DRP TURKEY

supported by TÜRK MATEMATİK DERNEĞİ

Deep Learning and Neural Networks

Mentee: Üzeyir Saçıkay

M.Sc. Theoretical and Mathematical Physics,
Ludwig Maximilian University of Munich

Mentor: Bengier Ülgen Kılıç

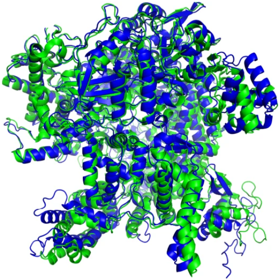
Department of Mathematics,
University at Buffalo, SUNY

Organizers

Bengier Ülgen Kılıç (University at Buffalo, SUNY)
Eda Kırımlı (University of Bristol and University of Neuchâtel)
Feride Ceren Köse (The University of Texas at Austin)
Şefika Kuzgun (University of Kansas)

We can use neural networks to do quite interesting things!

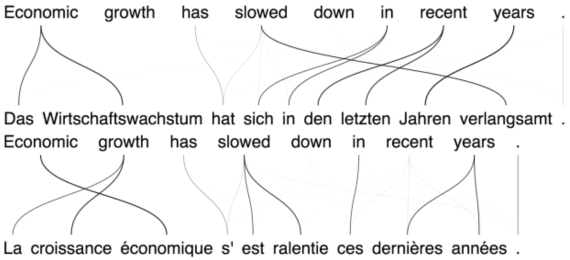
To predict the three-dimensional structure of all distinctive proteins on Earth (around 200 million!) with atomic precision,



r.m.s.d.$_{95}$ = 2.2 Å; TM-score = 0.96

To identify each individual zebra in a picture and when given different pictures of the same zebra, be able to recognize that it's the same zebra!

To accurately translate complex sentences between 133 languages!



- And to write a program that can recognize cats!

Figure 1: Cat sitting on a chair

Cat pictures app

- We want to create an app that consists of cat pictures.
- We are expecting the app to be popular, so we need a lot of cat pictures!
- How can we find a huge amount of cat pictures?

We can search for cat pictures manually and upload them to our app.

- It would be exhausting.
- We might forget the pictures we downloaded already and download the same pictures again.

We can write a program that searches "cat" on the web and downloads any picture that come up.

- How would we know that all the pictures that come up are of cats?

Below is a picture of an excavator that came up in the search!

Figure 2: An excavator of the brand CAT

Maybe we can write a program that would go through the downloaded pictures and pick out the cat pictures.

Figure 3: Cat or non-cat

We have no idea how to do that but we can outline a general "learning" plan:

1 - Make guesses

2 - Device a way of measuring how accurate our guesses are

3 - Find out the changes we need to make to make better guesses

4 - Rinse and repeat until satisfied with the guesses we are making

We still have no idea how to write such a program...

- McCulloch and Pitts invented the perceptron in 1943

We have $x_i \in \{0,1\}$ and $w_i \in \mathbb{R}$

$$y = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{threshold} \end{cases}$$

It is a simplistic model of a biological neuron.

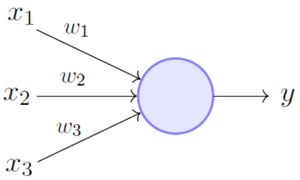


Figure 4: Perceptron with three inputs

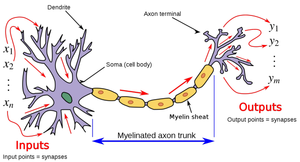
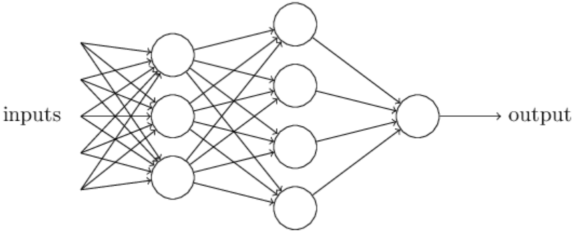
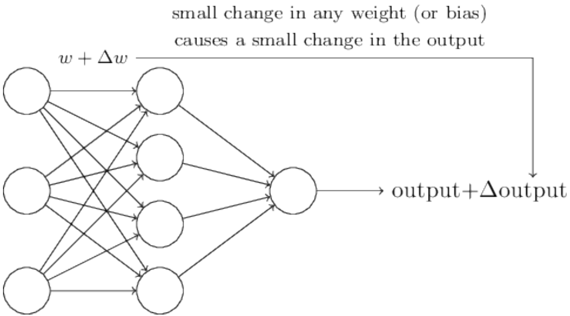


