Individual Assignment: Predicting Abalone Age from Categorical and Continuous Data Using Deep Learning

Mikolaj Hilgert | March 2025

1. Overview and Key Novelties

This project focuses on predicting the age of abalones based on physical measurements using Deep Learning with PyTorch. The age of an abalone corresponds to the number of rings in the abalone shell. The data consists of a tabular dataset with a mix of categorical and continuous features. For this purpose, a regression task is employed to predict the abalones age. This report discusses the decisions and results made throughout. The *README.txt* outlines instructions on how to run the code in inference mode.

Although the target values are integers, this assignment is framed as a regression problem, over classification. This allows to preserve the 'ordinal' nature of age and allows for measuring how close predictions are, unlike classification which would penalise all errors equally.

The key novelties of this project include:

- **Architecture Comparison**: A (simple) baseline model is created in addition with a more advanced model. A comparison is made between them to evaluate whether the added complexity brings tangible benefits.
- **Treatment as Regression problem**: As mentioned earlier, the problem is approached as a regression rather than a classification problem.
- **Extensive Experimentation:** The process involved a significant amount of experimentation with different design and architectural choices, to bring refined analysis.

2. Data Processing

An initial pairplot of the training data showed most distributions of the numerical features are right-skewed. The correlation matrix indicated a multi-collinear relationship between features. Given these insights and the requirement for micro-batch training, the data needed to be processed. Following PyTorch best practices, a custom PyTorch *Dataset* class was defined to handle the dataset. The *__init__* function loads the data and applies the following two transformations, given they are static transformations, they take place once:

1. **One Hot Encoding**: The dataset has one categorical feature, Sex can be one of the following 3 classes: M (male), F (female), I (infant). This is achieved using *pandas* and *PyTorch* to map each class to a tensor (eg, M → [1,0,0]).
2. **Standardization**: Transforms features to have a mean of 0 and unit variance (1), that all features contribute equally to the model's learning process (Blok, 2025), this is achieved using *numpy*.

When the dataset class is used in 'training' mode, the labels are also included in the data. The *__len__* and *__getitem__* functions are implemented in a standard fashion.

3. Neural Network Architecture and Design

As mentioned, a baseline model is defined to compare whether added complexity yields significant results. The two models are defined in the table below:

Network Feature	Baseline Model	Advanced Model
Architecture	Two hidden layers (64 → 32 → 1 neurons)	Same as Baseline
Activation Function	*ReLU* for stable gradients and efficient learning	Same as Baseline
Batch-Normalisation	None	*BatchNorm1d* to reduce internal covariance shift and stabilise training
Regularisation	None	*L1 regularisation* (Lasso) to reduce redundant feature use

There was quite a bit of experimentation during the definition of these final features; various changes were attempted, and the current architecture gave the best performance. The architecture was set at two hidden layers (shallow) due to the small size of the dataset. Different number sets of neurons in hidden layers were also extensively tested. The activation function was chosen as *ReLU*, with testing using Tanh also being made. In terms of the advanced model, batch normalization was included in an attempt to reduce internal covariate shift, thereby stabilising and accelerating the training process (Blok, 2025). Additionally, regularisation was introduced to improve generalization and prevent overfitting; *L1 regularisation*, in particular, was selected to address the high multicollinearity of features by encouraging sparsity in the model weights. Other regularisation methods were tested, including *Dropout*, but *L1* gave better performance.

4. Training and Inference protocol

The training dataset was randomly split 80% for training and 20% for validation. The training procedure involved micro-batch training of size 32 using the PyTorch DataLoader, which included shuffling for the training set. The selected loss function is MSELoss, this is due to the fact this is a regression task as well as the ability to compute an interpretable RMSE result.

