Machine Learning (ML) is becoming more and more accessible with numerous quality free courses available, growing cheap computing resources, commercial-grade tools like Google’s Tensorflow and Microsoft’s Cognitive Toolkit that are available to anyone for free.

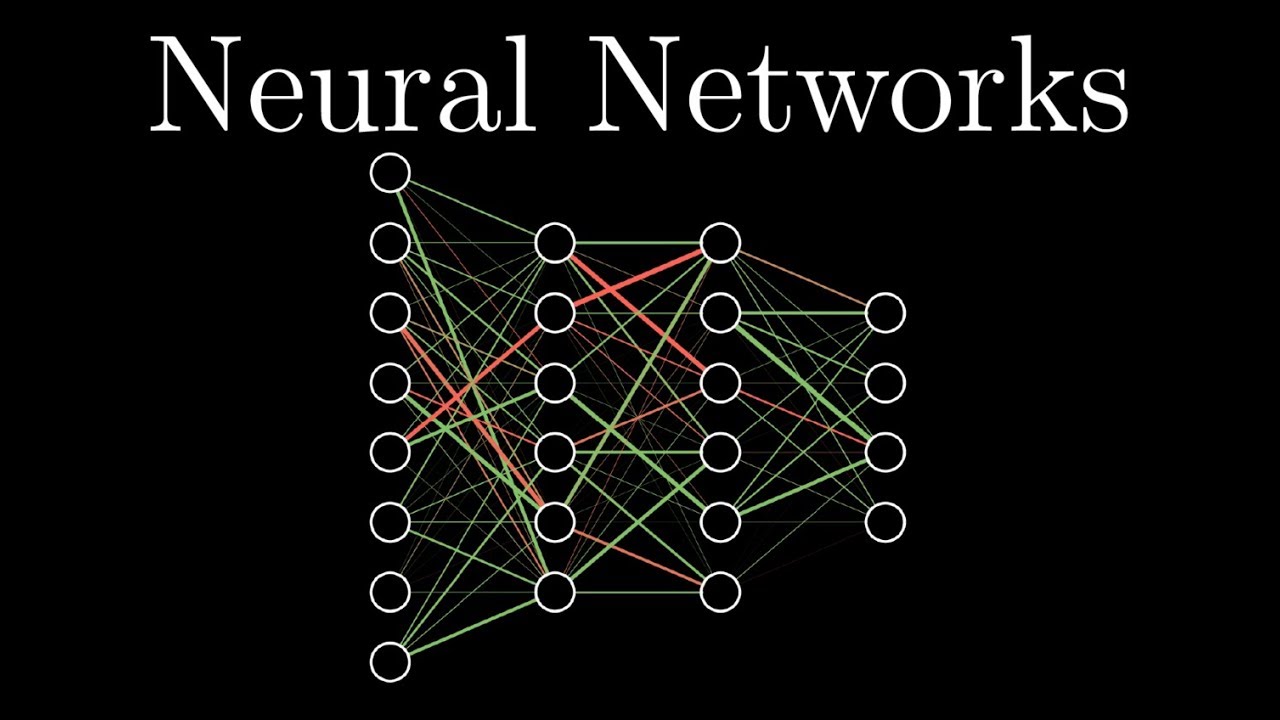
In one sentence – ML is just really fancy curve fitting.



The original work by Ci Rong implemented linear classifier, where specific frequency values that were observed to change the most between sharp and worn tool were separated by a simple linear boundary.

