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

Sagar Verma

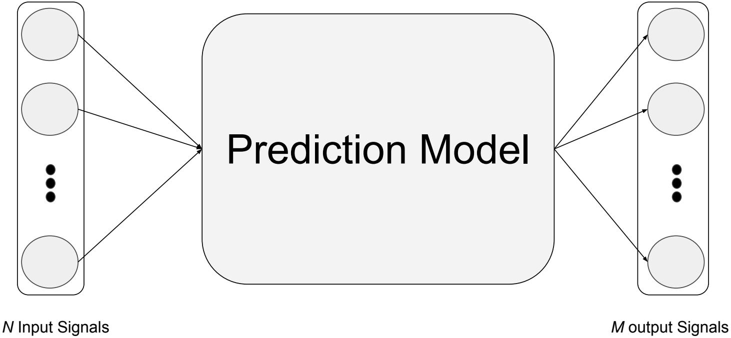
Centre de Vision Numérique,
Centralesupélec, Gif-sur-Yvette

November, 2018

Table Of Contents

Problem Statement

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



Problem Statement

Challenges

1. How to handle continuous 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

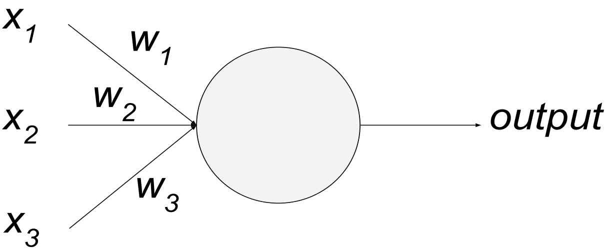
1. Perceptrons
2. Feed-forward Networks
3. Activation Functions
4. Loss Functions
5. Learning Weights
6. Sequential Networks

Background

Perceptrons

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

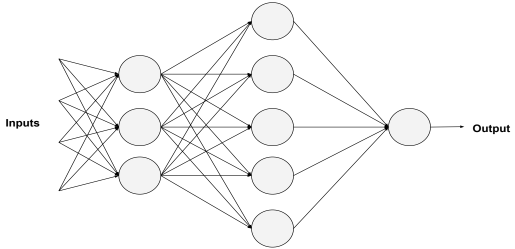
$$\text{output} = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{ threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{ threshold} \end{cases} . \tag{1}$$



Background

Feed-forword Networks

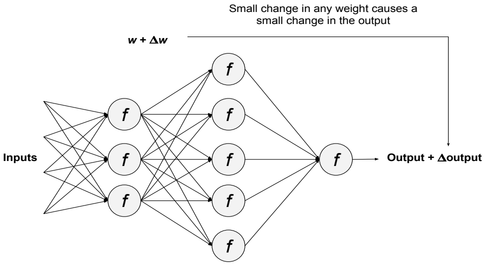
1. AKA Artificial Neural Networks (ANNs)
2. Multiple stacked perceptrons
3. #connections increases with #perceptrons
4. Learning weights is difficult
    4.1 Binary output does not tell us which weights are important and which are not
    4.2 A small change in input or weight could flip the output from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



Background

Activation Functions

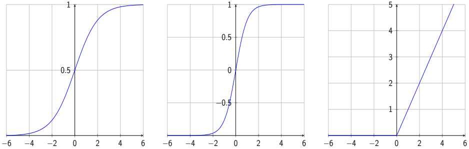
1. To learn weights of an ANN we need continuous output.
2. Function that can give small changes in output when small changes in weights are made.
3. Should give a smooth output.
4. f should be non-linear, $y = f(w^T x + b)$



Background

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): $f(x) = \begin{cases} 0 & \text{if } \sum_i w_i x_i < 0 \\ x & \text{if } \sum_i w_i x_i \geq 0 \end{cases}$



Sigmoid, Tanh, and ReLU

Background

Loss Functions

1. Tells us how good or bad network is.
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)
   $\mathcal{L}(y_{true}, y_{pred}) = -\sum_{c=1}^{C} y_{true}^c log(y_{pred}^c)$

Background

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

Background

Learning Weights

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

Background

Learning Weights: Optimizers

1. Can not 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.

Background

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

Background

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).

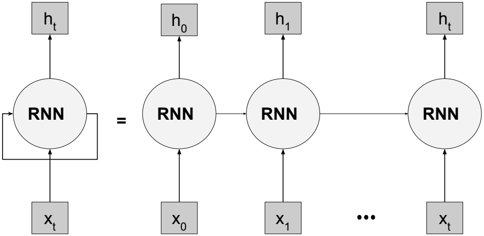


Figure: RNN unrolled in time.

Background

Sequential Networks: 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.

Background

Sequential Networks: Long-Short Term Memory

1. RNNs have vanishing and exploding gradients problem.
2. LSTM resolves above problems.
3. Computes when to forget and when to remember.

Neural networks for motor control

Dataset and Experiments

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.

