SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Convolution on compact groups and natural language processing

av

Mbwenga Maliti

2023 - No K28

Convolution on compact groups and natural language processing

Mbwenga Maliti

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Rikard Bøgvad

2023

Abstract

There is a mathematical framework that can be used for explicitly constructing neural networks with internal representations of its input that are equivariant to the action of an arbitrarily chosen compact group. I have described parts of it and used a simple model of languages and translations to examine a part of the aforementioned framework necessary for applying it to natural language processing. There is a significant amount of work remaining in order to build a full neural network according to the mathematical framework, as well as in order to incorporate more sophisticated models of language and translations into it.

Sammanfattning

Det finns ett matematiskt ramverk som kan användas för att konstruera neurala nätverk med interna representationer som är symmetriska under gruppverkan av en godtyckligt vald kompakt grupp. Jag beskriver delar av detta ramverk och använder en enkel model av språk och översättningen mellan dem för att undersöka delar av det förnämnda ramverket som behövs för att tillämpningar inom behandlingen av naturligt språk. Det återstår en betydande mängd arbete för att konstruera ett neuralt nätverke i enlighet med ramverket, samt att introducera mer sofistikerade modeller av språk och översättningen mellan dem.

Contents

1 Artificial neural networks **5**
1.1 Binary classification 5
1.2 Artificial neurons 5
1.3 Deep neural networks 11
1.4 The convolution layer 22

2 Group convolutional neural networks **26**
2.1 G-convolution layer 32
2.2 G-CNN construction 35

3 Linguistic convolution on finite language graphs **37**
3.1 Sample dataset 38
3.2 Translation group 39

4 Conclusions & next steps **39**

References **41**

1

Artificial neural networks

1.1 Binary classification

A common problem is the problem of binary classification. Given two properties A and B that some X may exhibit, how can we infer what property X has without knowing it beforehand? One example of a binary classification problem is where we have an 1000x1000 pixel image and want to determine if the image contains a bus or not. It may be an easy problem for a human to solve, but for an algorithm the only information it has is the vector in $\mathbb{R}^{10^6}$ that the image is represented by. And that is assuming the image is in black and white, where only opacity needs to be set. Representing an image with color would require a vector in $\mathbb{R}^{10^{18}}$, seeing as each color can be described as a combination of the foundational colors red, green and blue at varied levels of opacity. More abstractly, we may view the problem of binary classification as that of for two subsets A, B of some set C such that

$$A \cap B = \emptyset$$

finding a function $f_\delta : C \to \{\top, \bot\}$ such that:

$$f_\delta(x) = \begin{cases} \top & \text{if} \quad x \in A \\ \bot & \text{if} \quad x \in B \end{cases}$$

Sometimes, a solution to this problem is a *McCulloch-Pitt Neuron neuron*:

1.2 Artificial neurons

Definition 1.1. An *McCulloch-Pitt Neuron*(MPN) is a function $f(W(X + B)^\top)$ where $X = (x_1, \ldots\ x_n)$ are variables, $W = (w_1, \ldots\ w_n)$ and $B = (b_1, \ldots\ b_n)$ are

constants, $X, W, B \in \mathbb{R}^n$, $n \in \mathbb{N}$ and f is a function such that

$$f(C) = \begin{cases} 1 & \text{if C} > 0 \\ 0 & \text{if C} \leq 0 \end{cases}$$

.

The constants W of an MPN are referred to as the *weights* of the MPN. B is known as the *bias* of the MPN. Suppose that A_1 and A_2 are two disjoint regions of $\mathbb{R}^2$, and that they lie on each side of the line $x = -y$, with A_1 containing elements (x, y) such that $x+y > 0$. Then we can see that the MPN f with weights $W = (1, 1)$ and bias $B = (0, 0)$ can decide whether $X = (x, y) \in \mathbb{R}^2$ is in A_1 or A_2, since:

$$f(WX^\top) = \begin{cases} 1 & \text{then} \quad X \in A_1 \\ 0 & \text{then} \quad X \in A_2 \end{cases}$$

As an MPN can only classify regions separated by a hyperplane, the types of regions an MPN can classify are characterized by a result commonly referred to as the *hyperplane separation theorem*. Before proving it, we must cover some definitions:

Definition 1.2. A *metric space* is a set M with a function $d : M \to \mathbb{R}$ such that:

- $\forall x, y \in M : d(x, y) \geq 0$
- $\forall x, y \in M : d(x, y) = d(y, x)$
- $\forall x, y \in M : d(x, x) = 0$ and $d(x, y) = 0 \implies x = y$
- $\forall x, y, z \in M : d(x, y) + d(y, z) \geq d(x, z)$.

The function d is known as the *metric* of M.

