Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

Machine Learning on Microcontrollers

227-0155-00L

Exercise 2

Feature Extraction & Regularization in Machine Learning: avoiding overfitting, improving generalization.

Michele Magno, PhD
Viviane Potocnik
Marco Giordano
Pietro Bonazzi

Monday 13th March, 2023

1 Introduction

In this short exercise, we will see on the one hand how polynomial regression fails on noisy data while neural networks are much more robust. On the other hand, we will discuss regularization techniques that speed up learning and avoid overfitting in neural networks.

2 Fitting Noisy Polynomial Data

We have a noisy measurement of a polynomial

$$f(x) = 10 + \frac{5}{10}x - \frac{4}{10^2}x^2 + \frac{2}{10^3}x^3 + \frac{3}{10^4}x^4 - \frac{1}{10^5}x^5 + N \tag{1}$$

with noise N.

2.1 Polynomial Regression

Polynomial regression finds $y(x)$ that minimizes the Mean Squared Error (MSE) based on n samples of $f(x)$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y(x_i) - f(x_i))^2 \tag{2}$$

where $y(x)$ is a polynomial with degree d:

$$y(x) = \sum_{i=0}^{d} c_i x^i \tag{3}$$

In this exercise, we fit $y(x)$ with various degrees d and wish to obtain coefficients c_i that are identical to the ones of equation 1.

Student Task: What minimal degree d must $y(x)$ have such that an MSEa of 0 can be achieved? Answer: Having a polynomial of degree d means that we have $d+1$ parameters c_i that we can set (see equation 3). This means that with $d = n - 1$, we can fit the curve perfectly through n points.

Note: Having a polynomial degree greater than the one found above quickly leads to capturing the noise of the data. This phenomenon is called *overfitting* and can also happen to neural networks if a model contains too many trainable parameters.

We now turn to the `regression` code provided with this exercise. In that code, we randomly generate 100 points of data according to equation 1 above. We choose $N \in [-2, 2]$ to be uniformly distributed and fit polynomials of various degrees on the $x \in [10, 20]$ interval. Then, we see how well the obtained results generalize by extending the interval to $[8, 22]$.

Student Task: Run the `regression` code. Which polynomial degree generalizes best? Why? Answer: $n = 4$, even though the data is generated with 5. This is due to the susceptibility to noise. Choosing $n = 4$ *regularizes* the regression.

2.2 Deep Neural Networks

A deep neural network consists of daisy-chaining dense network layers. An individual layer of a such a dense network with n input units and a single output is characterized by

$$y(x) = \phi\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{4}$$

where b is a learnable bias, w_i are trainable parameters and $\phi()$ is a non-linear function. In this exercise, we consider both $\phi(x) = x^2$ and the so-called *exponential linear unit (ELU)*

$$\phi(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{else} \end{cases} \tag{5}$$

with a hyperparameter α that is set to 1 per default.

Student Task: Extend equation 4 to have k outputs.

Answer:

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(k)} \end{bmatrix} = \phi\left(\begin{bmatrix} b^{(1)} \\ b^{(2)} \\ \vdots \\ b^{(k)} \end{bmatrix} + \begin{bmatrix} w_1^{(1)} & w_2^{(1)} & \dots & w_n^{(1)} \\ w_1^{(2)} & w_2^{(2)} & \dots & w_n^{(2)} \\ \vdots & \vdots & & \vdots \\ w_1^{(k)} & w_2^{(k)} & \dots & w_n^{(k)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) \tag{6}$$

$$\vec{y} = \phi\left(\vec{b} + W \cdot \vec{x}\right) \tag{7}$$

Note: In this exercise, we have a deep neural network with one single input and one single output. Layers in between (the so-called *hidden layers*) are less trivial and resemble the form obtained in the student task just above.

We now turn to the `regularization` code of this exercise. In the same setting as in the `regression` code, we generate 100 data points on $x \in [10, 20]$ from which we use 80 as training and 20 as validation set. The data on $[8, 22]$ is then used as a test set.

Student Task: Run the `regularization` code. Is the neural network overfitting? How can one tell? Answer: Not overfitting as the validation loss is also minimized.

Additionally, there are three different configurations in the `regularization` that we can set: `dropout`, `batchnorm` and `normalize`.

Student Task:

1. What do these configurations do?
2. Play around with them and see how they affect the predictions. What do you observe?

Answer: Dropout deactivates units randomly. Batchnorm normalizes the data in each layer. Normalize normalizes the input data. You should avoid combining data normalization (preprocessing the inputs to have zero mean and unit variance) with batch normalization layers within your neural network because both techniques serve similar purposes at different stages, leading to redundancy and potential interference. Data normalization adjusts the input data before it enters the network, while batch normalization normalizes the outputs of intermediate layers during training. Using both can disrupt the internal normalization process of batch normalization, potentially causing unstable training, slower convergence, or degraded model performance due to over-normalization. Therefore, it's advisable to choose one method of normalization to ensure effective and efficient training.

3 Regularization Techniques for Neural Networks

To prevent overfitting, several techniques can be used on neural networks. In the following, we discuss several of them:

Batch size By backpropagating the average of several forward propagations, the training can be parallelized and thus accelerated. Furthermore, taking the average smoothes out noise, therefore regularizing the network.

Early stopping Stop training when the validation loss is as small as possible. Return the network before it can overfit.

