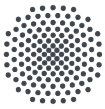
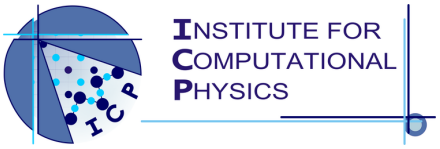


University of Stuttgart
Germany

Investigating the evolution of fisher information for neural network dynamics

Marc Sauter

ICP



University of stuttgart

A thesis presented for the degree of

B.Sc.

2023

I dedicate this ...

The Book of Nature is written in the language of mathematics.

— Galileo Galilei

Acknowledgements

I would like to express ...

Declaration

I, Marc Sauter, declare that ...

Signature	*Date*

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Keywords— Keyword1 - Keyword2 - Keyword3

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI . . . . . Artificial Intelligence

ML . . . . . Machine Learning

ReLU . . . . . Rectified Linear Unit

NN . . . . . Neural Network

SGD . . . . . Stochastic Gradient Descent

NTK . . . . . Neural Tangent Kernel

1 | Machine Learning Basics

1.1 Introduction to machine learning

Over 70 years ago in October of 1950, at a time when computers weighed several tons, could only perform a few thousand operations per second and the pinnacle of machine intelligence were analogous robots that could shakily follow light sources[2], Alan Turing published a paper in the journal of Nature discussing the question "Can machines think?"[1]. In there, he tries to tackle that question by replacing by instead proposing a game he calls the "imitation game". This game puts a human, whom we will call A , in a room where he can communicate via written messages with two different entities, one of which being a machine, the other being a human called B. A's goal is to determine from this simple communication alone which of the two entities is the human. The goal of both the machine and B is convincing A of themselves not being a machine.

The largest execution of such an experiment to date took place in early 2023 in the form of an online chat portal, where players had two minutes to talk to either another human or an artificial intelligence without knowing the type of their interlocutor. After more than two million participants had played the game for a total of more than 15 million conversations, only $68\,\%$ of the attempted classifications were correct guesses. All of the advanced AI-bots used in this experiment were achieved using machine learning methods.

The Oxford Learning Dictionary defines machine learning as "a type of artificial intelligence in which computers use large amounts of data to learn how to do tasks rather than being programmed to do them"[3]. How we can do this with neural networks and use them to learn complex tasks is explained in the following sections.

1.2 Neural networks

1.2.1 Neurons

The term "neural network" is a reference to the workings of nervous systems of animals. These systems consist of a net of neurons, which are biological cells that are intricately connected to other neurons through structures called synapses. These connections carry electrical pulses between the different neurons that can excite them when they exceed certain thresholds that vary from neuron to neuron and change over time. When this excitation occurs, new pulses in turn propagate from the excited neuron outwards to possibly excite other neurons. This interplay between excitation and transmission through the network of neurons is believed to be create what we perceive as thinking. Mathematical analyses of these systems in attempts to eventually understand and replicate this thinking process have been done as early as the 1940's.[4]

The artificial neural networks that we use for machine learning are modern attempts to replicate parts of those biological networks. The neurons, which are the building bricks of the artificial neural networks, are mathematical entities that take a fixed number of scalar values as inputs and convert them into a single output value. For a more visual explanation let's take a look at fig. 1.1. This is a case with two inputs. The big circle in the middle is representing the neuron itself. It takes as input the activation values a_i, multiplies them with their corresponding weights ω_i, sums them up, adds a bias value b and then applies the activation function to get the resulting output value. The input activations a_i correspond to the electronic pulses in the nerve system, with the strength of the pulse coded into the value itself. No pulse in the biological system would be represented by an activation of zero in the mathematical model. The weights ω_i are a representation

