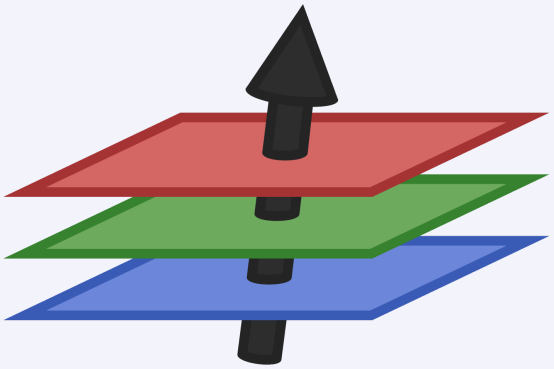
Machine learning in Julia with Flux



Marie-Hélène Burle

training@westgrid.ca

May 13, 2020

Paradigm of the dominant current approach to AI:

Feeding vast amounts of data to algorithms
It is a **data driven** approach
It is about **learning**
i.e. strengthening **pathways**

Example in image recognition:

Rather than coding all the possible ways—pixel by pixel—that a picture can represent an object, examples of image/label pairs are fed to a neural network

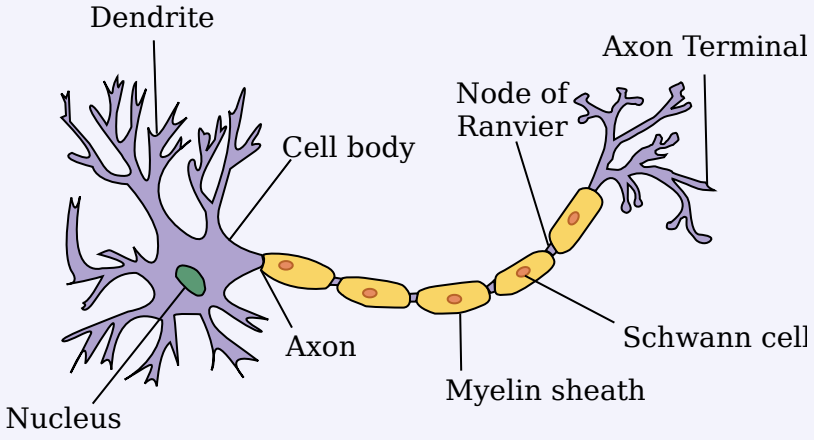
Machine learning

Machine learning

Computer programs whose performance at a task improves with experience

Neural networks

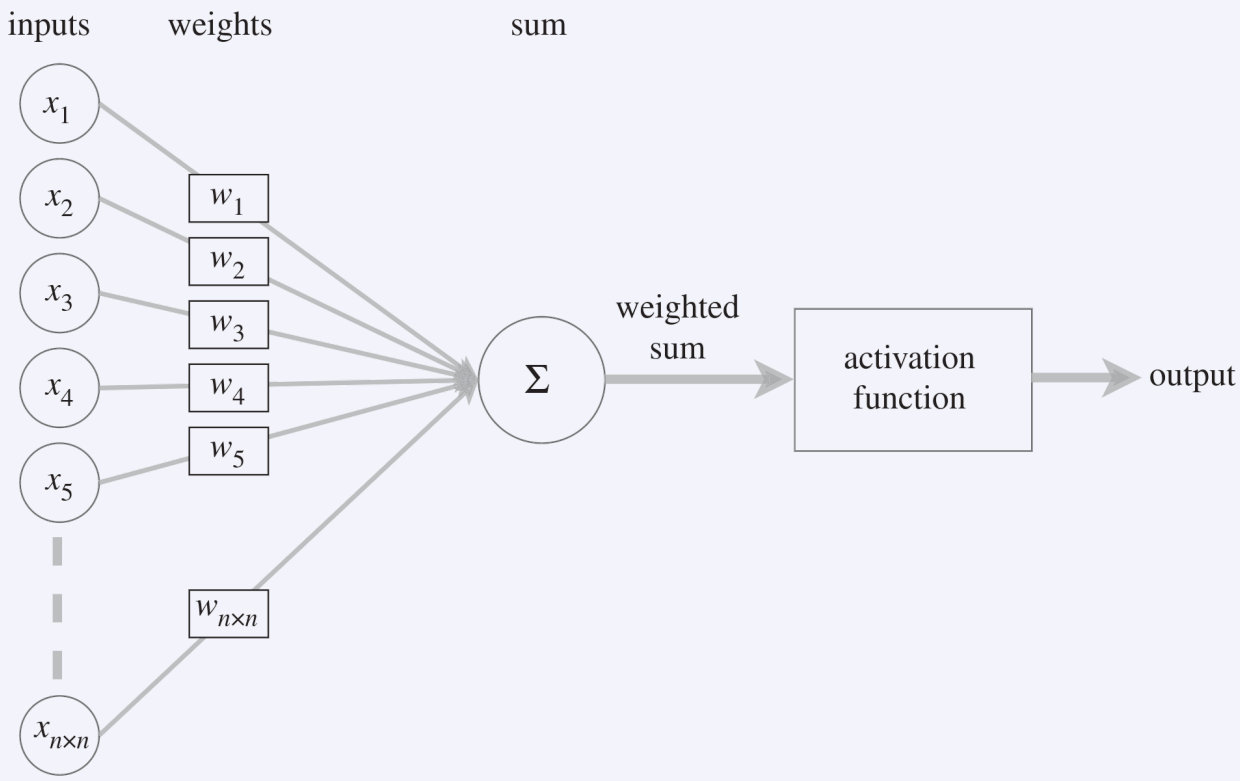
Biological neuron



Schematic from Dhp1080, Wikipedia

Electrically excitable cell receiving information through dendrites & transmitting the compiled output through the axon

Artificial neuron



Modified from O.C. Akgun & J. Mei 2019

Analogous design

The weighted sum of a set of numeric inputs is passed through an activation function before yielding a numeric output