It is clear that $\mathbb{R}^n$ can be viewed as a metric space with euclidean distance as its metric.

Definition 1.3. A subset of a metric space $N \subset M$ is *bounded* if there exists some real number $r > 0$ such that $\forall x, y \in N : d(x, y) < r$.

For any $C \subset \mathbb{R}^n$, the following definitions apply:

Definition 1.4. A point $a \in \mathbb{R}^n$ is known as a *limit point* of the subset if every open ball around a contains both points from C and the complement of C.

Definition 1.5. C is known as *closed* if it contains all of its limit points.

Definition 1.6. If C is both bounded and closed, it is *compact*.

Definition 1.7. If for all $x, y \in C$ the line segment connecting x and y is contained in C, then C is *convex*.

We now state a remark necessary for proving the theorem:

Remark 1.8. Let A and B be two disjoint closed subsets of $\mathbb{R}^n$, and assume A is compact. Then there exists points $a_0 \in A$ and $b_0 \in B$ minimizing the distance $||a - b||$ over $a \in A$ and $b \in B$.

Proof. Let $a \in A$ and $b \in B$ be any pair of points, and let $r_1 = \|b - a\|$. Since A is compact and thus bounded, it is contained in some ball centered on a; let the radius of this ball be r_2. Let $S = B \cap \overline{B_{r_1+r_2}(a)}$ be the intersection of B with a closed ball of radius $r_1 + r_2$ around a. Then S is compact since the intersection of two closed and bounded sets is closed and bounded, and nonempty because it contains b. Furthermore, the product of two compact sets in $\mathbb{R}^n$ is a compact set in $\mathbb{R}^n \times \mathbb{R}^n$. And a continous function on a compact set attains its minimum and maximum value[PB2]. Since the distance function is continuous, there exist points a_0 and b_0 whose distance $\|a_0 - b_0\|$ is the minimum over all pairs of points in $A \times S$. It remains to show that a_0 and b_0 in fact have the minimum distance over all pairs of points in $A \times B$. Suppose for contradiction that there exist points a' and b' such that $\|a' - b'\| < \|a_0 - b_0\|$. Then in particular, $\|a' - b'\| < r_1$, and by the triangle inequality, $\|a - b'\| \leq \|a' - b'\| + \|a - a'\| < r_1 + r_2$. Therefore b' is contained in S, which contradicts the fact that a_0 and b_0 had minimum distance over $A \times S$. □

We can now prove the theorem:

Theorem 1.9. *If $A, B \subset R^n$ are disjoint, closed, bounded and convex then there exists some hyperplane separating them.*

