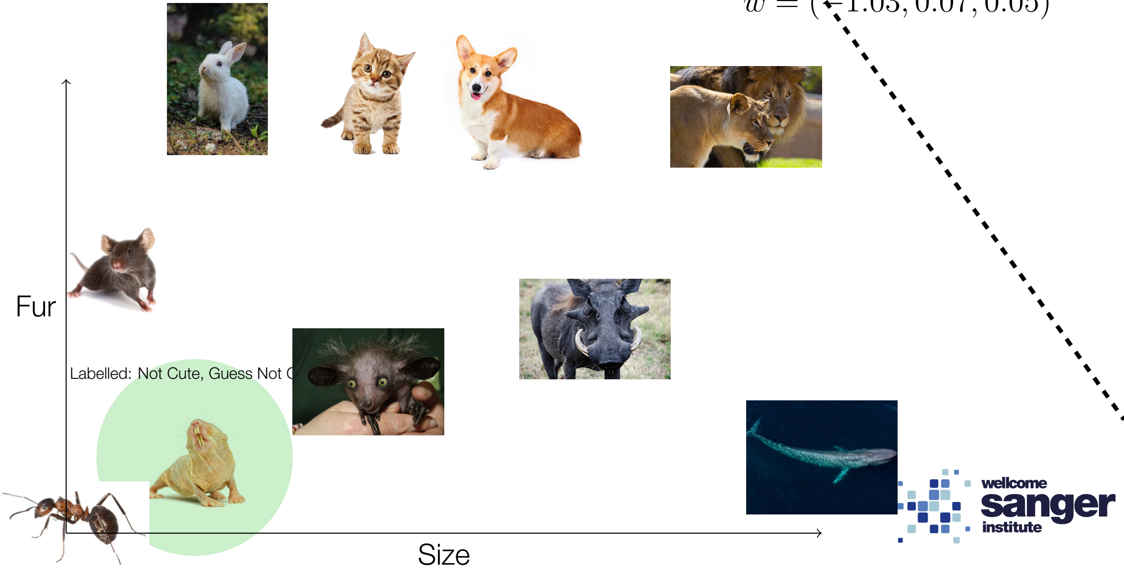
Training Perceptron

$$\vec{w} = (-1.02, 0.19, 0.09)$$

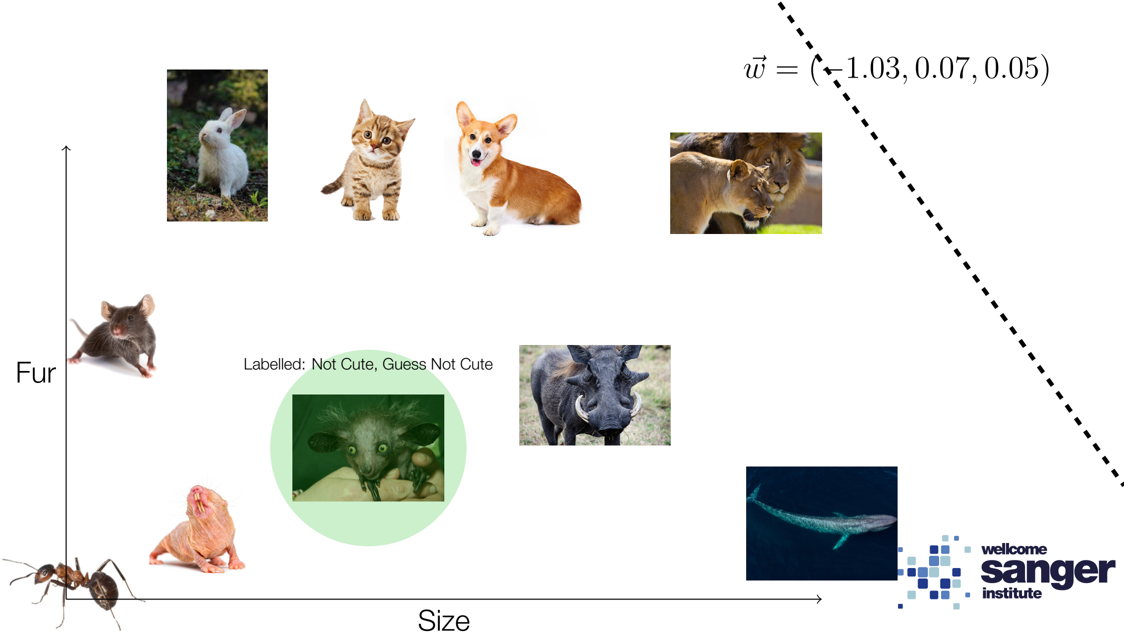


Training Perceptron

$$\vec{w} = (-1.03, 0.07, 0.05)$$



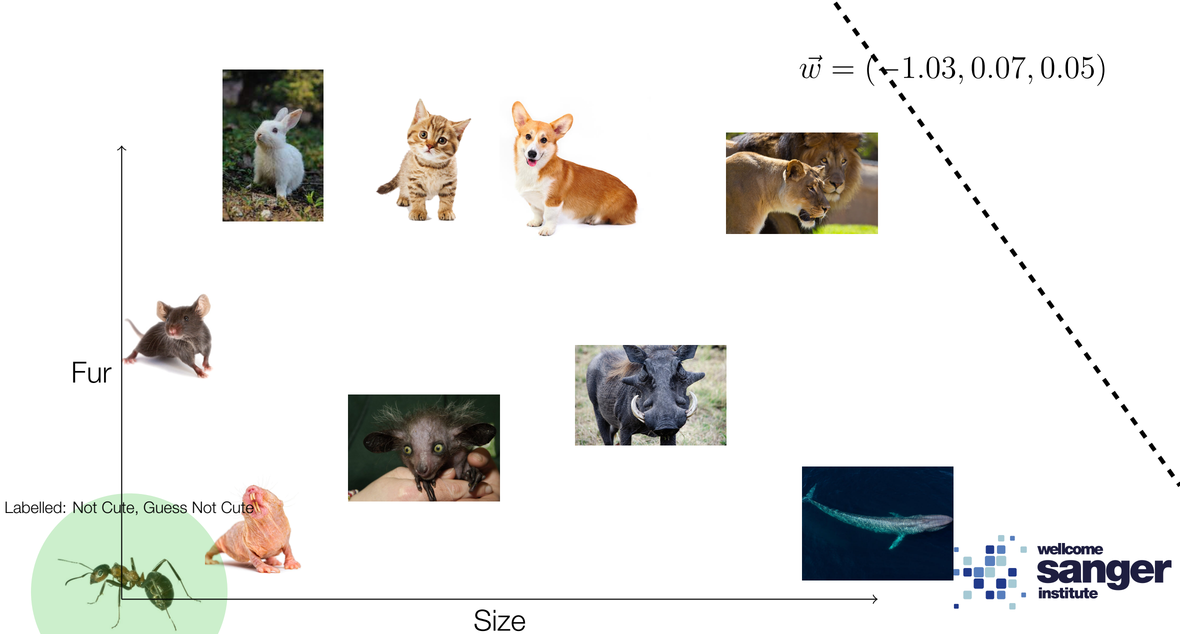
Training Perceptron



wellcome sanger institute

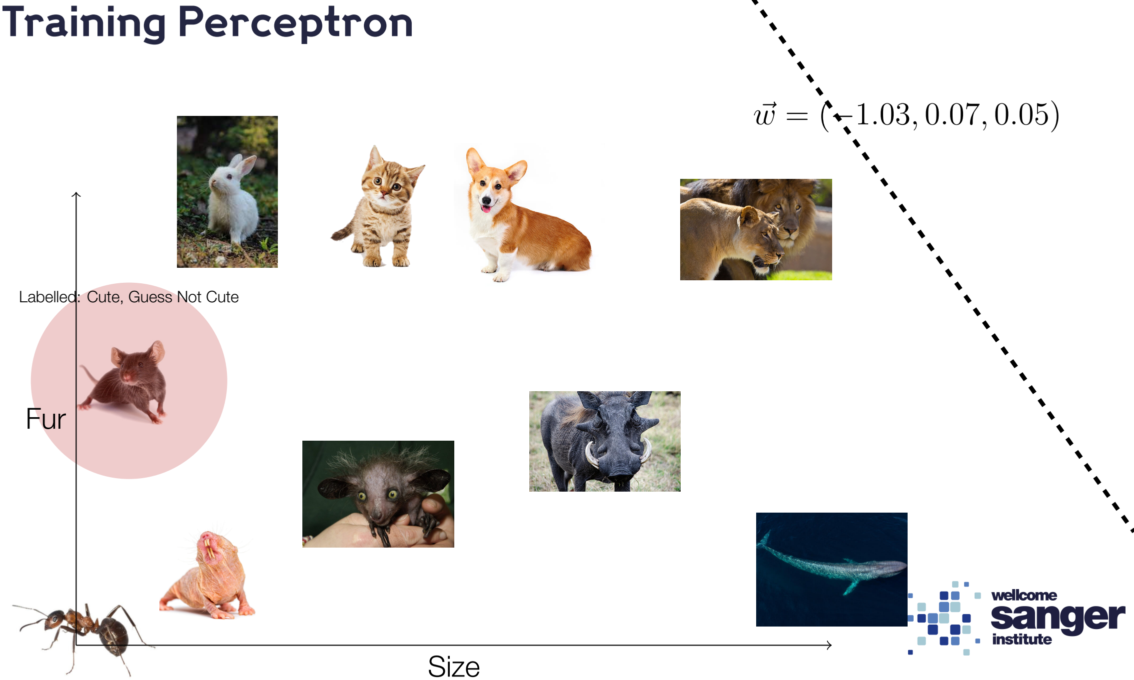
Training Perceptron

$$\vec{w} = (-1.03, 0.07, 0.05)$$



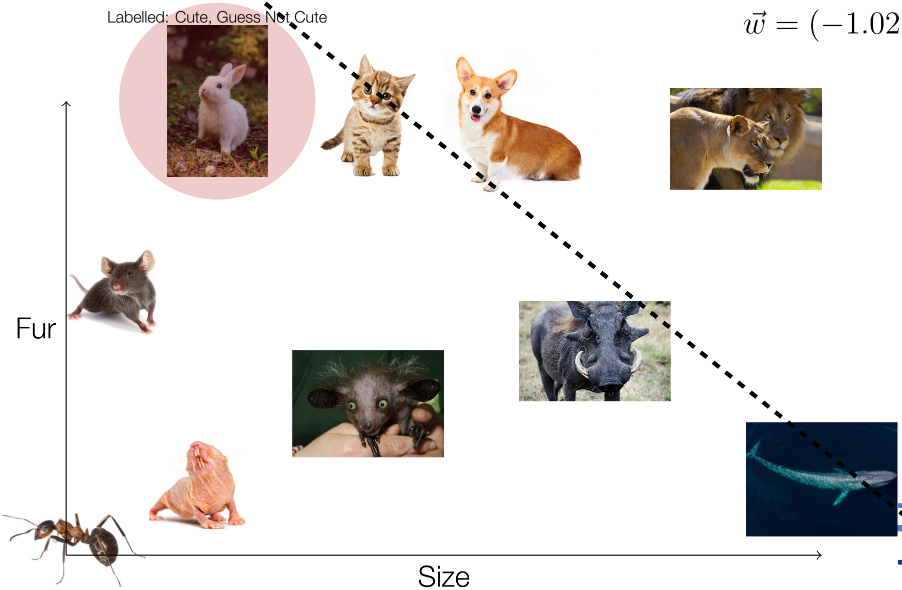
Training Perceptron

$\vec{w} = (-1.03, 0.07, 0.05)$



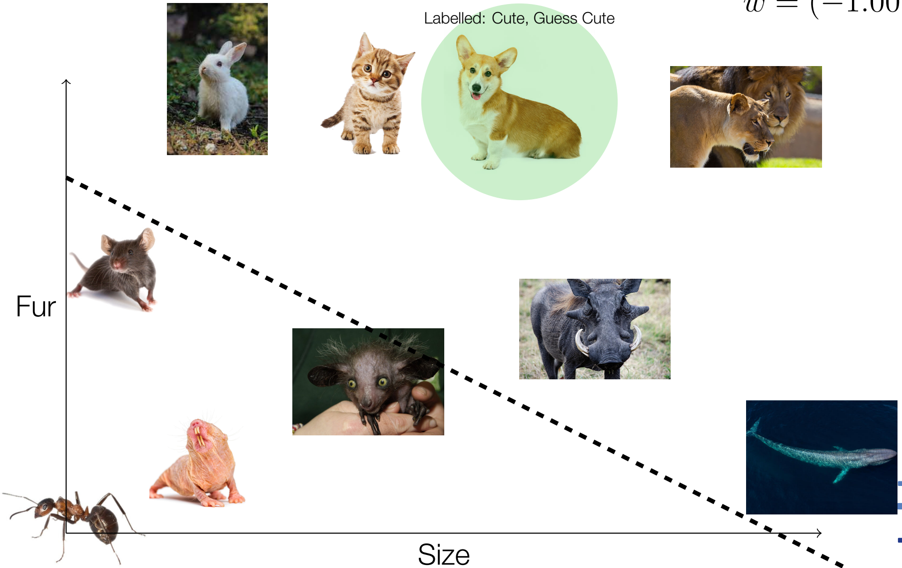
Training Perceptron

$$\vec{w} = (-1.02, 0.08, 0.11)$$



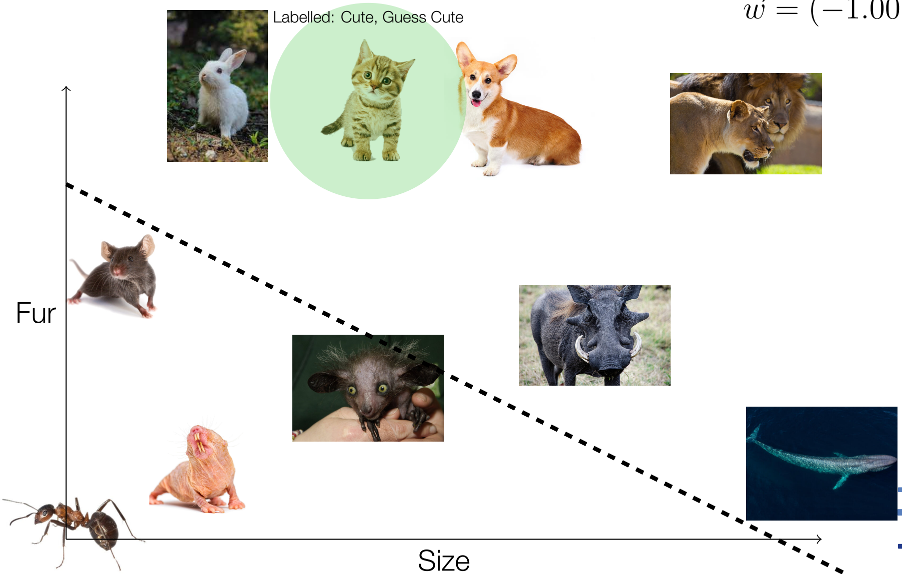
Training Perceptron

$$\vec{w} = (-1.00, 0.12, 0.21)$$



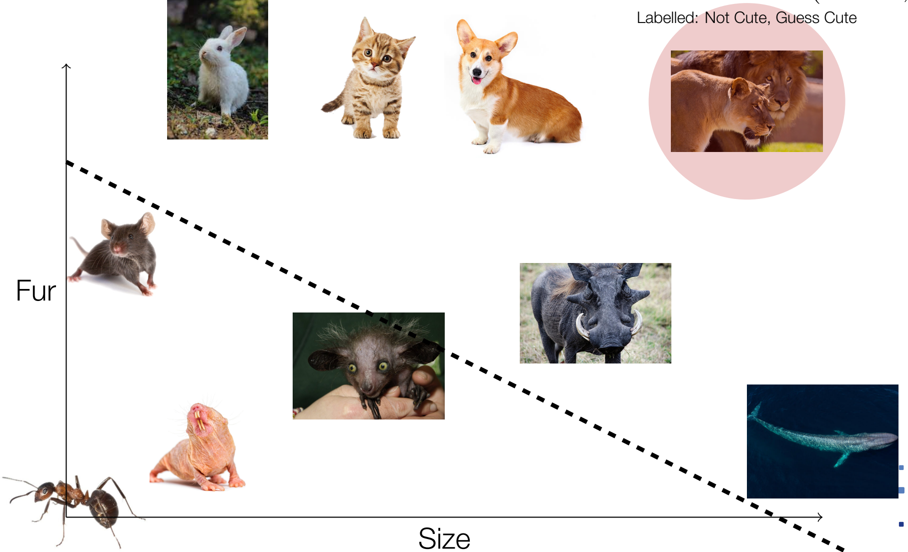
Training Perceptron

$$\vec{w} = (-1.00, 0.12, 0.21)$$