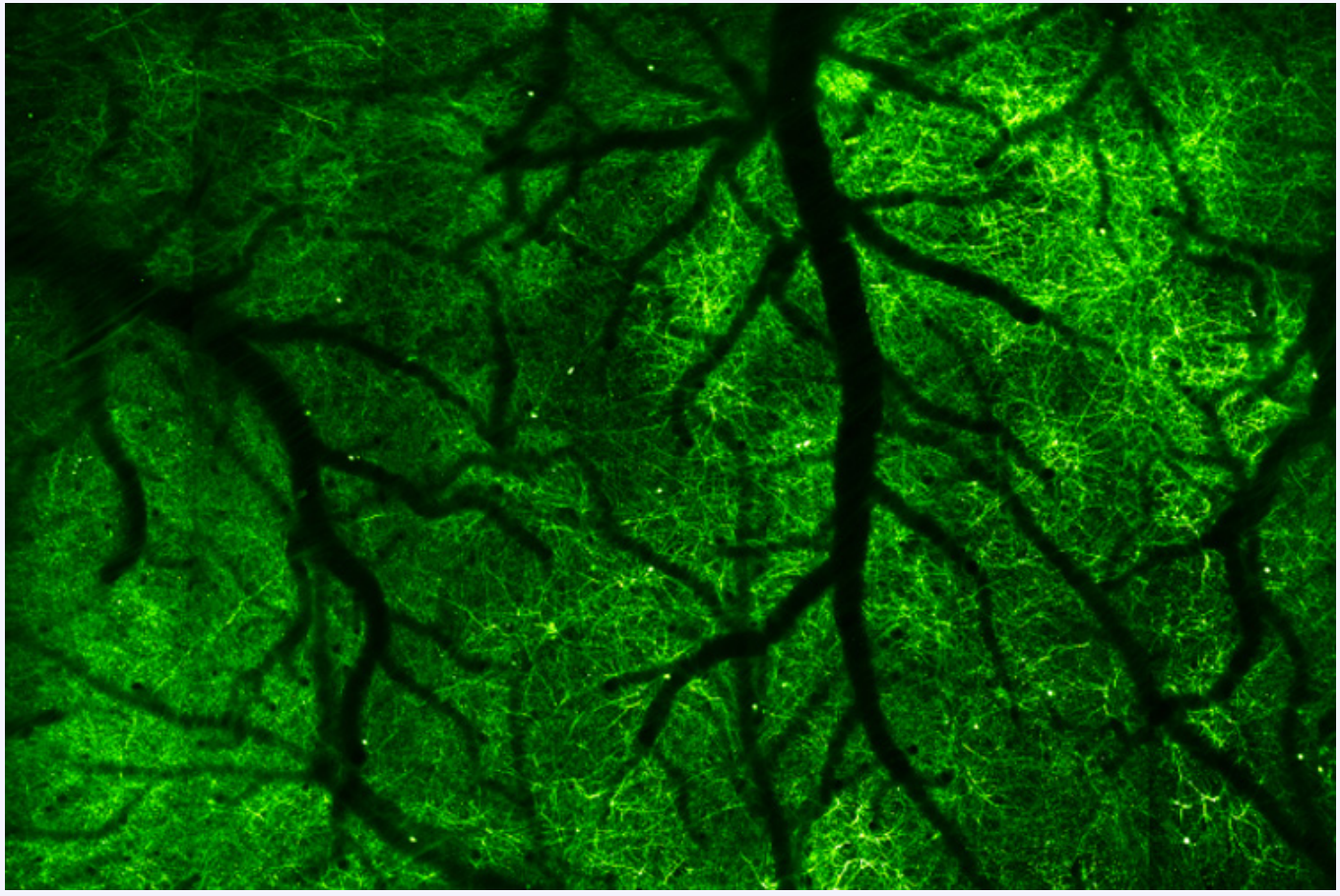
More realistic biological neuron

a: axon, c: dendrites

Drawing from stained neuron from Gray's Anatomy

The actual level of branching is vastly superior

Biological neural network

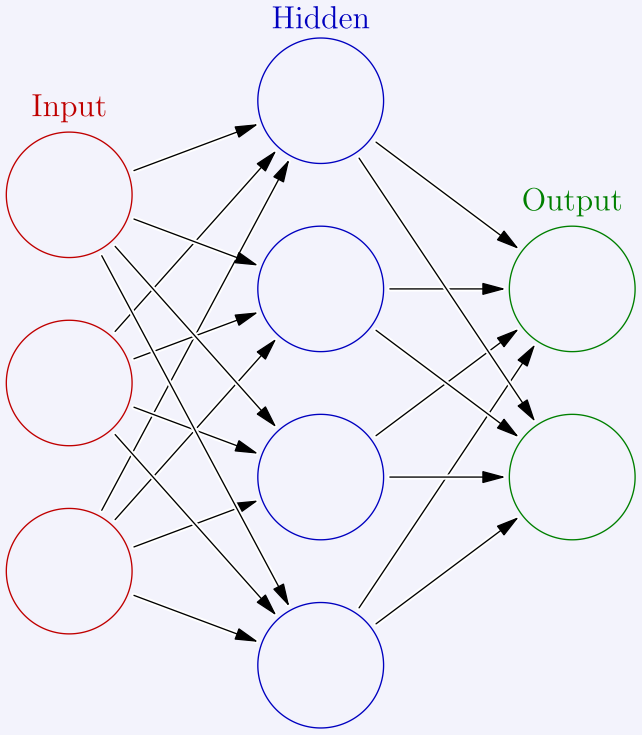


Neurons (in green) in mouse cortex; the dark branches are blood vessels

Image by Na Ji, UC Berkeley

The human brain has 65–90 billion of them
and it is a system with emergent properties

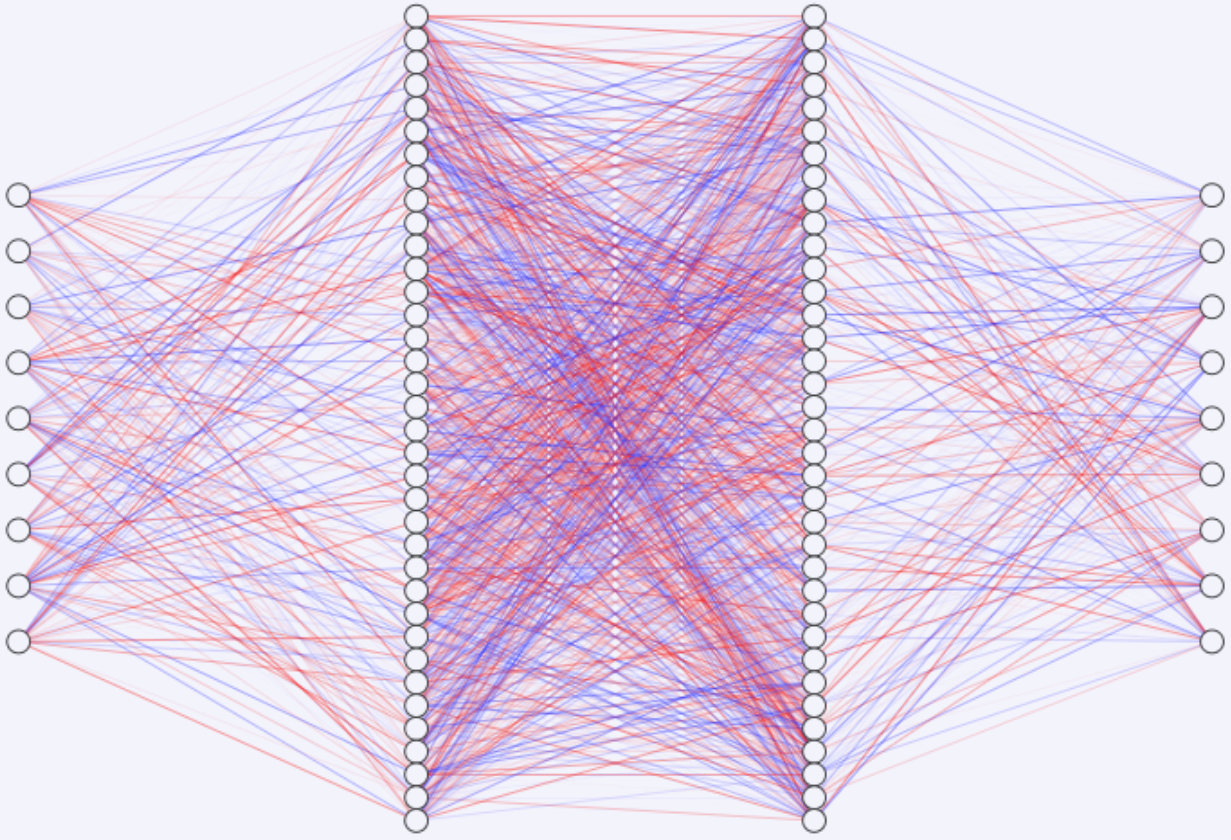
Artificial neural network (ANN)



