Multi-Input Multi-Output Electric Motor Signal Prediction using Neural Networks

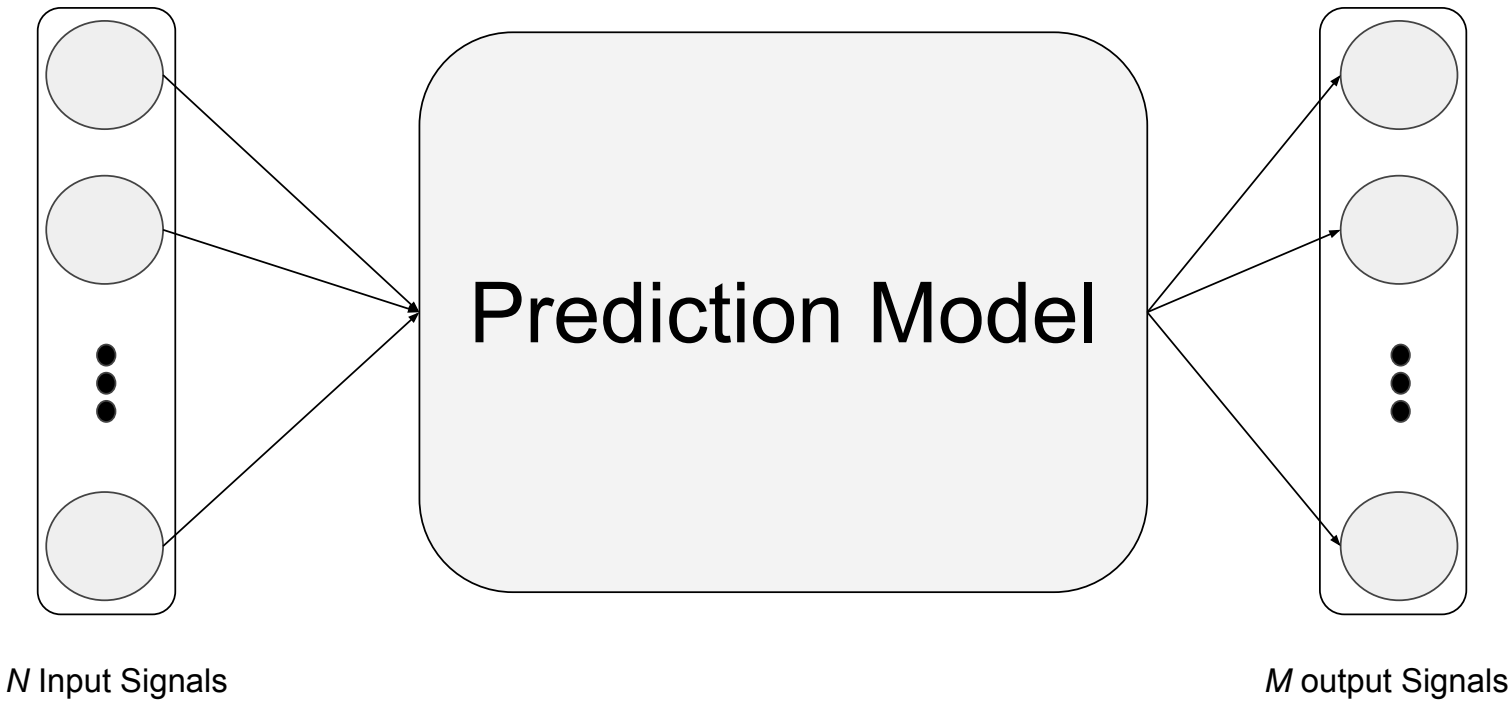
Sagar Verma

Centre de Vision Numérique,
Centralesupélec, Gif-sur-Yvette

November, 2018

Problem Statement

1. Input: multiple signals
2. Output: multiple signals.
3. Continuous signals: regression problem.



Challenges

1. How to handle continous signals?
2. How to handle long sequences? Electric motors generally operate for long hours.
3. Which prediction model to use?
4. How to handle multiple outputs?

