Building a Sudoku Solver

ML/AI @ SSN Coding Club: Meet 5

Part 2: Deep Neural Networks

Scan to ask questions anonymously

…or just ask away in Teams chat!

What are we covering?

- An introduction to neural networks, the major components in them, how they work, with resources to learn about them deeply and to test them out right on the browser.
- Coding neural networks from scratch and with libraries and applying it to our sudoku problem.
- Looking at how neurons learn patterns from images without explicitly hard-coding them.

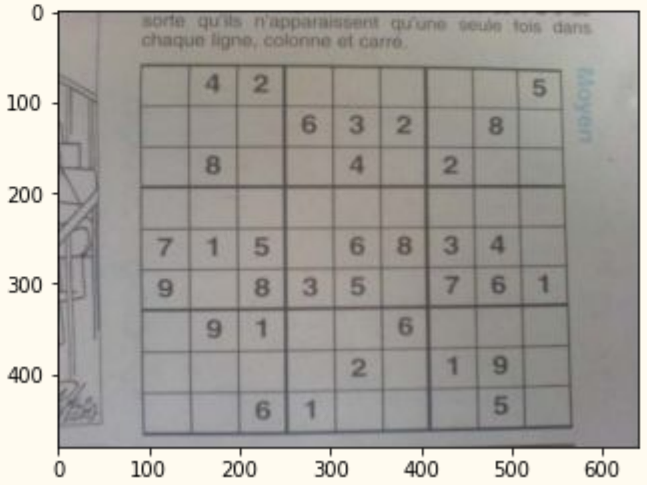
Where did we stop?

1. Finding the largest square in the image
2. Unskewing the image
3. Extracting the cells
4. Deciding if each cell contains a digit or not
5. Applying an ML model like a neural network to recognize the digits
6. Solving the sudoku

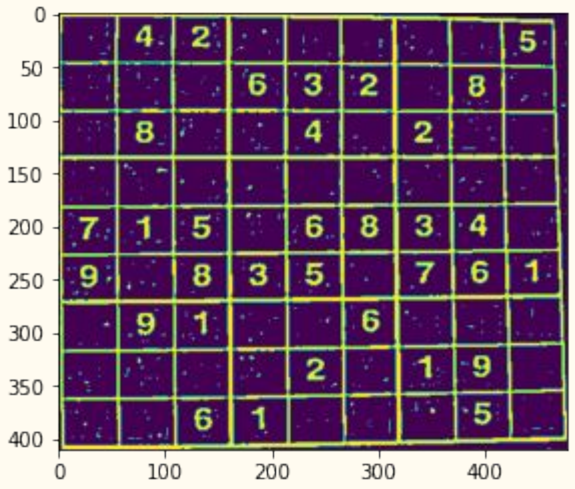
- We have finished up to step 4 in the last meet, this meet covers step 5.

Steps 1 & 2

Original Image


sorte qu'ils n'apparaissent qu'une seule fois dans chaque ligne, colonne et carré.
Moyen
0
100
200
300
400
500
600

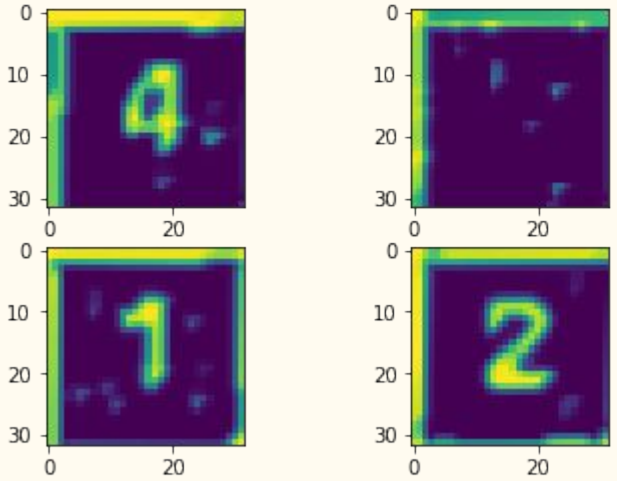
After steps 1-3 using image processing methods



Step 3 & 4

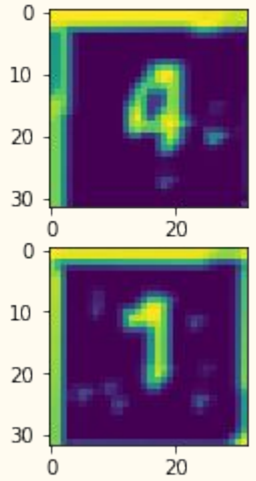
Divide the image-area into 81 (9x9) equal square cells. Each cell is 32x32.

