Neural Network Debugging with TensorBoard

 Romeo Kienzler

Welcome to the TensorBoard into - a visual way to debug your neural networks

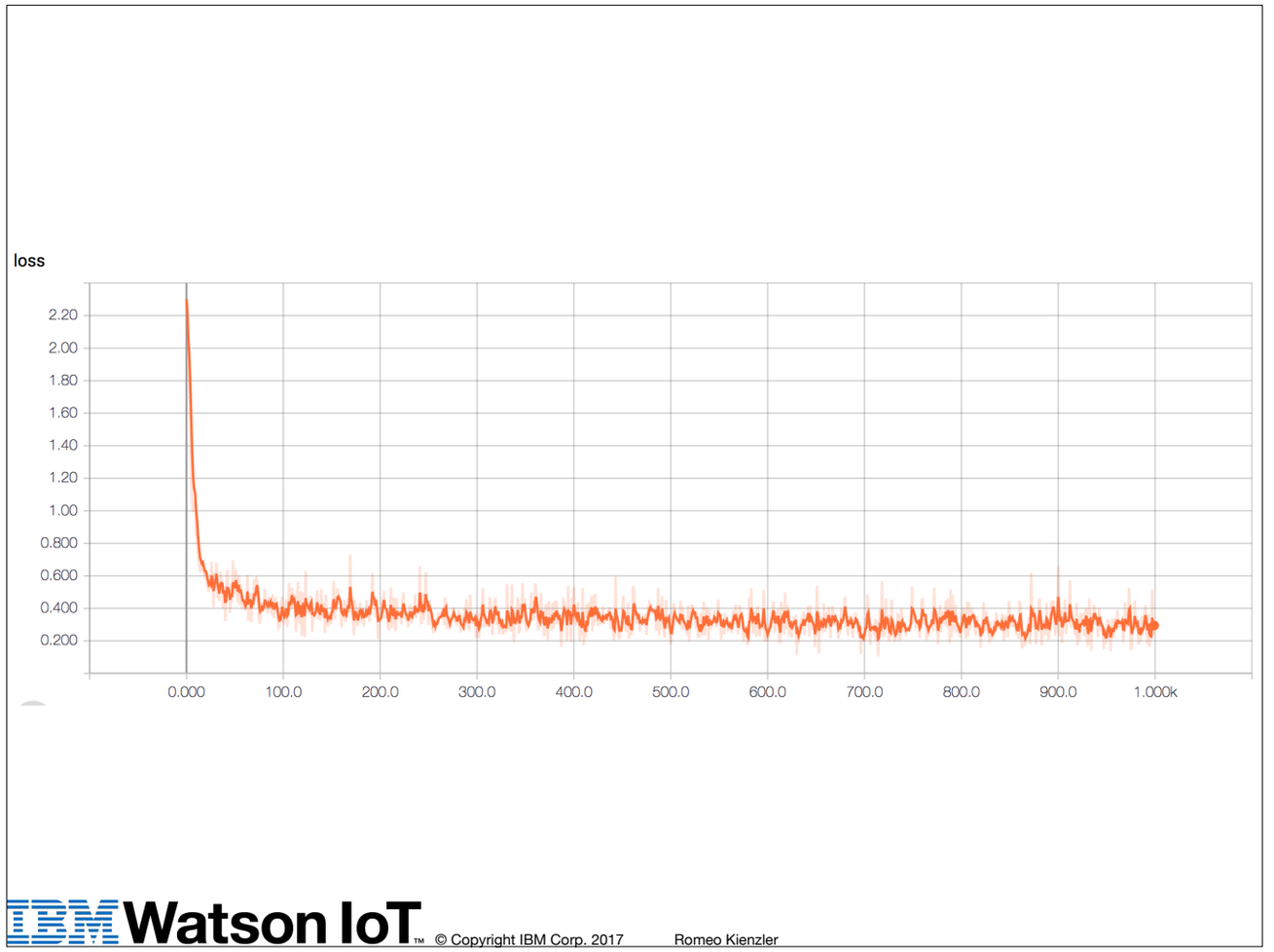
Neural Net

Neural Net

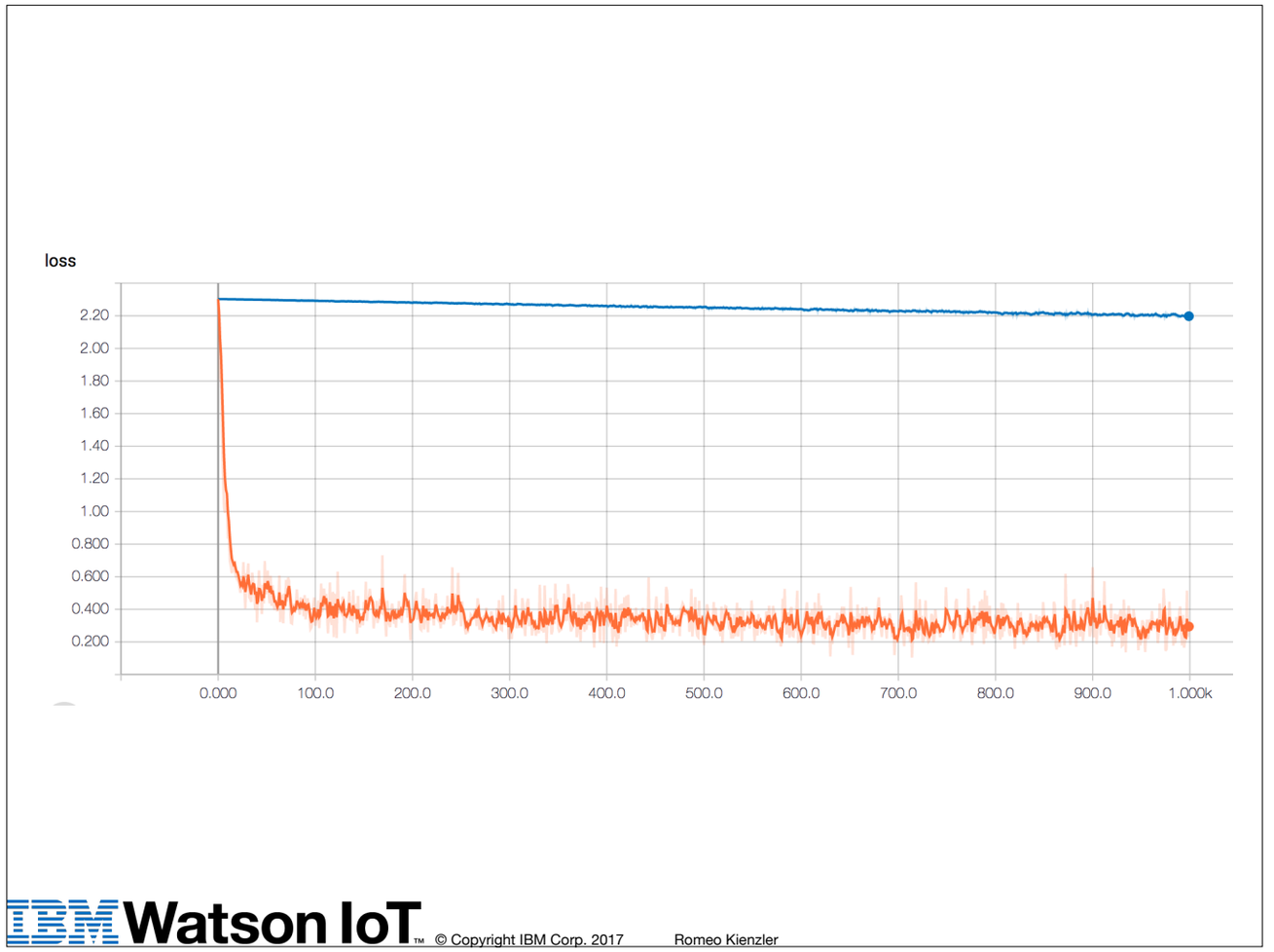
a3_m1__s2_v2_tbintro_v1

 Romeo Kienzler

