Estimating The Health of Helicopter Turbine Engines by Means of Regression and Classification Using a Probabilistic Neural Network

Tyler Romano[1], Nathan Siegel [2], Samuel T. Willis III [3], William Henn[4], Rishie Seshadri [5]

[1,2,3,4,5] *Belcan, Windsor, CT,06095,United States*
tromano@belcan.com
nsiegel@belcan.com
stwillis@belcan.com
whenn@belcan.com
rseshadri@belcan.com

Abstract

This paper presents team Mad SoftMax's approach to the 2024 data challenge presented by the Prognostics and Health Management Society. The competition tasked participants with estimating the health of helicopter turbine engines by calculating torque margin via regression and classifying engines as either healthy or faulty. Probabilistic regression was employed to estimate the torque margin at each measurement, and a neural network classifier was used to categorize each observation within the dataset as belonging to a healthy or faulty engine. Both the regression and classification networks were developed using open-source libraries such as TensorFlow. These networks were tested in isolation using training data and evaluated for performance before integration for use on evaluation data. The team was able to successfully construct a system of models that achieved a final score of 0.849 out of a maximum score of 1.

1. Introduction

The 2024 PHM Society Data Challenge tasked participants with estimating the health of helicopter turbine engines (PHMSociety, 2024). The data for this year's challenge was gathered from seven helicopter engines of the same make and model. These engines each captured metric data such as: outside air temperature *oat*, mean gas temperature *mgt*, pressure altitude *pa*, indicated airspeed *ias*, net power *np*, compressor speed *ng*, and torque measured trq_{meas}. The data was split into three datasets, a training set, a daily test dataset, and a final validation dataset. The engines in all sets of data were shuffled and had any asset identifiers removed to anonymize the data. The initial four engines made up the training dataset while the remaining three were withheld for the testing and validation phases. The test data was available to test against daily to allow for teams to validate their approach methods of classifying engine health and determination of torque margins. The validation data, however, was only available to run a single time at the end of the challenge.

The teams entered in the challenge were asked to provide three items for each data point. Teams were asked to classify the health of the engine as either nominal or faulty, as well as assigning a confidence to that engine state determination. In addition, teams were tasked with determining an estimation for torque margin expressed as a probability distribution for each data point. Daily scoring of models was available to allow teams to compare against testing data to ensure the accuracy of their approach.

2. Methods

Upon entering the competition, the team sought to understand the importance of each parameter within the dataset. The first step in this process was to separately learn about the environmental and engine performance parameters to determine if and how these parameters may influence each other. The environmental parameter set made up of *oat*, *pa*, and *ias* helped to understand the flight envelope without additional data such as weight and true altitude (FAA, 2019). The engine performance parameters included: *mgt*, *np*, *ng*, and trq_{meas}. While trq_{meas} and the torque margin trq_{mar} were provided in the training set, the target torque trq_{tgt} was not. The relation between the three parameters is shown in Eq. (1), which was provided in the problem statement (PHMSociety, 2024).

$$trq_{mar}(\%) = 100 * \frac{trq_{meas} - trq_{tgt}}{trq_{tgt}} \tag{1}$$

$$trq_{tgt} = \frac{trq_{meas}}{\frac{trq_{mar}}{100} + 1} \tag{2}$$

By manipulating Eq. (1) into the form shown in Eq. (2), trq_{tgt} may be calculated from the parameters provided in the training dataset.

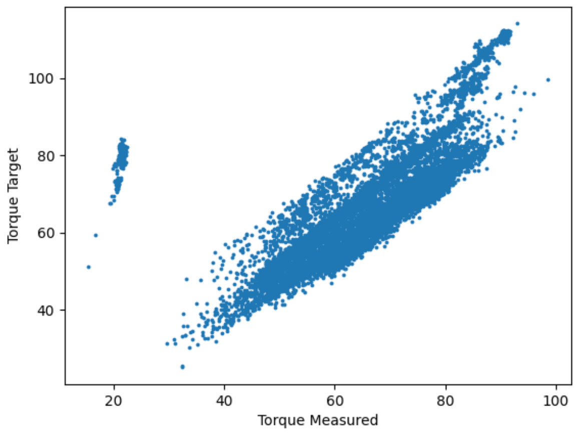

Figure 1. Calculated torque target vs measured torque for the training set.