Use these cells for digit recognition. Now we train a classifier to classify if a cell **contains a digit in it or not**.



We used logistic regression for this task and achieved good accuracy (94%).

Step 5



Now for those cells that do have a digit, we need to find which digit (1-9) is present.

We attempted to use 9 different logistic regression classifiers for this task and only got a 57% accuracy.

How can we improve the accuracy?

We need a more complex model, like a neural network.

With a neural network, we can also combine step 4 and 5 into a single step, making the network classify the 10 different possible outcomes: the digits 1-9 and the absence of a digit.

Deep Learning

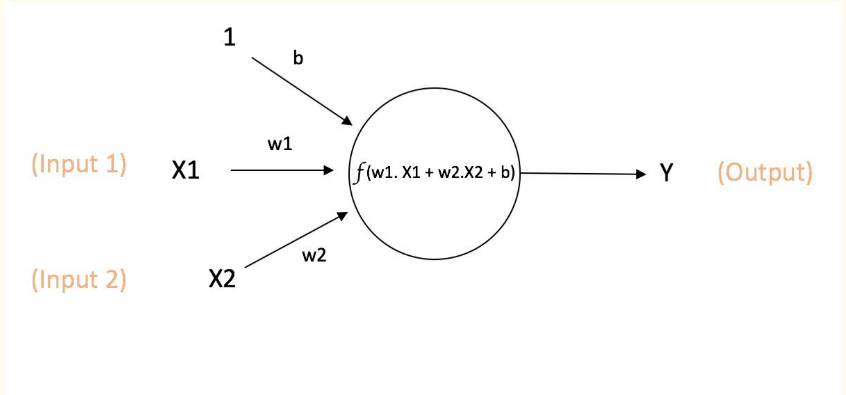
- An approach to model data where **raw data** can be sent directly to the model, without engineering any features explicitly.
- In the last meet, to use logistic regression to differentiate 2 digits, we used properties of the digits like pixel values sum, the number of border pixels etc.
- We handcrafted these features as we know that these features can differentiate a 0 and 1 (for example). We only feed these features to the model, not the raw pixel data in the image itself.
- Deep learning – we can directly feed raw image data to the models. They automatically learn features without human domain knowledge.

Deep Learning

- The simplest deep learning model is a **feed-forward neural network** (NN), also called a fully-connected neural network.
- These artificial neural networks aim to loosely mimic how biological neurons in the brain react to each other.
- Note that, what changes in deep learning is the fact that raw data is fed (usually) and **how the model is defined**. How we feed the data, optimizer, loss function etc. stay the same as traditional machine learning.
- Neural networks can capture non-linear data.

What is a neuron?

- A neuron is the simplest unit of a neural network.
- It receives a set of input values, it **aggregates** them (weighting each input based on a set of **weights**) and then applies a nonlinear function called the **activation function** to the weighted sum.
- This activation value is used as the **input to other neurons down the line**.



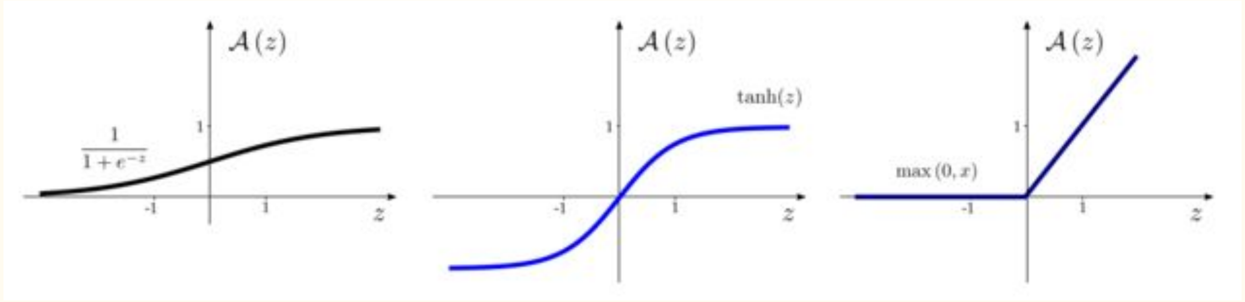
Output of neuron $= Y = f(w1.X1 + w2.X2 + b)$

Activation functions - Why?