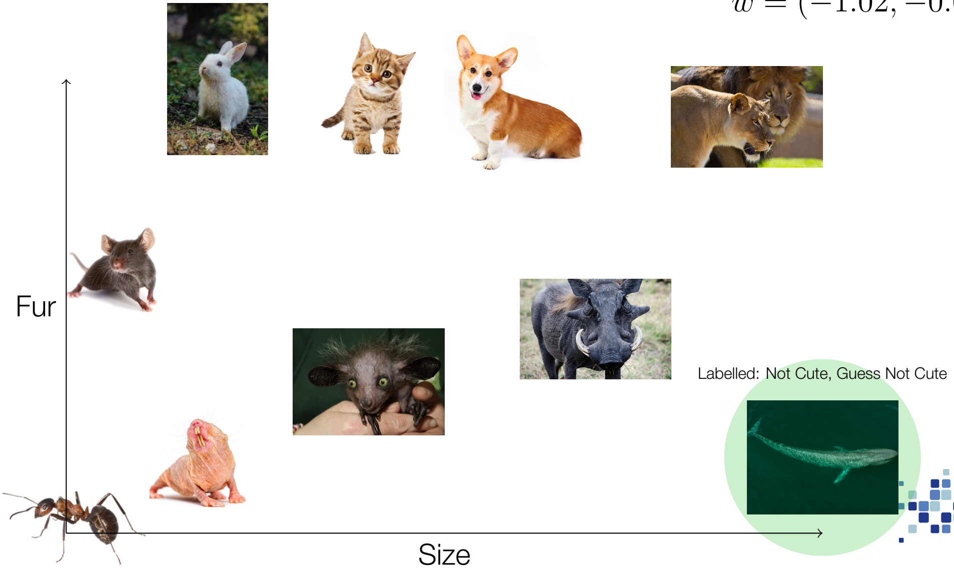
Training Perceptron

$$\vec{w} = (-1.00, 0.12, 0.21)$$



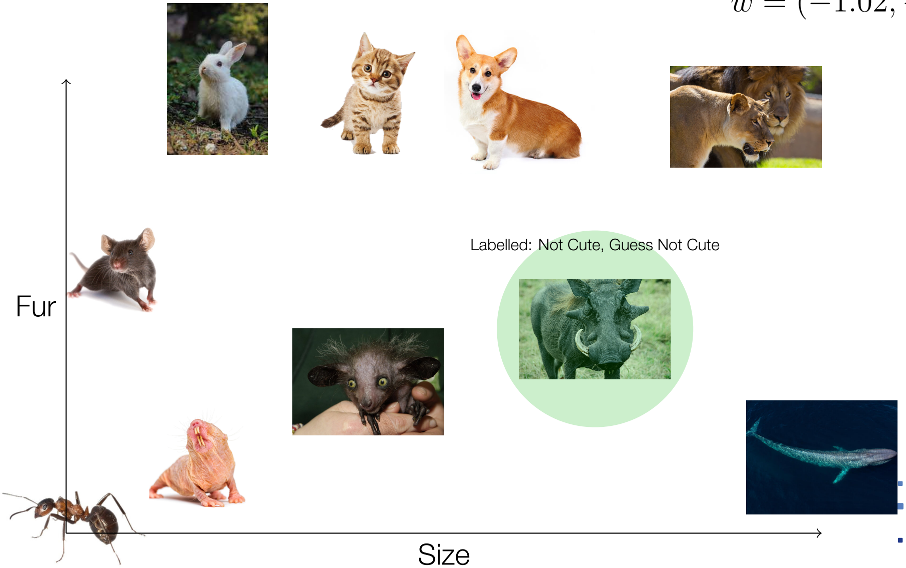
Training Perceptron

$$\vec{w} = (-1.02, -0.04, 0.12)$$



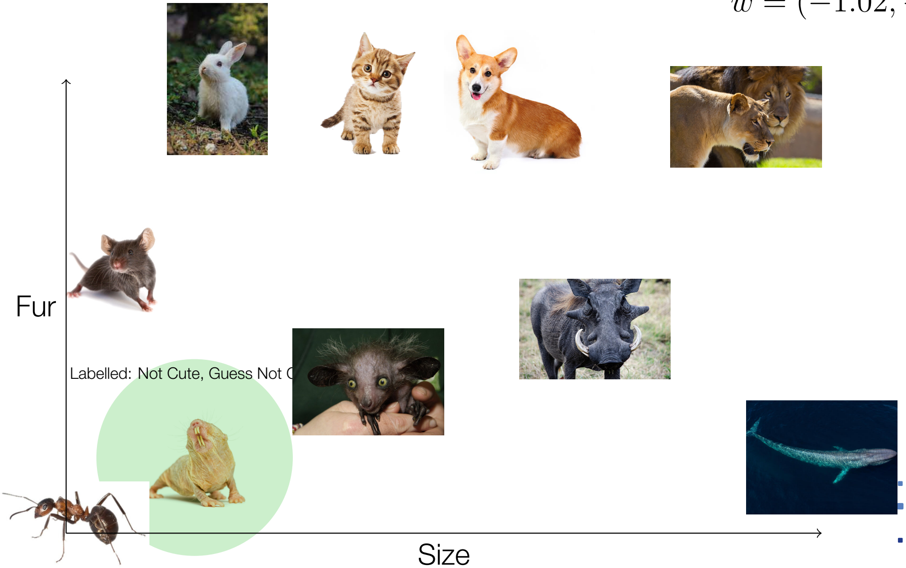
Training Perceptron

$$\vec{w} = (-1.02, -0.04, 0.12)$$



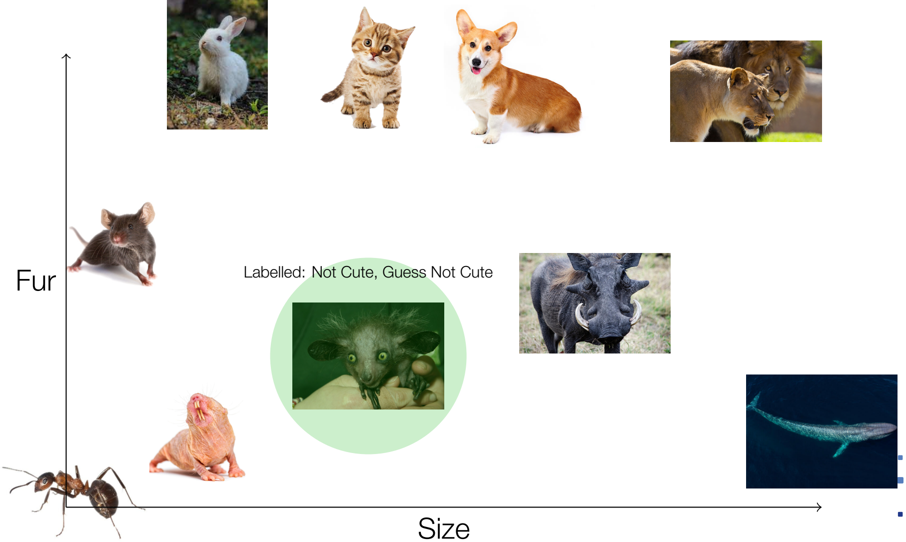
Training Perceptron

$$\vec{w} = (-1.02, -0.04, 0.12)$$



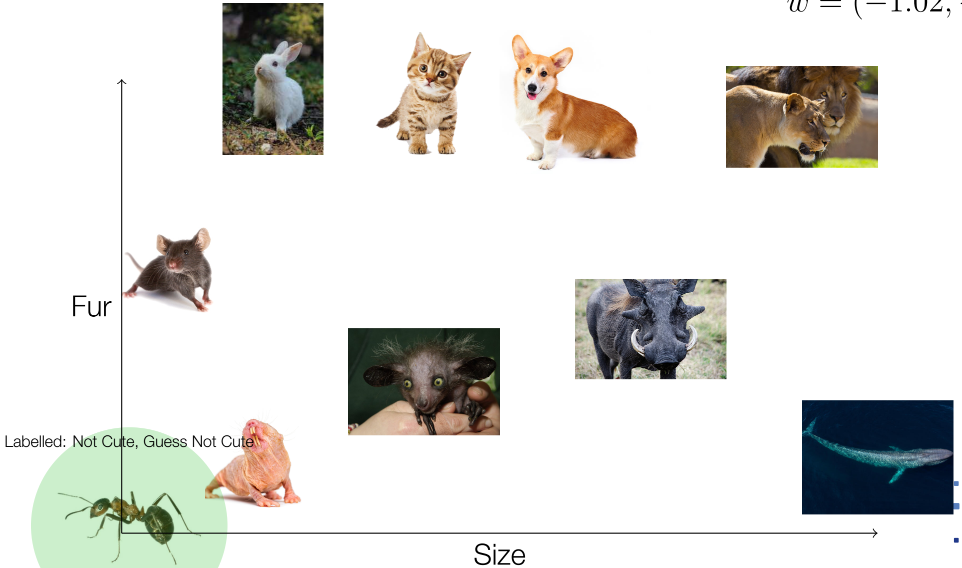
Training Perceptron

$$\vec{w} = (-1.02, -0.04, 0.12)$$



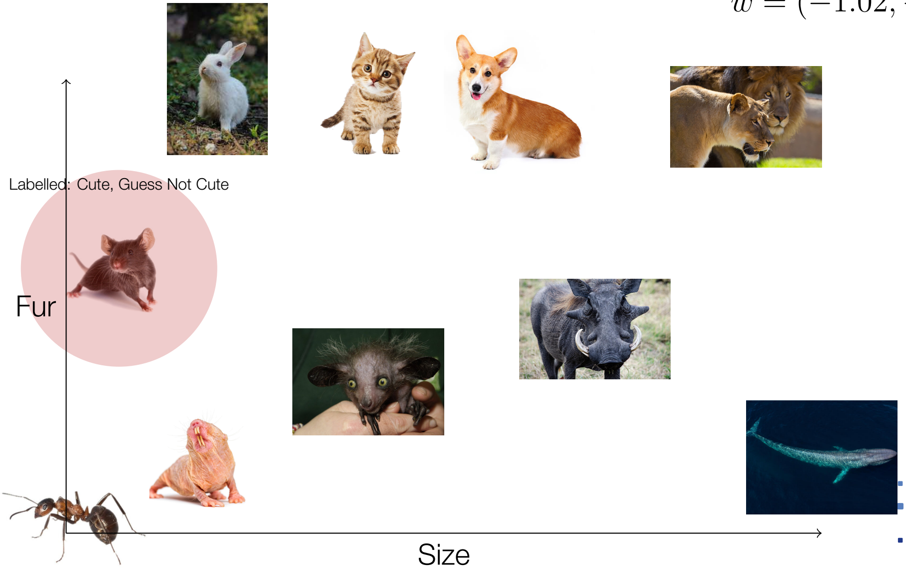
Training Perceptron

$$\vec{w} = (-1.02, -0.04, 0.12)$$

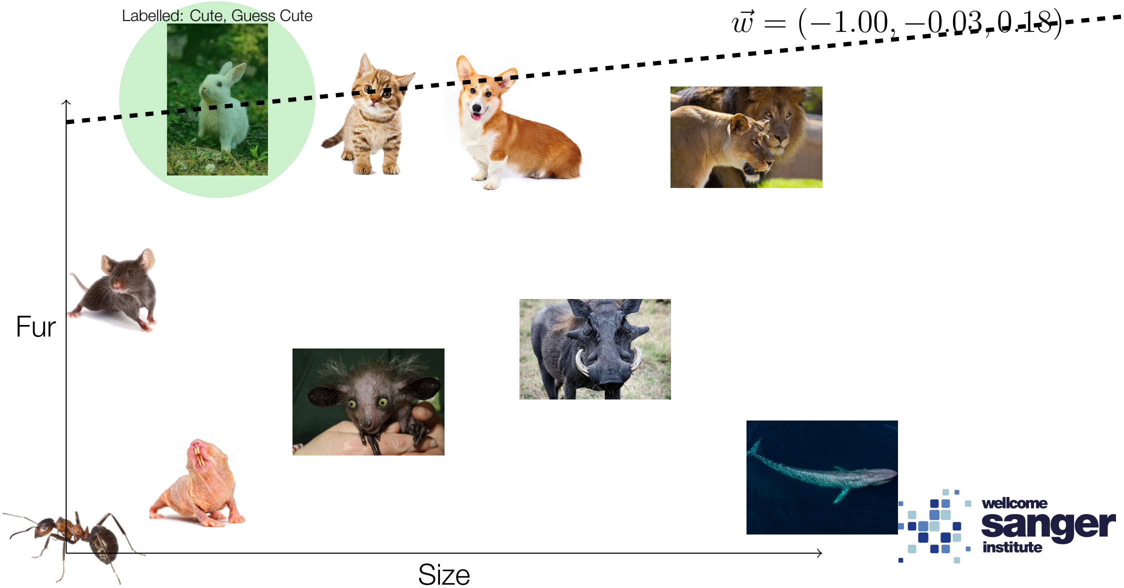


Training Perceptron

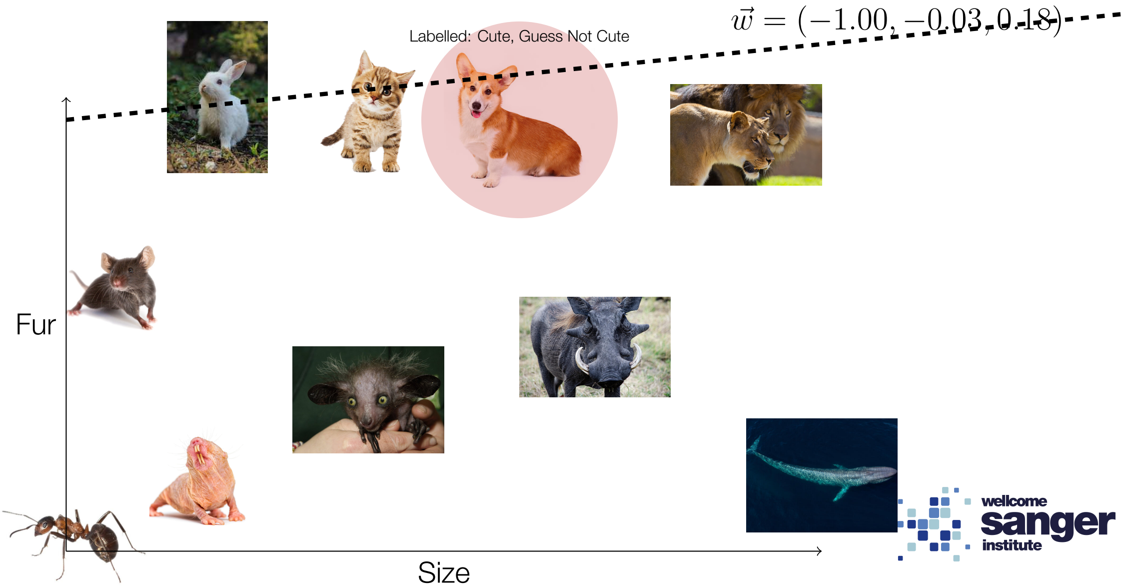
$$\vec{w} = (-1.02, -0.04, 0.12)$$



Training Perceptron

