# Applications for AI tools and machine learning

## a case study: Kaleva

Tero Halkoaho
Loihde

To try out what this new AI can be used for, our team of three – the author of this article, Tapani Mattila, and Jussi Puhakka – were assigned a new quite challenging project: to help Kaleva save energy in their printing processes. The main goals were:

- To help Kaleva save energy, cut costs, and decrease carbon emissions.
- To probe the potentials of neural networks in project related to energy consumption management.
- To learn to use ChatGPT as a tool in building neural networks.
- To increase our knowledge about the subject and processes related to it.

We were constrained in several significant ways, the biggest one being that no new hardware was to be added. This meant that our main focus was to find ways to use existing hardware to our advantage.

We started by asking ChatGPT what approaches it recommended, added in few of our own, and then proceeded to interview the representatives of Kaleva about our thoughts. Some of which we had thought to be most promising ones, like rescheduling the print times to the times of cheaper electricity, were ruled as not feasible due to strict time tables for the printing. Any saving would be negligible, and would only introduce unwanted complexity into the process.

What was interesting, however, was that Kaleva had an extensive record of their past energy consumption statistics, which meant that we had plenty of energy consumption data to start with. This gave us an idea to use machine learning to detect anomalies from their energy consumption data, which could perhaps be used to flag faulty machinery before things get dangerous.

### *Preparation*

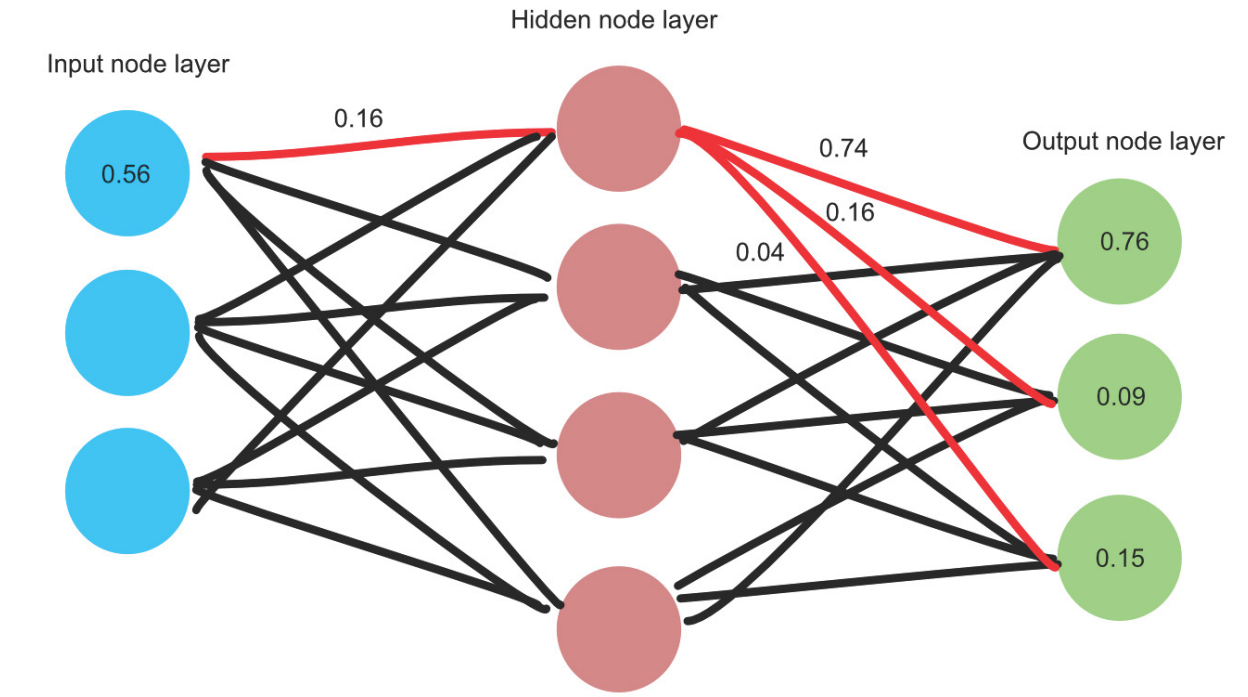
As none of us had any prior experience in building neural networks, and I was the most enthusiastic about this exact approach, I decided to take the lead. This meant grabbing a couple of books about the subject and spending more than a few hours enjoying the short Finnish summer in a hammock reading about neural networks.

