Comparative Analysis of Linear Models, Feedforward Neural Networks, and Autoencoder-Enhanced Networks Across Diverse Machine Learning Tasks

Srikanta Mehta SMEHTA44@JH.EDU
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21205, USA

Abstract

This study explores the performance of linear models, feedforward neural networks (FFNs), and autoencoder-enhanced networks across a variety of machine learning tasks, including both classification and regression. Utilizing datasets from the UCI Machine Learning Repository, we examine how these models perform under different conditions, focusing on Mean Squared Error (MSE) for regression and cross-entropy loss for classification. Our experimental results provide insights into how the complexity of model architectures influences performance. Specifically, we observe that while more complex models like FFNs and autoencoder-enhanced networks often yield improvements in handling non-linear and high-dimensional data, their effectiveness is not uniformly superior across all datasets. The performance of each model type is heavily dependent on specific dataset characteristics, such as attribute types, the complexity of relationships among variables, and the nature of the task. These findings support the hypothesis that there is no statistically significant difference in performance across all datasets when comparing different models; instead, model selection must be tailored to the unique characteristics of each dataset. The study concludes with a discussion of the implications of these findings for machine learning model selection and suggests avenues for future research, including the refinement of model architectures based on detailed dataset analysis.

Keywords: Linear Models, Feedforward Neural Networks, Autoencoders, Classification, Regression, Model Complexity

1. Introduction

In machine learning, the selection of an appropriate model architecture is crucial for achieving optimal performance across various tasks. Linear models, often selected for their simplicity and interpretability, and neural networks, known for their powerful capacity to model complex non-linear relationships, represent two ends of the complexity spectrum in machine learning techniques. Among neural networks, feedforward neural networks (FFNs) and autoencoder-enhanced networks have gained popularity for their ability to handle high-dimensional data and perform intricate feature extraction.

Stemming from the early work on perceptrons in the 1950s (Rosenblatt, 1958), neural network research has evolved dramatically, leading to the development of deep learning architectures that address both regression and classification challenges with high performance. Autoencoders, introduced for unsupervised learning of efficient codings (Hinton and Salakhutdinov, 2006), further the capability of neural networks by enabling a form of feature learning that can be particularly effective for complex datasets.

This investigation focuses on the hypothesis that the performance of these advanced models (FFNs and autoencoder enhanced FFNs) does not uniformly surpass simpler linear models across all datasets. Instead, the effectiveness of each model is hypothesized to depend heavily on specific dataset characteristics such as the types of attributes, the complexity of the relationships among variables, and the overall task complexity.

To evaluate this hypothesis, the structure of this report is as follows: Section 2, *Algorithms and Experimental Methods*, outlines the frameworks of linear models, feedforward neural networks, autoencoders, and the experimental setup used for performance assessment. Section 3, *Results*, offers a detailed analysis comparing these models across a variety of datasets, utilizing metrics such as Mean Squared Error (MSE) for regression and cross-entropy loss for classification. In Section 4, *Discussion*, the implications of the findings are explored, particularly how dataset attributes and other considerations influence the differential performance between the models. The report concludes in Section 5 with conclusions and propositions for further research efforts.

2. Algorithms and Experimental Methods

The objective of this project was to evaluate the performance of linear models, feedforward neural networks (FFNs), and combined models incorporating autoencoder encoding layers for both classification and regression tasks. We developed implementations that allow for flexibility in handling different types of data and adapting to various experimental settings based on dataset characteristics. This section will offer a concise review of the datasets selected from the UCI Machine Learning Repository, outline the specifics of the linear models, FFNs, and the construction of autoencoder-based combined models, and describe the experimental approach employed to investigate our research hypothesis.

2.1 Data Sets and Preprocessing

Our study leveraged six datasets from the UCI Machine Learning Repository, divided equally for classification and regression tasks, to assess the performance of linear models, feedforward neural networks (FFNs), and combined models utilizing autoencoder encoding layers. To ensure a fair comparison of the algorithms' performance across these varied datasets, we standardized the data preprocessing procedures as much as possible. This involved encoding all ordinal and nominal attributes, normalizing the features using z-score standardization, and imputing missing values using the mean for continuous features and mode for discrete features respectively.

