# Hello World: TinyML

Dr Sudeepta Mishra

# Topics

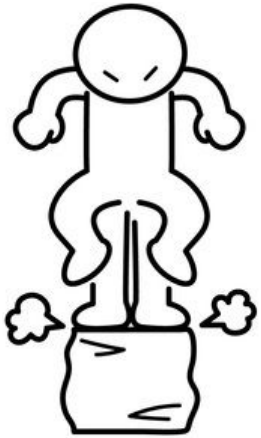
- Tiny models
- Hello World
- NN size
- Deployments[Wokwi?]
- Another example

# Tiny Models

# How to get tiny models?

# How to get tiny models?

- Compress large models
- Train small models



# What kind of models are compressible?

# What kind of models are compressible?

Neural Network models.

# What happens to models when compressed?

# What happens to models when compressed?

Following things change:

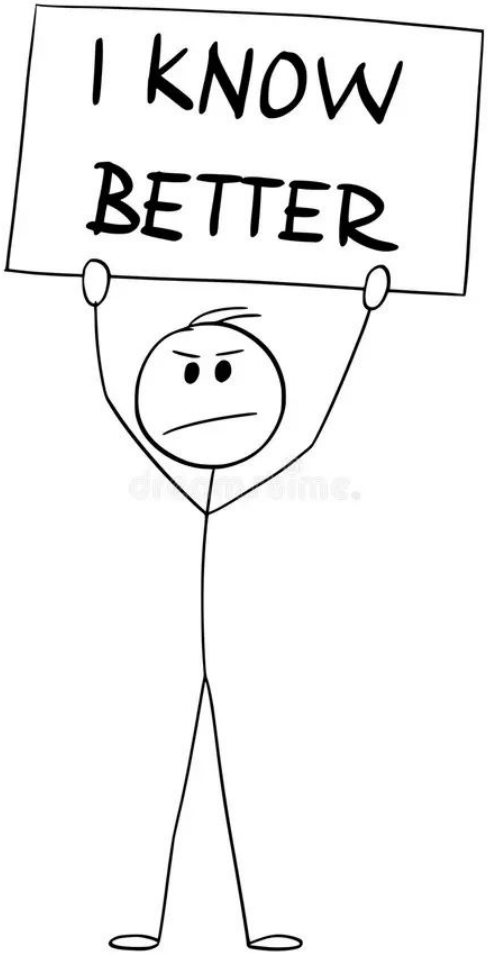
- Model size
- Model latency
- Model accuracy

# How to compress a model?

- Pruning
- Quantization
- Knowledge distillation

# How to compress a model?

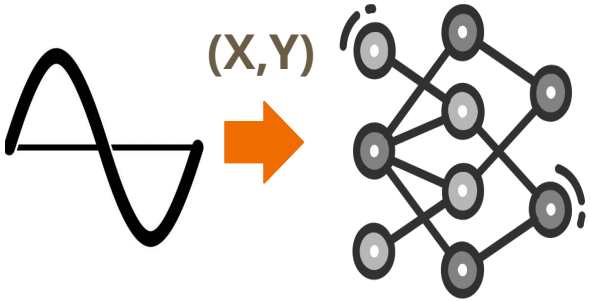
- Pruning
- Quantization
- Knowledge distillation