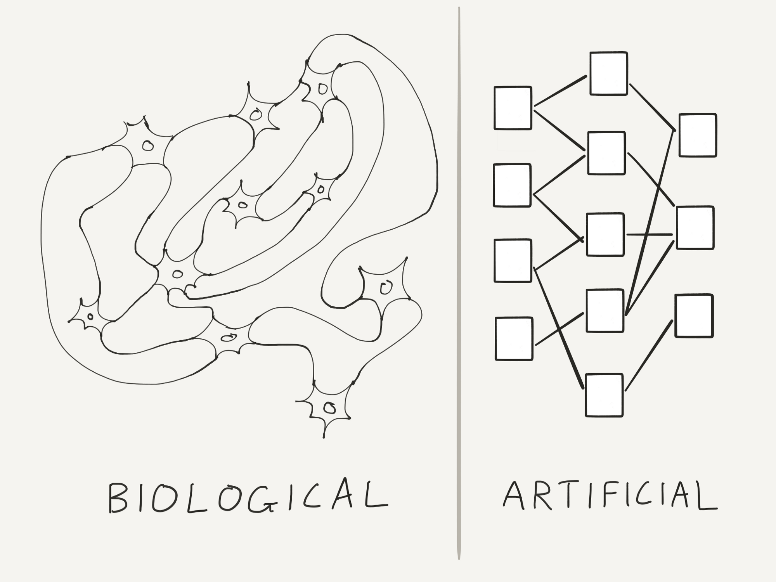


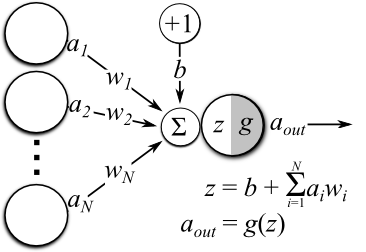
The results were promising, however data needed to be expertly processed thoroughly understood.



Neural Networks are more computationally intensive, but they may require less user input. They allow calculation of complex non-linear data relationships without having to manually iterate any condition changes.

If an expert in the field can use the data, then generally a Neural Network can be trained for the same purpose.



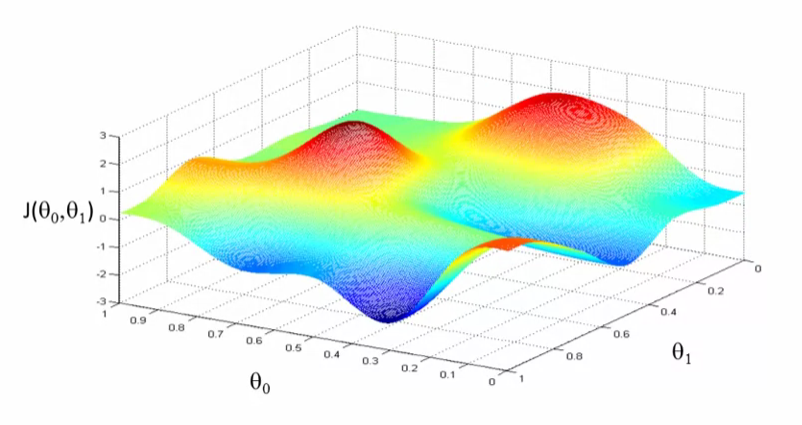


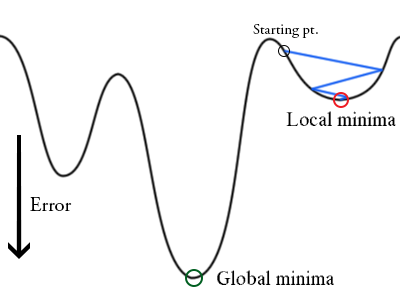
Supervised learning sequential model.

A Neural Network ‘learns’ by adjusting the weights/coefficients associated with inputs. In this way data that seems to changes without effect on the desired metric is given small coefficients and doesn’t skew the results.

The training algorithm is initialized with random non-symmetric weights. Then real data is used to calculate the outcome. The difference between initial network computation and real observed condition is used to adjust weights (each weight should be slightly different so the network can understand better which neuron the error came from most).

Each training iteration is about computing what the network currently outputs and then changing it based on error. How big the error is can be visualized as a function of weights. In this sense, we need to minimize the error by looking at the gradient of such plot and try to descend into global minima.