The idea behind the activation function is:

1. To introduce non-linearity in the model
2. To squish the output to a fixed small range (for eg: sigmoid)

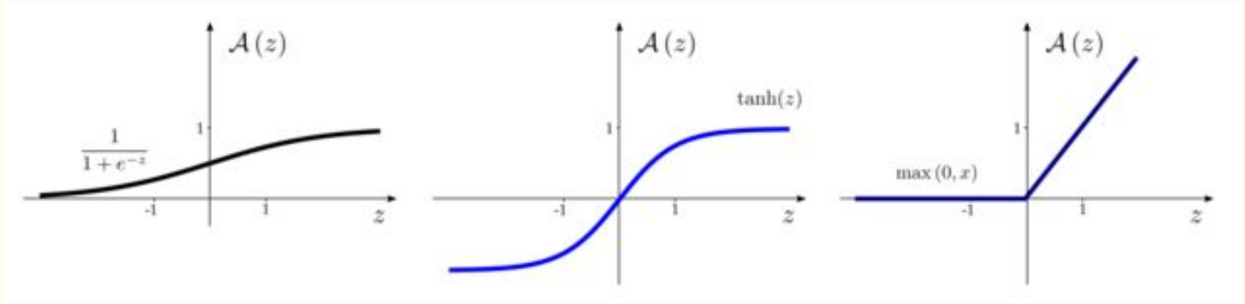
(A single sigmoid neuron is essentially just logistic regression)



sigmoid **tanh** **ReLU**
(Rectified Linear Unit)

Activation functions

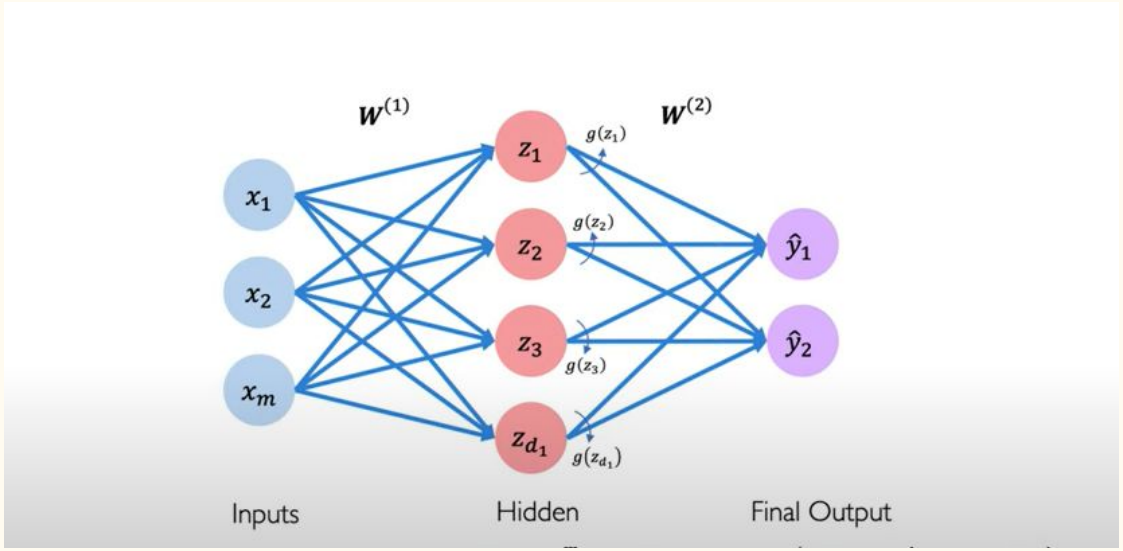
There are many choices for activation functions and reasons to choose one over the other. Traditionally, sigmoid functions were used, now improved functions like ReLU are used widely.