By using the calculated trq_{tgt} values, it is possible to train a neural network to predict a probabilistic distribution representing $\hat{trq}_{tgt}$. The neural network structure chosen for this task was a multi-layer perceptron, or MLP, with output features that defined a probability distribution. The classification model was also chosen to be an MLP, although with a different configuration. The classification network was designed to use the predicted $\hat{trq}_{mar}$ generated by the probabilistic model alongside the given environmental parameters as inputs.

While a deterministic neural network can predict a singular value given a set of inputs, a probabilistic model instead predicts a distribution that represents that value and an associated confidence metric. To appropriately characterize that currently unknown distribution, sources of inaccuracy must be considered. Identified sources of inaccuracy include sensor noise in the input data, noise inherent to the process that generated the ground truth trq_{mar}, and inaccuracy present in the model itself. Unfortunately, no information was provided in the problem statement regarding sensor accuracy or the process used to determine trq_{tgt}. The third source, the model's inherent inaccuracy, is at least known to be symmetrical, given that the distribution chosen to be trained on in subsection 2.1 is also symmetrical. Since over 740,000 points were provided in the training set, the team decided that the central limit theorem applied (Rouaud, 2013, p. 10), and decided to model all the identified sources of noise as a zero-mean normal distribution with standard deviation σ, as shown in Eq. (3). Without additional information, it would be difficult to infer more detail about the distribution, such as bias, skew, or kurtosis.

$$\hat{trq}_{tgt} = trq_{tgt} + \mathcal{N}(0, \sigma^2) \tag{3}$$

2.1. Probabilistic Model

The probabilistic model was designed using both the base TensorFlow library's Keras API (Keras, 2024) as well as TensorFlow Probability (*TensorFlow Probability*, n.d.), a supplementary library that allows easy integration of probabilistic elements into standard TensorFlow models. The network structure chosen, shown in Figure 2, was an MLP architecture made up of the following layers: layer 1, a 6-parameter input scaling layer, layers 2 & 3, 256-node layers using the sigmoid activation function described in Eq. (4), layer 4, a 2-node linear scaling layer, and layer 5, an IndependentNormal layer from TensorFlow Probability. This final layer takes the two outputs from layer 4 and interprets them directly as the parameters $\hat{\mu}_{tgt}$ and $\hat{\sigma}_{tgt}$, defined in Eqs. (5, 6), of the predicted distribution $\hat{trq}_{tgt}$. If this were an otherwise identical deterministic neural network, layer 4 would be a single node, and layer 5 would not be present. As a result, the network would only be capable of learning the expected value of the output feature $\hat{trq}_{tgt}$ and could not characterize the error associated with that prediction.

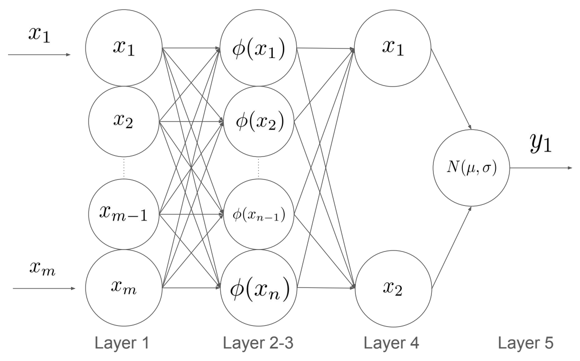

Figure 2. MLP architecture with probabilistic output features for a regression problem.

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The Adam optimizer, a popular first-order optimizer (Kingma & Ba, 2014), was used to train the network. The learning rate, LR, was decreased as training went on, allowing the network to converge much tighter than if a constant value were set.

The six chosen input parameters for this network were: *oat*, *mgt*, *pa*, *ias*, *np*, and *ng*, and the singular output parameter was a normal distribution predicting $\hat{trq}_{tgt}$. This output distribution is then converted to torque margin for the final probabilistic output. First, the mean of the predicted torque margin, $\hat{\mu}_{mar}$, is calculated in Eq. (5) using Eq. (1) and $\hat{\mu}_{tgt}$, the mean of the predicted torque target distribution, since the mean value of the target is the expected value.

$$\hat{\mu}_{mar} = 100 * \frac{trq_{meas} - \hat{\mu}_{tgt}}{\hat{\mu}_{tgt}} \tag{5}$$

The standard deviation of the torque margin, $\hat{\sigma}_{mar}$, is calculated using Eq. (6).

$$\hat{\sigma}_{mar} = 100 * \frac{\hat{\sigma}_{tgt}}{\hat{\mu}_{tgt}} \tag{6}$$