The training was set for 100 epochs. The implementation of the training code includes early stoppage, namely with a patience of 25 epochs, which works on the basis that if in that time the validation loss does not improve, training stops. If a new ‘best’ validation loss is achieved (the smaller the better), the model weights of the best performing epoch are saved such that they can be loaded later for inference. The Adam optimiser is also used, which combines the advantages of RMSprop and Momentum (Blok, 2025), with the learning rate (0.001) which performed best.

For inference, the best model weights are loaded, and the model is switched to evaluation mode with a *model.eval()* function call. Then *torch.no_grad()* is used to iterate through the test DataLoader and make predictions. These predictions are then appended to the original test dataset and saved to *test_with_predictions.csv*. The code also allows a variable to set whether the prediction should be rounded to the nearest integer (if this is at any point needed).

5. Evaluation, Challenges, and Future Improvements

This section goes over the results, touches upon some challenges and improvements to be made.

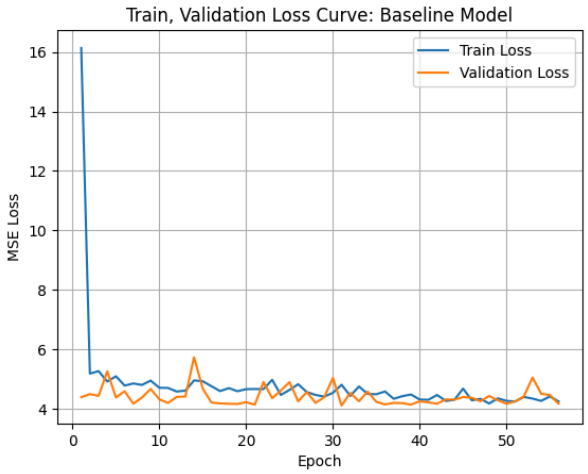

Figure 1: Baseline Model Loss Curve (Best MSE: 4.10)

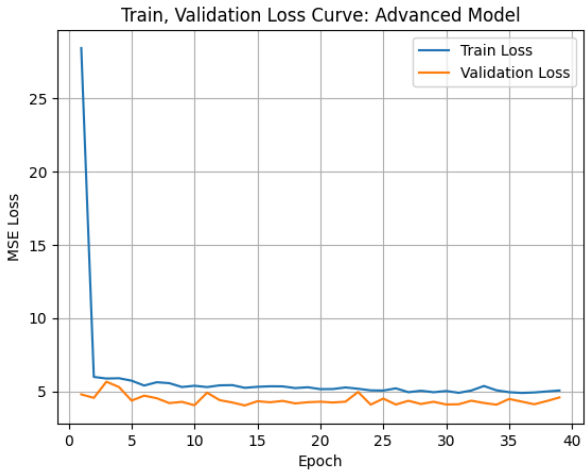

Figure 2: Advanced Model Loss Curve (Best MSE: 4.04)

Looking at the above results, the performance between the two models is very similar. One clear thing to note is that with the advanced model, the validation loss is much more stable, confirming that BatchNormalisation can help in overfitting. It is noted that in both cases there was an early stoppage. Reasoning about the results, the advanced model has an RMSE of ~2, meaning that on average, the model's predictions are within approximately 2 units of the true abalone age.

In terms of encountered challenges, it was difficult to motivate choices for the NN architecture, therefore most of it was based on information found on the slides as well as extensive experimentation. It was also a bit discouraging to see marginal differences between the baseline and any version of the advanced model, even if many drastic changes were tested.

To further enhance model performance, more data could be collected that is not so highly correlated with the other features. Additionally, exploring ensemble techniques or deeper network design may help reduce prediction error and improve model generalisation more drastically, given that the current setup despite tweaks has not yielded significant improvements.

6. Conclusion

This experiment seems to have successfully achieved the goal of predicting abalone age based on physical measurements, showing solid performance and stable learning in the advanced model.

References

Blok, P. (2025). *Course modules- Deep Learning: Lesson 6 - Pytorch in Production*. https://tilburguniversity.instructure.com/courses/17218/modules/items/830578