sigmoid **tanh** **ReLU**
(Rectified Linear Unit)

Forming a layer

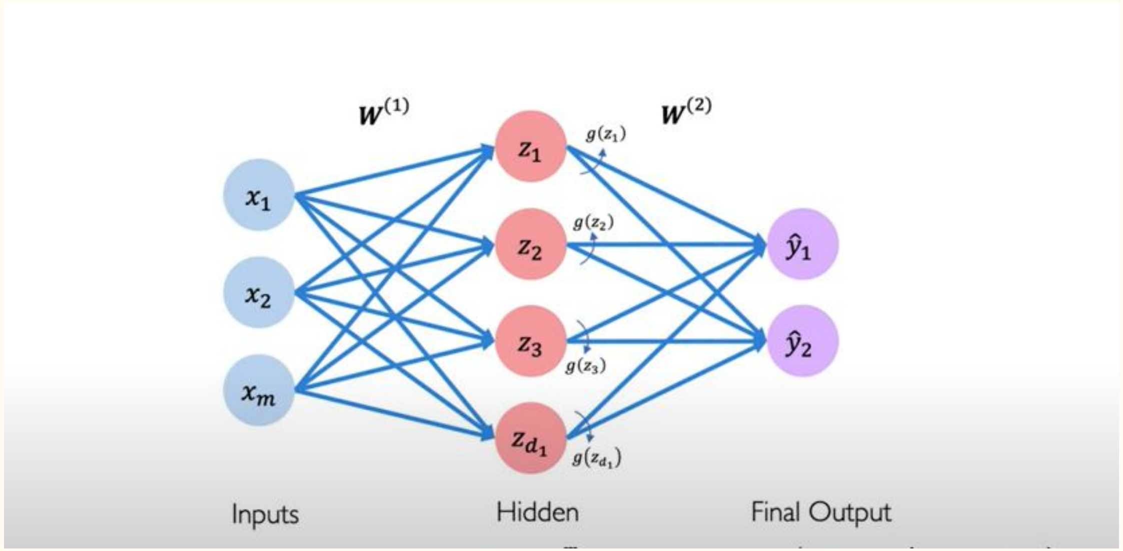
We arrange a set of neurons to form a layer, where each neuron’s activation value is calculated from all the neurons in the previous layer.



Note: The biases are not shown here

Input and output layer

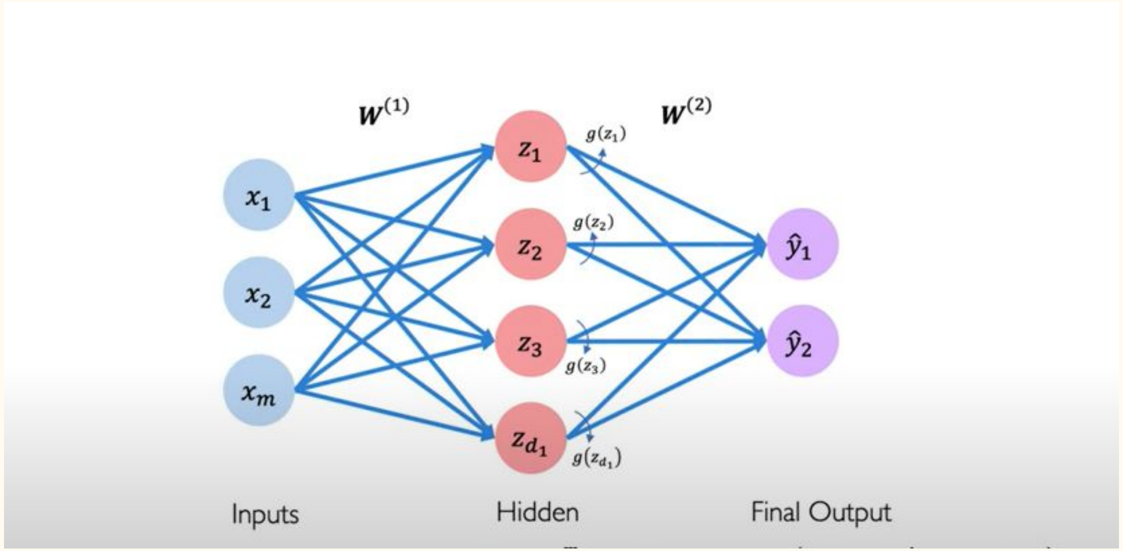
The first layer, where we directly feed our input data (the features for each training example) is called the input layer, and the output is produced in the final layer.



