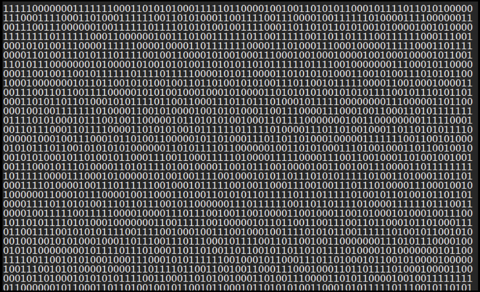
Using Machine Learning to tell if a Spider Died

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



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?

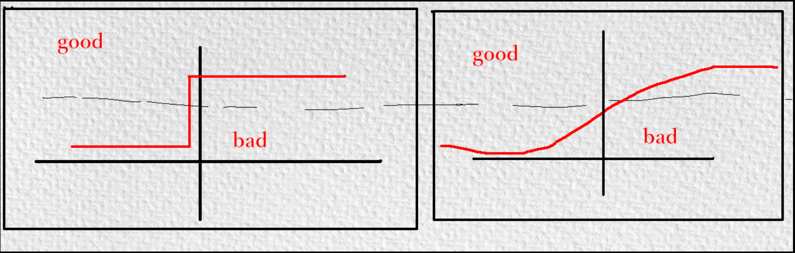
Yes

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)

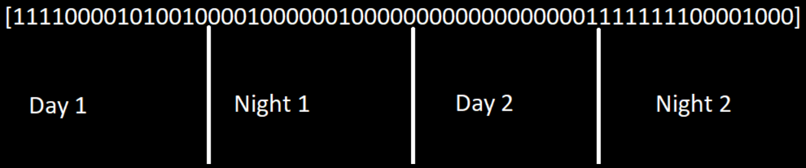


- $\sigma \rightarrow$ outputs on a gradient
  - $0.0 \rightarrow$ The spider died (earlier)
  - $1.0 \rightarrow$ The the spider is totally still alive
  - $0.25 < x < 0.75 \rightarrow$ ‘Meh’

Solution 1: 'Meh' Classifier

Pros	Cons
▪ Simple, understandable, we've had these since 1989 ▪ I can (and have) implemented this with only numpy ▪ quick/inexpensive	▪ results may be uninformative ▪ Vanishing Gradient Problem - we can't to better than 'meh'

Pre-Classification

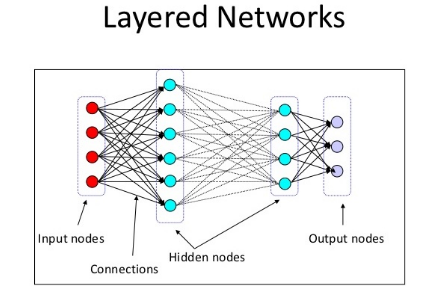


- split the dataset into m groups
- implement m hidden layers for the perception and do **much** better than ‘meh’
- Unfortunately, with sigmoidals this is not possible (Vanishing Gradient)

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	Cons
■ more informative and reliable output ■ I still have a thourough understanding of how it works	■ more hidden layers are more expensive to train

PyTorch

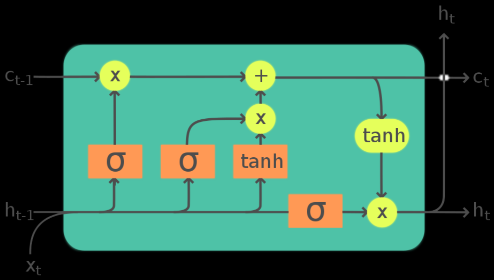
PyTorch Package for Python

"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	Cons
invented in 1995	might be expensive
Bill Gates likes it	going to require more research
OpenAI and Deepmind use it	I definitely couldn't make this without PyTorch
made for problems like ours	