2.1.1 Abalone Dataset

A regression dataset aiming to predict abalone age from physical measurements, the Abalone dataset comprises 4177 instances with 8 attributes, predominantly continuous, except for the nominal Sex attribute.

2.1.2 Forest Fires Dataset

This regression dataset, consisting of 517 instances and 13 attributes, predicts forest fire areas from meteorological data. It presents a regression challenge, especially due to data skew towards smaller fires.

2.1.3 Computer Hardware Dataset

Featuring 209 instances and 10 attributes, this dataset aims at modeling relative CPU performance from hardware specifications. Attributes include various hardware specifications, with Vendor Name and Model attributes dropped for non-predictiveness.

2.1.4 Congressional Vote Dataset

A classification dataset based on the 1984 US Congressional Voting Records. It includes 435 instances with 16 votes each, aimed at predicting party affiliation (Democrat or Republican) from voting patterns. Missing votes were treated as abstentions.

2.1.5 Car Evaluation Dataset

This classification dataset assesses car acceptability over 1728 instances based on six attributes, categorizing cars into four acceptability classes. It evaluates cars on criteria such as price, maintenance cost, and safety, among others.

2.1.6 Breast Cancer Dataset

A classification dataset with 699 instances, each with 10 attributes. it predicts breast cancer malignancy from diagnostic imaging. Attributes cover various characteristics of cell nuclei from breast mass biopsies, classifying tumors into benign or malignant.

2.2 Algorithms

2.2.1 Implementing Linear and Logistic Regression

Linear and Logistic Regression predict outcomes by learning a linear relationship between the input features and the target variable.

Model Representation Both models can be described by:

- **Weights** ($\mathbf{W}$): A matrix of coefficients that linearly combine with the feature vector.
- **Bias** (b): An additional intercept term that allows the model to fit data that does not pass through the origin.

Activation Functions and Loss Functions

- **Logistic Regression** uses the sigmoid function to map predictions to probabilities:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

where z is the linear combination of weights and features. The cross-entropy loss, is used to measure the error:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

enhancing model fitting by penalizing incorrect classifications.

- **Linear Regression** directly uses the linear scores without additional transformation for continuous outcome prediction. The loss function used is the Mean Squared Error (MSE):

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

minimizing the average squared difference between the observed actual outcomes and the predictions.

Training Procedure Both models follow a gradient descent optimization procedure:

1. **Initialize Parameters:** Begin with randomly assigned small weights to prevent large initial predictions.
2. **Compute Model Output:** For logistic regression, calculate $z = XW + b$ and apply the sigmoid function. For linear regression, compute $z = XW + b$ directly.
3. **Calculate Gradients:** The gradient of the cross-entropy loss for logistic regression is:

$$\nabla_W L = -\frac{1}{N} X^T (Y - \sigma(XW)),$$

and for MSE in linear regression:

$$\nabla_W MSE = -\frac{2}{N} X^T (Y - XW),$$

where X is the matrix of input features, and Y is the vector of target values.

4. **Update Weights:** Adjust the weights by a fraction of the gradient scaled by the learning rate η:

$$W = W - \eta \nabla_W L \quad (\text{use } \nabla_W MSE \text{ for linear regression})$$

5. **Iterate Until Convergence:** Repeat the process for a maximum number of epochs or until the validation loss stops decreasing, implementing an early stopping criterion based on validation loss to prevent overfitting.

These linear models, while simpler than non-linear counterparts, are particularly effective for datasets where relationships between features and outcomes are approximately linear, offering the advantage of easy interpretability and computational efficiency.

2.2.2 Building Feedforward Neural Networks

Feedforward Neural Networks (FFNs) capture complex patterns in data through multiple layers of neurons. The architecture used here consists of two hidden layers, which allows for learning nonlinear functions and interactions between features effectively.

Network Architecture The network is structured as follows:

- **Input Layer**: Number of neurons equals the number of features in the dataset.
- **First Hidden Layer** (W_{hidden_1} and b_{hidden_1}): Transforms the input via weighted sums followed by a nonlinear activation (tanh).
- **Second Hidden Layer** (W_{hidden_2} and b_{hidden_2}): Further processes the data from the first hidden layer, using the same mechanism.
- **Output Layer** (W_{output} and b_{output}): Computes the final output of the network. The activation function here depends on the task: softmax for classification (multi-class) and identity for regression.

