Machine Learning & its Application

Fully Connected Networks

03/10/2023
Done by: Issa Hammoud

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Introduction

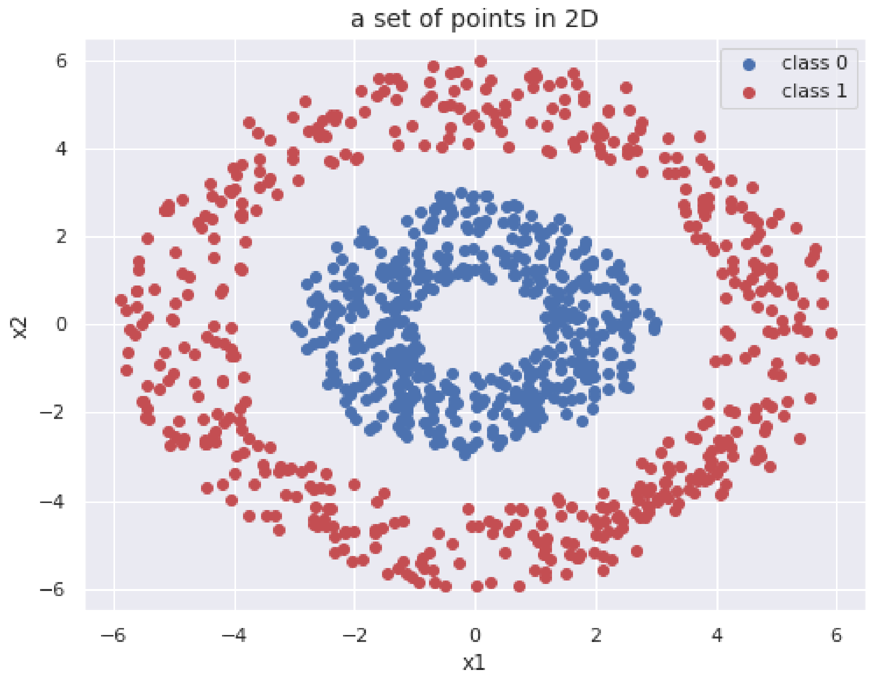
- We learned how to solve linear problems whether for regression or classification.
- However, in real life almost all problems are non-linear.
- We will see how we can develop solutions for such problems.

Outline

- Introduction
- The Problem of Non-Linearity
    - Introducing Non-Linearity
    - Activation functions
- Model Capacity
    - Underfitting and Overfitting
    - Regularization Techniques
- Parameters Initialization
- Practical Setup

The Problem of Non-Linearity

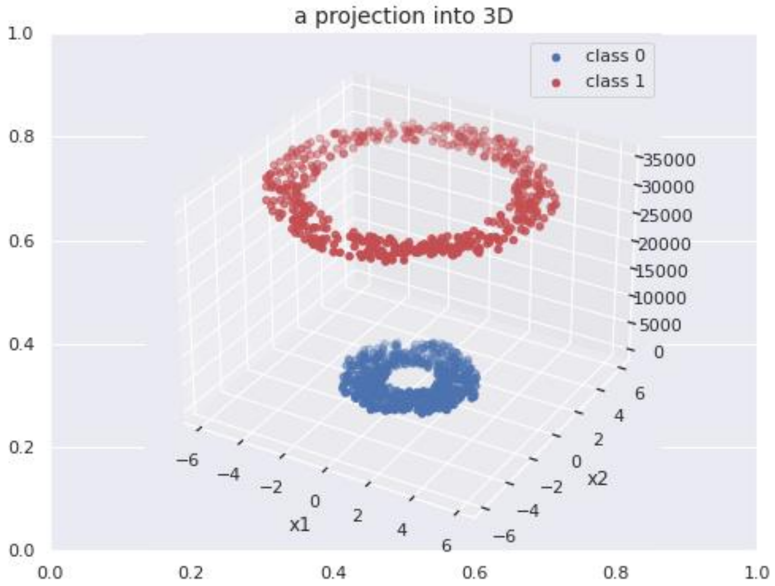
- Let's begin with a 2D binary classification example for simplicity.



- These data points cannot be separated with a line.

The Problem of Non-Linearity

- One way to solve the problem is as follows:
  1. Transform the data into a space where it is linearly separable.
  2. Apply linear model.
- We know that these points can be separated in a 3D space.