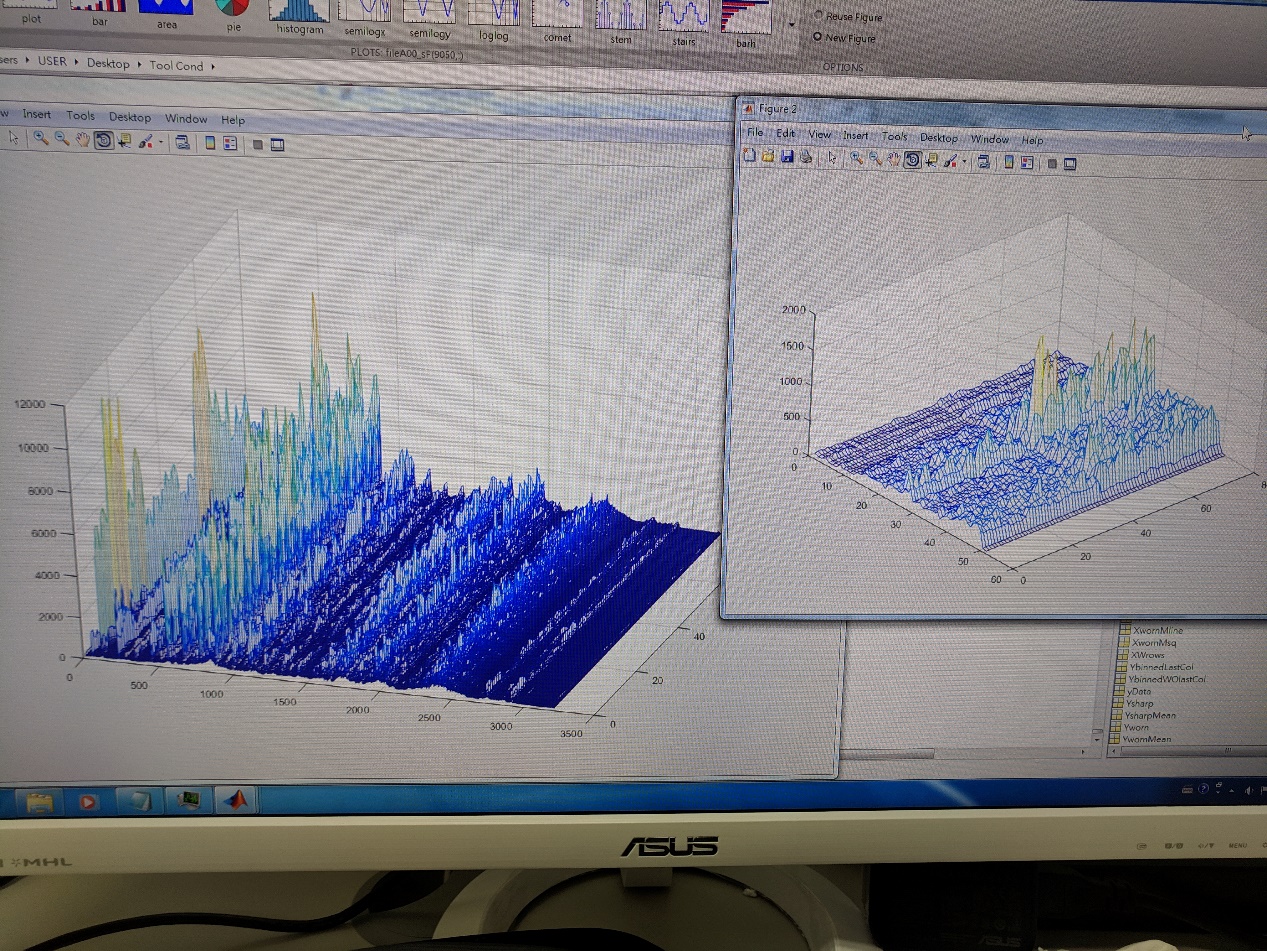


Depending on the initial starting point it is possible to never reach the best solution, but usually local minima is good enough according to experts in the field.