Both Eq. (5) and Eq. (6) require the observation that the measured torque satisfies Eq. (7) and Eq. (8) and that the distribution representing $\hat{trq}_{mar}$ is scaled relative to the expected value of $\hat{trq}_{tgt}$.

$$\mu_{meas} = trq_{meas} \tag{7}$$

$$\sigma_{meas} = 0 \tag{8}$$

Typically in deterministic approaches for regression, a mean squared error method is used to fit the model to the data (Terven, Cordova-Esparza, Ramirez-Pedraza, Chavez-Urbiola, & Romero-Gonzalez, 2024). This approach is useful for finding the expected value of each point and making predictions, however, it does not provide any information about the distribution around that point. A second approach is to maximize the likelihood that the training data matches an estimated distribution (Adams, 2018). Adams shows that in the case where the noise is Gaussian and the standard deviation σ is known and constant, the problem reduces to the least-squares method to estimate the mean $\hat{\mu}$. To fully predict the distribution, σ must also be allowed to vary and be fit with a function. Adams also describes this issue, and his solution is another maximization problem. By considering both of these maximization problems both $\hat{\mu}$ and $\hat{\sigma}$ may be predicted, fully defining the probability distribution output by a neural network.

However, it can be difficult to maximize both of these problems directly. Adams explains how the log-likelihood is easier to manipulate and compute than the likelihood, and that the negative log-likelihood is used when a minimization problem is desired instead (Adams, 2018). For this reason, the loss function selected was the negative log-likelihood. This way, the optimizer, in this case Adam (Kingma & Ba, 2014), can minimize the loss, thus maximizing the probability that the training sets points are part of the neural network's distribution.

Over the whole training dataset, models that have accurate means but wide distributions and models that have inaccurate means but narrow distributions both have a sub-optimal log-likelihood. A high log-likelihood, and thus a minimized negative log-likelihood, is only possible to achieve by matching the mean and distribution output of the network to the mean and distribution of the dataset.

2.2. Classification Model

Multiple different methods were used to classify the engines' fault status, eventually converging on the method yielding the most performant model. The initial approach solely used the statistical properties of trq_{mar}, while the subsequent two used different neural-network structures to approach the classification problem.

The initial method did not use machine learning and was chosen for its simplicity. As observed in Figure 3, while there is substantial overlap between trq_{mar} for known nominal and faulty engines in the training set, most points should still be possible to accurately evaluate if nominal or faulty. Generally speaking, significantly positive values for trq_{mar} indicate nominal status, while significantly negative values indicate faulty status.

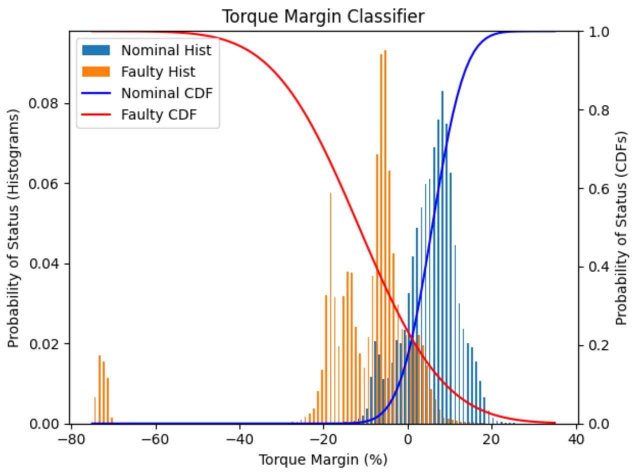

Figure 3. Fault classification histograms and cumulative distribution functions for nominal and faulty engine data.

By calculating normal distributions for both fault types, the probability that any torque margin value t is nominal can be calculated by the cumulative distribution function in Eq. (9), and that the variable is faulty by calculating the survival function as in Eq. (10) where T_n and $T_{\mathfrak{f}}$ are normally distributed random variables for the nominal and faulty engines respectively, and P_n and $P_{\mathfrak{f}}$ are the probability density functions for those same engine classes. Eq. (9) and Eq. (10) differ because the probability the engine is nominal increases as trq_{mar} increases, but the same change causes the probability the engine is faulty to decrease.

$$F_n(t) = P_n(T_n \leq t) \tag{9}$$

$$S_{\mathfrak{f}}(t) = P_{\mathfrak{f}}(T_{\mathfrak{f}} > t) \tag{10}$$

If Eq. (11) is satisfied, then the data point is considered nominal. If Eq. (12) is satisfied, then the data point is considered faulty.