The Problem of Non-Linearity

- Classical algorithms, like SVM, uses the kernel trick to find this transformation.
- This means that we need to choose the kernel function ourselves.
- In deep learning, however, we learn this transformation.
- Let's design a 2-layer model, when at first we project into 3D and then classify:

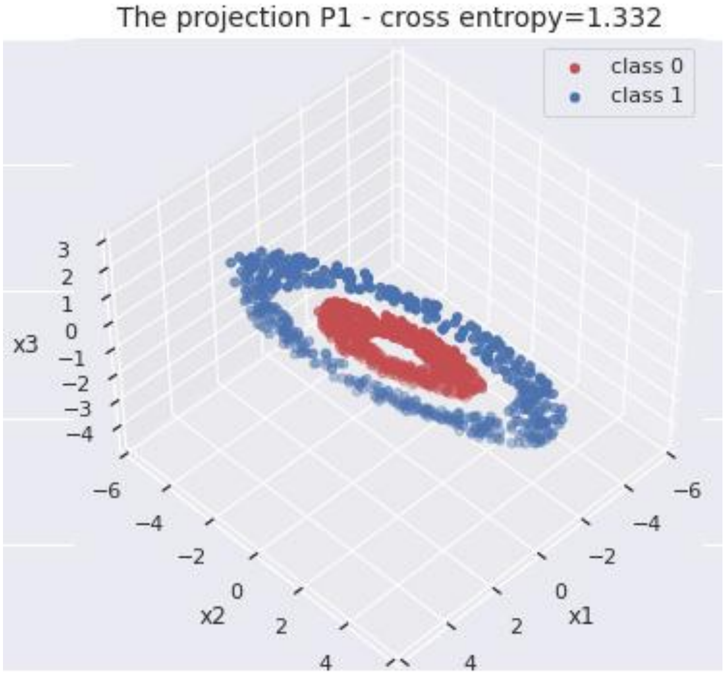
$$P_1 = A_1X \; ; \; X \in R^2 \; ; \; P_1 \in R^3 \; ; \; A_1 \in R^{(3,\,2)}$$
$$Z \;=\; A_2P_1 \; ; \; Z \in R \; ; \; A_2 \in R^{(1,\,3)}$$
$$Y' = \sigma(Z) \; ; \; Y' \in R$$

$$\text{binary cross entropy} : f(A_1,\, A_2) = \frac{1}{n}\sum_{i}^{n} -Y_i\log(Y_i') \;-\; (1-Y_i)\log(1-Y_i')$$

$$\frac{\partial f}{\partial A_1} = \frac{1}{n}\sum_{i}^{n}(X_iA_2)^T(Y_i' - Y_i) \; ; \; \frac{\partial f}{\partial A_2} = \frac{1}{n}\sum_{i}^{n}(A_1X_i)^T(Y_i' - Y_i)$$

The Problem of Non-Linearity

- First, we projected the data into 3D with a matrix A_1.
- Then, we applied logistic regression of top of it.
- We computed the derivatives so we can apply gradient descent.



The Problem of Non-Linearity

- We plotted the projection in 3D to see how the data is being separated.
- As we can see, the model is not able to separate them.
- This is normal, because a matrix multiplication can rotate the data not segregate it.
- In fact, this model is equivalent to any logistic regression algorithm.
- We can replace the multiplication of A_2A_1 with a single matrix A.
- We need to add a component so the model can separate the data.

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

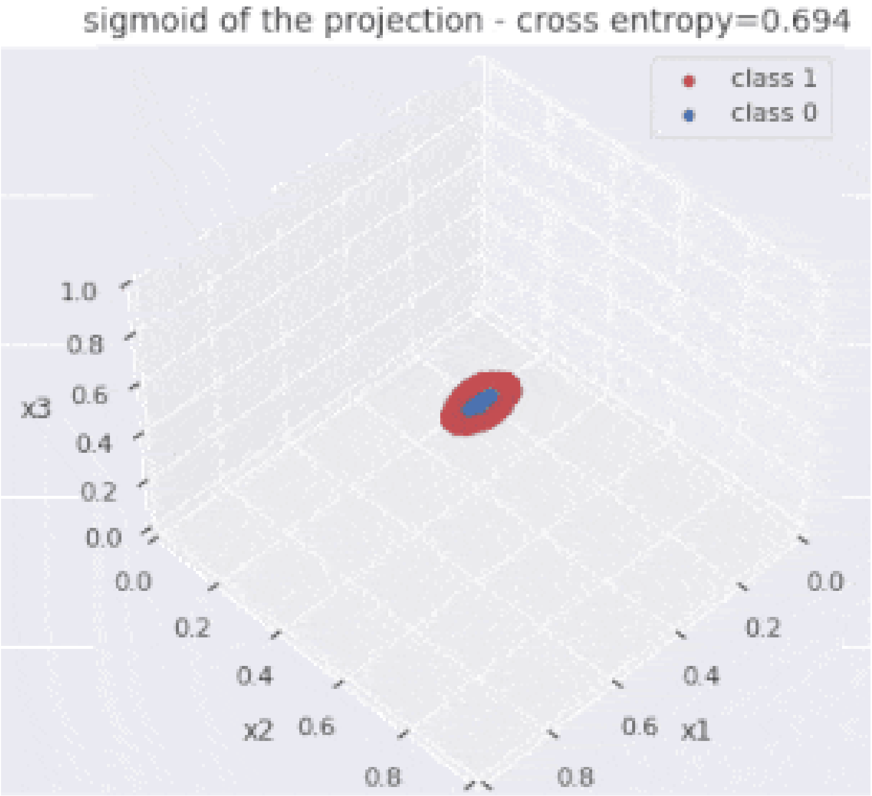
Introducing Non-Linearity

- We want the model to treat the data points separately, not in a proportional way.
- This can be done by introducing a non-linear function between the 2 layers.
- A non-linear function that we already met is sigmoid.
- Let's define the new model and see if the data will be separated.