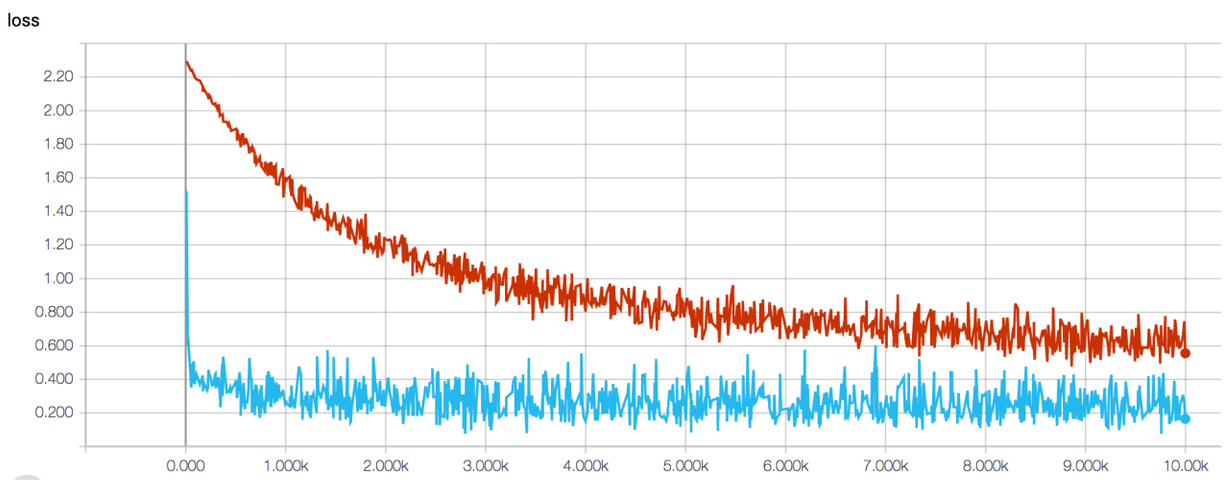
So a neural network often is some sort of …a black box and it’s very hard to see what’s going on during training. So usually people tend to print out all sorts of measures during the gradient descent loop in order to debug and make sense of the training phase.