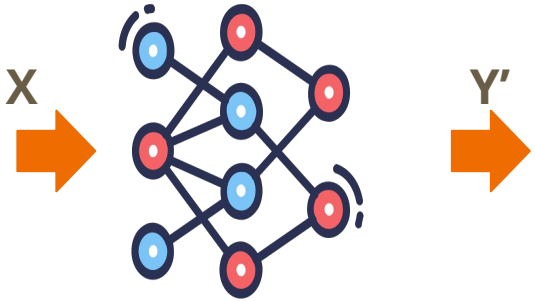
# Hello World

# Sinewave Prediction

**Learn Y=Sin(X)**



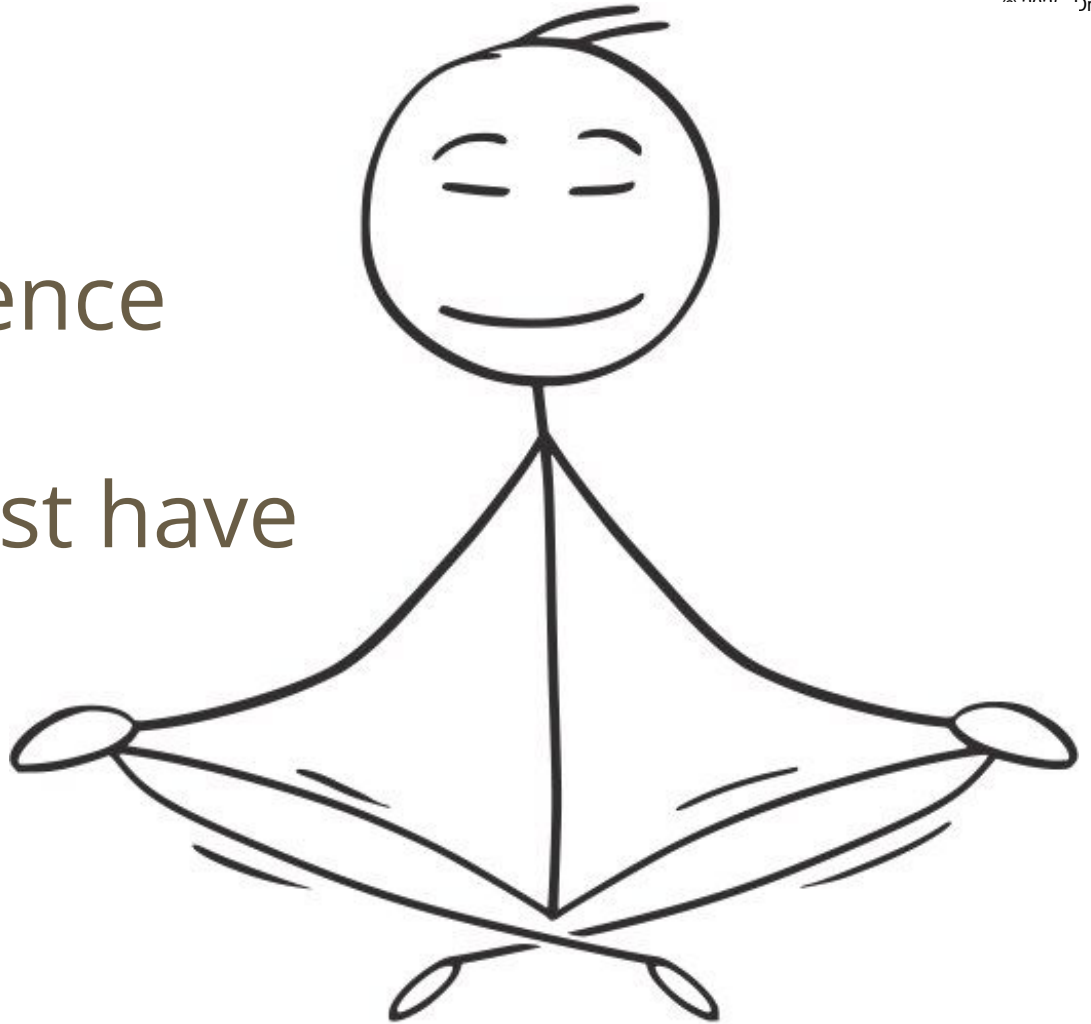

Train



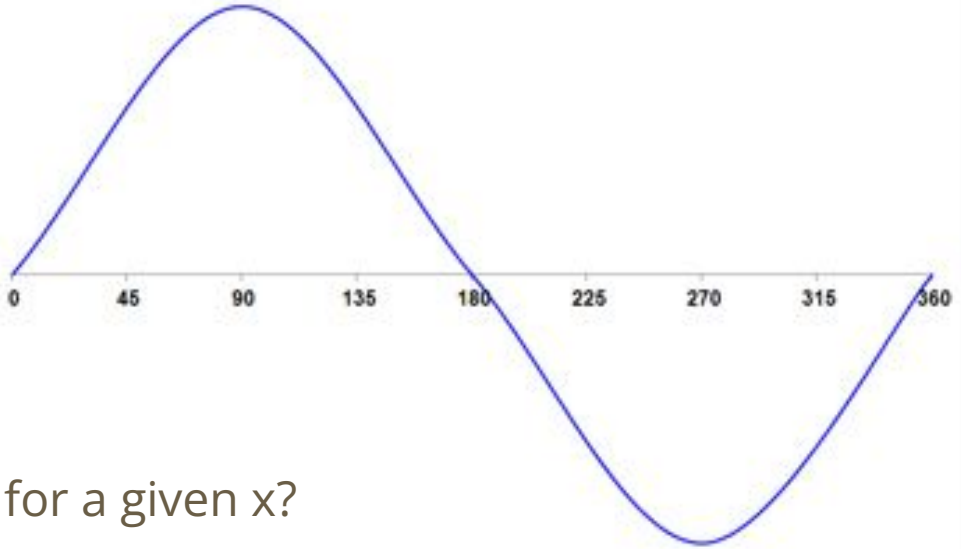

Predict