$$F_n(t) > S_{\mathfrak{f}}(t) \tag{11}$$

$$F_n(t) < S_{\mathfrak{f}}(t) \tag{12}$$

To calculate the confidence *C* of this model, Eq. (13) is used. This way, the confidence evaluates to 0 when both probabilities are equal and to 1 when one of them is maximized.

$$C = \left| \frac{F_n(t) - S_{\mathfrak{f}}(t)}{F_n(t) + S_{\mathfrak{f}}(t)} \right| \tag{13}$$

To use this method, the cumulative distribution functions, or CDFs, are calculated from the training set and Eqs. (11, 12, 13) are evaluated at $t = \hat{\mu}_{mar}$, the mean of the probabilistic regression network's predicted torque margin distribution. While this approach provided decent results, there was too much ambiguity in the region where the two distributions overlap, resulting in low classification confidence in that region. A more complex approach involving neural networks was used to increase the confidence in the overlapping regions.

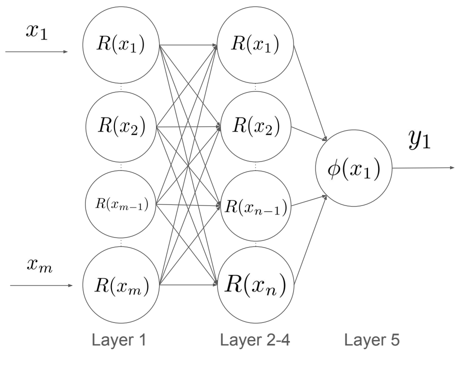

Figure 4. Deep MLP architecture for a classification problem.

As with the probabilistic model, the neural network classifiers use the Keras API (Keras, 2024) to structure, train, and run. The second classification architecture that was developed, shown in Figure 4, followed an MLP architecture and was made up of the following layers: layer 1, an 8-node input layer using the rectified linear unit, or ReLU, activation function shown in Eq. (14), layers 2-4, each 20-node hidden layers also using ReLU, and layer 5, a single-node output scaling layer using the sigmoid activation function in Eq. (4). The deeper configuration was chosen to promote stronger interconnection of the hidden layers compared to the regression network's flatter architecture, which is usually desirable for a classification problem where complex modeling ability is needed.

$$R(x) = \max(0, x) \tag{14}$$

The input parameters chosen for this network were $\hat{trq}_{mar}$, trq_{meas}, oat, mgt, pa, ias, np and ng. Since the predicted torque margin by itself was insufficient to fully determine the fault state of the engine, the assumption was made that other environmental factors could provide additional information, even if the direct influence those parameters had was unknown.

While Adam was still used as the optimizer for this network, a binary cross-entropy loss function was chosen as an appropriate loss function for the 1-label output (Terven et al., 2024, p. 13). For nominal engines, the desired output of the network is 0 and for faulty engines, the desired output is 1. For intermediate values, the closest label was chosen as the identified label, and the confidence C was calculated from the output value x using Eq. (15).

$$C = 2\,|x - 0.5| \tag{15}$$

Like in Eq. (13), Eq. (15) evaluates to 0 when the network is uncertain of the state, occurring when $x = 0.5$ in this case, and evaluates to 1 when the network is fully certain of the output state. This network performed much better when tested against a subset of the training set reserved for verification than the single-input method, and also performed similarly well against the test set.

Like the second model, the third also followed an MLP architecture. This network, shown in Figure 5, was made up of the following layers: layer 1, a 7-node input layer, layer 2, a 32-node hidden layer using the sigmoid activation function in Eq. (4), and layer 3, a 2-node output layer using the softmax activation function shown in Eq. (16). This network was initially tested with a range of up to 100 hidden nodes, but significant overfitting was noticed at higher node counts. The final number of nodes, 32, was chosen as a balance between potential to over fit the data and overall accuracy.