From Glosser.ca, Wikipedia

Fully-connected, **single-layer**, **feedforward** neural network

Artificial neural network (ANN)

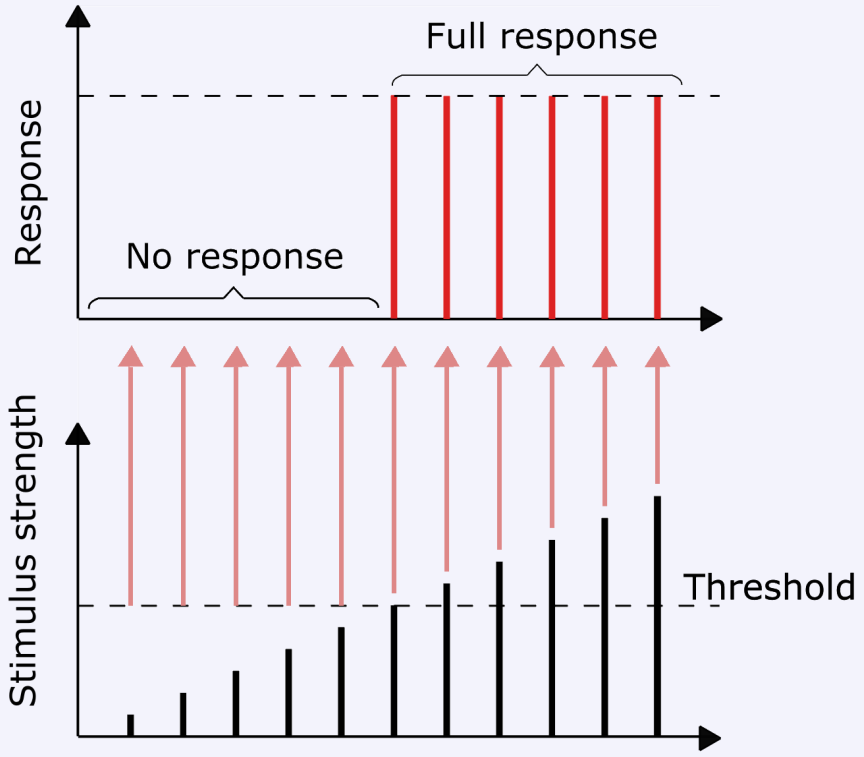


Neural network with 2 hidden layers

From The Maverick Meerkat

Fully-connected, **feedforward**, **deep** neural network

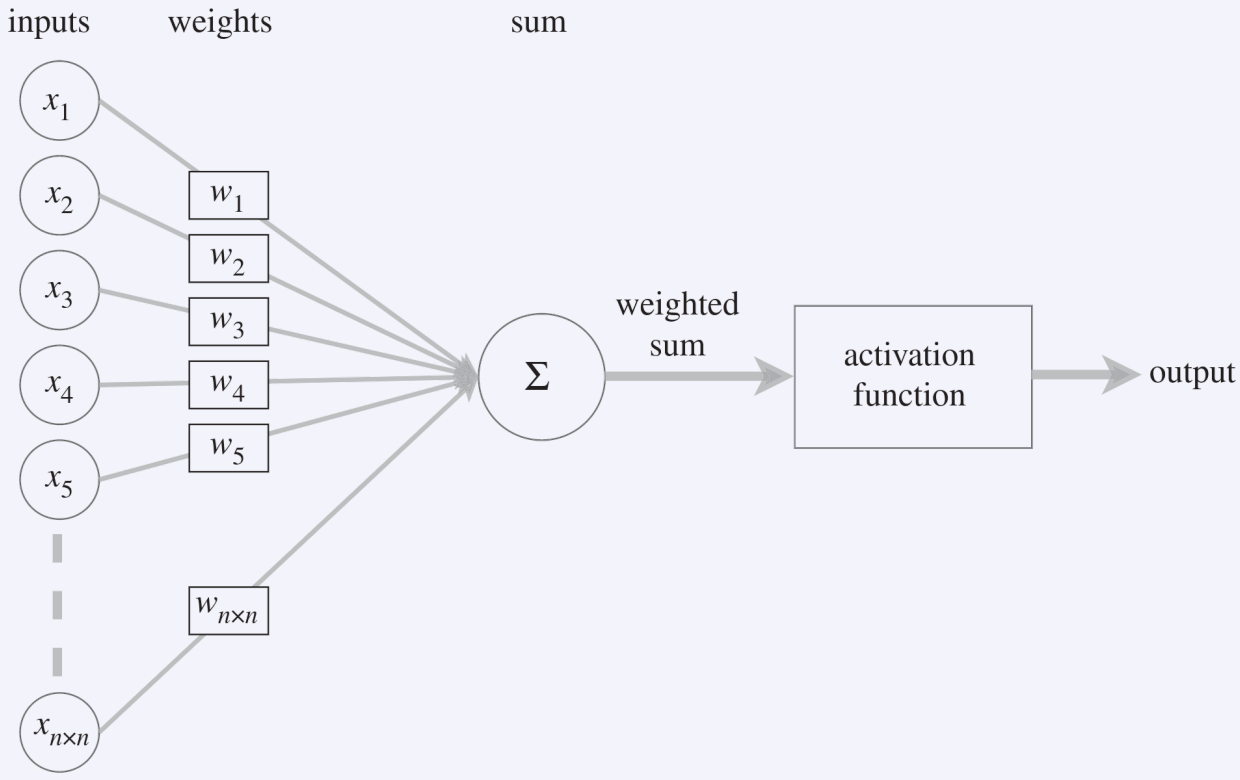
Threshold potential



Modified from Blacktc, Wikimedia

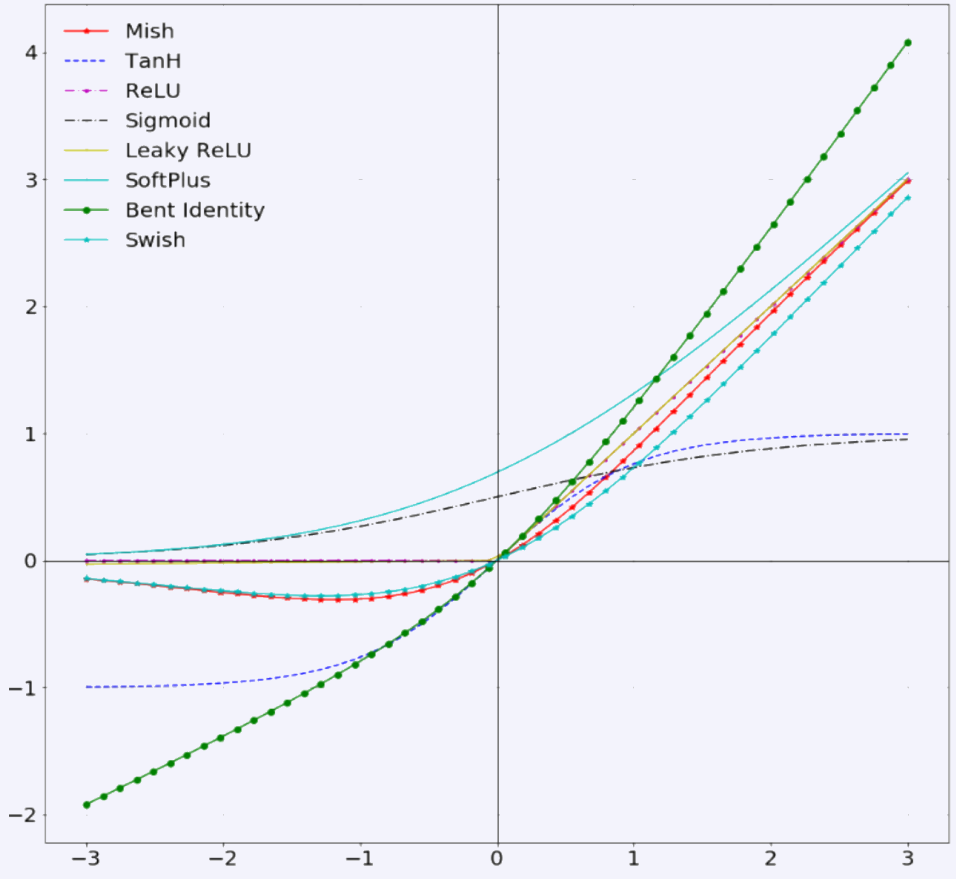
The information is an all-or-nothing electrochemical pulse or action potential
Greater stimuli don't produce stronger signals but increase firing frequency

Artificial neuron



Modified from O.C. Akgun & J. Mei 2019

Activation functions

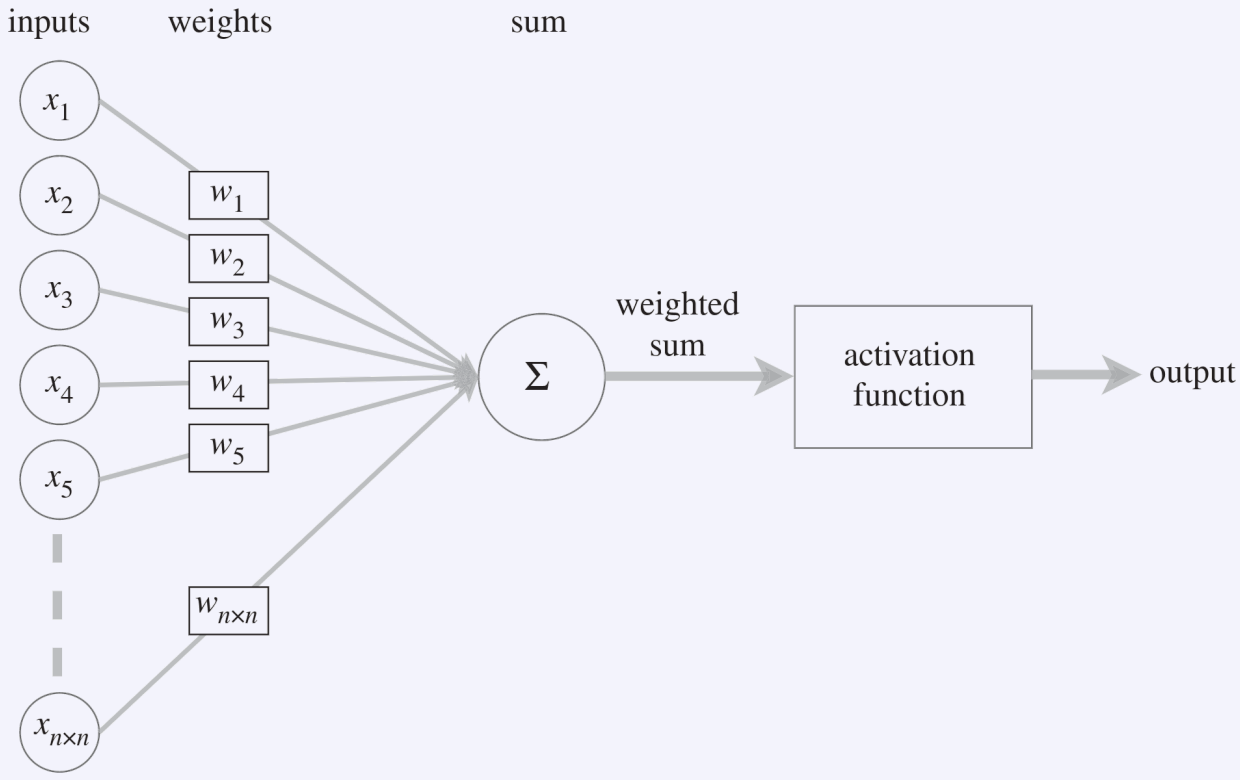


From Diganta Misra 2019

Many activation functions have been tried

Choices based on problem & available computing budget

Artificial neuron



Modified from O.C. Akgun & J. Mei 2019

Bias

Allows to shift the output of the activation function to the right or to the left
→ Offset

How do neural networks learn?

How do neural networks learn?

They adjust the weights and biases in an iterative manner

Supervised learning

Training set of example input/output (x_i, y_i) pairs

Goal:

If X is the space of inputs and Y the space of outputs, find a function h so that for each $x_i \in X$, $h_\theta(x_i)$ is a predictor for the corresponding value y_i
(θ represents the set of parameters of h_θ)

→ i.e. find the relationship between inputs and outputs

Examples:

Continuous outputs: **Regression**
Discrete outputs: **Classification**

Supervised learning