The two books I chose were Tariq Nashiid's “Make Your Own Neural Network”, which worked as a decent start to my journey to machine learning. After finishing that appetizer, I chose to go for the main dish and decided to read Nithin Buduma's “Fundamentals of Deep Learning”. Compared to the relative light reading of the previous book, this book was like being hit repeatedly in the head with a math-mallet. Mathematics teachers everywhere: when students ask you where does one need all these complex mathematical concepts, you can answer “neural networks”, and you are probably correct no matter what the exact subject matter is.

### *So what, exactly, is a neural network?*

In its most basic form, a neural network consists of three rows of what we call 'nodes'. The nodes themselves function only as a crossroads the paths lead through, and do not do anything themselves. The data fed into the network is normalized to contain only values from almost-0 to almost-1. The data is then sent from the nodes in the input layer (each of which can be, for example, a luminosity value of a pixel), to every node in the next hidden layer, from which it is forwarded to every node in the output layer, where all the incoming signals are combined to form a value.

The intelligence built in the system comes from the weight of each of these connections. So a trained network sends the data through the same route as an untrained network, but the weight of the signal changes on the way. In the image below, a signal of strength 0.56 is received by the input node from the data, is forwarded to the hidden node with he weight of 0.16, and forwarded to the output nodes weighted differently. Of course it is not this simple, with a lot of complex math going in to the distribution of the signal in the network, but you probably get the idea.




When all the signals coming from all of the hidden nodes are summed up in the output layer, one of the output nodes has a higher total score than others, and is therefore the answer – or part of the answer, depending what you are doing. In this case the topmost node of the output layer is the answer. What the topmost node refers to depends on what you are doing. It might be a letter, a number, “yes”, a word, or something completely different.

So, how does it know how the signal should be weighted? This is where training the network comes to play. On each run we punish the path that came to the wrong conclusion by lowering the weights of the connections that took part in that result, each route decreased depending on the amount of weight they had in the answer, and we increase the weights of the paths that came to the correct conclusion. Once the network has been trained, it becomes a magical black box full of indecipherable numbers which comes to the right conclusion for reasons a human has no way of understanding.

So now, armed with a dizzying amount of maths under my belt, I got to work. But first we needed to look what kind of data we had been given access to. There was a lot of it: tens of millions of rows worth of data.

## *The data*

Getting the data was as simple as asking ChatGPT to create me a Python script to fetch it from the remote database and save it locally. Gathering interesting data from the data tables was as simple as asking ChatGPT to create me a Python script that prints out all the tables along with their columns, and "some interesting statistics" about the data.

After browsing the data a bit it turned out that all that log data was in a single table called DataLog2, but after some highly sophisticated mapping of the data tables, the logic opened itself up to us. The data had a column called "value", which contained the actual data, and the two other columns "SourceID" and "QuantityID" specified where it came from and what it represented. Turned out that I could simply ask ChatGPT what those names in the QuantityID -column were, and it would explain to me in detail what they meant and which would be most suited to observing anomalies.

So for testing, I chose SourceID 33 because ChatGPT-generated scripts gave me this statistical information about that specific SourceID, which made it look promising for my purposes:

SourceID: 33, Source Name: PK2.01A_ERWEKO
- Count of Values: 1663565
- Average Value: 1423314.58
- Minimum Value: -122.9766616821289
- Maximum Value: 8516355.0
- Standard Deviation: 2718675.03
- Median Value: 103.95

The QuantityID ChatGPT recommended me to focus on was 107, which pointed to the "current energy consumption" row in the Quantity -table.