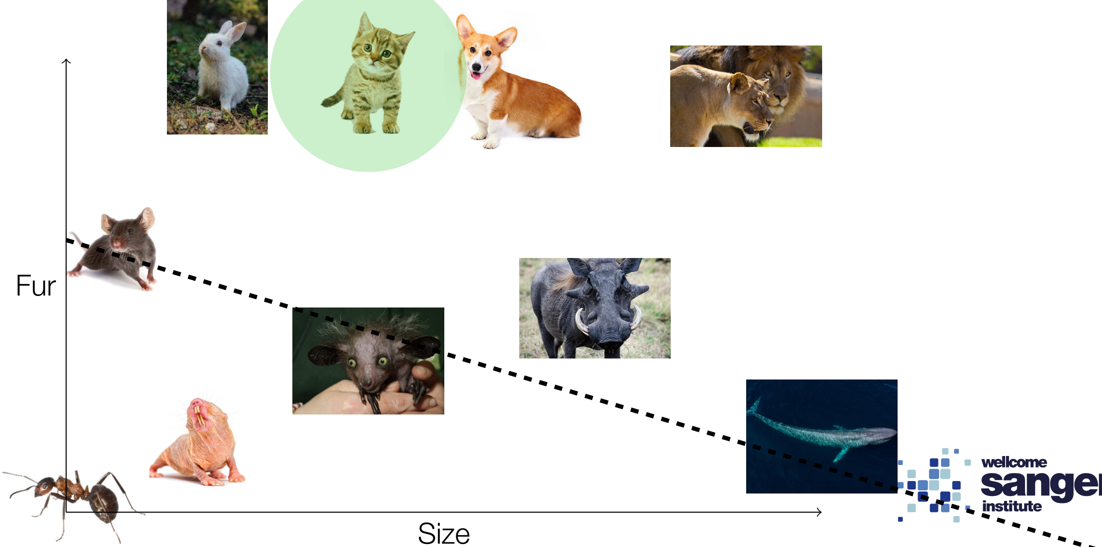


Training Perceptron



Training Perceptron

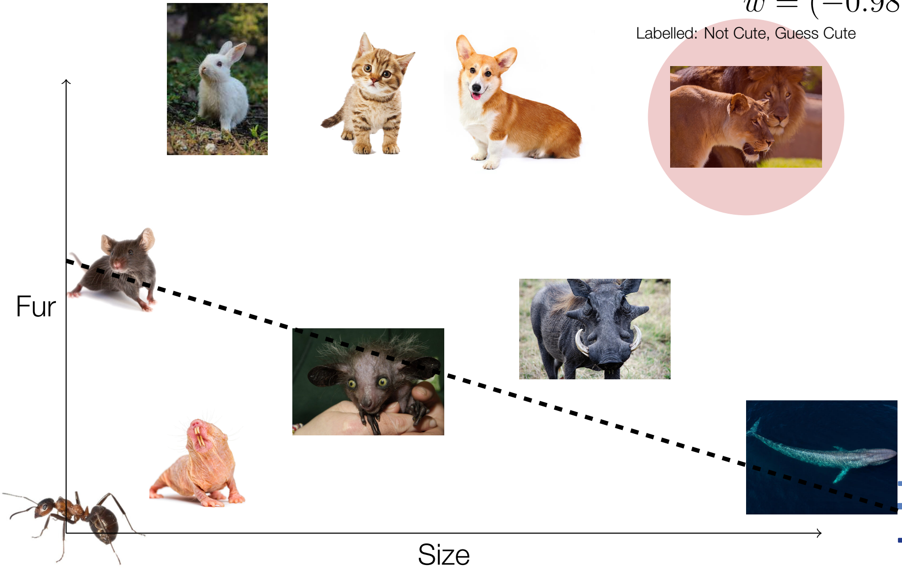
$$\vec{w} = (-0.98, 0.08, 0.27)$$



wellcome sanger institute

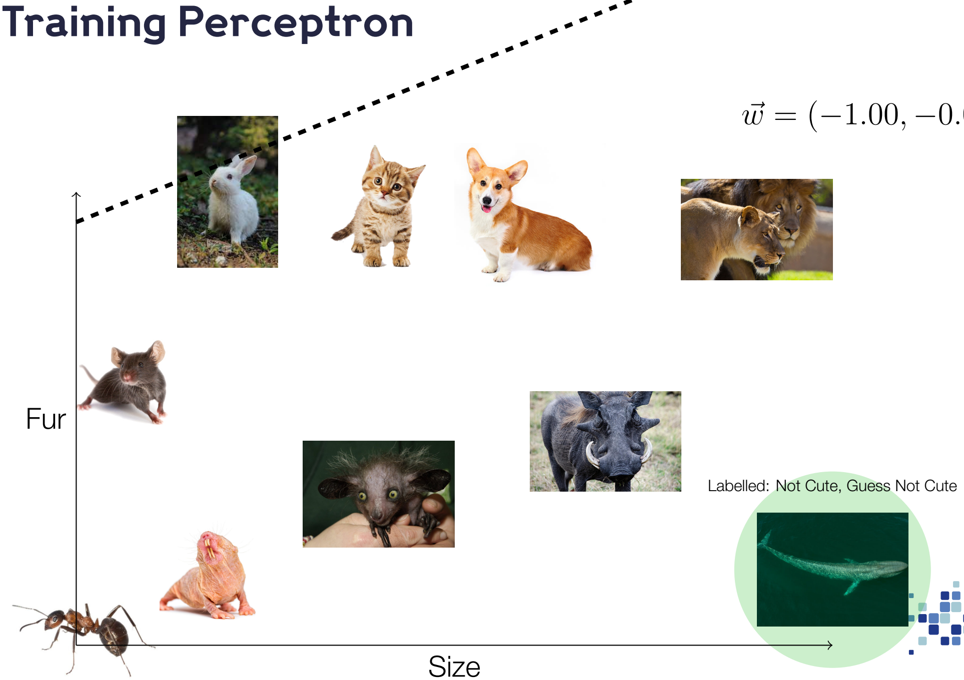
Training Perceptron

$$\vec{w} = (-0.98, 0.08, 0.27)$$



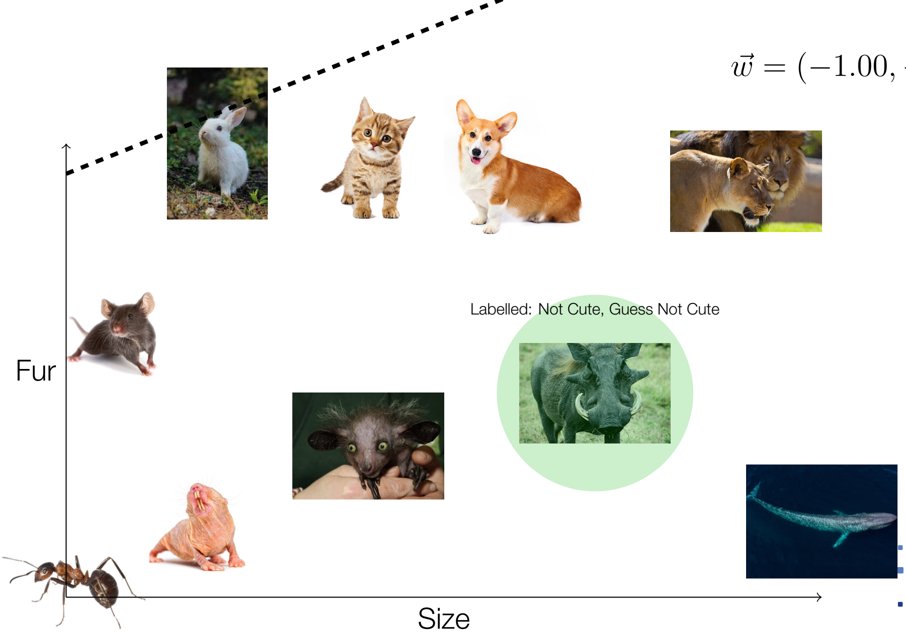
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$



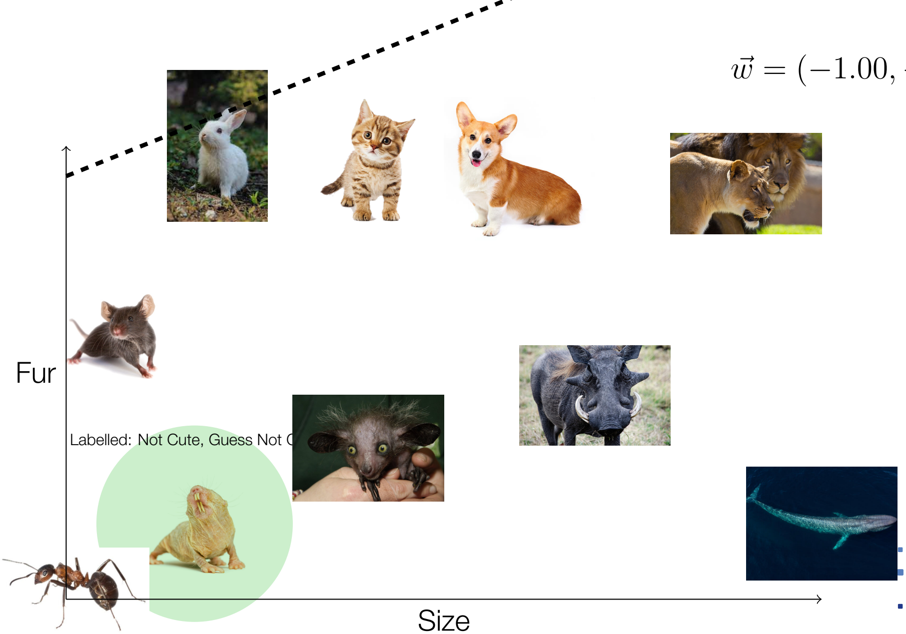
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$



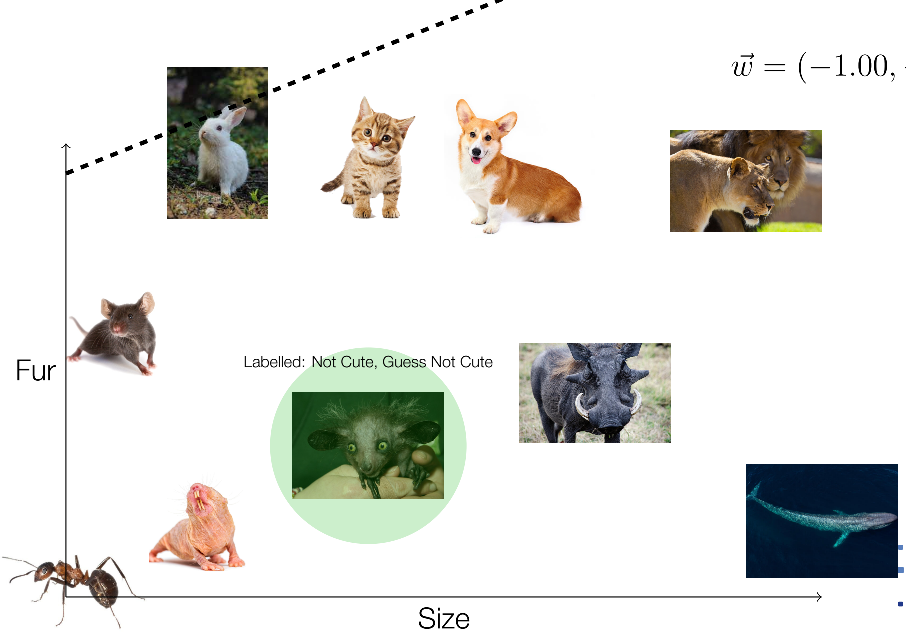
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$



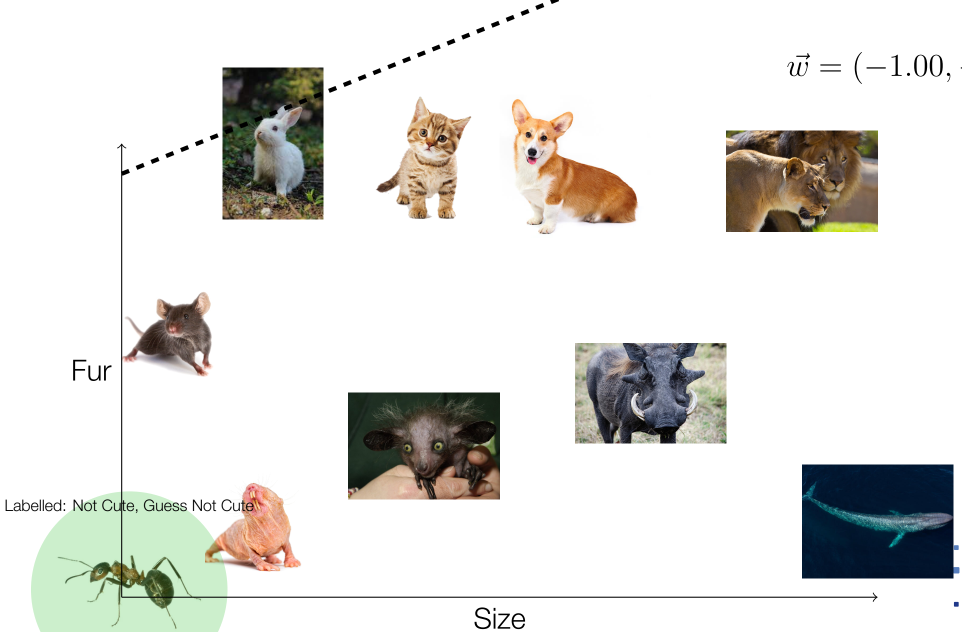
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$



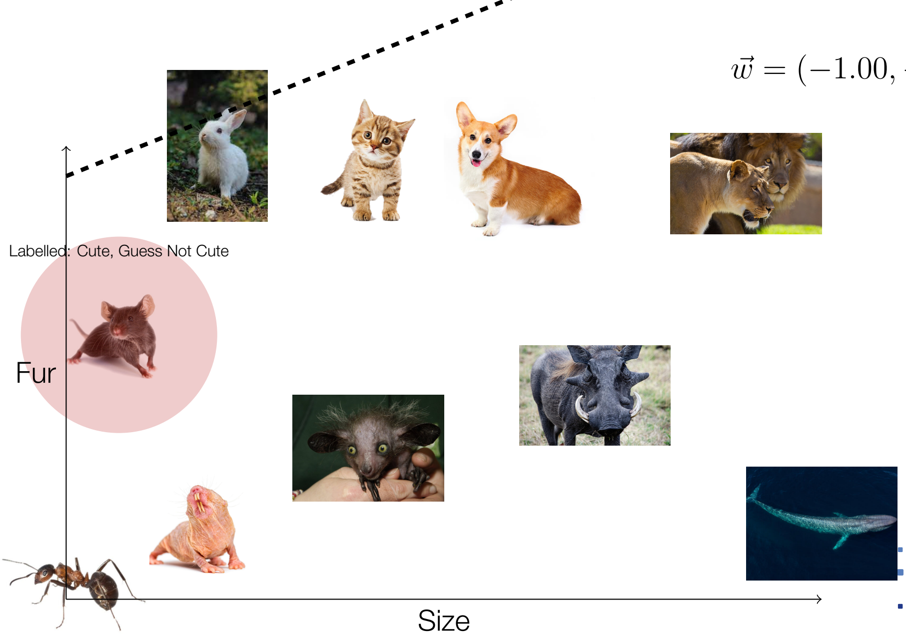
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$