$$\boxed{\begin{gathered} P_1 = A_1 X \,;\; X \in R^2 \,;\; P_1 \in R^3 \,;\; A_1 \in R^{(3,\,2)} \\ P_2 = \sigma(P_1)\,;\; P_2 \in R^3 \\ Z = A_2 P_2 \,;\; Z \in R \,;\; A_2 \in R^{(1,\,3)} \\ Y' = \sigma(Z)\,;\; Y' \in R \end{gathered}}$$

Introducing Non-Linearity

- As we can see, the model separated the data successfully!

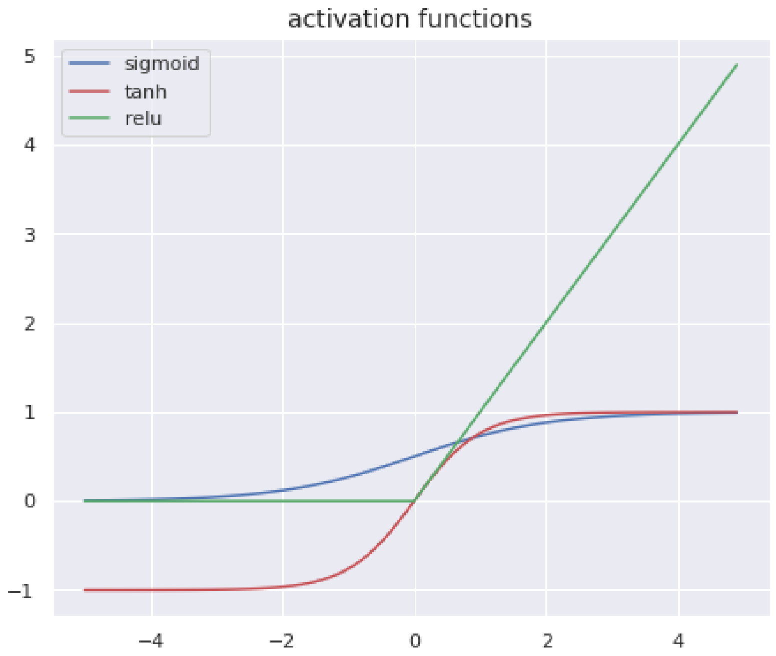


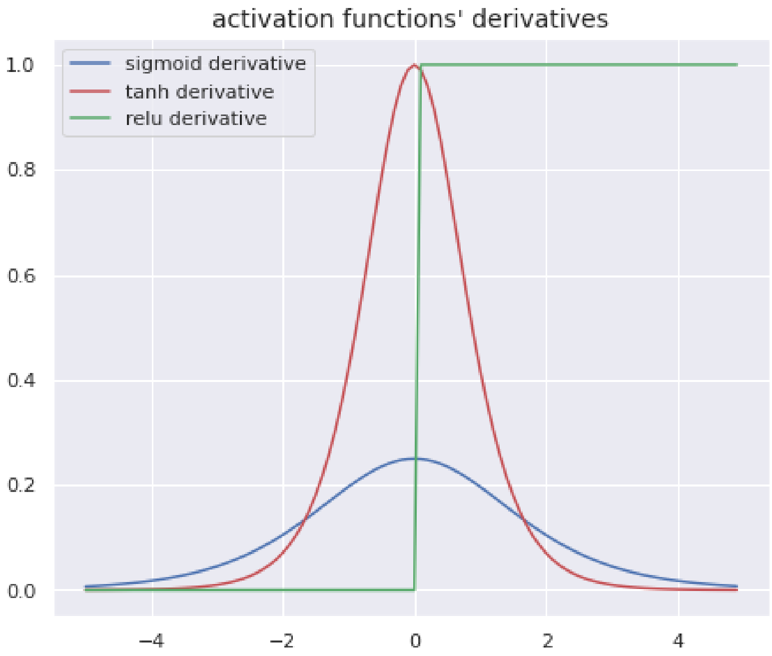
Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Activation Functions

- Non-linear functions inside a model are called activation functions.
- Historically, sigmoid and tanh were the first choices as activation functions.
- Nowadays, we use Rectified Linear Unit (ReLU).





Activation Functions

- Sigmoid and Tanh were replaced because they saturate quickly.
- It means that they can backpropagate gradient outside some ranges.
- This is problem is known as vanishing gradient.
- The required characteristics of activation functions are:
  - To be non-linear (obviously).
  - To be fast to compute.
  - Doesn't have an infinite number of non-differentiable points.
  - Doesn't saturate quickly.