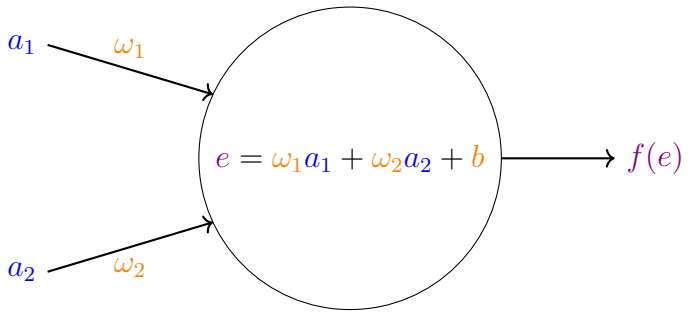


Figure 1.1: This figure aids the explanation of the general operating principle of neurons in neural networks as they are used in this thesis. The parameters ω_i are denoted in orange, the input activations a_i in blue and the activation e with its corresponding activation function f in violet.

of how important single input values are for the activation of the neuron itself. In the biological counterpart this might correspond to how thick or conductive the connections between the nerve cells are. Finally, the combination of bias b and activation function describes how high the sum of the input-weight-pairs has to be to activate the neuron and how the resulting value for the activation of the neuron changes for higher input activations. For example, a very simple output activation function would be a heaviside function ($f(x) = 1$ for $x \geq 0$ and $f(x) = 0$ for $x < 0$). The neuron then outputs 1 if the sum of the input-weight-pairs is bigger than the negative bias $-b$, which results in a positive value for e and 0 otherwise. One could say that the neuron can only be on or off. A more common activation function that will be used for the rest of this thesis is the ReLU function (Rectified Linear Unit). This function is defined as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{1.2.1}$$

Here the neuron gets activated as soon as the sum of the input-weight-pairs is higher than $-b$ and the value of the activation increases linearly with e for those

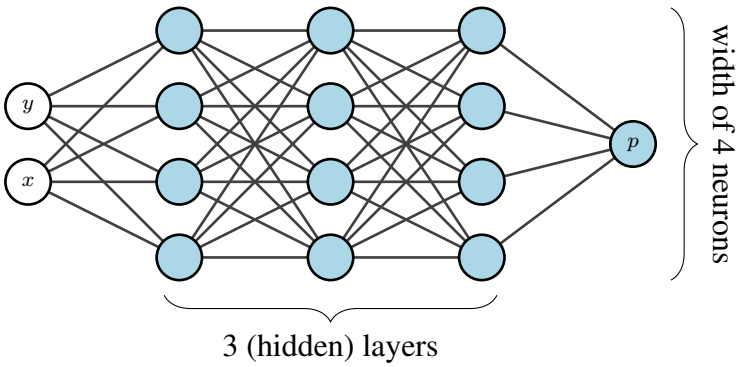


Figure 1.2: This figure shows an example of a neural network. It consists of multiple neurons connected to each other. This specific network has 2 input values and one output value and a width of 4 neurons for a total of 3 layers. The names of those input and output values correspond to the use case of identifying if a 2d point lies within a region in 2d space.

high sums.

1.2.2 Neural networks

A neural network can be built from these neurons, by connecting the outputs of certain neurons to the inputs of others. For example lets take a look at fig. 1.2. This neural network consists of three layers of four neurons each, takes two values as input and outputs one value at the end. It could for example be trained for detecting if a point on a 2d-grid is inside or outside of a given region. The input values would be the x and y coordinates of the point and the output value could represent the probability of the point being inside the desired region. How exactly this training would work will be explained in later sections.

This network is only a very specific example of what a neural network might look like. In reality, there are various kinds of networks used to learn different tasks. For this thesis we will only talk about "fully connected" or "dense" neural networks. This means that every neuron in the first layer will receive every possible input value, and every neuron in later layers will receive the output of every neu-

ron in the previous layer as an input. The activation function is the same for every neuron in this case, but the weights and biases vary throughout. How the output of the network gets handled may still differ through different use cases. Most of the time, the output is generated by regular neurons and gets fed into a softmax at the end. Details on the softmax function can be found in [5]. For our understanding it's enough to know that the softmax function converts the values of the output nodes into probabilities that add up to 1, where a higher value results in a higher corresponding probability. The structure of a neural network is generally called the architecture of the network, with the hidden layers or simply layers referring to the columns of neurons in between the input values and output neurons and the amount of neurons per layer referred to as the "width" of the network. This is also denoted in fig. 1.2.