Matlab prototype

<https://www.coursera.org/learn/machine-learning> - recommended learning

Originally I’ve only received frequency data with two states – sharp and worn. FFT was already applied to it. I’ve also used Ci Rong’s Mean Scatter Criteria function on it to obtain a visual representation of what Neural Network is supposed to recognize.



The above picture shows clearly the consistent difference in energy between sharp and worn tools.

Understanding data structure is first step to designing a NN. It may not always be possible to visualise the pattern or even understand it. At minimum we must define number of inputs and assign an outcome for supervised learning.

Number of inputs should stay consistent, but in some designs it doesn’t have to be.

## Python steps:

1. Understand data, if not able to analyse, know dimensions, number of inputs, outputs, desired classification.
2. Create model
3. # input variables denoting the features and label data
4. feature = C.input\_variable(input\_dim) # C. is CNTK
5. label = C.input\_variable(out\_classes)
6. # Instantiate the feedforward classification model
7. def create\_model(features):
8. with C.layers.default\_options(init = C.layers.glorot\_uniform(), activation = C.ops.relu):
9. h = features
10. for \_ in range(num\_hidden\_layers):
11. h = C.layers.Dense(hidden\_layers\_dim)(h)
12. r = C.layers.Dense(out\_classes, activation = None)(h)
13. return r
14. z = create\_model(feature)
15. # Another way (KERAS) to create the model, which is simpler:
16. # my\_model = C.layers.Sequential ([
17. # Dense(hidden\_layers\_dim, activation=C.ops.relu),
18. # Dense(hidden\_layers\_dim, activation=C.ops.relu),
19. # Dense(out\_classes, activation=None)])
20. # z = my\_model(feature)

3.a Create loss and metric functions used to train the model

loss = C.losses.cross\_entropy\_with\_softmax(z, label)

label\_error = C.classification\_error(z, label)

3.b Create learning parameters and learning algorithm used to train the model

# Instantiate the trainer object to drive the model training

lr\_schedule = C.learning\_parameter\_schedule(learning\_rate, minibatch\_size=num\_mb\_iter) #C.learners.IGNORE

learner = C.sgd(z.parameters, lr\_schedule) # stochastic gradient descent/minibatch descent

3.c Create trainer object that coordinates model training

trainer = C.Trainer(z, (loss, label\_error), [learner], prog\_printer)

4. Create reader stream that reads data in from .ctf file or load data into numpy array. Reader stream:

# Read a CTF formatted text using the CTF deserializer, from a .ctf file

def create\_reader(path, is\_training, input\_dim, num\_label\_classes):

return C.io.MinibatchSource(C.io.CTFDeserializer(path, C.io.StreamDefs(

labels = C.io.StreamDef(field='labels', shape=num\_label\_classes, is\_sparse=False),

features = C.io.StreamDef(field='features', shape=input\_dim, is\_sparse=False)

)), randomize = is\_training, max\_sweeps = C.io.INFINITELY\_REPEAT if is\_training else 1)

# Create the reader to training data set

reader\_train = create\_reader(dataPath\_train, True, input\_dim, out\_classes)

# Map the data streams to the input and labels.

input\_map = {

label : reader\_train.streams.labels,

feature : reader\_train.streams.features

}

z.save(model\_output\_file)

5. Loop the following:

- Get next data batch from the stream

- Pass the data to the trainer

Optional: Run test on unseen data from a different stream/array and determine accuracy periodically. If the number of samples is low, NN can easily overfit the data and can produce worse results on unseen data. This can be avoided is # of samples is high.

6. Save the best model.

To use the model in python, simply create data stream, load model, and run evaluation function.

z = C.load\_model(model\_file)

out = C.softmax(z) # assigns probability based values

predictions = out.eval({out.arguments[0]: features})