Outline

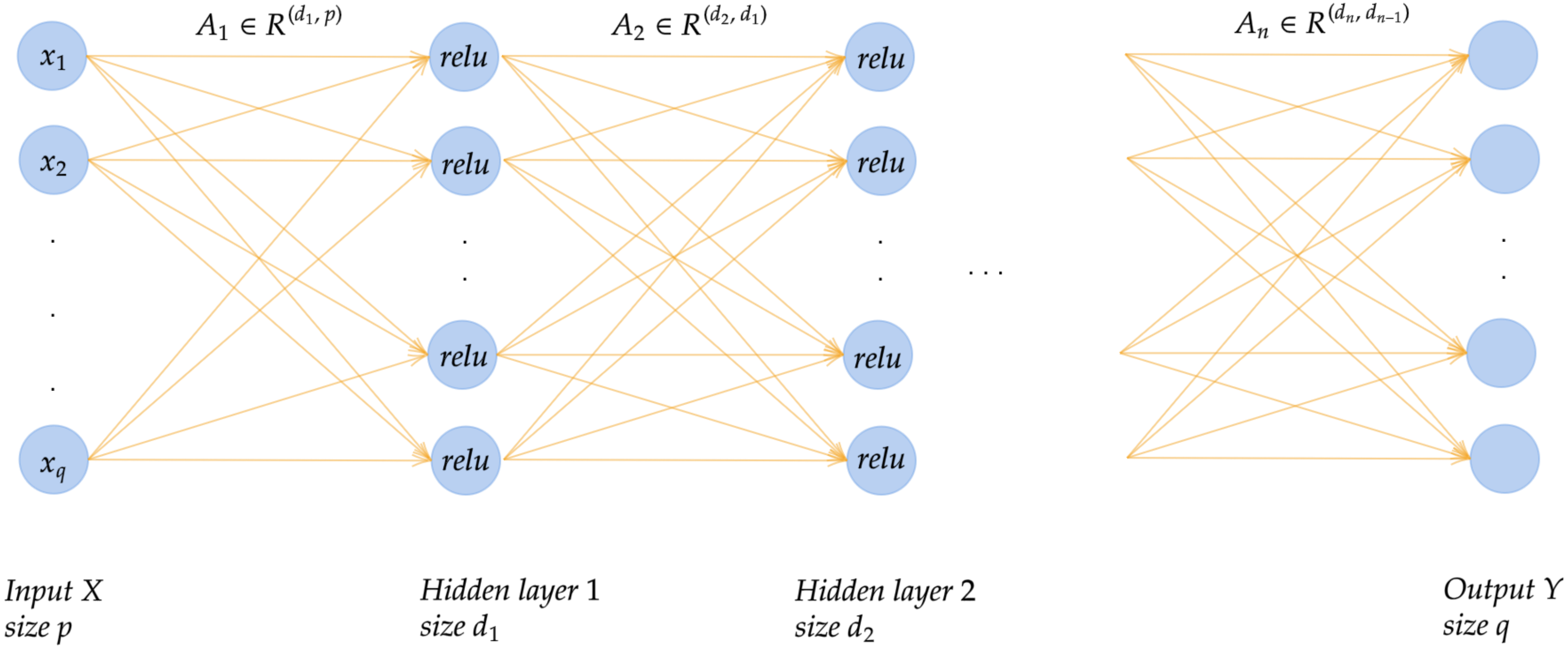
- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Model Capacity

- In the above example, we developed our first neural network.
- We call the intermediate projection a hidden layer, and each output value a neuron.
- However, it was easy to see that a projection in 3D will solve the problem.
- Can a non-linear projection solve any kind of problems?
- The **universal approximation theorem** guaranty that a neural network with a single hidden layer can approximate any functions, with enough neurons.
- By to find what is enough? In other words how to define model capacity?

Model Capacity

- Even though a single hidden layer is enough theoretical, we use many in practice.
- It is easier to optimize multiple layers with a fewer parameters than a single huge one.
- A typical neural network looks like this:



Model Capacity

- However, we didn't answer the question of how to choose how many neurons?
- In fact, we made it harder, we need to choose how many neurons and layers now.
- But, can we just a very big model to solve both easy and complex problems?
- The answer is No! This is due to a problem known as overfitting.

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Underfitting and Overfitting

- There are 2 problems that can happen when fitting a dataset:
  - Underfitting: when a model is not capable of learning the underlying relation.
  - Overfitting: when the model is too capacitive so it can find multiple relations that explain the data.
- We already saw an example of underfitting: using linear models on non-linear data.
- To solve underfitting we need to add more capacity to the model.
- Capacity is mainly composed on 2 ingredients:
  - The design choices: for instance, adding activation functions.
  - The number of parameters: adding more layers and more neurons per layer.