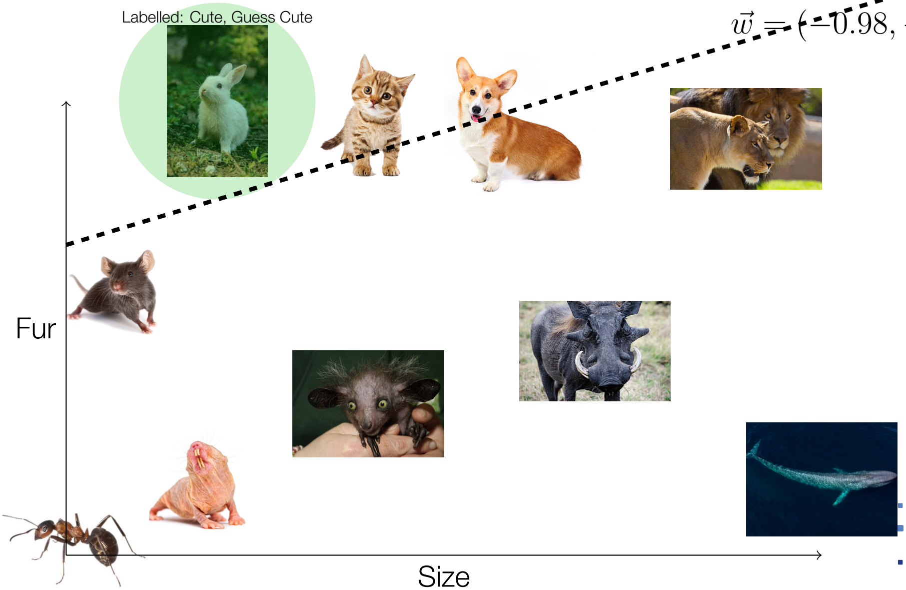
Training Perceptron

$$\vec{w} = (-1.00, -0.08, 0.18)$$

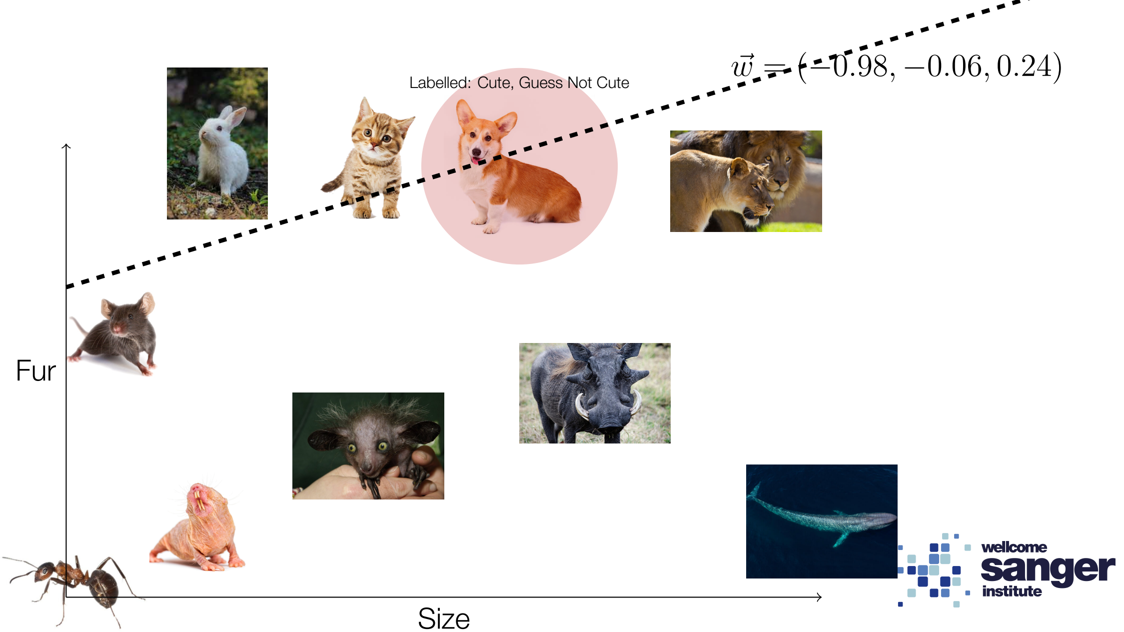


Training Perceptron

$\vec{w} = (-0.98, -0.06, 0.24)$

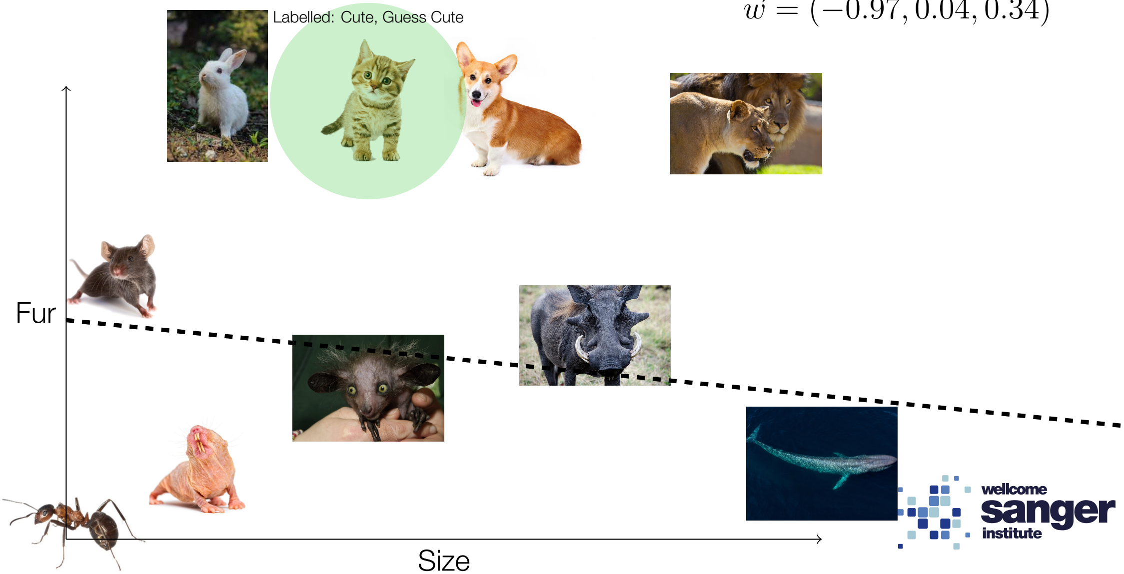


Training Perceptron



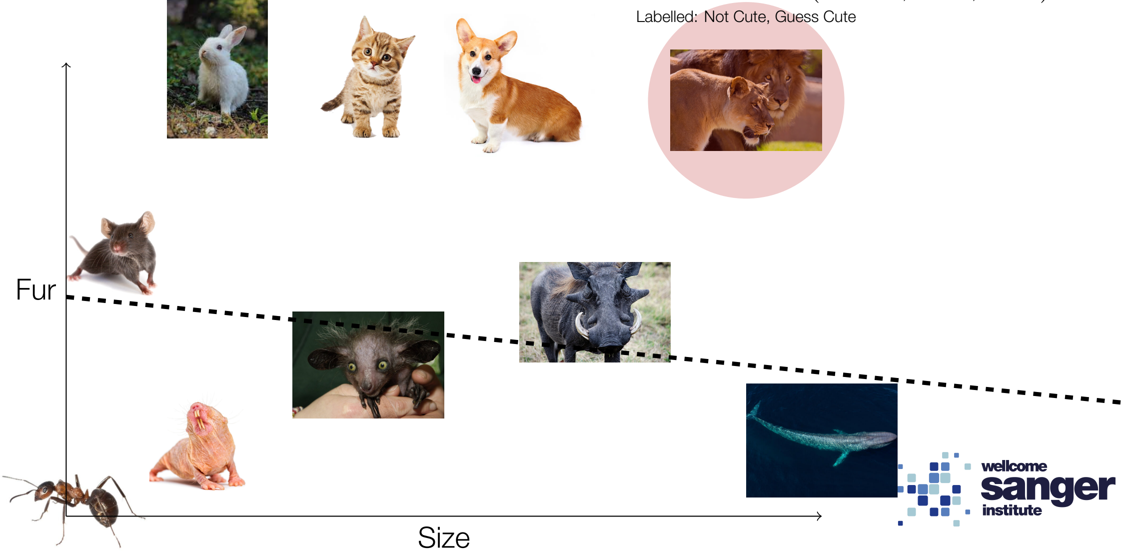
Training Perceptron

$$\vec{w} = (-0.97, 0.04, 0.34)$$



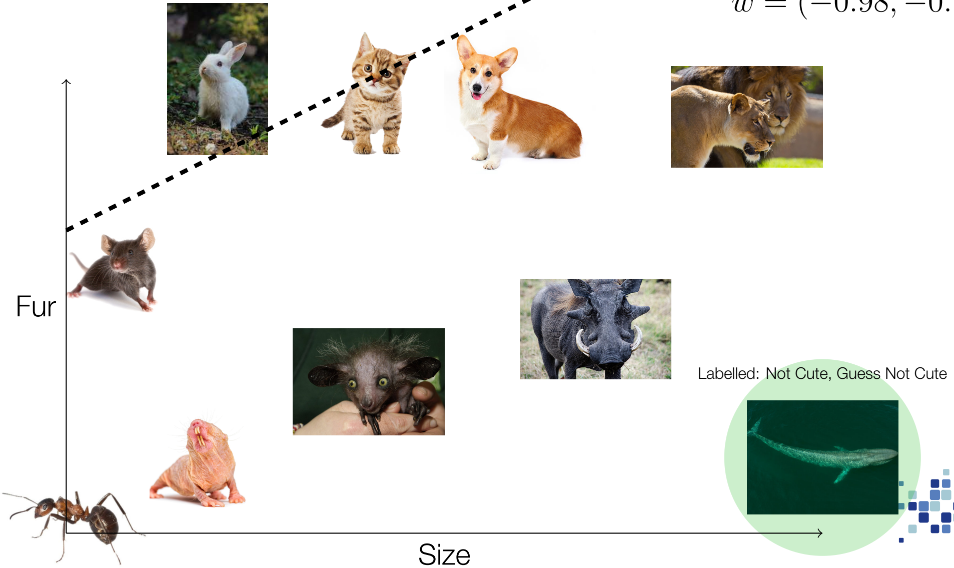
Training Perceptron

$$\vec{w} = (-0.97, 0.04, 0.34)$$



Training Perceptron

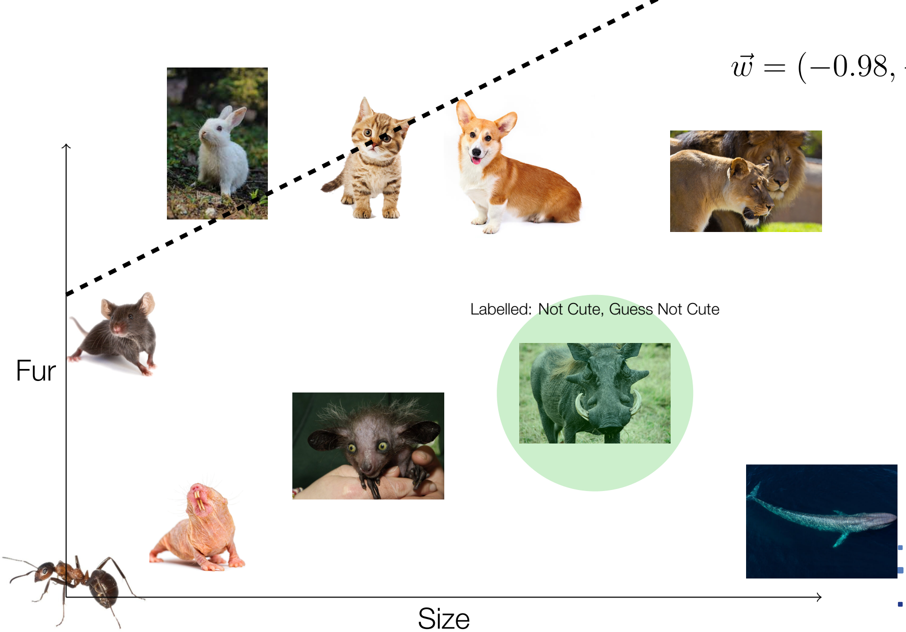
$$\vec{w} = (-0.98, -0.12, 0.24)$$



wellcome sanger institute

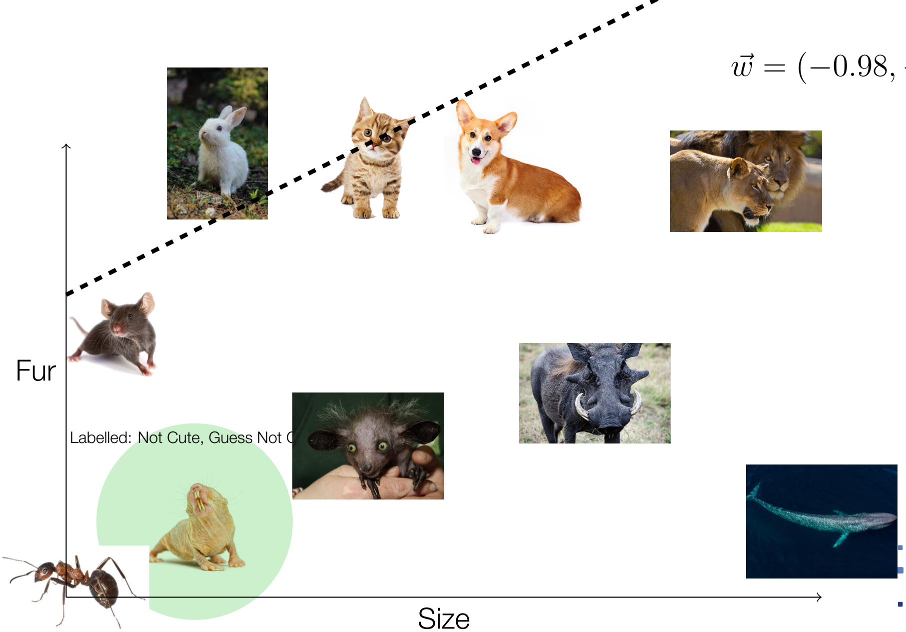
Training Perceptron

$$\vec{w} = (-0.98, -0.12, 0.24)$$



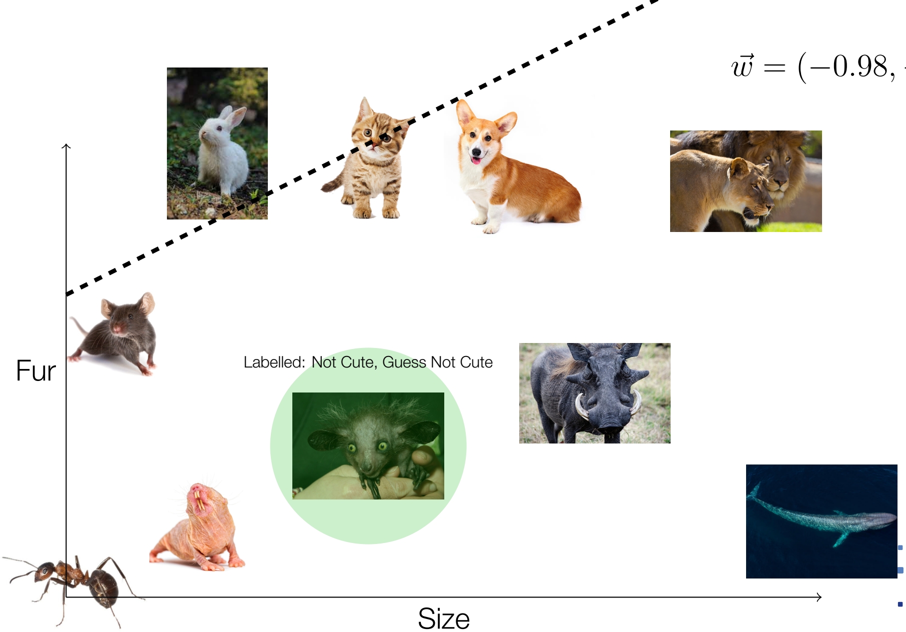
Training Perceptron

$$\vec{w} = (-0.98, -0.12, 0.24)$$



Training Perceptron

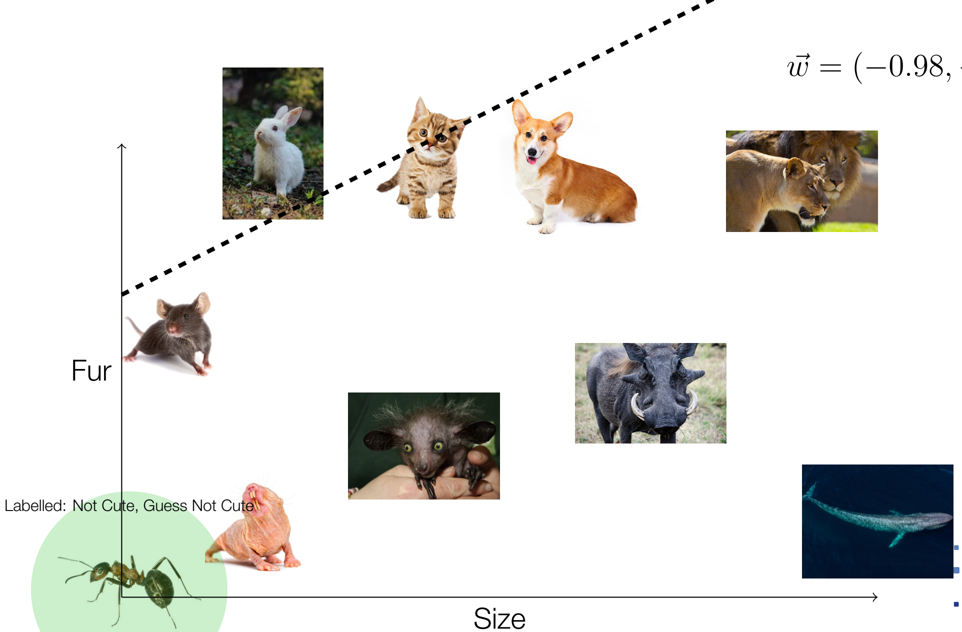
$$\vec{w} = (-0.98, -0.12, 0.24)$$



