RADONC AI CURRICULUM

Introduction to Deep Learning

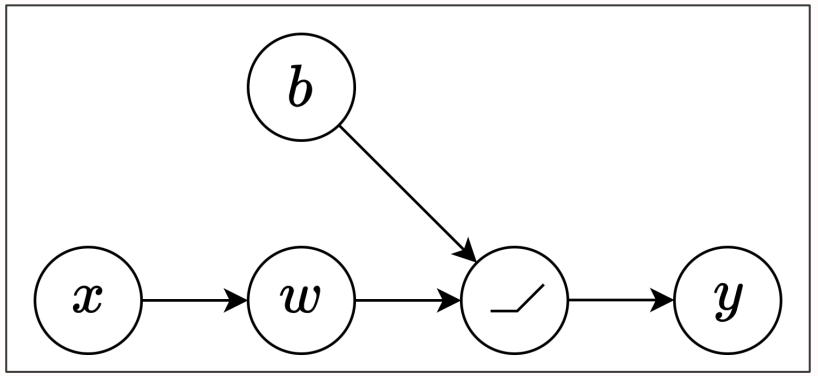
LECTURE 2: From Single Neurons to Neural Networks

ANDREW Y. K. FOONG, PH.D.

April 9th 2026

MAYO CLINIC

Radiation Oncology
AI & Data Analytics
AIDA



Today's lecture

1. The problem with linearity
   - *What are the limitations of linear regression?*
2. Non-linearities
   - *Going beyond linear regression*
3. Some mathematics
   - *Partial derivatives*
4. The single neuron
   - *What functions can it fit?*
5. Some philosophy
   - *Why do we need many neurons?*
6. Our first neural network
   - *Construction and terminology*
7. Gradient descent
   - *Learning with our first neural network*
8. Scaling up
   - *Adding even more neurons*
9. Q&A

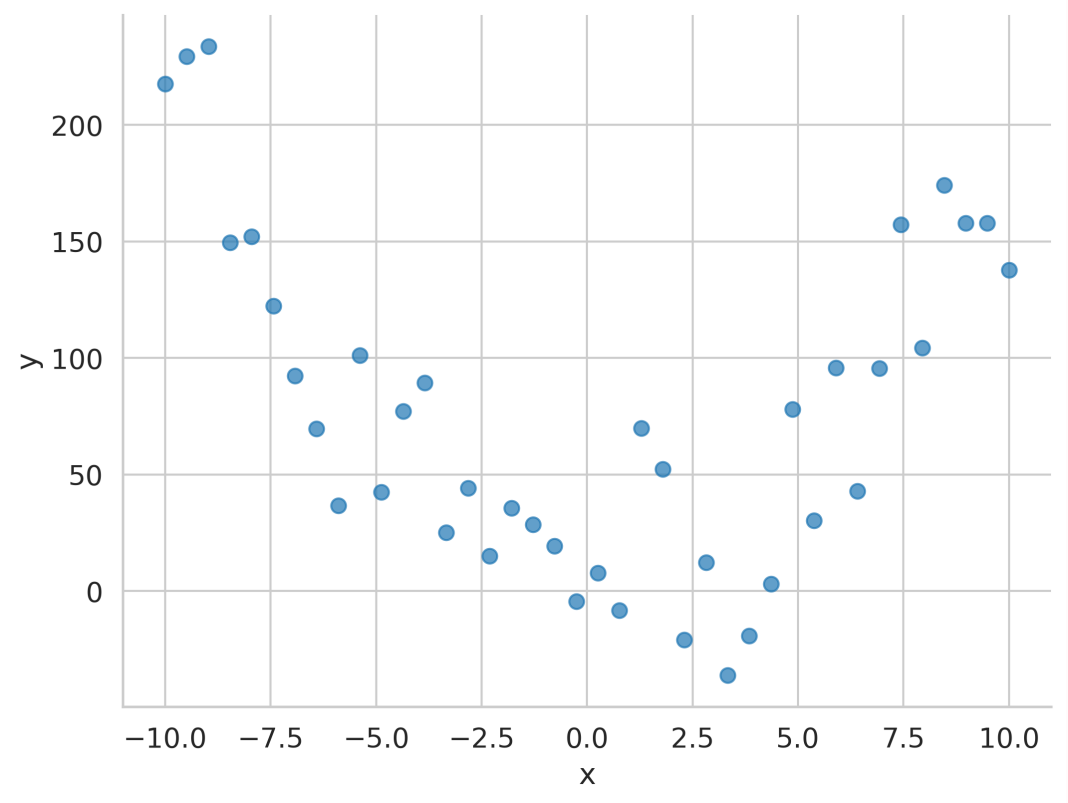
The Problem with Linearity

What are the limitations of linear regression?

The problem with linearity

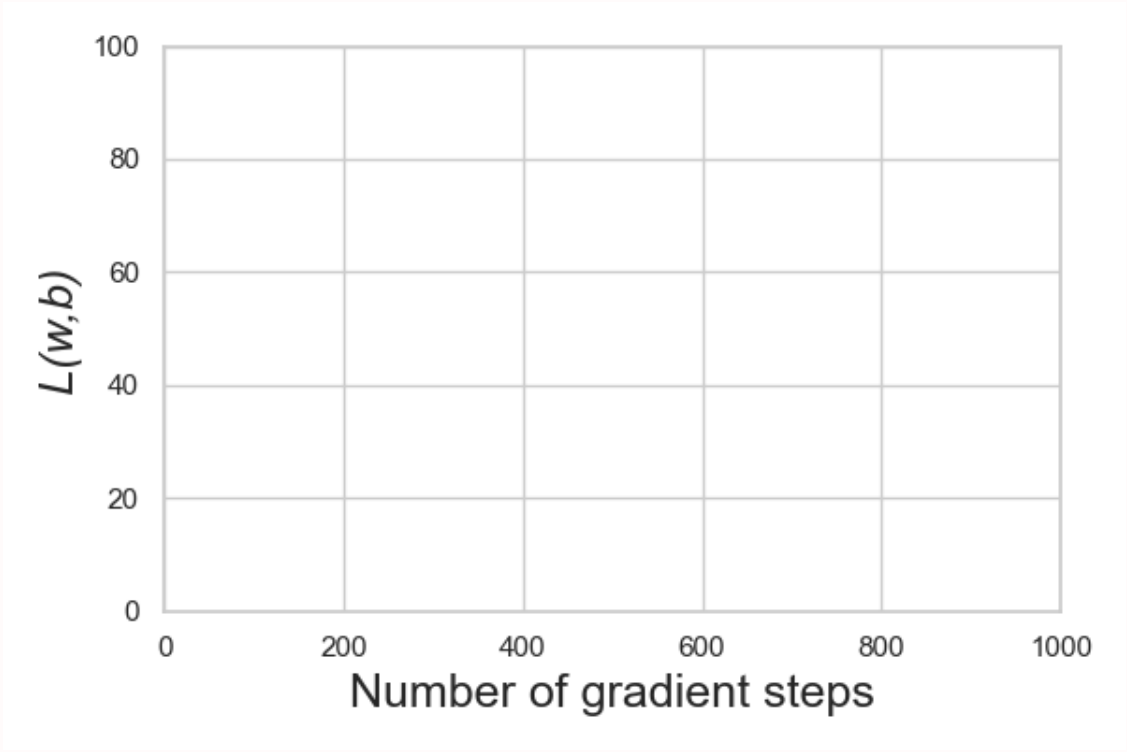
EXAMPLE:

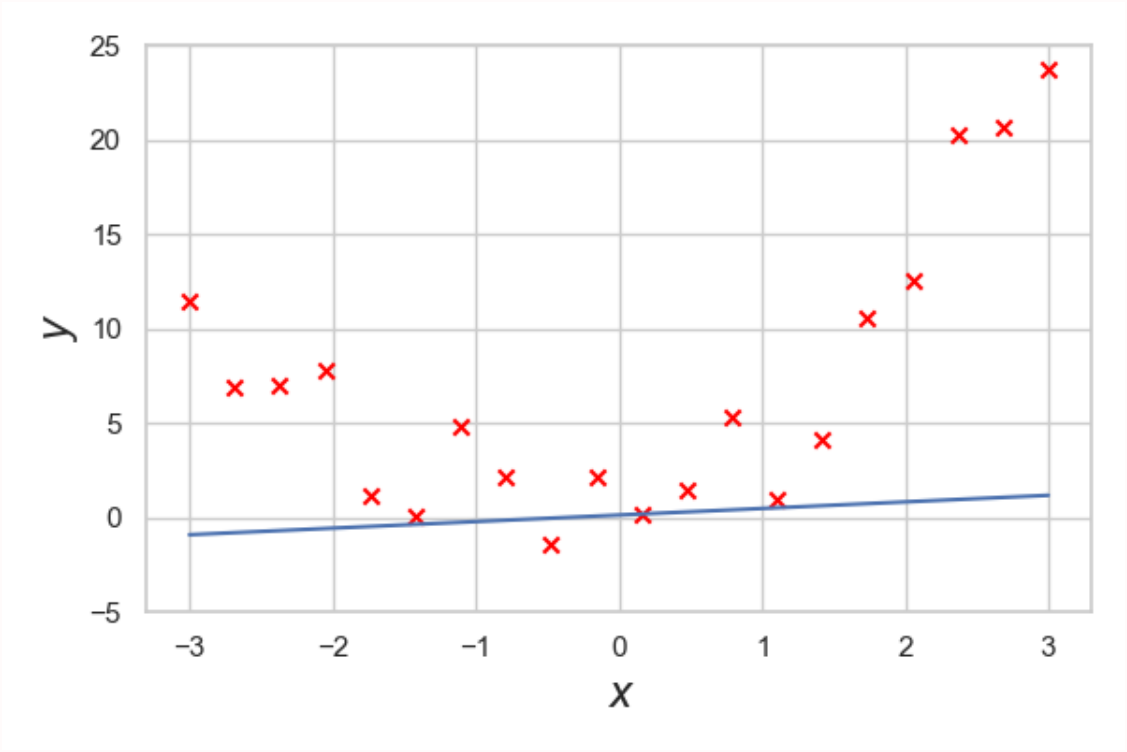
- Not every dataset can be fit will with a straight line!
- What happens if we apply linear regression anyway?
- Can run gradient descent again.
- *What will happen?*



The problem with linearity

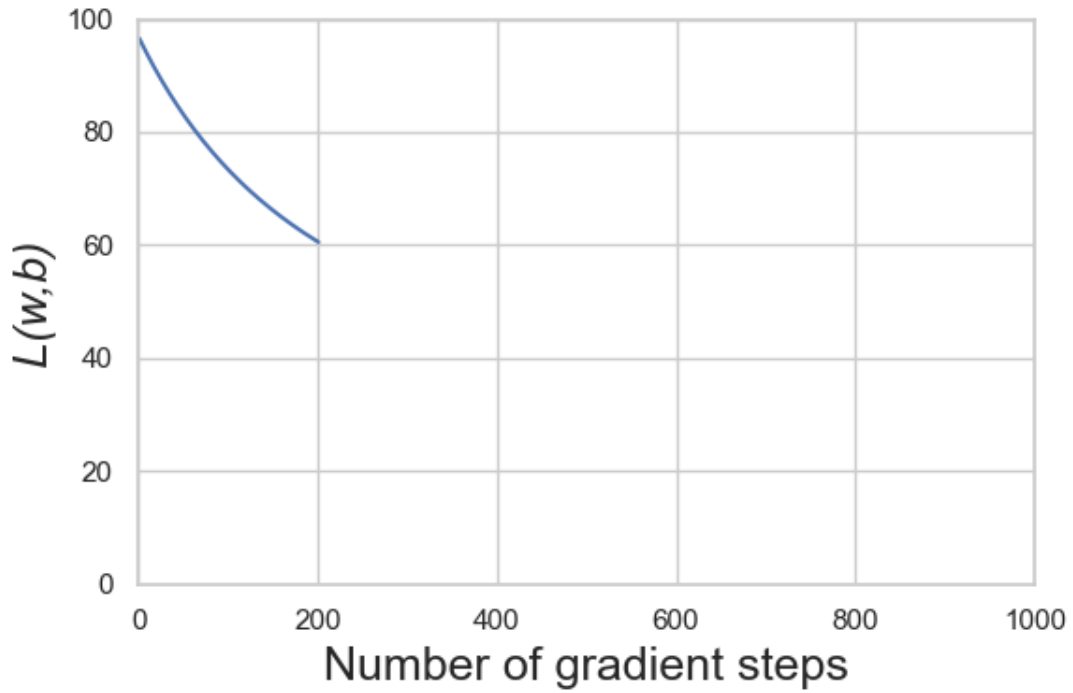
EXAMPLE:

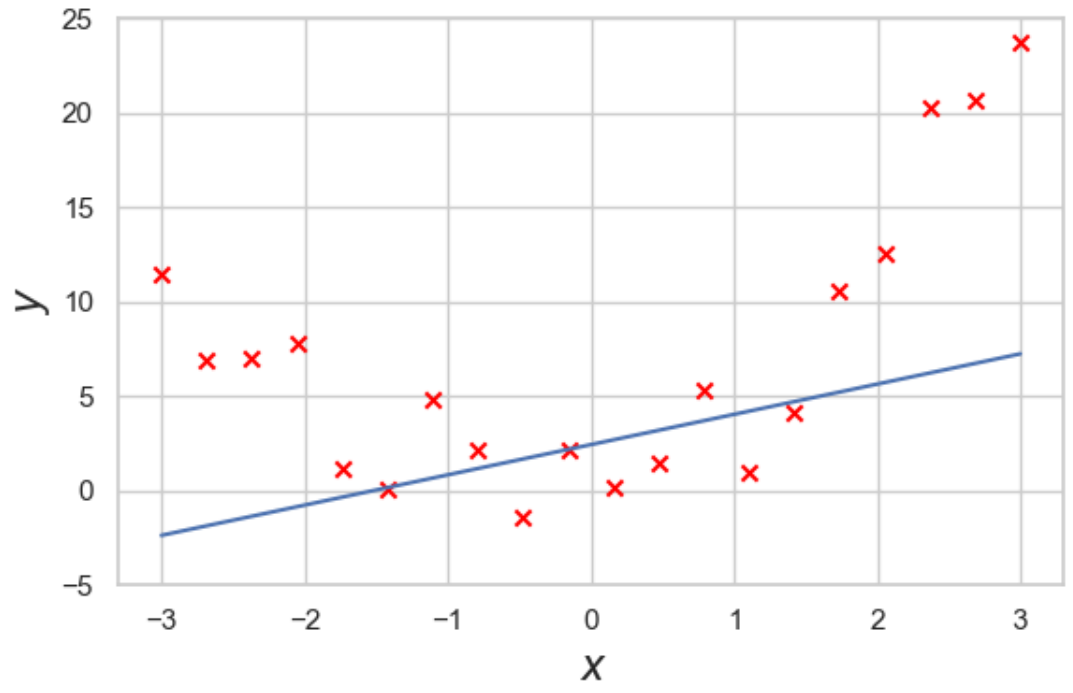




The problem with linearity

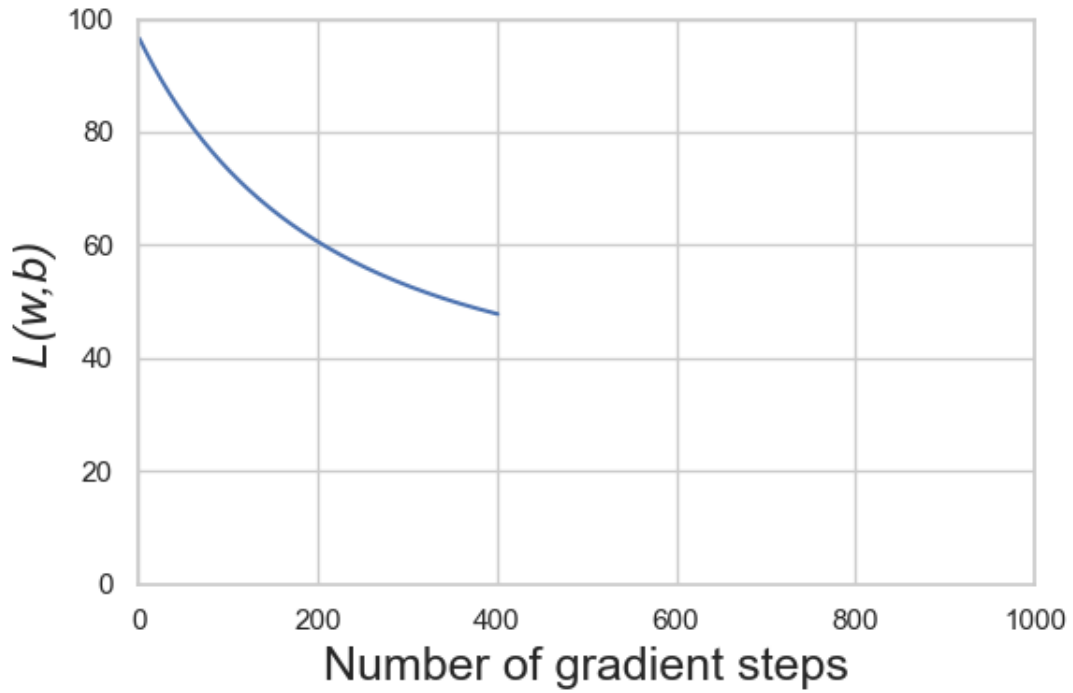
EXAMPLE:

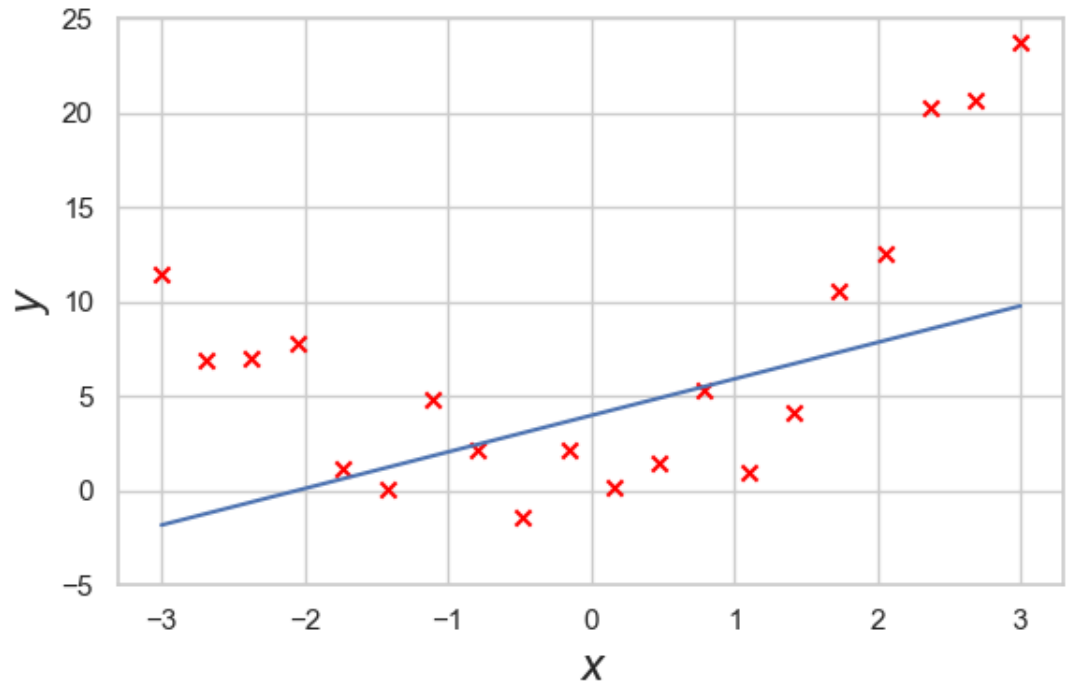




The problem with linearity

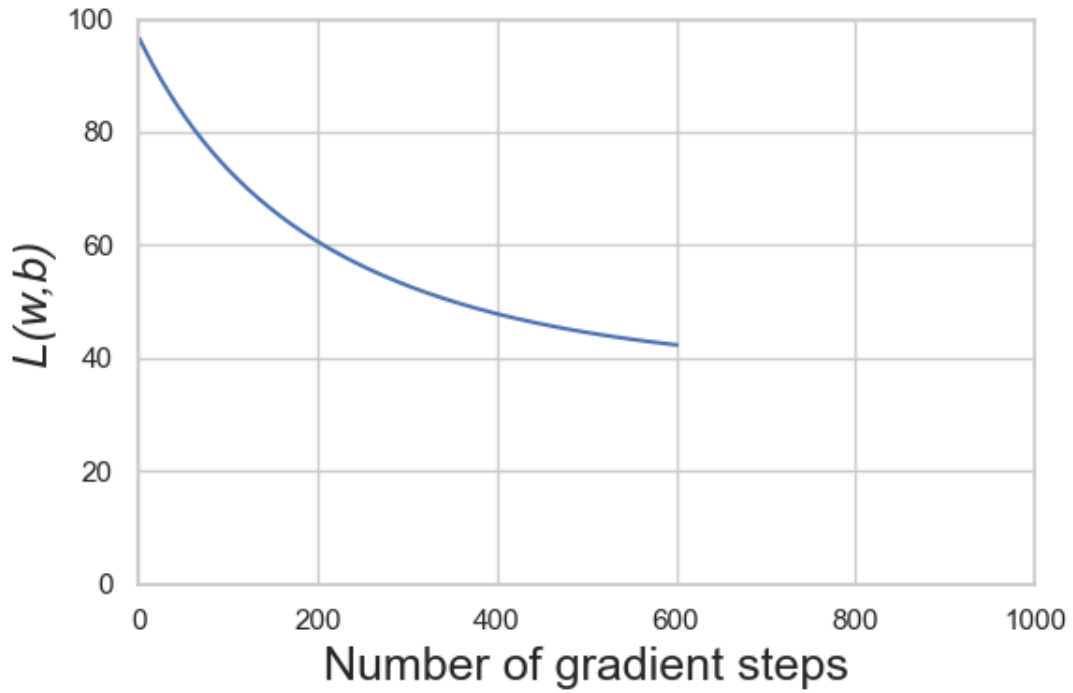
EXAMPLE:

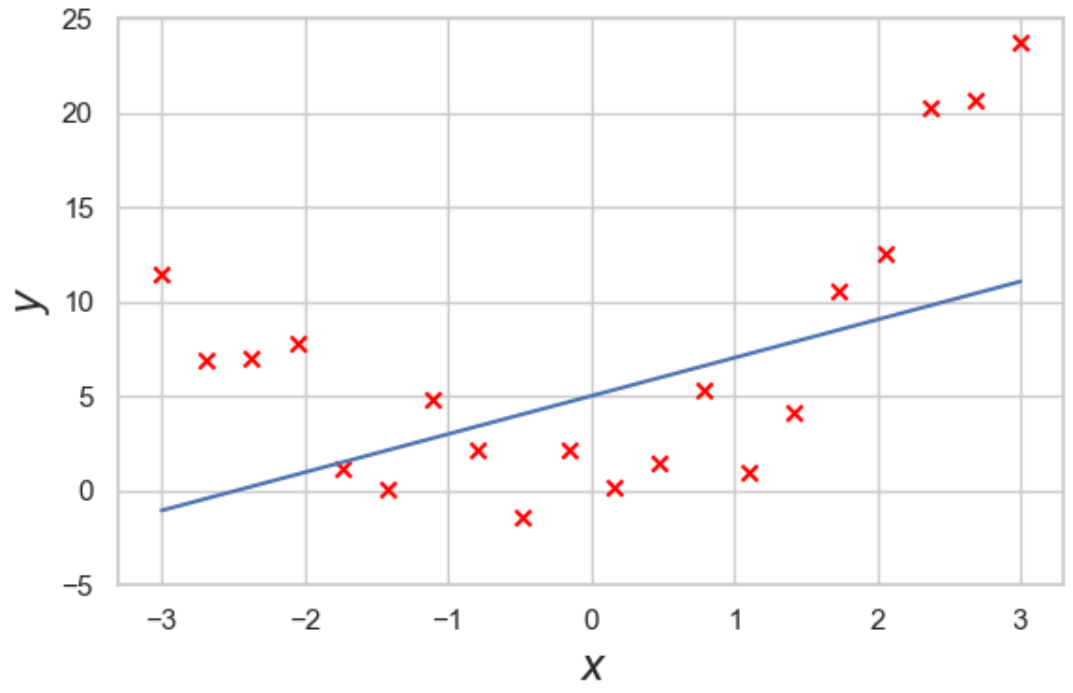




The problem with linearity

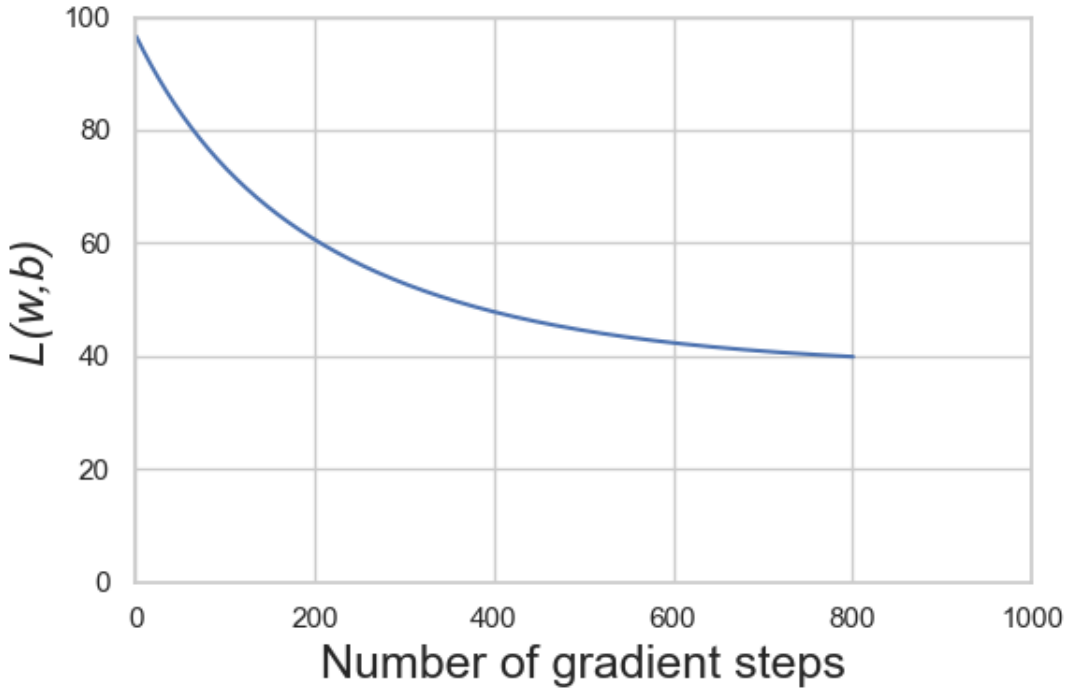
EXAMPLE:

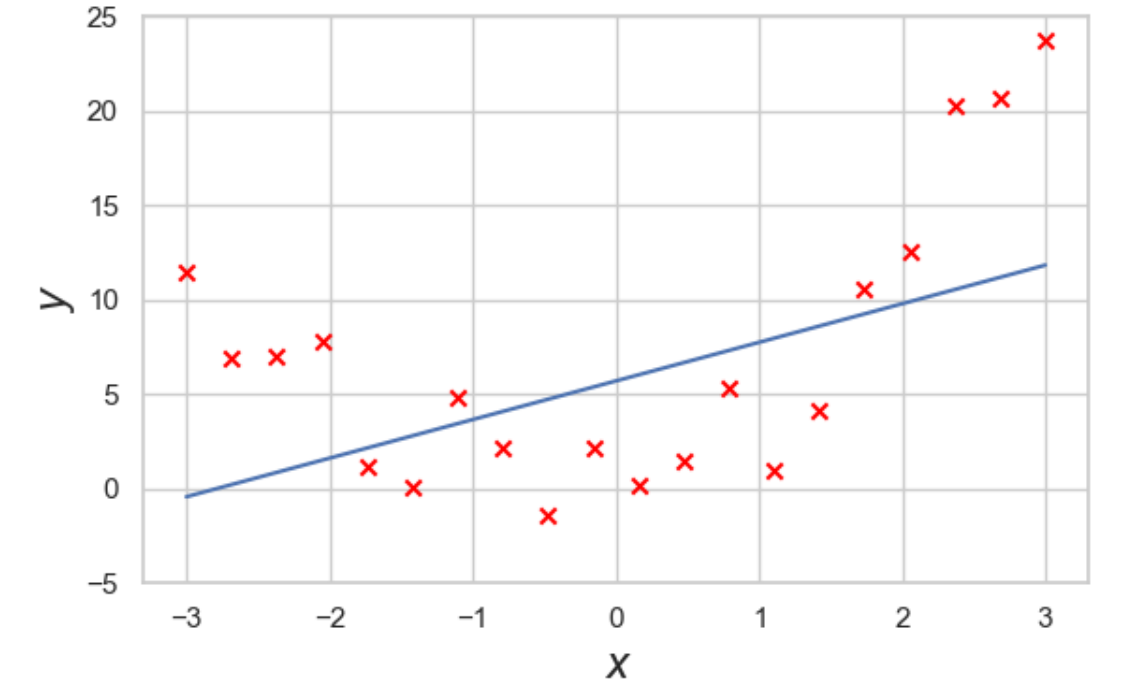




The problem with linearity

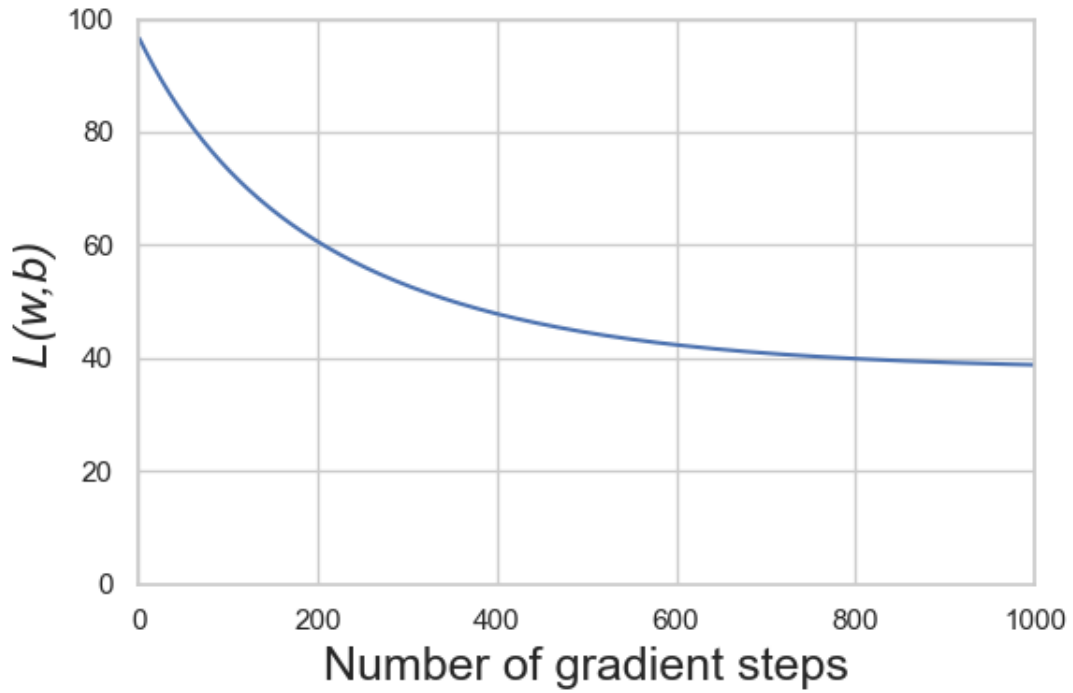
EXAMPLE:

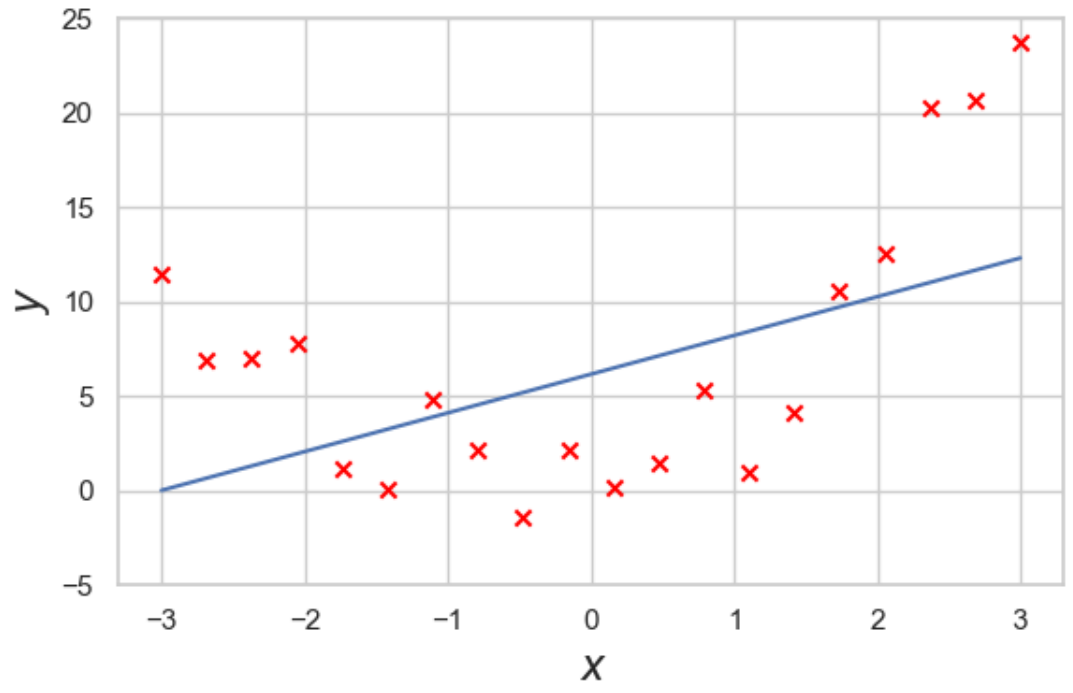




The problem with linearity

EXAMPLE:





- Loss remains high (≈40), fit still poor after optimization.
- *No way to fit this data well with straight line* $f(x) = wx + b$.
- Need to use a function with more flexibility!