Patience

You must have



# Sinewave Prediction



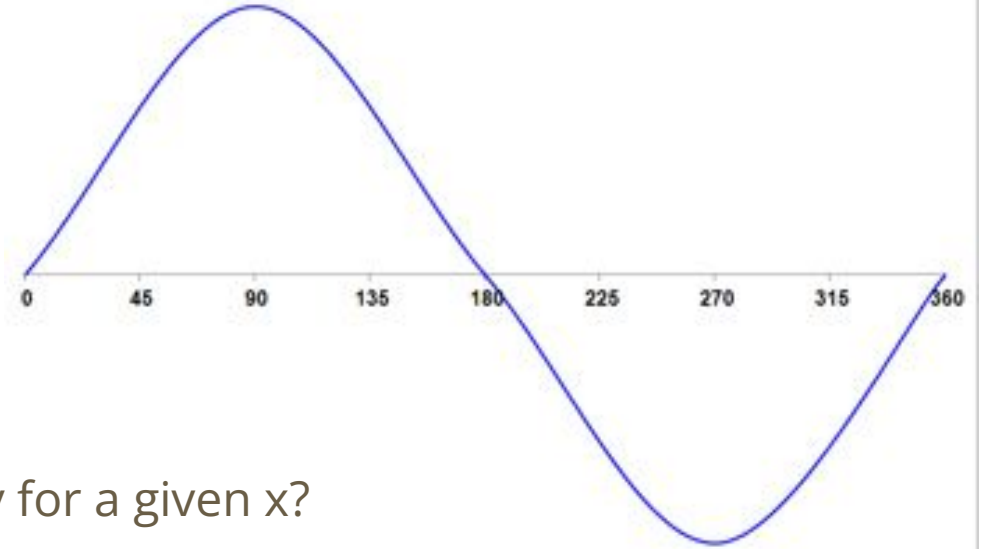

What is a sinewave

$y=A \sin(x)$ ; //$x=\omega t$

What about using a NN model to get y for a given x?

How?