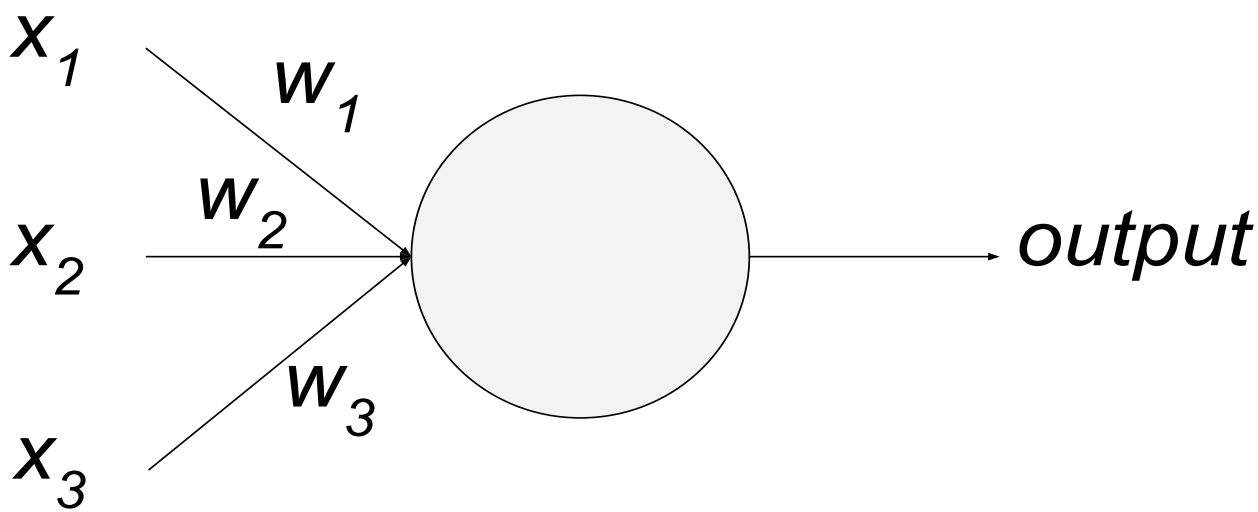
Background

1. Perceptrons
2. Feed-forward Networks
3. Activation Functions
4. Backpropagation
5. Loss Functions
6. Sequential Networks

Perceptrons

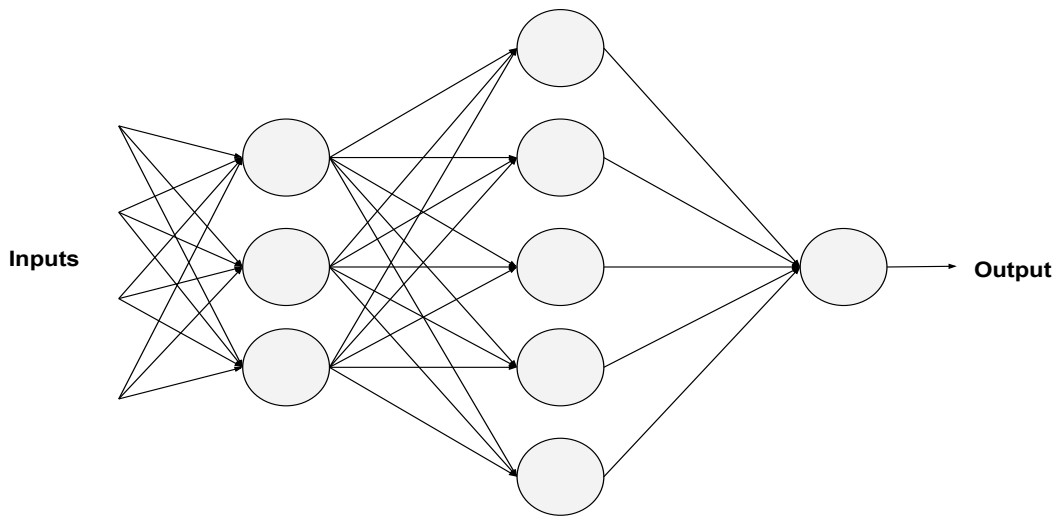
1. Basic block of neural networks
2. Input: binary variables, x_i
3. Output: Binary decision, y
4. w_i is a weight, output is calculated using

$$\text{output} = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{ threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{ threshold} \end{cases} \quad (1)$$