1.2.3 Mathematical view

The previous explanations have been very visual and step by step in an attempt to make the topic more accessible. However, these concepts can be broken down to rather short mathematical expressions.

To start off we can define the inputs as $a_i^{(0)}$, $i = 1, \cdots n$, with n being the amount of inputs. The weights of the first hidden layer can be denoted by $\omega_i^{(1)}$, $i = 1, \cdots n$. Further we define $\omega_{i,j}^{(k)}$ as the weight that connects the i-th neuron in the k-th layer with the j-th neuron in the $(k-1)$-th layer. The values that i and j can assume depend on the widths of the respective layers, k can reach values between 1 and the amount of hidden layers plus 1, for the output layer. The bias of the i-th neuron in the k-th layer is denoted as $b_i^{(k)}$. Using this notation we can instantly write out

the output of the i-th neuron in the $k+1$-th layer as

$$a_i^{(k+1)} = f\left(\sum_j \left(\omega_{i,j}^{(k+1)} a_j^{(k)}\right) + b_i^{(k+1)}\right). \tag{1.2.2}$$

To actually calculate the value, the $a_j^{(k)}$ values have to be recursively replaced with their full calculation until one arrives at the input values of the network.
For simplicity reasons we will refer to the weights $\omega_{i,j}^{(k)}$ and biases $b_i^{(k)}$ together as "parameters" of the neural network. We will denote these collected in one ordered set as $\theta = \{\theta_i\}_{i=1}^N$ if N is the amount of weights and the amount of biases added together. The mapping of the parameters onto θ is not important as long as it is known so that one can calculate the output of a neural network if given θ in the same way as when one is given the parameters. Another possible representation that we will use often is to write this set as a vector $\theta \in \mathbb{R}^N$. A short notation for the output of the neural network will be explained in the next section.

1.3 Training of neural networks

In the previous sections we discussed how neural networks are built up. Now we will give an example of how they can be trained and used to perform specific tasks. Specifically, we will look at how they can be trained on data sets in a process called supervised learning. For this we fix the architecture of our neural network, which includes the organisation of neurons, the input and output handling and the activation function. The things we can vary during training to make our network perform better are the parameters θ.

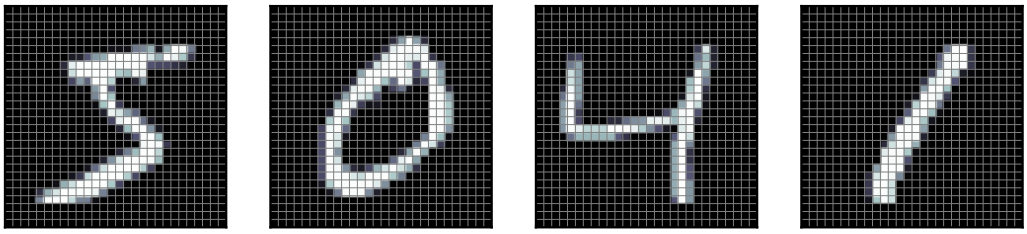


Figure 1.3: Displayed in this figure are 4 examples of input values for the MNIST dataset. This dataset represents handwritten digits from 0 to 9. Each input value consists of a 28 by 28 grid of grayscale pixel values.

1.3.1 Datasets

First we need to assume that we have a given dataset containing input-output pairs that represent the task we want the neural network to perform. The neural network is supposed to learn which output is supposed to be generated from which input by evaluating the given inputs and desired outputs. We will call those desired outputs "targets" to distinct them from the actual output of the neural network during training. A good example of this would be the MNIST database, which was first used in 1994 in [6]as a modification of an earlier database, and has since become a popular entry-level classification task for machine learning. Examples of input data for this database are shown in fig. 1.3. This dataset consists of various handwritten digits from 0 to 9 represented by a 28 by 28 grid of grayscale pixel values. The values of the pixels in these grids act as the 784 input values for the neural network, the targets should be shaped according to the output of the neural network. For example one might use a scalar output of the neural network that should be equal to the value of the digit drawn in the input pixel grid. In this case the targets are scalar values from 0 to 9 corresponding to their input pictures. Another way would be to use 10 output values together with the previously mentioned softmax function (see section 1.2.2) to recieve 9 values as probabilities for the different numbers as output. We would then change our targets to vectors with 9 elements,

where the entry corresponding to the handwritten digit in the input picture would be 1, every other value would be 0.