# Sinewave Prediction

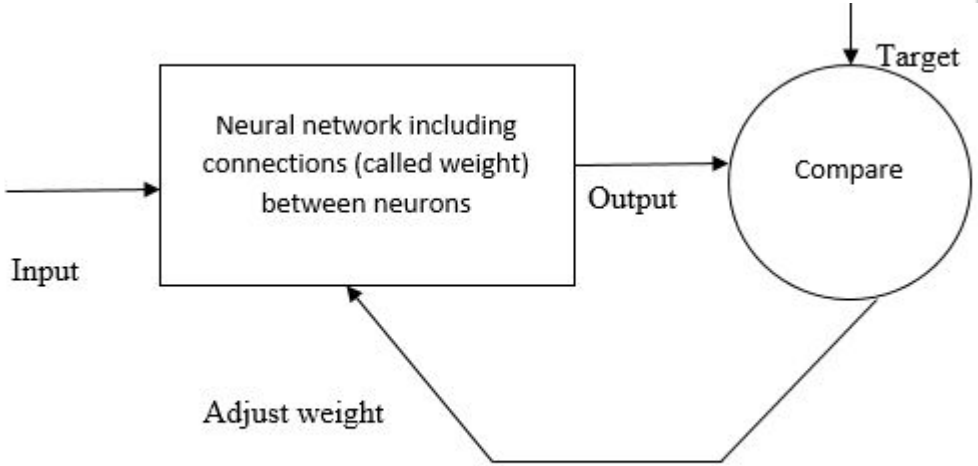



What is a sinewave

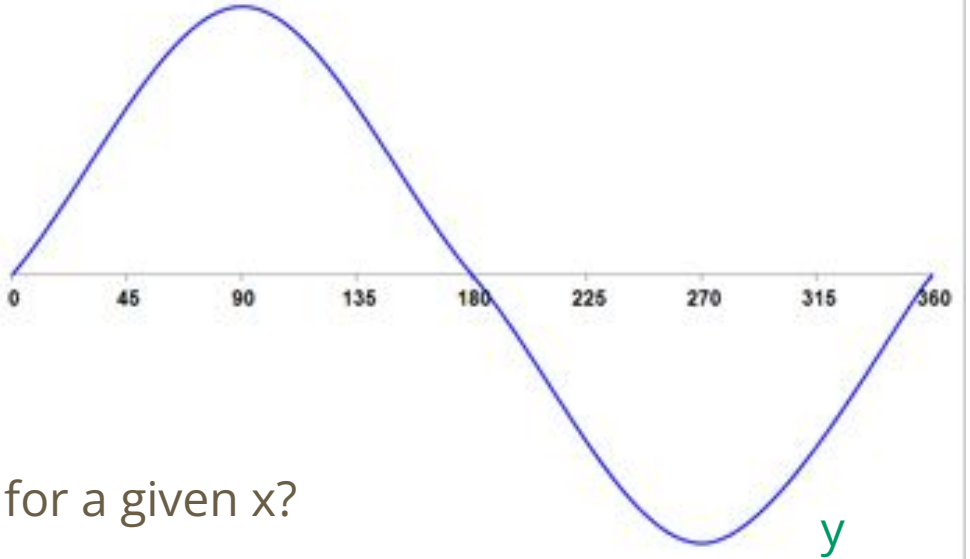
y=A sin(x) ; //x=**ω**t

What about using a NN model to get y for a given x?

How?



# Sinewave Prediction

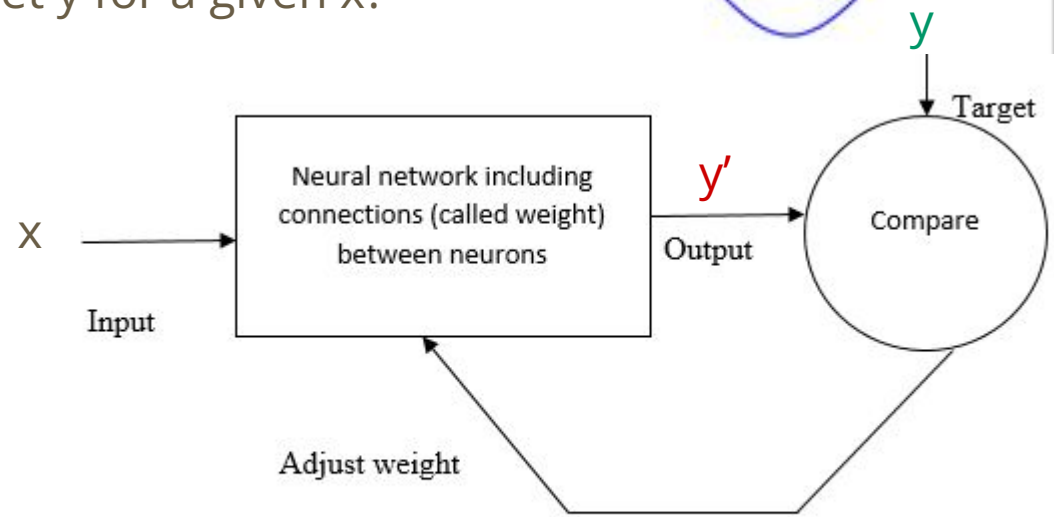



What is a sinewave

y=A sin(x) ; //x=**ω**t

What about using a NN model to get y for a given x?

How?



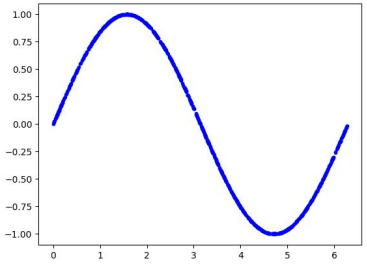
# Bill of materials/tools

- Lots of patience
- Google Colab
- TensorFlow Lite/Macro
- ESP32 or Nano 33 BLE etc.
- Wokwi [optional]

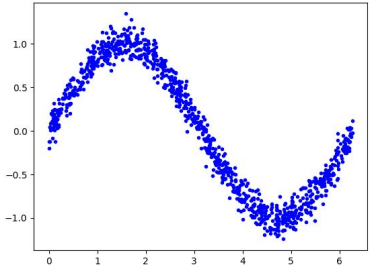
# Let's do it the hard way

# Bill of materials/tools

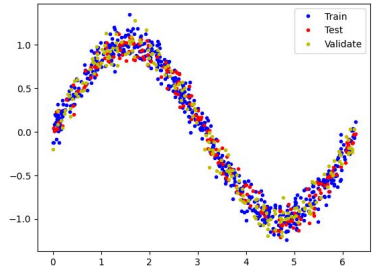
- Lots of patience
- Google Colab
- MCU and accessories
- Without special library
- Very small footprint