Training set of example input/output (x_i, y_i) pairs

Goal:

If X is the space of inputs and Y the space of outputs, find a function h so that for each $x_i \in X$, $h_\theta(x_i)$ is a predictor for the corresponding value y_i
(θ represents the set of parameters of h_θ)

→ i.e. find the relationship between inputs and outputs

Examples:

Continuous outputs: **Regression**
Discrete outputs: **Classification**

Supervised learning

Training set of example input/output (x_i, y_i) pairs

Goal:

If X is the space of inputs and Y the space of outputs, find a function h so that for each $x_i \in X$, $h_\theta(x_i)$ is a predictor for the corresponding value y_i
(θ represents the set of parameters of h_θ)

→ i.e. find the relationship between inputs and outputs

Examples:

Continuous outputs: **Regression**
Discrete outputs: **Classification**

Supervised learning

Training set of example input/output (x_i, y_i) pairs

Goal:

If X is the space of inputs and Y the space of outputs, find a function h so that for each $x_i \in X$, $h_\theta(x_i)$ is a predictor for the corresponding value y_i
(θ represents the set of parameters of h_θ)

→ i.e. find the relationship between inputs and outputs

Examples:

Continuous outputs: **Regression**
Discrete outputs: **Classification**

Supervised learning

Training set of example input/output (x_i, y_i) pairs

Goal:

If X is the space of inputs and Y the space of outputs, find a function h so that for each $x_i \in X$, $h_\theta(x_i)$ is a predictor for the corresponding value y_i
(θ represents the set of parameters of h_θ)