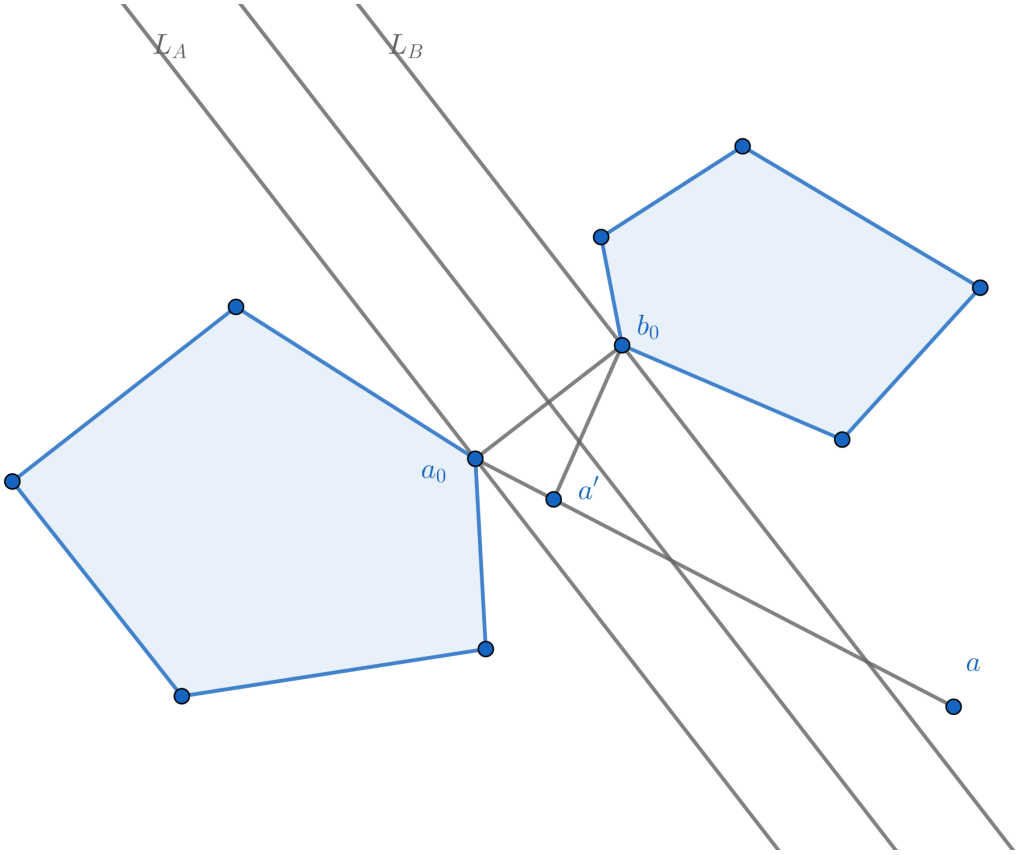


Figure 1: Proof illustration

Proof. By Definition 1.6, both sets are compact. By the previous remark, there exist points $a_0 \in A$ and $b_0 \in B$ of minimum distance to each other. Now, construct two hyperplanes L_A, L_B perpendicular to the line segment $[a_0, b_0]$, with L_A across a_0 and L_B across b_0. We claim neither A nor B enters the space between L_A, L_B, and thus perpendicular hyperplanes to (a_0, b_0) satisfy the theorem.

Algebraically, the hyperplanes L_A, L_B are defined by the vector $v := b_0 - a_0$, and two constants $c_A := \langle v, a_0 \rangle < c_B := \langle v, b_0 \rangle$, such that $L_A = \{x : \langle v, x \rangle = c_A\}, L_B = \{x : \langle v, x \rangle = c_B\}$. Our claim is that $\forall a \in A, \langle a, v \rangle \leq c_A$ and $\forall b \in B, \langle b, v \rangle \geq c_B$. Suppose there is some $a \in A$ such that $\langle a, v \rangle > c_A$, then let a' be the foot of perpendicular from b_0 to the line segment $[a_0, a]$. Since $\langle a, v \rangle > \langle a_0, v \rangle$ implies that $\langle a - a_0, v \rangle > 0$, it is apparent by the properties of the scalar product of two vectors in euclidean space that the angle between $a - a_0$ and v must be less than $\frac{\pi}{2}$. Thus the side $[a_0, a]$ of the triangle a_0, b_0, a must contain the point a'. Since A is convex and thus the line segment $[a_0, a]$ must be contained in A, a' is inside A. By considering that $[a_0, b_0]$ forms the hypotenuse of a triangle with $[a', b_0]$ as one of it's sides, it is

clear that a' is closer to b_0 than a_0, a contradiction. Similar argument applies to B.

□

Assuming that we have two sets of points in R^n, each of which are closed, connected and convex, we can see that MPNs can be used as a tool to classify which set any given point from A or B is contained in. The same guarantee can however not be made for points in non-convex sets. By first generalizing the MPN into an *artificial neuron*:

Definition 1.10. An *artificial neuron*(AN) is a function $f(X) = \alpha(W(X + B)^\top)$ where $X = (x_1, \ldots\ x_n)$, $W = (w_1, \ldots\ w_n)$, $B = (b_1, \ldots\ b_n)$, $X, W, B \in \mathbb{R}^n$, $n \in \mathbb{N}$ and α is a real valued function. The function α is known as the *activation function* of f.

and defining a tuple of ANs as a *perceptron*:

Definition 1.11. A perceptron is a function $f(X) = (m_1(X), \ldots m_l(X))$ on $X = (x_1, \ldots\ x_n) \in \mathbb{R}^n$ where each m_i for $1 \leq i \leq l$ is an artificial neuron

We can see that by composing perceptrons, it is in some cases possible to approximately classify points from two sets even if they are not convex. Using the two dimensional example below:

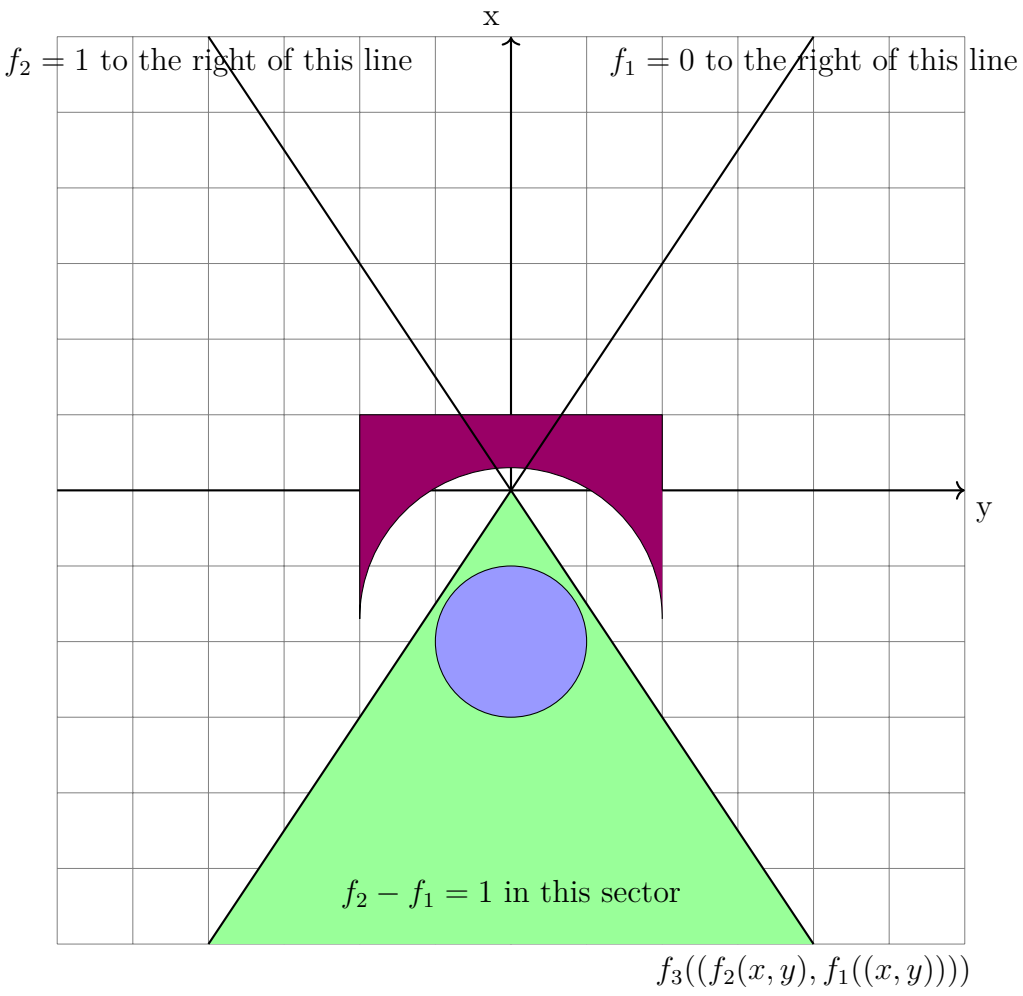


Figure 2: Non-convex set and lines between composed MPN regions

where f_1,f_2 are MPNs with the weights $(-3, 2)$, $(3, 2)$ respectively and f_3 an AN with the weights $(1, -1)$ and following activation function:

$$\alpha(x) = \begin{cases} 1 & \text{if} \quad x \geq 1 \\ 0 & \text{if} \quad x < 1 \end{cases}$$

we can see that any point in the circular set would be classified as 1, and any point in the non-convex set would be classified as 0. A composition of perceptrons is referred to as a *multi-layer perceptron*:

Definition 1.12. A *multi-layer perceptron*(MLP) is a function $f(X) = L_l(L_{l-1}(\dots(L_1(X))$ on $X = (x_1, \dots\ x_n) \in \mathbb{R}^n$ where each L_i for $1 < i \leq l$ is a perceptron.

1.3 Deep neural networks

Deep neural networks[GBA16] are computer algorithms designed for processing and modeling numeric data. They are fundamentally artificial neural networks, ANNs. The most general definition of an artificial neural network is that of a computer algorithm that at some point in it's execution simulates a perceptron. It may from this be apparent that in practice the perceptron being simulated is often a small part of the algorithm as a whole. They are referred to as deep because the perceptrons being simulated often come in dimensions far exceeding what was feasible at the time of the invention of the MPN. For instance, a deep neural network named PEGASUS[ZZSL19] created by a research team at Google for automatic summarizing of text contains about 560 million elements across its weights and bias matrices. And this model is a relatively small model when compared to models such as GPT-3 whose weights and bias matrices contain over a hundred billion elements in total. Deep neural networks are commonly represented visually as directed graphs. The vertices are often referred to as nodes in this case. The input data is represented by the nodes at the base of the graph and each non-input node represents an artificial neuron. Each set of neurons at a specific distance from the input nodes are known as *hidden layers*. It is through the activation functions that the data is processed and passed between neurons and their respective layers. Activation functions can be seen as loosely inspired by the synapses in the human brain[GBA16, p.13], regulating both when a signal is to be passed between neurons and with which strength. A more tangible example of an MLP $f(X)$ with 5 input nodes, 2 hidden layers with 10 nodes with sigmoidal activation functions f_1, f_2, f_3 and a single sigmoidal output node, could be represented as the represented by:

$$X = X_{(0)} = [x_1, x_2, x_3, x_4, x_5]^\top \in \mathbb{R}^5,$$

$$W_1, \in \mathcal{M}_{10\times 5}(\mathbb{R}), \quad W_2 \in \mathcal{M}_{10\times 10}(\mathbb{R}), \quad W_3 \in \mathcal{M}_{1\times 10}(\mathbb{R}),$$

$$B_1, B_2 \in \mathcal{M}_{1\times 10}(\mathbb{R}), \quad B_3 \in \mathbb{R},$$

$$X_{(i)} = f_i(W_i X_{i-1} + B_i) \quad \text{for} \quad i \in \{1, 2, 3\}.$$

$$f(X) = f_3(W_3(f_2(W_2(f_1(W_1(X + B_1)^\top) + B_2)^\top) + B_3)^\top).$$

The matrices W_i and B_i are the weights and biases, respectively, of the MLP. The sigmoid function, represented explicitly as:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

is applied elementwise on matrices and vectors. In this example, each element in the vectors X_i can be seen as the nodes of the MLP, and each vector as the individual layers of the MLP. If we were to remove the sigmoid function from the MLP, we would see that any individual node of the first hidden layer could be described through the familiar expression of a two-dimensional real line:

$$y \in X_1 \implies y = wx + b, \quad w \in W_1, x \in X_0, b \in B_1.$$

The same property holds for the other layers aswell, albeit in higher dimensions. The sigmoid function is for this reason known as a *non-linearity function*. Given that the output of the neural network is a number in the interval $(0, 1)$, it would be most appropriately deployed towards some form of binary classification problem for a set of classes C_1 and C_2, as described in section 1.1. For instance guessing if a point is contained in a poorly defined region of 5-dimensional euclidean space or, somewhat equivalently in theory, deciding if a loan applicant of a given income level, age, account balance, net worth and date of birth will complete all their payments in a timely manner. Since this is an issue of binary classification, we can see that by Theorem 1.9, it is possible to automate the decision making using an artificial neural network with appropriate weights and biases. Assuming that the two classes of applicants each lie in compact, disjoint and convex sets. Even with such a non-trivial assumption, the appropriate weights and biases are difficult to know beforehand, and become more and more complicated to compute symbolically as the amount of variables considered, size of each dataset and complexity of the task increases. The most common approach is therefore to initialize the weights and biases semi-randomly, and iteratively adjust them based on sample data. Artificial neural networks can have their weights and biases adjusted through a process commonly referred to as *training*. Given a set of samples X from an input space, their corresponding samples Y from the desired output space and a distance function d for elements in the output space it is possible to, for some artificial neural network F, optimize the weights of F in order to minimize the distance between $F(X) = \hat{Y}$ and Y. In the case of loan applications, X could contain information about previously

approved loan applicants and Y their respective payment adherence represented by 1 if they followed their payments and 0 if they didn't. The distance function d could for example be addition or subtraction. One source of inspiration for distance functions can be found in the field of statistics. The *Mean Squared Error*, or MSE for short, is a sort of averaging of distances between predicted and observed values. For a set of n pairs of observed values Y_i and predicted values $\hat{Y}_i$, it is defined as:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

To provide an explicit example of training of a neural network, we can assume that d is the MSE on a single pair of observed an predicted values:

$$d(Y, \hat{Y}) = (Y - \hat{Y})^2$$

and that F is the MLP exemplified earlier. The predicted value is in this case the value of F. We also have a sample data set X that consists of 100 randomly sampled points inside and outside a 5 dimensional unit sphere centered at the origin of radius 10, including the point:

$$\frac{1}{\sqrt{5}}[1, 1, 1, 1, 1].$$

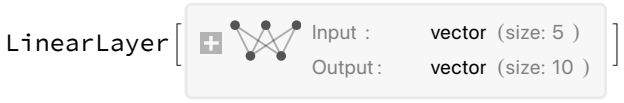
If we label points inside the sphere with 1 and points outside it with 0, then we can describe the loss for the point above with the following expression:

$$d(1, F([1, 1, 1, 1, 1])) = (1 - F([1, 1, 1, 1, 1]))^2.$$

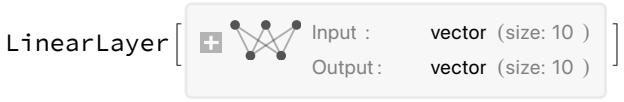
By randomly initializing the weights of F and setting its bias vectors as zero valued, we can compute it's value on $[1, 1, 1, 1, 1]$ using the following mathematica code:

9]:=
```
L1 = NetInitialize@LinearLayer[10, "Input" → 5]
L2 = NetInitialize@LinearLayer[10, "Input" → 10]
L3 = NetInitialize@LinearLayer[1, "Input" → 10]

NetExtract[L1, "Weights"] // Normal
NetExtract[L2, "Weights"] // Normal
NetExtract[L3, "Weights"] // Normal

NetExtract[L1, "Biases"] // Normal
NetExtract[L2, "Biases"] // Normal
NetExtract[L3, "Biases"] // Normal

F = NetChain[{L1, ElementwiseLayer[LogisticSigmoid], L2,
    ElementwiseLayer[LogisticSigmoid], L3, ElementwiseLayer[LogisticSigmoid]},
  "Input" → 5, "Output" → NetDecoder["Scalar"]]

F[{1, 1, 1, 1, 1}]

Export["MLPdemo.pdf", EvaluationNotebook[]]
```

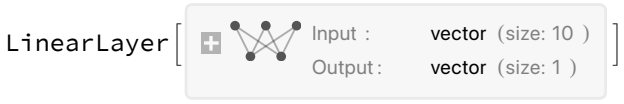
59]=



60]=



61]=



62]=

```
{{-0.227335, -0.0315741, -0.712812, 0.687186, 1.19765},
 {-0.588905, -0.489504, 0.0639282, 0.0465843, -0.748237},
 {0.0159885, 0.0761096, 0.418003, -0.369785, -0.0406799},
 {0.462154, -0.162575, 0.0843404, -0.0608847, -0.0375435},
 {0.249559, -0.966555, 0.004463, -0.380593, -0.405908},
 {-0.447447, 0.219026, 0.171084, -0.201943, 0.510055},
 {0.0658155, 0.306806, 0.572441, -0.541783, -0.259413},
 {-0.539585, 0.249051, 0.0122437, -0.185271, 0.369974},
 {0.0435727, -0.406934, 0.441859, 0.0714822, -0.328253},
 {-0.293133, -0.199171, 0.163789, -0.114177, 0.111943}}
```

63]=

```
{{-0.16075, -0.0223263, -0.504034, 0.485914,
  0.846864, -0.416418, -0.346132, 0.0452041, 0.03294, -0.529084},
 {0.0113056, 0.0538176, 0.295572, -0.261477, -0.028765,
  0.326793, -0.114958, 0.0596377, -0.043052, -0.0265473},
 {0.176465, -0.683457, 0.00315582, -0.26912, -0.28702, -0.316393, 0.154875,
  0.120975, -0.142795, 0.360663}, {0.0465386, 0.216944, 0.404777, -0.383099,
  -0.183433, -0.381545, 0.176106, 0.00865758, -0.131006, 0.261611},
 {0.0308106, -0.287745, 0.312442, 0.0505455, -0.23211,
  -0.207276, -0.140835, 0.115816, -0.0807354, 0.0791557},
 {0.104774, 0.0995118, 0.542574, -0.04669, 0.119367, 0.163544, 0.00857594,
  0.14041, 0.541274, 0.166406}, {0.193987, 0.185644, 0.134926, -0.278123,
  0.00645927, 0.0578803, 0.699784, -0.000506443, -0.115662, -0.442993},
 {-0.364381, -0.154783, -0.0643412, 0.0395997, -0.334149,
  -0.0114371, -0.42682, 0.0838717, 0.0572576, -0.0541785},
 {-0.143424, 0.0608095, 0.093548, 0.0907857, -0.340566, -0.183062,
  0.379418, 0.643919, -0.3876, 0.508179}, {0.18519, -0.357849, 0.178986,
  -0.0694283, -0.266553, 0.0359188, 0.288282, -0.732395, 0.471446, 0.395521}}
```

64]=

```
{{-0.16075, -0.0223263, -0.504034, 0.485914,
  0.846864, -0.416418, -0.346132, 0.0452041, 0.03294, -0.529084}}
```

65]=

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

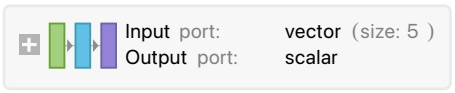
66]=

```
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}
```

67]=

```
{0.}
```

68]=

NetChain[]

69]=

```
0.401506
```

We can see in the code that the layers of the MLP is represented by the variables $L1$, $L2$, $L3$. Using the Mathematica function NetExtract, we can view the randomized weights and biases of the network, shown as the nested and single lists respectively in the code printout. The layers are combined using the Mathematica function NetChain, allowing the activation function of each layer to be defined aswell. Finally, at the end of the printout, the value of the MLP on the point $(1, 1, 1, 1, 1)$ is shown. Thus with the given sample datapoint and model parameters, the loss evaluates to:

$$(1 - F([1, 1, 1, 1, 1]))^2 = (1 - 0.401506)^2 \sim 0.3582.$$

Searching for local minimums to this expression with regards to the weight and bias parameters of F is what is meant by "training" an MLP. By findinng a local minimum to this expression, we make the neural network model our sample data more closely, in the hopes of it generalizing to the set of data it will process when solving it's intended task.