We will denote the data sets as mathematical sets of input-target pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ are the input values and $\mathbf{y}_i$ are the targets. Here we called the amount of training data N. The mathematical dimension of $\mathbf{x}_i$ and $\mathbf{y}_i$ can vary and are dependent on the network architecture, but we will assume that they are vectors of the real numbers $\mathbf{x}_i \in \mathbb{R}^a$ $\mathbf{y}_i \in \mathbb{R}^b$. Due to this property we also sometimes refer to them as "points". If our inputs values are organized in a different manner, for example the MNIST data being matrices of $\mathbb{R}^{28\times 28}$ we can rearrange these numbers into a vector of $\mathbb{R}^{784}$ in any way we want. The only important thing is that we convert every data point into a vector in the same way.
Going further we will denote all the output values of the neural network for the input $\mathbf{x}_i$ by $f_\theta(\mathbf{x}_i)$. This means that we mathematically describe the whole behavior of the neural network as a function

$$f : \mathbb{R}^a \times \mathbb{R}^p \to \mathbb{R}^b \tag{1.3.1}$$

with p being the amount of parameters in the network.

1.3.2 Loss function

Now we have defined what a neural network is and how the input data we want to train on is structured. In order to train our networks to behave according to our data set, we now need to introduce a way to measure how well our network is performing. Once we can evaluate the performance of our network, we can then introduce ways to optimize that performance.
This measure of the performance of a neural network is called the "loss function"

$\mathscr{L}$. It is also sometimes referred to as the cost function in literature. For this study we will only consider loss functions of the form

$$\mathscr{L}\left(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, \theta\right) = \frac{1}{N}\sum_{i=1}^{N} \ell_\theta\left(\mathbf{x}_i, \mathbf{y}_i\right). \tag{1.3.2}$$

As a reminder, θ is a set containing the parameters of the neural network. This is the variable of the loss function that describes the neural network. How one calculates the output of the neural network using the inputs and θ has to be intrinsically defined as well. Sometimes it's more convenient to also denote this dependency, which would make the loss function

$$\mathscr{L}\left(\{(f_\theta(\mathbf{x}_i), \mathbf{y}_i)\}_{i=1}^N\right) = \frac{1}{N}\sum_{i=1}^{N} \ell_\theta\left(f_\theta(\mathbf{x}_i), \mathbf{y}_i\right). \tag{1.3.3}$$

How exactly the loss function should be defined cannot be generally stated. The only thing certain is that the value of the loss function should generally be smaller the closer the output of the neural network is to the corresponding targets.
In most of our cases we use loss functions with

$$\ell_\theta\left(\mathbf{x}_i, \mathbf{y}_i\right) = d\left(f_\theta(\mathbf{x}_i), \mathbf{y}_i\right), \tag{1.3.4}$$

where d represents the distance between the output vector of the neural network and the target vector according to different metrics. One very simple example would be

$$\mathscr{L}\left(\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, \theta\right) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(f_\theta(\mathbf{x}_i)_j - y_{i,j}\right)^2, \tag{1.3.5}$$

with $f_\theta(\mathbf{x_i})_j$ and $y_{i,j}$ being the j-th component of the network output and target vectors. Further examples and more general loss functions can be seen in [7].

1.3.3 Optimization

To quickly recap, we have now defined what a neural network is and that we want a fixed network architecture during training whose parameters we can vary to optimize the performance. We also assume that we have a data set that we want our network to behave like and a loss function that measures how well the current setup of our network is performing. We will now talk about how we can change the parameters in an attempt to optimize performance, which is achieved by lowering the value of the loss function.