→ i.e. find the relationship between inputs and outputs

Examples:

Continuous outputs: **Regression**
Discrete outputs: **Classification**

Unsupervised learning

Unlabelled data (training set of x_i)

Goal:

Look for structure within the data

Examples:

Clustering
Social network analysis
Market segmentation
PCA
Cocktail party algorithm (signal separation)

Unsupervised learning

Unlabelled data (training set of x_i)

Goal:

Look for structure within the data

Examples:

Clustering
Social network analysis
Market segmentation
PCA
Cocktail party algorithm (signal separation)

Unsupervised learning

Unlabelled data (training set of x_i)

Goal:

Look for structure within the data

Examples:

Clustering
Social network analysis
Market segmentation
PCA
Cocktail party algorithm (signal separation)

Unsupervised learning

Unlabelled data (training set of x_i)

Goal:

Look for structure within the data

Examples:

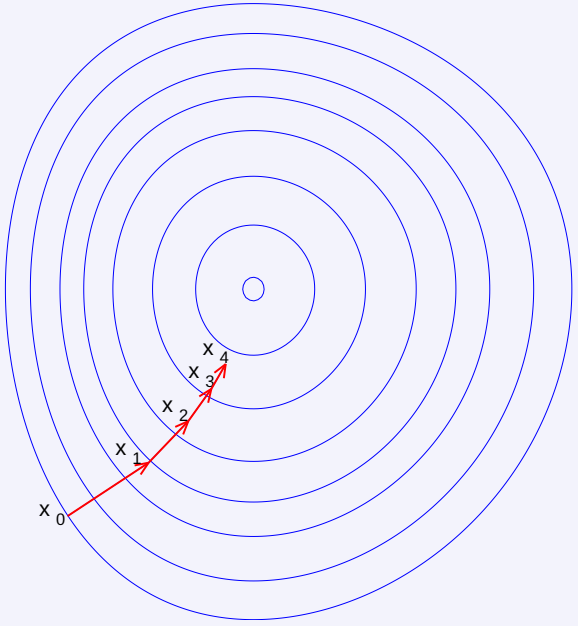
Clustering
Social network analysis
Market segmentation
PCA
Cocktail party algorithm (signal separation)

Cross entropy loss function

Cross entropy:
Distance between predicted and real distribution

ML Objective:
Minimizing it

Gradient descent



From Olegalexandrov & Zerodamage, Wikipedia

Iterative optimization method
Adjust the weights and biases

Batch gradient descent

Use all examples in each iteration

Slow for large data set:
Parameters adjusted only after all the samples have been through

Stochastic gradient descent

Use one example in each iteration

Much faster than batch gradient descent:
Parameters are adjusted after each example

But no vectorization

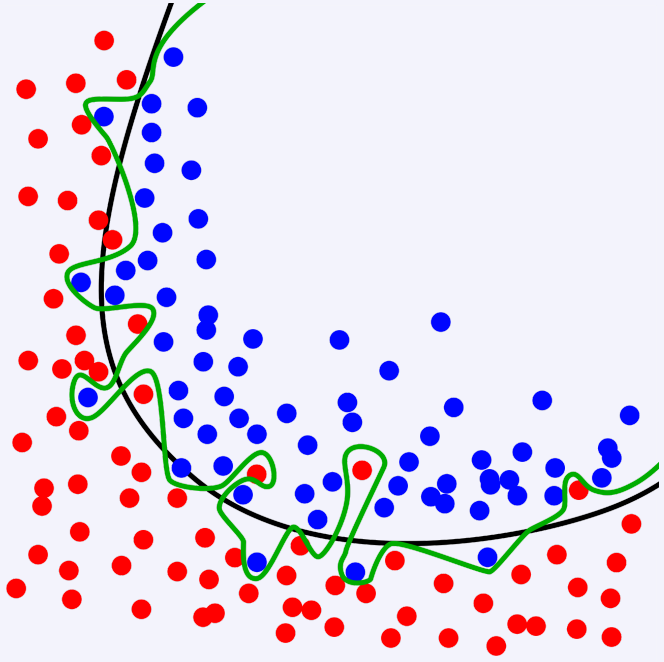
Mini-batch gradient descent

Intermediate approach:
Use mini-batch size examples in each iteration

Allows a vectorized approach that stochastic gradient descent did not allow
→ parallelization

Variation: Adam optimization algorithm

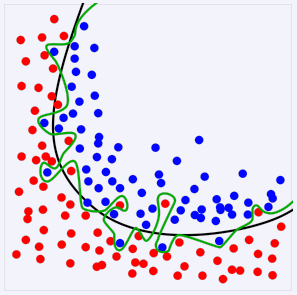
Overfitting



From Chabacano, Wikipedia

Some noise from the data extracted by the model while it does not represent general meaningful structure and has no predictive power

Overfitting: solutions



From Chabacano, Wikipedia

Regularization by adding a penalty to the loss function

Early stopping

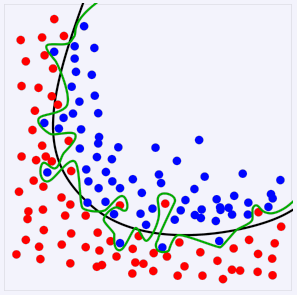
Increase depth (more layers), decrease breadth (less neurons per layer)

→ less parameters overall, but creates vanishing and exploding gradient problems

Neural architectures adapted to the type of data

→ fewer and shared parameters (e.g. convolutional neural network, recurrent neural network)

Overfitting: solutions



From Chabacano, Wikipedia

Regularization by adding a penalty to the loss function

Early stopping

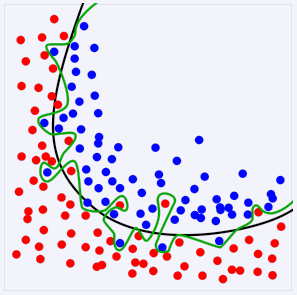
Increase depth (more layers), decrease breadth (less neurons per layer)

→ less parameters overall, but creates vanishing and exploding gradient problems

Neural architectures adapted to the type of data

→ fewer and shared parameters (e.g. convolutional neural network, recurrent neural network)

Overfitting: solutions



From Chabacano, Wikipedia

Regularization by adding a penalty to the loss function

Early stopping

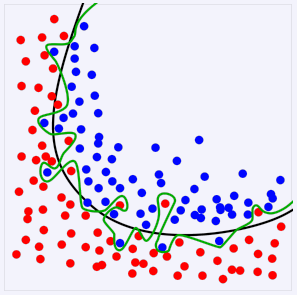
Increase depth (more layers), decrease breadth (less neurons per layer)

→ less parameters overall, but creates vanishing and exploding gradient problems

Neural architectures adapted to the type of data

→ fewer and shared parameters (e.g. convolutional neural network, recurrent neural network)

Overfitting: solutions



From Chabacano, Wikipedia

Regularization by adding a penalty to the loss function

Early stopping

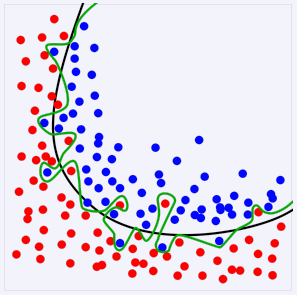
Increase depth (more layers), decrease breadth (less neurons per layer)

→ less parameters overall, but creates vanishing and exploding gradient problems

Neural architectures adapted to the type of data

→ fewer and shared parameters (e.g. convolutional neural network, recurrent neural network)

Overfitting: solutions



From Chabacano, Wikipedia

Regularization by adding a penalty to the loss function

Early stopping

Increase depth (more layers), decrease breadth (less neurons per layer)

→ less parameters overall, but creates vanishing and exploding gradient problems

Neural architectures adapted to the type of data

→ fewer and shared parameters (e.g. convolutional neural network, recurrent neural network)

Convolutional neural network (CNN)

Used for spatially structured data (e.g. image recognition)

Fully connected layer: each neuron receives input from every element of the previous layer. Images have huge input sizes and would require a very large number of neurons.