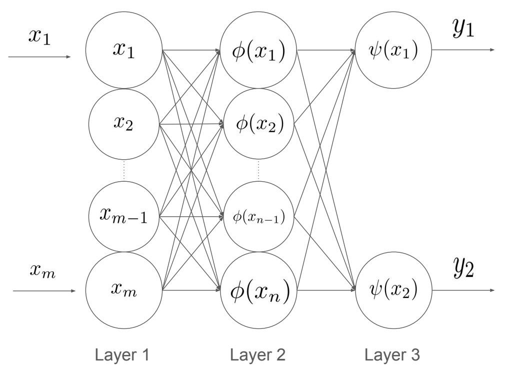

Figure 5. Shallow MLP architecture for a classification problem.

$$\psi(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{16}$$

The input parameters for this network differed from the deeper architecture. While that network used the measured torque and the predicted torque margin in conjunction, this network solely used the torque margin. Thus, the chosen input parameters were $\hat{trq}_{mar}$, oat, mgt, pa, ias, np, and ng. This decision was made because $\hat{trq}_{mar}$ is derived from trq_{meas} and $\hat{\mu}_{tgt}$ using Eq. (5). Since the relationship between trq_{meas} and $\hat{trq}_{mar}$ is already known, it is not necessary to include trq_{meas} as an input parameter if $\hat{trq}_{mar}$ is provided.

Adam was also used as an optimizer for this network, alongside the categorical cross-entropy loss function. This choice was made because there was more than a single identified output label for this network, making the binary cross-entropy inappropriate for such an application (Terven et al., 2024, p. 14). The nominal label x_n was expected to be equal to 1 when the engine was identified as nominal and the faulty label $x_\mathfrak{f}$ was expected to be equal to 1 when the engine was identified as faulty.

Fault labeling and confidence calculation were done similarly to the first classification approach, as shown in Eqs. (11, 12, 13). If Eq. (17) is satisfied, as with Eq. (11), the data point is considered nominal. Similarly, if Eq. (18) is satisfied, as in Eq. (12), the data point is considered faulty.

$$x_n > x_\mathfrak{f} \tag{17}$$

$$x_n < x_\mathfrak{f} \tag{18}$$

Like in Eqs. (17, 18), Eq. (13) needs to be adapted to Eq. (19) to use the new labels to calculate the confidence C.

$$C = \left| \frac{x_n - x_\mathfrak{f}}{x_n + x_\mathfrak{f}} \right| \tag{19}$$

Both neural-network approaches were developed in tandem and compared at different points throughout the challenge. To accurately compare them a subset of the training data that was not used in training in either network was used. While the deep network performed better in these tests, both performed better than the initial approach, so both were tested during the testing phase. The wide neural network was discovered to have performed better than the deep network on the test set, which was unexpected. Because of this, the wide neural network was selected for final use.

2.3. Model Training and Testing

While both the classification and regression networks were trained independently, they were combined as shown in the runtime architecture shown in Figure 6. This architecture feeds the $\hat{trq}_{mar}$ calculated by the regression network into the classification network in place of trq_{mar}. This allowed the classification model to make predictions without access to trq_{mar}, with high accuracy so long as the $\hat{trq}_{mar}$ estimate was accurate.

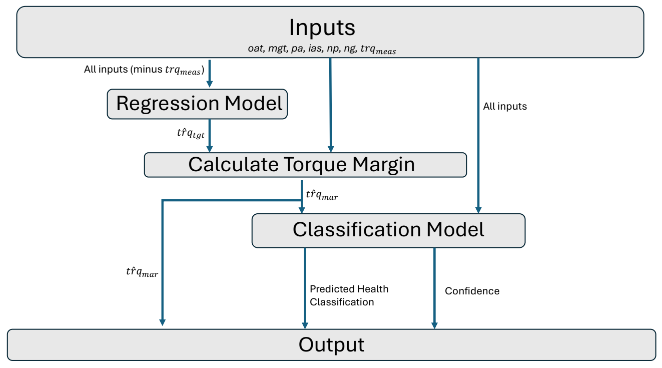

Figure 6. Integrated system flow for calculating $\hat{trq}_{mar}$ and classifying health state of engine.

The provided training data was split into a training set and an internal test set using an 80% to 20% batching process. Both the classification and regression models were evaluated using the 20% of points reserved for the internal test set after training had completed. This allowed rapid tests of newly developed approaches and revisions to existing models without waiting for the daily score of the test set to be released.

Table 1. Hyperparameters used for training the probabilistic regression neural network. Adam was always configured with $\beta_1 = 0.9$ & $\beta_2 = 0.99$.