wellcome sanger institute

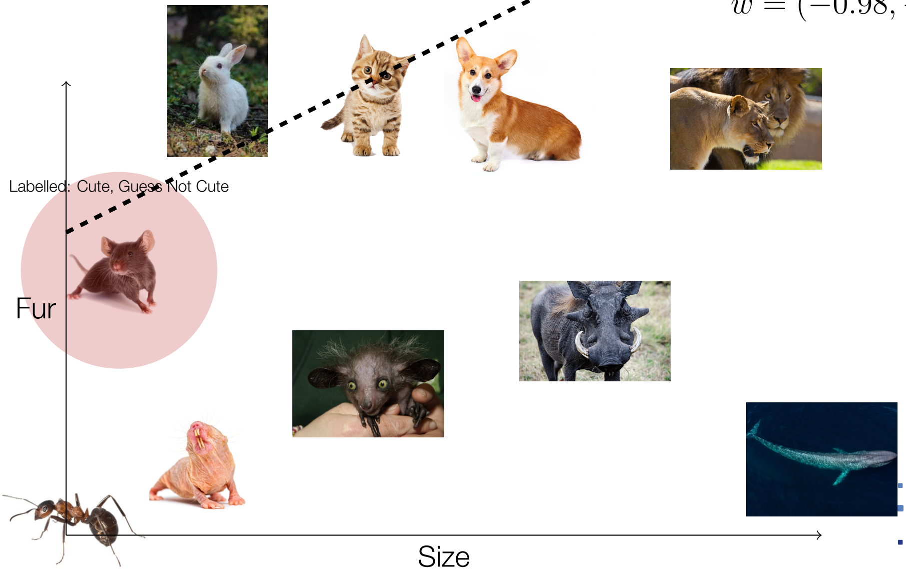
Training Perceptron

$$\vec{w} = (-0.98, -0.12, 0.24)$$

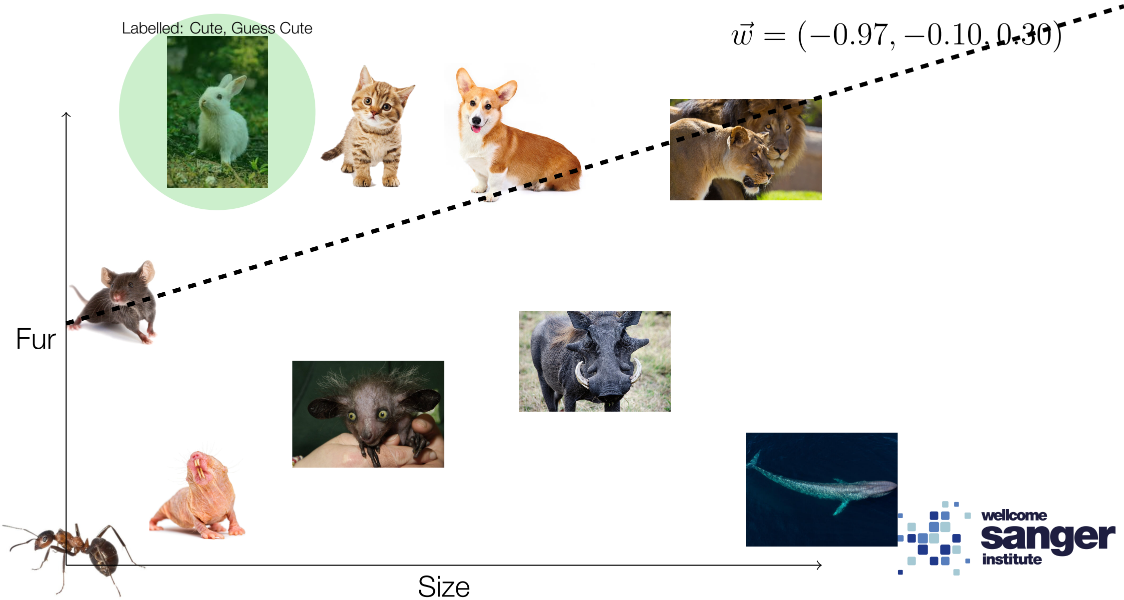


Training Perceptron

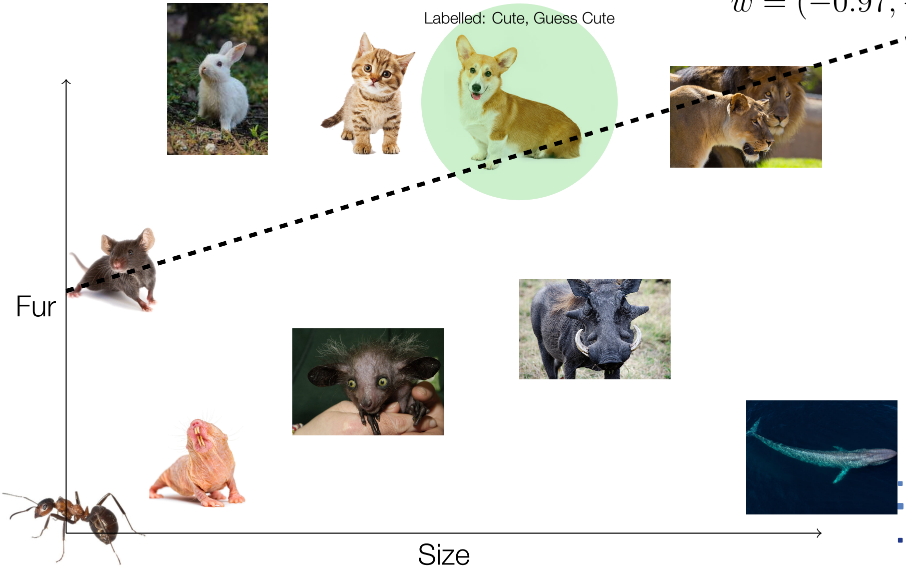
$$\vec{w} = (-0.98, -0.12, 0.24)$$



Training Perceptron

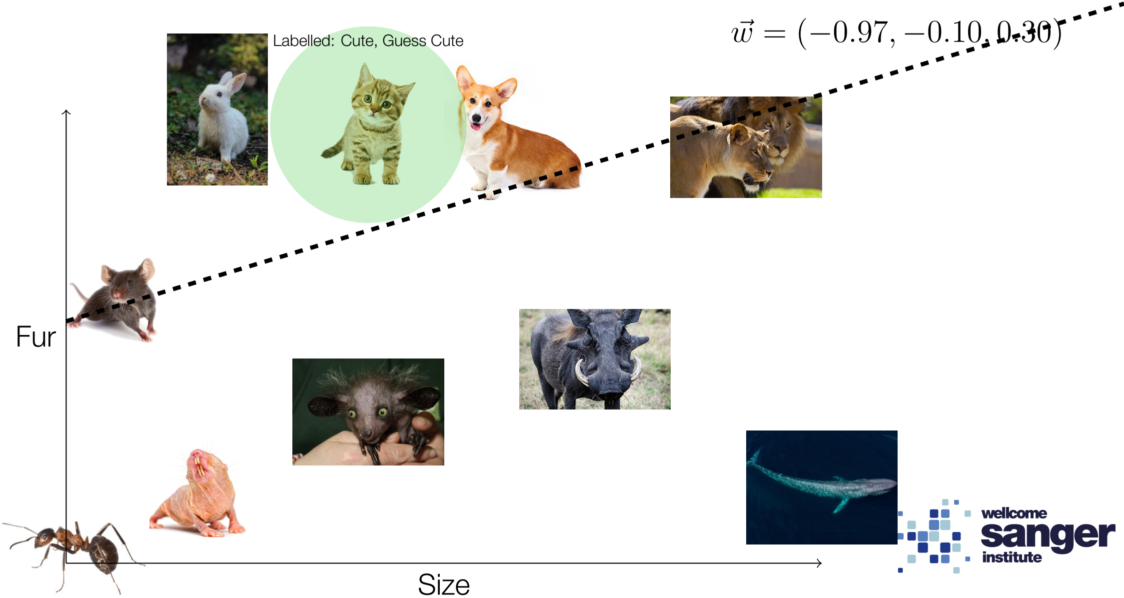


Training Perceptron

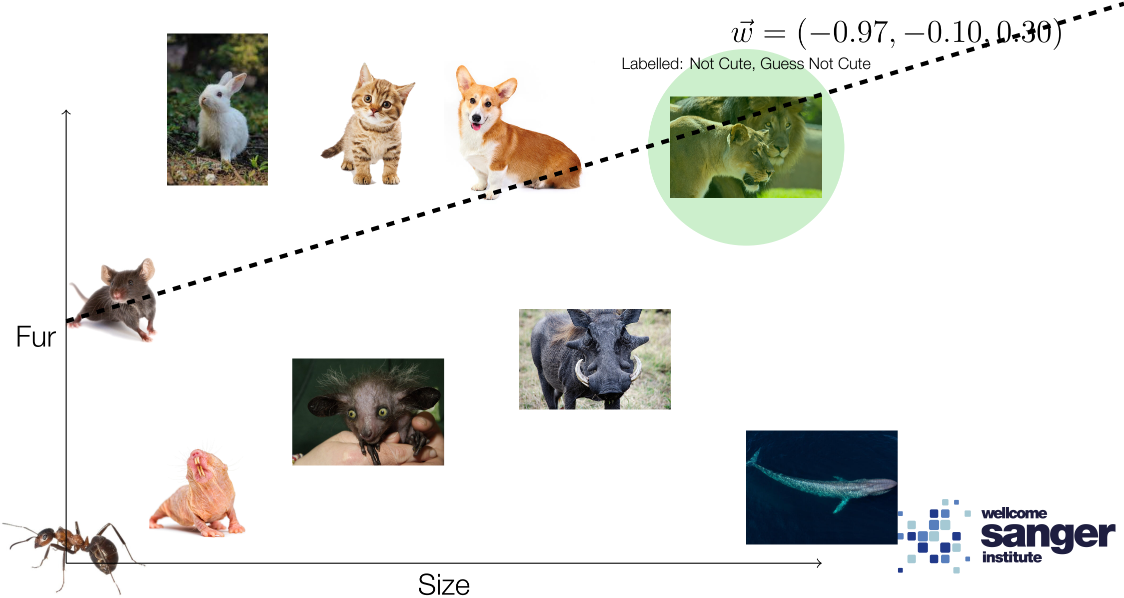


$$\vec{w} = (-0.97, -0.10, 0.30)$$

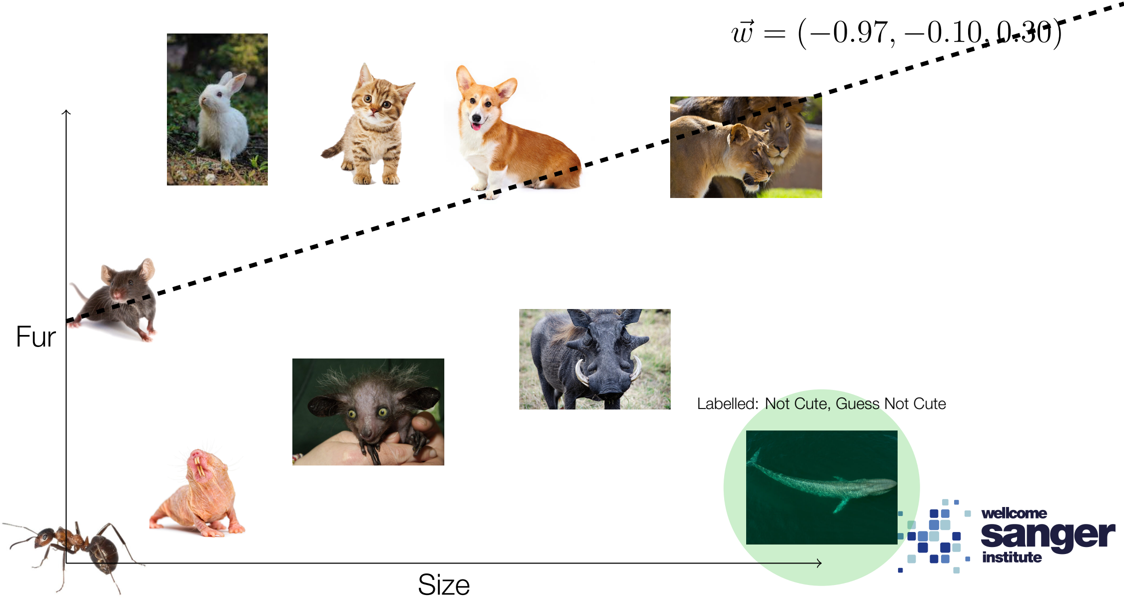
Training Perceptron



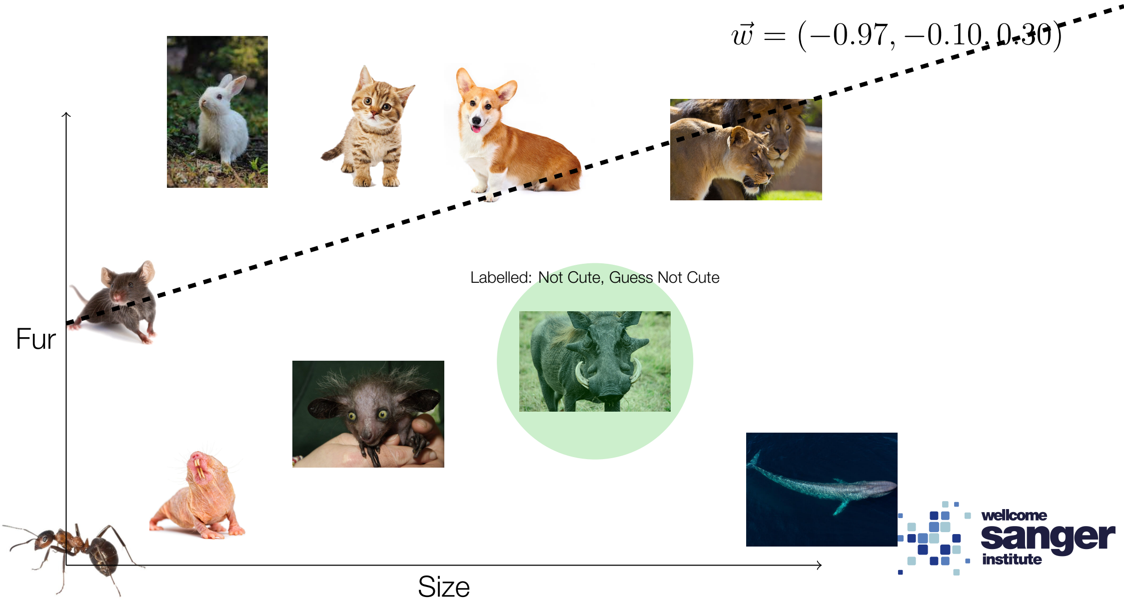
Training Perceptron



Training Perceptron

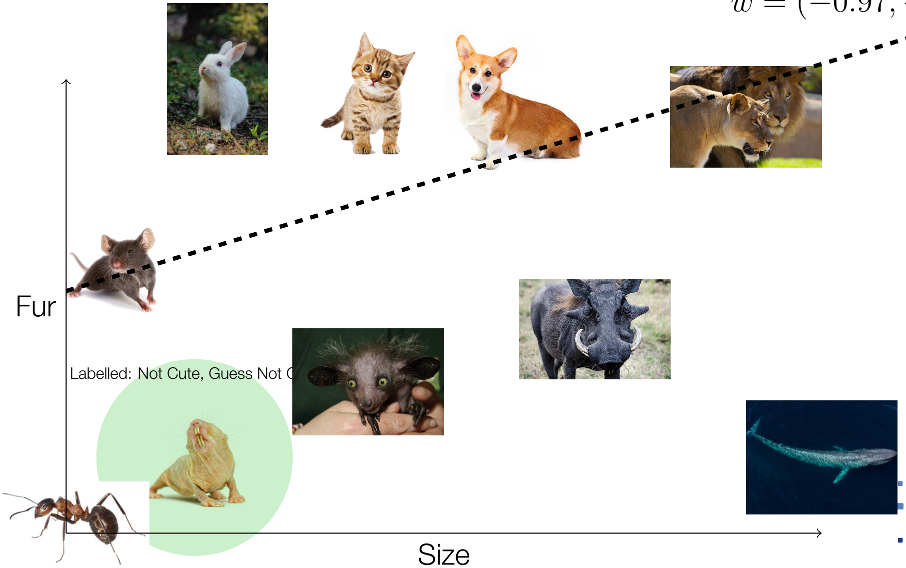


Training Perceptron

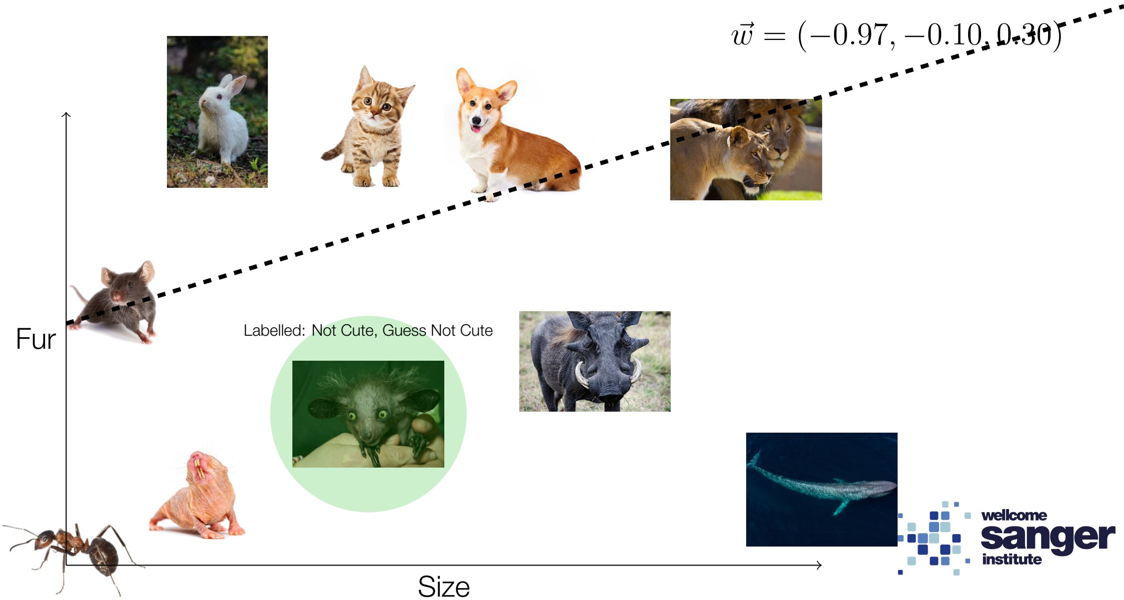


Training Perceptron

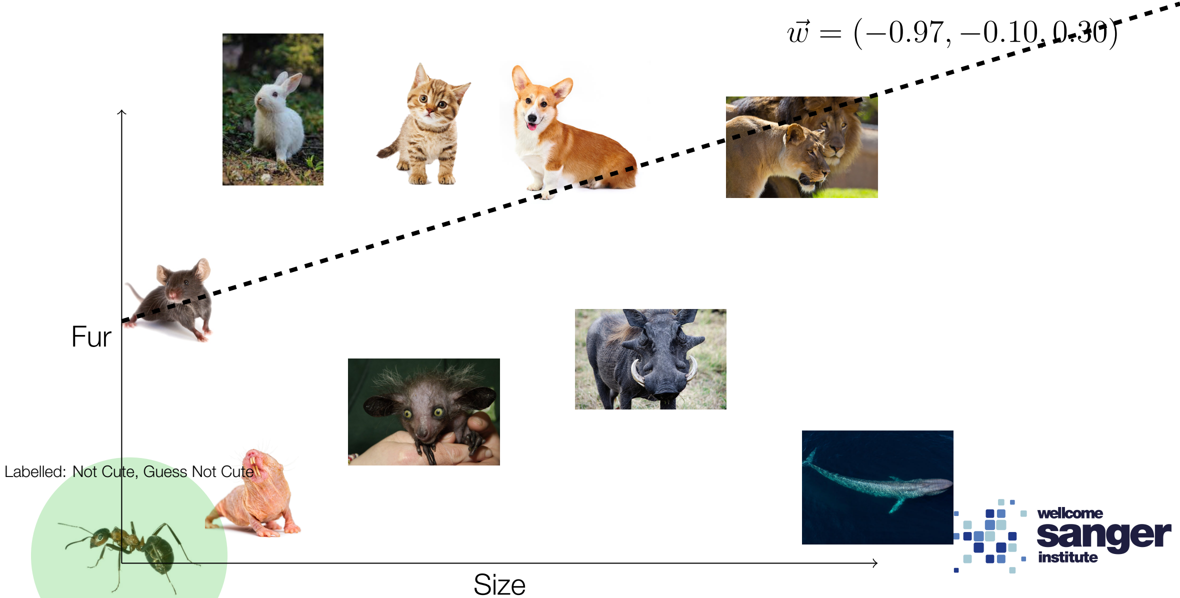
$\vec{w} = (-0.97, -0.10, 0.30)$



Training Perceptron

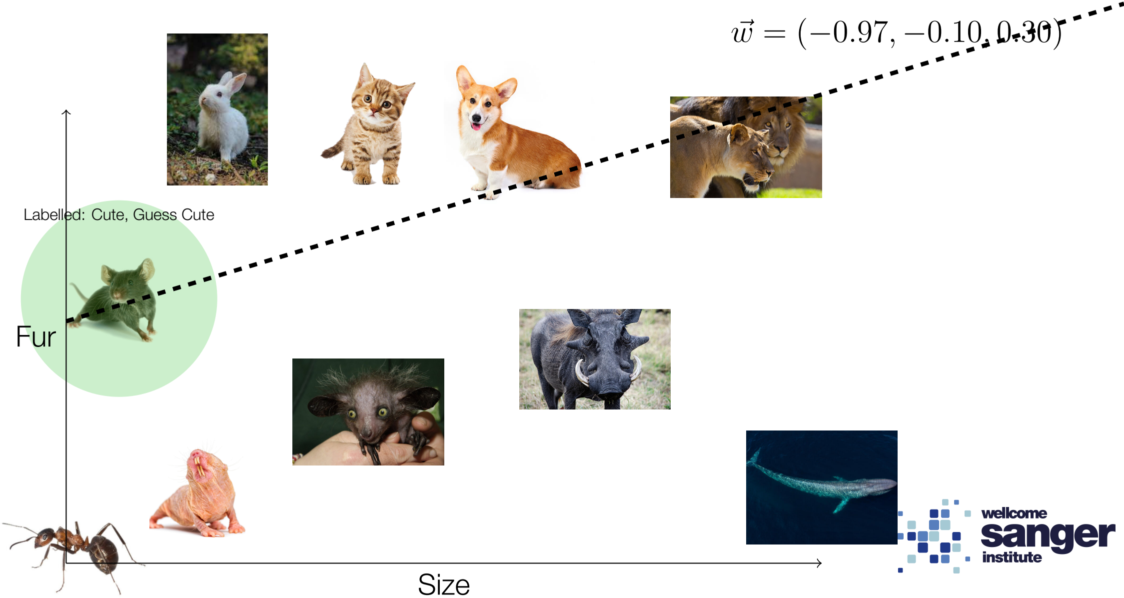


Training Perceptron