Figure 5: A biological neuron

To make things more complex, we connect multiple perceptrons end-to-end to get a network design. In a manner, we are dividing the overall decision process into subtasks. This way, we can make more subtle decisions.



It would be impossible to perform learning with perceptrons due to their discrete nature.



Getting rid of the threshold

- What if we remove the threshold condition and modify the perceptron output as just

$$y = \sum_i w_i x_i + b = w^T x + b$$

where $b = -$threshold.

- This solves the problem of discreteness but creates another problem: Any network of perceptrons would be equivalent to a single perceptron as composition of affine transformations is still an affine transformation.

Adding non-linearity

- We then arrive at the modern neuron units:

$$y = \sigma\left(w^T x + b\right)$$

  where $\sigma : \mathbb{R} \to \mathbb{R}$ is non-linear. σ is usually called the activation function.
- We can view the network of such units as a composite function $f_L : \mathbb{R}^d \to \mathbb{R}$ where d denotes the number of inputs.

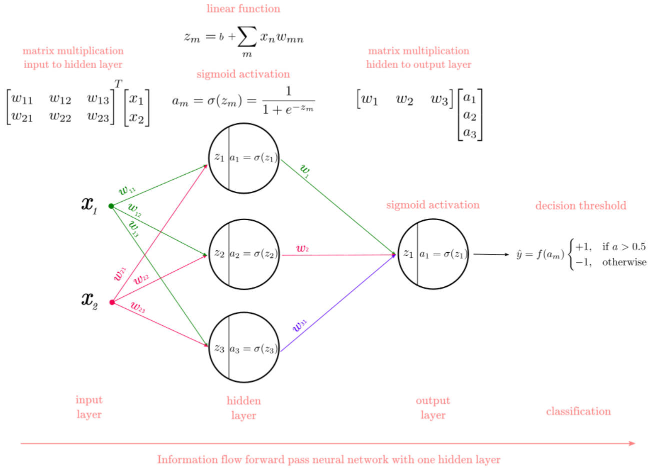
$$f_L(x) = \sigma\left(b_L + W_L \sigma\left(b_{L-1} + W_{L-1} \sigma\left(\cdots \sigma\left(b_1 + W_1 x\right) \cdots\right)\right)\right)$$

Theorem

$\mathcal{NN}^d_{\sigma,L}$ is dense in $\mathcal{C}(K,\mathbb{R})$ for $L \geq 1$.

Here $\mathcal{NN}^d_{\sigma,L} \subset \{F : \mathbb{R}^d \to \mathbb{R}\}$ is the class of neural networks with d inputs, L hidden layers and σ activation function. And $K \subset \mathbb{R}^d$ is compact. Note that there are some mild conditions on σ too but for our purposes they can be ignored.

Forward pass



Back to cats

- Neural networks seem promising
- How can we integrate them into our "learning" plan?

"Learning"

1 - Make guesses

2 - Device a way of measuring how accurate our guesses are

3 - Find out the changes we need to make to make better guesses

4 - Rinse and repeat until satisfied with the guesses we are making

How to make guesses?