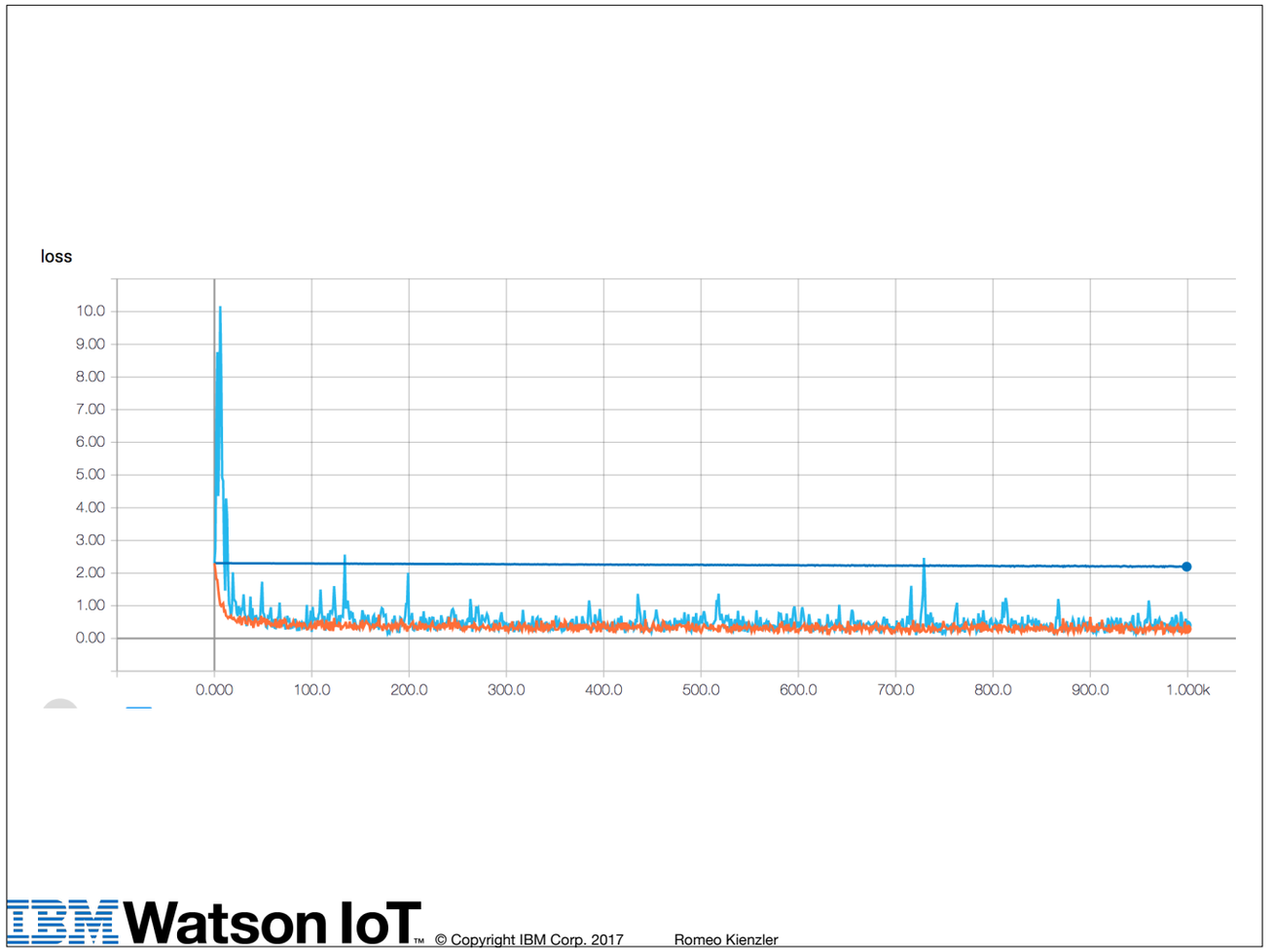
So what are the most important parameters to be looked at when training a neural network? So the most important is definitely the loss over time. So as you remember from previous videos - loss defines how good or bad the neural network fits the data and by the way, this here is a quite good loss function because it converges to a local optima quite fast