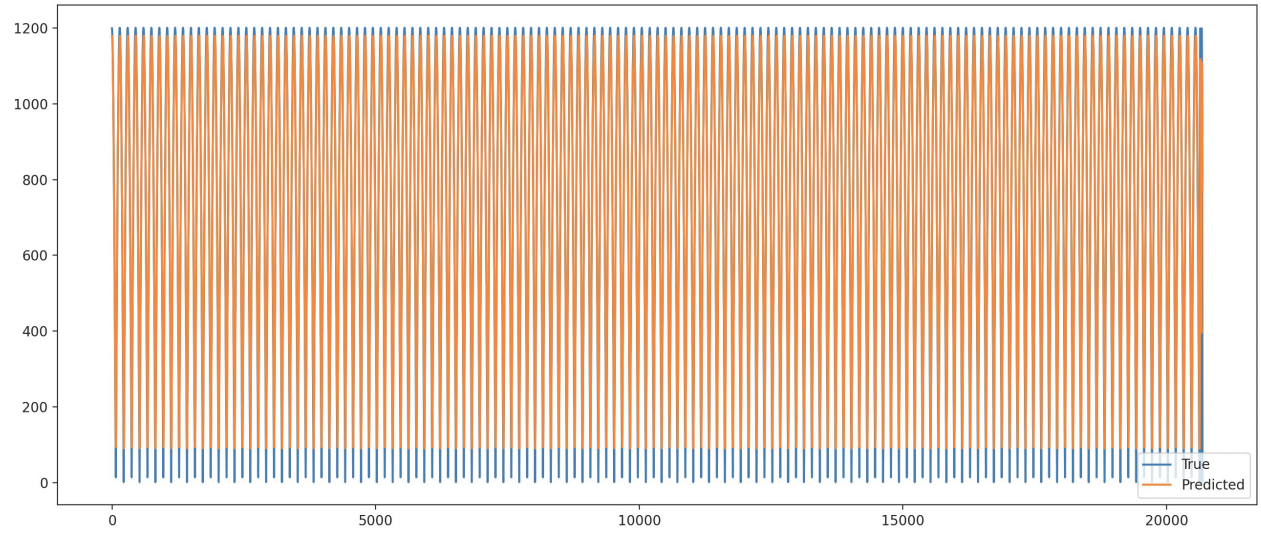
Next we had to clean up the data we got. Cleaning up the data is one of the most – if not *the* most – important steps in constructing a neural network. Unless the data is properly cleaned, the model that resulted would not amount to anything.

The most obvious fault was that there were some rows that didn't contain any value. Those would break the neural network learning process, so they had to be either removed or replaced with something else. I decided to go through the data and replace the missing values with 0.1. Why not with zero? Because zeroes break the training algorithm because in the depths of the logic there will be a divide by zero.

So I got to work, selecting Python as the preferred language, and Python's TensorFlow module as the core of my neural network. Based on what I had read, I decided that for detecting anomalies, the neural network type called 'Recurrent Neural Network' was the one we should go for. This specialized neural network type excels in finding patterns in data when there is a time dimension involved. So, in our case, it would learn that a certain type of energy consumption seemed to follow some other type of energy consumption. My basic premise was that we could use that to see what a "normal" energy consumption looked like, and therefore to detect what *wasn't* normal.

ChatGPT was eager to help at any turn, and soon I had the first version of my model ready to be trained. To make sure it would certainly find a pattern, I decided to use training data that most certainly would contain a pattern to recognize: a sine wave.

So, with that set of data, the model came alive:



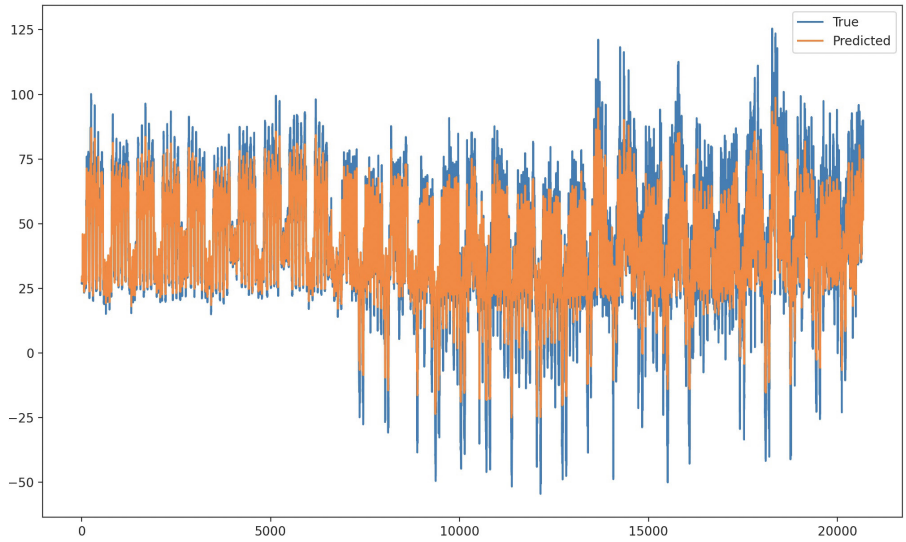

Now, for many that doesn't say much. But to me that looked like a field full of flowers at a summer's eve. What that was was *a prediction*.