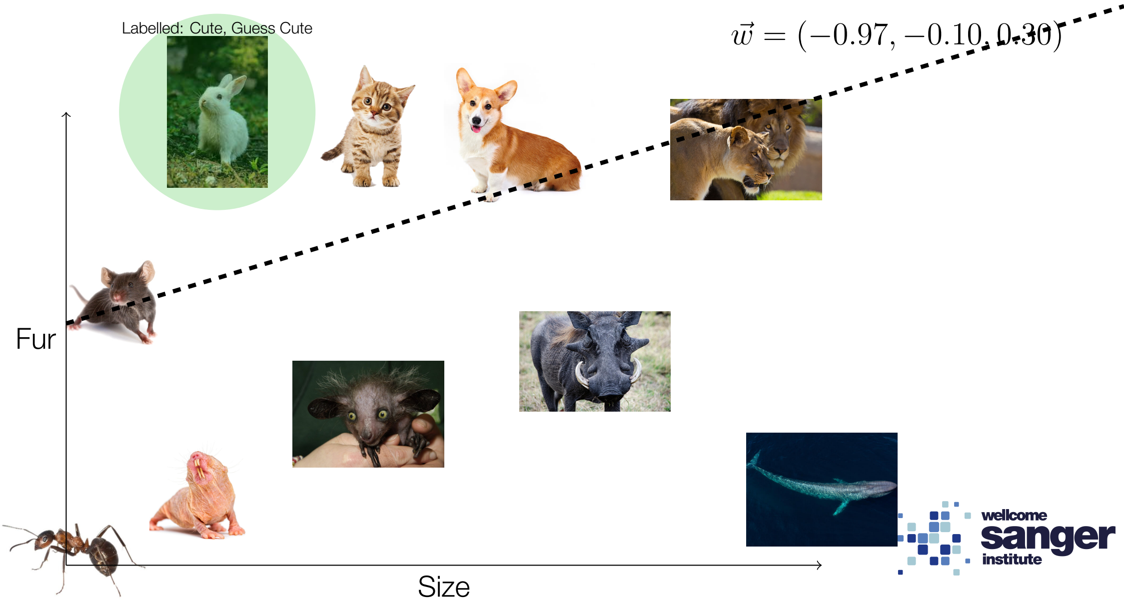


wellcome sanger institute

Training Perceptron

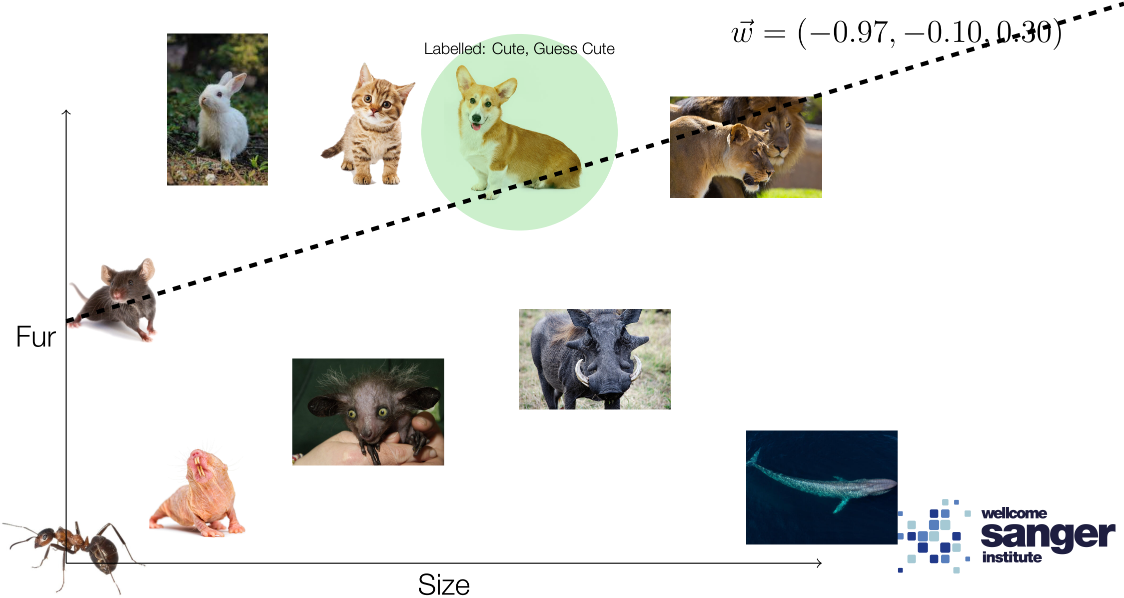


Training Perceptron

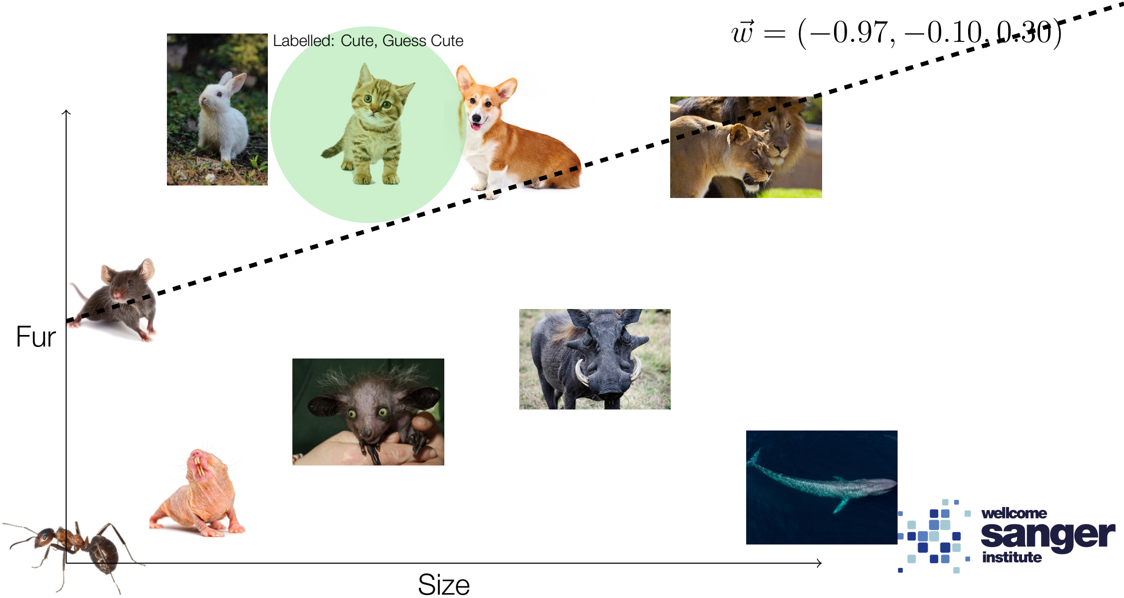


wellcome sanger institute

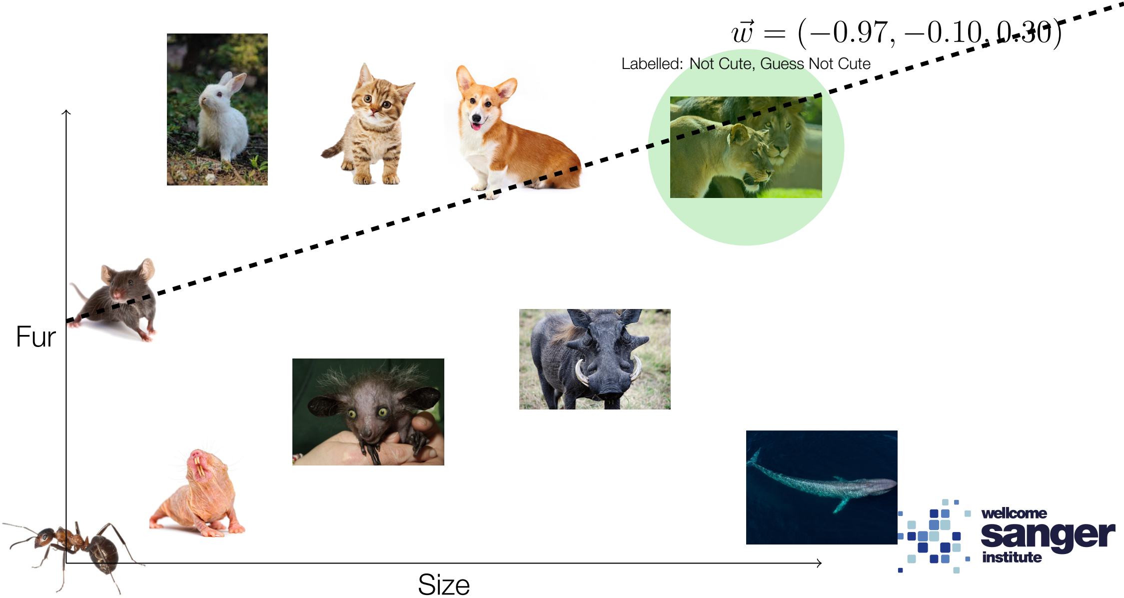
Training Perceptron



Training Perceptron

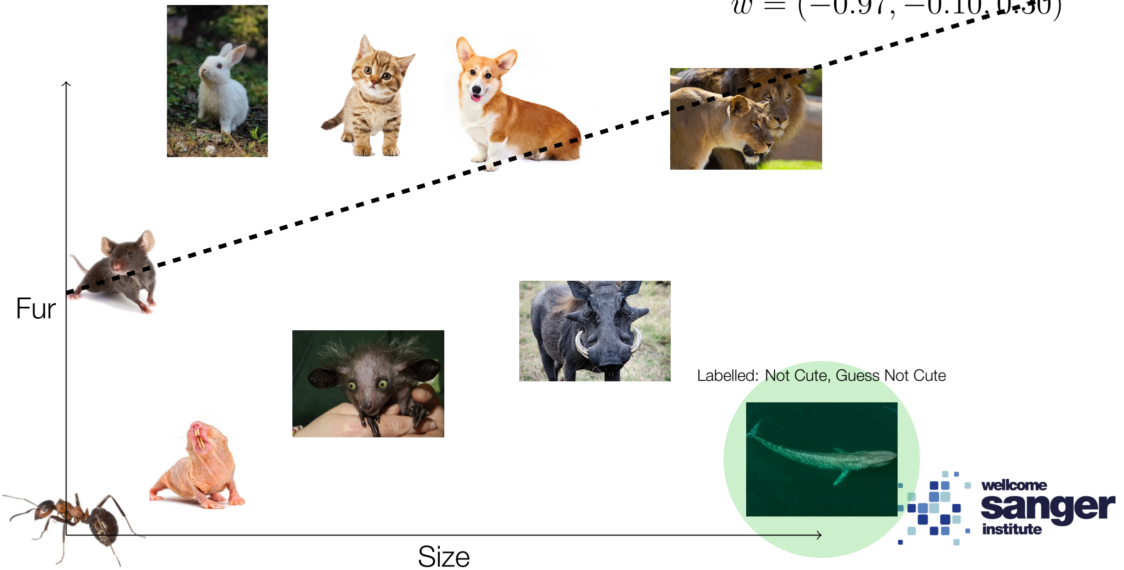


Training Perceptron

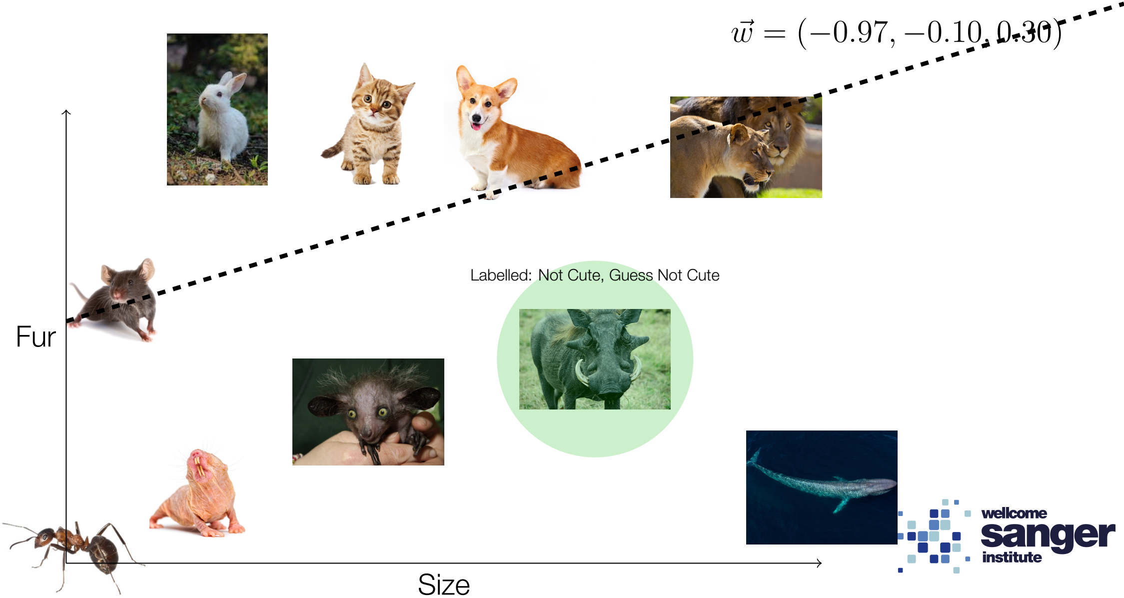


Training Perceptron

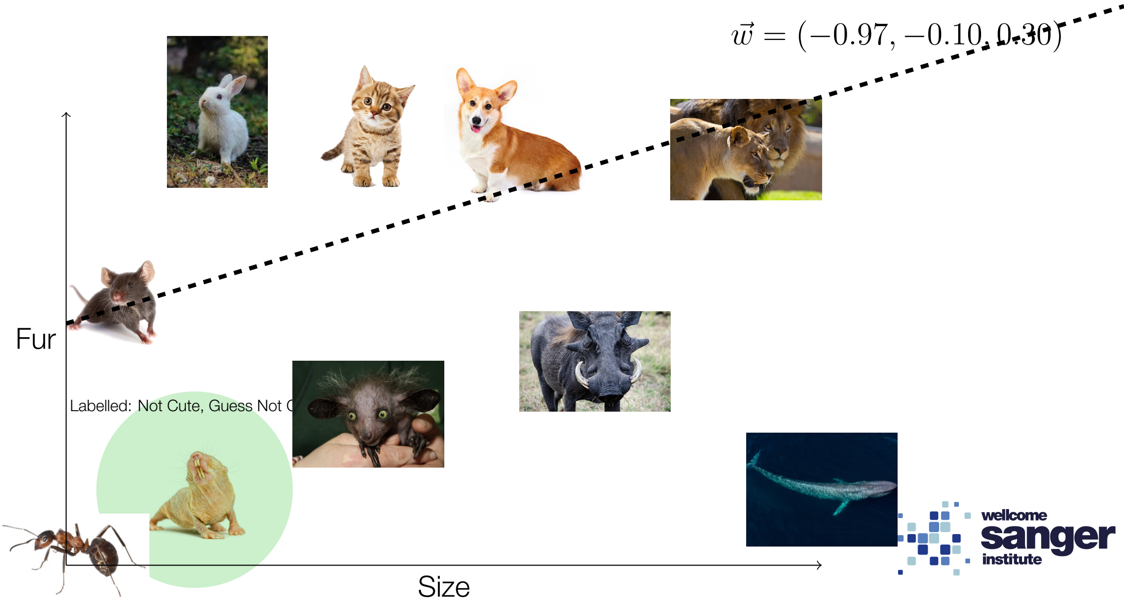
$$\vec{w} = (-0.97, -0.10, 0.30)$$



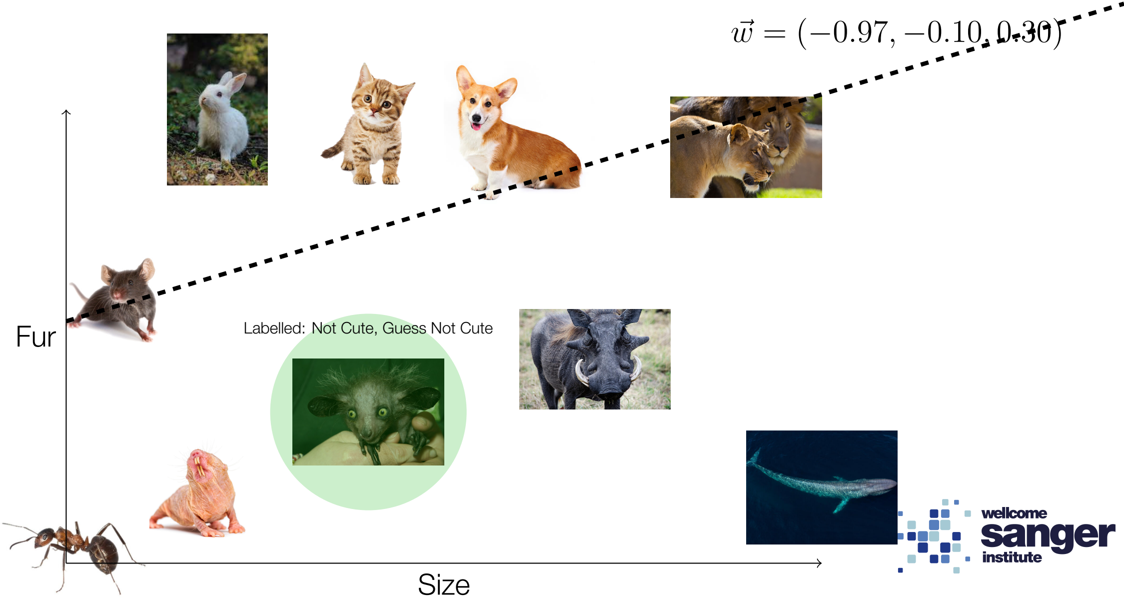
Training Perceptron



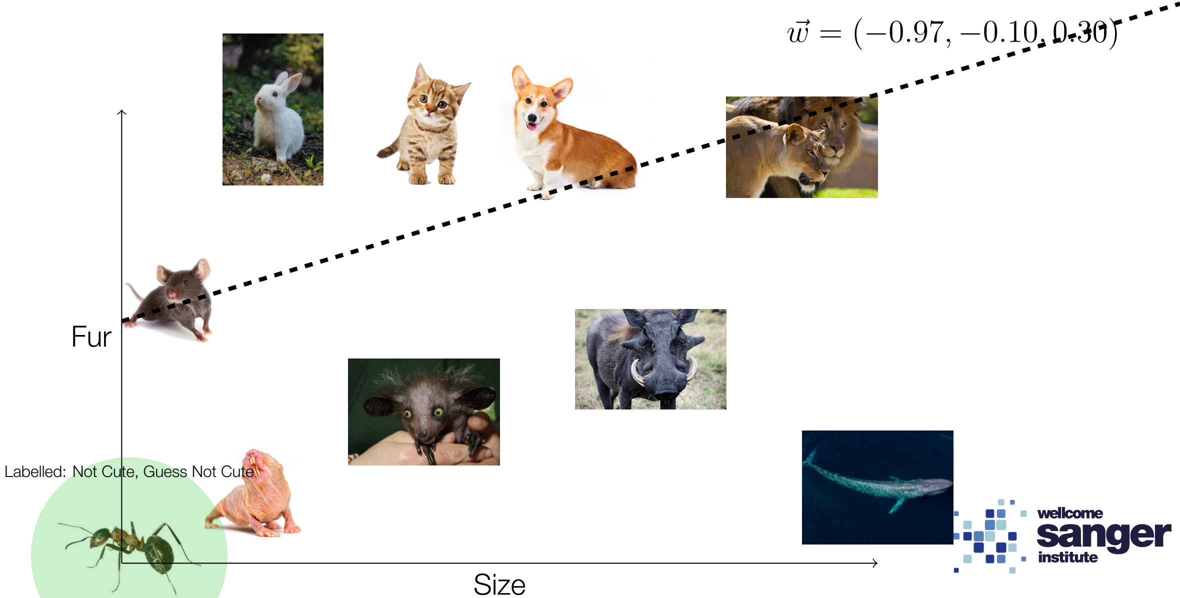
Training Perceptron



Training Perceptron



Training Perceptron



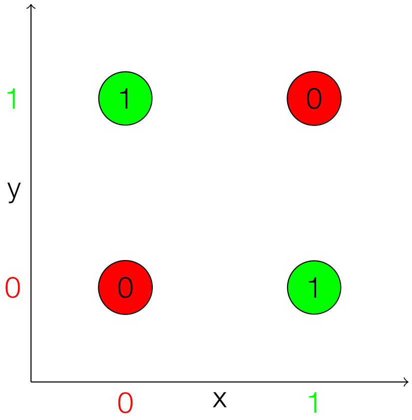
Getting clever....

Can get non-straight lines by using $(x, x^2, x^3, y, xy, \cos(y^2 x^3), \ldots)$

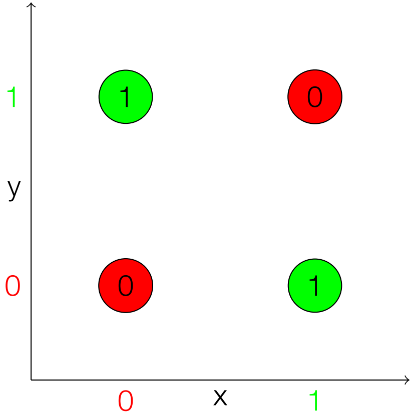
Getting clever....

