Deep Learning

Pietro Marcatti

First Semester 2022/2023

1 History of Deep Learning

1.1 Perceptron

The perceptron algorithm was invented in 1958. The perceptron became the first model for binary classification. It has one weight w_i per input x_i. If the result is larger than a threshold it returns 1 otherwise 0 or -1 (non linearity ?). To train a perceptron we repeat the following steps:

- Initialize weigths randomly
- Take one sample x_i and predict y_i
- For erroneous predictions update weights
  - If prediction $y = 0$ and ground truth $y_i = 1$, increase the weights
  - If prediction $y = 1$ and ground truth $y_i = 0$, decrease the weights
- Repeat until no errors are made

However the perceptron can't solve a simple, although non-linear, problem such as the XOR. To improve on the perceptron model you must add new layers but there was a stagnation on the neural networks research. The stagnation was caused by a lack of motivation from the community due to the discouraging results of the first perceptron models. Still during the AI winter a couple important findings were published such as back-propagation and recurrent neural networks.

In 2009 the ImageNet dataset was published. It colleted images for each of the 100k terms in WordNet (16M images in total). Terms were organized hierearchally, es: Vehicle →Ambulance. The ImageNet challange was instituted: 1 million images, 1000 classes, top-5 and top-1 error measured. To build ImageNet they started colleting candidate images from the internet. They then classified the candidates with Amazon Mechanical Turk service.
A more recent important achievement was the one obtained by AlphaGo a deep learning model, based on reinforced learning, that in 2016 defeated the best Go player.

Deep learning is the first class of learning algorithms that is scalable: performance just keeps getting better as you feed them more data. Instead when working on a small amount of data the performance of a traditional learning model (logistic regression, SVM, decision tree etc) is better.
The three key factors for deep learning scaling are:

- Data
- Computation/hardware
- Algorithms

2 Logistic Regression

Let's start with a simple two feature model:

- x_1 number of lectures you attend
- x_2 hours spent on the laboratory activities

With logistic regression we want to learn a probabilistic function:

$$\hat{y} = P(y = 1|x)$$

In particular the goal is to find the parameters w and b of the following function (hypothesis).

$$H_{w,b}(x) = g = (w^T \cdot x + b) = \frac{1}{1 + e^{-(w^T \cdot x + b)}}$$

where $g(z)$ is the sigmoid function so that:

$$\begin{cases} H_{w,b}(x) >= 0.5 & \text{if } y = 1 \\ H_w, b(x) < 0.5 & \text{if } y = 0 \end{cases}$$

To get our discrete classification we map the output of the hypthesis function as follow:

$$\begin{cases} H_{w,b}(x) >= 0.5 & \rightarrow \text{"1"} \\ H_w, b(x) < 0.5 & \rightarrow \text{"0"} \end{cases}$$

The decision boundary is $H_{w,b}(x) = 0.5 \rightarrow w^T \cdot x + b = 0 \rightarrow -3 + x_1 + 2x_2$ supposing we have $b = 3$ and $w = [1, 2]$. The hypotesis function is > 0.5 when the argument is > 0, that is becuause of the shape and output of the sigmoid.

2.1 Cost Function

To find w and b so that:

$$\begin{cases} H_{w,b}(x) >= 0.5 & \text{if } y = 1 \\ H_w, b(x) < 0.5 & \text{if } y = 0 \end{cases}$$

the logistic classifirer defines the following cost function:

$$J(w,b) = \frac{1}{m} \cdot \sum_{i=1}^{m} Cost(h_{w,b}(x^i), y^i) \tag{1}$$

$$Cost(h_{w,b}(x^i), y^i) = -y^i \cdot ln(h_{w,b}(x^i)) - (1 - y^i) \cdot ln(1 - h_{w,b}(x^i)) \tag{2}$$

This cost function or loss function is convex and is derivable respect to w and b In general we call the function to learn the **hypotesis** but in deep learning it's also called **model**, the **cost function** in deep learning is also called **loss function**.

Why is random initialization of weights is important? If we used 0 the first hidden layer would all be 0, because we multiplied all inputs for 0.
This results that the partial derivates for all nodes in the hidden layers are equal, this is called the symmetry problem.

We divide our training data set into smaller batches of usually around 16-32-64 samples. We compute forward and backwards on every single sample. Once we complete all the batches we completed an epoch. We can then start again but now, on the first batch, our neural network will have weights that will have already changed thanks to all other batches

Regularization

Logistic Regression: Minimization problem with regularization: $\min_{w,b} J(w,b)$

$$J(w,b) = \frac{1}{m} \left[\sum_{i=1}^{m} Cost(h_{w,b}(x^{(i)}), y^{(i)}) + \frac{\lambda}{2} \sum_{j=1}^{n} w_i^2 \right]$$

λ is called the regularization parameter, usually b is ignored in the regularization process. By setting a big regularization parameter we are saying that our minimization algorithm should focus on reducing the weights. The goal is to have the weights all in the same order of magnitude.

Doesn't regularization kill the importance of some features over others?

Regularization with Neural Networks:

$$J(W^{[1]}, b^{[1]} W^{[2]}, b^{[2]}) = \frac{1}{m} \left[\sum_{i=1}^{m} \mathbb{L}(\hat{y}, y^{(i)}) + \frac{\lambda}{2} \sum_{i=1}^{L} \left\| W^{[l]} \right\|_F^2 \right]$$

Where λ is the regularization parameter, l is the layer.

$$\left\|W^{[i]}\right\|_F^2 = \sum_{i=1}^{n[l-1]} \sum_{j=1}^{n^{[l]}} (W_{i,j}^{[l]})^2$$

Where $W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$
Regularization helps preventing overfitting because by using a big value for λ we minimize weigths close to 0 and some of them are basically dead (almost 0)

Too many dead nodes means underfitting? Would it ever be useful to have a small λ