Semestral project

Adam Vrba

July 2025

1 Introduction

In this semestral project, we compare the gradient weight search algorithm against the evolutionary approach for finding weights of neural networks. In chapter 2 we describe the product datasets on which we will train and test neural networks. In the chapter 3 we will briefly describe which algorithms we will use in the experiments. In chapter 4 we describe preprocessing methods and used hyperparameters for the gradient and evolutionary approach. Lastly, for each dataset and preprocessing method, we compare the best networks found by the evolutionary and traditional approaches.

2 ProMap datasets

Product matching is a process of comparing a pair of products, each taken from different e–shops, and determining if they are the same (matching) or if they are different (non-matching). For this problem, we will use product mapping datasets.

These datasets contain data of two pairs of products such as the name of the product, short and long descriptions, specifications, product price, and other features. Datasets also contain information which tells us if this pair of products is matching or not. All these features have been preprocessed by Kateřina Macková and Martin Pilát to numerical features. Their ProMap datasets are available on their Github. For this semestral project, we chose to use 4 datasets: Amazon–Google, Amazon–Walmart, ProMapCz, and ProMapEn. All datasets contain around 2000 samples of data. The datasets were divided into an 80% training set and a 20% test set.

3 Weight search methods

As we mentioned in the introduction, we use gradient and evolutionary algorithms to find the weights of neural networks. In this section, we will in detail explain both approaches.

3.1 Gradient approach

For the gradient approach, we employ the Adam optimizer. To ensure a fair comparison with the evolutionary approach, which contains a population of neural networks, we decide to use grid search to identify the optimal hyperparameters for the Adam optimizer. The specific hyperparameters will be described in detail in section 4.

3.2 Evolutionary approach

For the evolutionary approach, we employ CMA–ES (Covariance Matrix Adaptation Evolution Strategy). This strategy performs well in continuous problem optimization, which is exactly our weight search problem.

4 Experiment results

For each ProMap dataset we use 3 different preprocessing methods: PCA, LDA, and RAW (no preprocessing was used). After preprocessing the data we run Gradient and Evolutionary weight search approaches on two layer sizes: small (consisting of 2 hidden layers of size 16 and 8) and large (consisting of 3 hidden layers of size 8, 4 and 2). For each combination of preprocessing method and weight search approach, we will identify the best architecture and compare their F1–score.

4.1 Data preprocessing

In this subsection, we will describe hyperparameters for PCA and LDA preprocessing.

4.1.1 PCA

For PCA preprocessing, we have selected the number of components such that the cumulative explained variance of data is greater than or equal to 95%. Other PCA parameters were kept to default values.

4.1.2 LDA

For LDA preprocessing, we have selected the number of components to be 1. Other LDA parameters were kept to default values.

4.2 Hyperparameters settings

In this subsection, we will discuss hyperparameter settings for the evolutionary and gradient approaches.

4.2.1 Evolutionary approach

For the evolutionary approach, we chose to use the deap library because it provides flexibility and ease of use. As mentioned, we used the CMA-ES algorithm to optimize weights in neural networks. The algorithm was run for 50 generations. We initialized centroid as a vector where each element was 0.1 and the length was equal to the total number of network parameters. The initial standard deviation of the distribution (parameter σ) was set to 0.5. The number of children to produce at each generation (parameter λ) was set to the total number of the network parameters. And lastly, the number of parents to keep from the lambda children (μ) was set to a fourth of λ. The fitness of each individual was evaluated using the F1-score. Other hyperparameters were kept at their default values.

4.2.2 Gradient approach

To ensure a fair comparison and identify optimal hyperparameters, we employed a grid search with 5–fold cross–validation for 50 max iterations to determine the best hyperparameters for the gradient approach. This grid explored various activation functions such as relu, tanh, and logistic, parameter α set to 0.0001 and 0.001, and hidden layer sizes set to small (16, 8) and large (8, 4, 2). After identifying the optimal hyperparameters, we retrained the network on the best hyperparameters for 50 iterations.

4.3 Results

In table 1 we compared the best models found by two approaches and on all datasets including the preprocessed dataset via PCA and LDA.