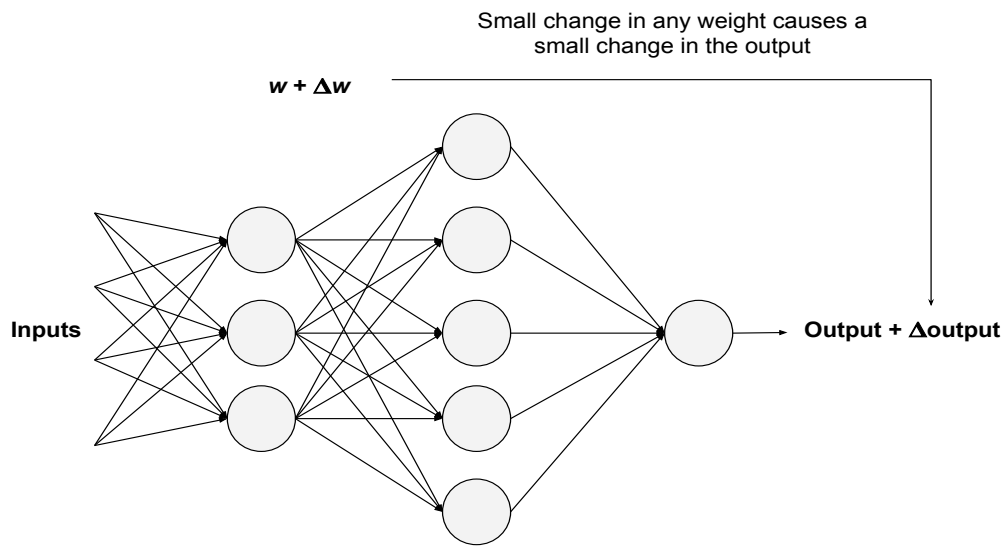
Feed-forword Networks

1. AKA Artificial Neural Networks (ANNs)
2. Multiple stacked perceptrons
3. #connections increases with #perceptrons, input size, and output size
4. Learning weights is difficult
   - 4.1 Binary output does not tell us which weights are important and which are not
   - 4.2 A small change in input or weight could flip the output from 0 to 1 or vice versa



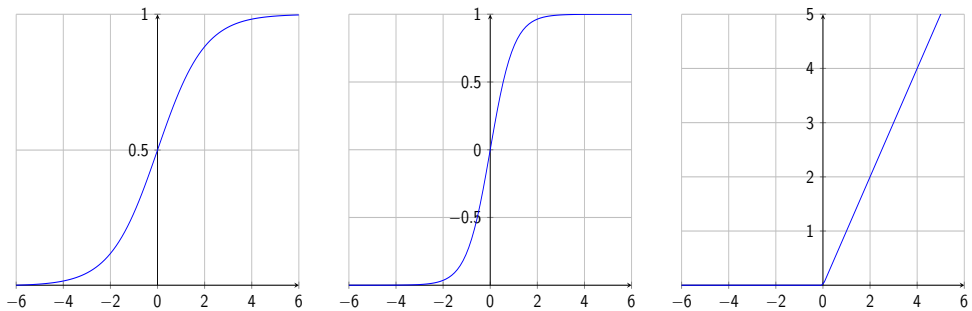
Activation Functions

1. To learn weights of an ANN we need continous output.
2. Function that can give small changes in output when small changes in weights are made.
3. Should give a smooth output.
4. f should be non-linear, $y = f(w^T x + b)$



Activation Functions: Examples

1. Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-\sum_i w_i x_i)}$
2. Tanh: $f(x) = tanh(x) = \frac{\exp(\sum_i w_i x_i) - \exp(-\sum_i w_i x_i)}{\exp(\sum_i w_i x_i) + \exp(-\sum_i w_i x_i)}$
3. Rectified Linear Unit (ReLU): $f(x) = \begin{cases} 0 & \text{if } \sum_i w_i x_i < 0 \\ x & \text{if } \sum_i w_i x_i \geq 0 \end{cases}$



Loss Functions

1. Tells us how good or bad weights are.
2. **L1 Loss** $\mathcal{L}(y_{true}, y_{pred}) = |y_{true} - y_{pred}|$
3. **MSE Loss** $\mathcal{L}(y_{true}, y_{pred}) = (y_{true} - y_{pred})^2$
4. **Cross Entropy Loss** (classification)
   $\mathcal{L}(y_{true}, y_{pred}) = -\sum_{c=1}^{C} y_{true}^{c} log(y_{pred}^{c})$

We have a network (ANN + activation function).
We can judge it (loss function).
How do we learn weights?

