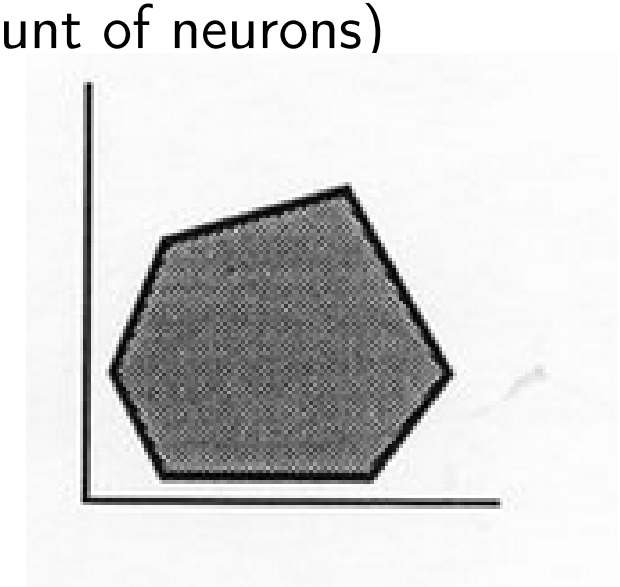
Deep Neural Networks

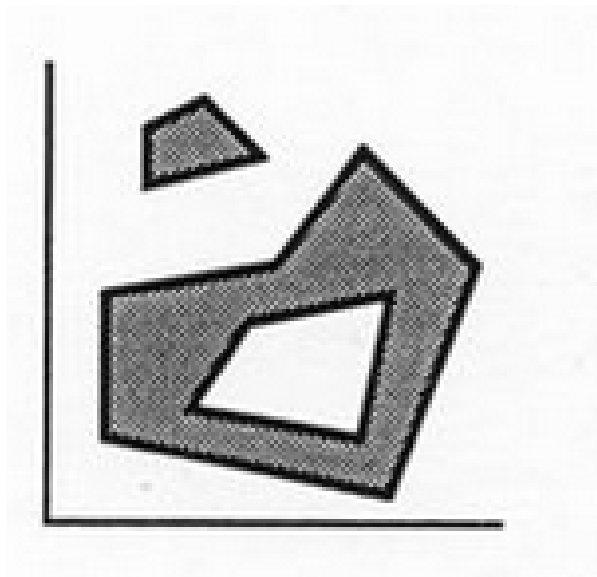
Tamás Grósz

Simple neural networks (ANN)

- Traditional NNs have one or two hidden layers
- There is a theorical proof that a ANN with two hidden layer can learn any function (assuming that we can use infinite amount of neurons)

