Reviewing the Efficacy Predicting Cancellation Days in Regional School Unit 1 Using a Multilayer Perceptron Deep Learning Model

Lochlan Aldrich and Declan Wright
Advanced Placement Statistics
May 29, 2025

1 Introduction on Model Training

We introduce a sophisticated multilayer perceptron (MLP) neural network to predict snow days using data from historical weather datasets. Our model underwent extensive training across two computational environments: 96 hours on a local GPU, followed by an additional 6 hours on Google Cloud's v2-8 tensor processing unit. We acknowledge Google Cloud for providing access to their computational infrastructure.

This extensive training process required over 159 quadrillion floating-point operations (FLOPs) during computation. While these numbers represent a seemingly implausible scale, modern compute hardware makes such processing achievable within reasonable time frames. The combination of local GPU training runs and cloud-based TPU acceleration for testing and refinement enabled us to complete this computationally intensive training in approximately 100 total hours.

2 Neural Network Architecture

To understand how an MLP can predict snow days, an understanding of their internal structure is required. These models loosely resemble the human brain, with neurons internally representing different functions. In our model, we use a single hidden layer with 50 neurons. Larger models may use more layers or more neurons, but the concept is the same.

In an MLP, each neuron is defined as the sum of each preceding weight plus a bias. Both weights and biases are learned through training using a backpropagation algorithm, essentially a function that calculates derivatives using the chain rule with respect to every parameter in the model. An optimizer function (in our model we use Adam) attempts to minimize a loss function in a model. Each time the model is exposed to a full pass of the training dataset, backpropagation calculates the gradients and Adam minimizes loss, essentially teaching the model to recognize patterns in the data.

3 Model Size Optimization

Determining the optimal number of neurons is difficult. Too few hidden neurons may result in a model that is not able to capture complex patterns from the dataset. On the flipside, extremely large models often overfit to their training data, resulting in a model that performs well on information it has already seen, but cannot make accurate predictions in the real world.

To solve this issue, we create a simple random sample of 20% of the examples from the original dataset. This data is reserved as an unseen validation set, safeguarding against overfitting. To determine the optimal number of parameters to use internally, we run a series of tests with $n = 50$ training runs in each sample to estimate the best potential performance.

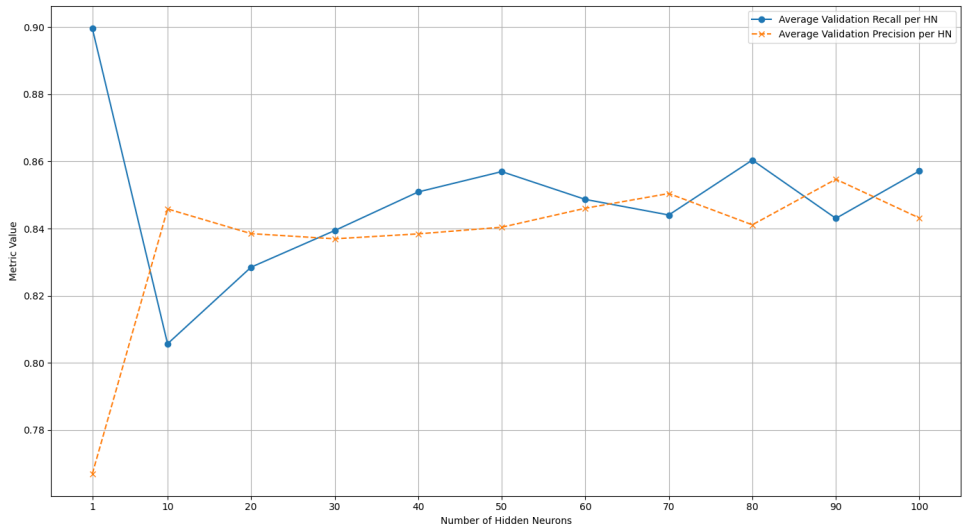

Figure 1: Mean Model Performance (validation) over Number of Hidden Neurons

From this figure, we can observe that using too many or too few hidden neurons results in unstable training, as seen in examples of over 70 hidden neurons, or neuron counts below 10. Within the stable range, 50 appears to offer the best combination of stable precision, and the highest recall. For this reason, 50 is chosen as the optimal model size for our network.

4 Training Process and Stability

In our model, we train for a total of 3,500 passes over the dataset (Epochs). The result from this training run is shown below.