The simplest and most common algorithm to do this is stochastic gradient descent (SGD). Here, the rule for updating the parameters looks like

$$\theta' = \theta - \eta \nabla_\theta \mathscr{L} \left[\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, \theta \right], \tag{1.3.6}$$

where η is a parameter of training called the "learning rate", which is a scalar value for SGD but could also vary through training and even be a tensor for more advanced optimization methods. The use of brackets instead parentheses is only for visibility reasons.

The idea behind this is that the gradient with respect to θ points in the direction of the steepest ascent of the loss when the parameters are varied. If we subtract this gradient from the parameters we change the loss in the opposite direction, which is the direction of steepest descent. To visualize this lets examine a single parameter that we will call θ_1. The update rule for this single parameter can be obtained from writing out the previous eq. (1.3.6) in its full vectorial component form and then pick out the line of θ_1. It reads

$$\theta_1' = \theta - \eta \frac{\partial}{\partial \theta_1} \mathscr{L} \left[\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N, \theta \right]. \tag{1.3.7}$$

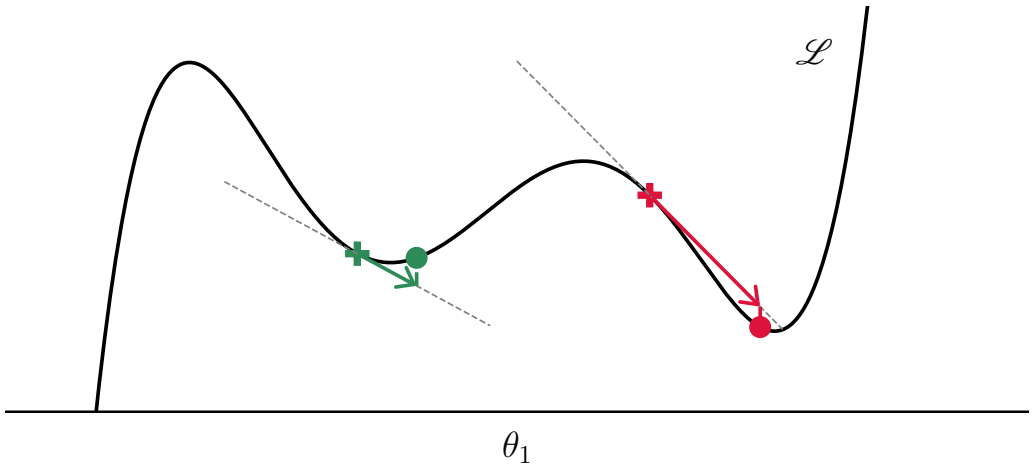


Figure 1.4: This figure is supposed to be a visual explanation for the stochastic gradient descent algorithm. The dot markers show where a parameter that starts at the cross marker might end up if the algorithm is applied.

To explain the concept of this algorithm more visually let's take a look at fig. 1.4. Here the cross markings symbolize the position of parameter θ_1 before the update step. The algorithm then calculates the derivative of the loss, which is the slope of the tangent. Now, to change the parameter, the negative slope is added to θ_1, which is represented by the arrows. These arrows are adjusted in length so that the distance they cover in the direction of θ_1 is proportional to the derivative. They point in the direction of the tangent to show their dependence on the slope. One can see that the red arrow on the right covers more distance in the direction of θ_1 than the green one on the left, which is a result of the corresponding slope being steeper. The circle markers then represent where the parameter values end up after the update step.

This visual example already illustrates some of the drawbacks of the SGD algorithm. For example, the green starting parameter will end up in a local minimum that has a higher loss value than the local minimum that the red parameter value tends to. Also going further to the left would make the loss function smaller than

both of these local minima. SGD just tends to find the closest local minimum instead of a global one. Another drawback results from the step size being finitely big, which makes the green parameter hop over the local minimum. Both of problems can't be solved with optimization of the learning rate η in general.

