Neural networks (Ch. 18)
Announcements

Final this Saturday (5/11) at 10:30am in this room

Projects due next Tuesday (5/14) at 11:55pm

Writing 4 regrades also due next Tuesday (5/14) at 11:55pm (email them to me)
Computer science is fundamentally a creative process: building new & interesting algorithms.

As with other creative processes, this involves mixing ideas together from various places.

Neural networks get their inspiration from how brains work at a fundamental level (simplification... of course).
Biology: brains

(Disclaimer: I am not a neuroscience-person)

Brains receive small chemical signals at the “input” side, if there are enough inputs to “activate” it signals an “output”
An analogy is sleeping: when you are asleep, minor sounds will not wake you up.

However, specific sounds in combination with their volume will wake you up.
Biology: brains

Other sounds might help you go to sleep (my majestic voice?)

Many babies tend to sleep better with “white noise” and some people like the TV/radio on
Neural networks are connected nodes, which can be arranged into layers (more on this later).

First is an example of a perceptron, the most simple NN; a single node on a single layer.
Neural networks are connected nodes, which can be arranged into layers (more on this later).

First is an example of a perceptron, the most simple NN; a single node on a single layer.
Mammals

Let's do an example with mammals...

First the definition of a mammal (wikipedia):

Mammals [posses]:
(1) a neocortex (a region of the brain),
(2) hair,
(3) three middle ear bones,
(4) and mammary glands
Mammals

Common mammal misconceptions:
(1) Warm-blooded
(2) Does not lay eggs

Let's talk dolphins for one second.

Dolphins have hair (technically) for the first week after birth, then lose it for the rest of life... I will count this as “not covered in hair”
Consider this example: we want to classify whether or not an animal is mammal via a perceptron (weighted evaluation)

We will evaluate on:
1. Warm blooded? (WB) Weight = 2
2. Lays eggs? (LE) Weight = -2
3. Covered hair? (CH) Weight = 3

If($2 \cdot WB - 2 \cdot LE + 3 \cdot CH > 1$) $\Rightarrow$ Mammal
Consider the following animals:
Humans \( \{WB=y, LE=n, CH=y\} \), mam=y
\[
2(1) + -2(-1) + 3(1) = 7 > 1 \quad \text{... Correct!}
\]
Bat \( \{WB=\text{sorta}, LE=n, CH=y\} \), mam=y
\[
2(0.5) + -2(-1) + 3(1) = 6 > 1 \quad \text{... Correct!}
\]
What about these?
Platypus \( \{WB=y, LE=y, CH=y\} \), mam=y
Dolphin \( \{WB=y, LE=n, CH=n\} \), mam=y
Fish \( \{WB=n, LE=y, CH=n\} \), mam=n
Birds \( \{WB=y, LE=y, CH=n\} \), mam=n
But wait... what is the general form of:

\[ w_1 x + w_2 \cdot y + w_3 \cdot z > c \]
Perceptrons

But wait... what is the general form of:

$$w_1 x + w_2 \cdot y + w_3 \cdot z > c$$

This is simply one side of a plane in 3D, so this is trying to classify all possible points using a single plane...
Perceptrons

If we had only 2 inputs, it would be everything above a line in 2D, but consider XOR on right

There is no way a line can possibly classify this (limitation of perceptron)
Neural network: feed-forward

Today we will look at feed-forward NN, where information flows in a single direction.

Recurrent networks can have outputs of one node loop back to inputs as previous.

This can cause the NN to not converge on an answer (ask it the same question and it will respond differently) and also has to maintain some “initial state” (all around messy).
Neural network: feed-forward

Let's expand our mammal classification to 5 nodes in 3 layers (weights on edges):

if Output(Node 5) > 0, guess mammal
You try Bat on this: \{WB=0, LE=-1, CH=1\}

Assume (for now) output = sum input

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

Output is -7, so bats are not mammal... Oops...

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

In fact, this is no better than our 1 node NN

This is because we simply output a linear combination of weights into a linear function (i.e. if \( f(x) \) and \( g(x) \) are linear... then \( g(x) + f(x) \) is also linear)

Ideally, we want a activation function that has a limited range so large signals do not always dominate
Neural network: feed-forward

One commonly used function is the sigmoid:

\[ S(x) = \frac{1}{1 + e^{-x}} \]
Back-propagation

The neural network is as good as its structure and weights on edges

Structure we will ignore (more complex), but there is an automated way to learn weights

Whenever a NN incorrectly answer a problem, the weights play a “blame game”...
- Weights that have a big impact to the wrong answer are reduced
Consider this example: 4 nodes, 2 layers