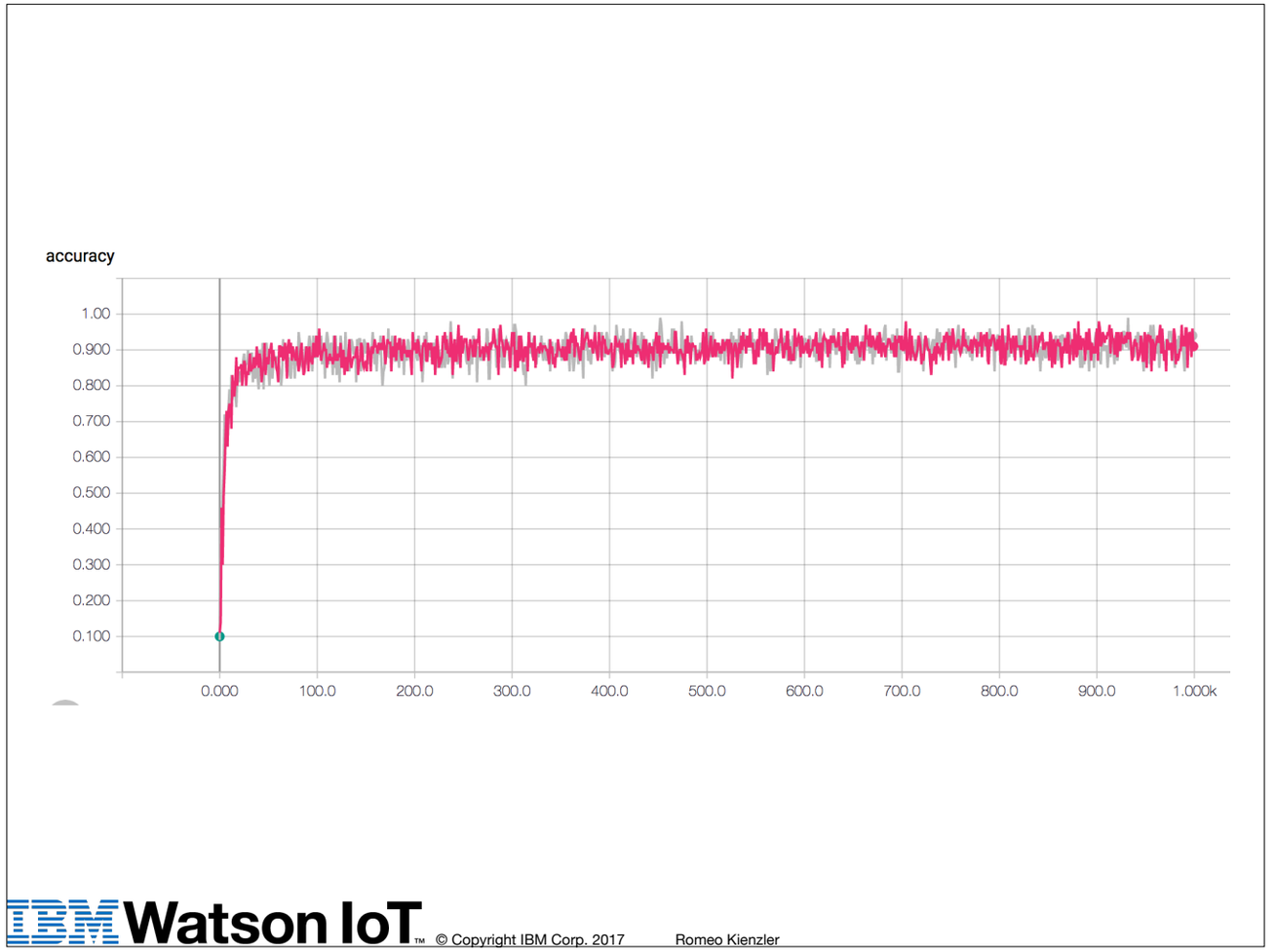
The blue line instead is indicating a learning rate which is too small. We are slowly converging against the optimum but it just takes too long.