Non-linearities

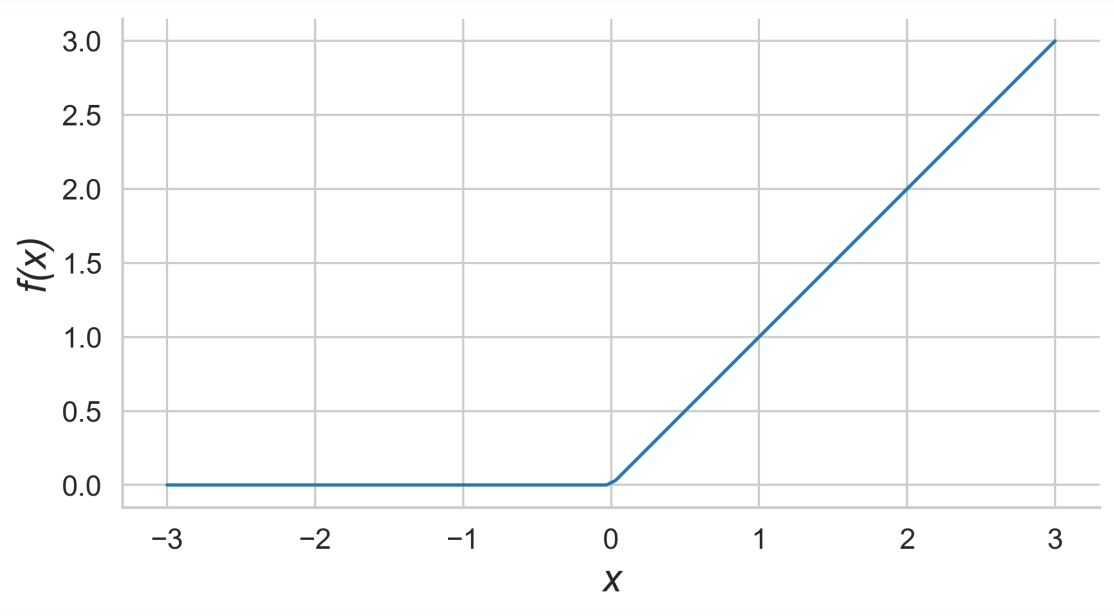
Going beyond linear regression

Non-linearities

- How can we move beyond straight lines?
- Choose a function that isn't a straight line!
- Rectified linear unit—ReLU:

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- ReLU is a non-linear function by construction.
- Can it fit our data now?
- *Not yet*: needs PARAMETERS that can be adjusted to fit data.

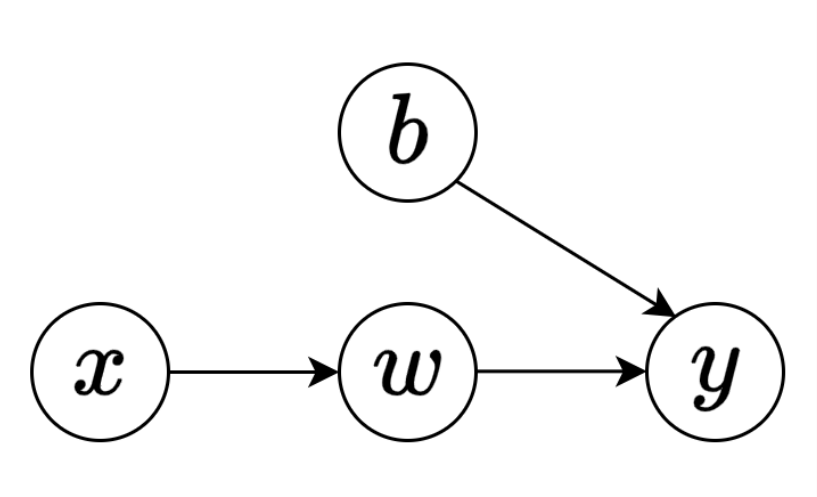


Rectified linear (ReLU) non-linearity

Architecture diagrams

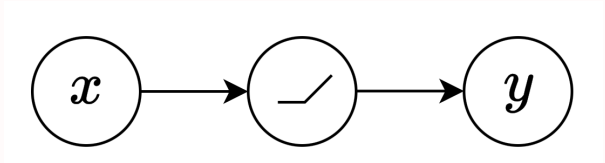
- How can we easily design parameterized functions?
- ARCHITECTURE DIAGRAMS: *Use diagrams to represent equations.*
- Can design complex neural networks.
- Represent $y = wx + b$ with DIAGRAM:
  1. *Circles* represent variables.
  2. *Arrows* represent flow of numbers.
     - Some represent *multiplication*.
     - Some represent *addition*.
  3. $x \longrightarrow w$:
     multiplication to form wx.
  4. $b \longrightarrow y$ and $wx \longrightarrow y$:
     addition to form $y = wx + b$.
- Nothing new so far—this is just linear regression.

EXAMPLE:

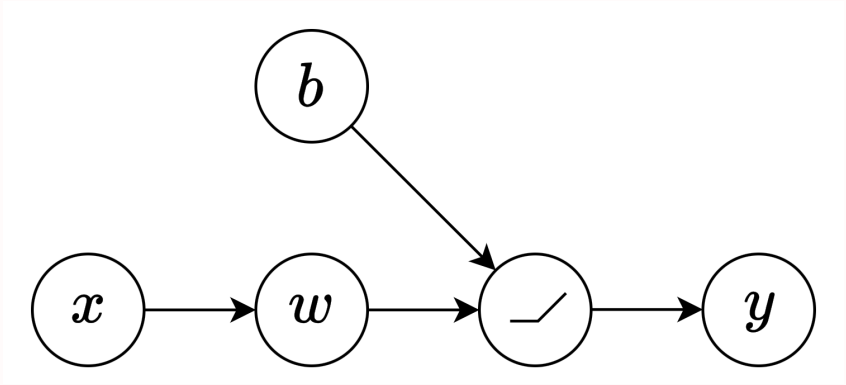


Architecture diagram for y = wx + b

ReLU with parameters



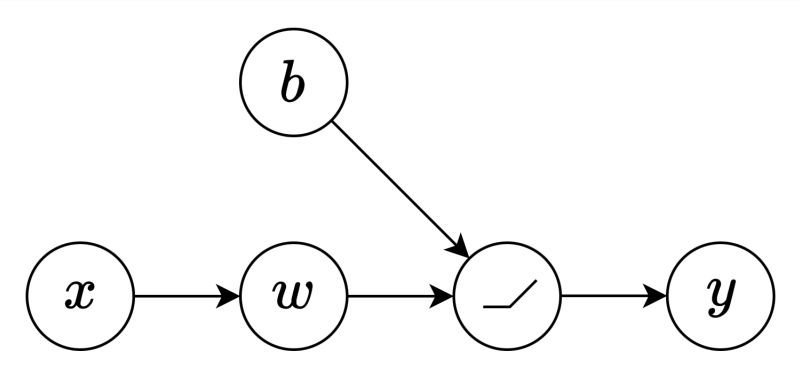
- NEW SYMBOL: Represents $y = \text{ReLU}(x)$.
- Now add parameters, similarly to linear regression:



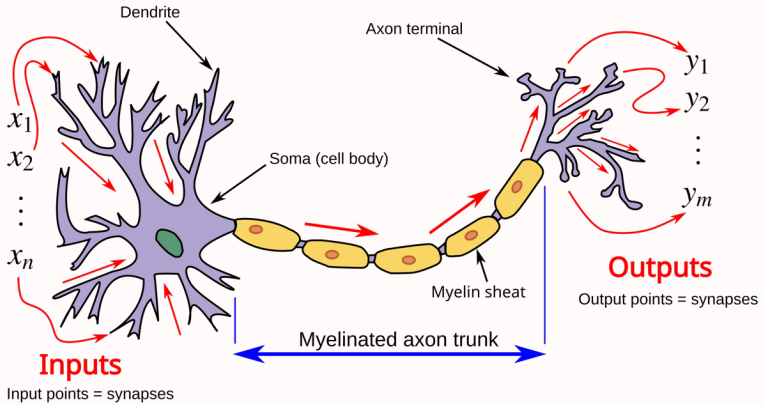
- Represents $y = \text{ReLU}(wx + b)$—scale and shift the input.
- Like linear regression that has been constrained to always be positive.
- AI scientists call this a *neuron*!

“Neurons”: a very loose analogy

- HISTORICAL ASIDE: Why do AI people call *this* a neuron?



≠

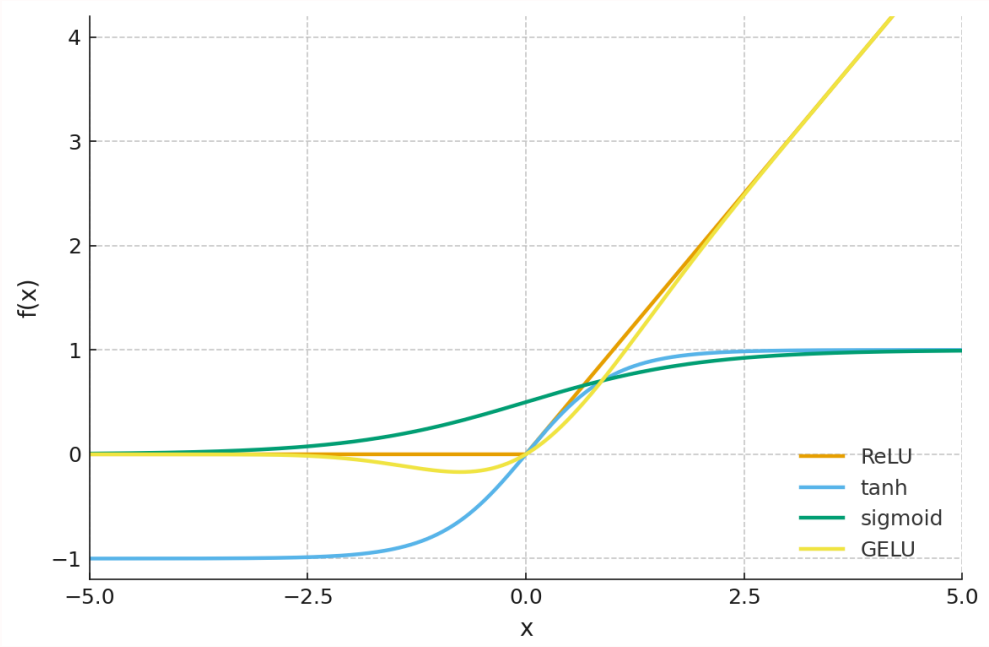


- Long history: MCCULLOCH AND PITTS (1943).
- Not accurate—but biological accuracy is not important for modern AI.
- Neuroscience inspiration can generally be ignored from an engineering perspective.
- Artificial neurons are the building block for *all* modern neural networks.

McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259

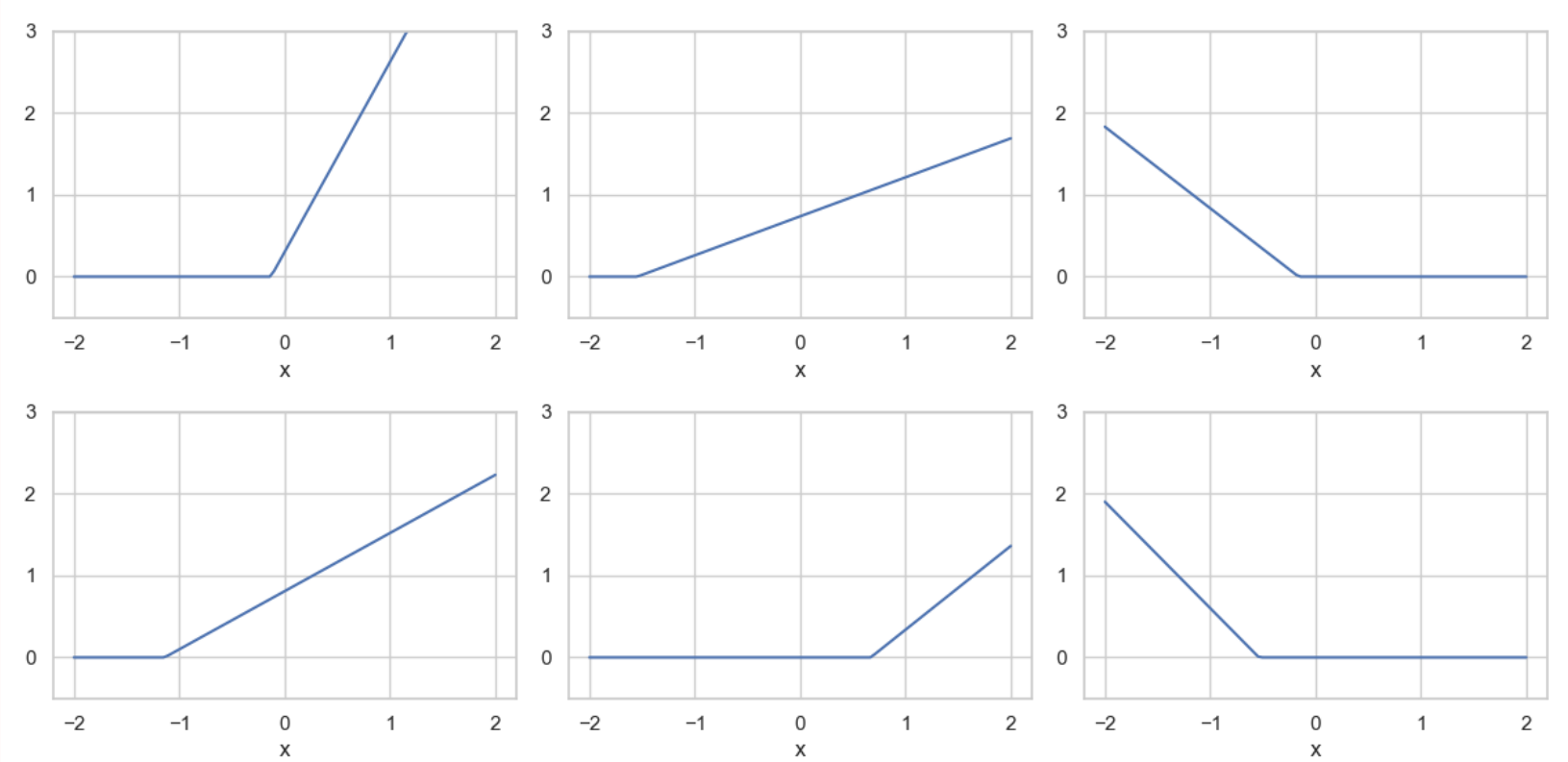
Other non-linearities

- Many ways to make a function non-linear.
- Why use ReLU?
  - ReLU is just one choice of non-linearity. Other alternatives:
    - *Hyperbolic tangent*:
      $$f(x) = \tanh(x)$$
    - *Sigmoid*:
      $$f(x) = 1 / (1 + \exp(-x))$$
    - *Gaussian Error Unit* (GELU):
      $$f(x) = 0.5x\left(1 + \tanh\left(\sqrt{(2/\pi)}(x + 0.044715x^3)\right)\right)$$
  - Best-performing non-linearities are discovered by trial-and-error or mathematical/theoretical motivation.
- ReLU is the simplest, and it usually works.



Single neuron functions

- What kinds of functions can a *single* ReLU neuron, $y = \text{ReLU}(wx + b)$ represent?
- Choose w, b at *random* a few times to get an idea:



Some Mathematics

Partial derivatives

Gradient descent with a ReLU neuron

- **PROCEDURE:**
  1. Choose w, b *randomly* (terrible loss!).
  2. Calculate derivative/gradient of $L(w, b)$—easy on a computer.
  3. Adjust w, b by small amount in (opposite) direction of gradient.
  4. Repeat 2–3 until $L(w, b)$ is low (good fit!).
- How do you compute the derivative?
- **PREVIOUSLY:**

$$L(w, b) = \sum_{n=1}^{N} (f(x_n) - y_n)^2$$
$$= \sum_{n=1}^{N} (wx_n + b - y_n)^2$$

- **NOW:**

$$L(w, b) = \sum_{n=1}^{N} (f(x_n) - y_n)^2$$
$$= \sum_{n=1}^{N} (\mathrm{ReLU}(wx_n + b) - y_n)^2$$

Gradients recap

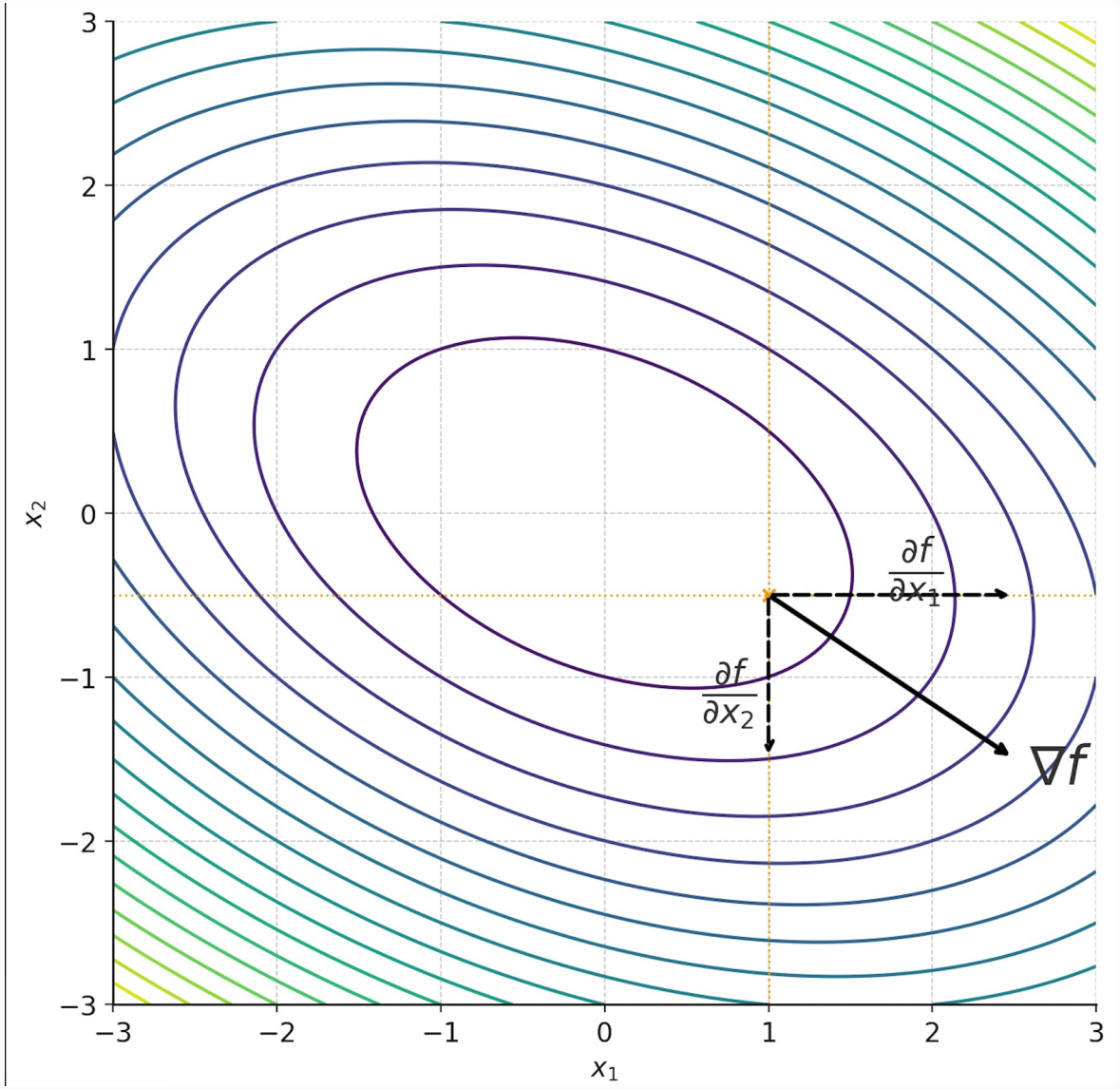
- A function $f(x)$ has a gradient-function called $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$.
- The gradient $\frac{\mathrm{d}f}{\mathrm{d}x}$ is a *function*, just like f.
- That means $\frac{\mathrm{d}f}{\mathrm{d}x}$ takes an input x and transforms it into an output $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$.
- This function has a special interpretation:

When given input x, the function $\frac{\mathrm{d}f}{\mathrm{d}x}$ outputs the value of the *slope* of the function f at the location x.

Partial derivatives

- When a function has *one* input, the gradient is easy to understand:
  - A function $f(x)$ has a gradient-function called $\frac{df}{dx}(x)$.
- What if it has *two* inputs?
- Then there are *two directions* to the gradient:
  - A function $f(x_1, x_2)$ has a gradient-function with two components: $\left[\frac{\partial f}{\partial x_1}(x_1, x_2), \frac{\partial f}{\partial x_2}(x_1, x_2)\right]$
  - This is a *vector* with two elements.
  - The *first* element, $\frac{\partial f}{\partial x_1}(x_1, x_2)$ is the PARTIAL DERIVATIVE of f at the *location* (x_1, x_2), *with respect to* x_1.
  - The *second* element, $\frac{\partial f}{\partial x_2}(x_1, x_2)$ is the PARTIAL DERIVATIVE of f at the *location* (x_1, x_2), *with respect to* x_2.

Partial derivatives



Partial derivatives

- A function $f(x_1, x_2)$ has a gradient-function called:
  $$\nabla f = \left[\frac{\partial f}{\partial x_1}(x_1, x_2), \frac{\partial f}{\partial x_2}(x_1, x_2)\right], \text{ pronounced "}\textit{grad}\ f\text{"}.$$
- The gradient ∇f is a *function*, just like f.
- That means ∇f takes an input (x_1, x_2) and transforms it into an output $\nabla f(x_1, x_2)$.
- This function has a special interpretation:

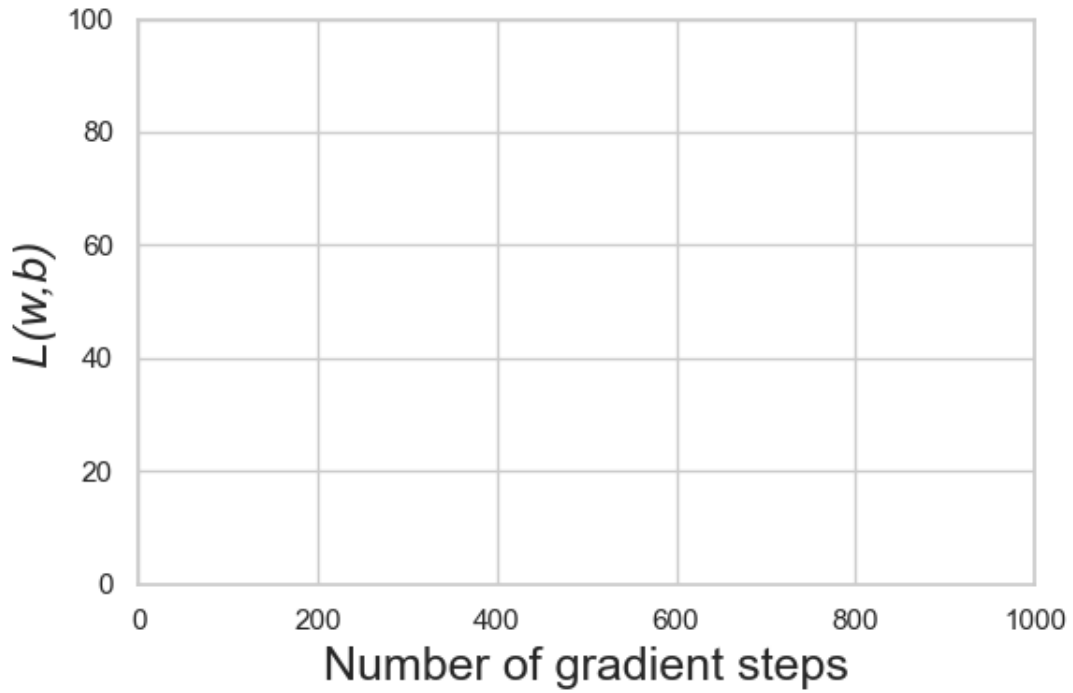
When given input (x_1, x_2), the function ∇f outputs a *vector*. Each component of the vector—each *partial derivative*—is the *slope* of the function f in the direction of x_1 and x_2 respectively.

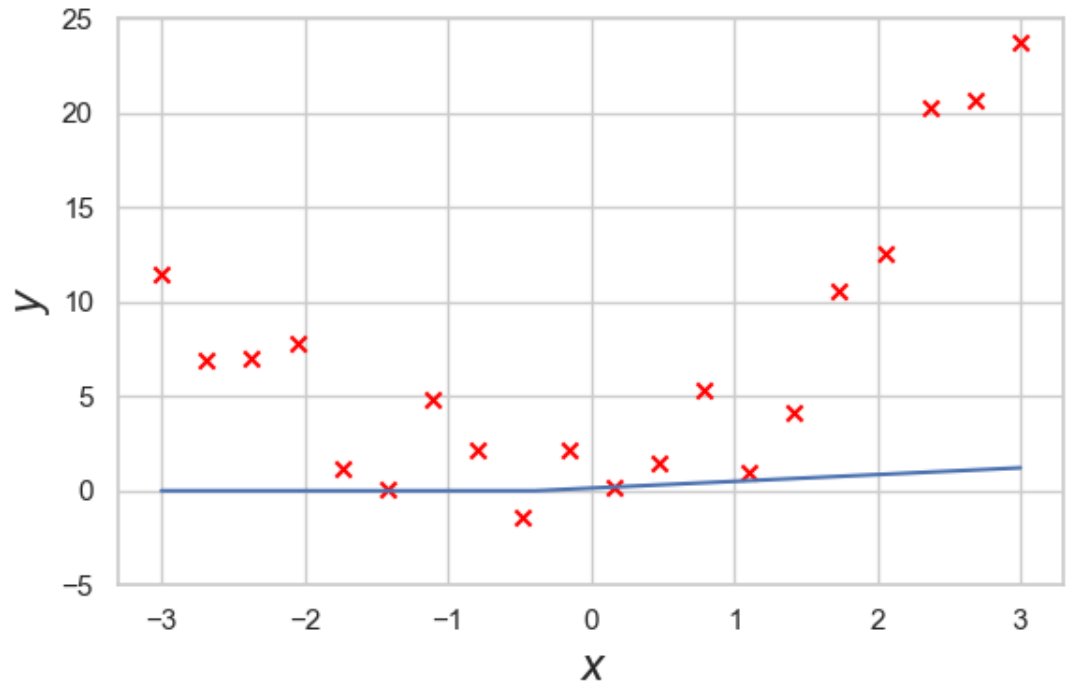
The Single Neuron

What functions can it fit?