Models trained and validated on the Amazon–Google dataset achieved 97% to 98% F1–score. We observe no significant difference between the evolutionary and gradient approaches, nor among the different preprocessing techniques.

On ProMapCz, the results were different. LDA models with evolutionary and gradient approach achieved 73% F1–score, whereas models trained on raw dataset achieved 70% to 71% F1 score. The worst result was when we used PCA preprocessing technique. Gradient approach achieved 0% F1–score and evolutionary approach achieved only 33% F1–score. This poor performance on both approaches is attributed to the dimensionality reduction performed by PCA, which probably removed too much critical information, making the dataset extremely challenging.

Models trained and validated on every preprocessed ProMapEn dataset achieved 59% to 60% F1 score. Only the evolutionary approach on the PCA preprocessed dataset achieved 56% accuracy.

Lastly, on the Amazon–Walmart dataset, all methods on all types of preprocessing achieved 90% to 91% F1–score.

Table 1: Comparison of model performance on training datasets.

Preprocess	Train	Method	F1	Pre	Rec	Acc	Size
lda	google	backpropagation	0.986513	0.996109	0.977099	0.989181	(8, 4, 2)
lda	google	evolutionary	0.984733	0.984733	0.984733	0.987635	(8, 4, 2)
pca	google	evolutionary	0.984615	0.992248	0.977099	0.987635	(8, 4, 2)
pca	google	backpropagation	0.978723	0.992157	0.965649	0.982998	(16, 8)
raw	google	evolutionary	0.986564	0.992278	0.980916	0.989181	(8, 4, 2)
raw	google	backpropagation	0.986460	1.000000	0.973282	0.989181	(8, 4, 2)
lda	promapcz	backpropagation	0.734694	0.734694	0.734694	0.826087	(8, 4, 2)
lda	promapcz	evolutionary	0.734694	0.734694	0.734694	0.826087	(16, 8)
pca	promapcz	evolutionary	0.339869	0.472727	0.265306	0.662207	(16, 8)
pca	promapcz	backpropagation	0.000000	0.000000	0.000000	0.672241	(16, 8)
raw	promapcz	evolutionary	0.719577	0.747253	0.693878	0.822742	(8, 4, 2)
raw	promapcz	backpropagation	0.702703	0.747126	0.663265	0.816054	(16, 8)
lda	promapen	backpropagation	0.598802	0.757576	0.495050	0.784566	(16, 8)
lda	promapen	evolutionary	0.602273	0.706667	0.524752	0.774920	(16, 8)
pca	promapen	backpropagation	0.604938	0.803279	0.485149	0.794212	(16, 8)
pca	promapen	evolutionary	0.568182	0.666667	0.495050	0.755627	(8, 4, 2)
raw	promapen	backpropagation	0.602410	0.769231	0.495050	0.787781	(16, 8)
raw	promapen	evolutionary	0.595506	0.688312	0.524752	0.768489	(8, 4, 2)
lda	walmart	evolutionary	0.919149	0.892562	0.947368	0.939587	(16, 8)
lda	walmart	backpropagation	0.918103	0.902542	0.934211	0.939587	(16, 8)
pca	walmart	backpropagation	0.904348	0.896552	0.912281	0.930048	(8, 4, 2)
pca	walmart	evolutionary	0.902386	0.892704	0.912281	0.928458	(8, 4, 2)
raw	walmart	backpropagation	0.907127	0.893617	0.921053	0.931638	(8, 4, 2)
raw	walmart	evolutionary	0.902386	0.892704	0.912281	0.928458	(8, 4, 2)

5 Summary

Results showed that evolutionary algorithms can achieve models which have similar performance as models trained with gradient approaches. Data preprocessing did not improve our model by a significant margin, only the LDA method on ProMapCz datasets achieved around 3% better performance than models trained on the unprocessed dataset.

A significant limitation of evolutionary weight search appeared with larger and deeper neural networks (for example networks with layer size (1000) or even with size (100, 50)), because they often failed to complete even their initial generation within a reasonable time. This leads to the conclusion that while evolutionary weight search can yield good results on small networks, its computational speed is significantly slower compared to gradient–based weight search methods.