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
in_1| N1 |
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
N2| N3 |
|   |   |
in_2| N4 |
|   |   |
1  |   |
```

This node as a constant bias of 1

Example from: [https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/]
Node 1: $0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$ input. Thus it outputs (all edges) $S(0.3775) = 0.59327$
Eventually we get: $\text{out}_1 = 0.751$, $\text{out}_2 = 0.773$

Suppose wanted: $\text{out}_1 = 0.01$, $\text{out}_2 = 0.99$
Back-propagation

We will define the error as: $\frac{1}{2} \sum_i (\text{correct}_i - \text{output}_i)^2$
(you will see why shortly)

Suppose we want to find how much $w_5$ is to blame for our incorrectness

We then need to find: $\frac{\partial \text{Error}}{\partial w_5}$

Apply the chain rule:

$\frac{\partial \text{Error}}{\partial \text{out}_1} \cdot \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} \cdot \frac{\partial \text{In}(N_3)}{\partial w_5}$
Back-propagation

\[ \text{Error} = \sum_i \frac{(\text{correct}_i - \text{output}_i)^2}{2} \]
\[ \frac{\partial \text{Error}}{\partial \text{out}_1} = -(\text{correct}_1 - \text{out}_1) \]
\[ = -(0.01 - 0.751) = 0.741 \]

As \( S'(x) = S(x) \cdot (1 - S(x)) \)

\[ \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} = S(\text{In}(N_3)) \cdot (1 - S(\text{In}(N_3))) \]
\[ = 0.751 \cdot (1 - 0.751) = 0.187 \]

\[ \frac{\partial \text{In}(N_3)}{\partial w_5} = \frac{\partial w_5 \cdot \text{Out}(N_1) + w_6 \cdot \text{Out}(N_2) + b_2 \cdot 1}{\partial w_5} \]
\[ = \text{Out}(N_1) = 0.593 \]

Thus, \( \frac{\partial \text{Error}}{\partial w_5} = 0.593 \cdot 0.187 \cdot 0.741 = 0.0822 \)
Now that we know $w_5$ is 0.08217 part responsible, we update the weight by:

$$w_5 \leftarrow w_5 - \alpha \times 0.0822 = 0.3959 \text{ (from 0.4)}$$

$\alpha$ is learning rate, set to 0.5
Back-propagation

For $w_1$ it would look like:

(book describes how to dynamic program this)
Back-propagation

Specifically for $w_1$ you would get:

$$\frac{\partial \text{Error}}{\partial S(\text{In}(N_1))} = \frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} + \frac{\partial \text{Error}_2}{\partial S(\text{In}(N_1))}$$

$$\frac{\partial S(\text{In}(N_1))}{\partial \text{In}(N_1)} = S(\text{In}(N_1)) \cdot (1 - S(\text{In}(N_1)))$$

$$= 0.593 \cdot (1 - 0.593) = 0.241$$

$$\frac{\partial \text{In}(N_3)}{\partial w_5} = \frac{\partial \text{In}_1 w_1 + \text{In}_2 w_2 + b_1 \cdot 1}{\partial w_5}$$

$$= \text{In}_1 = 0.05$$

Next we have to break down the top equation...
Back-propagation

\[
\frac{\partial \text{Error}}{\partial S(\text{In}(N_1))} = \frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} + \frac{\partial \text{Error}_2}{\partial S(\text{In}(N_1))}
\]

\[
\frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} = \frac{\partial \text{Error}_1}{\partial S(\text{In}(N_3))} \cdot \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} \cdot \frac{\partial \text{In}(N_3)}{\partial S(\text{In}(N_1))}
\]

From before...
\[
\frac{\partial \text{Error}_1}{\partial S(\text{In}(N_3))} \cdot \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} = 0.593 \cdot 0.187 = 0.111
\]

\[
\frac{\partial \text{In}(N_3)}{\partial S(\text{In}(N_1))} = \frac{\partial w_5 \cdot S(\text{In}(N_1)) + w_6 \cdot S(\text{In}(N_2)) + b_1 \cdot 1}{\partial S(\text{In}(N_1))} = w_5 = 0.4
\]

Thus,
\[
\frac{\partial \text{Error}_1}{\partial S(\text{In}(N_1))} = 0.111 \cdot 0.4 = 0.0444
\]
Back-propagation

Similarly for Error\(_2\) we get:

\[
\frac{\partial Error}{\partial S(In(N_1))} = \frac{\partial Error_1}{\partial S(In(N_1))} + \frac{\partial Error_2}{\partial S(In(N_1))} \\
= 0.0444 + -0.0190 = 0.0254
\]

Thus, \(\frac{\partial Error}{\partial w_1} = 0.0254 \cdot 0.241 \cdot 0.05 = 0.000306\)

Update \(w_1 \leftarrow w_1 - \alpha \frac{\partial Error}{\partial w_1} = 0.15 - 0.5 \cdot 0.000306 = 0.1498\)

You might notice this is small...
This is an issue with neural networks, deeper the network the less earlier nodes update
Despite this learning shortcoming, NN are useful in a wide range of applications:

- Reading handwriting
- Playing games
- Face detection
- Economic predictions

Neural networks can also be very powerful when combined with other techniques (genetic algorithms, search techniques, ...).
NN examples

Examples:
https://www.youtube.com/watch?v=umRdt3zGgpU
https://www.youtube.com/watch?v=qv6UVOQ0F44
https://www.youtube.com/watch?v=xcIBoPuNlw
https://www.youtube.com/watch?v=0Str0Rdkxxo
https://www.youtube.com/watch?v=l2_CPB0uBkc
https://www.youtube.com/watch?v=0VTI1BBLydE
AlphaGo/Zero has been in the news recently, and is also based on neural networks.

AlphaGo uses Monte-Carlo tree search guided by the neural network to prune useless parts.

Often limiting Monte-Carlo in a static way reduces the effectiveness, much like mid-state evaluations can limit algorithm effectiveness.
NN examples

Basically, AlphaGo uses a neural network to “prune” parts for a Monte-carlo search