Activation Functions The activation function for the hidden layers is the hyperbolic tangent (tanh), which introduces non-linearity to the learning process:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

The output layer uses softmax for classification tasks, providing a probability distribution over classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

For regression tasks, the output layer's activation is directly the computed values.

Forward Pass The forward pass involves sequentially computing the outputs from the input layer to the output layer, using matrix operations followed by activation functions:

$$\begin{aligned}
Z_1 &= XW_{hidden_1} + b_{hidden_1} && \text{(First hidden layer input)} \\
A_1 &= \tanh(Z_1) && \text{(First hidden layer activation)} \\
Z_2 &= A_1 W_{hidden_2} + b_{hidden_2} && \text{(Second hidden layer input)} \\
A_2 &= \tanh(Z_2) && \text{(Second hidden layer activation)} \\
Z_{output} &= A_2 W_{output} + b_{output} && \text{(Output layer input)} \\
A_{output} &= \text{softmax}(Z_{output}) \text{ or } Z_{output} && \text{(Output depending on task)}
\end{aligned}$$

Backpropagation To train the network, the backpropagation algorithm is used:

1. Compute the gradient of the loss function concerning each weight and bias through the network, propagating errors backward from the output layer to the input.

2. Update the network parameters using gradient descent to minimize the loss. For each layer l:

$$W_l = W_l - \eta \frac{\partial \text{Loss}}{\partial W_l}, \quad b_l = b_l - \eta \frac{\partial \text{Loss}}{\partial b_l}$$

where η is the learning rate.

This training process involves multiple iterations over the training dataset, adjusting the network weights and biases based on the loss gradient computed from each batch of data. The network utilizes early stopping based on validation loss to prevent overfitting, reverting to the best state if no improvement is observed within a specified patience period.

2.2.3 Training AutoEncoders and Their Integration into a Combined Model

AutoEncoders are specialized types of neural networks that aim to encode input data into a smaller-dimensional space and then reconstruct the output to match the input. The AutoEncoder implemented here is designed with symmetric encoder and decoder components.

AutoEncoder Architecture The AutoEncoder architecture is characterized by:

- **Encoder**: Maps the input to a hidden representation using weights ($W_{encoder}$) and biases ($b_{encoder}$), with a sigmoid activation function to ensure the output values are between 0 and 1.
- **Decoder**: Attempts to reconstruct the input from the hidden representation, using a separate set of weights ($W_{decoder}$) and biases ($b_{decoder}$).

Forward Pass During the forward pass, the input data (X) is first transformed by the encoder to produce a hidden representation ($A1$):

$$Z1 = XW_{encoder} + b_{encoder}, \quad A1 = \sigma(Z1)$$

where σ denotes the sigmoid function. The decoder then attempts to reconstruct the input from this representation:

$$A_{output} = A1W_{decoder} + b_{decoder}$$

Backpropagation and Training The training involves adjusting the weights and biases to minimize the reconstruction error, typically using mean squared error (MSE) between the input X and the output A_{output}. The backpropagation updates are computed as follows:

$$\begin{aligned}
\delta_{decoder} &= A_{output} - X \\
\delta_{encoder} &= (\delta_{decoder} W_{decoder}^T) \odot A1 \odot (1 - A1) \\
W_{encoder} &\leftarrow W_{encoder} - \eta (X^T \delta_{encoder}) \\
W_{decoder} &\leftarrow W_{decoder} - \eta (A1^T \delta_{decoder})
\end{aligned}$$

where $\odot$ denotes element-wise multiplication, and η is the learning rate.

2.2.4 Combined Model Using AutoEncoder for Feature Extraction

The Combined Model utilizes the encoder part of a pre-trained AutoEncoder to initially extract features, which are then further processed through additional neural network layers designed for specific tasks such as classification or regression.