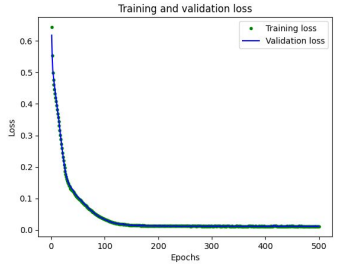
→ Add Noise →



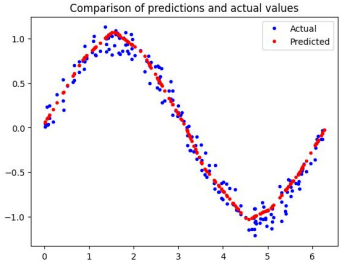
→ Split →




→ Train your model →




Uniformly distributed random numbers in the range from 0 to 2π covering a complete sine wave oscillation




Check against test set

The model consists of the following layers:

- input layer, 1 neuron
- dense layer, 16 neurons, ReLU activation
- dense layer, 16 neurons, ReLU activation
- dense layer, 1 neuron, no activation

There are three weight matrices:

- W1 shape (1, 16)
- W2 shape (16, 16)
- W3 shape (16, 1)

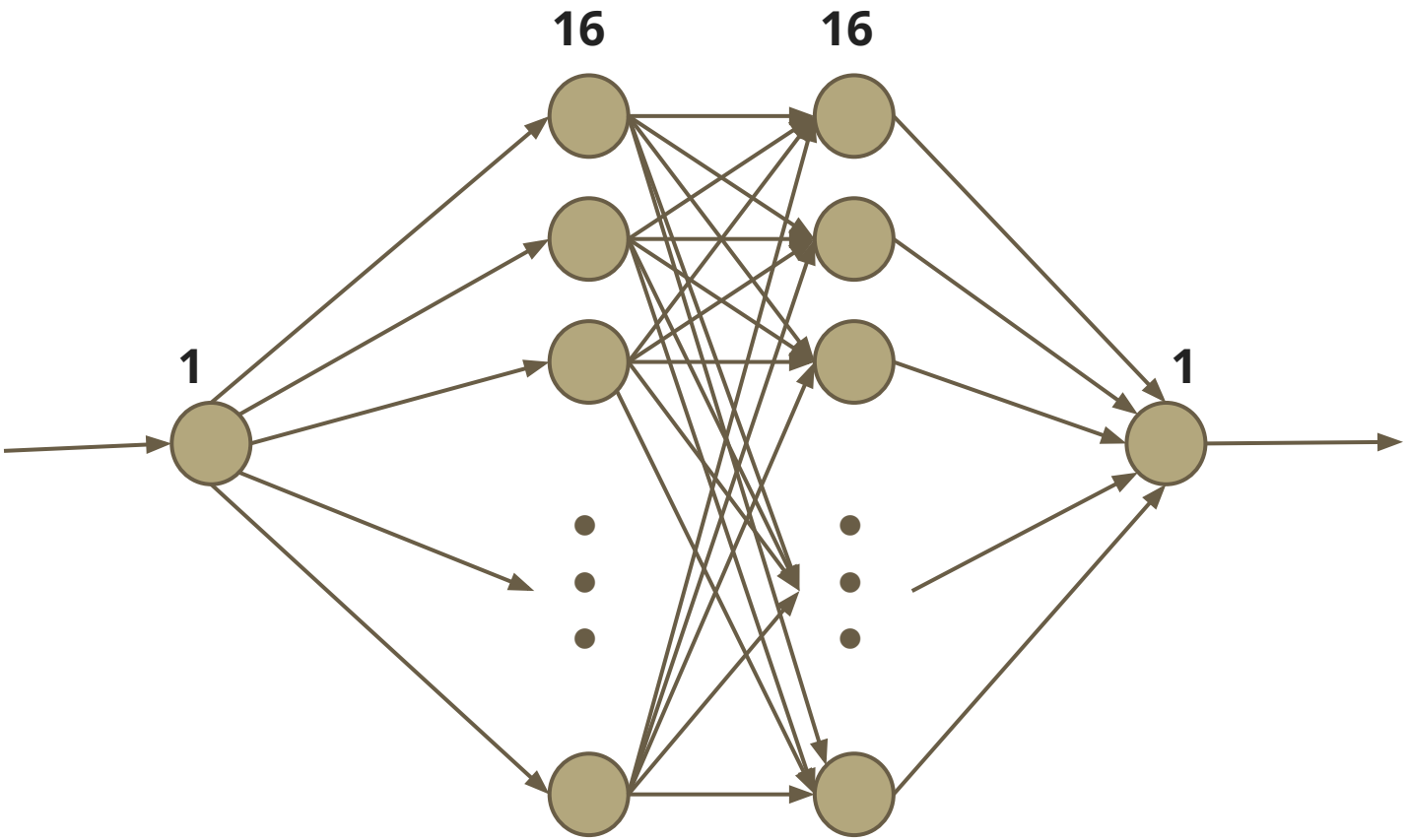
Each layer also has bias parameters

```
#TF model does internally:
ys = []
for x in xs:
        x = np.array([x])       # x should be array
        h1 = x @ W1 + b1   # dense layer
        h1 = np.maximum(0, h1)  # ReLU
        h2 = h1 @ W2 + b2   # dense layer
        h2 = np.maximum(0, h2)  # ReLU
        h3 = h2 @ W3 + b3  # dense layer
        ys.append(h3)
ys = np.stack(ys)
```