Fitting data with one ReLU

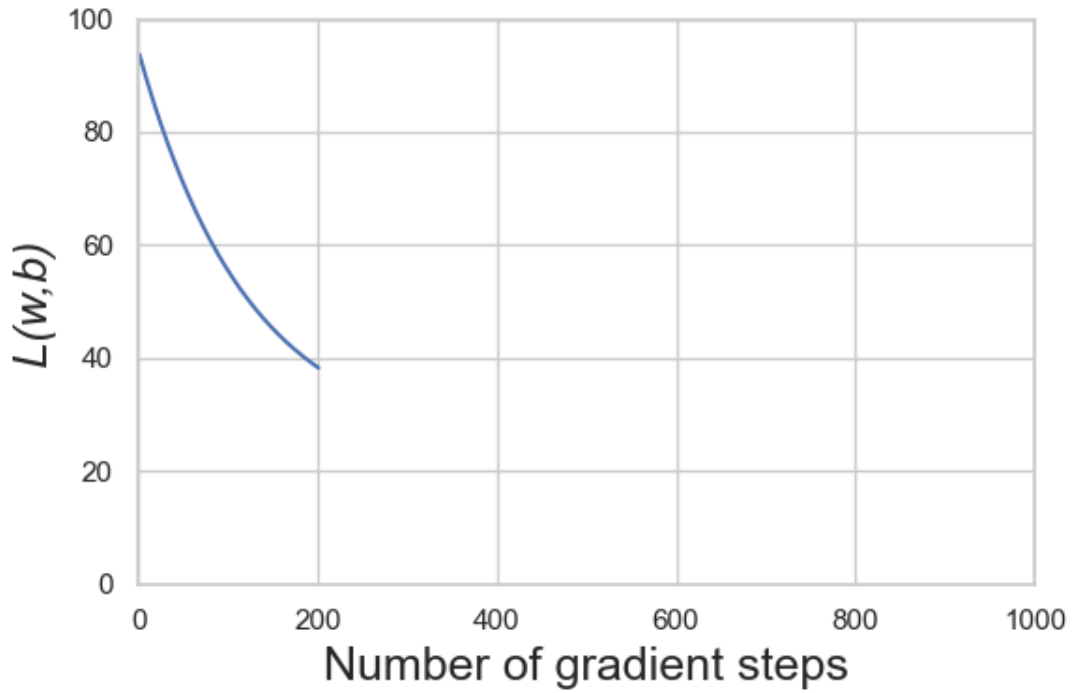
EXAMPLE:

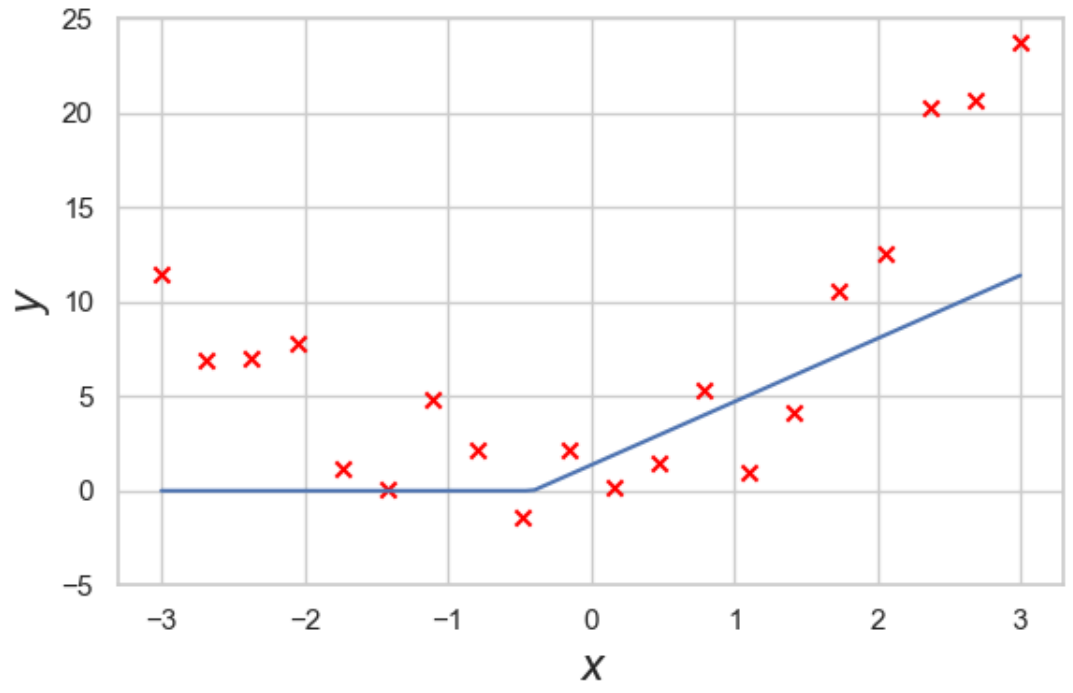




Fitting data with one ReLU

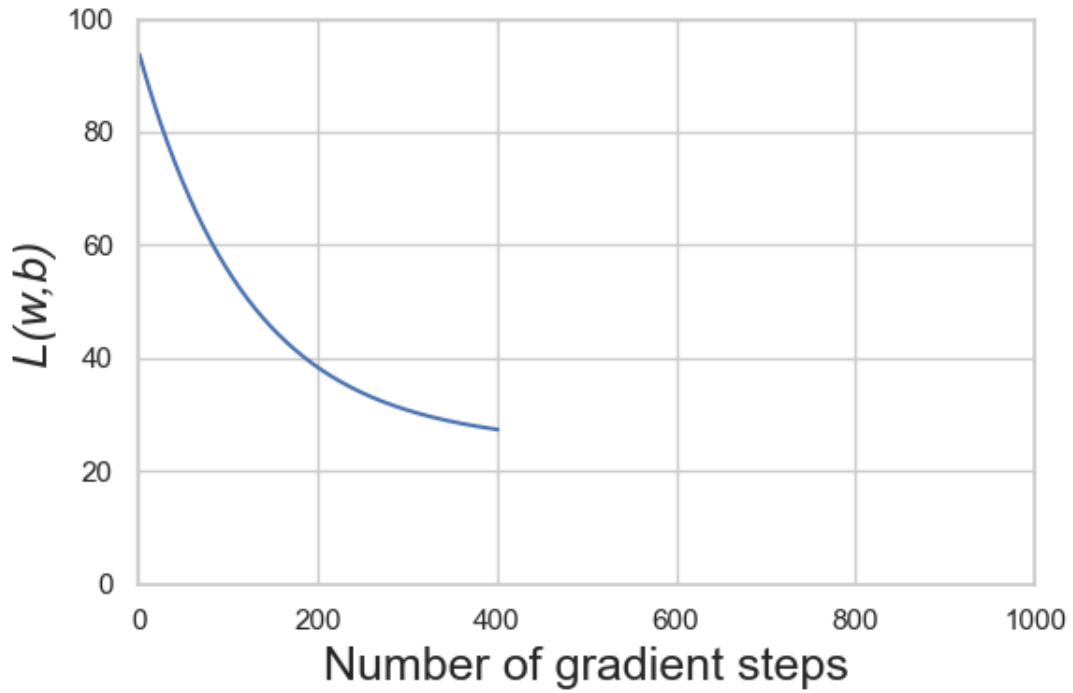
EXAMPLE:

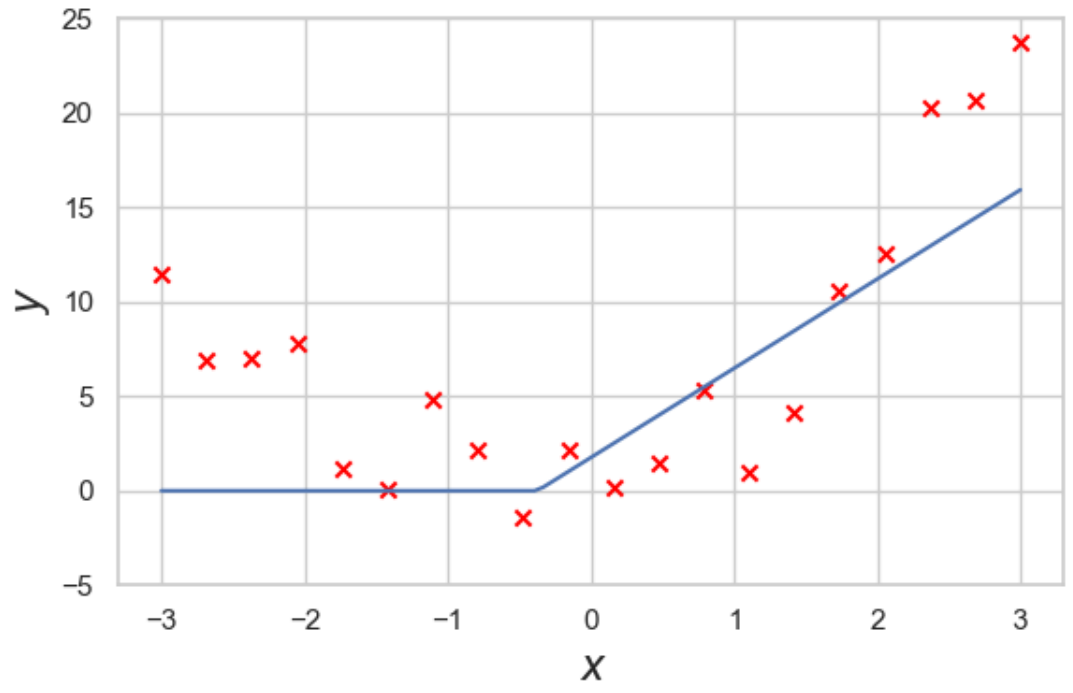




Fitting data with one ReLU

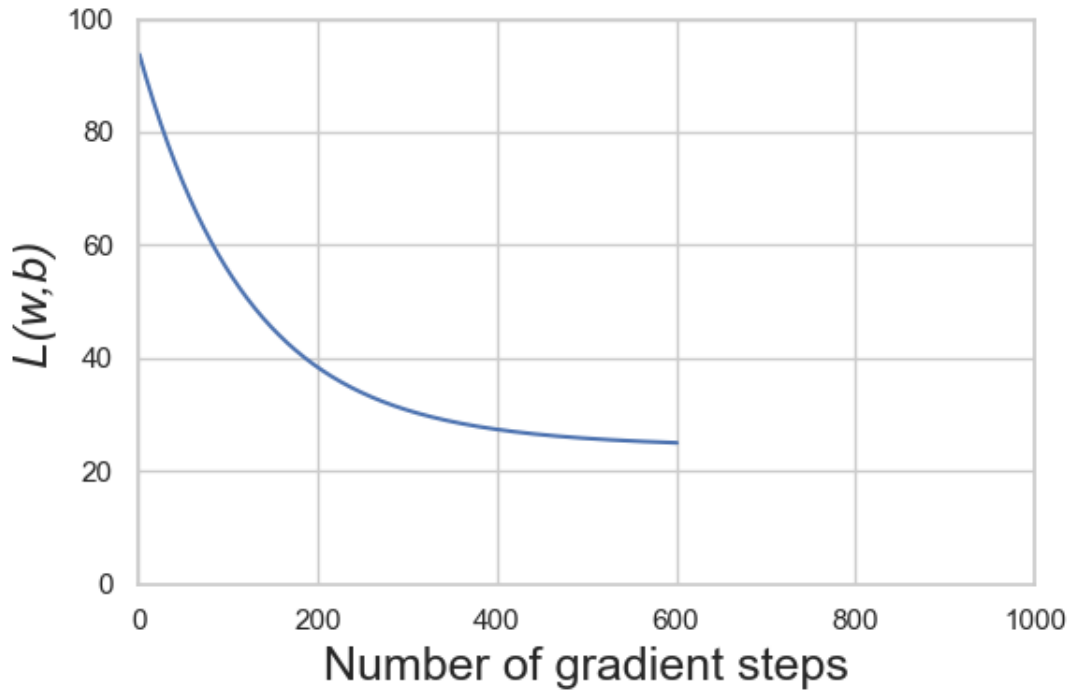
EXAMPLE:

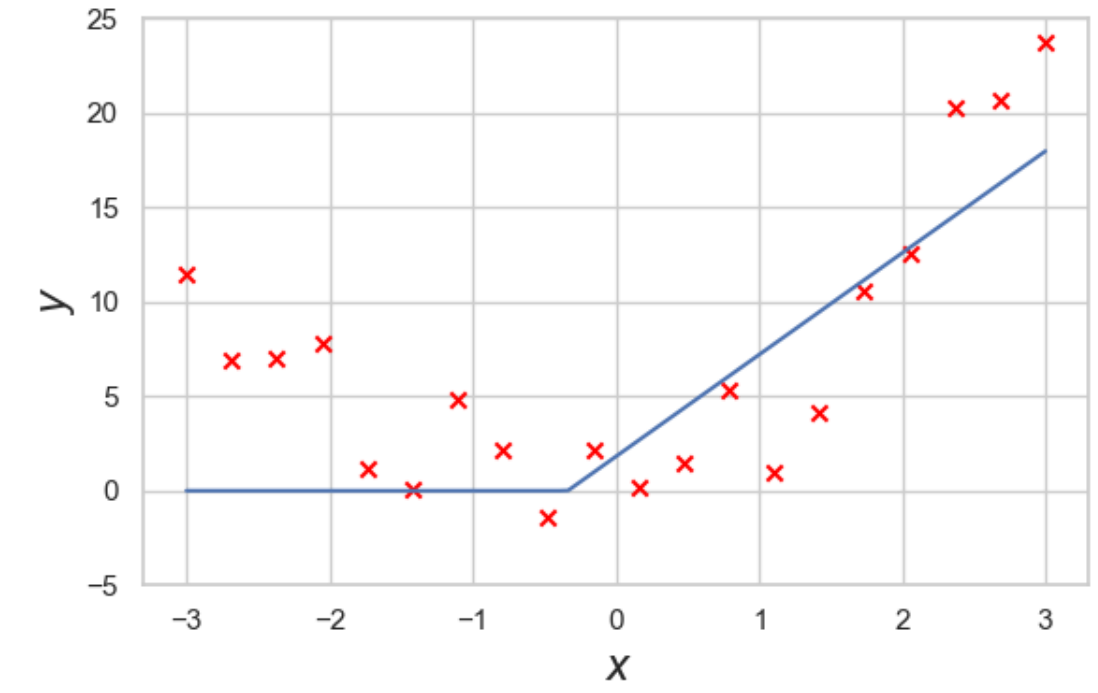




Fitting data with one ReLU

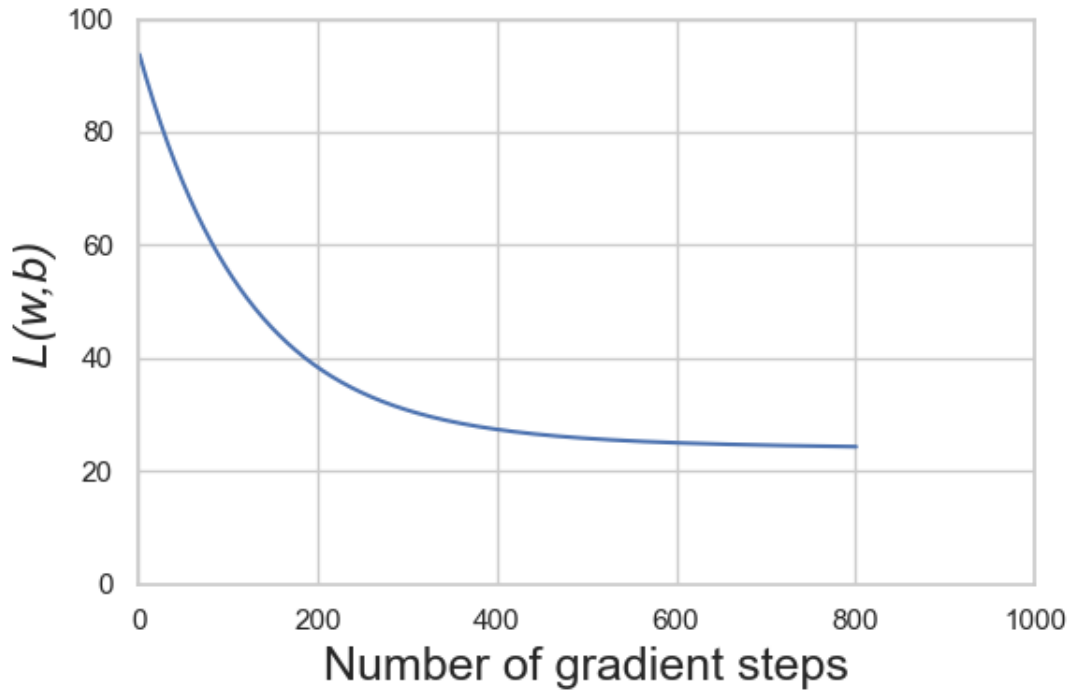
EXAMPLE:

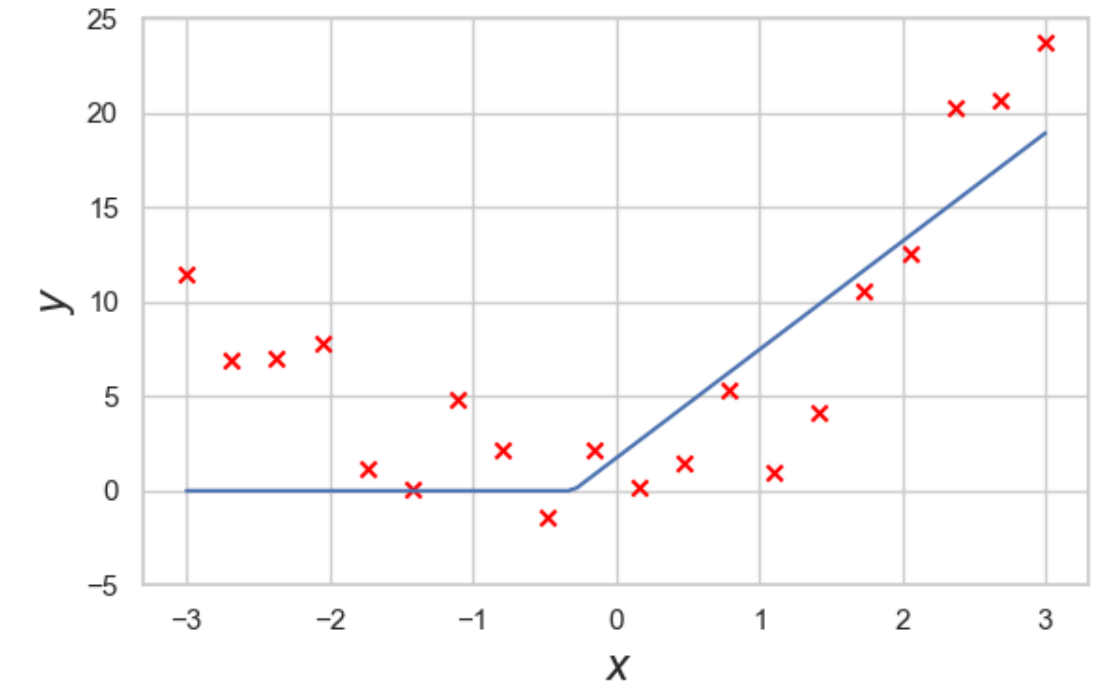




Fitting data with one ReLU

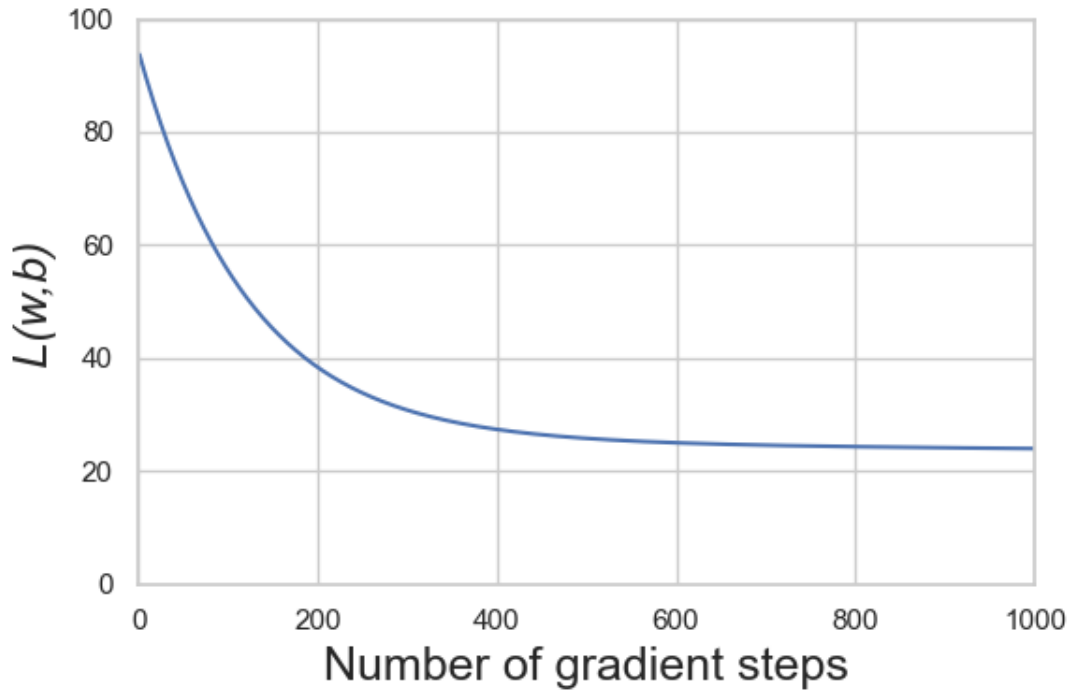
EXAMPLE:

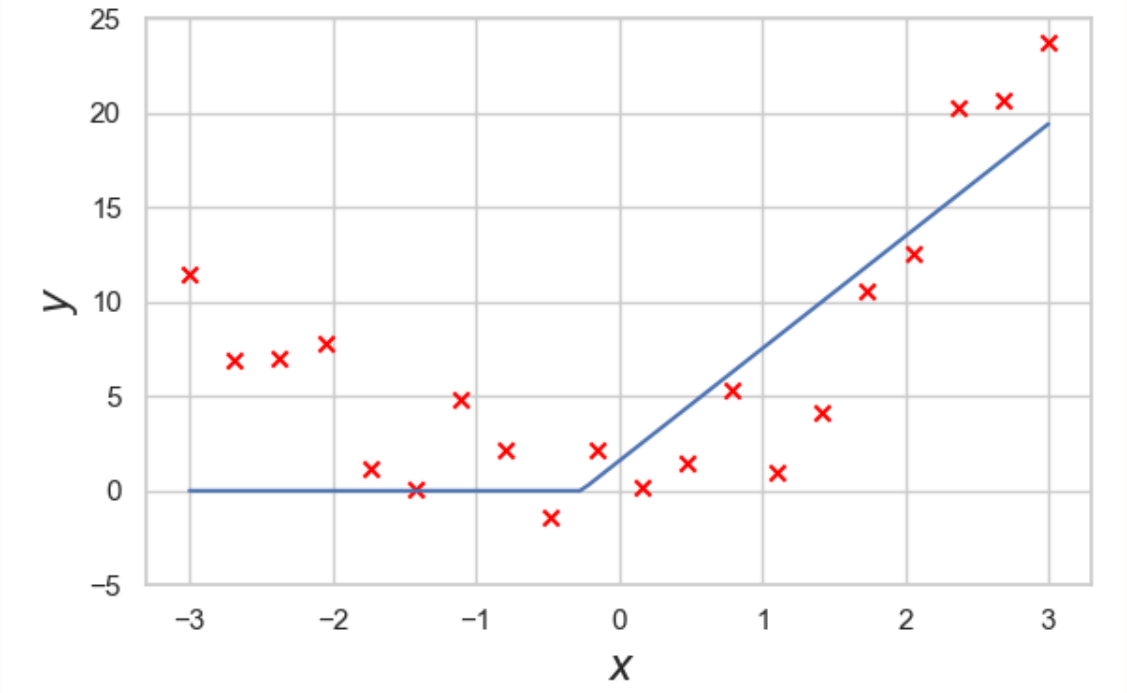




Fitting data with one ReLU

EXAMPLE:



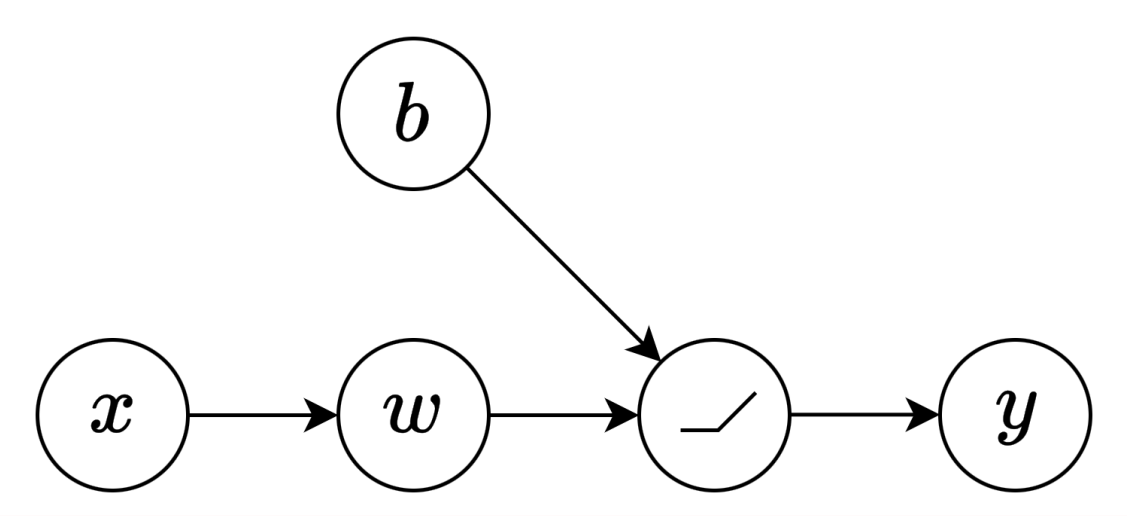


- Better fit than linear regression (loss ≈25 vs ≈40).
- Curve still misses many points. Need *even more* flexibility!