Model Architecture The architecture of the Combined Model extends the encoder with:

- **Second Hidden Layer** (W_{hidden_2} and b_{hidden_2}): This layer further processes the encoded features.
- **Output Layer** (W_{output} and b_{output}): Produces the final task-specific output, employing softmax for classification tasks or a linear activation for regression tasks.

Training and Forward Pass The training process involves updating not only the new layers' weights but also the weights of the encoder. This allows the model to fine-tune the feature extraction process specifically tailored to the downstream task, enhancing the overall model performance. The forward pass involves the following computations:

$$\begin{aligned} Z_2 &= A_1 W_{hidden_2} + b_{hidden_2}, \\ A_2 &= \tanh(Z_2), \\ Z_{output} &= A_2 W_{output} + b_{output} \end{aligned}$$

For classification tasks, the output activation function is softmax, providing a probability distribution over classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

For regression tasks, the output is given directly by Z_{output}, representing the predicted continuous values.

2.3 Experiment Design

The experiment is designed to evaluate the performance of three neural network architectures: a linear model, a standard feedforward network, and a combined model using features extracted by an autoencoder. The performance evaluation follows these steps:

1. **Hyperparameter Tuning**: Prior to performance evaluation, each model undergoes hyperparameter optimization through cross-validation where parameters such as learning rate, number of epochs, and hidden layer sizes are tuned based on the error metrics (loss and MSE). Random search is used over a predefined parameter space to identify the best settings. This tuning ensures that each model is efficiently optimized for the given data before performance evaluation begins.
2. **5x2 Cross-Validation**: For testing, a 5x2 cross-validation method is employed. This involves:

- Splitting the dataset into two halves five times, with each model trained on one half and validated on the other, then reversing the training and validation sets.
- This strategy helps in assessing the model stability and robustness against different data splits.

3. **Performance Metrics**: Metrics such as cross-entropy loss for classification tasks, and mean squared error for regression tasks, are recorded for each fold. The results from these metrics are then averaged to estimate the models' overall performance.

4. **Statistical Comparison**: By averaging performances across multiple runs and configurations, the experiment design aims to provide statistically reliable insights into how each model type performs under various data conditions.

This structured approach, including both parameter tuning and cross-validation, ensures a comprehensive evaluation of each neural network model, highlighting their performance and adaptability to different datasets and task requirements.

3. Results

This section presents the results obtained from evaluating three distinct neural network models on several datasets sourced from the UCI Machine Learning Repository. The aim was to determine and compare the effectiveness of linear models, feedforward neural networks (FFN), and combined models incorporating autoencoder-trained features across these datasets. Table 1 displays the mean squared error (MSE) performance metrics for the regression tasks, illustrating how each model type copes with the prediction of continuous outcomes. Table 4 provides the results for classification tasks, where each model's performance is evaluated based on the average cross-entropy loss, reflecting their accuracy in predicting categorical outcomes. Additionally, comparisons between the models are quantitatively analyzed, showing the statistical significance of performance differences through t-statistics and p-values, providing insight into the relative strengths of the models depending on the dataset characteristics.

Table 1: Regression Task Performance and Comparison

Table 2: Performance

Dataset	Algorithm	Average MSE
Abalone	Linear Model	5.082
	FFN Model	4.394
	Combined Model	4.345
Machines	Linear Model	5437.230
	FFN Model	6414.000
	Combined Model	5704.172
Forest Fire	Linear Model	2237.108
	FFN Model	3521.054
	Combined Model	3622.021

Table 3: Comparison

Comparison	t-statistic	p-value
Linear vs FFN	5.258	0.000053
Linear vs Combined	5.693	0.000021
FFN vs Combined	0.609	0.550
Linear vs FFN	-0.603	0.554
Linear vs Combined	-0.169	0.868
FFN vs Combined	0.359	0.724
Linear vs FFN	-2.569	0.019
Linear vs Combined	-2.622	0.017
FFN vs Combined	-0.244	0.810

Table 4: Classification Task Performance and Comparison

Table 5: Performance

Dataset	Algorithm	Avg CE Loss
Car	Linear Model	0.397
	FFN Model	0.072
	Combined Model	0.100
Breast Cancer	Linear Model	0.098
	FFN Model	0.103
	Combined Model	0.131
House Votes	Linear Model	0.100
	FFN Model	0.102
	Combined Model	0.104