L1/L2 regularization We extend the loss function with either the L1 term

$$\text{MSE} + \lambda \sum_{w \in W} |w| \tag{8}$$

or with the L2 term

$$\text{MSE} + \lambda \sum_{w \in W} w^2 \tag{9}$$

The non-negative λ can be seen as a "tuning knob". This means that the learnable weights w partly account for the loss and will therefore be kept lower than without L1 or L2 regularization.

Student Task:

1. Draw the gradients of $|x|$ and x^2.
2. Explain why in opposition to L2, L1 leads to sparse weights in the network.

Answer

1. The gradients look as follows:

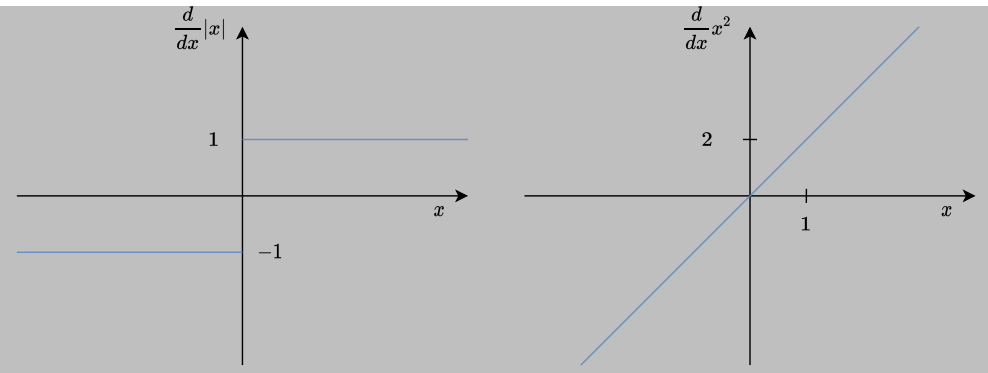

Figure 1: Gradients of $|x|$ and x^2

2. As can be seen from the gradients, L1 does not push the weights towards zero in accordance to their magnitude. L2 does, and thus has the tendency to minimize weights with bigger magnitudes.

Dropout Units are randomly disabled. This also speeds up learning and prevents units from co-adapting.

Student Task: Run the `regularization` code with dropout enabled by passing `dropout` as a command line argument. What do you expect to observe? Do the results match your expectations? Answer: No big difference as we are not overfitting. Training loss should be noisier as we randomly deactivate units. Confirmed in the plot.

Batch normalization Recenter and rescale the data using the following empirical mean and variance of the batch B:

$$\mu_B = \frac{1}{|B|} \sum_{x \in B} x \tag{10}$$

$$\sigma_B^2 = \frac{1}{|B|} \sum_{x \in B} (x - \mu_B)^2 \tag{11}$$

Then, the data is transformed with

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \ \forall x \in B \tag{12}$$

with non-negative ε.

Student Task: Why is there an ε in equation 12? Answer: For numerical stability. With decreasing noise, the variance goes to zero.

The `regularization` code has several other mechanisms implemented. Try them out if you like!

4 Feature Extraction

Now that you are more familiar with the technical tools, let's talk about an important concept you will find extremely useful: feature extraction.

When you want to analyze a particular object, like an image or an audio track, you firstly need a digital representation of it. In the case of an image, you will have a set of numbers describing the color of each pixel, in the case of an audio track, you will have a set of amplitudes and a sampling rate.

Sometimes, the information collected to describe the object can be redundant, or there are better ways to describe the same object reducing the required amount of data. The process of reducing the dimensionality of this information into more manageable descriptors (features) is called feature extraction. This process allows to compress the initial sets of data, easing the subsequent processing that would be otherwise impractical and too resources/time consuming.

Student Task 1 (Feature Extraction): Important features you can extract from an audio file are the Mel-frequency cepstral coefficients (MFCCs), coefficients that describe a Mel-frequency cepstrum, a representation of the short-term power spectrum of the audio.

You will now extract MFCCs from an audio sample with the aid of a Jupyter Notebook. Read the *Part 2: Audio Feature Extraction* of the Notebook.

To correctly run the code inside the notebook, you will need to download additional packages, like *librosa*. To do so, add "conda-forge" to Anaconda's channels list and update the indexes

before installing the packages. If you followed the instructions in the first lab (Anaconda Setup) and opened the notebook in the `mfcc_test` environment, everything should work.

Follow the instructions and run the code in the Notebook. Answer the following questions.

- Imagine to store all the samples in a micro-controller. How much memory do you need? Assume that the size of each sample is $2\,\mathrm{B}$ $(16\,\mathrm{bit})$.

  $Required\ memory$ = 132300 $\times$ 2 B = 264.6 kB

- How many features do you have before extracting the MFCCs? **(We count also the information about the Sampling Rate)**

  $Features_0$ = $Samples\ +\ 1$ = 132300 + 1 = 132301

- How many features do you have, after? **(We count also the information about the number of frames)**

  $Features_1$ = $MFCCs\ +\ 1$ = 5180 + 1 = 5181

- Suppose that a sample and a MFCC have the same size in memory. Calculate the achieved compression ratio. **(With this definition of CR, the lower the CR, the better.)**

  CR = $\frac{Features_1}{Features_0}$ = 3.92%

$\mathcal{E}$ **Congratulations! You have reached the end of the exercise. If you are unsure of your results, discuss with an assistant.** $\mathcal{E}$