Tuned Parameters				Result
Version	Nodes	Epochs	LR	Loss
v1.0	64	0-1k	10^{-4}	0.2824
v2.0	128	0-1k	10^{-5}	-0.1218
v3.0	256	0-2k	10^{-5}	-1.1662
v3.1	256	2k-2.1k	10^{-7}	-2.5097
v3.2	256	2.1k-3k	10^{-6}	-2.8252
v3.3	256	3k-3.5k	10^{-6}	-2.8941
v3.4	256	3.5k-4k	10^{-7}	-3.1786
v4.0	300	0-4k	10^{-5}	-2.5140
v4.1	300	4k-5k	10^{-5}	-2.6593
v4.2	300	5k-6k	10^{-5}	-2.7349
v4.3	300	6k-7k	10^{-6}	-4.0981

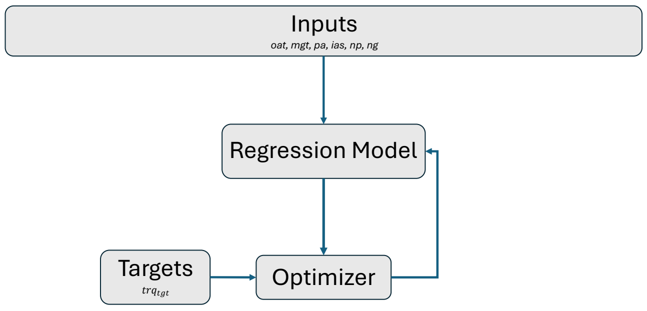

Figure 7. Training flow for probabilistic regression neural network.

The probabilistic regression network was trained with the architecture shown in Figure 7. Since the ground truth trq_{mar} was provided by the training dataset, the exact value of trq_{tgt} could be calculated, and was used for training. Initially, the number of nodes was kept small, but the required accuracy was not reached. As the number of nodes and training epochs was increased the loss was observed to decrease further and further. As training progressed and the trained distribution $\hat{trq}_{tgt}$ more closely approximated trq_{tgt}, the learning rate, had to be decreased in order for further improvements to be made, as shown in Table 1. Once the network grew above 256 nodes in the hidden layers, the testing batch started to perform worse overall despite the loss continuing to decrease. This was detected by a score decrease in the daily test set runs despite a decrease in training set loss, which is observed in Table 3.

This performance issue was thought to be due to overfitting, so a 256-node model that performed better on both the local test batch and the daily test set was chosen for the final submission.

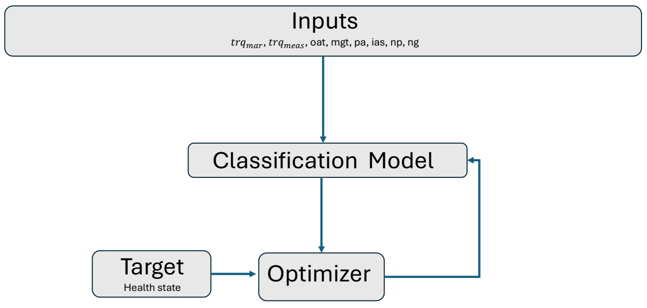

Figure 8. Training flow for fault classification neural network.

The neural network approach for training the fault classifier underwent a similar process, using a similar architecture, shown in Figure 8. Since the training dataset provided the ground truth trq_{mar}, the exact value could used as an input for training, although that value would not be available in the main runtime architecture and would be replaced by $\hat{trq}_{mar}$. By increasing the number of training epochs, nodes, and layers, lower losses and higher test batch accuracy was achieved, as shown in Table 2. While both types of network exhibited good overall performance, the deep neural network achieved lower loss and higher accuracy on the local test batch so the assumption was made that the deep neural network would perform better on the daily test set evaluation.

Table 2. Hyperparameters used for training the classification neural network. Adam was always configured with $\beta_1 = 0.9$ & $\beta_2 = 0.99$.

Tuned Parameters					Result
Deep Classifier					
Version	Nodes	Layers	Epochs	LR	Loss
v1.0	20	3	100	10^{-3}	0.0431
Wide Classifier					
Version	Nodes	Layers	Epochs	LR	Loss
v2.0	32	1	10	10^{-4}	0.1016
v3.0	64	2	30	10^{-5}	0.0735
v4.0	64	2	100	10^{-5}	0.0606

However, that assumption was found to be untrue. The wide neural network significantly outperformed the deep network on the daily test set as shown in Table 3, despite having the highest loss of the networks present in Table 2. Because of this, the wide network was selected to process the final validation dataset. This network was surprisingly small, with only 32 nodes, a single layer, and only a few training epochs. Despite this, the wide neural network performed very well.