Table 6: Comparison

Comparison	t-statistic	p-value
Linear vs FFN	58.165	6.06e-22
Linear vs Combined	32.782	1.67e-17
FFN vs Combined	-3.067	0.0066
Linear vs FFN	-0.543	0.593
Linear vs Combined	-2.501	0.022
FFN vs Combined	-1.944	0.068
Linear vs FFN	-0.195	0.848
Linear vs Combined	-0.348	0.733
FFN vs Combined	-0.119	0.907

4. Discussion

This study investigated the performance of linear models, feedforward neural networks (FFNs), and autoencoder-enhanced neural networks across a variety of datasets for both classification and regression tasks. As detailed in Tables 1 and 4, the experimental results confirm the hypothesis that no single model architecture consistently outperforms others across different datasets.

In regression tasks, the performance was evaluated using Mean Squared Error (MSE), where lower scores indicate better model accuracy. Interestingly, FFNs and combined models did not always outperform linear models. For example, in the Machines dataset, the linear model was not statistically outperformed by the FFN, as indicated by a non-significant negative t-statistic (-0.603), suggesting similar performances between the two. However, the Forest Fire dataset showed that both the FFN and combined models performed worse than the linear model, evidenced by negative t-statistics of -2.569 and -2.622 respectively, indicating the linear model's relative strength in handling this dataset's characteristics.

For classification tasks, the analysis was based on cross-entropy loss, where again, lower scores are preferable. The Car dataset demonstrated a significant improvement in FFN and combined models over the linear model, with t-statistics of 58.165 and 32.782 respectively, both positively significant, highlighting the advanced models' better fit for this dataset. Conversely, in the Breast Cancer dataset, while the linear model performed comparably to the FFN, as shown by a non-significant negative t-statistic (-0.543), it outperformed the combined model (t-statistic of -2.501), suggesting that the autoencoder's feature extraction capabilities were not particularly beneficial here.

These results emphasize the nuanced relationship between model complexity and dataset characteristics. The linear models, despite their simplicity, were not consistently outperformed by more complex models across all datasets, which challenges the notion that more layers or pre-training (as in autoencoders) invariably lead to better performance. In cases where the data complexity and relationships within the features were less pronounced, simpler models were sufficiently effective, demonstrating that increased complexity can sometimes lead to diminishing returns or overfitting, particularly in datasets not suited for such models.

5. Conclusion

This study investigated the comparative performance of linear models, feedforward neural networks (FFNs), and autoencoder-enhanced neural networks across diverse machine learning tasks, utilizing datasets from the UCI Machine Learning Repository. The analysis centered on both regression and classification tasks, providing a detailed assessment of how different model complexities adapt to and perform on varied datasets.

Our findings indicate that while advanced models such as FFNs and autoencoder-enhanced networks often provide superior performance in scenarios involving non-linear and high-dimensional data, they do not universally outperform simpler linear models. This underscores the importance of aligning model complexity with dataset characteristics to optimize performance. The varied performance results across different datasets validate the hypothesis that no single model architecture consistently excels, highlighting the necessity of tailored model selection based on specific task requirements and data properties.

Future research could delve deeper into refining these models to enhance their efficiency and effectiveness. Potential avenues include the exploration of more sophisticated network architectures, such as deep learning models. Additionally, investigating the impact of different activation functions, regularization techniques, and loss functions could provide insights into further optimizing model performance. Another area of improvement could come from further hyperparameter tuning. For this experiment, due to hardware limitations, the hyperparameter search space was restricted and random selection was used to keep the training times within a reasonable limit. Increasing the hyperparameter optimization search space as well as using other optimization techniques like ADAM (Adaptive Moment Optimizer) in training might lead to better performance.

In summary, this study not only demonstrates the varied capabilities of linear models, FFNs, and autoencoder-enhanced networks but also highlights the critical role of appropriate model selection in machine learning. By aligning model architecture with dataset specifics, one can significantly enhance the predictive accuracy and generalizability of their machine learning solutions.

References

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.