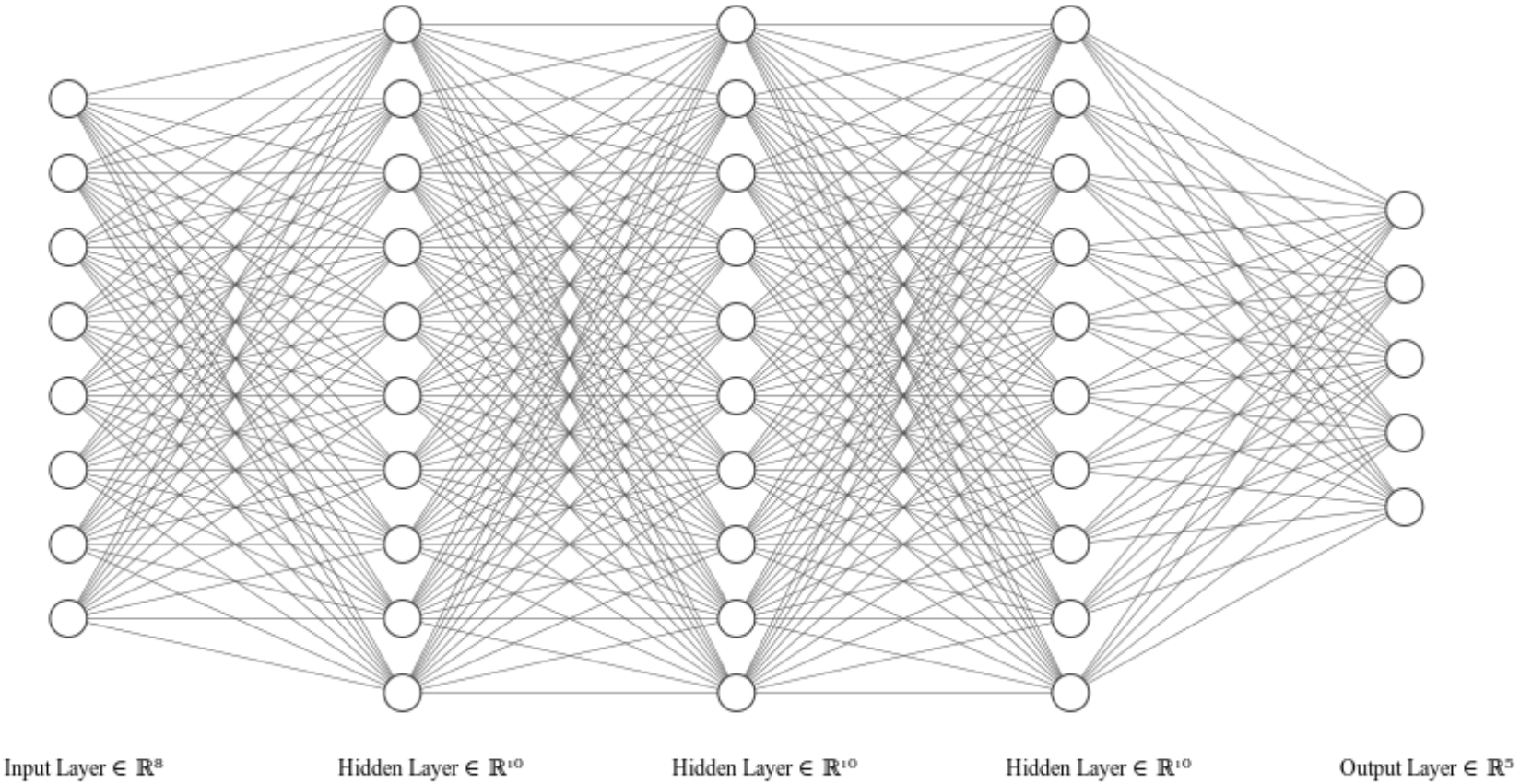


(a) One hidden layer can learn to recognize convex areas



(b) Two hidden layer can learn to recognize any non-covex, non-continous area

Deep Neural Networks



Why is a DNN better than a shallow ANN?

- More hidden layer means it can learn more complex functions
- It can learn hierarchical features
- If we have limited amount of neurons (computationsal power) it is advisable to use a deep structure
- DNNs have new training algorithms, which allow them to learn better models

Why is a DNN better than a shallow ANN?

- More hidden layer means it can learn more complex functions
- It can learn hierarchical features
- If we have limited amount of neurons (computationsal power) it is advisable to use a deep structure
- DNNs have new training algorithms, which allow them to learn better models

A large drawback is their computational requirements, this is why they were not used before.
Solution: use GPU for the computations (reminder: each layer requires a matrix multiplication, that can be done in parallel)

Training DNNs

Despite all their advantages, DNNs have many drawbacks too. During training, we propagate the error signal back through the layers, this process becomes problematic if we have many layers. We have to face two problem mainly:

- Vanishing Gradient problem
- Explaining Away effect

Vanishing Gradient

Reminder

The training algorithm propagates the gradient:
Gradient = (gradient of the error) * (gradient of the activation)

- The problem is with the gradient of the Sigmoid function $(o * (1 - o))$ as $o \leq 1$.