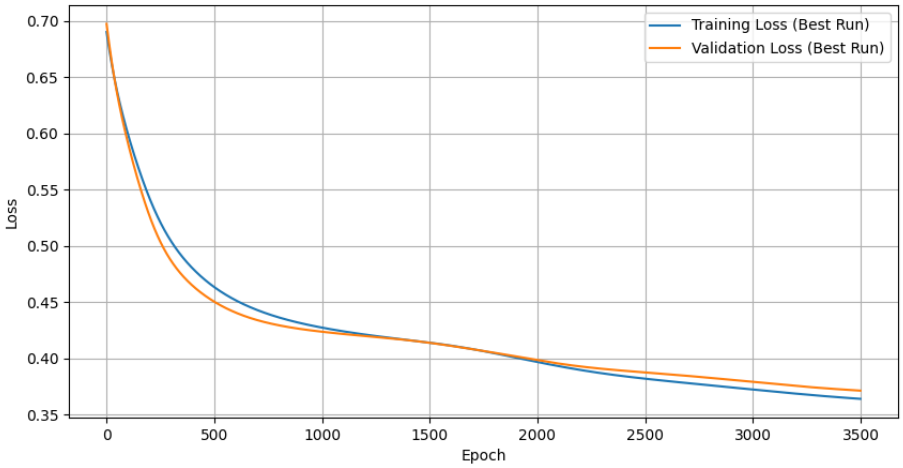

Figure 2: Loss Functions During Training

From this graph, we observe an incredibly stable training run. The smoothness of both curves shows strong learning ability. Additionally, the unseen validation set remains remarkably similar to the training data itself. This provides convincing evidence that the model is not overfitting to its training set.

5 Activation Function Evolution

During testing, we experiment with different inputs and model shapes. We experiment with several different activation functions, transformations that take the calculated weights and biases from each neuron and introduce nonlinearity.

5.1 Initial Implementation with ReLU

Initially, we used ReLU as our activation function. ReLU, which stands for Rectified Linear Unit, is simple and easy to compute, defined as a piecewise function where negative numbers equal zero, and positive numbers remain unchanged. Our first model is built using ReLU, and it achieves a respectable 73% precision on the validation set using just three input features: predicted snowfall, previous day's snowfall, and temperature.

5.2 Experimentation with GELU

Unfortunately, ReLU presents some issues and is outdated in modern models. For this reason, we develop a model using GELU, which is differentiable at zero (unlike ReLU) and scales negative values. Using this activation function, we achieve 79% validation precision.

5.3 Final Implementation with SwiGLU

In our final test, we make two changes. First, the activation function is again upgraded to SwiGLU, a state-of-the-art activation function that retains the benefits of GELU, but adds an additional trainable gating parameter, allowing the degree to which the activation function effects the flow of the data to be learned during training.

$$\text{SwiGLU}(x) = \text{Linear}_1(x) \times (\text{Linear}_2(x) \times \sigma(\text{Linear}_2(x)))$$

where σ is the sigmoid function.

6 Feature Engineering Experiments

In addition to the switch to SwiGLU as an activation function, we increase the number of input features from 3 to 222 using temporal features and additional metrics such as wind and visibility to improve our predictions. It works: this model achieves a stunning 91% validation precision. By performing a feature analysis using permutation, we observe interesting emergent behavior.

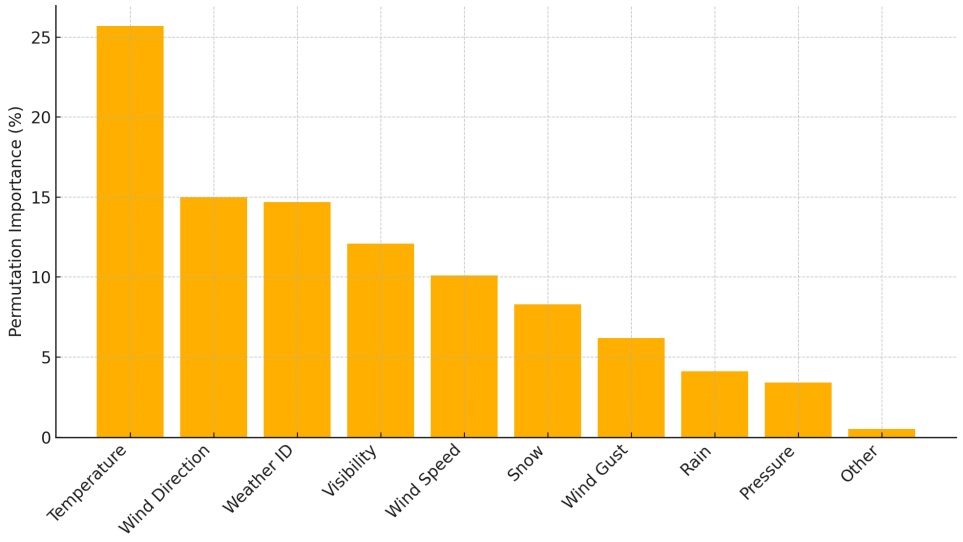

Figure 3: Feature Importance (permutation contributions)