Note: The biases are not shown here

Input and output layer

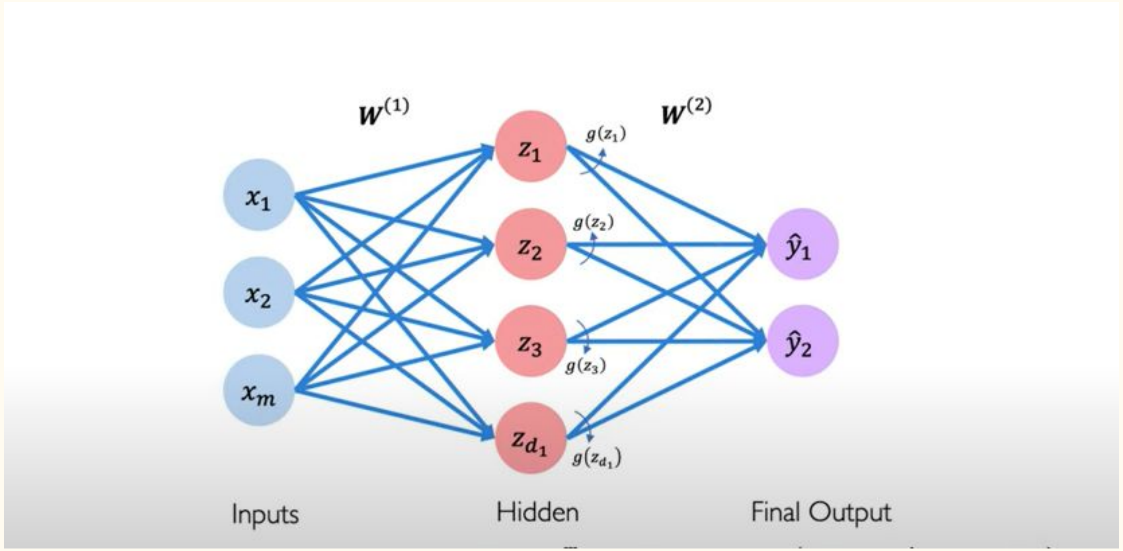
The number of neurons in the input layer equals the number of features. The number of output neurons depends on the labels/targets of our task.



Note: The biases are not shown here

Hidden layers

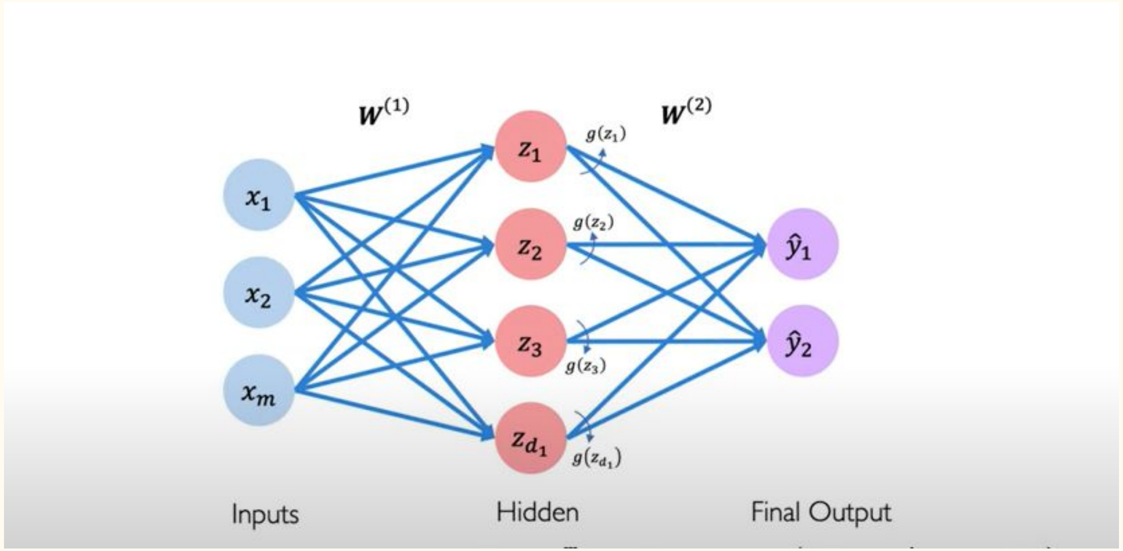
The number of hidden layers and the number of neurons per hidden layer is our choice. The more the number of hidden layers, the more **deeper** the model is!