Vanishing Gradient

Reminder

The training algorithm propagates the gradient:
Gradient = (gradient of the error) * (gradient of the activation)

- The problem is with the gradient of the Sigmoid function $(o * (1 - o))$ as $o \leq 1$.
- As we propagate the error signal, it becomes smaller and smaller
- To update the weights we also multiply by the learning rate $(lr < 1)$

Vanishing Gradient

Reminder

The training algorithm propagates the gradient:
Gradient = (gradient of the error) * (gradient of the activation)

- The problem is with the gradient of the Sigmoid function $(o * (1 - o))$ as $o \leq 1$.
- As we propagate the error signal, it becomes smaller and smaller
- To update the weights we also multiply by the learning rate $(lr < 1)$
- In DNNs the gradient simply "vanishes", casing the deepest layers to stay randomly initialized.

Explaining Away

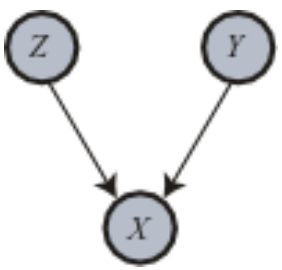
During training, the parameters associated with a single layer could become dependent if they are parents of common children in other layers.

Explaining Away

During training, the parameters associated with a single layer could become dependent if they are parents of common children in other layers.
A simple example:
A car's engine can fail (X). The reason might either be a dead battery Y or a blocked fuel pump Z.

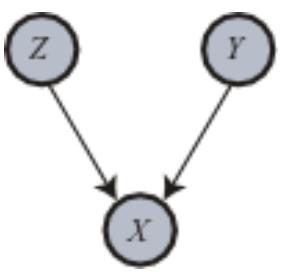
Explaining Away

During training, the parameters associated with a single layer could become dependent if they are parents of common children in other layers.
A simple example:
A car's engine can fail (X). The reason might either be a dead battery Y or a blocked fuel pump Z.



Explaining Away

During training, the parameters associated with a single layer could become dependent if they are parents of common children in other layers.
A simple example:
A car's engine can fail (X). The reason might either be a dead battery Y or a blocked fuel pump Z.



The problem: Given that you know the engine is broken, the event of a dead battery and a blocked fuel pump are suddenly not independent anymore!

Possible solutions

Pre-training methods

Use a pre-training step that initializes the parameters of each layer (layer-by-layer) so that the final training could find a better local minimum.

Possible solutions

Pre-training methods

Use a pre-training step that initializes the parameters of each layer (layer-by-layer) so that the final training could find a better local minimum.

Change the activation function

Use other activation functions, which do not cause the vanishing gradient effect.

Possible solutions

Pre-training methods

Use a pre-training step that initializes the parameters of each layer (layer-by-layer) so that the final training could find a better local minimum.

Change the activation function

Use other activation functions, which do not cause the vanishing gradient effect.

New structure

Change the structure of the network or the behaviour of the neurons

Pre-training algorithms

- The goal is to prevent vanishing gradient.
- The training is done in two steps:
  1. Pre-training: use some algorithm to learn the initial weights layer-by-layer
  2. Finetuning: train the DNN with traditional methods
- Deep Belief Network pre-training: unsupervised pre-training of hidden layers
- Discriminative pre-training: use the standard SGD and start with a simple ANN, then gradualy add more hidden layers

Changing the activation function

The main problem is with the Sigmoid function, let's change it!
The criteria for an activation function:

- It needs to be non-linear
- Must be continuous, so that we can calculate its gradient

Some options: https://en.wikipedia.org/wiki/Activation_function

Rectified Linear Units

One of the most famous ones, it is widely used.

$$ReLU(x) = max(0, x)$$

Properties

- Biological plausibility: inactive neurons (in the case of negative output)
- No vanishing gradient problems: the gradient is the step function
- Efficient computation: could be evaluated much faster than Sigmoid
- Sparse activation (usually half of the neurons are inactive), it can be exploited

New structures/neurons

Structures

- Use skip-layer connections
- Residual blocks

New structures/neurons

Structures

- Use skip-layer connections
- Residual blocks

Neurons

- Convolutional neurons: they see only a small portion of the input, but we convolve them over the entire input
- LSTM: a new type of recurrent neuron

Practice

Python tutorial: practice_05.ipynb