In this large model, we see that the majority of the model's predictive power does not come from snowfall, as might be expected. Instead, Wind, temperature, and visibility contribute to a majority of the model's predictive power. This highlights an interesting observation about deep learning models: while there is robust evidence to prove the efficacy of such models, we cannot readily explain how they work internally. This feature analysis gives a window into the model, but remains unintuitive to standard meteorological practices. This is one of the core strengths of machine learning overall, deeper insights that aren't intuitive to standard forms of pattern recognition. We hypothesize that these seemingly unimportant features, like wind or visibility, are strong markers of storm events in our training data, which would explain their importance to the model.

7 Final Model Selection and Optimization

While training a much larger model was an interesting case study in the predictive power of MLPs, we ultimately do not use this model in our final evaluation. Its size and complex set of input data features make practical implementation difficult. Instead, we use the SwiGLU activation to train a new model with the original three input features. Additionally, we normalize the input dataset using z-scores, which increases the stability of the model during training.

This new model is detailed above and evaluated in the rest of this report. We train several initial runs to de-risk the time investment on our final training model, before

training locally for 600 thousand loops. The final set of weights from training achieve 89% precision on the validation set. While around 2% less accurate overall than our best model, this final network uses only 251 parameters in total, which is about 96.7% more efficient than the larger MLP (7681 parameters) and simplifies deployment. Practicality and explainability of the model are, in this case, valued slightly over pure performance.

8 Calibration of the Final Model

Following the selection of our final model (89.2% precision), we perform a calibration step to fix systematic imbalances in the model's output predictions. Using a bucketing analysis, we observe that predictions in the 0.4–0.8 range are systematically wrong. The model has learned an observable association for these ranges, but in a way that is flipped from the standard prediction logic. Because the number of samples in this range is relatively small, the model's overall metrics are not significantly affected by this change. That being said, this region is probably the most important when it comes to real world applications, where borderline scenarios are the most difficult to predict.

To rectify this issue, we further calibrate this range, seeing an astonishing 68.6% improvement in "close call" predictions, and a respectable 5.64% global improvement in accuracy.

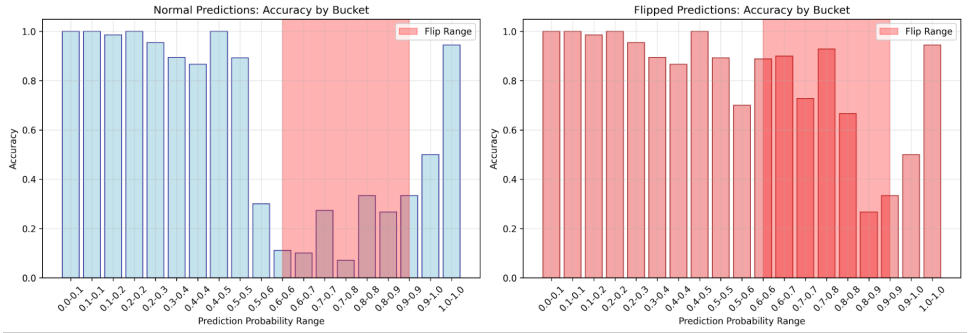

Figure 4: Calibration for Middle-region Predictions

Finally, we observe an interesting trend for extremely cold values when making predictions. For temperatures under 15 degrees Fahrenheit, the model produces probabilities close to 100%, even when the predicted snowfall is extremely low (i.e., 0.1 inches). This unusual behavior is likely an artifact of training. With only 12 samples (2.2% of dataset) falling below this threshold, there is likely not enough data for the model to learn how these temperature outliers affect predictions. To fix this issue, we use a data preprocessing step that clips temperatures below 10 degrees, effectively mitigating the issue and increasing the global accuracy by 0.1%. This is a small change but improves how the model processes outliers and extreme cases when making predictions.

9 Summary Statistics

Below, we present two tables summarizing the distributions of both our input training features and the results from our tests.