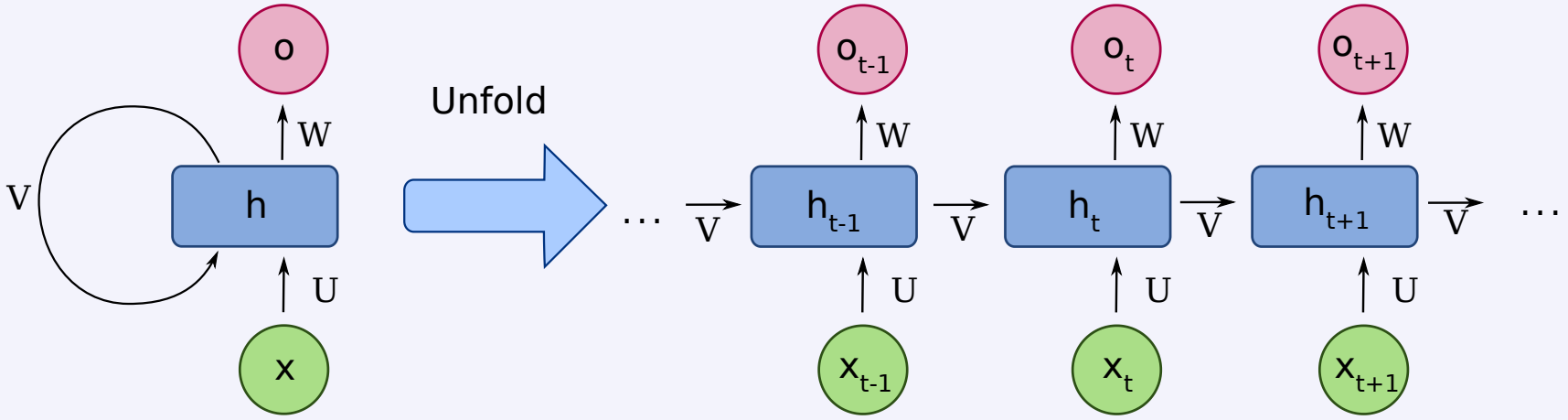
Convolutional layer: neurons receive input from a subarea (local receptive field) of the previous layer. Cuts the number of parameters.

–

Pooling (optional): combines the outputs of neurons in a subarea to reduce the data dimensions. The *stride* dictates how the subarea is moved across the image. Max-pooling uses the maximum for each subarea.

Recurrent neural network (RNN)

Used for chain structured data (e.g. text)



From fdeloche, Wikipedia

Example:
Long Short-Term Memory (LSTM)

Implementation

ML libraries

Most popular:

- PyTorch , developed by Facebook's AI Research lab
- TensorFlow , developed by the Google Brain Team

Both most often used through their Python interfaces

Julia's syntax is well suited for the implementation of mathematical models
GPU kernels can be written directly in Julia
Julia's speed is attractive in computation hungry fields

→ Julia has seen the development of many ML packages

Some of the ML packages in Julia

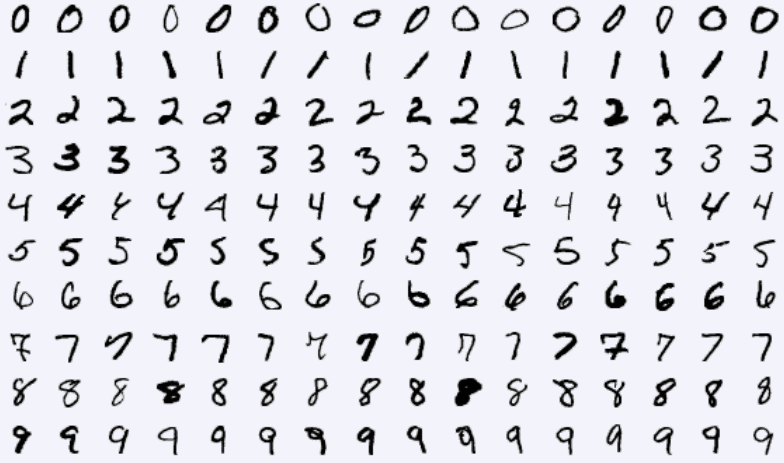
Flux.jl : a machine learning stack
Knet.jl : a deep learning framework
TensorFlow.jl : wrapper for TensorFlow
Turing.jl : for probabilistic machine learning
MLJ.jl : framework to compose machine learning models
ScikitLearn.jl : implementation of the scikit-learn API

Today, we will have a glance at Flux

Example:

Classifying the MNIST

The MNIST database



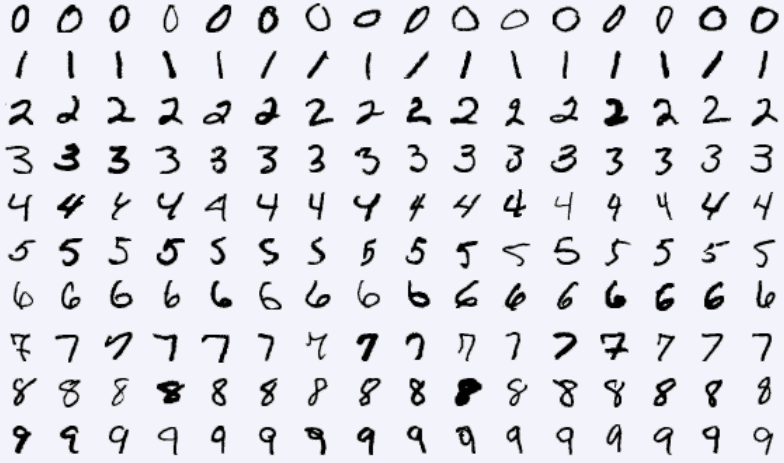
Modified from Josef Steppan, Wikimedia

Pairs of images of handwritten digits and labels
used for testing ML systems

The MNIST database

Images composed of 28x28 pixels of greyscale RGB codes from 0 to 255
Labels from 0 to 9 of the actual digits

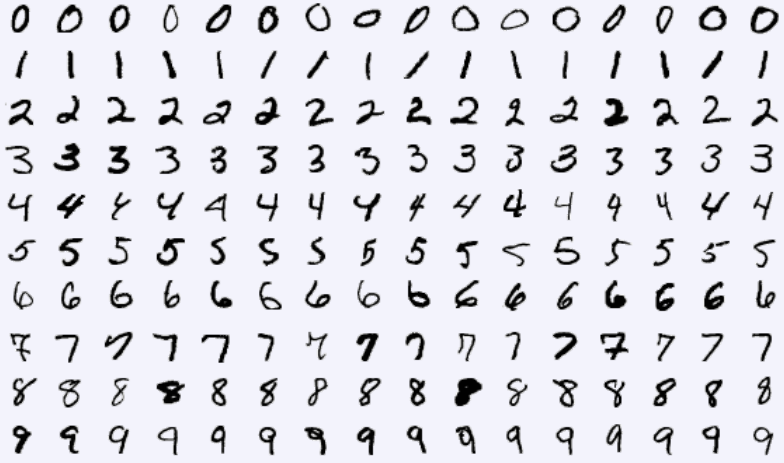
The MNIST database



Modified from Josef Steppan, Wikimedia

Goal: learn proper identification of handwritten digits
Typical case of supervised learning (classification problem)

The MNIST database



Modified from Josef Steppan, Wikimedia

60,000 images to **train**
10,000 images to **test**

How to get started?