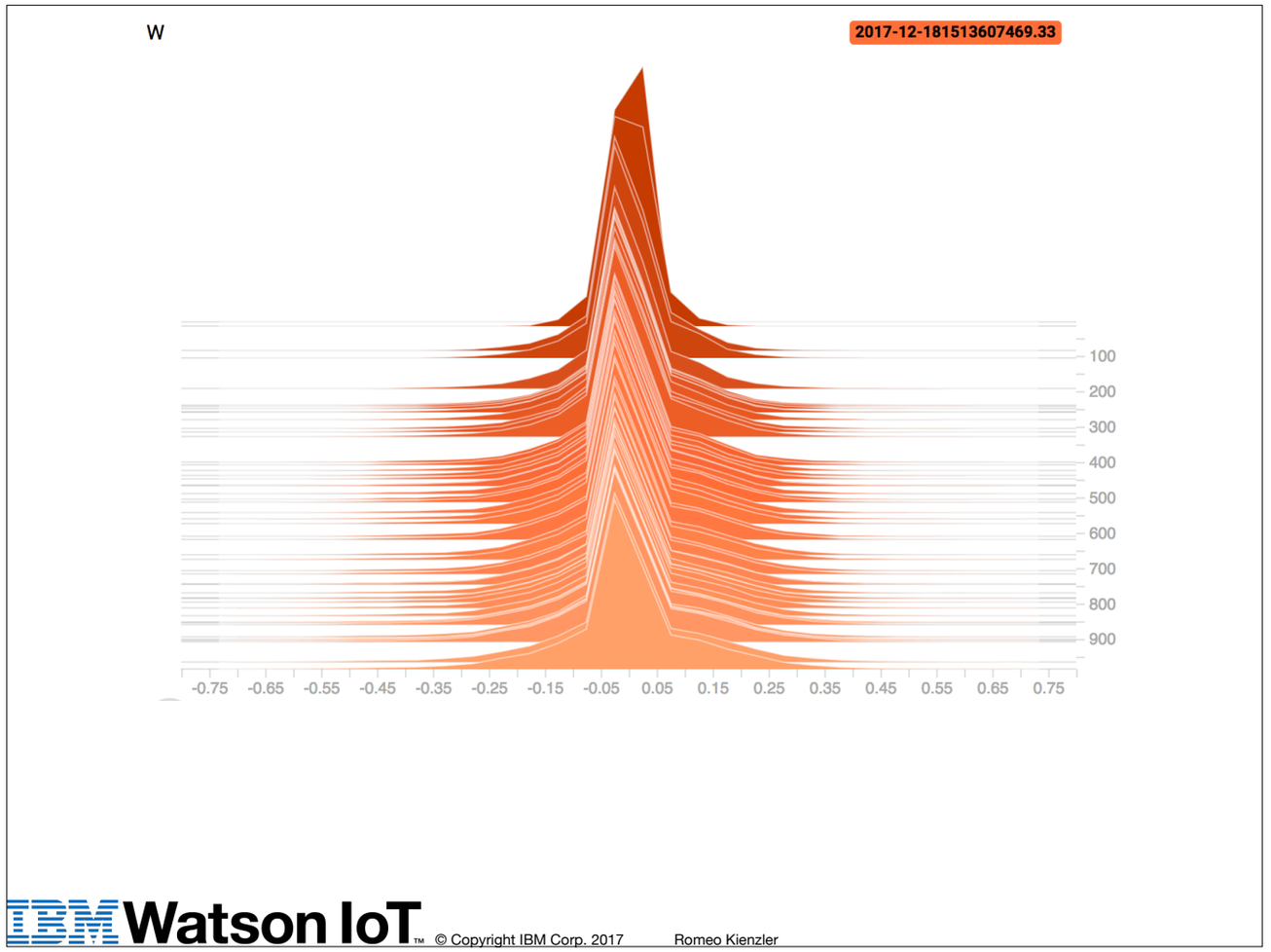
So let’s run this for 10 thousand iterations and we obtain the following diagram. It’s hard to say, but chances are high that with a too low learning rate we never reach the desired local optimum.