2.4. Validation Phase

During the testing phase, a submission could be uploaded and scored once a day. This meant that the team could train different models, decide which one was best to submit to be scored, and have feedback the next morning to see if improvements were made over the previous model. For the validation submission, the set of models that were selected to process the validation submission was the set that scored the highest of the team's attempts, with a score of 0.937 on the test set, some examples of which are shown in Table 3. The selected combination of models was not the most recent model trained, as changes made later on in the testing phase ended up resulting in lower scores.

The team decided to use the combination of models with the highest score on the test set to run the validation set for final submission, as shown in Table 3. Since these models performed better on the test set than others, despite not having the best score on the reserved portion of the training set, it was believed that these models would generalize better to unknown engines. The selected combination scored a 0.849 on the validation dataset.

3. RESULTS

The team was able to successfully produce a probabilistic MLP that could accurately estimate the torque margin to high accuracy as well as a classification model that could more accurately predict the health of an engine than the torque margin alone could.

Table 3. Selected full-system runs with combined output score.

Model Versions			Score
Regression	Classification	Dataset	
v2.0	CDF	Test	0.6319
v2.0	v2.0	Test	0.6532
v3.0	v2.0	Test	0.7327
v3.1	v2.0	Test	0.8764
v3.2	v2.0	Test	0.9358
v3.4	v2.0	Test	0.9370
v4.2	v2.0	Test	0.9247
v4.3	v4.0	Test	0.8861
v3.4	v1.0	Test	0.8543
v3.4	v2.0	Validation	0.8490

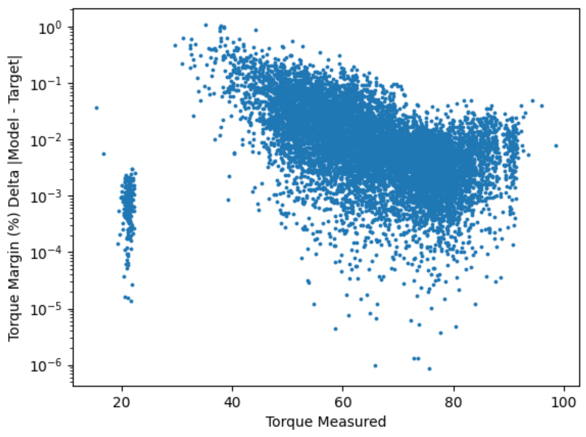

Figure 9. Absolute value of the model-calculated torque margin's delta with respect to the known torque margin.

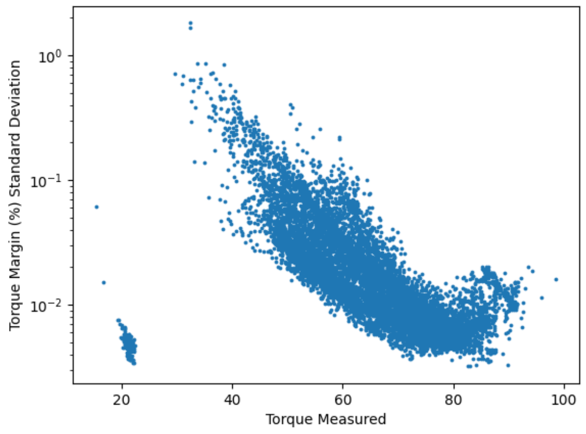

Figure 10. Model-predicted standard deviation of the torque margin.

As stated in subsection 2.4, the team was able to achieve a test score of 0.937 and a validation score of 0.849 with the chained neural network system. Since each of the datasets each have a different mix of engine assets that make up their data points, the strong performance across datasets indicates that both the probabilistic neural net and the classifier were able to generalize well and had not overfit significantly on any features of the engines in the training set. Figures 9 and 10 show that the final probabilistic regression network was able to accurately characterize $\hat{trq}_{mar}$, both achieving overall high accuracy for $\hat{\mu}_{mar}$ and predicting higher $\hat{\sigma}_{mar}$ where the model was less accurate, correctly indicating lower confidence. Figures 12 and 11 show the confidence of correctly and incorrectly classified points, respectively. As can be seen, the majority of points in Figure 12 have high classification, while the points in Figure 11 show mid-to-low confidence.