Single neuron functions

- Scale and shift output as well as input:

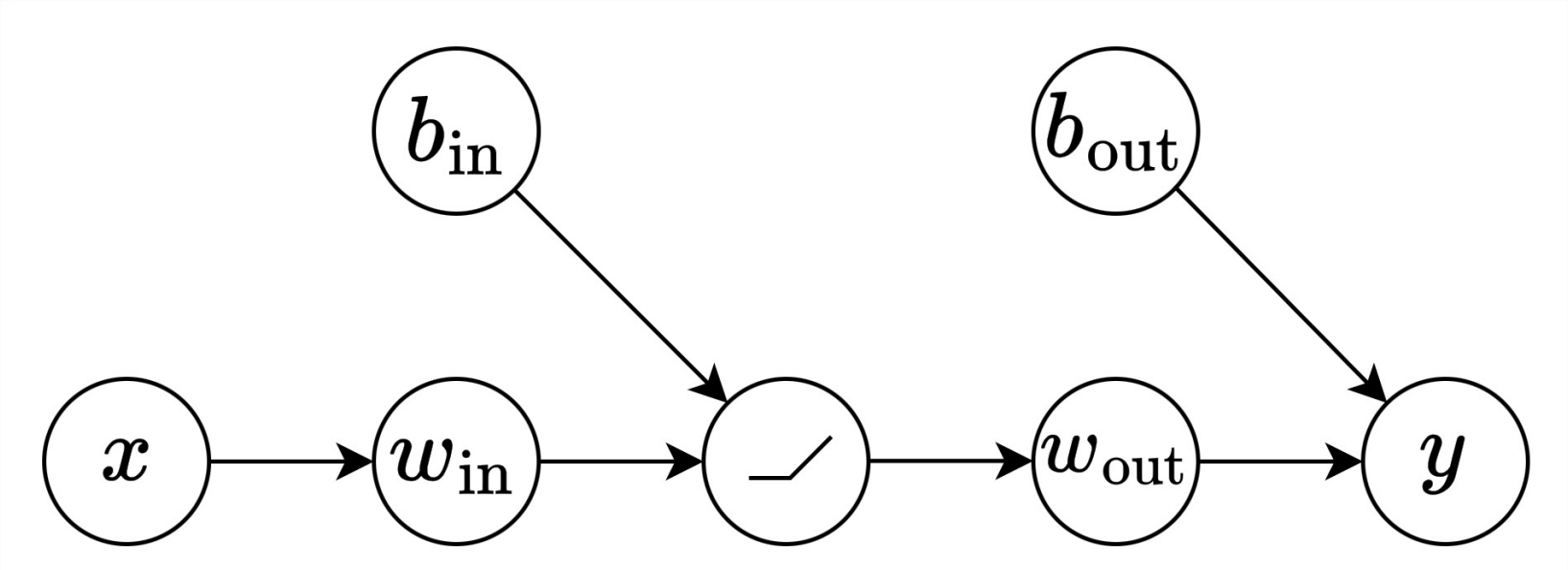
$$y = \mathrm{ReLU}(wx + b)$$



Single neuron functions

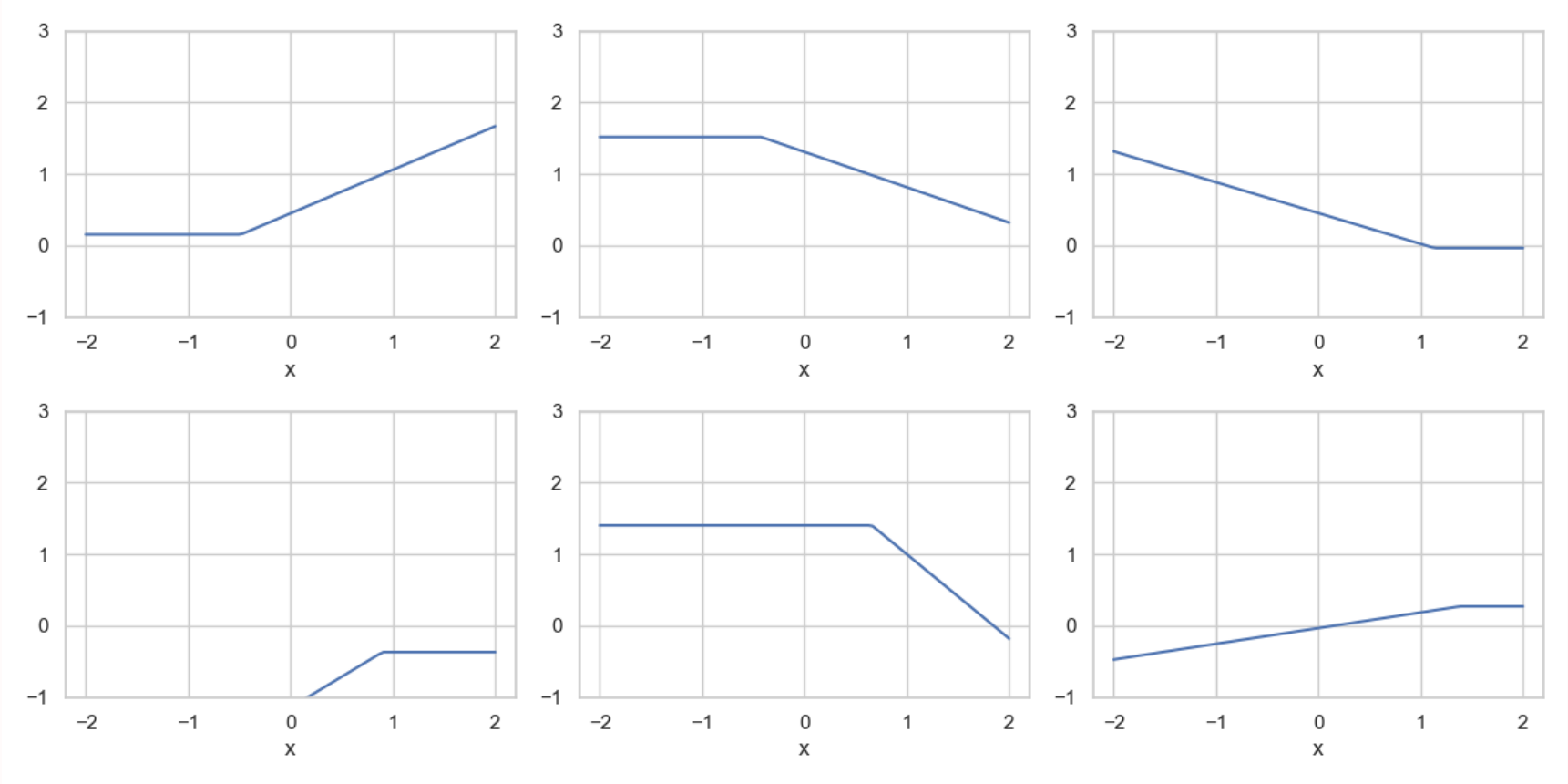
- Scale and shift output as well as input:

$$y = w_{\text{out}}\text{ReLU}(w_{\text{in}}x + b_{\text{in}}) + b_{\text{out}}$$



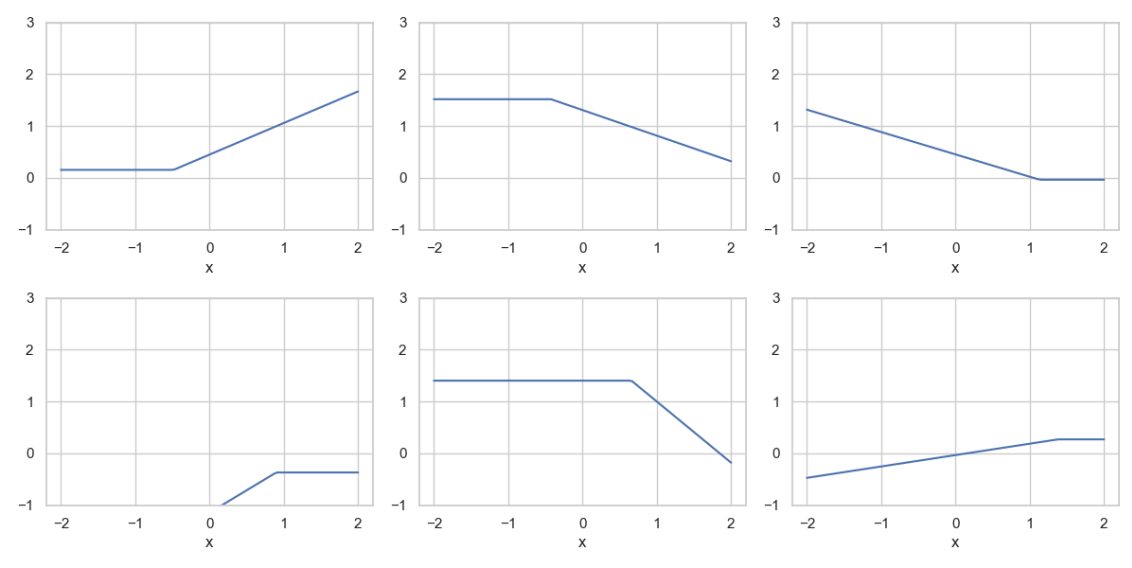
Single neuron functions

- What functions can $y = w_{\text{out}}\text{ReLU}(w_{\text{in}}x + b_{\text{in}}) + b_{\text{out}}$ represent?
- Choose $w_{\text{in}}, b_{\text{in}}, w_{\text{out}}, b_{\text{out}}$ at random a few times:

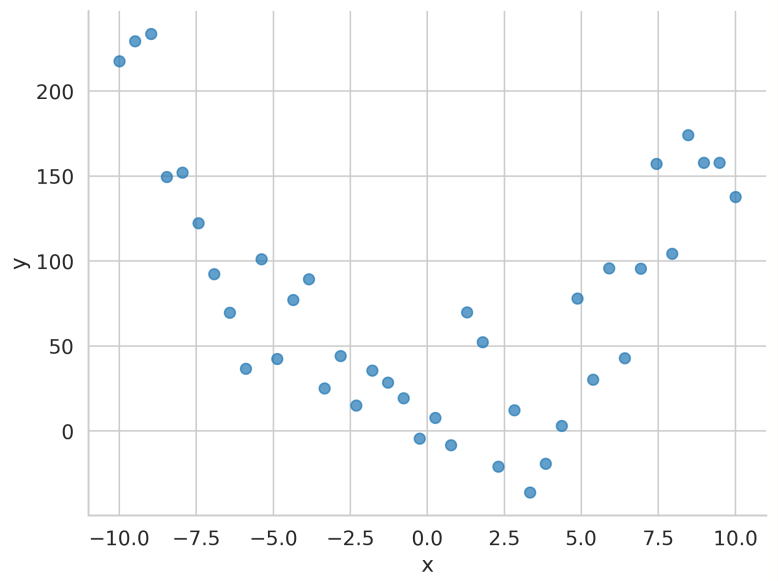


Single neuron functions

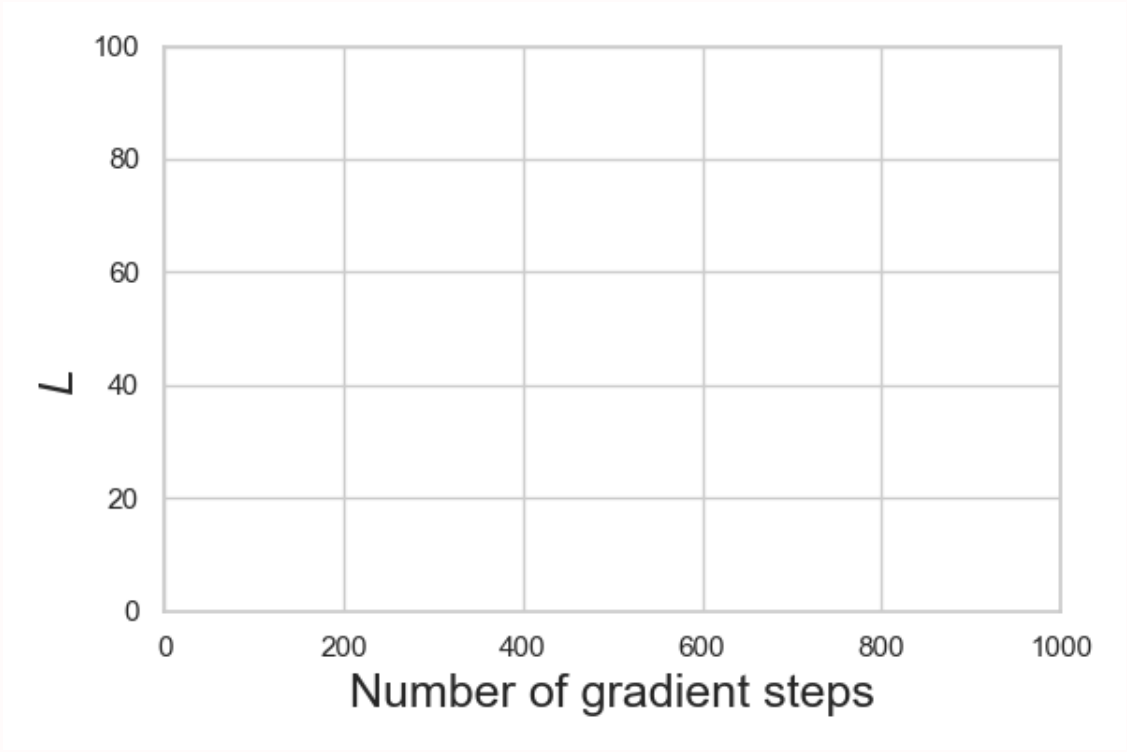
- Can $y = w_{\text{out}}\text{ReLU}(w_{\text{in}}x + b_{\text{in}}) + b_{\text{out}}$ fit the data?
  - Doesn't flatten to zero: b_{out}.
  - Can flip upside down: w_{out}.
- Still has one kink!
- Try gradient descent anyway!