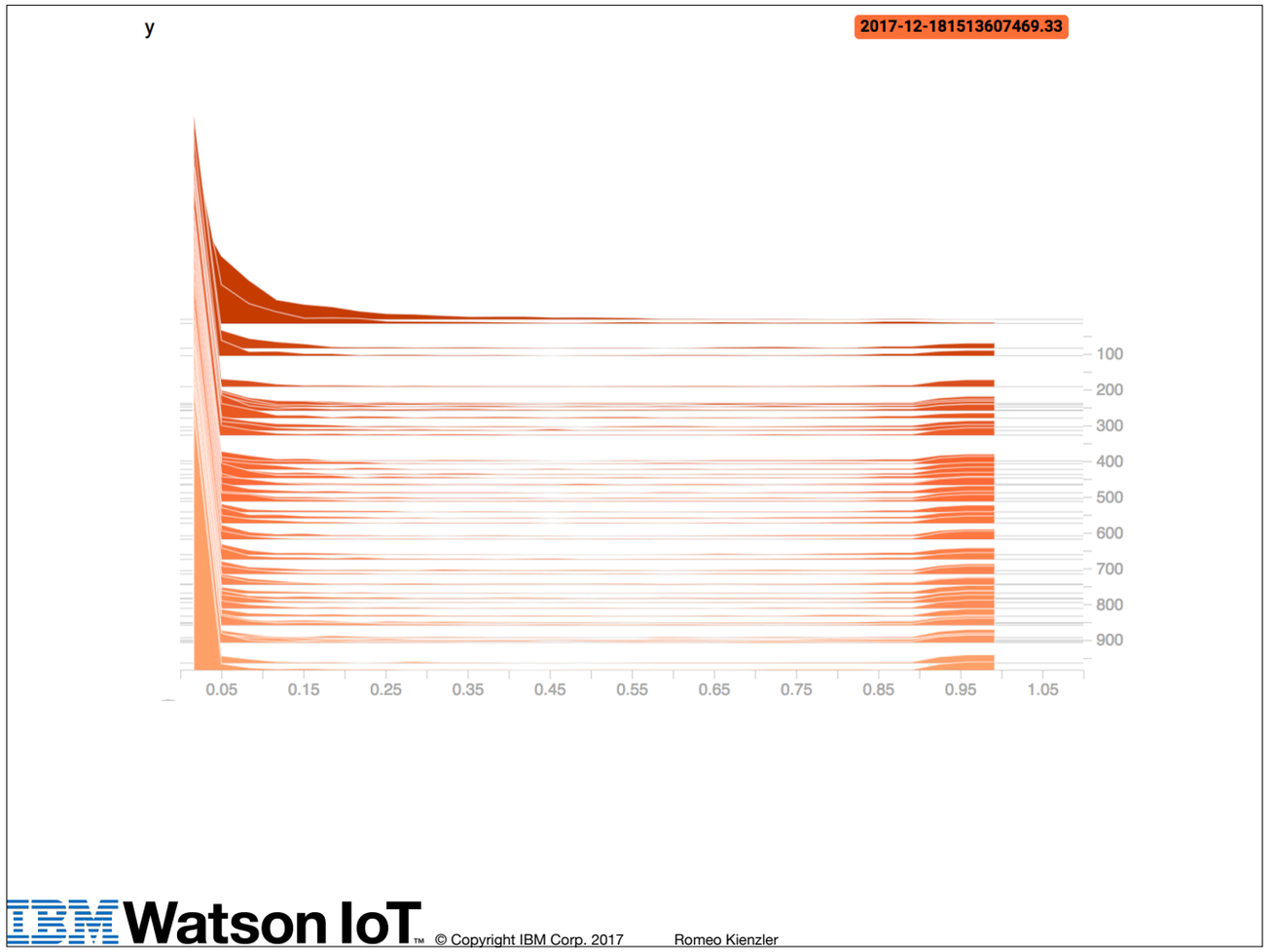
Finally, if we have chosen a learning rate which is too high we notice a considerable amount of bouncing and the only reason why we are still converging is that the underlying model we are optimising is rather simple.