1.3.4 Other optimizers

The SGD is a very simple and fast to compute algorithm, but it can only find local minima and can quite easily hop over the actual minima. That's why other algorithms have been developed that attempt to solve or mitigate some of the issues of SGD. For the training that will be performed later in this thesis the so-called ADAM optimizer was used. The intricacies of this optimization method are beyond the scope of this work. We refer the interested reader to [8], where the details of this algorithm are discussed. Another optimization algorithm that can be used to overcome some problems of the SGD is the Natural Gradient Descent, which will be mentioned in later sections.

1.4 Which assumptions are actually necessary

In the previous sections we took a brief look at machine learning and neural networks. Although this is still barely scratching the surface of the methods that are used today, it is still more than what's needed for the upcoming chapters.
It is sufficient for all mathematical considerations that follow, to view systems $f_\theta(\mathbf{x}_i)$ that depend on parameters θ, accept input vectors $\mathbf{x}_i$ of constant dimension and can be evaluated through the formalism of the loss function from equation eq. (1.3.2). For the discussions of the NTK, we can even disregard the assumption of the loss function splitting up into a sum of ℓ functions. This means that the exact properties of the neural networks we looked at earlier can be disregarded, which

means we can generalize the observations to many more network architectures or completely different systems than previously mentioned.

2 | Neural Tangent Kernel

2.1 Derivation from Gradient Flow

2.1.1 What we call time

A simple and intuitive way to introduce the NTK is via "**Gradient Flow**", which is an assumption related to the SGD algorithm from section 1.3.3. To quickly recap the update step from stochastic gradient descent, it is defined as

$$\theta' = \theta - \eta \nabla_\theta \mathscr{L} \left(\{ (f_\theta(\mathbf{x}_i), \mathbf{y}_i) \}_{i=1}^N \right). \tag{2.1.1}$$

The "flow" aspect arises when we start to ignore the discrete nature of these update steps and assume that the θ parameters change continuously. To do this, we first introduce a notion of "time" into our system. We try to visualize the optimization process as an evolution of our parameters θ through this variable called time, which converts updating the parameters into moving further along the timeline of our parameters. We now translate $\theta \to \theta(t)$ and $\theta' \to \theta(t + \Delta t)$. This time in our system doesn't work exactly the same way as physical time, but since the process of calculating better parameters and changing them is always associated with the expenditure of physical time, it is intuitive to refer to our system's propagation variable as "time".

Returning to the SGD algorithm, since the learning rate η affects the size of our update step, we will refer to η as the amount of time it takes to update a parameter $\theta' \to \theta(t + \eta)$. The whole SGD algorithm then becomes

$$\theta(t + \eta) = \theta(t) - \eta \nabla_{\theta(t)} \mathscr{L} \left(\{ (f_{\theta(t)}(\mathbf{x}_i), \mathbf{y}_i) \}_{i=1}^N \right) \tag{2.1.2}$$

which we can rewrite to

$$\frac{\theta(t+\eta)-\theta(t)}{\eta} = -\nabla_{\theta(t)}\mathscr{L}\left(\{(f_{\theta(t)}(\mathbf{x}_i),\mathbf{y}_i)\}_{i=1}^{N}\right). \tag{2.1.3}$$

When observing the term on the left side, readers who are familiar with calculus might recognize that it looks similar to the definition of a derivative in time

$$\frac{\partial}{\partial t}\theta(t) = \lim_{\eta\to 0}\frac{\theta(t+\eta)-\theta(t)}{\eta}. \tag{2.1.4}$$

This means that for very small learning rates we can approximate the SGD as

$$\frac{\partial}{\partial t}\theta(t) = -\nabla_{\theta(t)}\mathscr{L}\left(\{(f_{\theta(t)}(\mathbf{x}_i),\mathbf{y}_i)\}_{i=1}^{N}\right) \tag{2.1.5}$$

with the partial derivative of θ along the assumed time variable.

For visual simplicity reasons, let's define the j-th component of the network output for the i-th input point $f_{\theta(t)}(\mathbf{x}_i)_j$ as f_{ij} and assume Einstein summation for the rest of the chapter. This means that when an index is occuring twice we don't denote a hidden summation over all possible values for this index (for example $a_k b_k = \sum_k a_k b_k$).