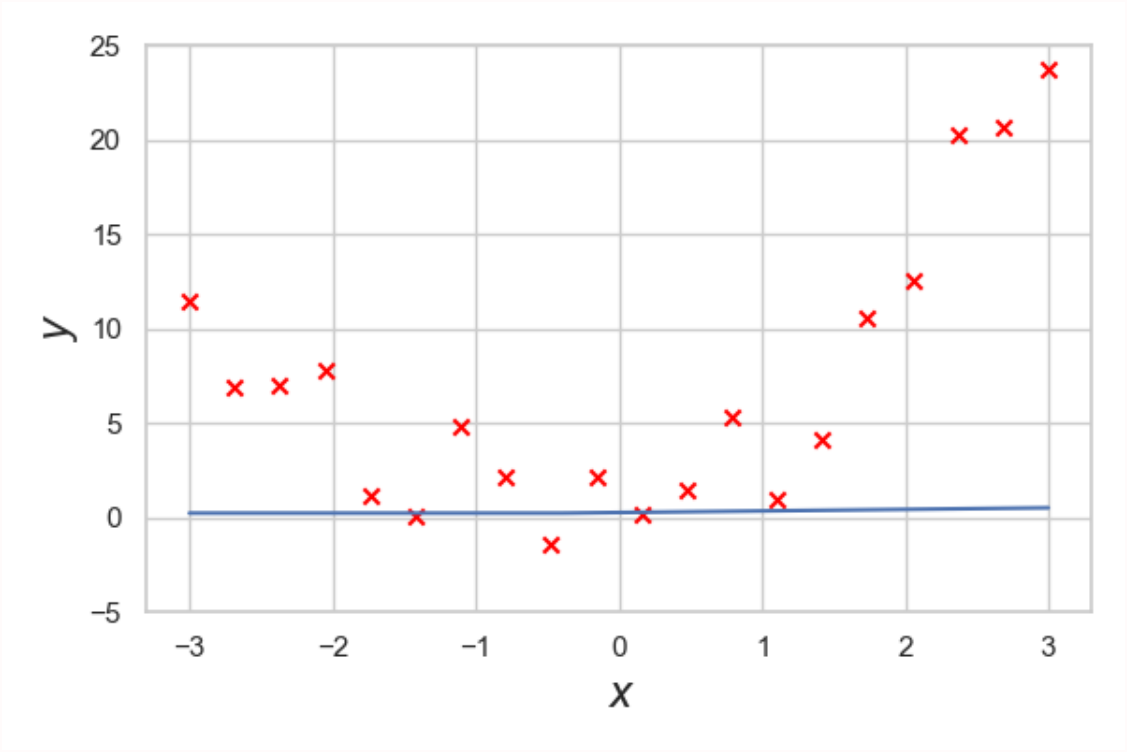


≠

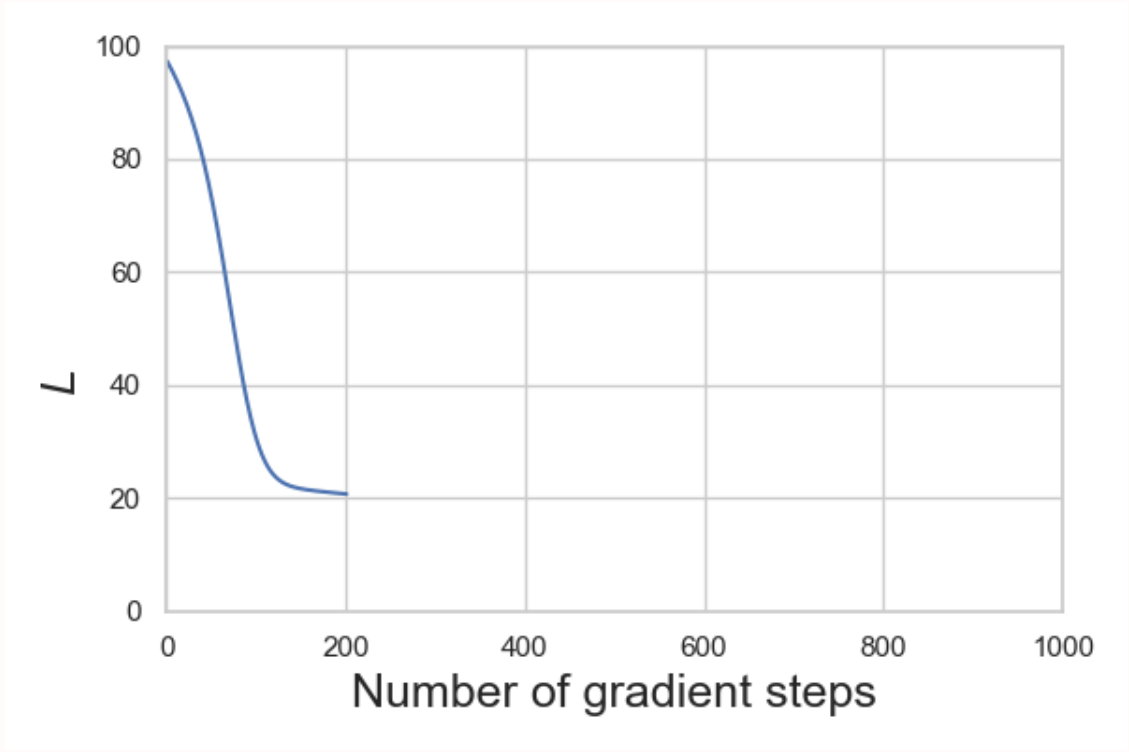


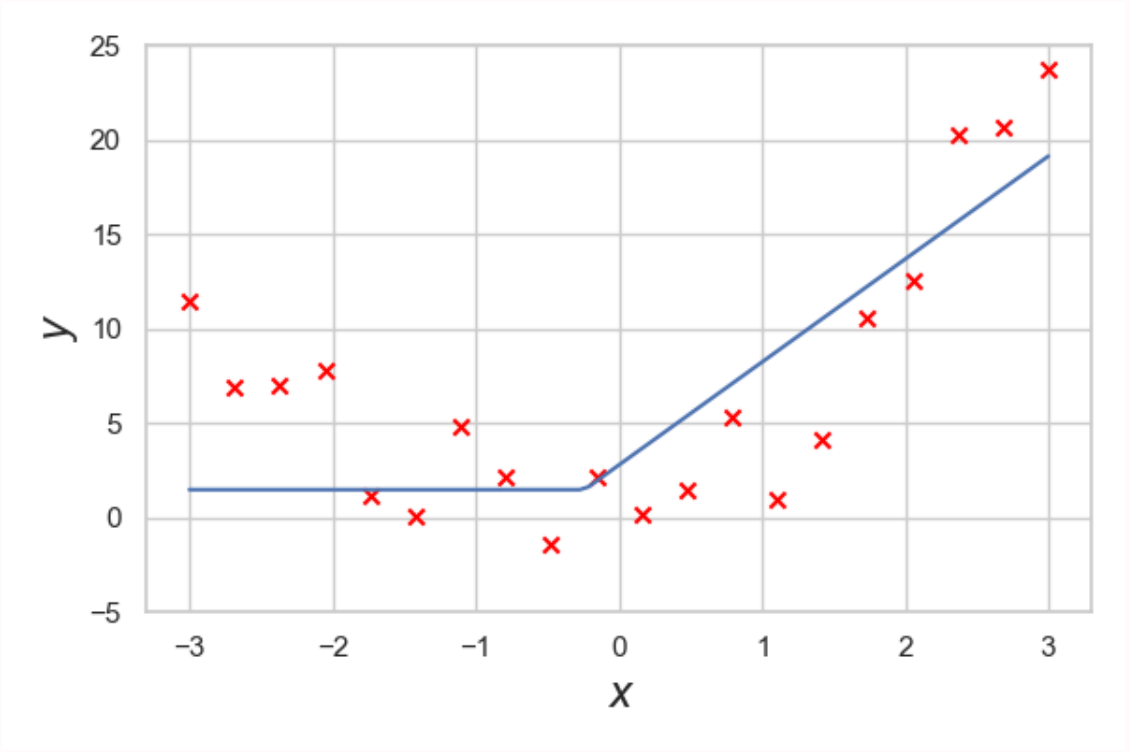
Fitting data with one scaled ReLU



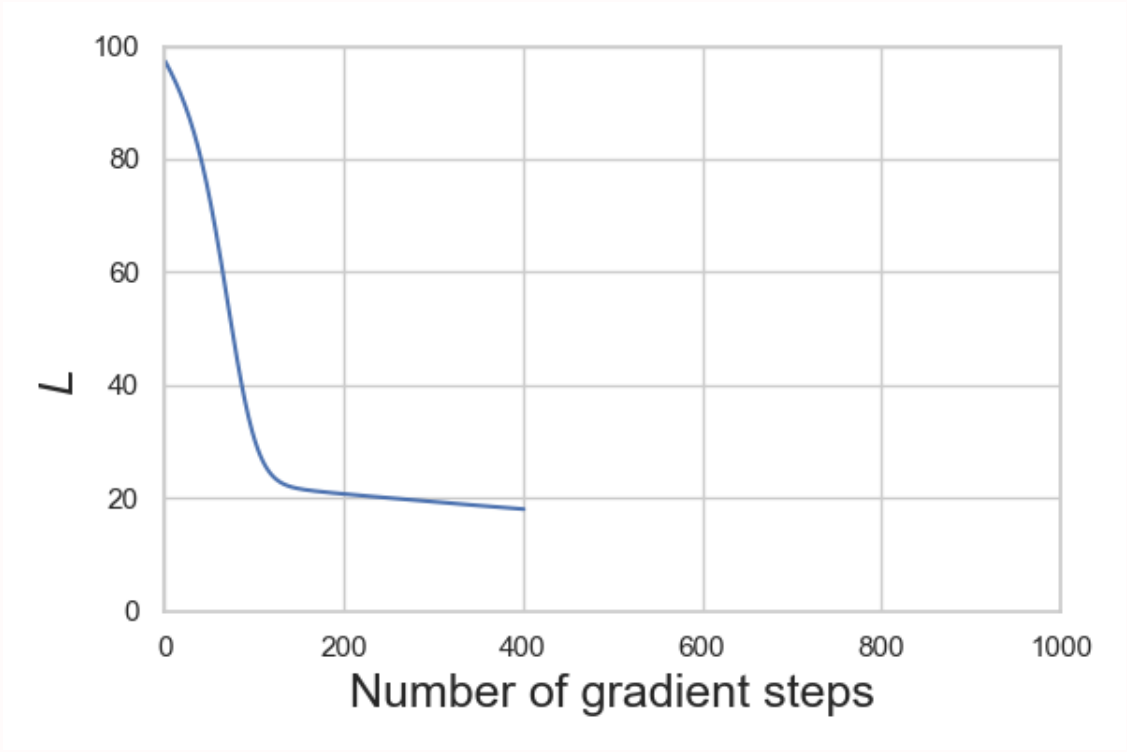


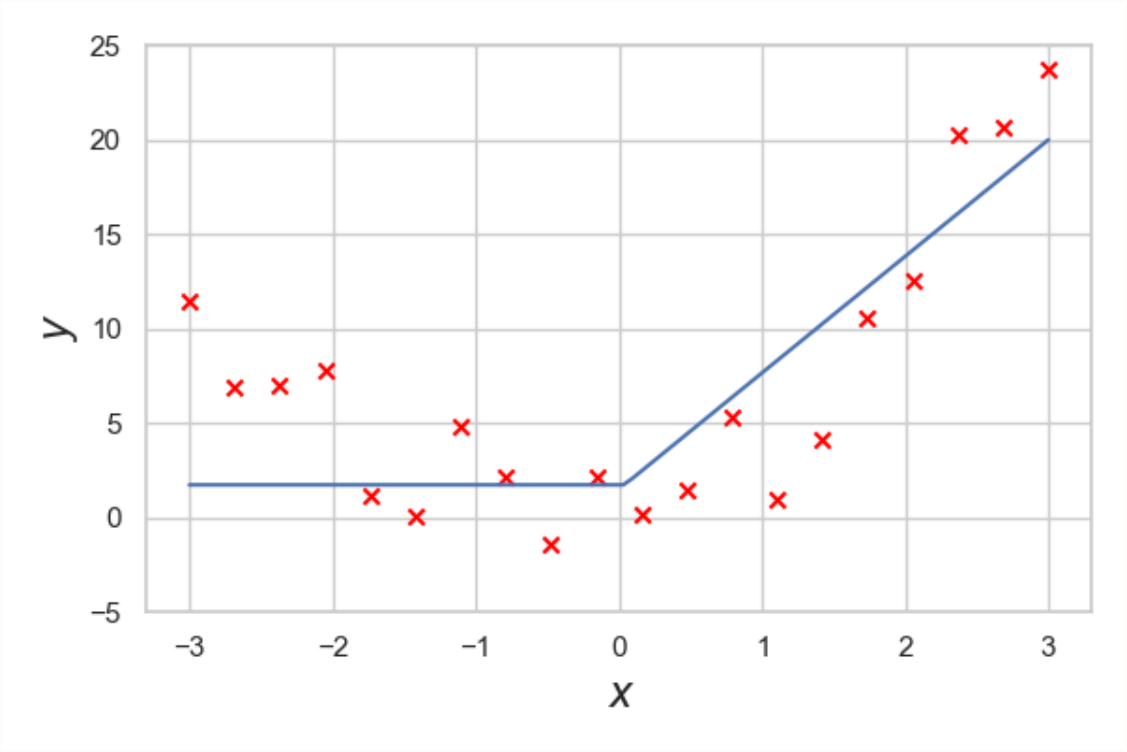
Fitting data with one scaled ReLU



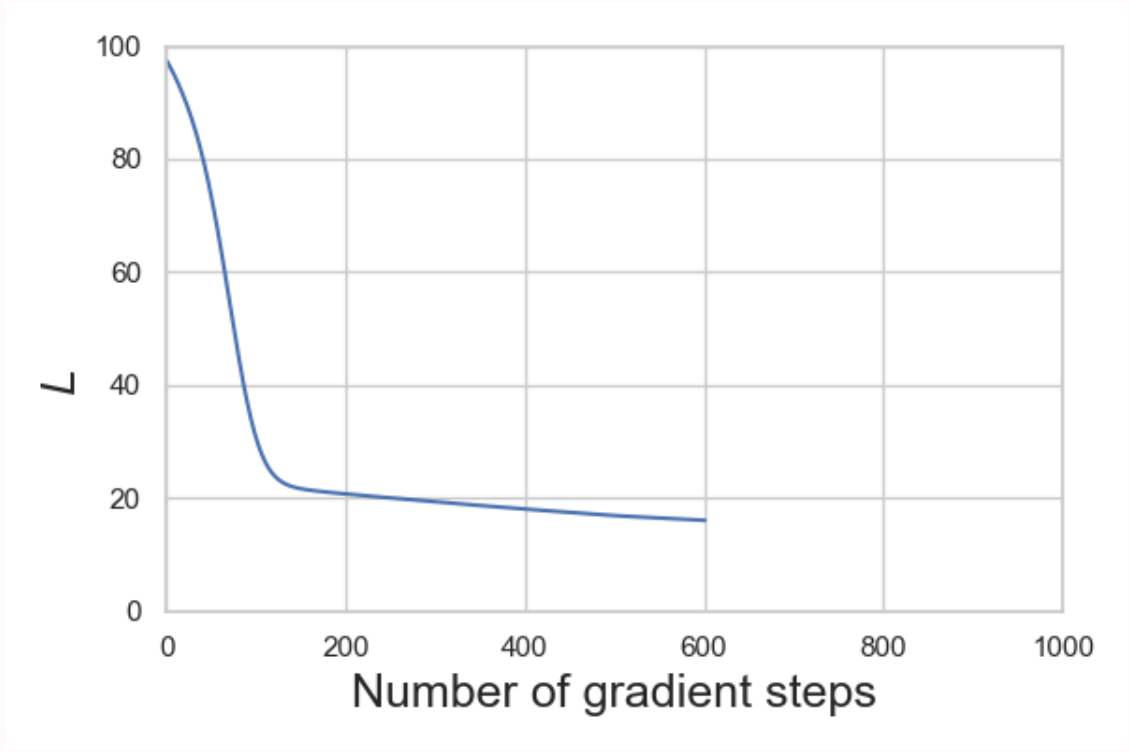


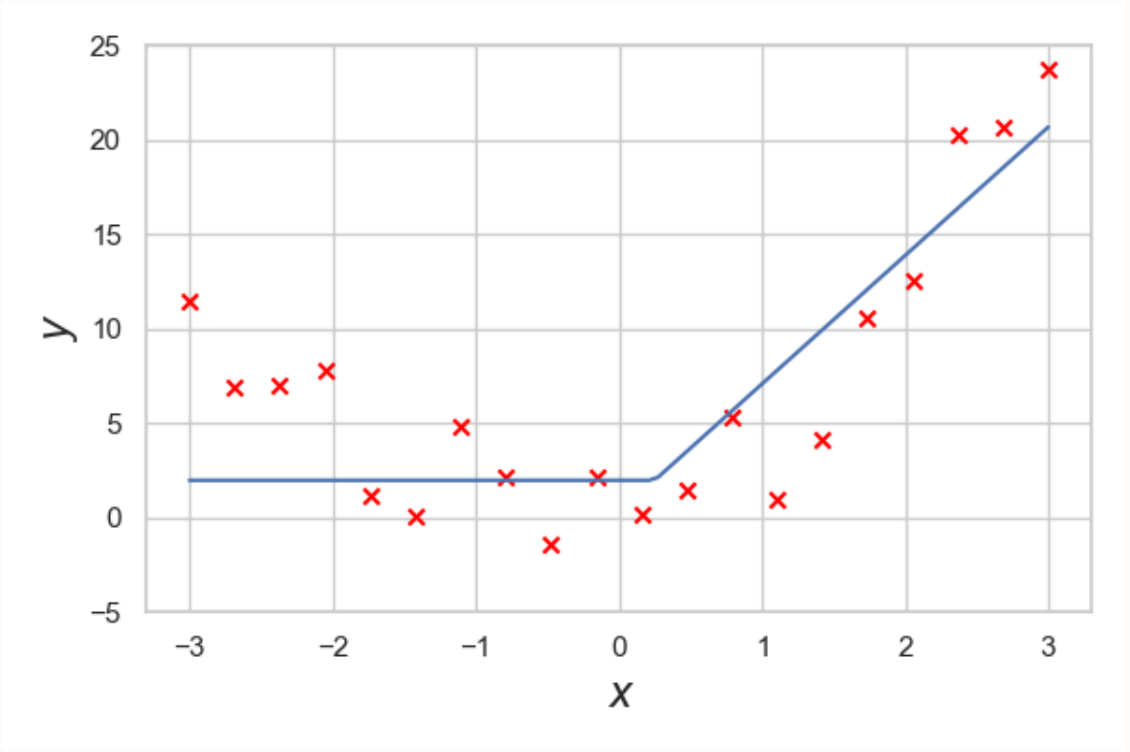
Fitting data with one scaled ReLU



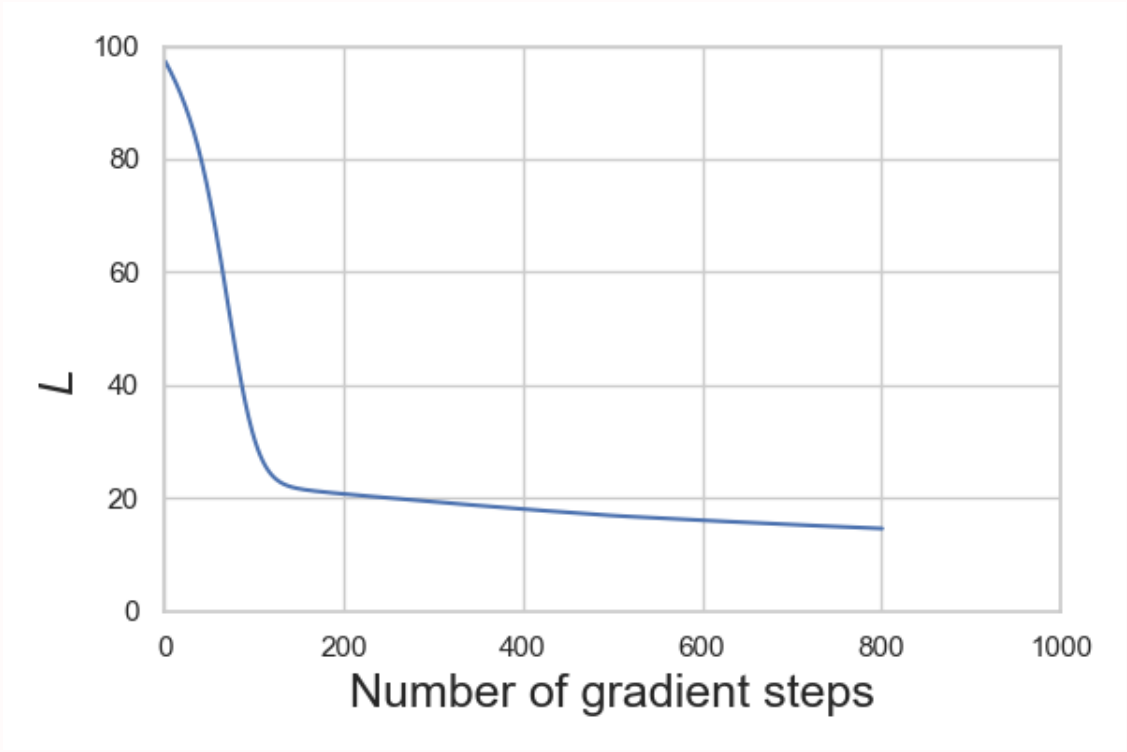


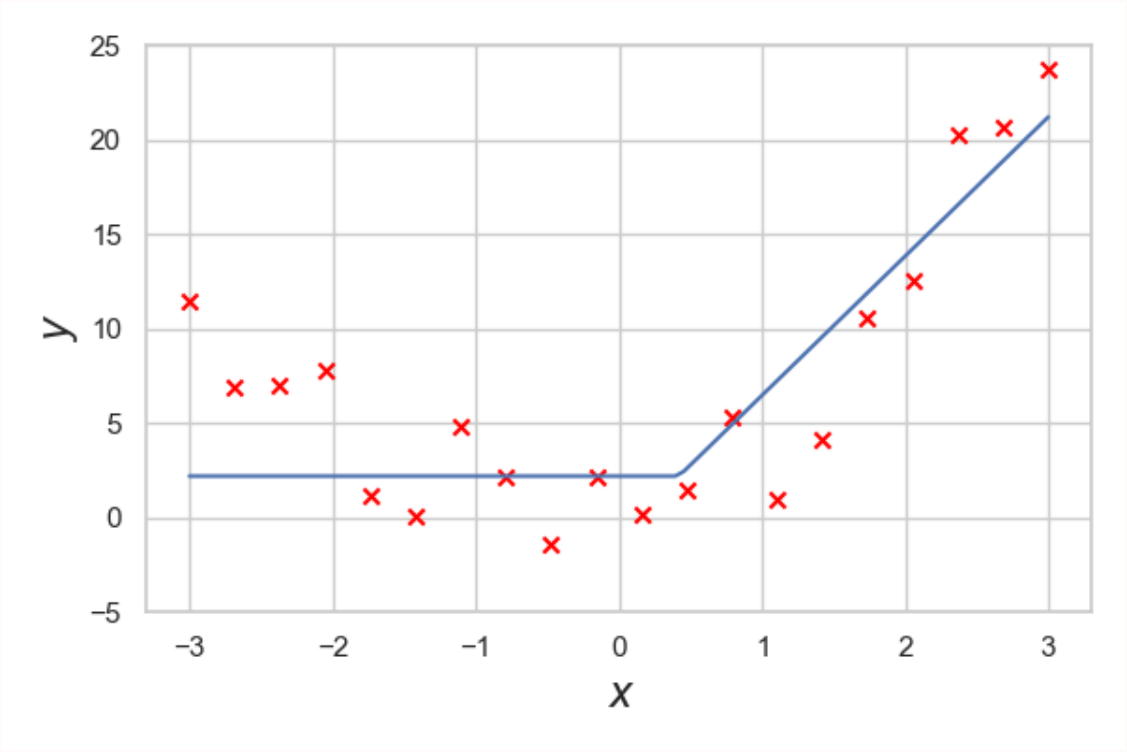
Fitting data with one scaled ReLU



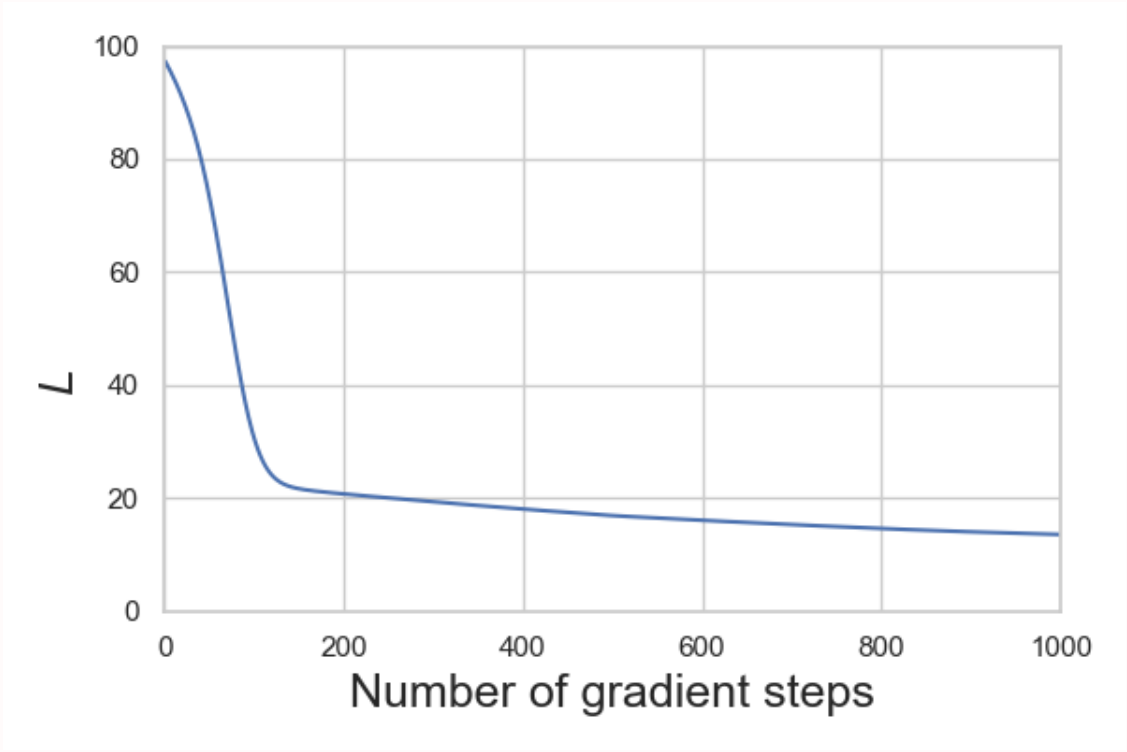


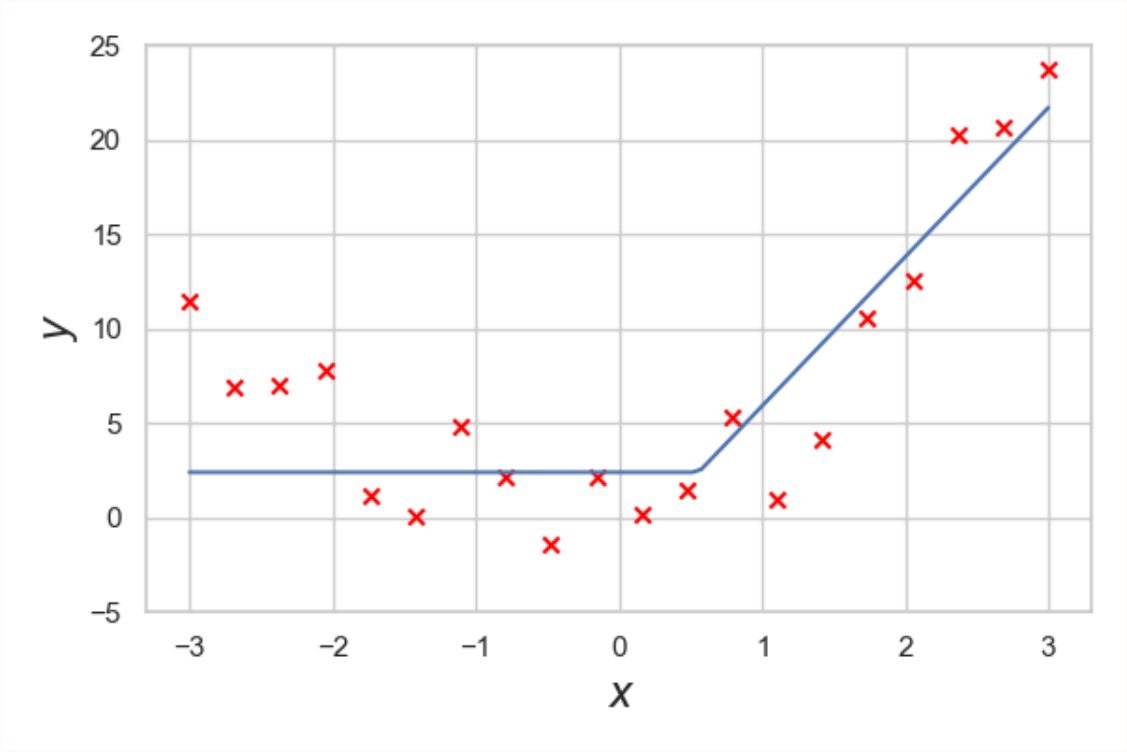
Fitting data with one scaled ReLU





Fitting data with one scaled ReLU



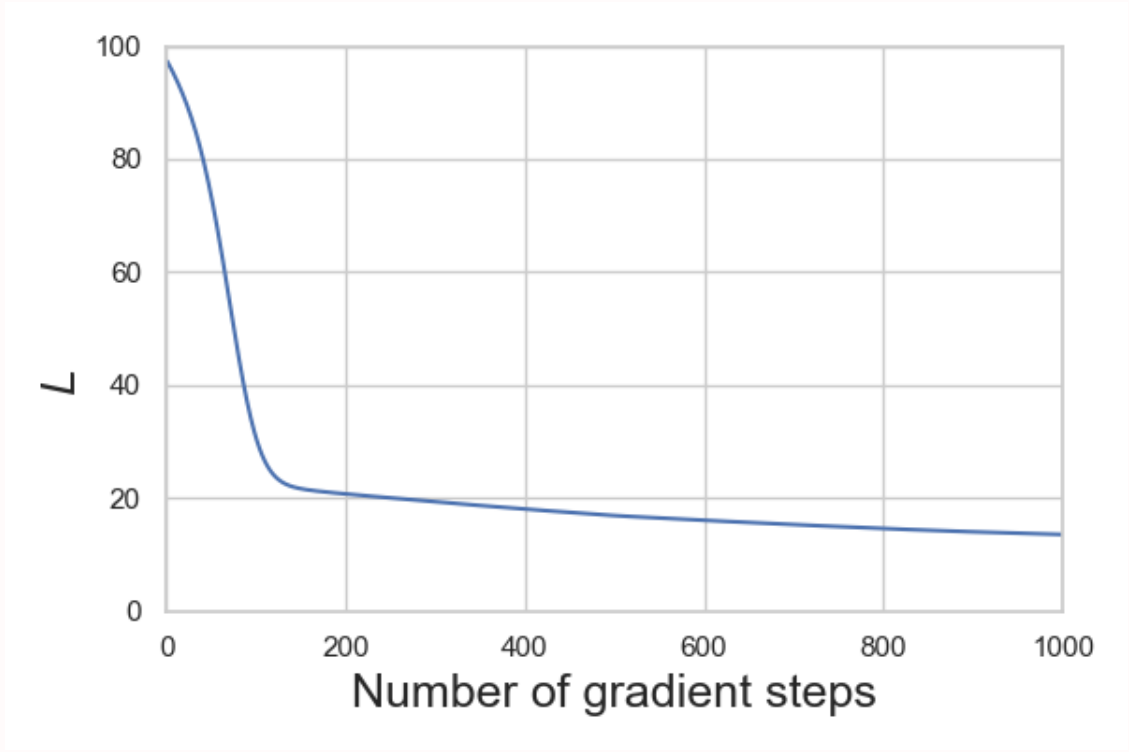


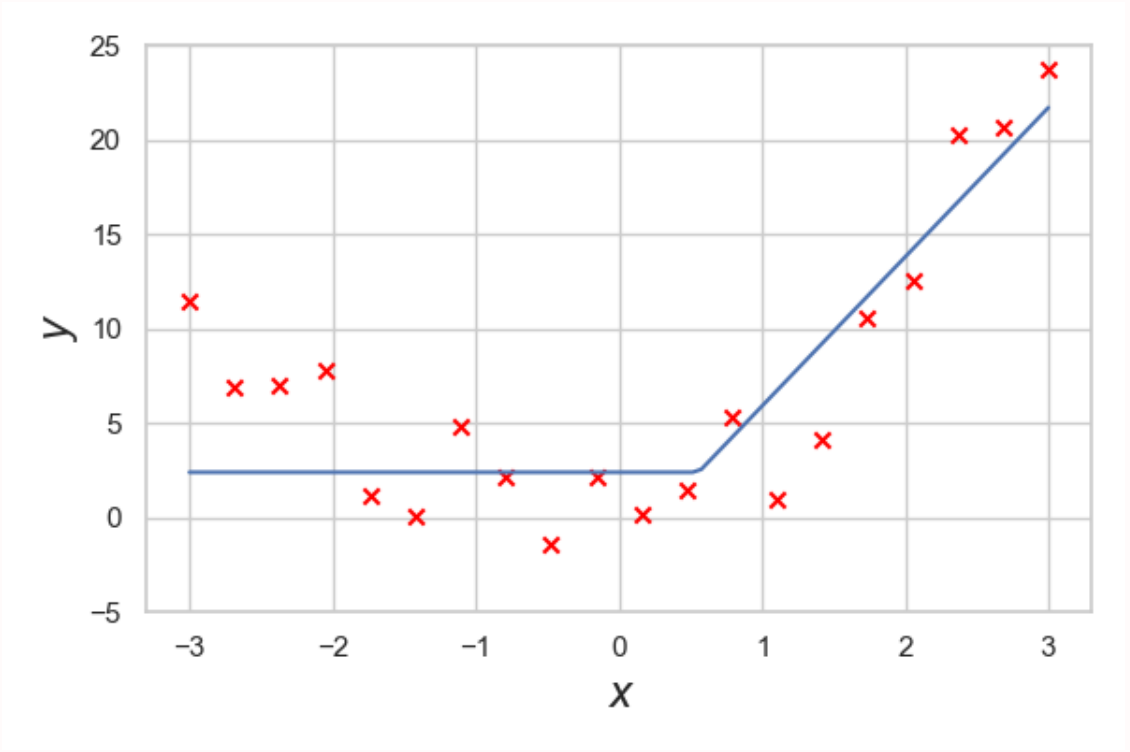
- Our best fit so far! (loss ≈ 15 vs ≈ 25 for non-scaled ReLU and ≈ 40 for linear regression).
- Still misses some points.
- Need *even more* flexibility.

Some Philosophy

Why do we need many neurons?

Fitting data with one scaled ReLU





- A single neuron can't fit the data.
- Need *even more* flexibility.

Neural networks

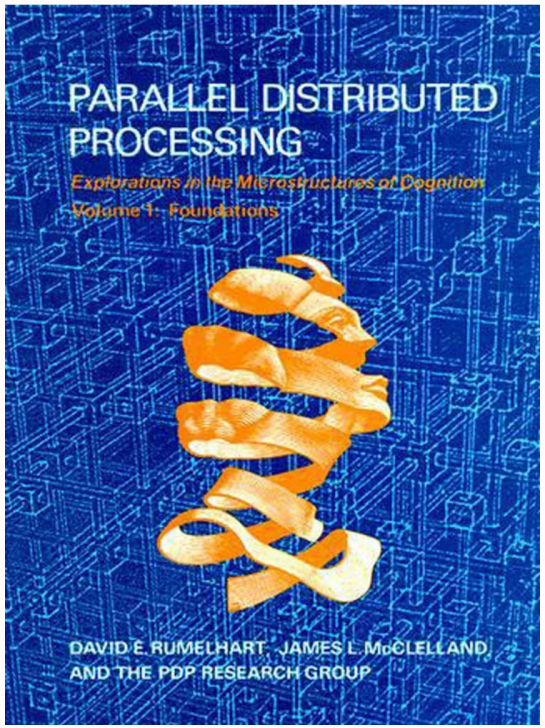
- Guiding principle of deep learning:
  - Complexity arises through the coordination of *many simple units*.
  - Each is insufficient to solve the problem on its own.
  - It is in *how the units interact* that information is stored.
- Solving non-trivial problems requires more units connected in more ways than a human can understand.
  - Need learning algorithms that require *minimal human intervention*.
  - *Interpretability* of the model becomes challenging.
  - Contrast with the field of STATISTICS.

Connectionism

SOME HISTORY:

- Much of this philosophy was outlined in a seminal 1989 textbook, *Parallel Distributed Processing*:

> "THE AUTHORS' theory assumes the mind is composed of a great number of elementary units connected in a neural network. Mental processes are interactions between these units which excite and inhibit each other in parallel rather than sequential operations. In this context, knowledge can no longer be thought of as stored in localized structures; instead, it consists of the connections between pairs of units that are distributed throughout the network."



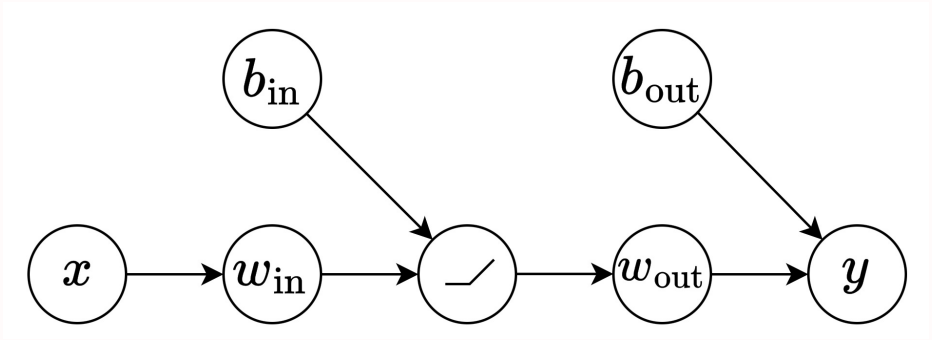
David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group (Eds.). 1986. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA.

Our First Neural Network

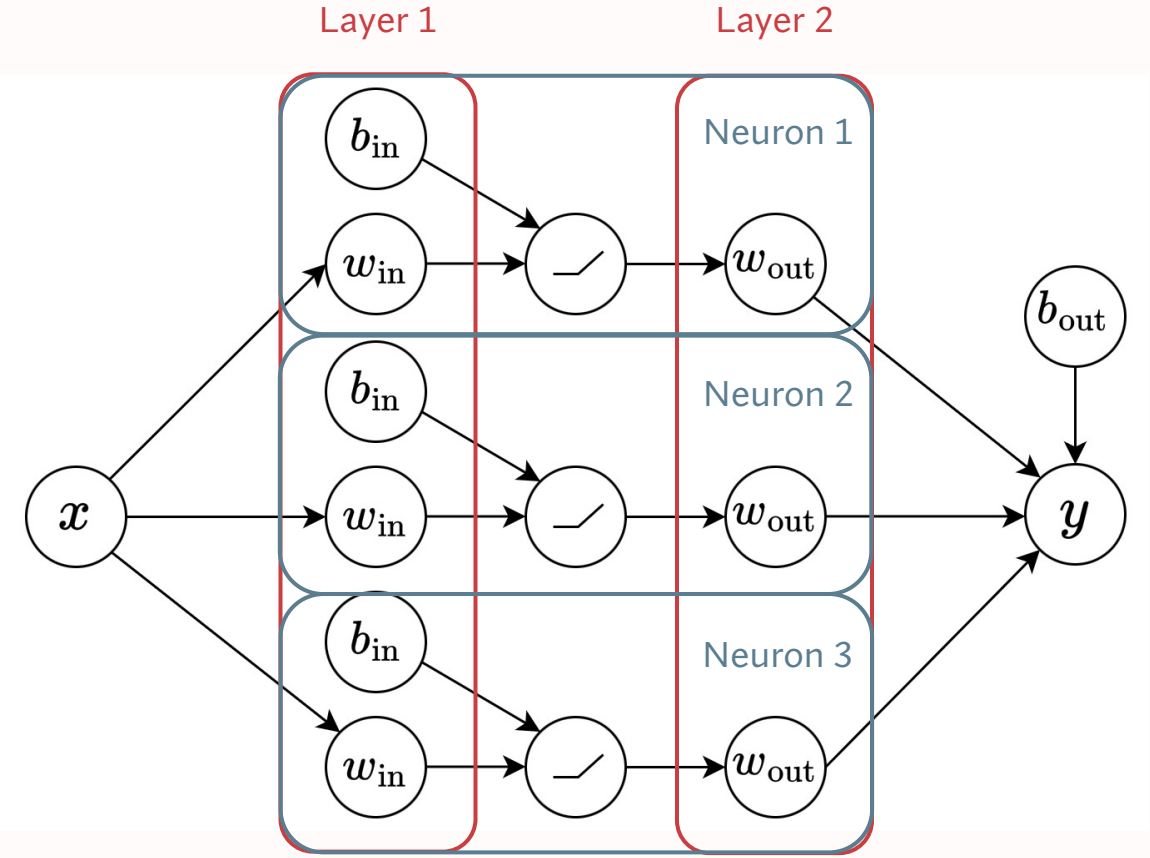
Construction and terminology

First neural network

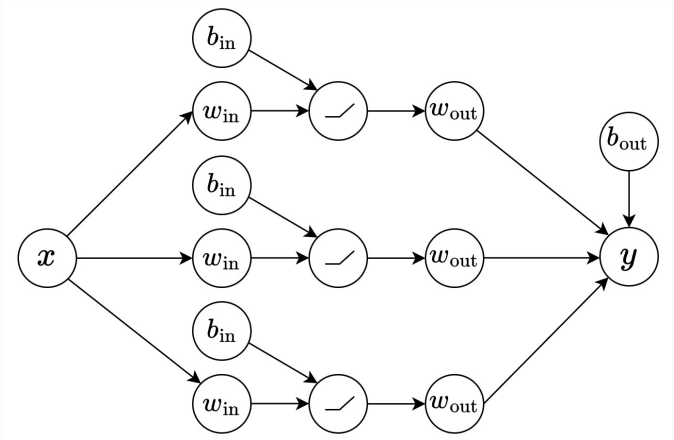
- Time to construct our first *neural network*.
- Add together many ReLU neurons.
- *Two-layer*, *three-neuron* neural network.



Scale up

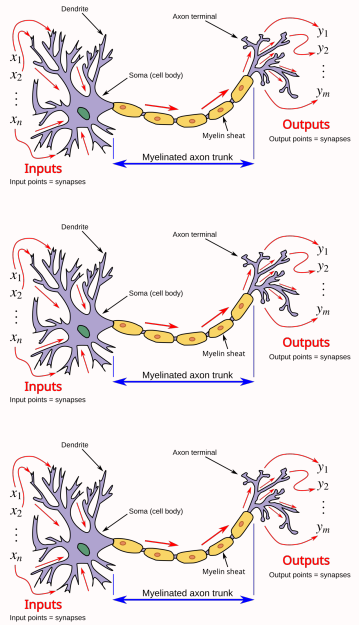


First neural network



$\approx$

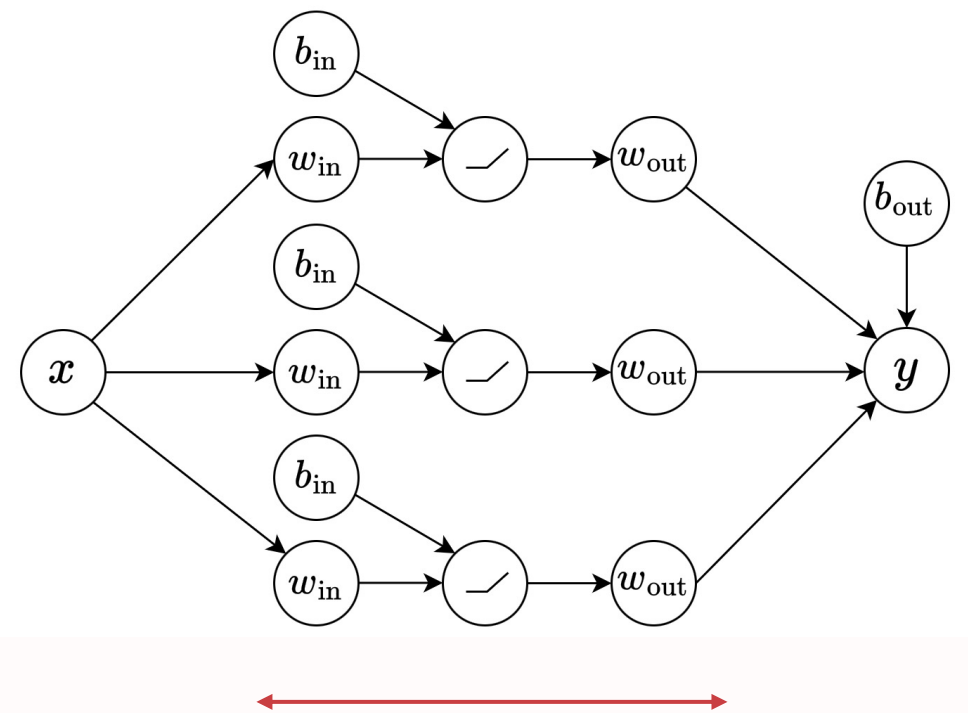
STIMULUS



RESPONSE

First neural network

Some terminology:



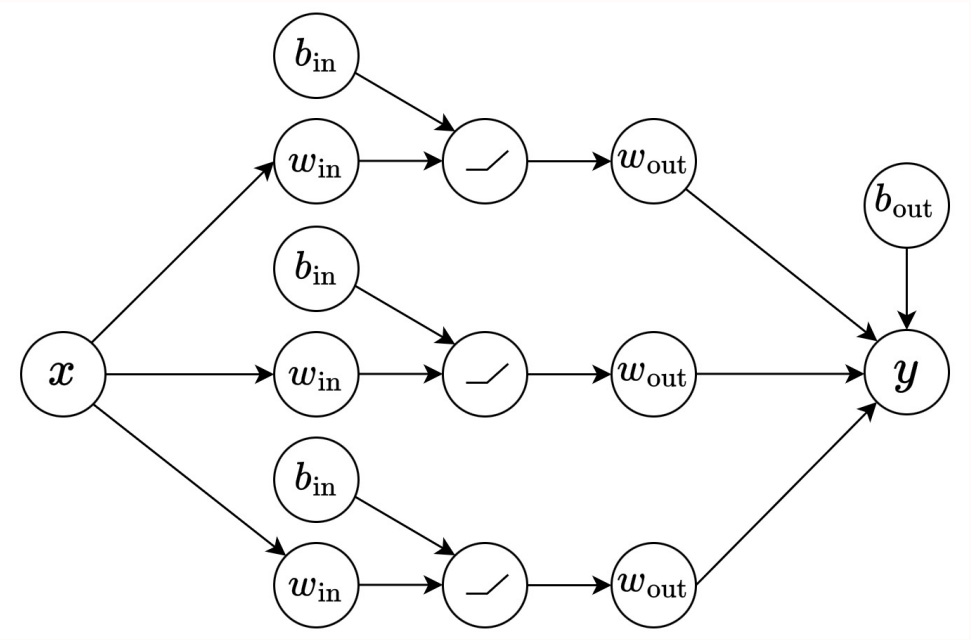
Depth: two-layer network

NOTE:

- The concept of depth is where we get the name *deep learning*.
- Linear regression can be thought of as a neural network with only one layer.