I had now taught to my neural network a pattern of behavior of the data. It would now estimate – to some degree of accuracy – the next value in a series of numbers, which in this case came from the absolute values of a sine wave. If that doesn't sound much, you'll have to take into account that: *it didn't know it was looking at a sine wave.* If you look at the produced graph, the blue line represents the actual data, and the orange data superimposed on top of it represents the prediction that my model made based on other data it had been taught with.

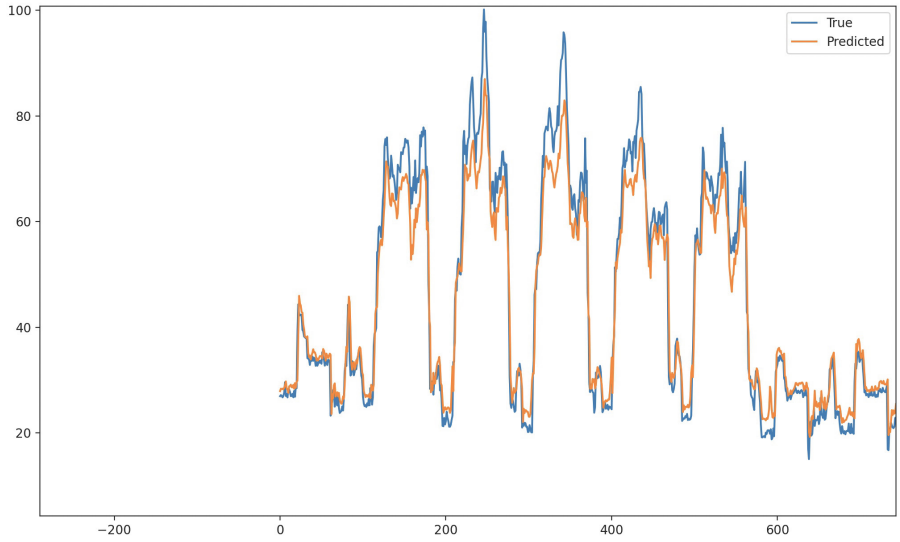
Basically, as it goes through the test data I'm giving it, it repeatedly answers this question: “Based on your experience with this kind of data, knowing that the last 30 numbers are as they are, what is the next number?” The model makes a prediction, and the prediction is then compared with the actual result. In a perfect situation there would not be any blue visible on the graph as he orange would hide it completely.

So, now I had a neural network that was actually trainable to find patterns in data. All I had to do was add in the real data and see what it learned from it. I took the whole dataset, replaced all zero values, as well as missing values, with 0.1, used 80% from the beginning to train the model and then used the last 20% to test whether the model could predict things.

This beautiful work of art was the result. Prediction vs. the last 20% of the data:




As you can see, it has actually figured out how the energy consumption trends go. If we zoom in on the beginning part of that graph, you can see better how it tries to estimate the energy consumption profile:




There is clearly logic there.

## *Improving what we got*

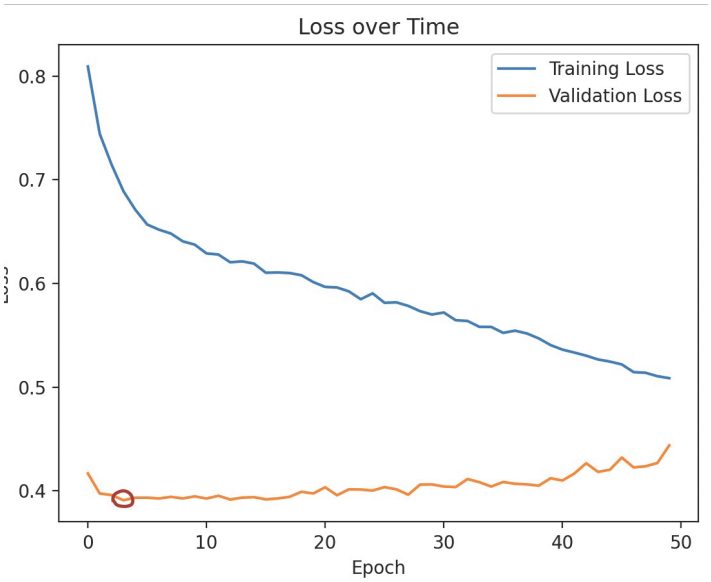
Now it was just a case of finding the correct parameters for the neural network to decrease the error margin between the prediction and the real data so that anomalies would be found while avoiding flagging normal events as anomalous. There are quite many variables in figuring out the optimal neural network architecture for an individual problem. For example:

- number of node layers
- number of nodes in each layer
- learning rate
- sequence length
- number of epochs (learning runs)
- dropout rate

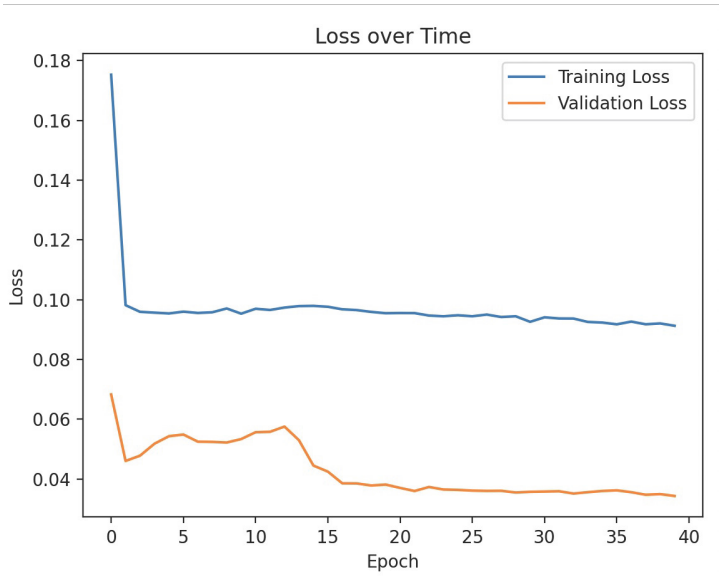
There are also possible variations for some of those values (like the learning rate) that might need to be decreased or increased based on the number of the epoch that is currently running.

There is no silver bullet, only a few simple guidelines, such as increasing the number of node layers and nodes in each layer tends to increase the ability to find more complex patterns form the data – if such exist. This doesn't, however, mean that adding more nodes does anything more than increase your training times. If you know there is a cycle, such as a weekly usage cycle, setting the sequence length to the length of the cycle helps in finding logic in that cycle and perhaps improve predictions. An epoch is a single learning run through all the training data, from a randomized starting point, and the end result is the combination of all the epochs during a training session. There is also an optimal number of epochs for each set of values, and you can try to see what that is by graphing out the validation loss rate.

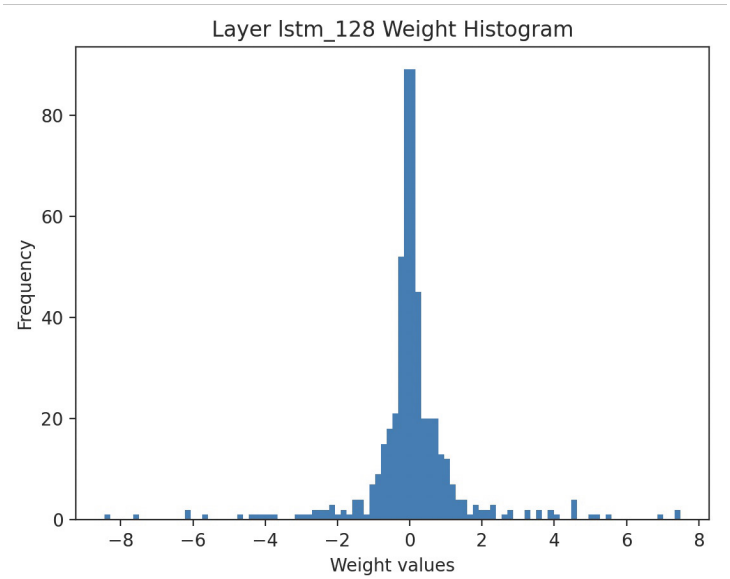
In this completely unsuitable model, the optimal number of epochs is around 3 (circled), after which the validation loss starts climbing rapidly. Decreasing **training loss** (how well it matches the data it is trained with) while the **validation loss** (how well the trained model predicts values) is increasing, means the model has stopped finding logic from the data and is just overfitting itself to the data it has been given – it stopped trying to understanding the data and instead is just memorizing it. And in doing so it is getting worse every time we train it:




The following is a better looking graph, as the training is able to squeeze a little bit more logic from each run, improving the prediction ever so slightly through time:

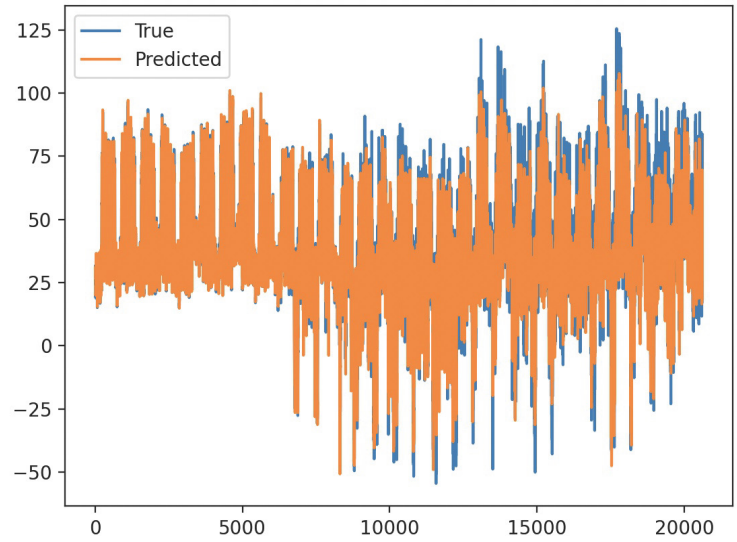



The only way to find out the optimal values for the task you are trying to teach the model to do, is to try out different combinations of values, examine the performance through graphs like the one below, and try to find out ways to improve the situation. I found ChatGPT to be *immensely* useful not only in creating the scripts that produce these graphs, but as a partner to bounce ideas with about what the significance of the shapes of these graphs were.

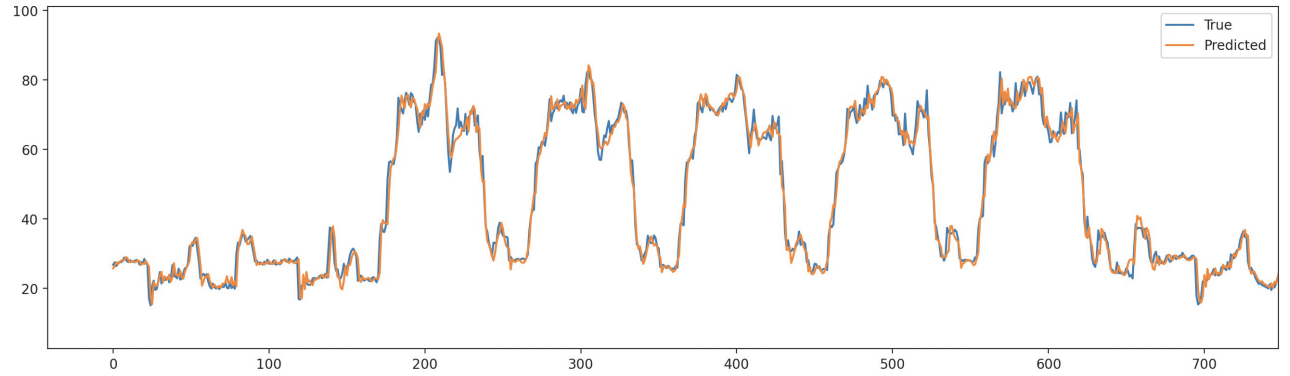



Increasing the amount of data, epochs, nodes, and sequence length increases the time each training run lasts. In most settings we were dealing with around 5-15 minutes per epoch, making most of the runs last several hours.

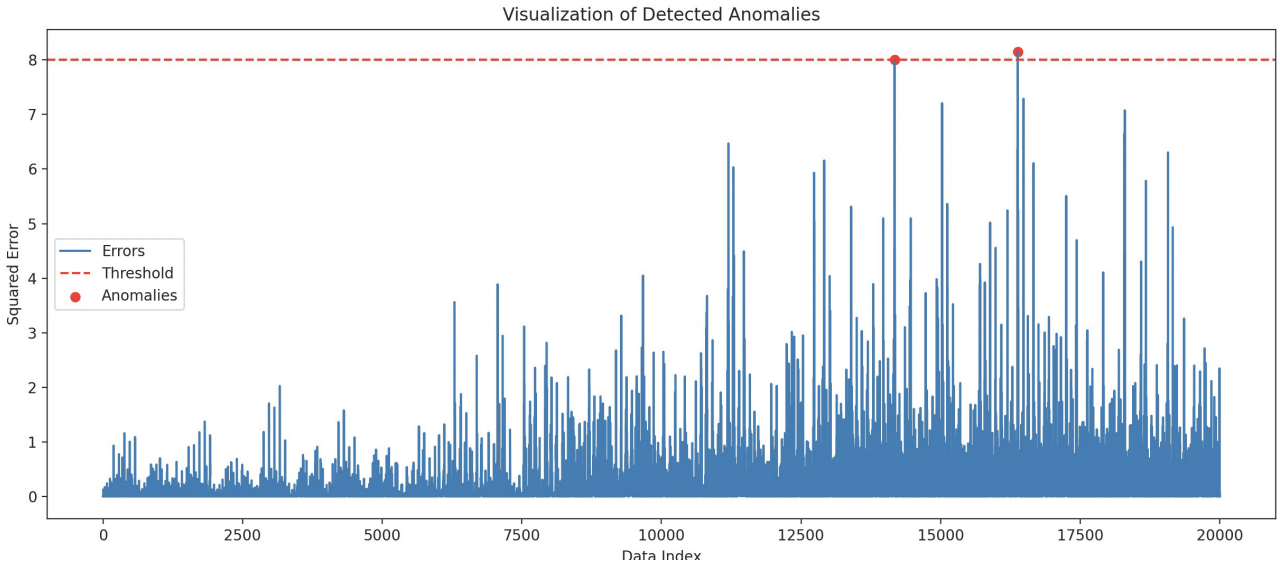
Then, after dozens of hours spent on finding the optimal values, this magnificent graph appeared almost without a warning:




It had much less visible blue than any of the previous runs. Looking at the part right at the beginning, you can see how perfectly the model matches the actual values:

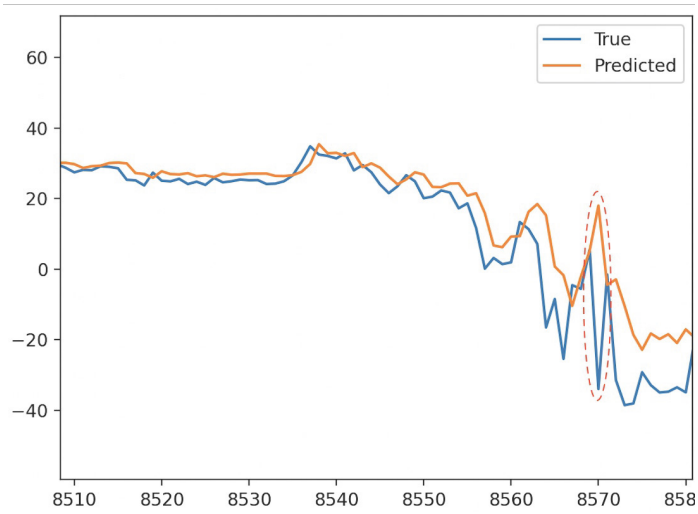



Now, what are the things we're now actually looking for? They are what are referred to as “anomalies”, I.e. values that deviate from the prediction too much. For example, in this instance we determined that any squared-error deviation of more than 8 was to be flagged as an anomaly. The listed deviations from the actual data was drawn in this graph:




This is far from a typical code where you might say “if the energy consumption is more than 15,000, send an alert”. In the data itself, an anomaly might not look anything special. For example

this is an anomaly, as reported by one version of the model:




Another detected an anomaly in the data at row 91724:

| | | |
|---|---|---|
| 91714 | 2023-04-30 22:00:00 | 27.3472194672 |
| 91715 | 2023-04-30 22:15:00 | 25.0626125336 |
| 91716 | 2023-04-30 22:30:00 | 24.8784713745 |
| 91717 | 2023-04-30 22:45:00 | 25.5963954926 |
| 91718 | 2023-04-30 23:00:00 | 24.118844986 |
| 91719 | 2023-04-30 23:15:00 | 24.7959728241 |
| 91720 | 2023-04-30 23:30:00 | 23.8801269531 |
| 91721 | 2023-04-30 23:45:00 | 25.8388214111 |
| 91722 | 2023-05-01 00:00:00 | 24.6142482758 |
| 91723 | 2023-05-01 00:15:00 | 24.9113349915 |
| **91724** | 2023-05-01 00:30:00 | 25.3974075317 |
| 91725 | 2023-05-01 00:45:00 | 25.2059764862 |
| 91726 | 2023-05-01 01:00:00 | 25.2459392548 |
| 91727 | 2023-05-01 01:15:00 | 24.1356258392 |
| 91728 | 2023-05-01 01:30:00 | 24.2413768768 |
| 91729 | 2023-05-01 01:45:00 | 24.9074993134 |
| 91730 | 2023-05-01 02:00:00 | 26.5651111603 |
| 91731 | 2023-05-01 02:15:00 | 30.3110294342 |
| 91732 | 2023-05-01 02:30:00 | 34.8066062927 |