Underfitting and Overfitting

- Underfitting is intuitive, but why overfitting happens? Why more capacity is not good?
- To understand the problem let take the following example.
- Imagine you have a sequence of numbers that you want to continue: 1, 1, 2, 2.
- Most people will think that each number is repeated twice: 1, 1, 2, 2, 3, 3 etc..
- However, a math geek may see another valid hypothesis: Euler totient sequence.
- This sequence is as follows: 1, 1, 2, 2, 4, 2, 6, 4 etc...

Underfitting and Overfitting

- Both hypothesis explain well the observed data, but only 1 is correct.
- By why we should choose the first?
- There is a principle in science known as **Occam razor**, states that among competing hypotheses that explain known observations equally well, we should choose the "simplest" one.
- So how to overcome overfitting? As we can conclude from the example, we can:
    1. Acquire more data, so this will narrow the possible hypothesis.
    2. Reduce the model capacity.

Underfitting and Overfitting

- Acquiring more data is not always possible.
- We can use some techniques like data augmentation, but also this can be limited.
- The other choice returns us back to the question of determining the model capacity.
- There is a lot of combinations to try before finding the correct capacity.
- This is impractical. However, we can do something smarter:
    1. Use a model with higher estimated capacity than needed.
    2. Use regularization techniques to automatically reduce the model capacity.

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Regularization Techniques

- Regularization techniques set constraint to the model to limit its capacity.
- There are 2 main ways to regularize a model:
  1. Parameters norm penalty: add a constraint on the parameters' norm.
  2. Dropout: randomly drop a set of neuron during training.
- Let's begin with the first one: Parameters Norm Penalty.
- The idea is to limit the range of values the parameters can take.
- It is applied to the norm instead of each value to give more flexibility to the model.

Regularization Techniques

- We will apply a penalty term to the 2-layers models we defined earlier.
- Norm constraints are added to the loss function, as follows:

$$P_1 = A_1 X \; ; \; X \in R^2 \; ; \; P_1 \in R^3 \; ; \; A_1 \in R^{(3,2)}$$

$$P_2 = relu(P_1) \; ; \; P_2 \in R^3$$

$$Z = A_2 P_2 \; ; \; Z \in R \; ; \; A_2 \in R^{(1,3)}$$

$$Y' = \sigma(Z) \; ; \; Y' \in R$$

$$\text{binary cross entropy loss}: L(A_1, \; A_2) = \frac{1}{n}\sum_{i}^{n} - Y_i \log(Y_i') \; - \; (1 - Y_i)\log(1 - Y_i')$$

$$\text{penalty term}: \; \lambda_1 ||A_1||_p^p \; + \; \lambda_2 ||A_2||_p^p \; ; \; \lambda_1, \; \lambda_2 \in R \; ; \; \lambda_1 \; \geqslant 0, \; \lambda_2 \geqslant 0$$

$$\text{total loss}: \; L \; + \lambda_1 \left(||A_1||_p^p \; - b_1 \right) + \; \lambda_2 \left(||A_2||_p^p - b_2 \right) \; ; \; b_1, b_2 \; \in R$$

Regularization Techniques

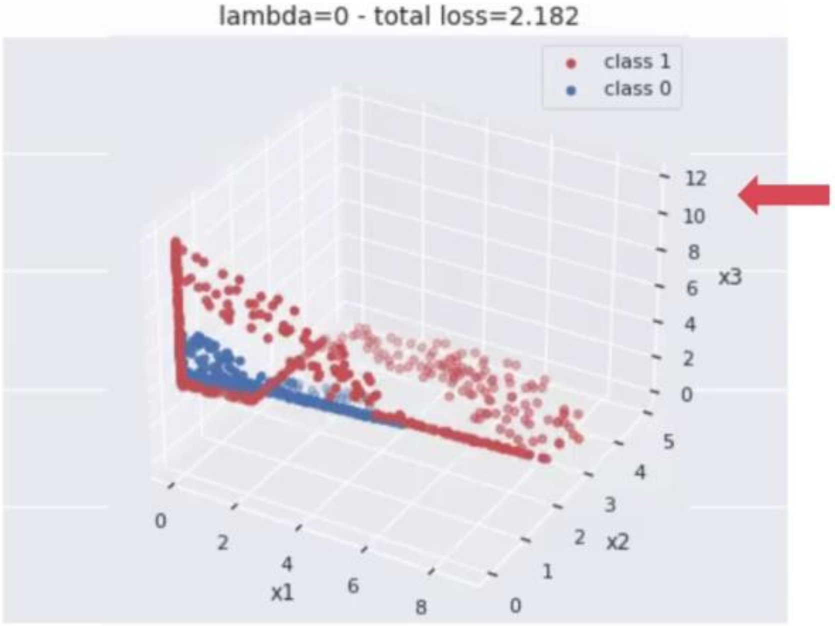
- We have some hyperparameters to choose:
  - We can apply it to some or all parameters.
  - We can set the value we want the norm to be near it (in general we push it toward zero).
  - We can choose which norm to use (e.g. L_1, L_2 etc..)
  - We can control the strength of the constraint with a weight parameter λ.
- In this example, we will apply it to A_1, push it toward 0 using L_2 norm with a weight λ_1.