Since MLPs are a composition of differentiable functions, it is clear that MLPs must be differentiable. Assuming that the set of data it is intended to model is convex, then the tools from the theory of convex optimization are available to use for training MLPs. One such tool that is commonly used for finding local minimums of MLPs is known as *gradient descent*:

Definition 1.13. Let $x \in \mathbb{R}^d$ and let $\gamma > 0$ be known as *step size*. The *Gradient Descent*(GD) algorithm defines a sequence $(x^t)_{t\in\mathbb{N}}$ satisfying

$$x^{t+1} = x^t - \gamma \nabla \mathbf{f}(x^t).$$

It is however not necessarily evident that gradient descent ever arrives at a local minimum for a given MLP. For the case of MLPs on compact sets consisting of convex and smooth activation functions, it can be proved that it does. We first define a *lipschitz constant*:

Definition 1.14. A *lipschitz constant* of a function $f : X \to Y$ with metric spaces (X, d_X), (Y, d_Y) is a real number L such that for all $x_1, x_2 \in X$:

$$\frac{d_Y(f(x_1), f(x_2)}{d_X(x_1, x_2)} \leq L.$$

.

We will be using this with $d(x,y) = ||x-y||^2$ for $x, y \in \mathbb{R}^n$. We introduce the notion of μ-strong convexity:

Definition 1.15. Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, and $\mu > 0$. We say that f is μ-strongly convex if, for every $x, y \in \mathbb{R}^n$, and every $t \in [0,1]$ we have that

$$\mu \frac{t(1-t)}{2}||x-y||^2 + f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

We say that μ is the strong convexity constant of f.

. An equivalent definition is by lemma 2.14 of [CH23]:

Definition 1.16. If a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is μ-strongly convex then

$$\forall x, y \in \mathbb{R}^n : \quad f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}||y-x||^2.$$

. We will define argmin f for some function f as the set of elements in its domain on which f attains its minimal value. We introduce lemma 2.18 of [CH23]:

Proposition 1.17. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable, and $\mu > 0$. If f is μ-strongly convex, then f is bounded from below, and for all $x \in \mathbb{R}^n$*

$$f(x) - inf\ f \leq \frac{1}{2\mu}||\nabla f(x)||^2.$$

Proof. As f is continuous and strongly convex, argmin f contains a unique element. Denote it by x^*. Thus $f(x^*) = \inf f$. Multiplying the inequality in definition 1.16 by minus one and substituting $y = x^*$, we have that

$$f(x) - f(x^*) \leq \langle \nabla f(x), x - x^* \rangle - \frac{\mu}{2}||x^* - x||^2$$

$$= -\frac{1}{2}||\sqrt{}(x - x^*) - \frac{1}{\sqrt{}}\nabla f(x)||^2 + \frac{1}{2\mu}||\nabla f(x)||^2$$

$$\leq \frac{1}{2\mu}||\nabla f(x)||^2.$$

□

We can now prove the following theorem:

Theorem 1.18. *Assume that $f : U \to \mathbb{R}$ is a differentiable function on the compact subset $U \subset \mathbb{R}^d$ with argmin $f \neq \emptyset$ that is μ-strongly convex and smooth, with L*

being some Lipschitz constant of its gradient such that Ł $\geq \mu > 0$. *Let* $(x^t)_{t\in\mathbb{N}}$ *be the sequence of iterates generated by the (GD) algorithm, with a step size satisfying* $0 < \gamma \leq \frac{1}{L}$. *Then, for* $x^* = argmin\ f$ *and for all* $t \in \mathbb{N}$*:*

$$||x^{t+1} - x^*||^2 \leq (1 - \gamma\mu)^{t+1}||x^0 - x^*||^2$$

Proof. By the (GD) algorithm:

$$||x^{t+1} - x^*||^2 = ||x^t - x^* - \gamma\nabla f(x^t)||^2$$

$$= ||x^t - x^*||^2 - 2\gamma\langle\nabla f(x^t), x^t - x^*\rangle + \gamma^2||\nabla f(x^t)||^2.$$

By definition 1.16 we can see that:

$$||x^t - x^*||^2 - 2\gamma\langle\nabla f(x^t), x^t - x^*\rangle + \gamma^2||\nabla f(x^t)||^2$$

$$\leq (1 - \gamma\mu)||x^t - x^*||^2 - 2\gamma(f(x^t) - \inf f) + \gamma^2||\nabla f(x^t)||^2.$$