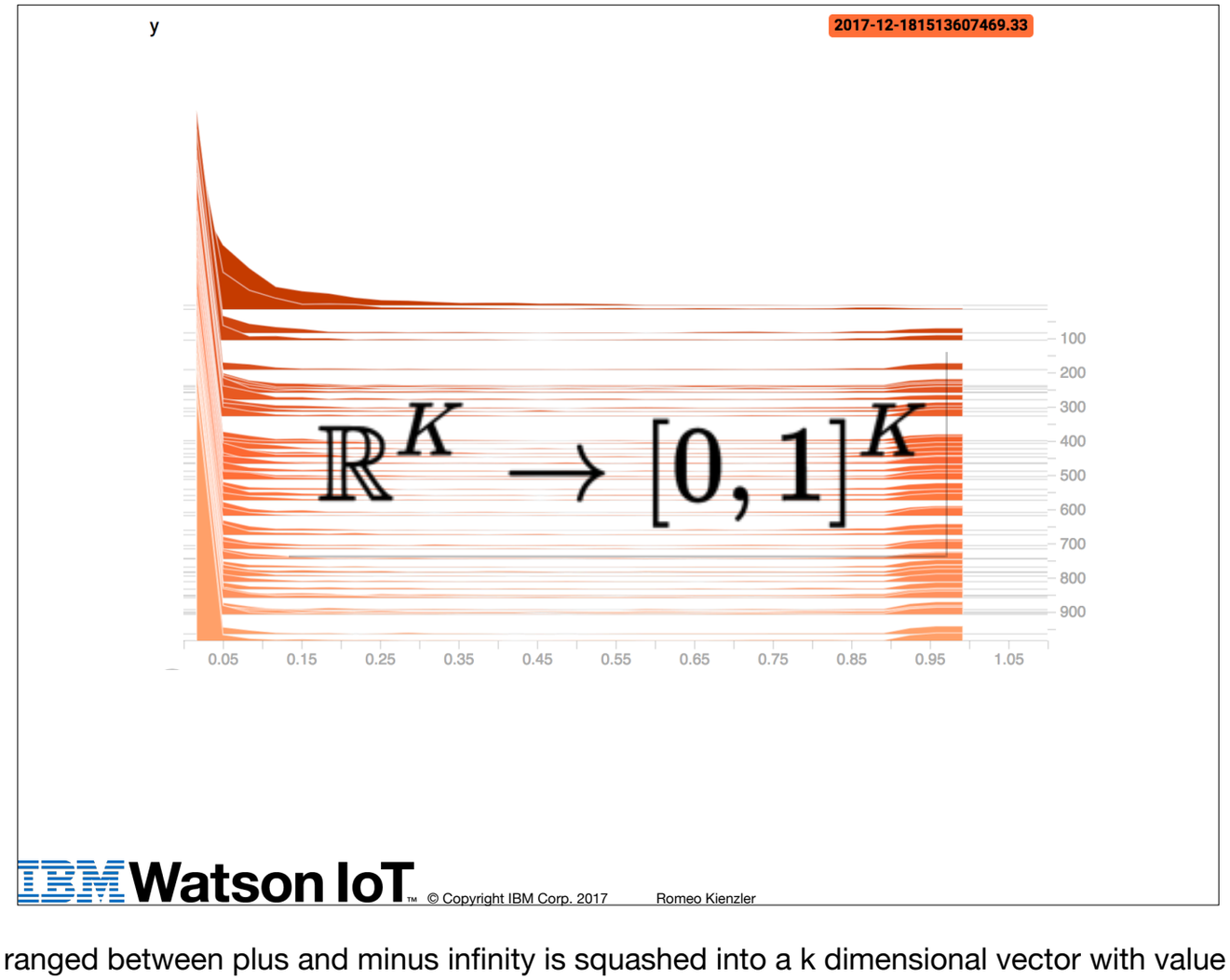
Another, maybe even more important measure is accuracy. Here we see accuracy over training iterations. And as loss goes down we should see accuracy going up. So those are scalar time series but now let's have a look at the weight matrices. Those contain many values and we have to find a way to visualise them at one.