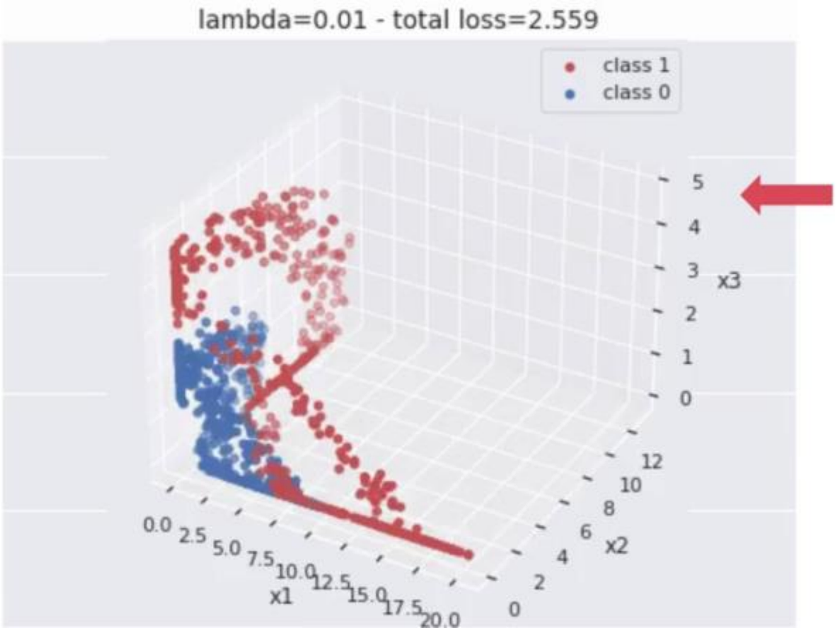
$$\text{total loss } L(A_1,\ A_2) = \ \frac{1}{n}\sum_{i}^{n} -Y_i \log(Y_i') \ - \ (1 - Y_i)\log(1 - Y_i') \ + \lambda_1 ||A_1||_2^2$$

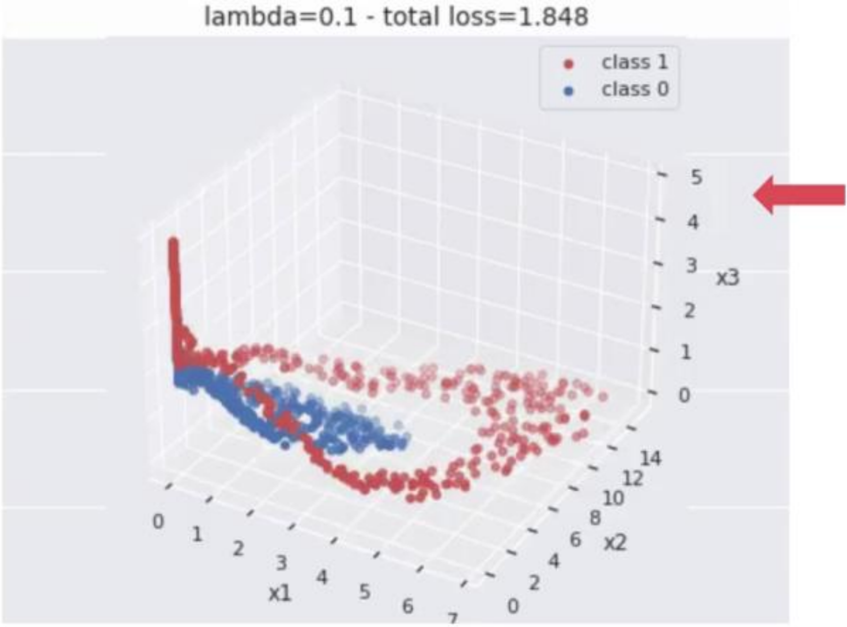
$$\frac{\partial L}{\partial A_1} = \frac{1}{n}\sum_{i}^{n}(X_i A_2)^T (Y_i' - Y_i)(P_2 > 0) + \boxed{2\lambda_1 A_1};\ \frac{\partial L}{\partial A_2} = \frac{1}{n}\sum_{i}^{n} P_2^T (Y_i' - Y_i)$$

Regularization Techniques

- This regularization (with L_2 norm toward 0) is known as weight decay.
- When used with a linear regression model, it is known as ridge regression.