Can get non-straight lines by using $(x, x^2, x^3, y, xy, \cos(y^2x^3), \ldots)$
But needs pre-defined user input!

Major limitations....



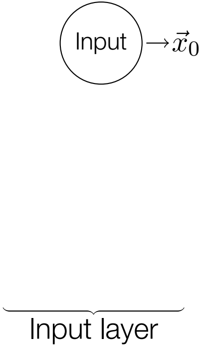
Major limitations....



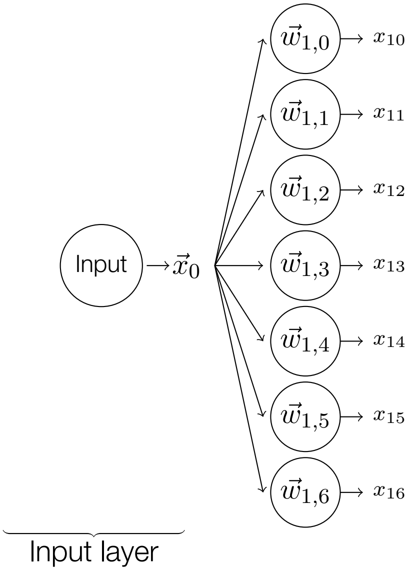
Can **never** be solved by perceptron

The Multilayered Perceptron

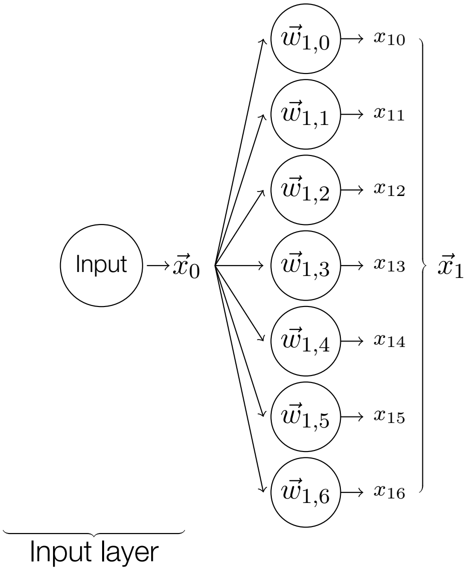
The Multilayered Perceptron



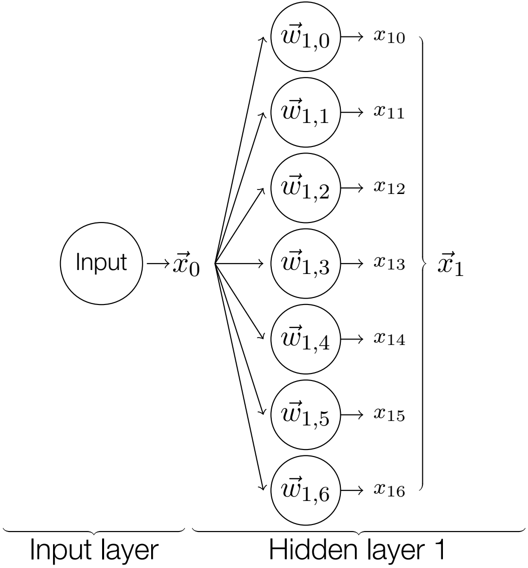
The Multilayered Perceptron



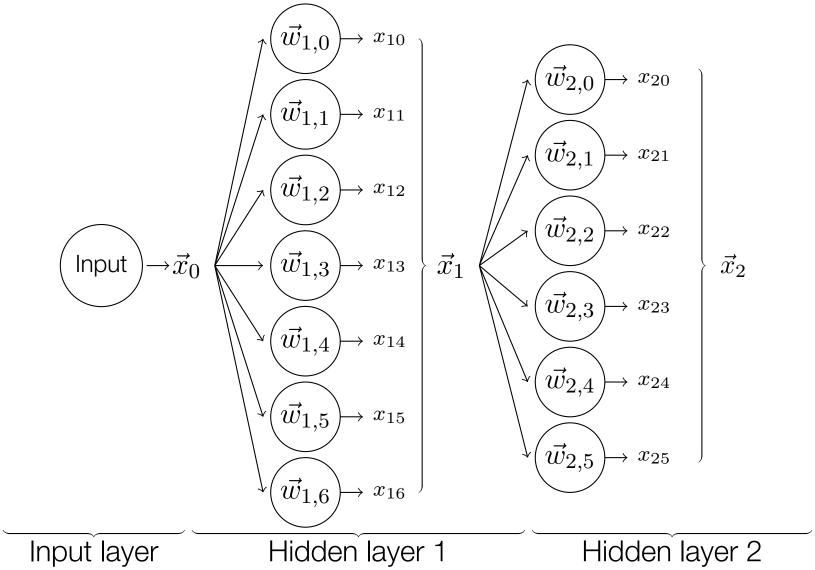
The Multilayered Perceptron



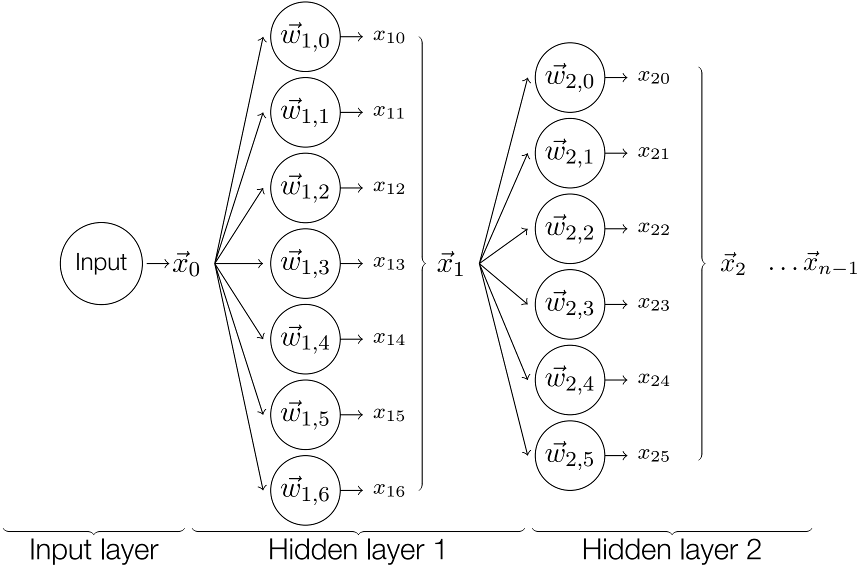
The Multilayered Perceptron



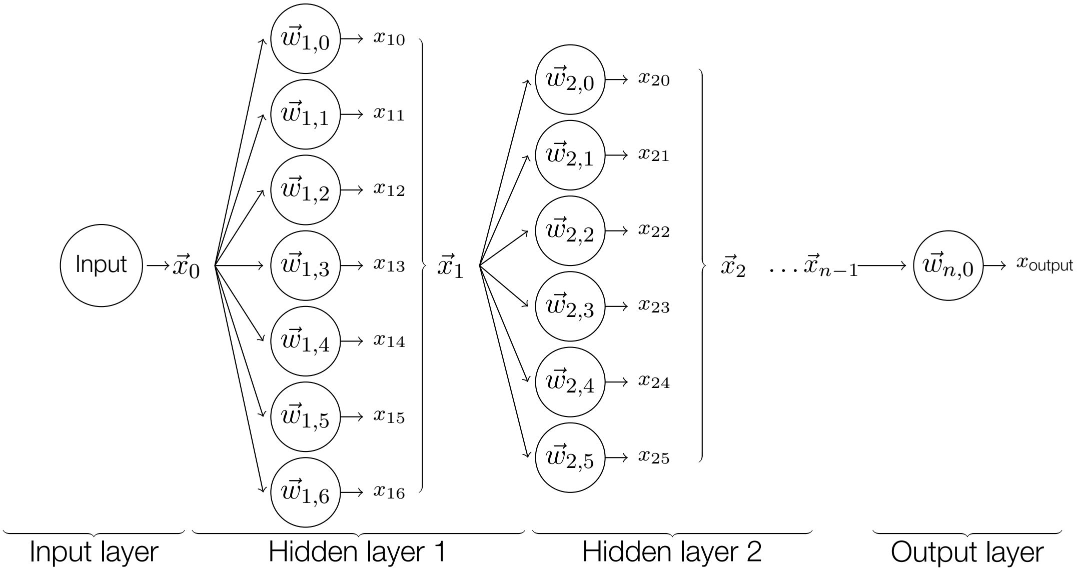
The Multilayered Perceptron



The Multilayered Perceptron



The Multilayered Perceptron



The Multilayered Perceptron

Perfect, right?

The Multilayered Perceptron

Perfect, right?

$$x_{10} = \sum_i \left[\vec{w}_{1,0}\right]_i \left[\vec{x}_0\right]_i \iff \vec{x}_1 = W_1 \vec{x}_0 \tag{3}$$

The Multilayered Perceptron

Perfect, right?

$$x_{10} = \sum_i \left[\vec{w}_{1,0}\right]_i \left[\vec{x}_0\right]_i \quad \Longleftrightarrow \quad \vec{x}_1 = W_1 \vec{x}_0 \tag{3}$$

And therefore:

$$\vec{x}_n = \left(\prod_{j=1}^{n} W_j \right) \vec{x}_0 \tag{4}$$

The Multilayered Perceptron

Perfect, right?

$$x_{10} = \sum_i \left[\vec{w}_{1,0}\right]_i \left[\vec{x}_0\right]_i \iff \vec{x}_1 = W_1 \vec{x}_0 \tag{3}$$

And therefore:

$$\vec{x}_n = \left(\prod_{j=1}^{n} W_j\right) \vec{x}_0 \tag{4}$$

Just one giant matrix product: still fundamentally linear!

wellcome sanger institute

The Multilayered Perceptron

Perfect, right?

$$x_{10} = \sum_i \left[\vec{w}_{1,0}\right]_i \left[\vec{x}_0\right]_i \quad \Longleftrightarrow \quad \vec{x}_1 = W_1 \vec{x}_0 \tag{3}$$

And therefore:

$$\vec{x}_n = \left(\prod_{j=1}^{n} W_j\right) \vec{x}_0 \tag{4}$$

Just one giant matrix product: still fundamentally linear!
For our ‘cute’ problem, even a 10,000 layer network with 5 million nodes will ultimately reduce into a single 2×1 matrix (i.e. a single dot product).

Breaking Linearity

Introduce the idea of ‘activation functions’:

$$x_{n,i} = \phi\left(\vec{w}_{n,i} \cdot \vec{x}_{n-1}\right) \tag{5}$$

ϕ has following properties:

Breaking Linearity

Introduce the idea of ‘activation functions’:

$$x_{n,i} = \phi\left(\vec{w}_{n,i} \cdot \vec{x}_{n-1}\right) \tag{5}$$

ϕ has following properties:

- A simple 1:1 function

Breaking Linearity

Introduce the idea of ‘activation functions’:

$$x_{n,i} = \phi\left(\vec{w}_{n,i} \cdot \vec{x}_{n-1}\right) \tag{5}$$

ϕ has following properties:

- A simple 1:1 function
- Non-linear

Breaking Linearity

Introduce the idea of ‘activation functions’:

$$x_{n,i} = \phi\left(\vec{w}_{n,i} \cdot \vec{x}_{n-1}\right) \tag{5}$$

ϕ has following properties:

- A simple 1:1 function
- Non-linear
- Works on each element of the vector individually

Breaking Linearity

Introduce the idea of ‘activation functions’:

$$x_{n,i} = \phi\left(\vec{w}_{n,i} \cdot \vec{x}_{n-1}\right) \tag{5}$$

ϕ has following properties:

- A simple 1:1 function
- Non-linear
- Works on each element of the vector individually

In practice, we like some other properties (analytical derivatives, easy to compute etc.)

wellcome sanger institute

Activation Functions

Some common choices:

Activation Functions

Some common choices:

- Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0 & \text{else} \end{cases}$$

Activation Functions

Some common choices:

- Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0 & \text{else} \end{cases}$$

- Leaky ReLU

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0.01x & \text{else} \end{cases}$$

wellcome sanger institute

Activation Functions

Some common choices:

- Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0 & \text{else} \end{cases}$$

- Leaky ReLU

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0.01x & \text{else} \end{cases}$$

- Sigmoid

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

wellcome sanger institute

Activation Functions

Some common choices:

- Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0 & \text{else} \end{cases}$$

- Leaky ReLU

$$\phi(x) = \begin{cases} x & \text{when } x > 0 \\ 0.01x & \text{else} \end{cases}$$