Weight matrices for the first and last layer are vectors because there is only one input neuron and one output neuron.
In total there are 321 parameters.

**If we store those as 32-bit (4 bytes) floats, it takes up 1284 bytes.** Really How?

# The NN



# Calculate the size of NN

The model consists of the following layers:

- Input layer, 1 neuron
  - No weight or bias value. It just passes the input to the inner layer. (copies the single input to 16 neurons). Thus, 0 floating point values.
- Dense layer, 16 neurons, ReLU activation
  - One input per neuron. 16 neurons=>16 input weights, then 16 bias values, thus we have 32 (16+16) floating point values.
- Dense layer, 16 neurons, ReLU activation
  - Each neuron will get 16 inputs. 16 neurons=>16x16 input weights, then 16 bias values, thus we have 272 (16x16+16) floating point values.
- Output layer, 1 neuron, no activation
  - The neuron will get 16 inputs, thus 16 weights, then a single bias value will be applied. Thus we have 17 (16+1) floating point values.

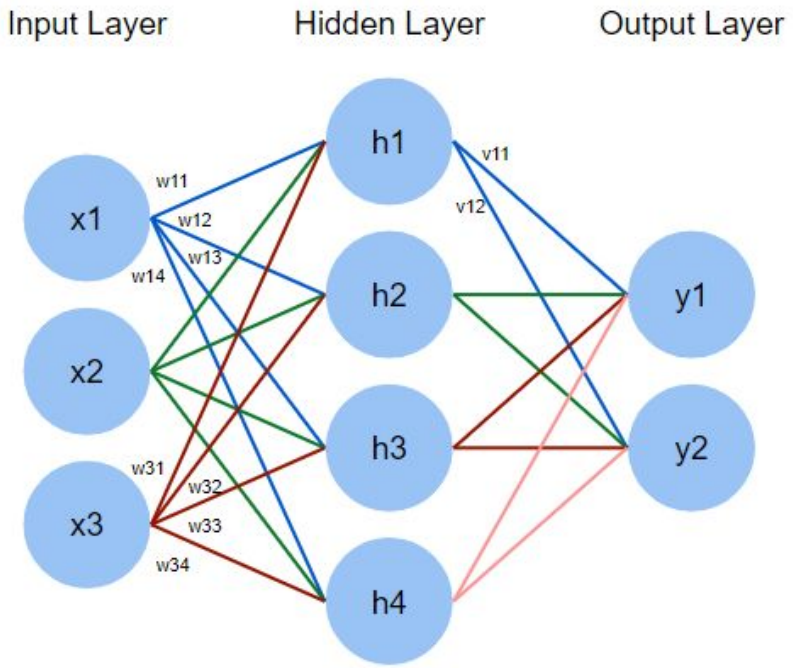
Total we have = 0+32+272+17=321 floating point values.

It takes up 1284 bytes.

# Inference, How?

The model needs to perform these different functions:

- Scalar-vector multiplication for the first layer
- Vector-matrix multiplication for the middle layer
- Dot product for the last layer
- Addition (vector) for the bias terms
- ReLU



# Thank You.