Because it will be convenient later we also spell out one component of the ∇_θ derivation of the loss function further by using the chain rule as

$$\begin{aligned}\frac{\partial}{\partial t}f_{\theta(t)}(\mathbf{x}_i)_j = \frac{\partial\theta_k}{\partial t} &= -\frac{\partial}{\partial\theta_k}\mathscr{L}\left(\{(f_{\theta(t)}(\mathbf{x}_i),\mathbf{y}_i)\}_{i=1}^{N}\right)\\ &= -\frac{\partial\mathscr{L}}{\partial f_{ij}}\cdot\frac{\partial f_{ij}}{\partial\theta_k}.\end{aligned} \tag{2.1.6}$$

2.1.2 Derivation of the NTK

This notion of time affects not only the parameters, but also everything that depends on them. For example, since the network output of a fixed architecture for a given input data point only depends on the parameters of the network, it can also be mathematically viewed as dependent on the time $f_\theta(\mathbf{x}_i) \to f_{\theta(t)}(\mathbf{x}_i)$. This means we can also calculate the derivative of one of the network outputs $f_{\theta(t)}(\mathbf{x}_i)_j = f_{ij}$ to

$$\begin{aligned} \frac{\partial}{\partial t} f_{ij} &= \frac{\partial f_{ij}}{\partial \theta_k} \frac{\partial \theta_k}{\partial t} \\ &= \frac{\partial f_{ij}}{\partial \theta_k} \left(-\frac{\partial \mathscr{L}}{\partial f_{ij}} \cdot \frac{\partial f_{ij}}{\partial \theta_k} \right) \\ &= -\underbrace{\frac{\partial f_{ij}}{\partial \theta_k} \frac{\partial f_{lm}}{\partial \theta_k}}_{=: \Lambda_{i,l,j,m}} \frac{\partial \mathscr{L}}{\partial f_{lm}}. \end{aligned} \tag{2.1.7}$$

The rank 4 hypermatrix Λ is what we call the Neural Tangent Kernel for SGD. We sorted the indices of this matrix so that the first two refer to the input points of f and the last two refer to the components of the output dimensions of the neural network. Note that this NTK is derived directly from the update algorithm of the SGD for infinitely small η, the equations above don't hold for other optimization systems.

Another way to derive the NTK using an approximation of $\Delta\mathscr{L}$ for small η can be seen around page 196 of [10]. The NTK derived there also works for tensorial gradient descent with a learning rate tensor η_{ij}.

2.2 Interpretation

Through eq. (2.1.7) the NTK is defined as

$$\Lambda_{i,j,\alpha,\beta} = \nabla_\theta f_\theta(\mathbf{x}_i)_\alpha \cdot \nabla_\theta f_\theta(\mathbf{x}_j)_\beta. \tag{2.2.1}$$

For now, let's assume that the neural network has only one output $f_\theta(\mathbf{x}_i)_\alpha \rightarrow f_\theta(\mathbf{x}_i)$, which gets rid of the α and β indices for us and makes the NTK a regular matrix. We can use this matrix to calculate the "time" derivative of the neural network output similar to eq. (2.1.7) by

$$\frac{\partial}{\partial t} f_{\theta(t)}(\mathbf{x}_i) = \sum_j \Lambda_{i,j} \left(-\frac{\partial \mathscr{L}}{\partial f_{\theta(t)}(\mathbf{x}_j)} \right). \tag{2.2.2}$$

This means that the evolution of the network output for input $\mathbf{x}_i$ is influenced by the outputs for other input values $\mathbf{x}_j$ through the NTK. We can investigate this further by taking a look at the definition of the NTK above in eq. (2.2.1).
A mathematical kernel K is defined as a function

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_i), \tag{2.2.3}$$

with another so-called "feature map" function ϕ that maps the points $\mathbf{x}_i$ into a higher dimensional inner product space called the "feature space". The kernel K assigns a scalar value to the two points by comparing their "features" via a scalar product in the feature space. In our case the feature map is ∇_θ which maps our scalar network output onto a vector that contains the derivatives of the network output with respect to all of the various parameters. This is our feature space, because what we're interested in is how moving in θ space changes the output values of our network. If we compare those two mappings we get a value that

measures how closely the direction that increases $f_\theta(\mathbf{x}_i)$ in the most effective way matches with the direction that increases $f_\theta(\mathbf{x}_j)$ most effectively (which are the directions the gradients points to). For example, $\Lambda_{i,j} = 0$ means that varying θ in the direction that changes $f_\theta(\mathbf{x}_i)$ most effectively results in 0 change for $f_\theta(\mathbf{x}_j)$.