First neural network

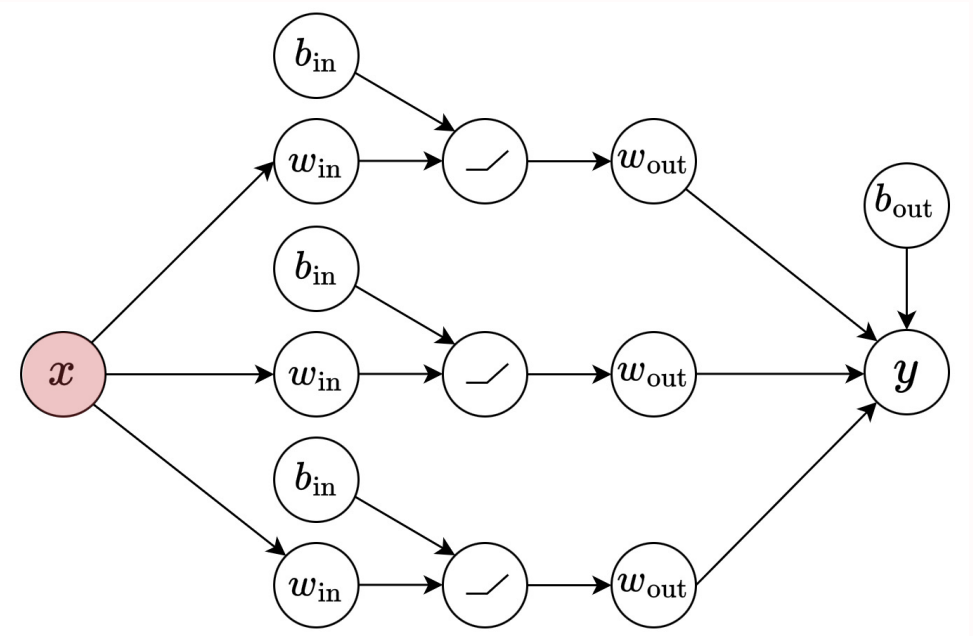
Some terminology:



Width:
three neurons wide

First neural network

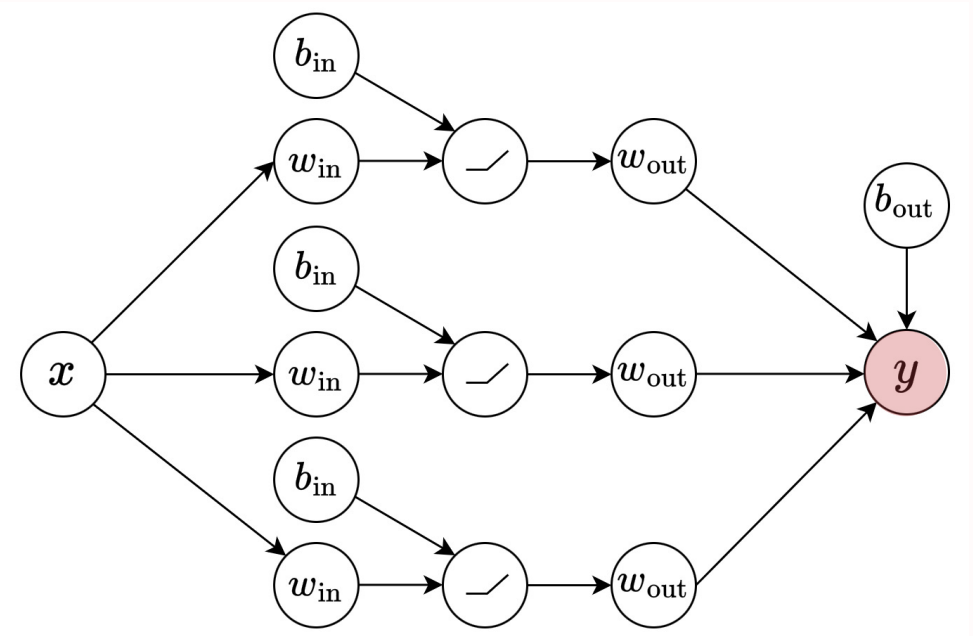
Some terminology:



Input

First neural network

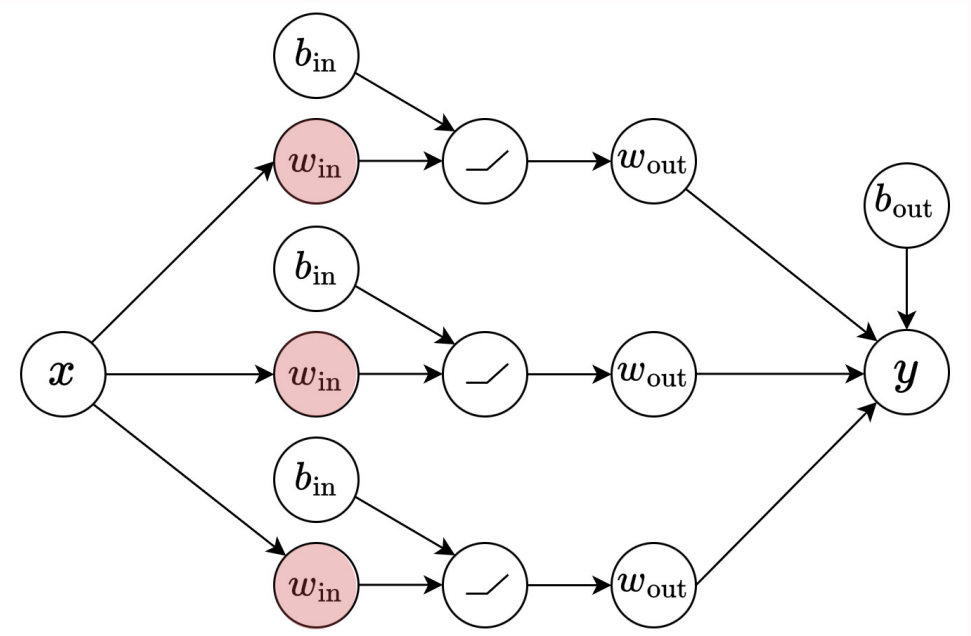
Some terminology:



Output

First neural network

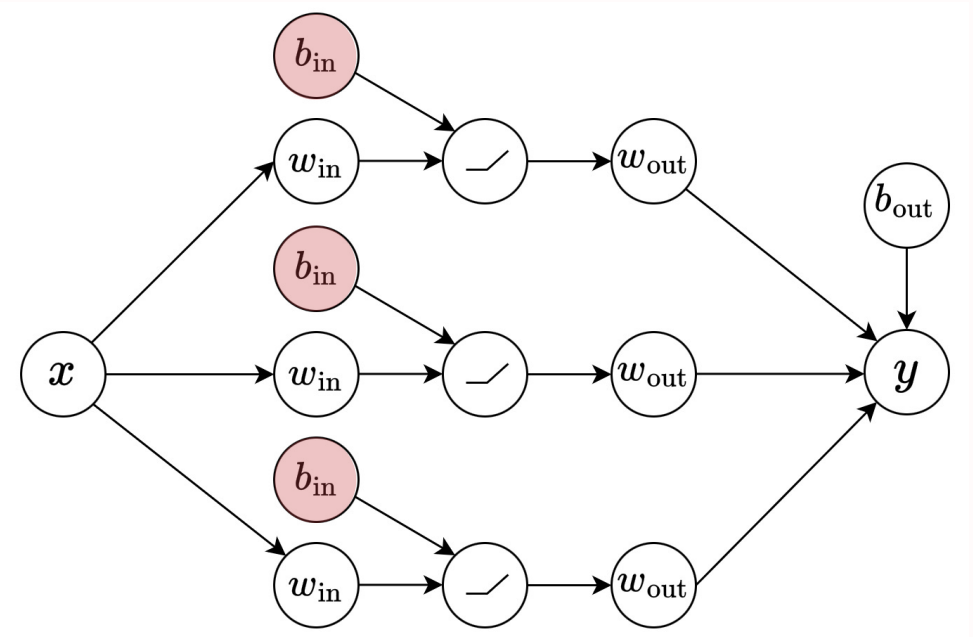
Some terminology:



First-layer weights

First neural network

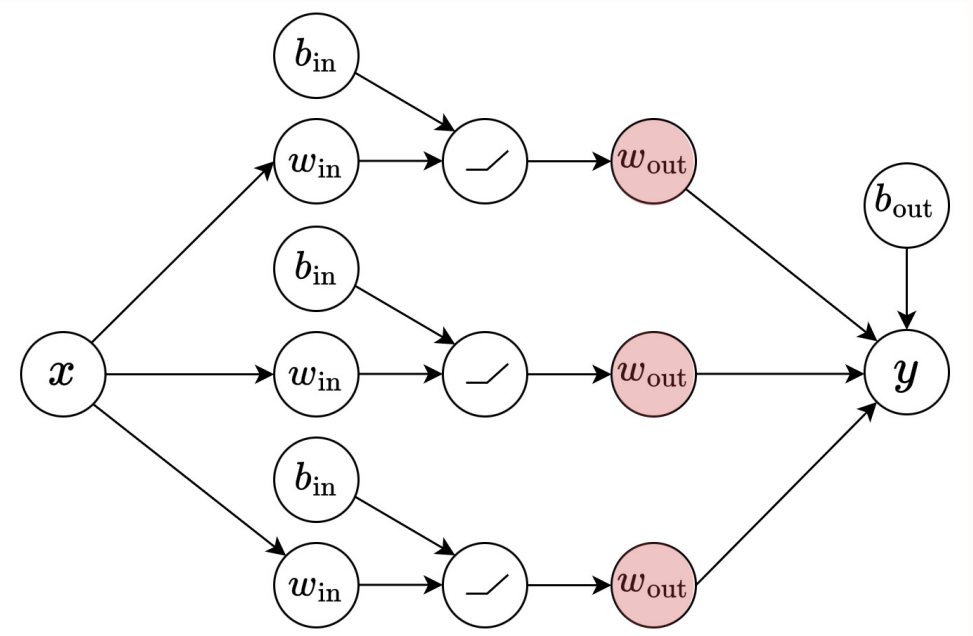
Some terminology:



First-layer biases

First neural network

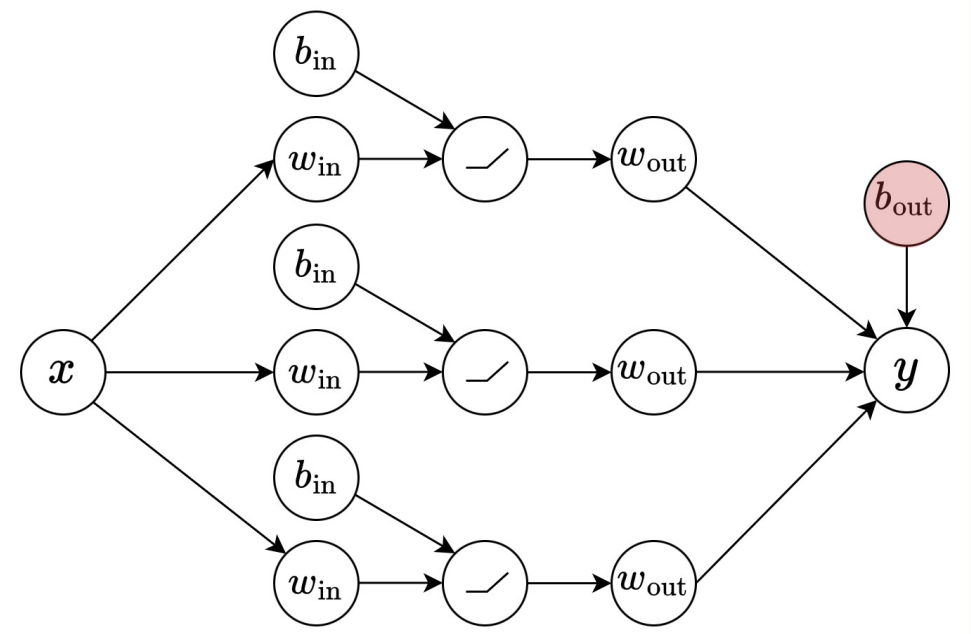
Some terminology:



Second-layer weights

First neural network

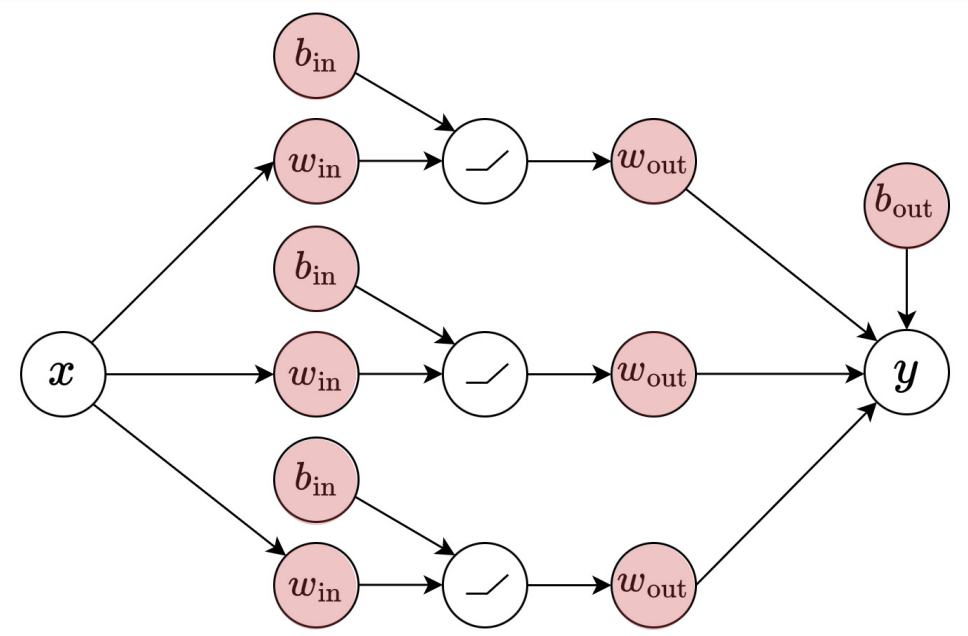
Some terminology:



Second-layer bias(es)

First neural network

Some terminology:



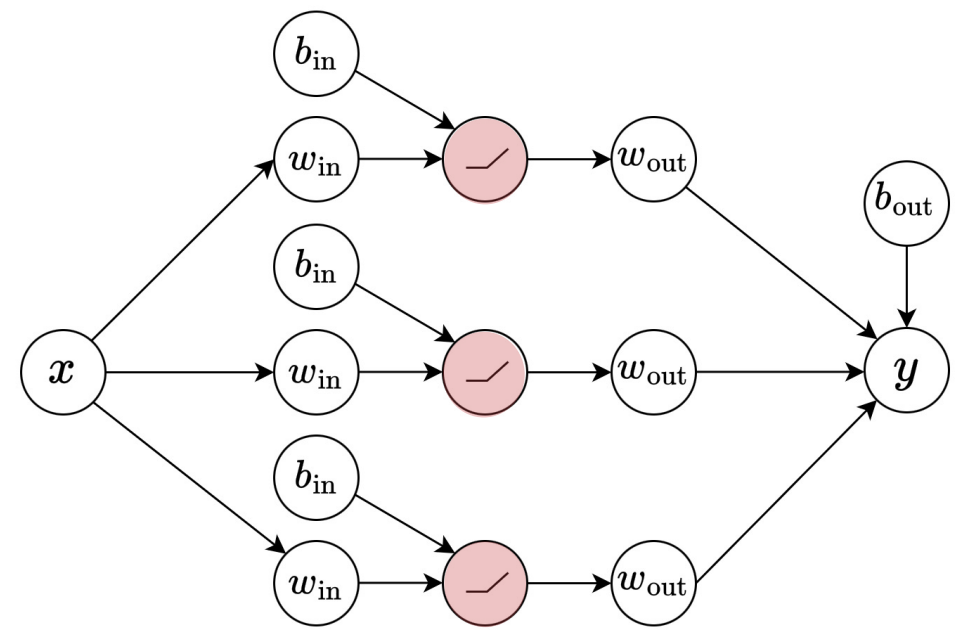
Parameters

NOTE:

- This neural network has ten parameters.
- It's common to refer to all parameters as *weights* even though this includes biases as well.
- ChatGPT uses networks estimated to have ≈ 1 trillion parameters.
- The exact number is undisclosed.

First neural network

Some terminology:



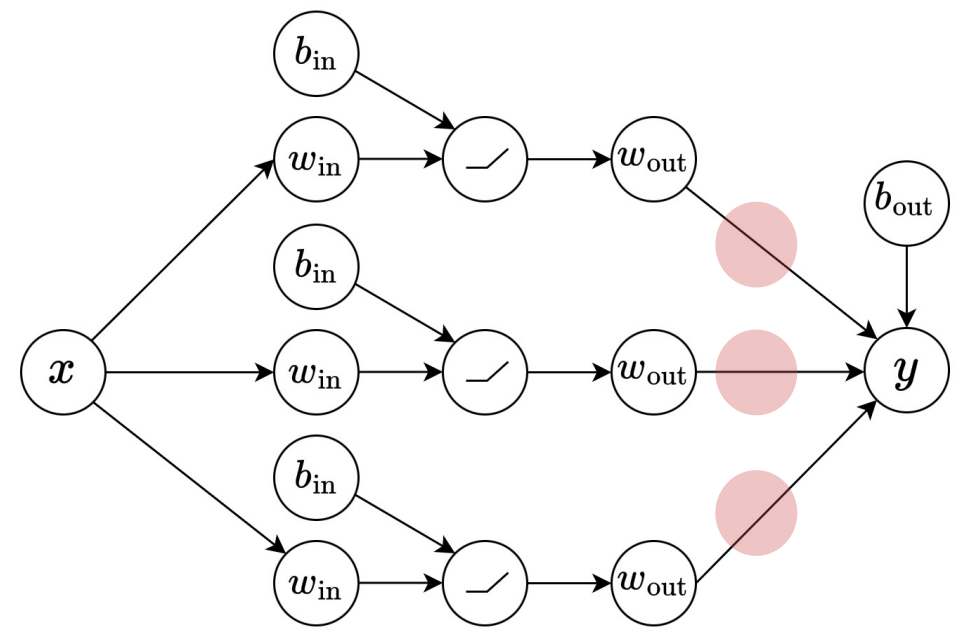
Non-linearities/activation functions

NOTE:

- The function itself is the activation function or non-linearity.
- But the *output* of the function is called the *activity*.

First neural network

Some terminology:



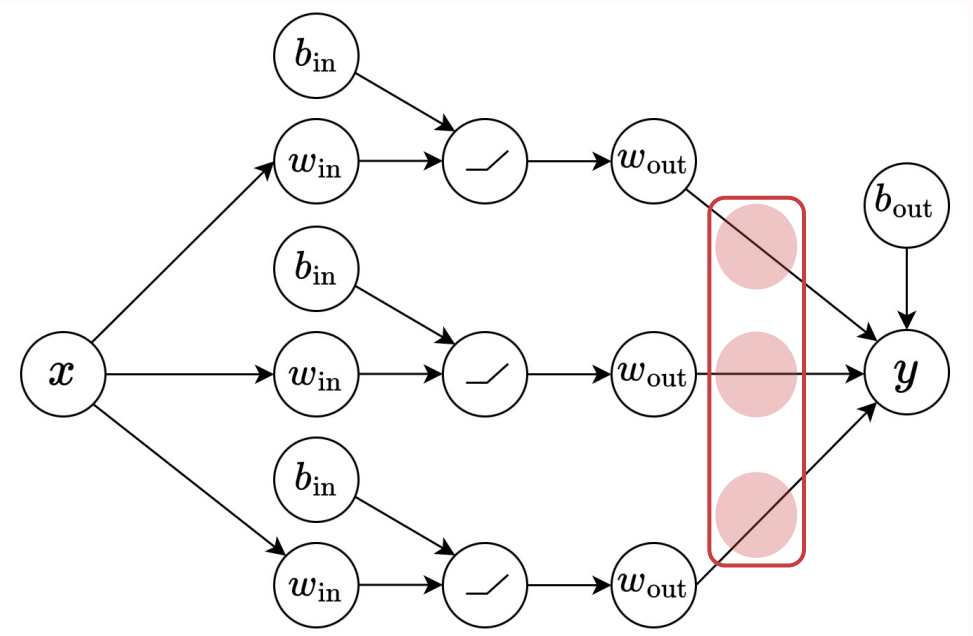
Hidden features

NOTE:

- The hidden features are the outputs of the individual neurons, *before* they are combined.
- I am representing them as "being at" the arrows after the output weights, which is non-standard.

First neural network

Some terminology:



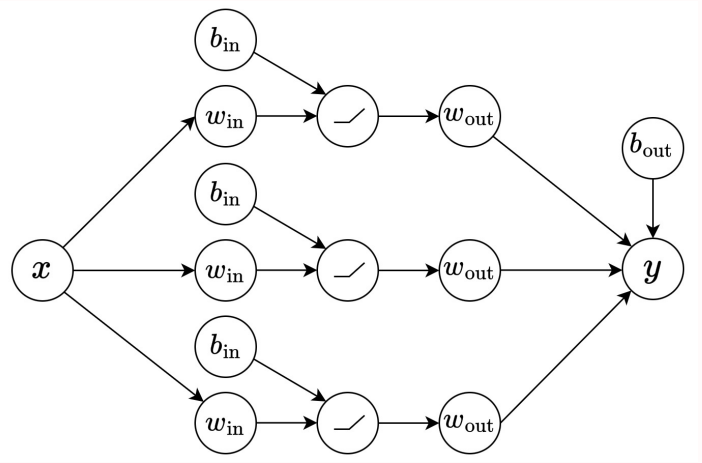
Hidden layer
This neural network only has one.

Gradient descent

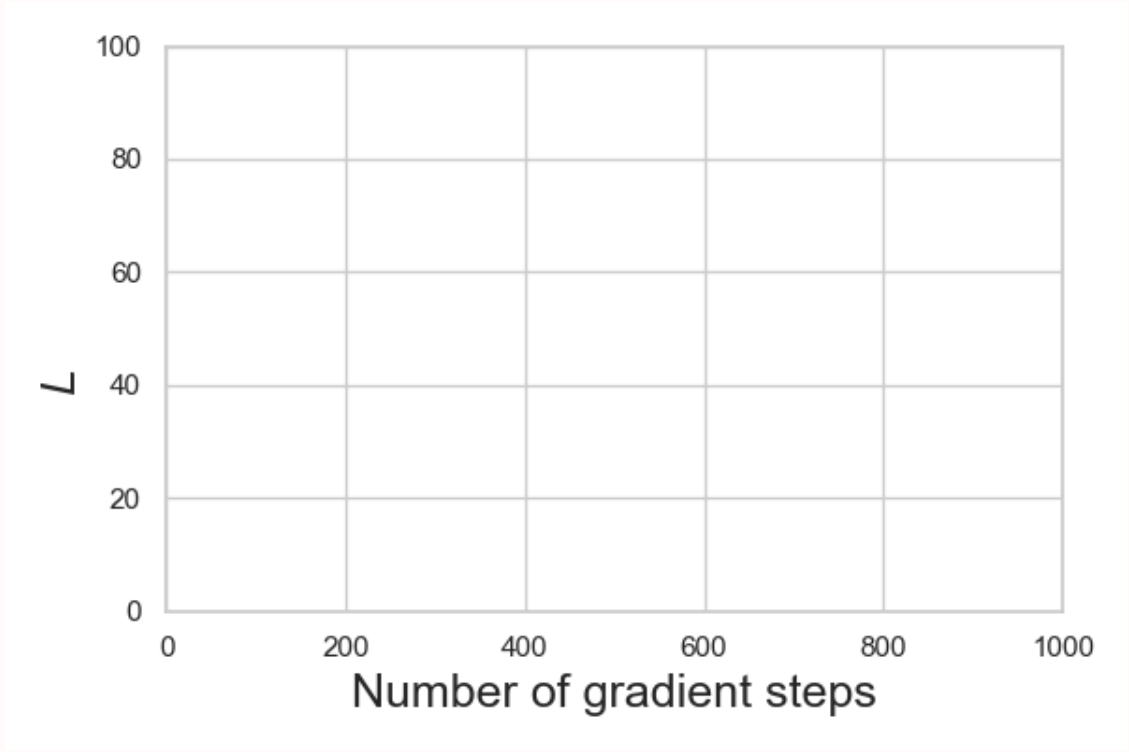
Learning with our first neural network

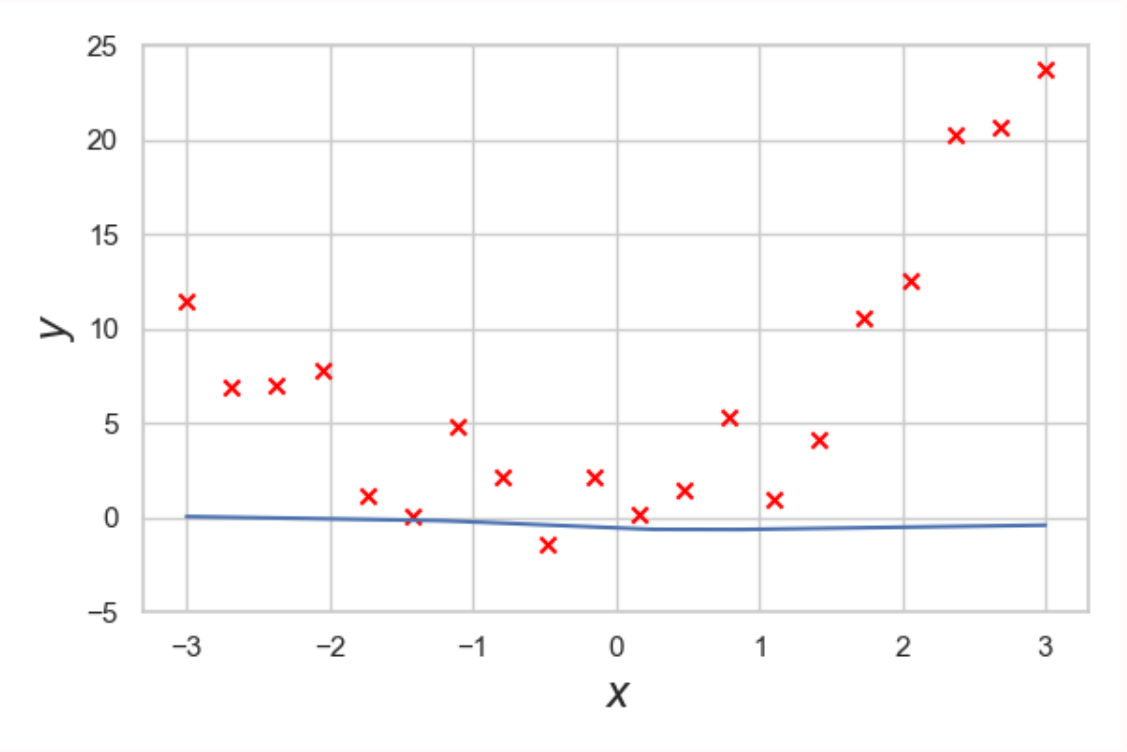
First neural network

- Try GRADIENT DESCENT again.
- Loss function now has *ten* inputs:
  - $L(w_{\text{in},1}, w_{\text{in},2}, w_{\text{in},3}, b_{\text{in},1}, b_{\text{in},2}, b_{\text{in},3}, w_{\text{out},1}, w_{\text{out},2}, w_{\text{out},3}, b_{\text{out}})$
  - Written $L(\mathbf{w})$ for short.
- Ten-dimensional optimization problem.
  - Cannot be visualized straightforwardly.
  - Gradient descent works just the same!
  - Need to compute *ten* partial derivatives:
  - $\nabla L(\mathbf{w}) = \left[\frac{\partial L}{\partial w_{\text{in},1}}(\mathbf{w}), \frac{\partial L}{\partial w_{\text{in},1}}(\mathbf{w}), \ldots\right]$.
- With multiple layers, computers use the *backpropagation algorithm* to compute $\nabla L(\mathbf{w})$.
  - This is called the *backward pass*.
  - Takes roughly the same time as computing y from x, which is called the *forward pass*.
  - Won't go into detail—uses chain rule of calculus.