Table 1: Summary statistics for our Training dataset

Feature	n	Mean	Std	Min	25%	50%	75%	Max
Predicted Snow (in)	537	1.50	1.61	0	0.5	0.9	2.0	11.6
Previous Snow (in)	537	0.08	0.59	0	0	0	0	7.9
Temperature (°F)	537	25.2	8.32	-4	19.6	26.6	31.2	43.8

Table 2: Summary statistics for Predictions on our dataset

n	Mean	Std	Min	25%	50%	75%	Max
537	0.2565	0.2805	0.0000	0.0265	0.1381	0.4177	1.0000

10 Chi-squared Hypothesis Test

Table 3: Observed and expected counts for snow days

Actual	Predicted	Observed	Expected
No Cancellation	No Cancellation	428	403.84
No Cancellation	Cancellation	66	90.15
Cancellation	No Cancellation	11	35.15
Cancellation	Cancellation	32	7.84
Total		**537**	**536.98**

To determine if the neural network is actually predicting cancellation days correctly, we conducted a Chi-Square test for homogeneity ($\alpha = 0.05$) over a simple random sample of the data set with the following hypotheses:

H_0 : There is no association between the model's predictions and the actual occurrence of snow days.

H_a : There is an association between the model's predictions and the actual occurrence of snow days.

The data meets the requirements for a chi-squared hypothesis test, as 537 days is less than 10% of the total number of school days. Additionally, each of the expected counts are over 5, so the large counts condition is met. Our validation set is constructed from a random sample, so our data is representative.

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

Calculation:

$$\frac{(428-403.7)^2}{403.7}+\frac{(66-90.3)^2}{90.3}+\frac{(11-35.3)^2}{35.3}+\frac{(32-7.7)^2}{7.7}=94.80$$

Using the equation for the test statistic and the data above, we calculated a χ^2 value of 94.79. With a p-value of essentially zero, we conclude that there is a statistically significant association between the model's predictions and the actual snow day occurrences. We have very strong evidence that the model is accurately predicting snow days.

11 Z Confidence Interval

To examine the true precision of the deep learning model, we produced a 95% confidence interval using one-proportion z-interval on the validation set ($n = 197$) in order to estimate the true precision value of the model. The validation set is randomly selected via a simple random sample, and the positive results used for precision are less than 10% of all snow days. Each of the true values of positive results has a count of at least ten, so the large counts condition is met.

Table 4: Truth value of positive results

Result	Count
True Positive (Cancellation)	87
False Positive (No Cancellation)	11
Total	**98**

where $\hat{p}=\frac{87}{98}$, and $n = 98$

$$SE=\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}=0.031293$$

where $\alpha = 0.05$

$$z^*=1.959964$$

$$ME=z^*\cdot SE=0.061333$$

$$\hat{p}\pm ME=(0.831141,0.953806)$$

Using the data and equations above, we generated a confidence interval of (0.831, 0.954). We are 95% confident that the true precision of the model is contained in the interval (0.831, 0.954).

12 Conclusion

With a stunning overall calibrated accuracy of 94.8%, and extremely strong statistical evidence that there is an association between our model's predictions and actual outcomes (p-value ≈ 0.000), we are confident in our research and its generalization to real-world production environments. For this reason, we introduce snowcall.org, a custom website using our model's advanced predictive power.

Appendix A: Python Implementation of a Simple Model

A full implementation can be found on GitHub.

Listing 1: Python Implementation

```python
from json import loads
from typing import Any
import numpy as np

def c_to_f(c: float) -> float:
    return c * 9/5 + 32

def mm_to_in(mm: float) -> float:
    return mm / 25.4

with open("model-a.json", "r") as file:
    model_a_parameters: dict[Any, Any] = loads(file.read())

def sigmoid(z: Any):
    return 1 / (1 + np.exp(-z))

def model_a(snowfall_mm: float, prev_snow_mm: float, temp_c: float) ->
   float:
    snowfall_in = mm_to_in(snowfall_mm)
    prev_snow_in = mm_to_in(prev_snow_mm)
    temp_f = c_to_f(temp_c)

    if snowfall_in < 0.2 and prev_snow_in < 0.2:
        return 0.0

    temp_f = max(temp_f, 10)

    initial_vector = np.array((snowfall_in, prev_snow_in, temp_f, 1))
    means_vector = np.array((*model_a_parameters["means"], 0))
    stdevs_vector = np.array((*model_a_parameters["stdevs"], 1))

    adjusted_vector = (initial_vector - means_vector) / stdevs_vector

    fc1_vector = np.array(model_a_parameters["fc1_weights"])
    z1_vector = np.dot(fc1_vector, adjusted_vector)

    swiglu_out_vector = np.append(z1_vector[:25] * sigmoid(z1_vector
       [25:]), 1.0)

    fc2_vector = np.array(model_a_parameters["fc2_weights"])
    z2_scalar = np.dot(fc2_vector, swiglu_out_vector)

    prediction = sigmoid(z2_scalar)

    if 0.51 <= prediction <= 0.85:
        prediction = 1 - prediction

    return min(prediction, 0.99)
```