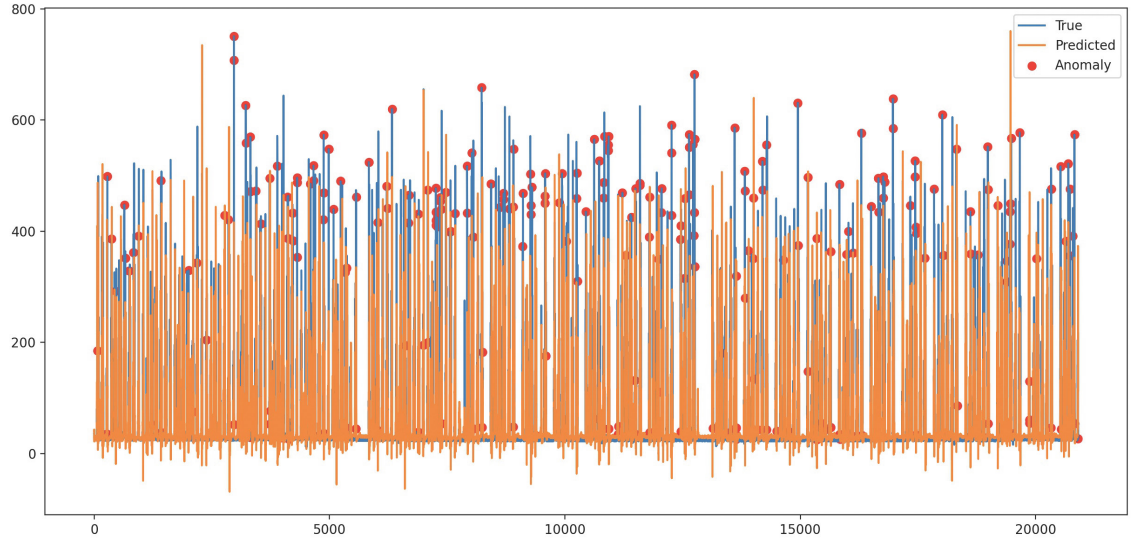
However, when I inserted anomalous random value into the test data, or moved some data around, the model flagged it perfectly as anomalous. We had actually built a neural network that could be used to detect anomalies from energy consumption data!

## *The Anti-Climax – a terrible disappointment*

As we had now grasped the concept of neural networks, and had successfully built one, now it was time to discuss our findings with Kaleva. They gave us more information about what energy consumption sources were important for them to monitor, and we went on to try our model against that data. This time things did not go well, however. The data contained the combined energy consumption of two printing presses plus some other associated devices. One or two of those printing pipelines could be running at any given moment, their running time varied, as well as the starting time. Each run lasted typically around an hour, and we only got combined energy consumption data once every fifteen minutes. This meant that for every run, we only got a couple of datapoints – four at most.

No matter how we tried, we just couldn't get our model to learn any patterns. The data was too random, variables too great, and any signal that would be there would simply disappear among the

noise. Attempts to clean up the data in ways that might reveal signals were a failure. It didn't get any better from this disaster:




If the frequency of the data recordings would be increased significantly to something like once a minute, a detectable signal might appear. If there were only one machine on this single log value, a signal might appear. But both combined – it didn't seem doable.

We could perhaps ask Kaleva if they could increase the rate of logging, but it would take so much time for enough data to be gathered to be useful in training a neural network, that the time window for our project would be well over by the time we could continue with more accurate data.

Despite a frustrating turn of events, the project was not a failure. The main goal was to gain experience in these kinds of approaches and technologies, and in that it was success. I personally went from having no experience in machine learning, to understanding the uses, limitations, and what to take into account when doing a project like this.

The final achievements of this project can be summarized in this way:

- Discovering that, indeed, ChatGPT is a great help in analyzing complex concepts, like performance graphs of neural network training.
- Gaining significant insight into constructing neural networks.
- Valuable experience in the application of neural networks in interpreting time series data, with the challenges building it, interpreting of what we have achieved, and debugging of how we could improve our algorithms to achieve a better result.
- The observation that neural networks can indeed be used to observe anomalies in the energy consumption data, but the data needs to be sufficiently detailed, and from a long enough period of time. This is a useful lesson for companies planning on utilizing this kind of machine learning algorithms: the saved observational data should be as detailed as possible, and *now* is the best time to start gathering more data.