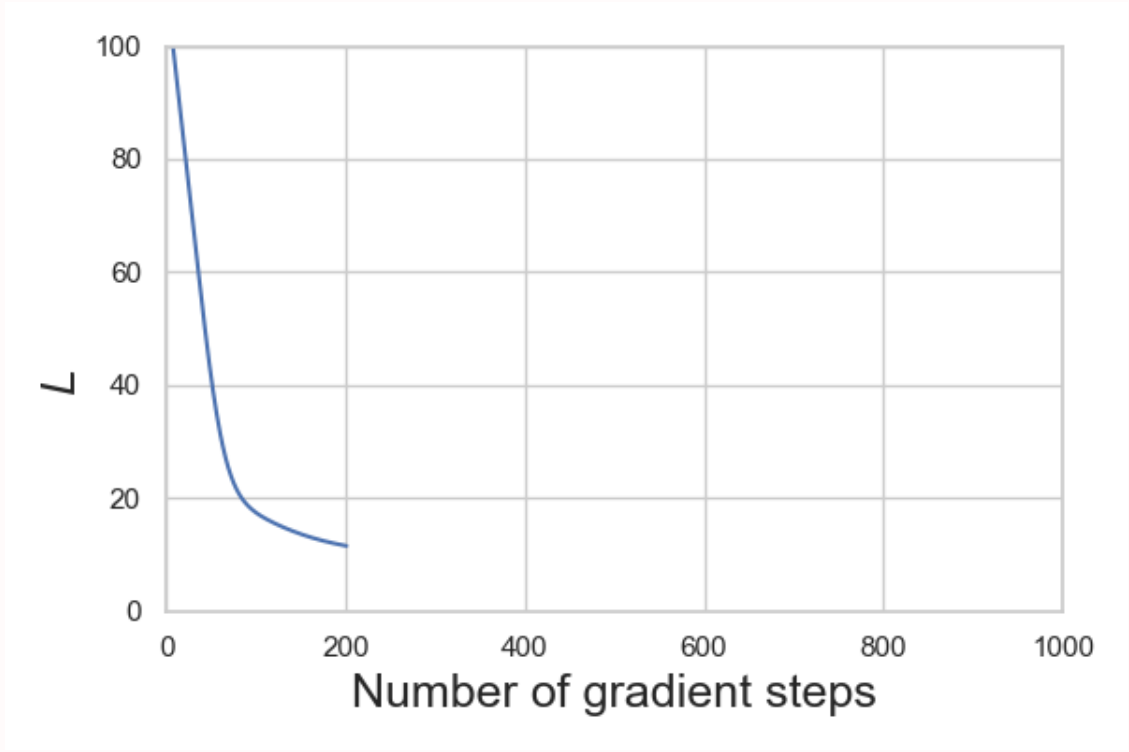


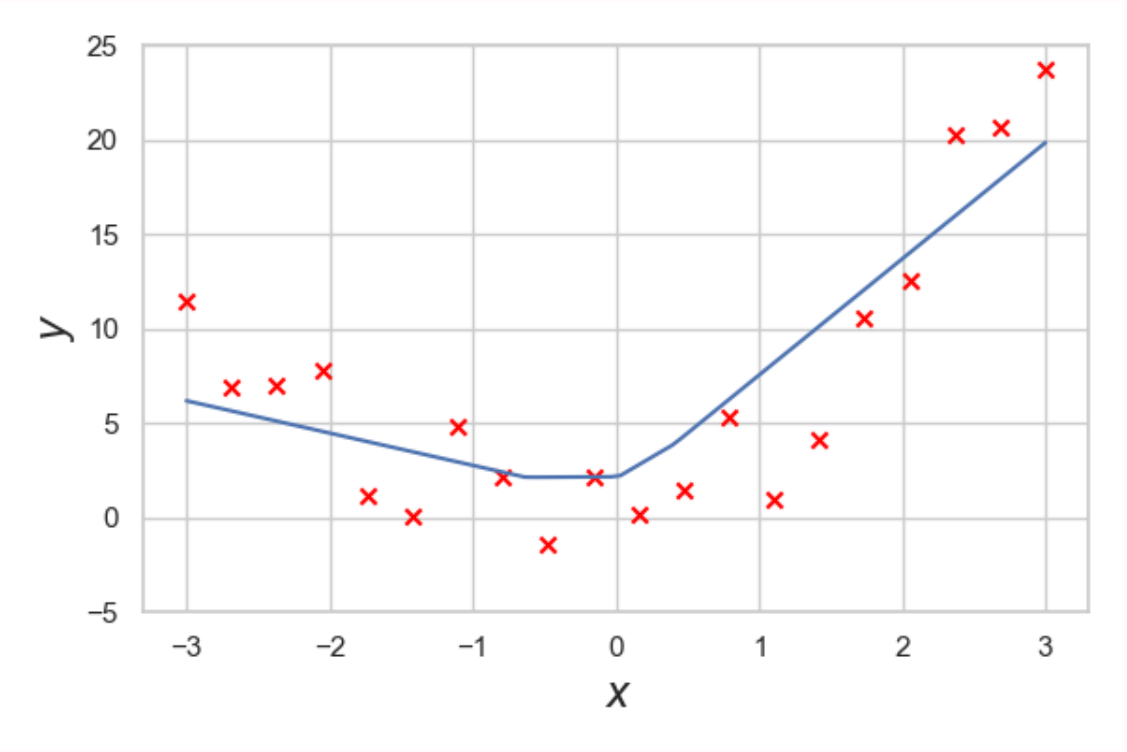
First neural network



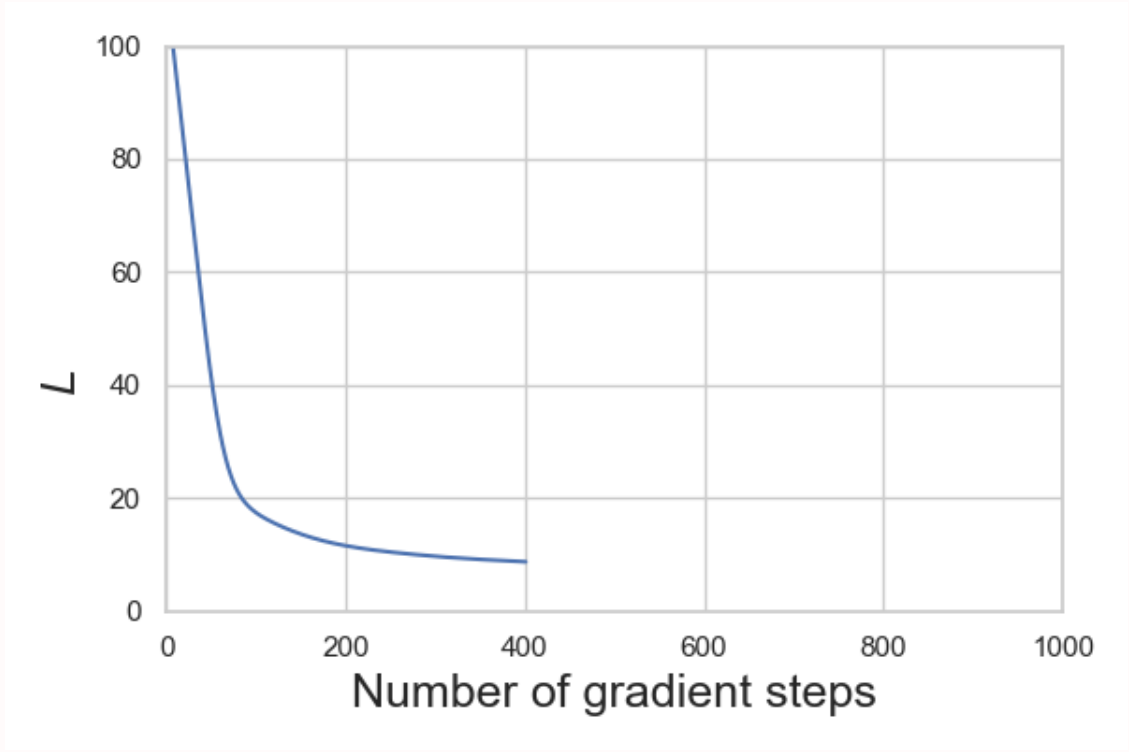


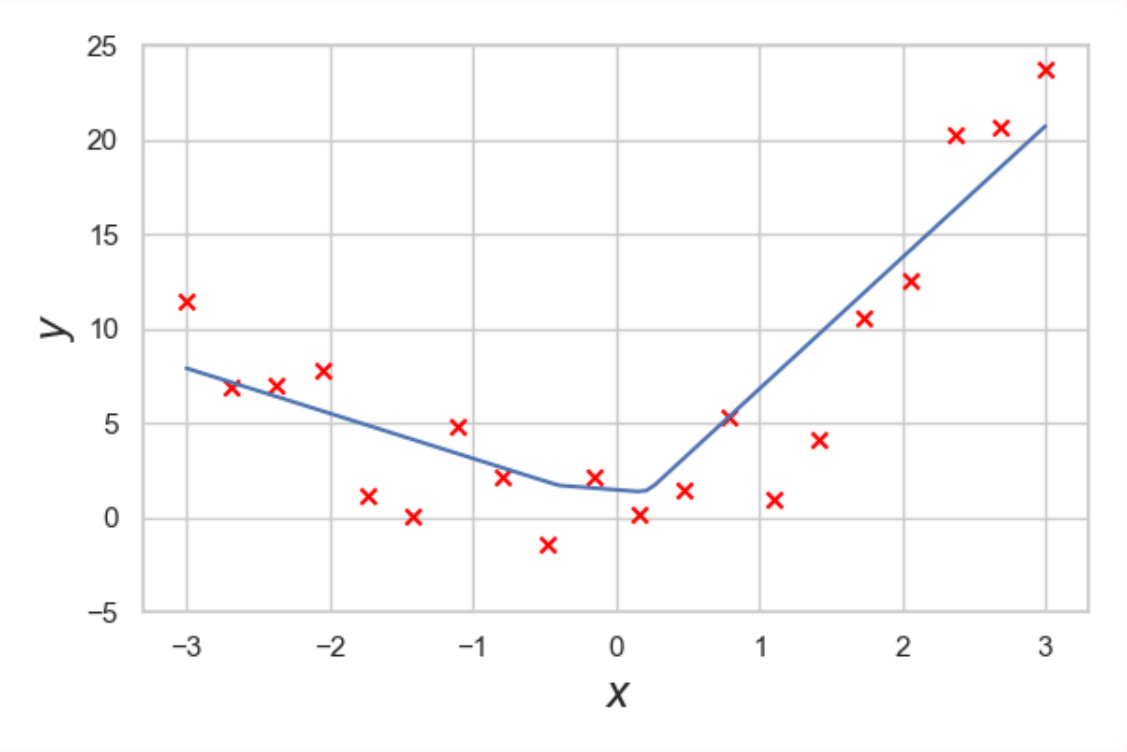
First neural network



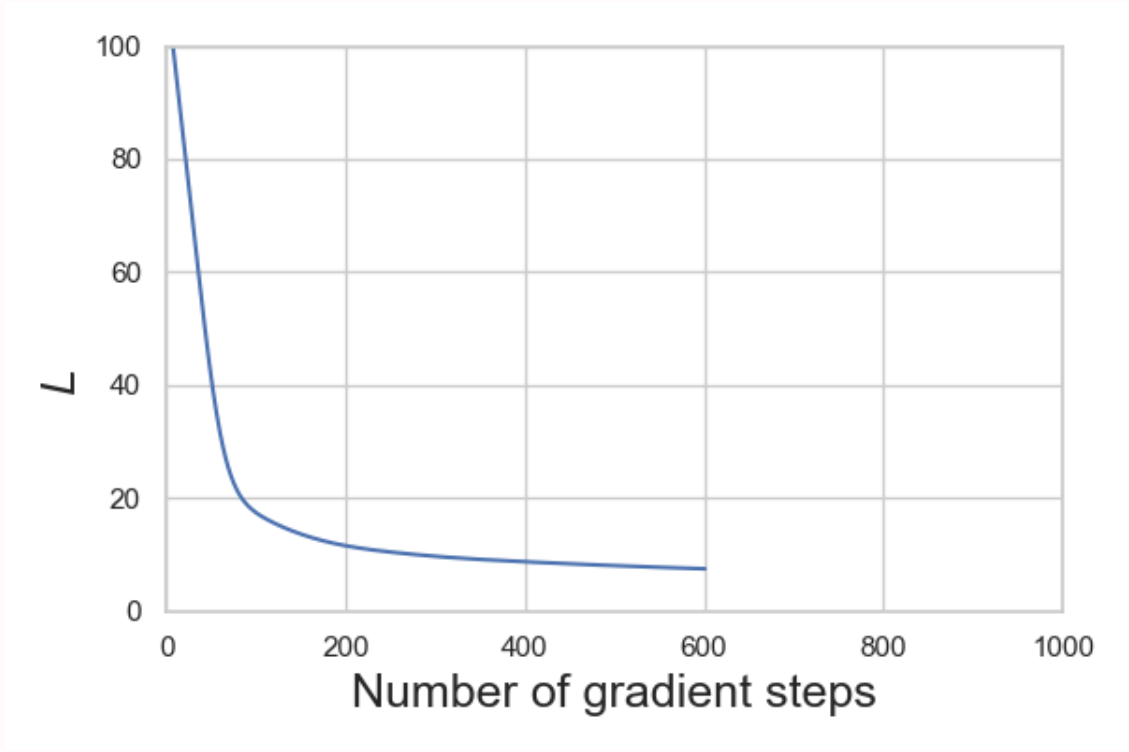


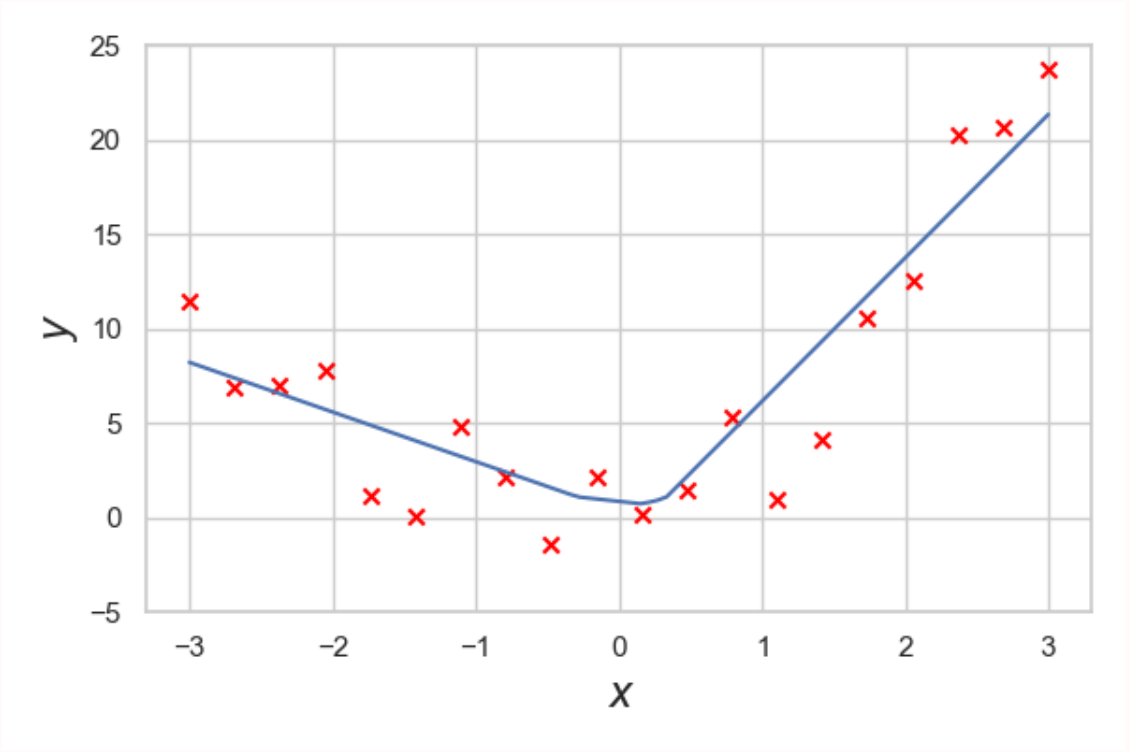
First neural network



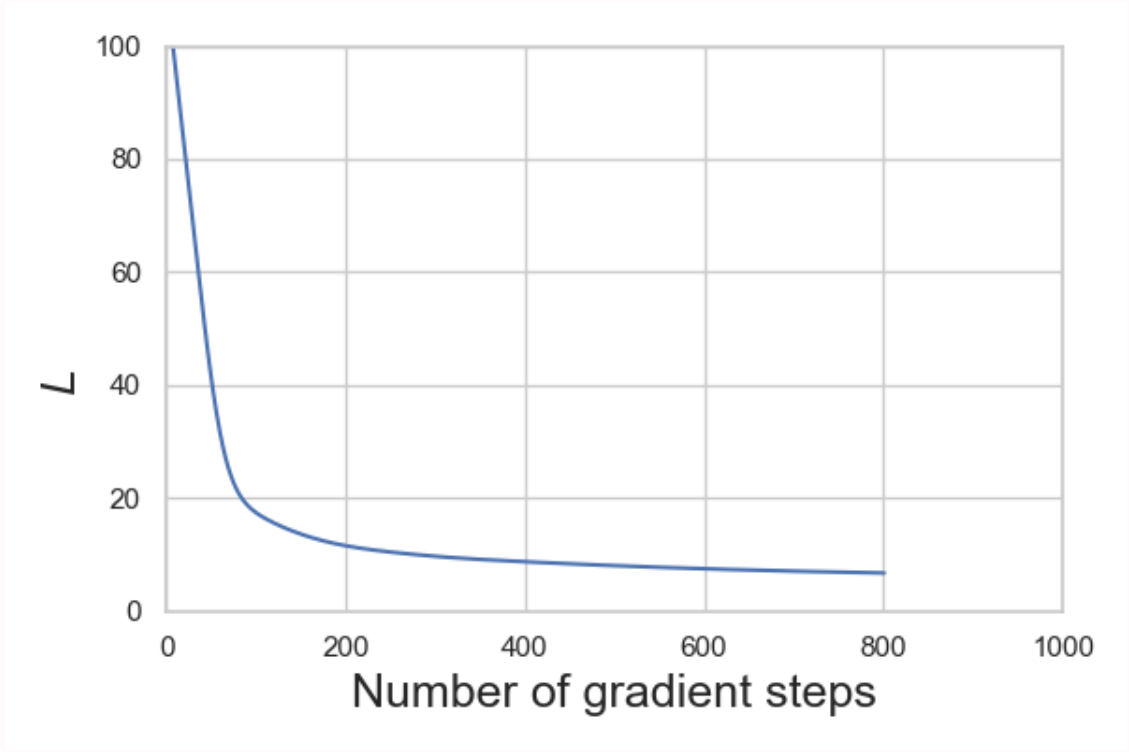


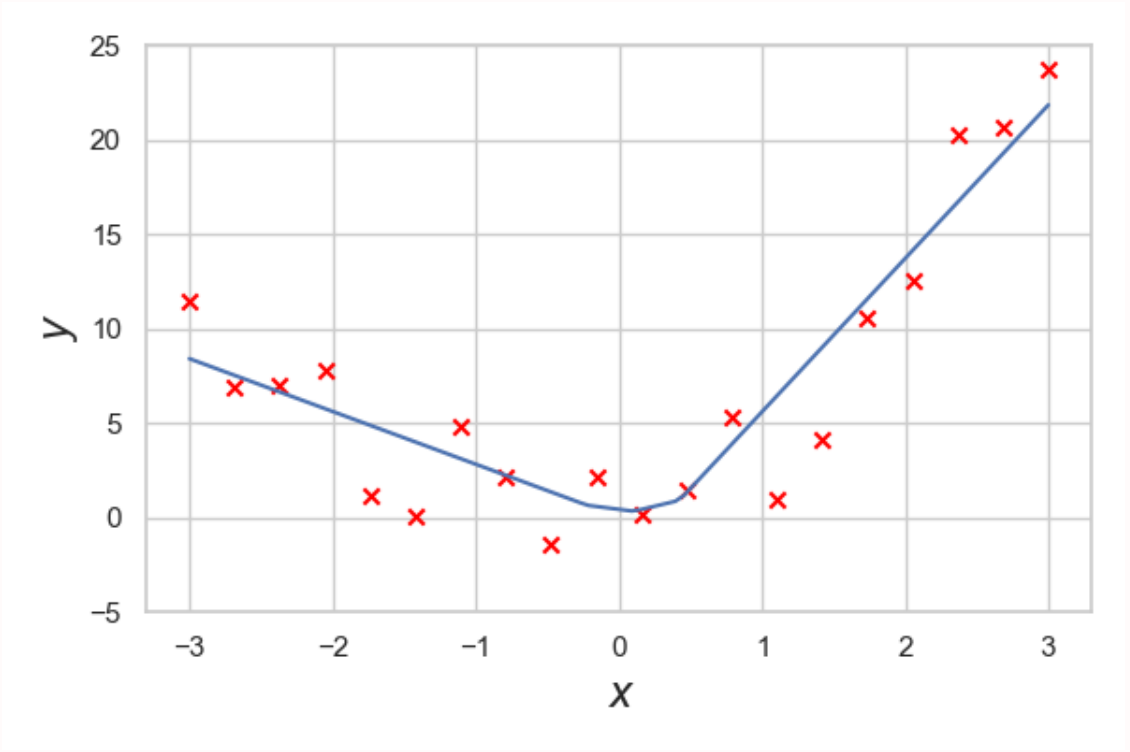
First neural network



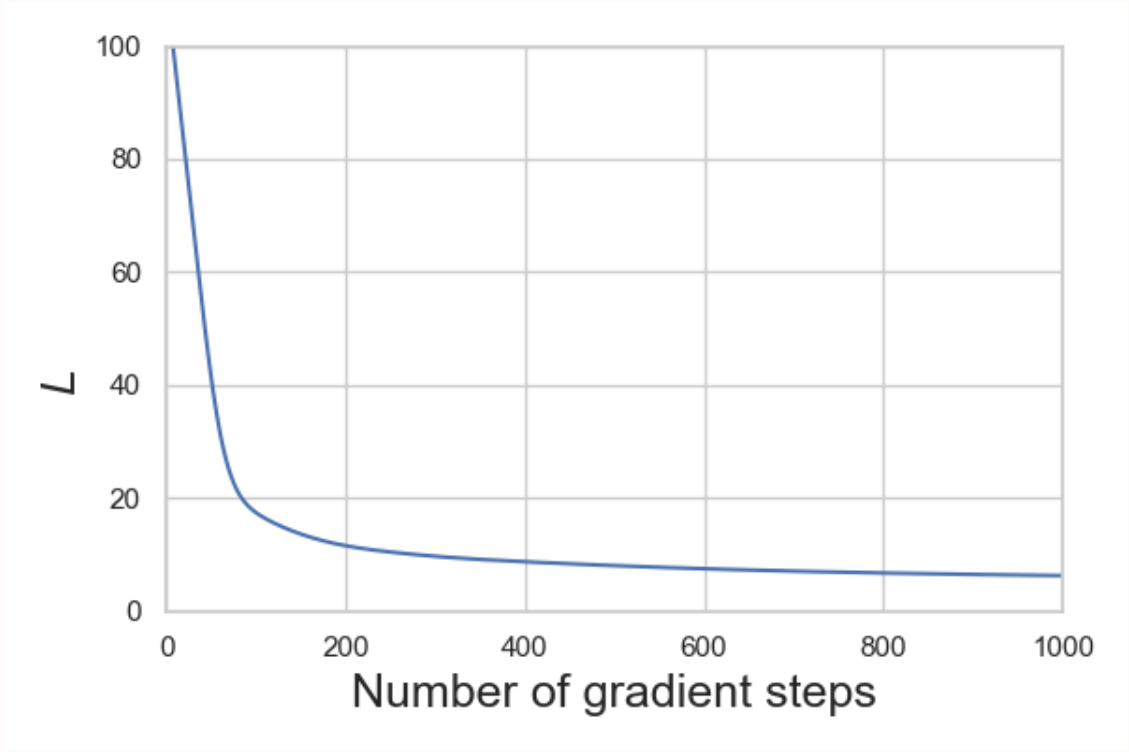


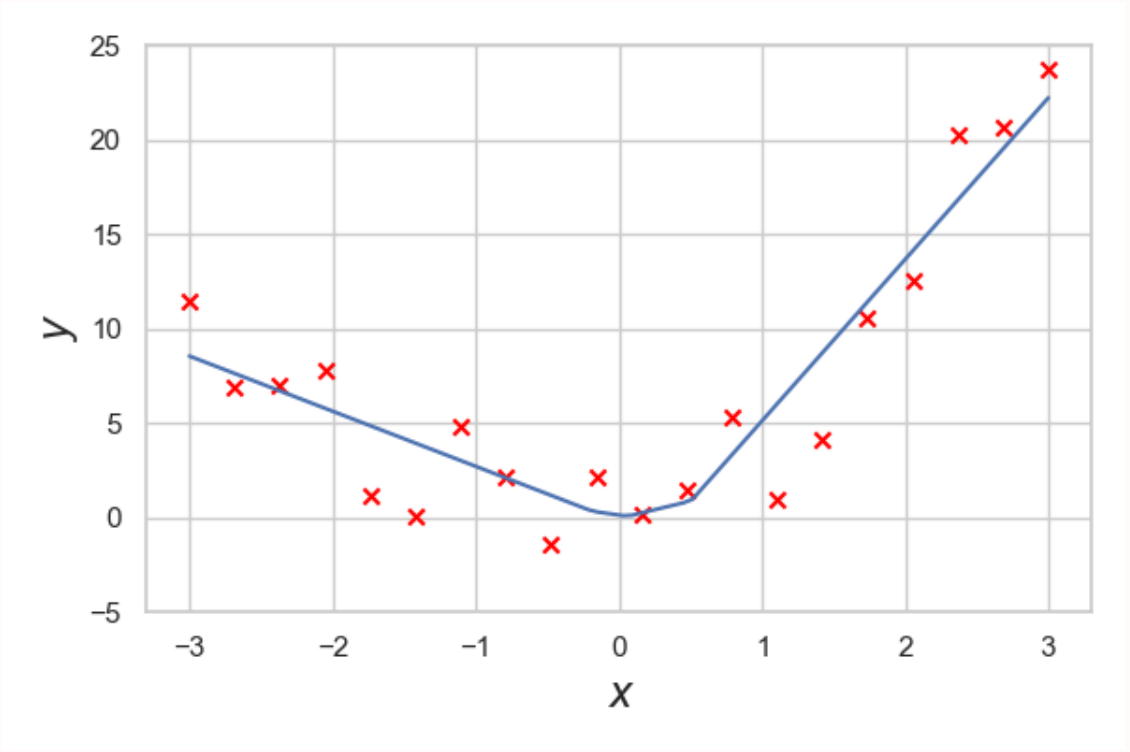
First neural network





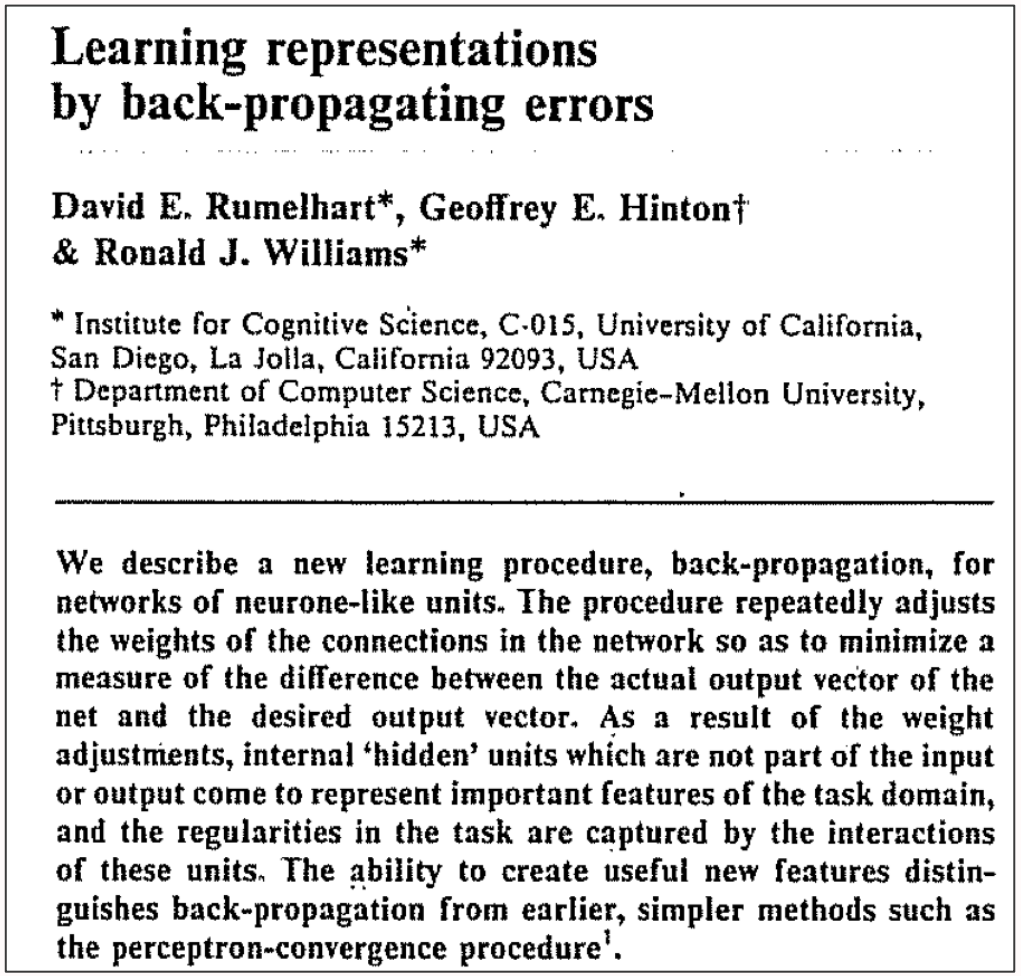
First neural network





- Our first good fit! (loss ≈10 vs ≈15/25/40).

Some history



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

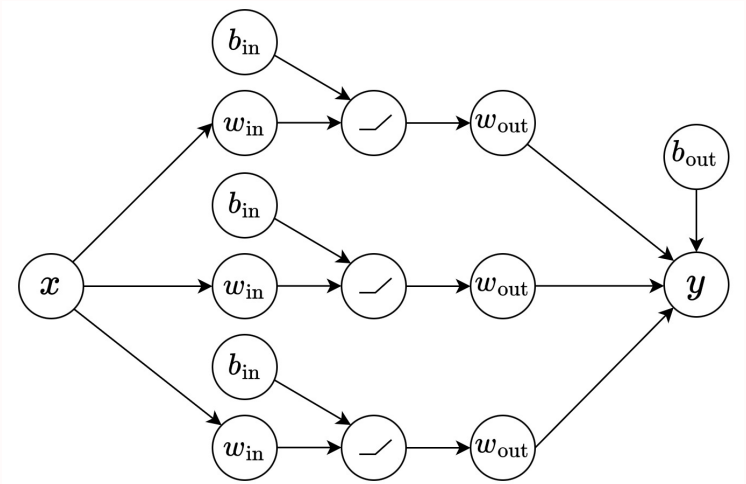
Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

Scaling up

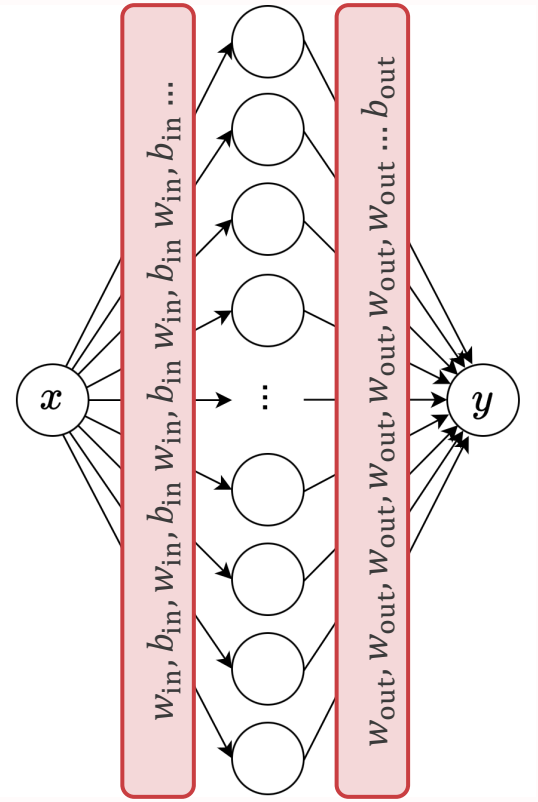
Adding even more neurons

Scaling up

- If one neuron is bad and three is neurons good...
- Maybe more neurons is better!