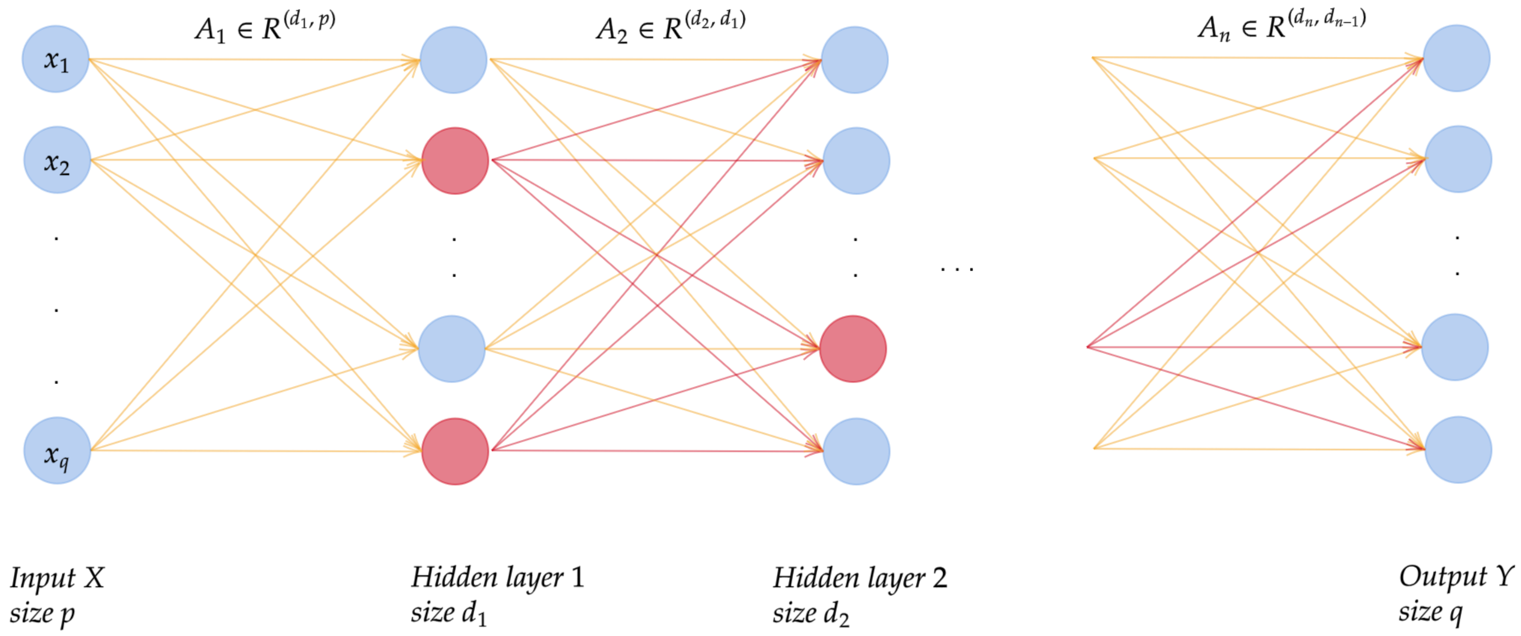


Regularization Techniques

- Norm constraints are powerful, but we still need to choose:
  - If we put constraint on some or all layers? Too much constraints may result in underfitting.
  - The weight parameter λ for each layer.
- In practice, it is preferable to use dropout instead.
- To understand dropout, we need to talk about **ensemble methods**.
- It is technique used in classical machine learning to reduce generalization error.
- The idea is to train multiple models and let them vote on a given prediction.

Regularization Techniques

- In deep learning, training a single model is costly.
- Dropout approximates ensemble learning in a computationally cheap way.
- By randomly dropping neurons, we are training an exponential number of submodels.



Regularization Techniques

- But how inference will be done? How to make the model vote?

Suppose we are training a linear regression model : $Y = AX \, ; \, X \in R^n$

By applying dropout, we train at each iteration i a model : $Y_i = A(X \odot d_i)$
where $d_i \in \{0,1\}^n \sim Bernoulli\ B(p) \, ; \, p$: dropout probability

During inference, we want to use the average model to predict : $Y_{average} = \frac{1}{2^n}\sum_i^{2^n} Y_i$

$$Y_{average} = \frac{1}{2^n}\sum_i^{2^n} Y_i = \frac{1}{2^n}\sum_i^{2^n} A(X \odot d_i) = \frac{1}{2^n}A\sum_i^{2^n}(X \odot d_i) = \frac{1}{2^n}A\left(X \odot \sum_i^{2^n} d_i\right)$$

$$Y_{average} = \frac{2^n p}{2^n}AX = pAX = pY$$

$$\sum_i^{2^n} d_i = \begin{pmatrix} \sum_i^{2^n} d_{1i} \\ \sum_i^{2^n} d_{2i} \\ \vdots \\ \sum_i^{2^n} d_{ni} \end{pmatrix} = \begin{pmatrix} 2^n p \\ 2^n p \\ \vdots \\ 2^n p \end{pmatrix} = 2^n p \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Regularization Techniques

- This rule is known as weight scaling inference rule.
- The proof only apply to linear models, but is used in all models in practice.
- It can be applied in one of 2 ways:
  - Multiply the output activation by p (the drop probability) during inference.
  - Divide the output activation by p during training.
- We can see dropout from another perspective as a regularization mechanism.
- By randomly dropping neurons, it forces them to learn useful features independently.