- Sigmoid

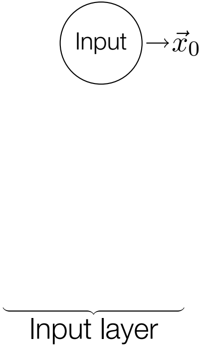
$$\phi(x) = \frac{1}{1 + e^{-x}}$$

- Softplus

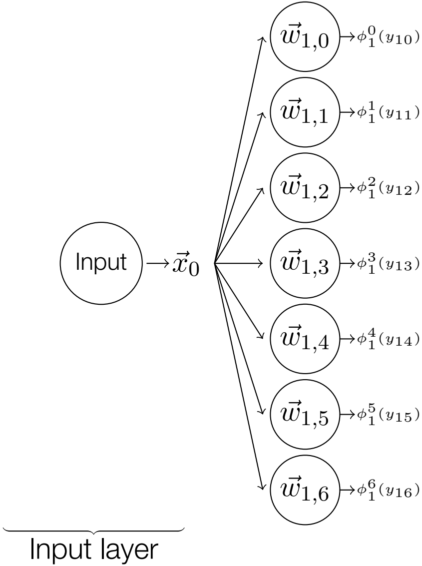
$$\phi(x) = \ln\left(1 + e^{x}\right)$$

The Multilayered Perceptron

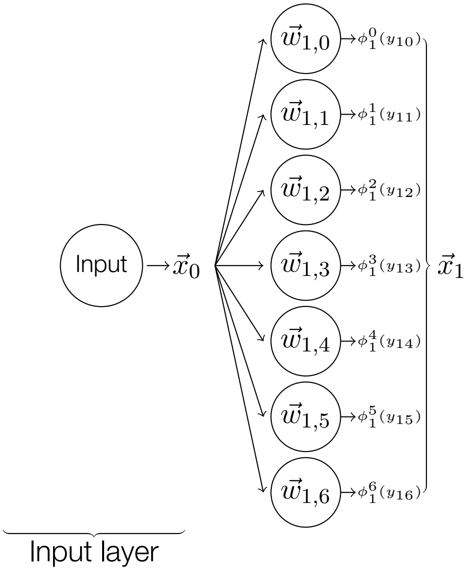
The Multilayered Perceptron



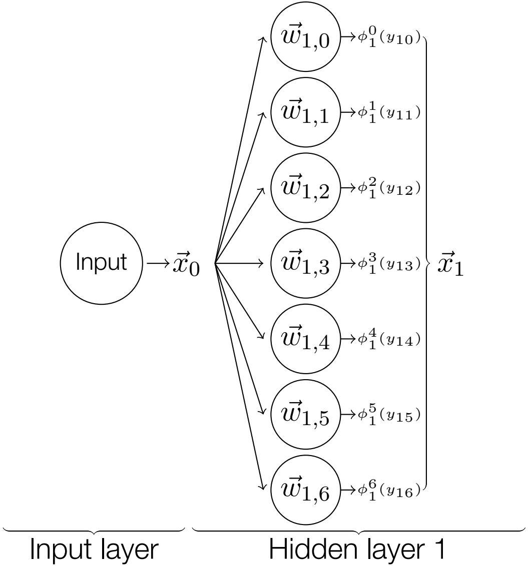
The Multilayered Perceptron



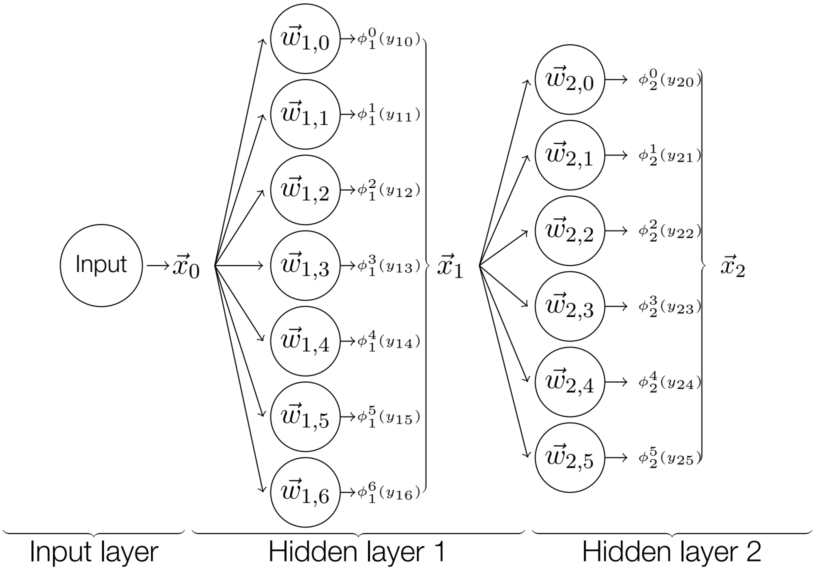
The Multilayered Perceptron



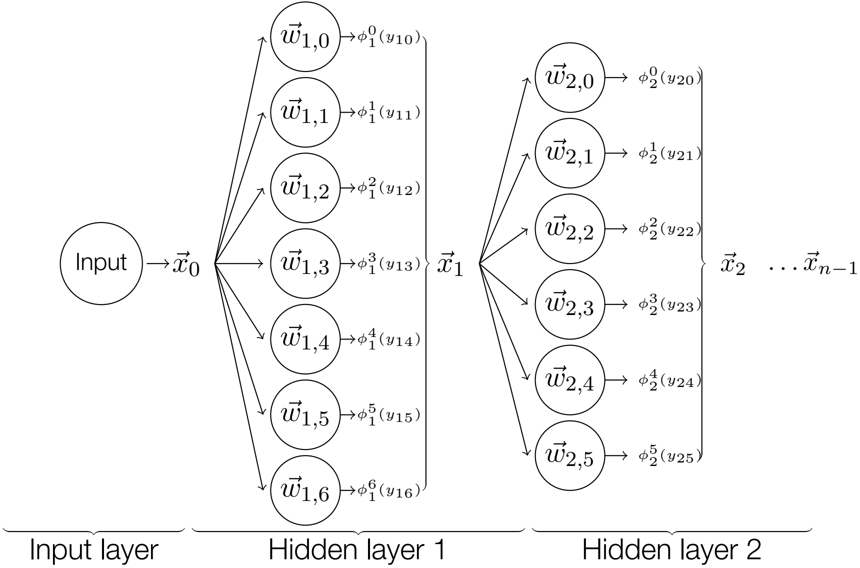
The Multilayered Perceptron



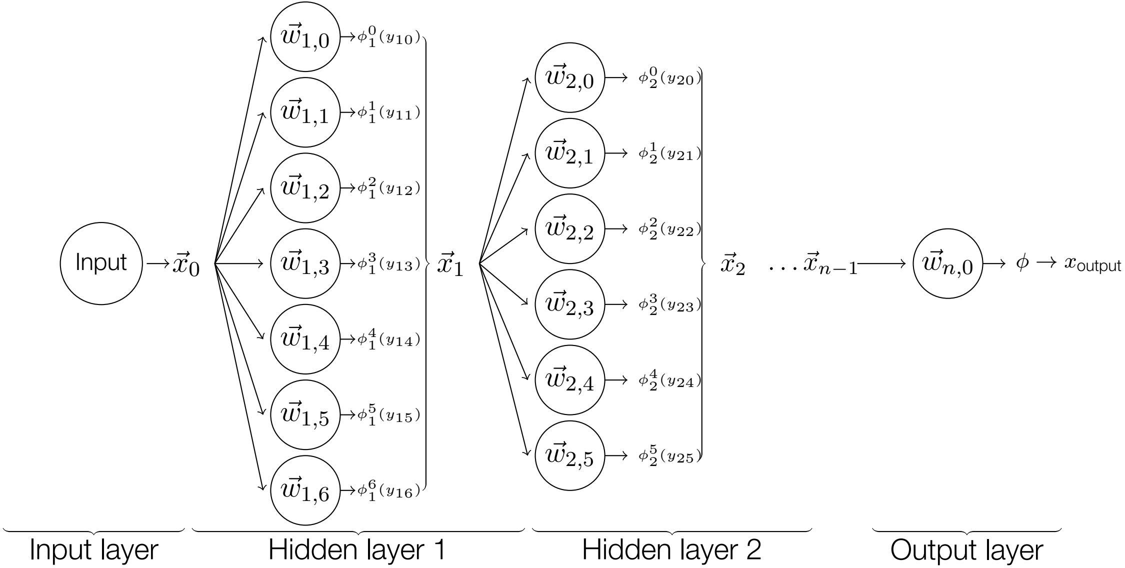
The Multilayered Perceptron



The Multilayered Perceptron



The Multilayered Perceptron



Training

Training

- ‘Training’ is about finding the set $\{\vec{w}_{ij}\}$ which produce the most accurate predictions

Training

- 'Training' is about finding the set $\{\vec{w}_{ij}\}$ which produce the most accurate predictions
- It is an *optimisation* problem

Training

- 'Training' is about finding the set $\{\vec{w}_{ij}\}$ which produce the most accurate predictions
- It is an *optimisation* problem

Naive optimisation function (L_2 cost function):

$$\mathcal{L}(\{\vec{w}_{ij}\}) = \sum_{q \text{ in dataset}} \left(\mathcal{P}(\vec{x}_q|\{\vec{w}_{ij}\}) - L_q\right)^2 \tag{6}$$

Training

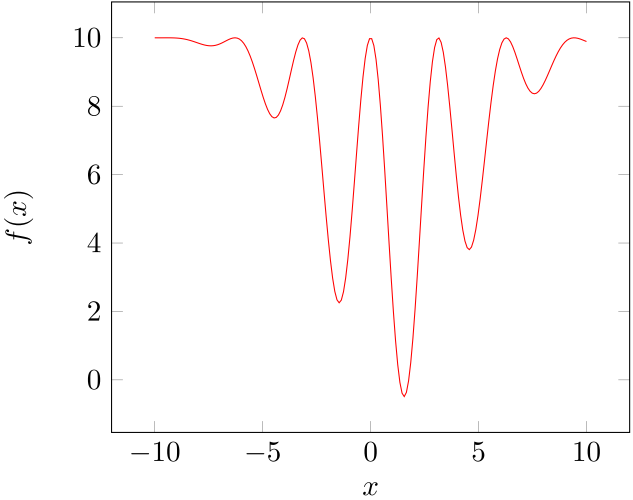
- 'Training' is about finding the set $\{\vec{w}_{ij}\}$ which produce the most accurate predictions
- It is an *optimisation* problem

Naive optimisation function (L_2 cost function):

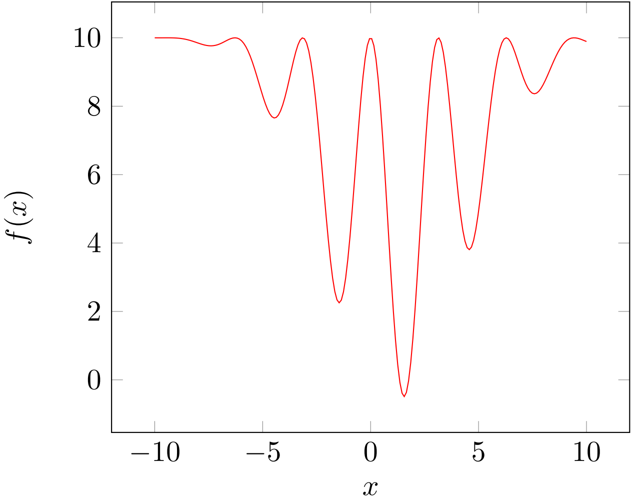
$$\mathcal{L}(\{\vec{w}_{ij}\}) = \sum_{q \text{ in dataset}} \left(\mathcal{P}(\vec{x}_q | \{\vec{w}_{ij}\}) - L_q\right)^2 \tag{6}$$

Big when $\mathcal{P} \neq L$, small when $\mathcal{P} = L$ $\rightarrow$ need to *minimise* this function

Optimisation



Optimisation



What value of x has lowest value of $f(x)$?

Optimisation

Optimisation

Easy in 1D:

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.

wellcome sanger institute

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.
- 400 years of support!

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.
- 400 years of support!

In higher dimensions....gets very tricky:

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.
- 400 years of support!

In higher dimensions....gets very tricky:

- First order derivative, $\nabla y(\vec{x})$, has n components

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.
- 400 years of support!

In higher dimensions....gets very tricky:

- First order derivative, $\nabla y(\vec{x})$, has n components
- Second order methods require inverting $H_{ij} = \frac{\partial^2 y}{\partial x_i \partial x_j}$

Optimisation

Easy in 1D:

- If $\frac{\mathrm{d}y}{\mathrm{d}x}$ computable, then $\frac{\mathrm{d}^2y}{\mathrm{d}x^2}$ etc.
- 400 years of support!

In higher dimensions....gets very tricky:

- First order derivative, $\nabla y(\vec{x})$, has n components
- Second order methods require inverting $H_{ij} = \frac{\partial^2 y}{\partial x_i \partial x_j}$
- Theoretical solutions not **practical**

wellcome sanger institute

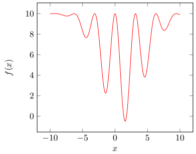
Optimisers

Optimisers

Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!

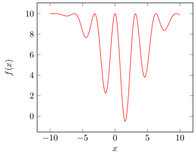
Optimisers

Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use 'dumb' first order methods:



Optimisers

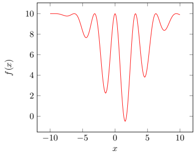
Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use 'dumb' first order methods:



1. Start at random network initialisation w_0

Optimisers

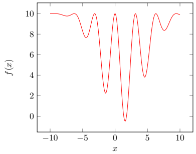
Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use 'dumb' first order methods:



1. Start at random network initialisation w_0
2. Find gradient $\nabla_w \mathcal{L}$ at current position

Optimisers

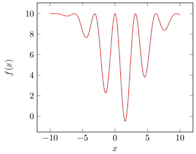
Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use ’dumb’ first order methods:



1. Start at random network initialisation w_0
2. Find gradient $\nabla_w \mathcal{L}$ at current position
3. Take step in (opposite) direction of gradient, update weights

Optimisers

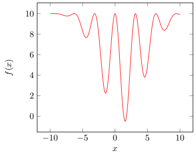
Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use 'dumb' first order methods:



1. Start at random network initialisation w_0
2. Find gradient $\nabla_w \mathcal{L}$ at current position
3. Take step in (opposite) direction of gradient, update weights
4. Go to 2)

Optimisers

Simple MLPs have $> 1000s$ parameters – squarely in the high dimensional space!
We therefore use 'dumb' first order methods:



1. Start at random network initialisation w_0
2. Find gradient $\nabla_w \mathcal{L}$ at current position
3. Take step in (opposite) direction of gradient, update weights
4. Go to 2)

Various modifications (adaptive learning, momentum etc.)

Derivatives of Networks

Derivatives of Networks

Fundamentally need ***derivatives***: in which direction does changing $\{\vec{w}\}$ improve $\mathcal{L}$?

$$\mathcal{L}(\{\vec{w}_{ij}\}) = \sum_{q \text{ in dataset}} \left(\mathcal{P}(\vec{x}_q|\{\vec{w}_{ij}\}) - L_q\right)^2 \tag{7}$$

Derivatives of Networks

Fundamentally need ***derivatives***: in which direction does changing $\{\vec{w}\}$ improve $\mathcal{L}$?

$$\mathcal{L}(\{\vec{w}_{ij}\}) = \sum_{q \text{ in dataset}} \left(\mathcal{P}(\vec{x}_q|\{\vec{w}_{ij}\}) - L_q\right)^2 \tag{7}$$

But $\mathcal{L}$ is a *complicated* function of $\{\vec{w}\}$!

Chain Rule: Crash Course

If y is a function of x, and f is a function of y, then:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{8}$$

Chain Rule: Crash Course

If y is a function of x, and f is a function of y, then:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{8}$$

The rate of change of f with x is the rate at which y changes with x, times the rate at which f changes with y

Chain Rule: Crash Course

If y is a function of x, and f is a function of y, then:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{8}$$

The rate of change of f with x is the rate at which y changes with x, times the rate at which f changes with y

In higher dimensions, have to sum over all possible combinations:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_i \frac{\partial f}{\partial y_i}\frac{\partial y_i}{\partial x} \tag{9}$$

Chain Rule: Crash Course

If y is a function of x, and f is a function of y, then:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{8}$$

The rate of change of f with x is the rate at which y changes with x, times the rate at which f changes with y

In higher dimensions, have to sum over all possible combinations:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_i \frac{\partial f}{\partial y_i}\frac{\partial y_i}{\partial x} \tag{9}$$

The rate of change of f with x is the sum of all of the rates at which y_i changes with x, times the rate at which f changes with y_i

wellcome sanger institute

Back to the optimisation problem

w_{ijk} is the k^{th} element of weight vector in the j^{th} node, in the i^{th} layer.

$$\frac{\partial \mathcal{L}}{\partial w_{ijk}} = \sum_{i \text{ in dataset}} \left(\mathcal{P}(\vec{x}_i | \{\vec{w}_{ij}\}) - L_i\right) \frac{\partial P}{\partial w_{ijk}} \tag{10}$$

Back to the optimisation problem

w_{ijk} is the k^{th} element of weight vector in the j^{th} node, in the i^{th} layer.

$$\frac{\partial \mathcal{L}}{\partial w_{ijk}} = \sum_{i \text{ in dataset}} \left(\mathcal{P}(\vec{x}_i | \{\vec{w}_{ij}\}) - L_i\right) \frac{\partial P}{\partial w_{ijk}} \tag{10}$$

Have two cases: i is the final layer, or it is not. If in the final layer, this is trivial:

$$\mathcal{P} = x_{\text{final}} = \phi(\underbrace{\vec{w}_f \cdot \vec{x}_{i-1}}_{y_{ij}})$$

$$\frac{\partial P}{\partial \vec{w}_{ij}} = \left.\frac{\partial \phi}{\partial y}\right|_{y=y_{ij}} \vec{x}_{i-1} \tag{11}$$

$$= \phi'(y_{ij})\vec{x}_{n-1}$$

Back to the optimisation problem

w_{ijk} is the k^{th} element of weight vector in the j^{th} node, in the i^{th} layer.

$$\frac{\partial \mathcal{L}}{\partial w_{ijk}} = \sum_{i \text{ in dataset}} \left(\mathcal{P}(\vec{x}_i | \{\vec{w}_{ij}\}) - L_i\right) \frac{\partial P}{\partial w_{ijk}} \tag{10}$$

Have two cases: i is the final layer, or it is not. If in the final layer, this is trivial:

$$\begin{aligned} \mathcal{P} = x_{\text{final}} &= \phi(\underbrace{\vec{w}_f \cdot \vec{x}_{i-1}}_{y_{ij}}) \\ \frac{\partial P}{\partial \vec{w}_{ij}} &= \left.\frac{\partial \phi}{\partial y}\right|_{y=y_{ij}} \vec{x}_{i-1} \\ &= \phi'(y_{ij})\vec{x}_{n-1} \end{aligned} \tag{11}$$

Therefore:

$$\frac{\partial \mathcal{L}}{\partial w_{\text{final}k}} = \sum_{i \text{ in dataset}} \left(\mathcal{P}(\vec{x}_i | \{\vec{w}_{ij}\}) - L_i\right) \phi'(y_{\text{final}}) \left[\vec{x}_{i-1}\right]_k \tag{12}$$

What about non-final layers?

Consider:

What about non-final layers?

Consider:

$$\vec{x}_i = \begin{pmatrix} \phi(\vec{x}_i \cdot \vec{w}_{i,0}) \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,1}) \\ \vdots \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,j}) \end{pmatrix} = \begin{pmatrix} \phi(y_{i,0} \\ \phi(y_{i,1}) \\ \vdots \\ \phi(y_{i,j})) \end{pmatrix} \tag{13}$$

By the chain rule:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ijk}} &= \frac{\partial \mathcal{L}}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial w_{ijk}} \\ &= \frac{\partial \mathcal{L}}{\partial y_{ij}} \left[\vec{x}_{i-1}\right]_k \end{aligned} \tag{14}$$

What about non-final layers?

Consider:

$$\vec{x}_i = \begin{pmatrix} \phi(\vec{x}_i \cdot \vec{w}_{i,0}) \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,1}) \\ \vdots \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,j}) \end{pmatrix} = \begin{pmatrix} \phi(y_{i,0} \\ \phi(y_{i,1}) \\ \vdots \\ \phi(y_{i,j})) \end{pmatrix} \tag{13}$$

What about non-final layers?

Consider:

$$\vec{x}_i = \begin{pmatrix} \phi(\vec{x}_i \cdot \vec{w}_{i,0}) \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,1}) \\ \vdots \\ \phi(\vec{x}_{i-1} \cdot \vec{w}_{i,j}) \end{pmatrix} = \begin{pmatrix} \phi(y_{i,0} \\ \phi(y_{i,1}) \\ \vdots \\ \phi(y_{i,j})) \end{pmatrix} \tag{13}$$

By the chain rule:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ijk}} &= \frac{\partial \mathcal{L}}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial w_{ijk}} \\ &= \frac{\partial \mathcal{L}}{\partial y_{ij}} \left[\vec{x}_{i-1}\right]_k \end{aligned} \tag{14}$$

wellcome sanger institute

Almost there....

Finally, we note that $\frac{\partial \mathcal{L}}{\partial y_{i+1,j}}$ is simply a statement about how changing y (the dot product) alters L, but that via the chain rule again:

$$\frac{\partial \mathcal{L}}{\partial y_{ij}} = \sum_{p \text{ nodes in next layer}} \frac{\partial \mathcal{L}}{\partial y_{i+1,p}} \frac{\partial y_{i+1,p}}{\partial y_{ij}} \tag{15}$$

Almost there....

Finally, we note that $\frac{\partial \mathcal{L}}{\partial y_{i+1,j}}$ is simply a statement about how changing y (the dot product) alters L, but that via the chain rule again:

$$\frac{\partial \mathcal{L}}{\partial y_{ij}} = \sum_{p \text{ nodes in next layer}} \frac{\partial \mathcal{L}}{\partial y_{i+1,p}} \frac{\partial y_{i+1,p}}{\partial y_{ij}} \tag{15}$$

The rate of change of L with respect to y_{ij} is equal to the sum of rates at which y_{ij} changes $y_{i+1,p}$ in the next layer, multiplied by how $y_{i+1,p}$ changes $\mathcal{L}$

wellcome sanger institute

Backpropagation

Backpropagation

How does this help?

Backpropagation

How does this help?
We go ***backwards*** through the network. Start at the final layer:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}}} = \phi'(y_{\text{final}}) \tag{16}$$

Backpropagation

How does this help?
We go *backwards* through the network. Start at the final layer:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}}} = \phi'(y_{\text{final}}) \tag{16}$$

Then feed this into the next layer down:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}-1,j}} = \phi'(y_{\text{final}-1}) \frac{\partial \mathcal{L}}{\partial y_{\text{final}}} w_{\text{final,j}} \tag{17}$$

Backpropagation

How does this help?
We go *backwards* through the network. Start at the final layer:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}}} = \phi'(y_{\text{final}}) \tag{16}$$

Then feed this into the next layer down:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}-1,j}} = \phi'(y_{\text{final}-1}) \frac{\partial \mathcal{L}}{\partial y_{\text{final}}} w_{\text{final,j}} \tag{17}$$

And so on...

Backpropagation

How does this help?
We go *backwards* through the network. Start at the final layer:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}}} = \phi'(y_{\text{final}}) \tag{16}$$

Then feed this into the next layer down:

$$\frac{\partial \mathcal{L}}{\partial y_{\text{final}-1,j}} = \phi'(y_{\text{final}-1}) \frac{\partial \mathcal{L}}{\partial y_{\text{final}}} w_{\text{final,j}} \tag{17}$$

And so on...
This is **backpropagation**: a fancy way of using the chain rule

And....you're done!

This is all you need

And....you're done!

This is all you need

Let’s see it in action!