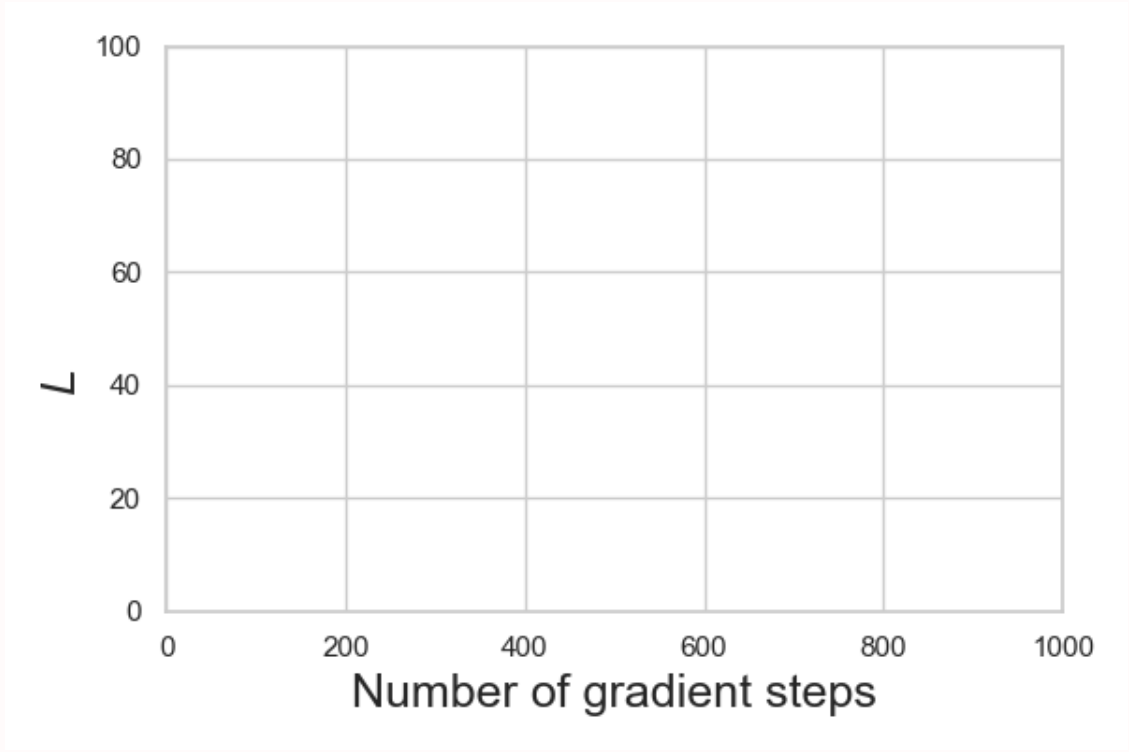


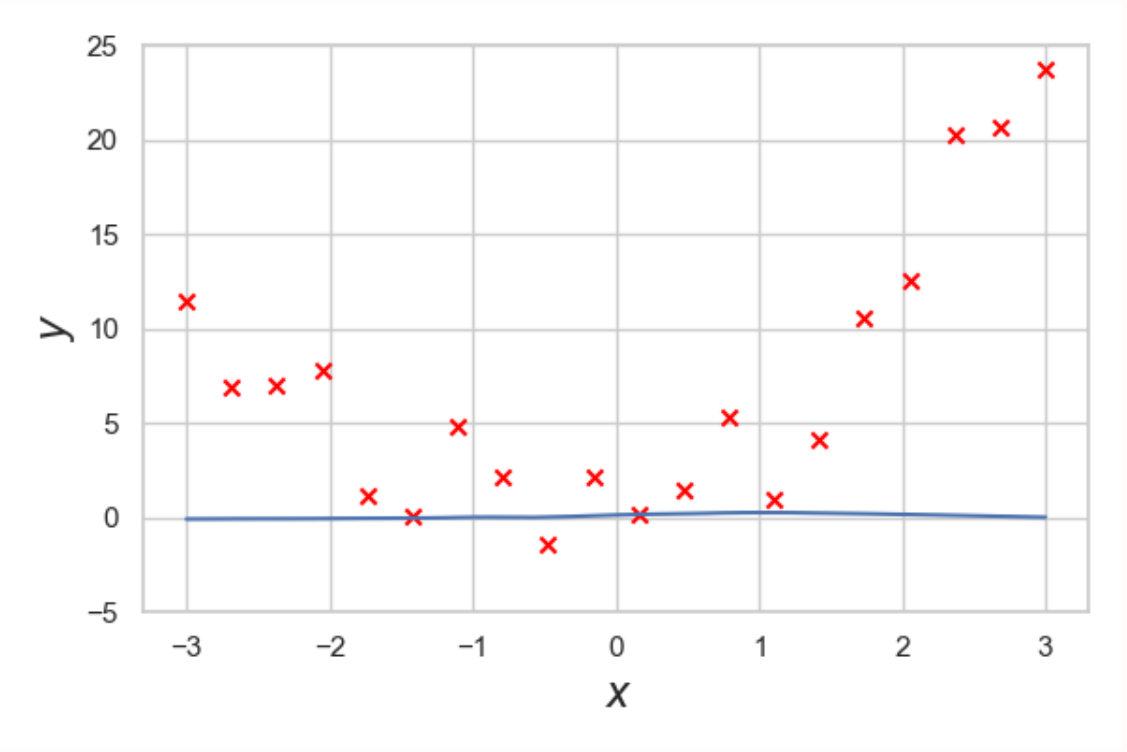
Scale up →



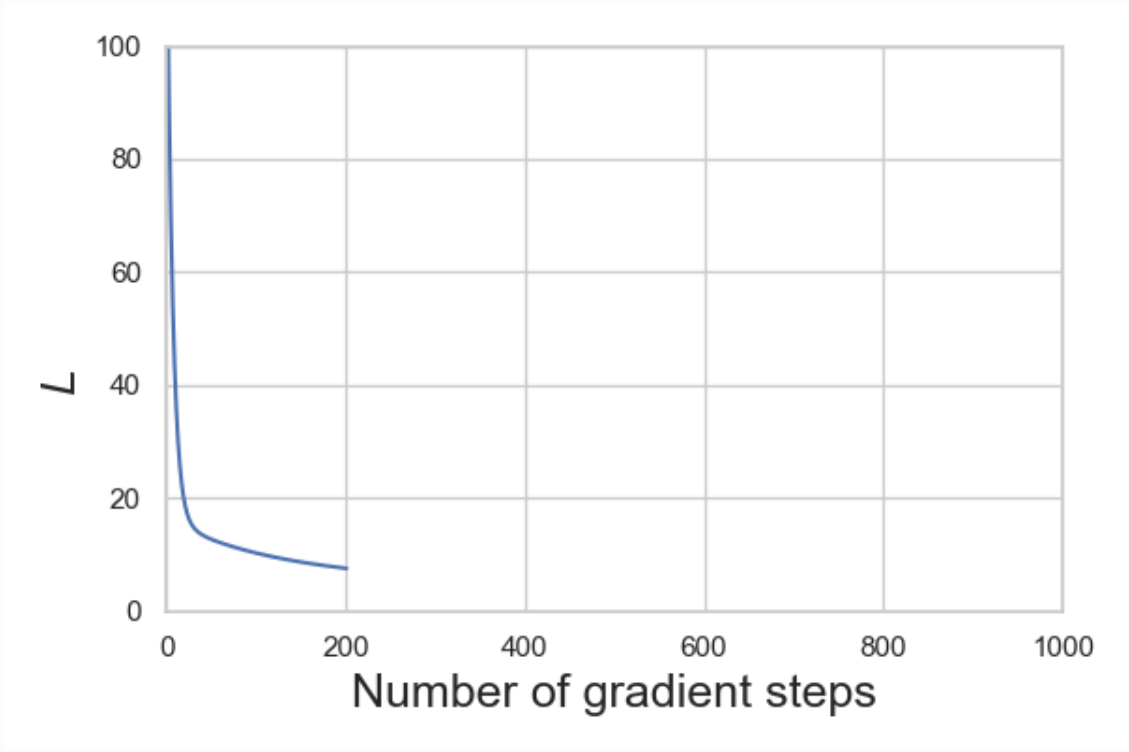
- From now on, won't draw w, b, but they're still there.
- *What will happen if we try to fit the dataset with a 100-neuron network?*

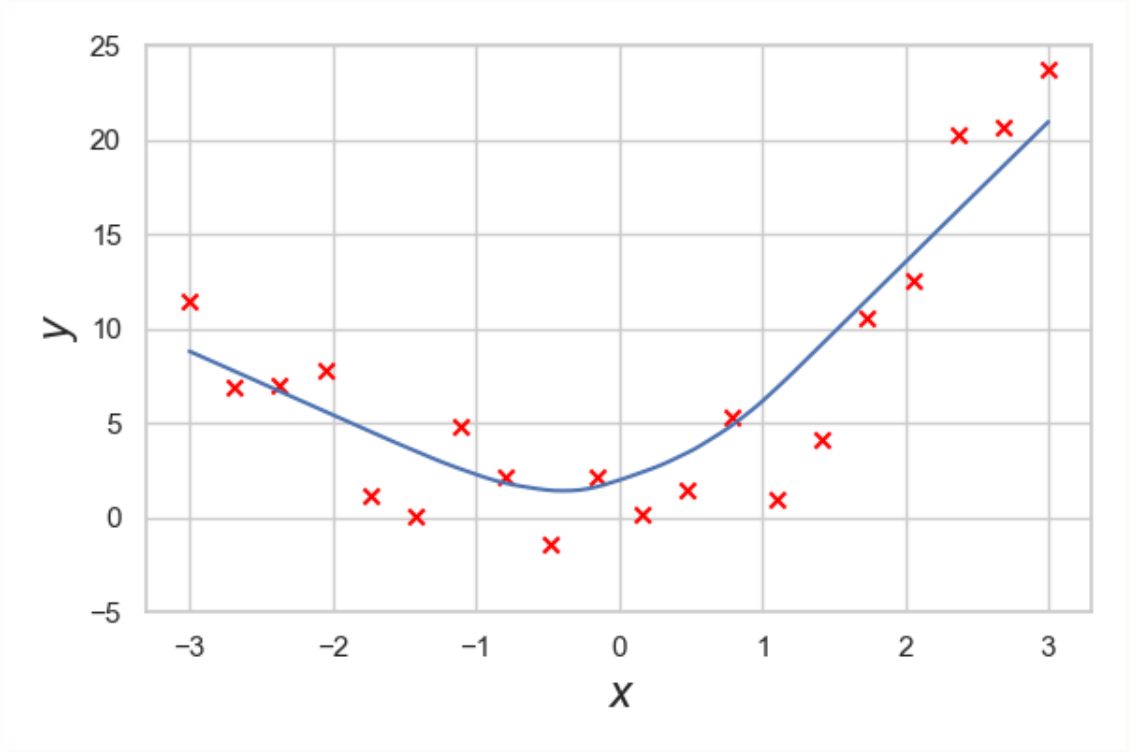
Fitting a 100-neuron network



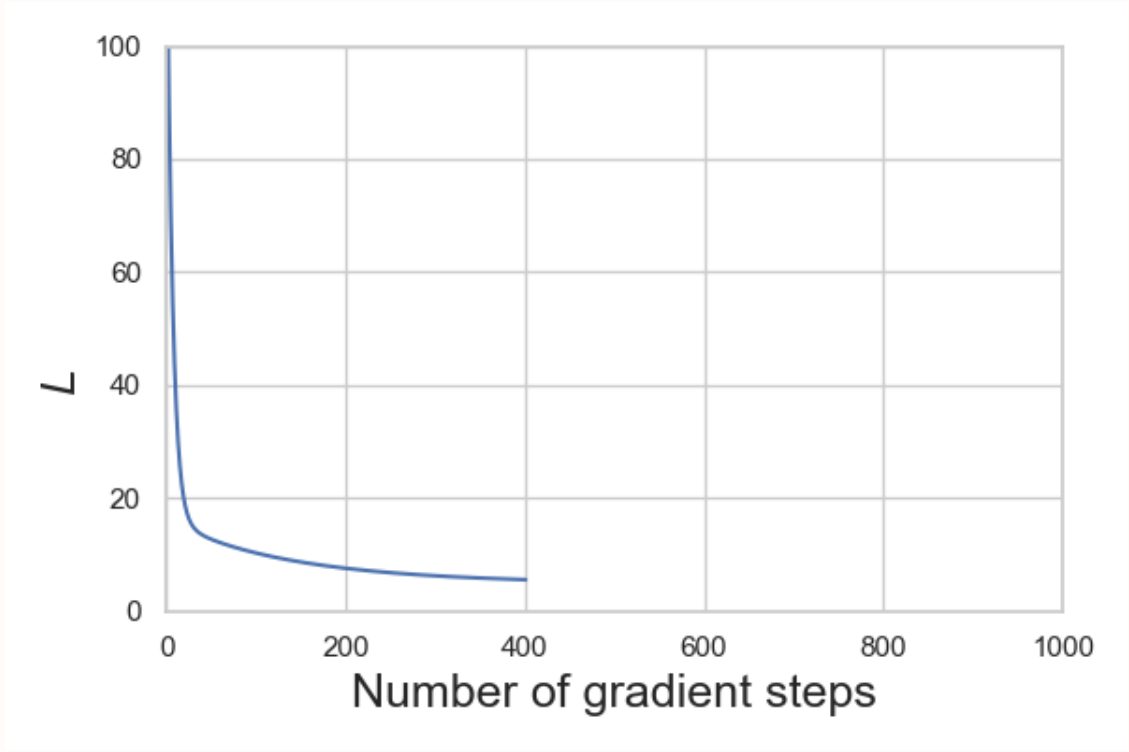


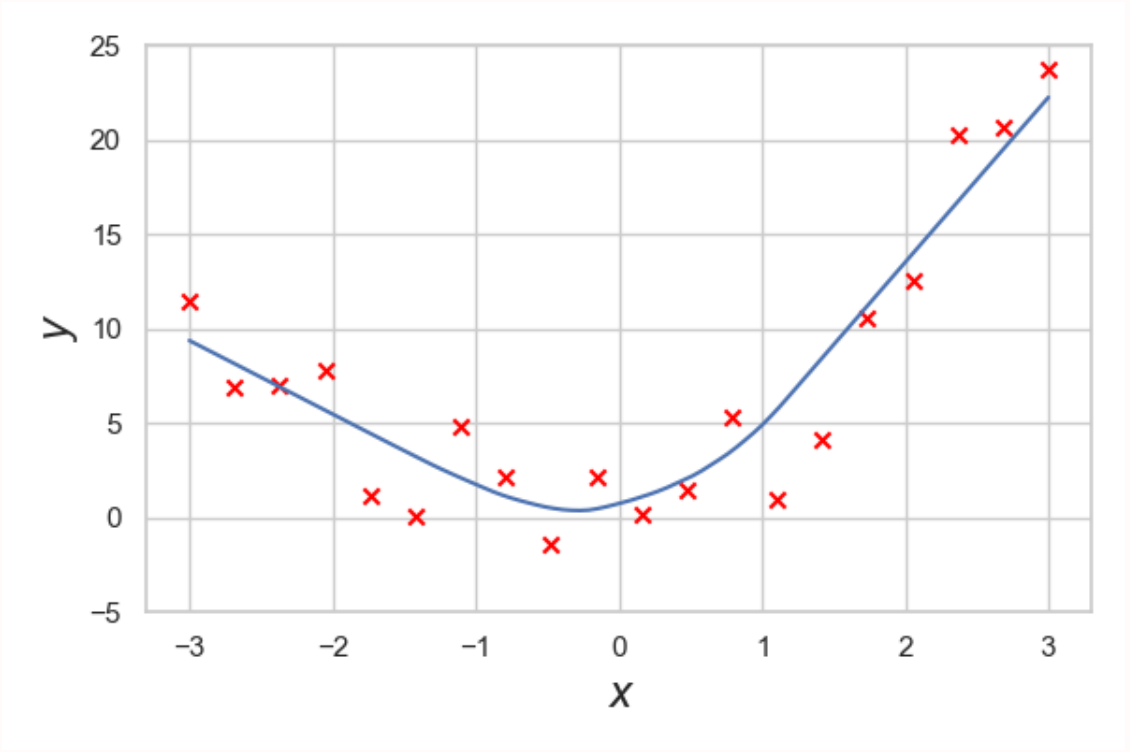
Fitting a 100-neuron network



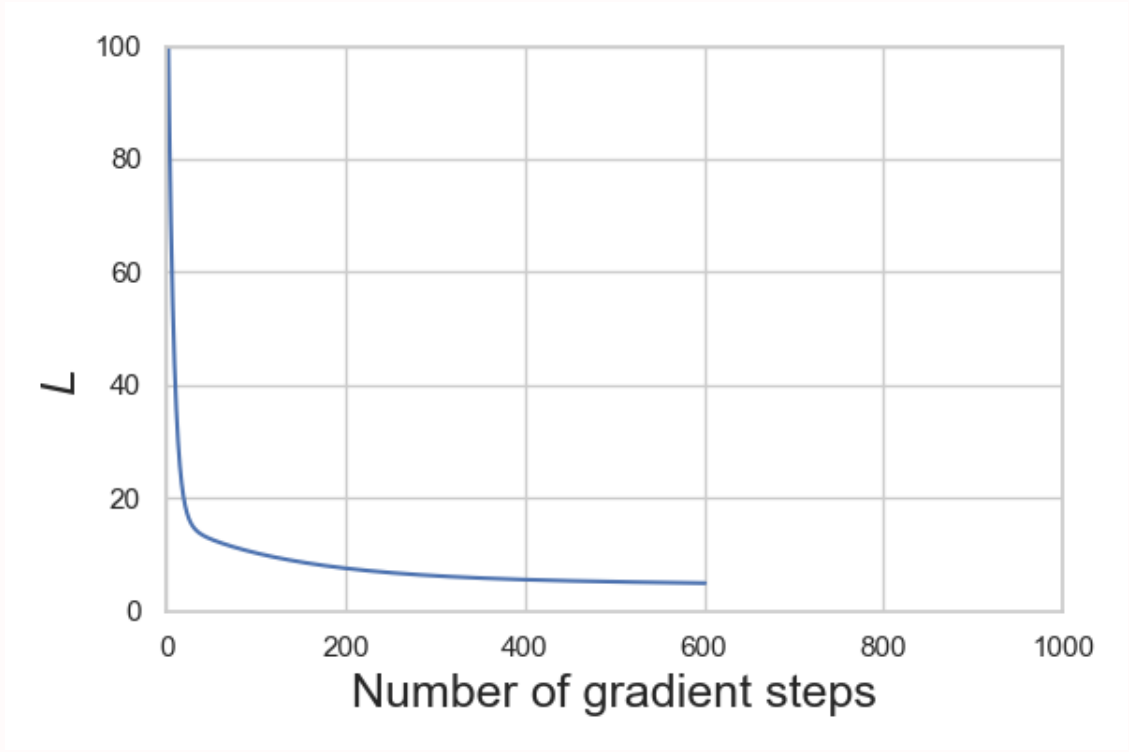


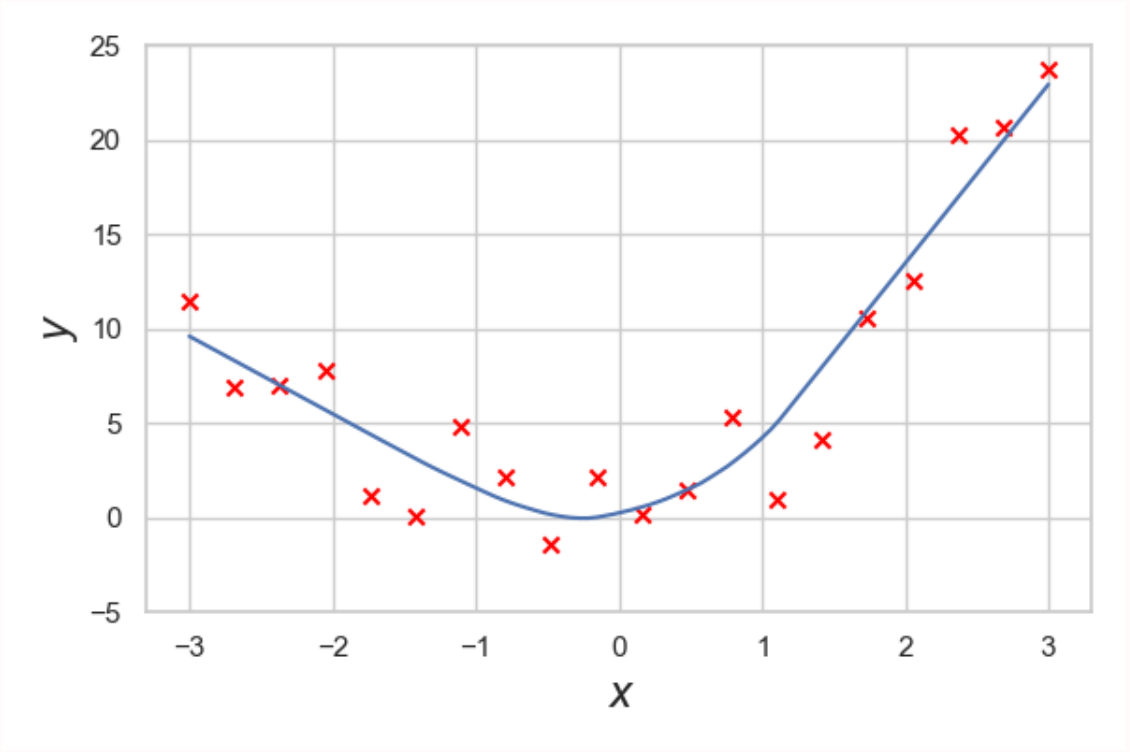
Fitting a 100-neuron network



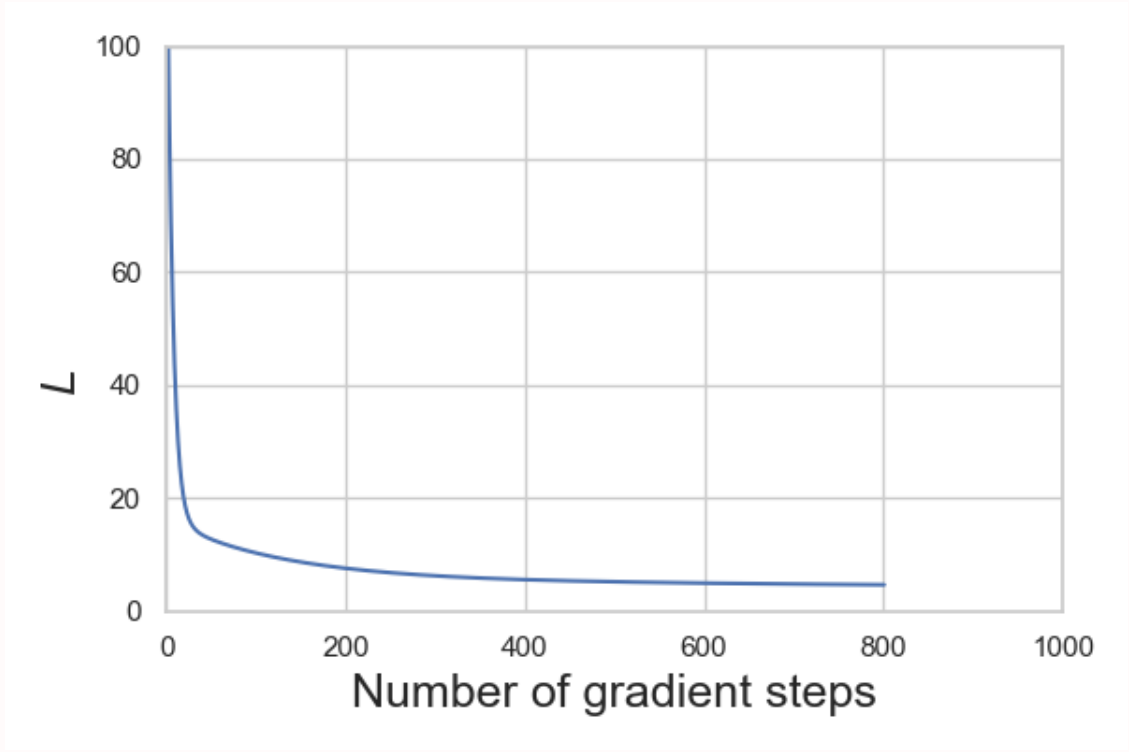


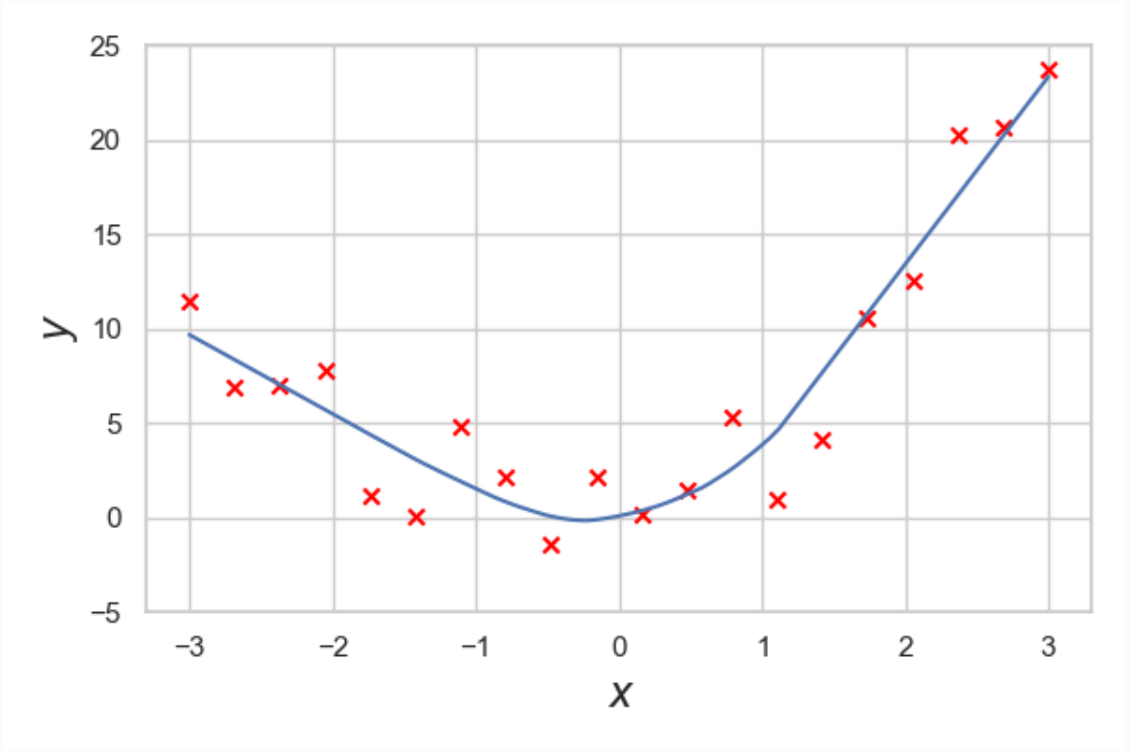
Fitting a 100-neuron network



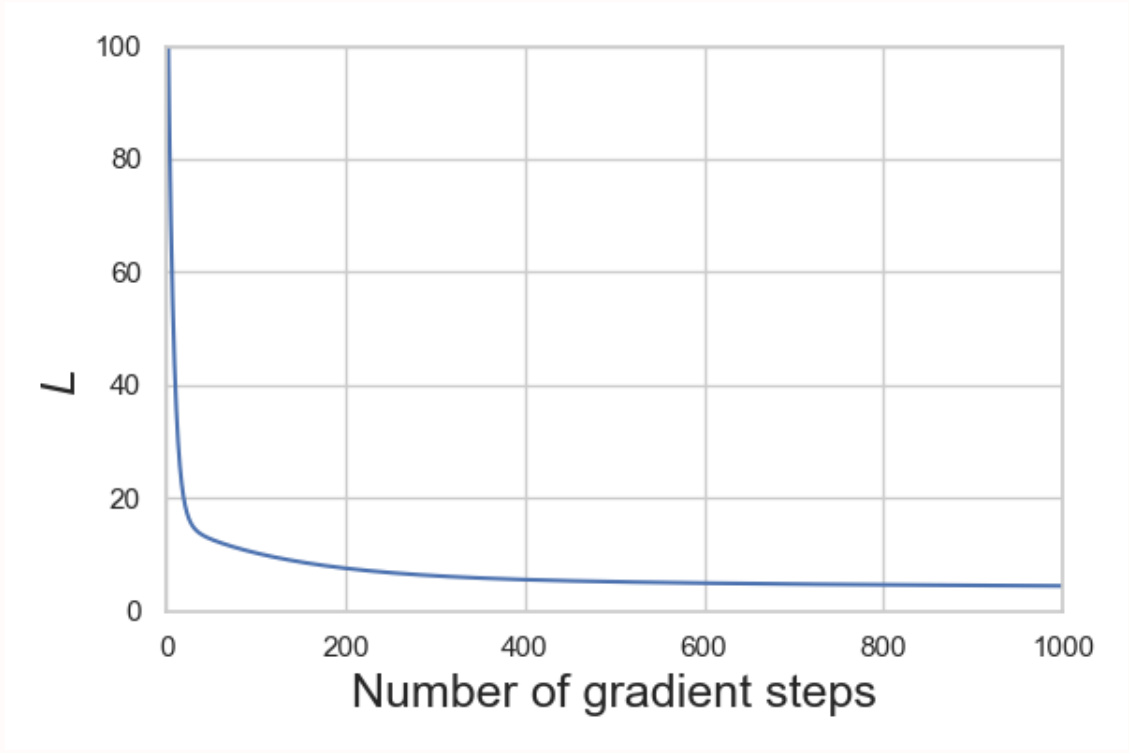


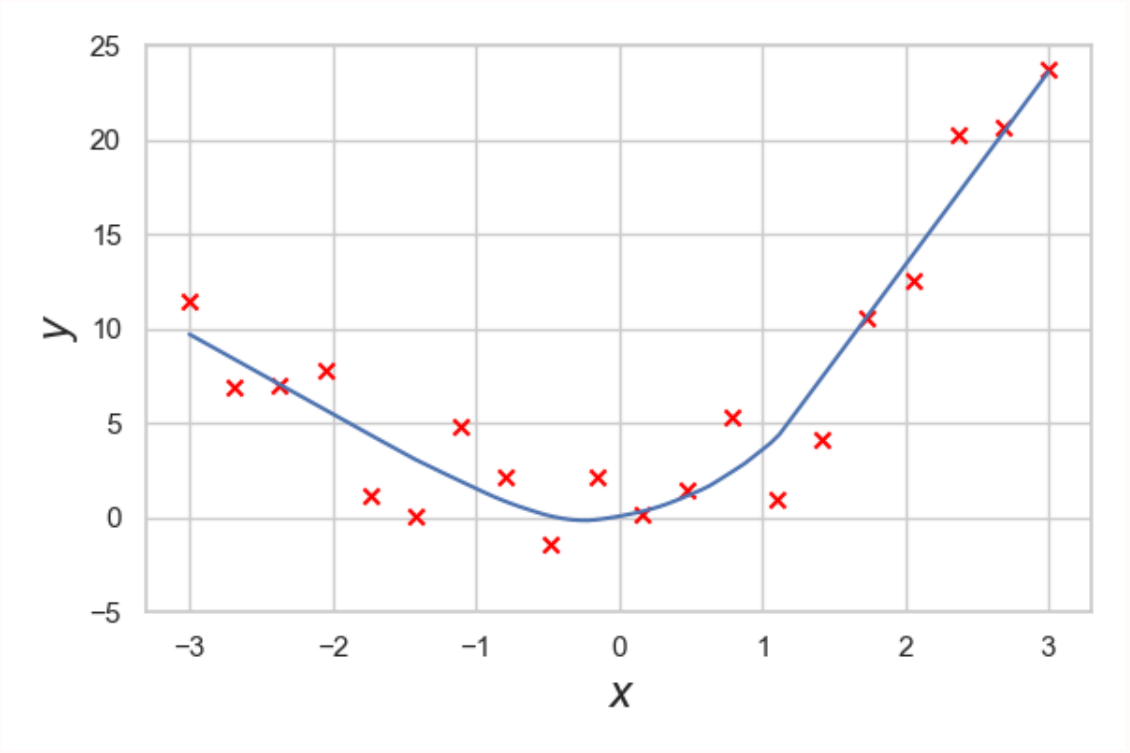
Fitting a 100-neuron network





Fitting a 100-neuron network





- An excellent fit! We've successfully trained a 100-neuron neural network.
- *Many of the breakthroughs of the last decade have come from scaling up neural networks.*

Question & Answer