Learning Weights

1. Small loss means good weights.
2. Learn weights using *Optimizers*.

Optimizers

1. Cannont process large dataset at once.
2. Process data in small parts (mini-batches).
3. Optimizers for learning weights using mini-batches.
4. Stochastic Gradient Descent is the most used optimizer.

We can now train an ANN.
Can we process continous signals?

Sequential Networks

1. ANNs can be used by splitting sequences.
2. Sequence networks for temporal learning.
3. Recurrent neural networks (RNNs) and Long-Short Term Memorry (LSTM).

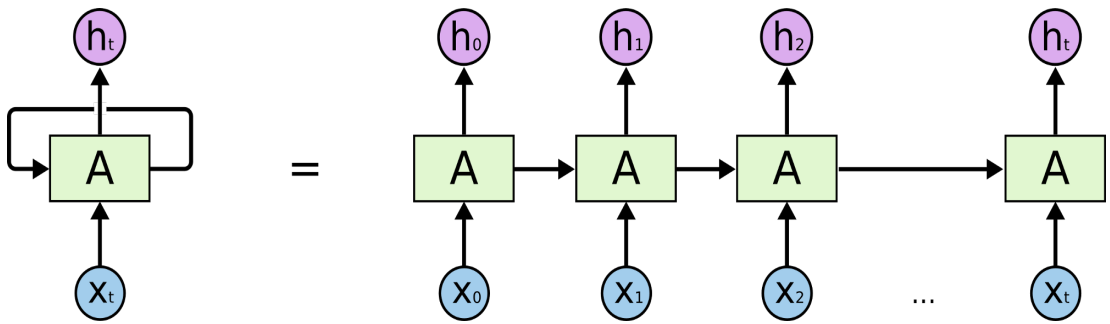


Figure: RNN unrolled in time. *source*

Recurrent Neural Networks

1. Perform same task for every element of a sequence.
2. Output depends on previous elements.
3. RNNs can be seen as a neural network having "memory".

$$h_t = tanh(Wx_t + Uh_{t-1}), \tag{2}$$

where W and U are weights, h is the hidden vector and x_t is the input at time t.

Long-Short Term Memory

1. RNNs have vanishing and exploding gradients problem.
2. In practice gradients become too big or small for long sequences.
3. LSTM resolves above problems.
4. Knows when to forget and when to remember.

Neural networks for motor control

Dataset

1. *CNC Mill Tool Wear* dataset from *kaggle*.
2. Provides real world motor speed.
3. 18 experiments under different conditions.
4. Mean experiment time is 136.6389 seconds.
5. 14 experiments are used for training and 4 for testing.

Experimental Setup

1. Simulink model generates voltages, currents and torque.
2. PyTorch for network implementation.
3. Dataset scaled between $(-1, 1)$.
4. Simulink gives data at 20KHz, downsample to 100Hz.
5. Splitting data, window size: 60 at stride: 10.
6. Mean Square Error (MSE) to evaluate.

ANN for signal prediction

1. Three outputs, three networks.
2. Input: $180, 60 \times 3$
3. Output: 60
4. Architecture: 3 Layers, 180x360 -¿ 360x120 -¿ 120X60
5. Current1 model MSE: 0.043
6. Current2 model MSE: 0.105
7. Torque model MSE: 0.564

Model	Current1	Current2
Torque MSE		

LSTM for signal prediction

Three outputs, three networks.

Input: 60×3

Output: 60

Current1 model MSE: 0.043

Current2 model MSE: 0.105

Torque model MSE: 0.564

Multiple models or Single model?

Conclusions

Thank you
Questions?