- We can feed images to a neural network and propagate through the layers
- The output would be our guess
- We can choose the initial weights randomly as we are hoping to find a way of getting the optimal weights

1 - ~~Make guesses~~ $\rightarrow$ Forward propagation

2 - Device a way of measuring how accurate our guesses are

3 - Find out the changes we need to make to make better guesses

4 - Rinse and repeat until satisfied with the guesses we are making

- We need to know what the output was supposed to be
- We can manually label some pictures by hand and use that set of images as training data
- Define a metric to measure the distance between true and guessed labels
- Single image error $\rightarrow$ Loss function: $\mathcal{L}(y, \hat{y})$
- Average error over training data $\rightarrow$ Cost function:

$$\mathcal{J}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \hat{y}_i)$$

1 - ~~Make guesses~~ $\rightarrow$ Forward propagation

2 - ~~Accuracy of guesses~~ $\rightarrow$ Loss function

3 - Find out the changes we need to make to make better guesses

4 - Rinse and repeat until satisfied with the guesses we are making

- Better guesses in the sense that guesses with lower loss
- $\nabla \mathcal{J} = \left(\frac{\partial \mathcal{J}}{\partial w_{ij}} \right)$ gives the direction of steepest ascent
- Gradient descent:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial \mathcal{J}}{\partial w_{ij}}$$

  where $\eta \in \mathbb{R}^{+}$ is a scalar to control the convergence rate, called "learning rate".
- Update all weights using this rule

1 - ~~Make guesses~~ $\rightarrow$ Forward propagation

2 - ~~Accuracy of guesses~~ $\rightarrow$ Loss function

3 - ~~Make better guesses~~ $\rightarrow$ Backward propagation with GD

4 - Rinse and repeat until satisfied with the guesses we are making

1 - ~~Make guesses~~ $\rightarrow$ Forward propagation

2 - ~~Accuracy of guesses~~ $\rightarrow$ Loss function

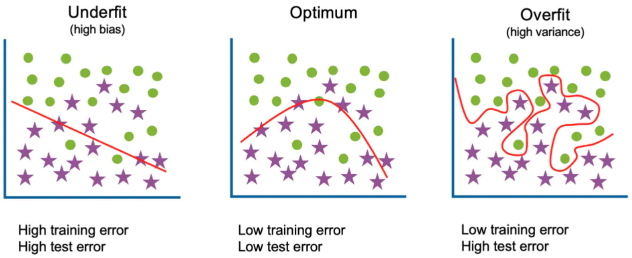
3 - ~~Make better guesses~~ $\rightarrow$ Backward propagation with GD

4 - ~~Repeat~~ $\rightarrow$ Iterate until the loss is minimized

Which minima is better?

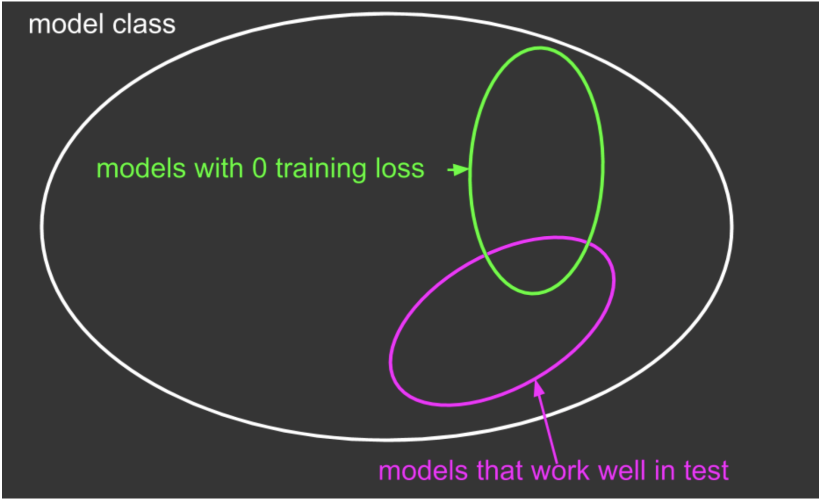
- Usually loss function has several minima
- Some are more preferable than others
- Ones that result in better generalization performance

- Some of the minima might be a result of focusing too much on the noise:



- We manually label more images to be used as test data
- Benchmark performance on training data against performance on test data

Model class


model class
models with 0 training loss
models that work well in test

1 - Forward propagation

2 - Loss function

3 - Backward propagation with GD

4 - Iterate until the loss is minimized

5 - Check performance on the test data

Future Work

- Meow recognition using RNNs
- Add a feature that lets users add pictures of cats as cat owners love showing pictures of their cats $\rightarrow$ How to integrate?

Thank you for listening!

Figure 6: Hellim

Figure 7: A campus cat