Note: The biases are not shown here

The overall model

So to get the output of the model, we feed the input data to the input layer, and propagate it forward layer-by-layer, till it reaches the output layer.



Note: The biases are not shown here

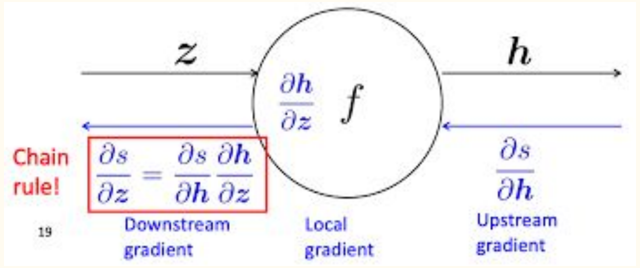
Training a neural network

- The loss function, optimizer, metrics etc. remain exactly the same as those in ML we discussed in the previous sessions. But how do we calculate the gradient? Using **backpropagation**.
- In contrast to how we forward propagate the data from input layer to output layer, to calculate the gradient of the loss with respect to all the weights, we **propagate errors backward,** starting off with the output layer.

Training a neural network

- Explaining backpropagation deeply is beyond the scope of this meet, but the core idea behind it lies in this formula:

Downstream Gradient = Upstream Gradient x Local Gradient


z
h
$\frac{\partial h}{\partial z}$
f
Chain rule!
$\frac{\partial s}{\partial z} = \frac{\partial s}{\partial h}\frac{\partial h}{\partial z}$
$\frac{\partial s}{\partial h}$
19
Downstream gradient
Local gradient
Upstream gradient

- The best resource to learn neural networks very deeply: **http://neuralnetworksanddeeplearning.com/**

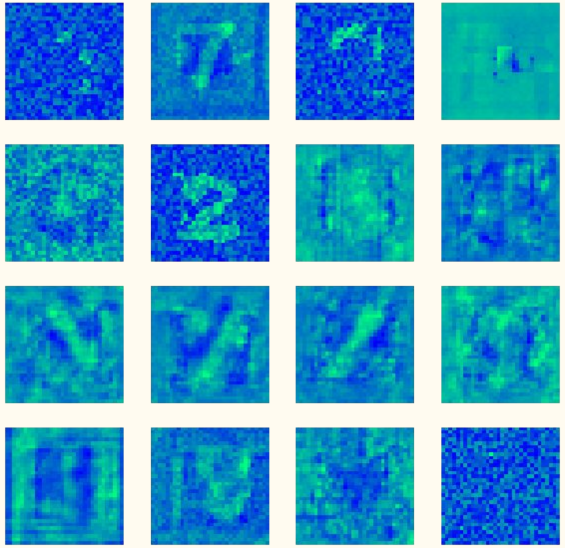
Coming back to the sudoku task..

In our case, we have a 32x32 image. So our neural network would look something like:

1. An **input layer of 1024 (32x32) neurons** where we feed the pixel values of these images.
2. **A hidden layer**, we can test a few values for the number neurons here, let's say 16, 32, 64 etc. The **ReLU** activation function can be used.
3. An **output layer of 10 neurons**, one for each digit 1-9 and the last one being for the absence of digits. We use the sigmoid activation here, since we need a probability value between 0 and 1 that says how confident the network is about the presence of each digit (or no digit).

What do the hidden layers learn?

We trained this neural network with 16 hidden layer neurons. Each of these 16 neurons has 1024 weights to the input layer, arranging these into a 32x32 image, we see something interesting..



Thank you