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

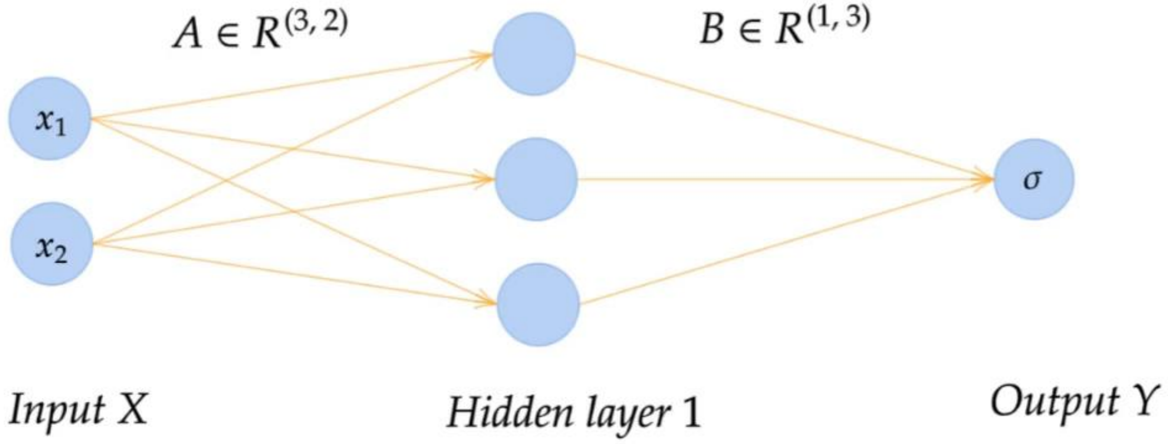
Parameters Initialization

- We did talk about almost everything except parameters initialization.
- When initialization the parameters, we need to answer 2 main questions:
  1. Can we initialize the parameters (the matrices A) with fixed values, or should be random?
  2. How to choose the scale of the initialization? Of which order of magnitude should it be?
- In general, we set biases to zero, but initialize the weights randomly. But why?
- The answer is to break the symmetry. Let's look what that means.

Parameters Initialization

- We will take a simple 2-layers model without activation for simplicity.
- We will initialize some weights by the same values and how they will be updated.
- Parameters initialized by the same value work as mere copies!

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$



$$B = \begin{pmatrix} b_1 & b_2 & b_3 \end{pmatrix}$$

Parameters Initialization

- So we need to initialize the weights randomly.
- In general, we use Uniform or Gaussian distributions centered at 0.
- But to choose weights variance?
- ***He*** proposed an initialization for networks with ReLU activations.
- It forces the activation variance to stay the same.

Outline

- Introduction
- The Problem of Non-Linearity
  - Introducing Non-Linearity
  - Activation functions
- Model Capacity
  - Underfitting and Overfitting
  - Regularization Techniques
- Parameters Initialization
- Practical Setup

Practical Setup – Data Preparation

- Check the amount of data you have because it will influence the solution choice.
- Prepare a test set that reflects the diversity of real world.
- Use 10-20% of your training data for validation.
- The purpose of validation set is to monitor the training and generalization errors:
  - If validation errors are still decreasing, we should iterate more because we may improve.
  - When the validation error reaches a steady phase and begins increasing, we should stop iterating.
- The training error will continue to decrease except if there is an underfitting.

Practical Setup – Model Design

- The design decisions depend on the problem, but we can have generic guidelines:
    - Choose the model capacity based on the task complexity and the size of your training set.
    - Use dropout to automatically reduce the model capacity.
    - Use ReLU activation function, specifically with deep models.
    - It is better to have many hidden layers with fewer hidden units, than fewer layers with more units.
    - Think of normalizing your input to set it to the same scale and stabilize your training.
- Begin with a baseline model then iterate for a more complex solutions.
- In the coming lesson we will talk more about design choices.

Practical Setup – Debugging

- Once you train your first model, it will probably not work.
- The ***FIRST*** thing to do is to check your learning curves.
- Learning curves are the curves of training and validation errors w.r.t epochs.
- Ask yourself the following questions:
  - Did the training loss decrease? If not, we are underfitting, and we need more capacity.
  - Does the validation loss stop decreasing? If not, we need to iterate more because we may improve.
  - Are the curves decreasing very slowly? If yes, we may need to increase the learning rate.
  - Are the curves oscillating strongly? If yes, we may need to decrease the learning rate, use a schedule if not yet used, or increase the batch size.

Exercise

- We will implement our first neural network with TensorFlow.
- The goal is to implement a gender classification solution.
- We will work on Kaggle as the data is there, and to train on GPU.
- Download the notebook from my github and upload it to your Kaggle account.
- Add the following data to your notebook:
  - gender-classification-dataset (by Ashutosh Chauhan).
  - Images to test on.