The Flux Model Zoo provides examples which are great starting points

How to get started?

Let’s have a look at a few functions together

Load packages and bring functions into scope

[In]

```
using Flux
using Flux: onehotbatch, onecold, crossentropy, throttle
# saves us from having to type Flux.onehotbatch(), etc.
```

Training data

[In]

```
img = Flux.Data.MNIST.images();   # create an array with the training images
lab = Flux.Data.MNIST.labels();   # create an array with the training labels
```

What does the **image** data look like?

[In]

```
typeof(img)
```

[Out]

```
Array{Array{ColorTypes.Gray{FixedPointNumbers.Normed{UInt8,8}},2},1}
```

[In]

```
length(img)
```

[Out]

```
60000
```

`img` is a one-dimensional array of 60,000 two-dimensional arrays

[In]

```
img[1]
```

[Out]

```
28×28 Array{Gray{N0f8},2}
with eltype ColorTypes.Gray{FixedPointNumbers.Normed{UInt8,8}}:
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)  …  Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)     Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)     Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)     Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)     Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
Gray{N0f8}(0.0)  Gray{N0f8}(0.0)  …  Gray{N0f8}(0.0)  Gray{N0f8}(0.0)
```

Each of these arrays represents one training image and is made of `28x28` values of the gray scale of each pixel of the image

[In]

```
img[1][1, 1]
```

[Out]

```
Gray{N0f8}(0.0)
```

This is the value of the top left pixel of the first image of the training dataset

[In]

```
float(img[1][1, 1])
```

[Out]

```
0.0
```

It can be converted to a float

[In]

```
img[1][1, 1]
```

[Out]

```
Gray{N0f8}(0.0)
```

This is the value of the top left pixel of the first image of the training dataset

[In]

```
float(img[1][1, 1])
```

[Out]

```
0.0
```

It can be converted to a float

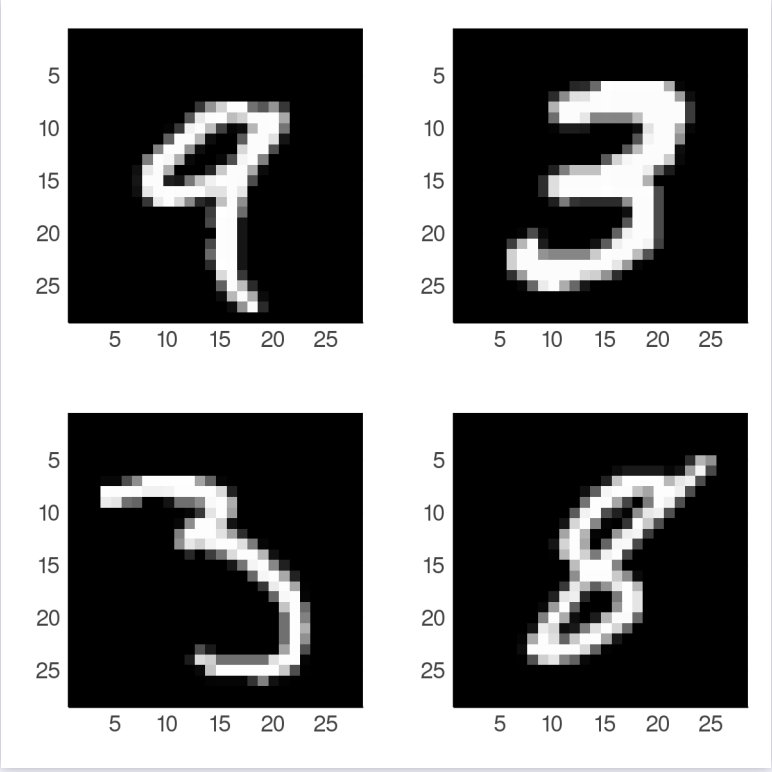
Let's see what a few of these images look like:

[In]

```
using Plots

plot(plot(img[5]), plot(img[8]), plot(img[87]), plot(img[203]))
```

[Out]



What does the **label** data look like?

[In]

```
typeof(lab)
```

[Out]

```
Array{Int64,1}
```

[In]

```
length(lab)
```

[Out]

```
60000
```

`lab` is a one-dimensional array of 60,000 integers

[In]

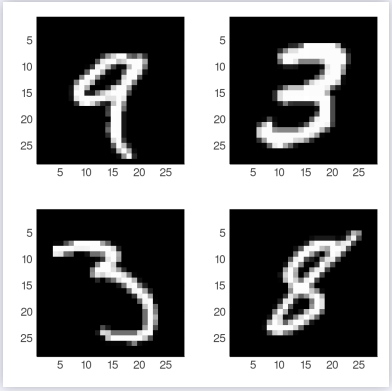
```
lab[1]
```

[Out]

```
5
```

This is the value of the first label

The 5^{th}, 8^{th}, 87^{th}, and 203^{rd} images were:



Here are the corresponding labels:

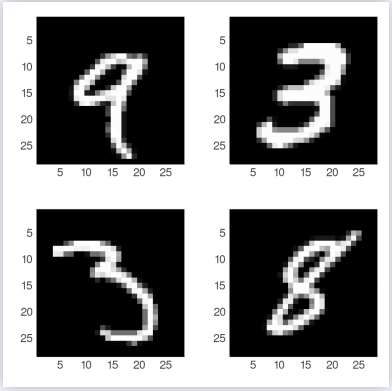
[In]

```
[lab[5], lab[8], lab[87], lab[203]]'
```

[Out]

```
1×4 LinearAlgebra.Adjoint{Int64,Array{Int64,1}}:
 9  3  3  8
```

The 5^{th}, 8^{th}, 87^{th}, and 203^{rd} images were:



Here are the corresponding labels:

[In]

```
[lab[5], lab[8], lab[87], lab[203]]'
```

[Out]

```
1×4 LinearAlgebra.Adjoint{Int64,Array{Int64,1}}:
 9  3  3  8
```

onehotbatch

`onehotbatch()` turns a batch of labels into a binary matrix

`onecold()` performs the inverse operation

onehotbatch

[In]

```
lab[1:3]
```

[Out]

```
3-element Array{Int64,1}:
 5
 0
 4
```

onehotbatch

[In]

```
onehotbatch(lab[1:3], 0:9)
```

[Out]

```
10×3 Flux.OneHotMatrix{Array{Flux.OneHotVector,1}}:
 0  1  0
 0  0  0
 0  0  0
 0  0  0
 0  0  1
 1  0  0
 0  0  0
```

onecold

[In]

```
onecold(onehotbatch(lab[1:3], 0:9), 0:9)
```

[Out]

```
3-element Array{Int64,1}:
 5
 0
 4
```

Splatting

Consider this simple example:

[In]

```
+(2, 3)
```

[Out]

```
5
```

[In]

```
a = (2, 3);
+(a)
```

[Out]

```
ERROR: MethodError: no method matching +(::Tuple{Int64,Int64})
```

Splatting

Consider this simple example:

[In]

```
+(2, 3)
```

[Out]

```
5
```

[In]

```
a = (2, 3);
+(a)
```

[Out]

```
ERROR: MethodError: no method matching +(::Tuple{Int64,Int64})
```

Splatting

Consider this simple example:

[In]

```
+(2, 3)
```

[Out]

```
5
```

[In]

```
a = (2, 3);
+(a...)
```

[Out]

```
5
```

Splatting

Consider this simple example:

[In]

```
+(2, 3)
```

[Out]

```
5
```

[In]

```
a = (2, 3);
+(a...)
```

[Out]

```
5
```

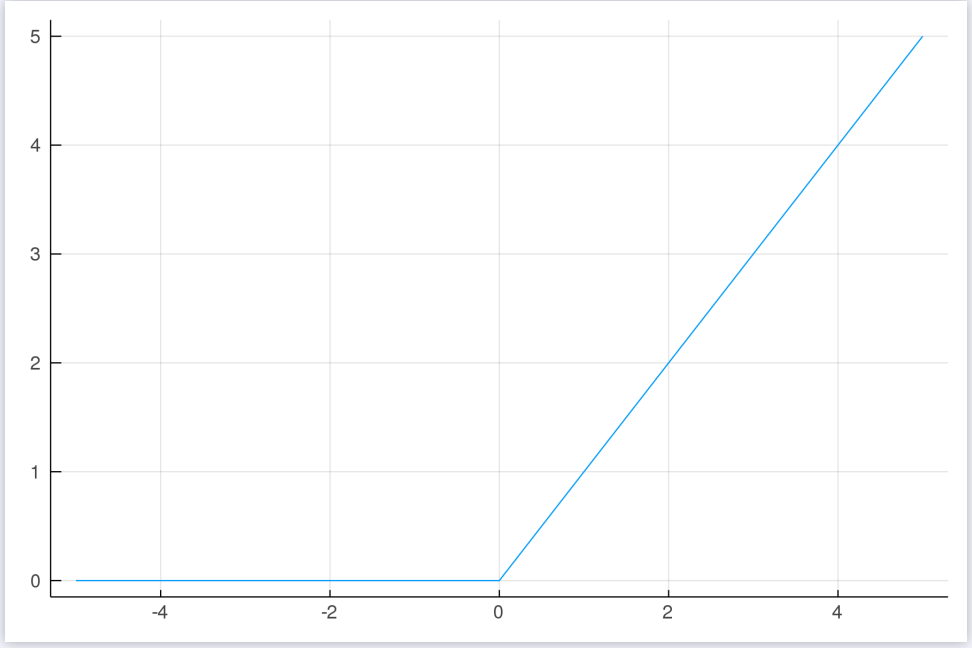
Relu: rectifier activation function

$f(x) = max(0, x)$ In Flux: `relu()`

[In]

```
x = -5:5; plot(x, relu.(x), legend = false)
```

[Out]



Softmax activation function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

($\mathbb{R}$ represents the set of real numbers)

Normalizes a vector of real numbers
into a vector of numbers between 0 and 1 which add to 1

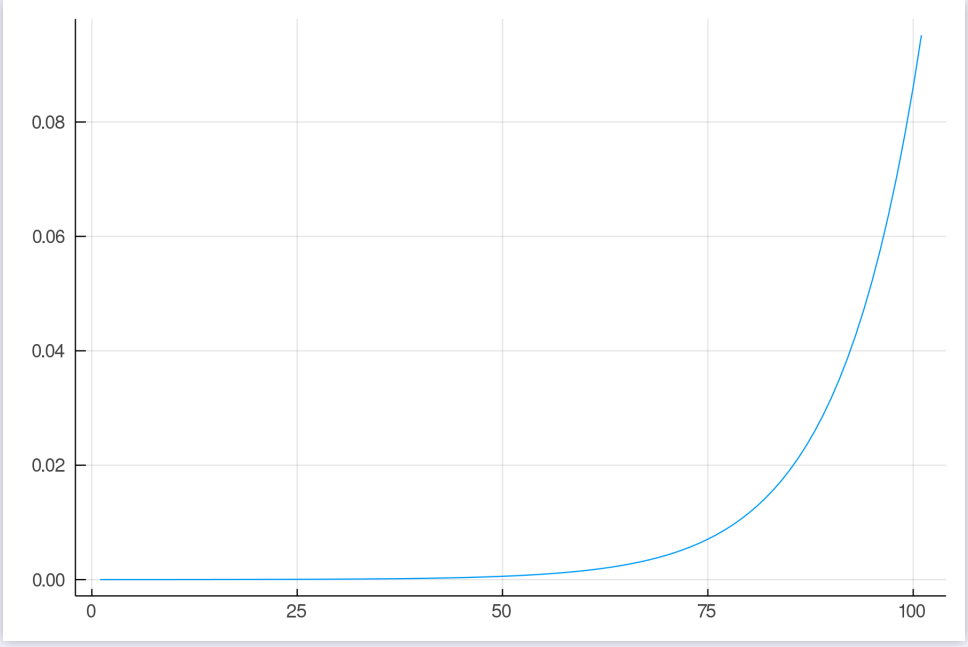
Softmax activation function

In Flux: `softmax()`

[In]

```
plot(softmax(-5.0:0.1:5.0))
```

[Out]



Multi-layer perceptron from Model Zoo

```
using Flux, Statistics
using Flux.Data: DataLoader
using Flux: onehotbatch, onecold, logitcrossentropy, throttle, @epochs
using Base.Iterators: repeated
using Parameters: @with_kw
# using CUDAapi
using MLDatasets
# if has cuda()
```

CNN from Model Zoo

```
using Flux, Flux.Data.MNIST, Statistics
using Flux: onehotbatch, onecold, logitcrossentropy
using Base.Iterators: partition
using Printf, BSON
using Parameters: @with_kw
# using CUDAapi
# if has_cuda()
#     @info "CUDA is on"
```

Saving models

BSON.jl](https://github.com/JuliaIO/BSON.jl) allows to save models in [Binary JSON format

Transfer learning (TL)

Instead of random initialization,
start from model trained on related problem for weight initialization

→ Allows to get good results on small datasets

ONNX.jl](https://github.com/FluxML/ONNX.jl) allows to read pre-trained models from [ONNX format to Flux.

GPU support

CuArrays.jl provides GPU functionality to Flux

[In]

```
using CuArrays
```

The function `gpu()` converts the parameters and moves data to the GPU
Code can for instance be piped into it:

[In]

```
<some code> |> gpu
```

The function `cpu()` converts the parameters and moves data back to the CPU

Questions?