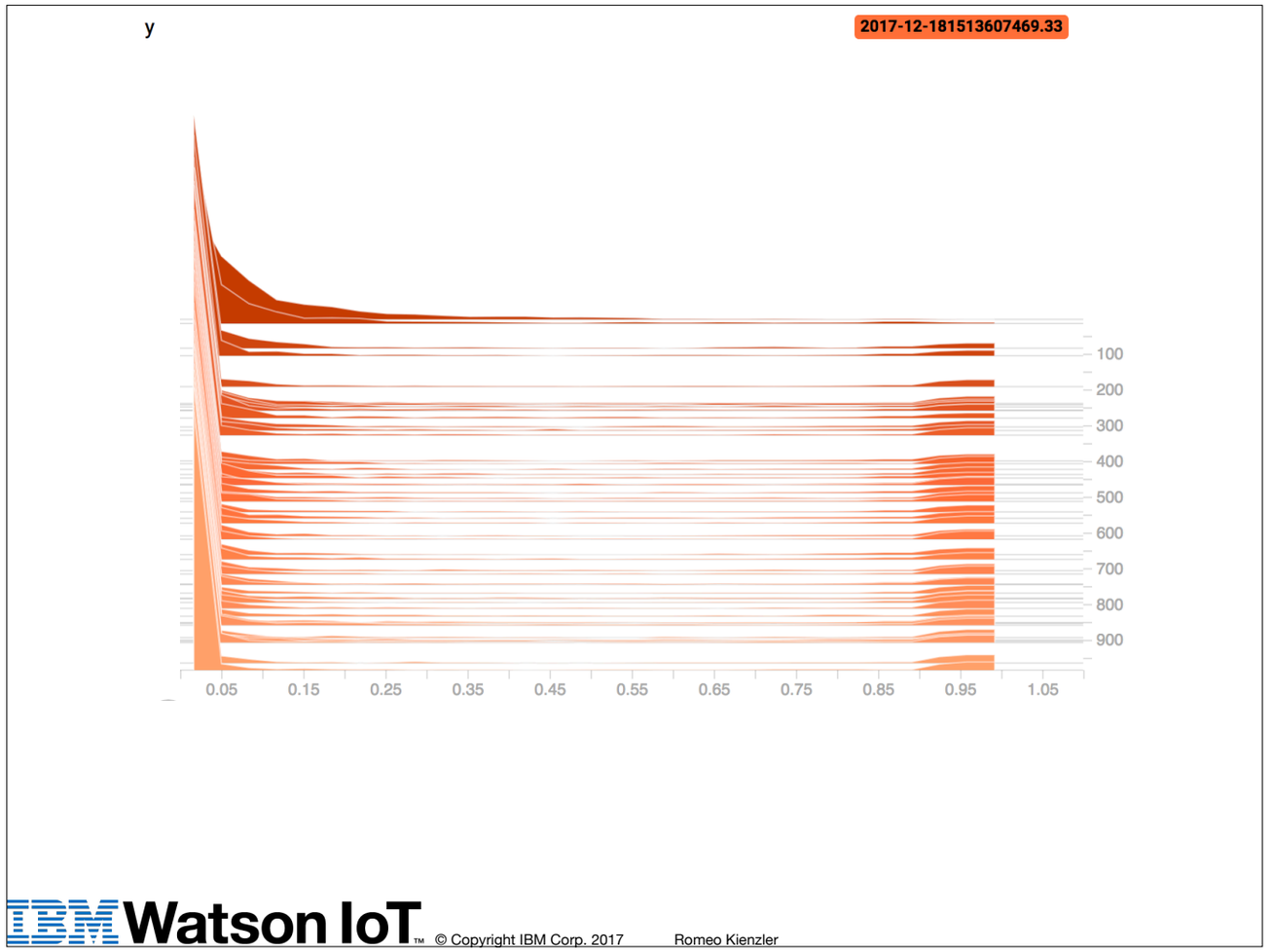
Fortunately TensorBoard does a pretty good job in creating summary over a weight matrix of arbitrary shape. You can find more on how those summaries are calculated in the video description. But here we see pretty healthy distribution of weights. Bad examples include cases where all values are very close to zero or a uniform distribution resembling the random weight initialisation. This can be an indication that the actual layer hasn't learned anything. Besides monitoring weights, we can also monitor activations. Note that the activation is the output of particular layer with the activation function applied. In this simple case we just use y.



Remember that y is the output of the softmax function.



So any k dimensional vector with value ranged between plus and minus infinity is squashed into a k dimensional vector with value ranges between zero and one. This is the default activation function for the output layer in classification tasks.



So here we see what we expect in a healthy classifier. Most of the values are close to zero because those are assigned for the probability a particular input is NOT in the class. And we also see a fair amount of values near to one. Those are the cases where a particular input IS in the class. Since we have ten different classes it is obvious to see more values close to zero that close to one.

a3_m1_s2_v2_tbintro_v3

 Romeo Kienzler

So let’s have a look at TensorBoard. Instructions on how to access TensorBoard from IBM DataScience Experience are given in the description section of this video.

TensorBoard can visualise multiple runs simultaneously in order to compare amount those but let’s only have a look at a single training run for now.

First we have a look at the scalar view. So in this tab, all summaries of scalars and how they evolve over training time are displayed. In our example we’ve recorded the two most important measures. Loss and accuracy. As you can see they are inverse which is actually a very good sign. The lower the loss, and therefore the error of the neural network is, the better the accuracy. We can maximise the plot and also adjust the smoothing parameter. If we set it to one we obtain a line which isn’t displayed here. So let’s decrease it slowly. As we can see, this line more and more fits the actual trajectory and with zero point nine six we can clearly see the trend without getting lost in too much details. And here again we can have a look at those two important measures and see that they are inverse. Remember that loss is based on the defined loss function - cross entropy in this case and accuracy tells us the fraction of correctly classified examples over the total number of examples.

So now have a look at the graph tab. This graph should be read bottom up. So we start with the weight matrix double you and the placeholder x for our input data. This is multiplied and then the bias variable b is added. This result is squashed using softmax and the final result is stored in y.

Note that y underscore and y are then used to calculate accuracy by taking the arg max of both. By comparing those and taking the mean of this vector we obtain the accuracy. The other branch of the graph computes loss using the cross entropy function.

Note that parts of the graph are hidden to us in an externalised subgraph. As you can see, the connection points of the sub graph are turning red now. We can include this subgraph but then the graph becomes more complex. This is because the gradient computation is reading values from variables and placeholder all over the place and doesn't add more information at that point. Therefore let’s remove it again.

Summary

There is much more to say about TensorBoard but we've covered the most important aspects for this course and we will see in future lectures how those metrics can be used in order to debug neural networks during training.

Automatic Differentiation

 Romeo Kienzler

In the next module we will cover Automatic Differentiation - one of the key features of TensorFlow