

Using Machine Learning to tell if a Spider Died

Jack Bosco

February 14, 2023

Goal

- Want to learn about circadian rhythms in spiders
 - How does a usual (12L:12D) cycle compare to a (12D:12D) cycle?
- We have large sets of data to help us determine this

```
1111100000001111110001101010100011111011000010010011010101000101111011010100000
1110001111000101000111111001101010001001111001100001001111101000011100000011
00111001110000001001111101111010100100111000110110101101010010100001001010000
111111110111110001100000010011101001111101100111100110101110011111000111001
000101010011100001111100001000011011111100001110000100000111110001101111
000011010011101011101111001001100001010010001110001001000100001001000100001011001
110101110000000101000010100101010011010110101111011100100000001100010110000
0001100100110010111101111011110000101011000010101010100011001010001101010100
10001000000010110100101010010011011001010100111011001011110000110010001000011
00111001101100111100000101010010001000101000011010101001010101110010111010101
000110101101010000101011110110011000111011011010000111100000000011000001101100
000100100111111101000011001010000100101010001100110000110001001100011010111111
011110101000101100100110000010110101010010001101111000000100110000000011110001
00110111000010111100001101010100101111101111010000110101010010001101010101110
00000100010011100010110011000001011010001101101101000100000111111001100101000
010101110110010101010100000011010111010000010011010100011101001000110110010010
00101010000101010010110001110011000111110100001111000011100110010001101001001001
00111000010111000001101011110100100001100101110010000100110010011100001101111111
10111110000111000101000001010010011110010001010110101011110100110100001101001
000111100000101101111110010001011110010011000110010011011101000011100010010
1000000100001111000010011000110100110101011011110110111101001011010010110101
000011110101001110101110011100101100000011011111001101101110100001111110110011
0000100111110011110000100001110111001001100100001100100011001010001011001100
101101011110100010000000110011111001000001011010011001110011011000010101000111
0110011110010101010111100111001000100110010001110101011001111010010101001010
001001001010100100011011100111011000101111001101100100110000000111010110000100
01010100000000010111101101000110110011011001011010101110100001010000000101100
111100110010101000100011100010101111100100010110001101010001011001010000100000
10011100100100001000011011110110011001001100011000100011010111101000100001100
0001011010001010101011110011000110101010010001101001110000101011000010010011111
011000000101100011011010010010110010110001010101010011000010101111011001010101
```

Problem

Null Character

Inaction (spider not doing anything) is coded similarly to sleeping.

Actually, its exactly the same



The spider is not Dead



The spider died

Problem Redefined

Dead|Not-Dead

A classification algorithm *should* be able to use pattern matching and see when or if a spider died.

For an input string of n 0s or 1s...

- Action Space: n . We should be able to point (somewhat accurately) to the moment when the spider died.
- State Space: 2^n . Every possible arrangement of 0s and 1s needs to be classifiable. This number is HUGE.

Our state and action space makes training **Dead|Not-Dead** non-trivial.

Can we do it?



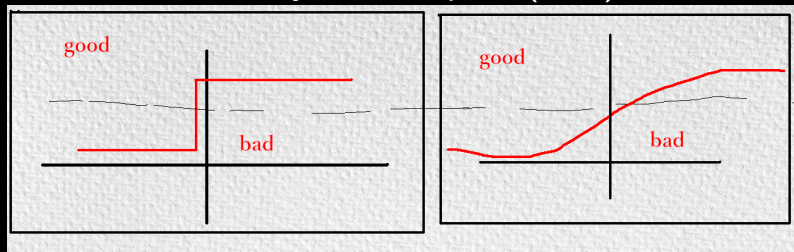
Yes

If the problem is solvable, then a perceptron can learn to solve it
But we need to be smart about it...



Solution 1: 'Meh' Classifier

Multilayered Perception (MLP)



- $\sigma \rightarrow$ outputs on a gradient
 - 0.0 \rightarrow The spider died (earlier)
 - 1.0 \rightarrow The the spider is totally still alive
 - $0.25 < x < 0.75 \rightarrow$ 'Meh'

Solution 1: 'Meh' Classifier

Pros

- Simple, understandable, we've had these since 1989
- I can (and have) implemented this with only numpy
- quick/inexpensive

Cons

- results may be uninformative
- Vanishing Gradient Problem
 - we can't do better than 'meh'

Pre-Classification

```
[1111000010100100001000000100000000000000000111111100001000]
```

Day 1

Night 1

Day 2

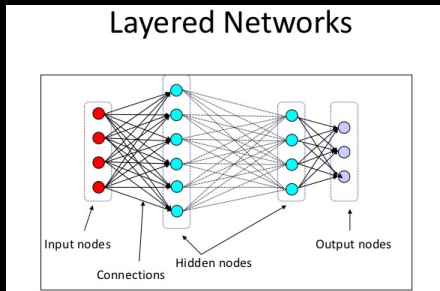
Night 2

- split the dataset into m groups
- implement m hidden layers for the perception and do **much** better than 'meh'
- Unfortunately, with sigmoids this is not possible (Vanishing Gradient)

Solution 2: We Go Deeper

Rectified Linear Activation Function (ReLU)

A piecewise function with constant derivative (linear). It's just a line with a cutoff!



Overhaul sigmoidals and use a ReLU instead

Solution 2: We Go Deeper

Pros

- more informative and reliable output
- I still have a thorough understanding of how it works

Cons

- more hidden layers are more expensive to train

PyTorch

PyTorch Package for Python

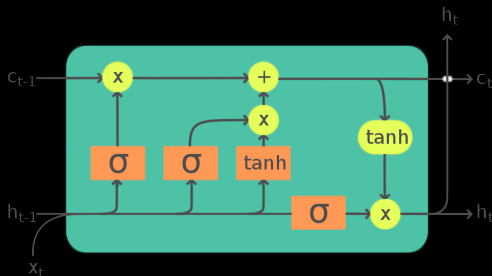
“An open source machine learning framework that accelerates the path from research prototyping to production deployment”



```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

PyTorch makes it easy to build things I don't understand!

Solution 3: LSTM



- Recursive (not feedforward) Neural Net
 - feedback connections allow for processing entire sequences of data - Wikipedia
- Sigmoidal - but somehow still works
 - solves Vanishing Gradient problem by allowing gradients to flow *unchanged* - Wikipedia

Solution 3: LSTM

Pros

- invented in 1995
- Bill Gates likes it
- OpenAI and Deepmind use it
- made for problems like ours

Cons

- might be expensive
- going to require more research
- I definitely couldn't make this without PyTorch