Coming back to eq. (2.2.2) the right side is the negative loss derivative with respect to $f_\theta(\mathbf{x}_j)$. Since we update the parameters in a way that minimizes the loss most effectively in gradient flow, the negative loss derivative with respect to $f_\theta(\mathbf{x}_j)$ describes how beneficial increasing $f_\theta(\mathbf{x}_j)$ is for our system. If we now multiply this with the NTK, which is a kernel that describes how similar $f_\theta(\mathbf{x}_i)$ and $f_\theta(\mathbf{x}_j)$ behave when changing theta, and sum everything up, we get the change of the function value $f_\theta(\mathbf{x}_i)$ as result.

In θ space, we can directly relate the evolution of θ to its negative loss derivative, because the parameters themselves are what we update with our algorithm. For the derivative of the output values we need the NTK as well, because we don't change the output values directly. If the loss derivative with respect to an output value tells the system that it *should increase this output value*, the system *changes the parameters* in a way that increases this output value. The difference to the derivative of θ is that the change of parameters now doesn't reflect in just one output, but in every other output value as well. That's where the NTK arises in the equation. The NTK is, in a way, a translation of the SGD into the space of outputs of the neural network.

All of the explanations above apply to the 4 dimensional hypermatrix form of the NTK for multidimensional neural network output as well, which can be seen when comparing eq. (2.2.2) to eq. (2.1.7).

Literature

[1] A. M. TURING. "I.COMPUTING MACHINERY AND INTELLIGENCE". In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[2] Esther Inglis-Arkell. Mar. 7, 2012. URL: https://gizmodo.com/the-very-first-robot-brains-were-made-of-old-alarm-cl-5890771 (visited on 08/23/2023).

[3] URL: https://www.oxfordlearnersdictionaries.com/us/definition/english/machine-learning (visited on 08/23/2023).

[4] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133. DOI: https://doi.org/10.1007/BF02478259.

[5] Bolin Gao and Lacra Pavel. "On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning". In: (2018). arXiv: 1704.00805 [math.OC].

[6] L. Bottou et al. "Comparison of classifier methods: a case study in handwritten digit recognition". In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*. Vol. 2. 1994, 77–82 vol.2. DOI: 10.1109/ICPR.1994.576879.

[7] Qi Wang et al. "A Comprehensive Survey of Loss Functions in Machine Learning". In: *Annals of Data Science* 9 (2 2022), pp. 2198–5812. DOI: 10.1007/s40745-020-00253-5.

[8] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2017). arXiv: 1412.6980 [cs.LG].

[9] Colin Campbell. “An introduction to kernel methods”. In: *Studies in Fuzziness and Soft Computing* 66 (2001), pp. 155–192.

[10] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, May 2022. DOI: 10.1017/9781009023405. URL: https://doi.org/10.1017%2F9781009023405.

[11] Daniel Jannai et al. *Human or Not? A Gamified Approach to the Turing Test*. 2023. arXiv: 2305.20010 [cs.AI].

[12] S. Amari and S.C. Douglas. “Why natural gradient?” In: 2 (1998), 1213–1216 vol.2. DOI: 10.1109/ICASSP.1998.675489.

[13] Enzo Grossi and Massimo Buscema. “Introduction to artificial neural networks”. In: *European Journal of Gastroenterology and Hepatology* 19.12 (Dec. 2007), pp. 1046–1054. DOI: 10.1097/MEG.0b013e3282f198a0.

A | Appendix Example