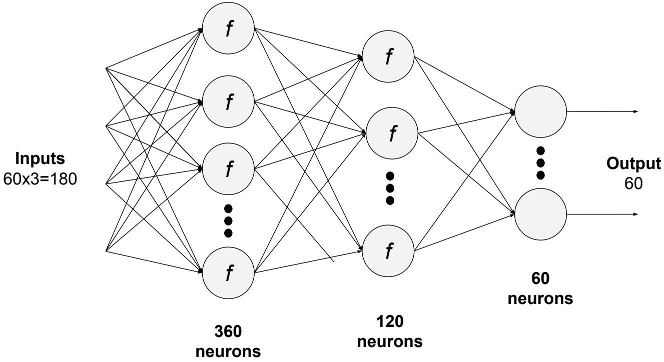
Dataset and Experiments
Experimental Setup

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. Window size, w: 30, 60, and 100 at stride s: 10.
6. Mean Square Error (MSE) to evaluate.

Dataset and Experiments

ANN for signal prediction

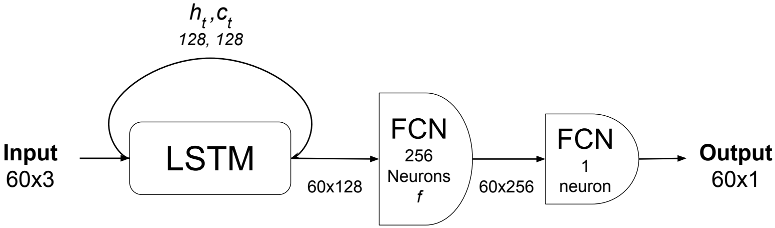
1. Three outputs, three networks.
2. Input: $w \times 3$ 1-D vector, Output: w
3. Activation, f: tanh



Dataset and Experiments

LSTM for signal prediction

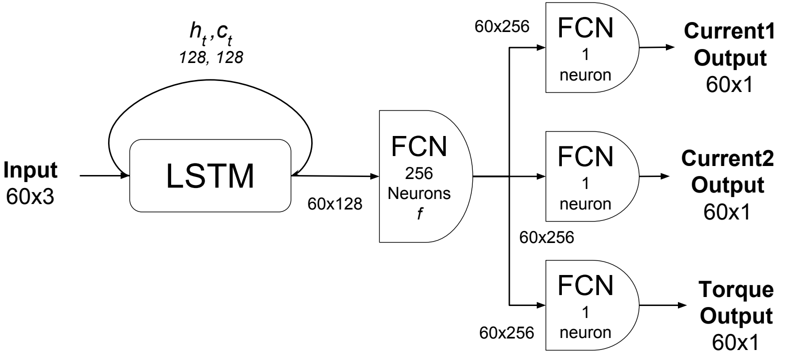
1. Three outputs, three networks.
2. Input: $w \times 3$ 2-D vector, Output: w
3. Activation, f: tanh



Dataset and Experiments

Multi-Output LSTM model

1. Input: $w \times 3$ 2-D vector, Output: $w \times 3$
2. Activation, f: tanh



Results and Conclusions

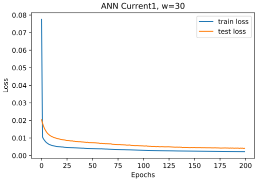
Best Model

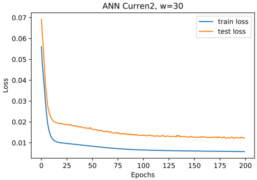
Model	w	Current1	Current2	Torque
ANN	100	0.072	0.197	0.672
	60	0.043	0.105	0.564
	30	**0.031**	**0.091**	**0.056**
LSTM	100	0.146	0.182	0.723
	60	0.136	0.107	0.688
	30	0.045	0.105	0.072
MO-LSTM	60	0.139	0.112	0.691
	30	0.051	0.109	0.081

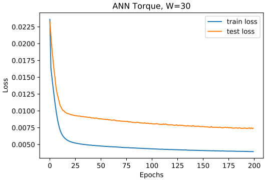
Table: MSE of different models with different sequence lengths.

Results and Conclusions

Model Convergence







Results and Conclusions

Example Outputs

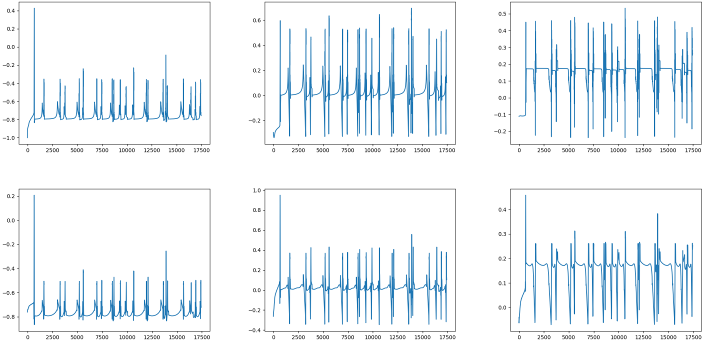


Figure: Top row: ground truth, bottom row: predicted signal, left to right: current1, current2, and torque

Results and Conclusions

Some Answers

1. How to handle continous signals? **Use tanh**
2. How to handle long sequences? **Find optimal subsequence**
3. Which prediction model to use? **ANN**
4. How to handle multiple outputs? **Independent Models**

Thank you!
Questions?