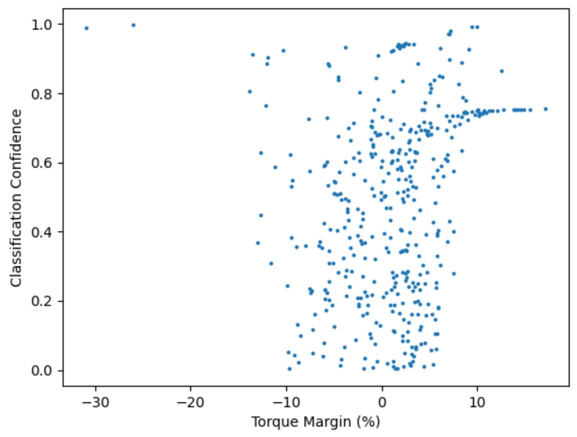

Figure 11. Confidence of incorrectly classified points from the training set.

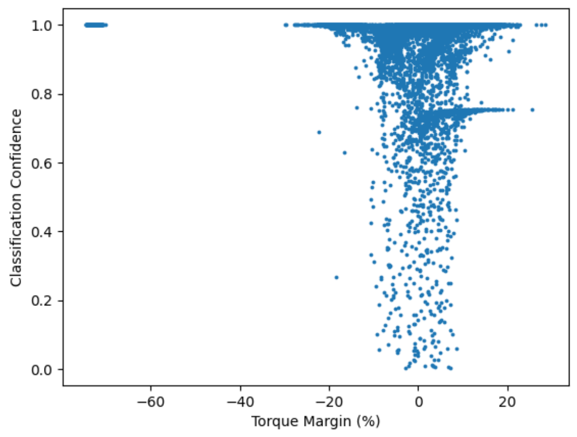

Figure 12. Confidence of correctly classified points from the training set.

4. Conclusion

To develop a novel and robust solution, the team opted to approach the combined problems of probabilistic regression and fault classifications from multiple angles. Overall, the team was successful, able to develop accurate and robust models that achieved the goal set out in the challenge. By properly training and evaluating the models' performance, the team was able to detect and avoid common issues with neural networks, such as overfitting.

While more complex methods such as ensemble models, neural networks comprised of multiple smaller models, have the potential for higher overall performance, the team found that the comparatively simpler models chosen for this project provided good performance without the additional training and logistical overhead of more complex architectures.

Nomenclature

oat	Outside Air Temperature
pa	Power Available
mgt	Mean Gas Temperature
np	Net Power
ng	Compressor Speed
trq_{mar}	Torque Margin
trq_{meas}	Measured Torque Output
ias	Indicated Air Speed
trq_{tgt}	Design Torque
μ_{mar}	Mean of Torque Margin
μ_{tgt}	Mean of Torque Target
σ_{mar}	Standard Deviation of Torque Margin
σ_{tgt}	Standard Deviation of Torque Margin
MLP	Multilayer Perceptron
LR	Learning Rate
t	Any Torque Margin Value
T_n	Nominal Normally Distributed Random Variable
$T_{\mathfrak{f}}$	Faulty Normally Distributed Random Variable
P_n	Nominal Probability Density Functions
$P_{\mathfrak{f}}$	Faulty Nominal Probability Density Functions
CDF	Cumulative Distribution Function
C	Confidence
x_n	Nominal Label
$x_{\mathfrak{f}}$	Faulty Label

REFERENCES

Adams, R. P. (2018, September). *Linear regresssion via maximization of the likelihood.* Retrieved from https://www.cs.princeton.edu/courses/archive/fall18/cos324/files/mle-regression.pdf

FAA. (2019). *Helicopter flying handbook*. U.S. Department of Defense.

Keras. (2024, Jun). *Tf.keras.layers.dense tensorflow v2.16.1.* Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Kingma, D. P., & Ba, J. (2014, December). Adam: A method for stochastic optimization. *arxiv*. Retrieved from https://arxiv.org/abs/1412.6980

PHMSociety. (2024, Jun). *Phm north america 2024 conference data challenge.* Retrieved from https://data.phmsociety.org/phm2024-conference-data-challenge/

Rouaud, M. (2013). *Probability, statistics and estimation*. Lulu Press, Inc.

Tensorflow probability. (n.d.). Retrieved from https://www.tensorflow.org/probability

Terven, J., Cordova-Esparza, D. M., Ramirez-Pedraza, A., Chavez-Urbiola, E. A., & Romero-Gonzalez, J. A. (2024). *Loss functions and metrics in deep learning.* Retrieved from https://arxiv.org/abs/2307.02694