Where inf f is the minimal value of f on U. Since f is smooth and U compact, ∇f must attain a maximum and minimum value on U. Thus by definition 1.16:

$$(1 - \gamma\mu)||x^t - x^*||^2 - 2\gamma(f(x^t) - \inf f) + \gamma^2||\nabla f(x^t)||^2.$$

$$\leq (1 - \gamma\mu)||x^t - x^*||^2 - 2\gamma(f(x^t) - \inf f) + 2\gamma^2 L(f(x^t) - \inf f$$

$$= (1 - \gamma\mu)||x^t - x^*||^2 - 2\gamma(1 - \gamma L)(f(x^t) - \inf f).$$

Since $\frac{1}{L} \geq \gamma$ we have that $-2\gamma(1 - \gamma L)$ is negative, and thus can be safely dropped to give

$$||x^{t+1} - x^*||^2 \leq (1 - \gamma\mu)||x^t - x^*||^2.$$

□

Applying gradient descent to training an MLP F relies on computing the gradient of the expression $d(Y, \hat{Y})$ with regards to the weights of F. Computing gradients becomes impractical to do symbolically for artificial neural networks with many layers and nodes. Thus it is often done iteratively, through a process known as *backpropagation*. The gradient is through this process computed layer by layer, backwards from the final layer of the network, using the multivariate chain rule. Given a neural network F and its distance function d, commonly known in machine

learning literature as its *loss function*, we may for a given sample $y_{train} \in Y$ and output $F(x_{train})$ for $x_{train} \in X$ compute a *loss* $d(y_{train}, F(x_{train}))$. Using a distance function such as the MSE, as defined in equation 1, it is possible to do this for the whole dataset at once. Finding the minimum of $d(1, F(\frac{1}{\sqrt{5}}[1,1,1,1,1]))$ using gradient descent and backpropagation involves the following computations:

$$W_i \to \nabla_{W_i} d(1, F(\frac{1}{\sqrt{5}}[1,1,1,1,1])) = \nabla_{W_i}(1 - F(\frac{1}{\sqrt{5}}[1,1,1,1,1]))^2,$$

$$B_i \to \nabla_{B_i} d(1, F(\frac{1}{\sqrt{5}}[1,1,1,1,1])) = \nabla_{B_i}(1 - F(\frac{1}{\sqrt{5}}[1,1,1,1,1]))^2.$$

As finding gradients in closed form may be difficult as the MLP grows to the size of hundreds of layers and thousands of nodes, and the directional derivative relies upon the gradient of d, backpropagation may be used to compute the gradient and the directional derivatives of d numerically. An algorithm for doing so can be described using the following pseudocode:

Algorithm 1 Backpropagation

After the forward computation, the gradient on the output layer is computed:

$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} d(y, \hat{y})$

for $k = l, l-1 ... 1$, **do**

▷ Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)})$

▷ Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{B_k} J = g + \lambda \nabla_{B_k} \Omega(\theta)$

$\nabla_{W_k} J = g h^{(k-1)\top} + \lambda \nabla_{W_k} \Omega(\theta)$

▷ Propagate the gradients w.r.t. the next lower-level hidden layer's activations: $g \leftarrow \nabla_{h^{(k-1)}} J = W_k^\top g$

end for=0

also found at [GBA16, p.213]. Stepping through the code using the prior described sample data and MLP as input, and assuming no regularization is done, we can explicitly show the computations involved in one backward pass as the following:

$$\hat{y} = F([1,1,1,1,1]), y = 1$$

$$\implies g = \nabla_{\hat{y}} d(y, \hat{y}) = (\frac{\partial d(y, \hat{y})}{\partial y}, \frac{\partial d(y, \hat{y})}{\partial \hat{y}}) \cdot \hat{y}$$

$$= (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y}.$$

$$k = 3$$

$$\implies a^{(k)} = a^{(3)} = W_3 X_2 + B_3.$$

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$\implies f'(x) = -\frac{e^{-x}}{(1 + e^{-x})^2} = f(x)(f(x) - 1)$$

$$\implies g \leftarrow (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)$$

$$\implies \nabla_{B_3} J = (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1),$$

$$\nabla_{W_3} J = ((2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) f(a^{(2)})^\top,$$

$$g \leftarrow W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1).$$

$$k = 2$$

$$\implies a^{(k)} = a^{(2)} = W_2 X_1 + B_2.$$

$$g \leftarrow (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)$$

$$\implies \nabla_{B_2} J = (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1),$$

$$\nabla_{W_2} J = ((W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)) f(a^{(2)})^\top,$$

$$g \leftarrow W_2^\top (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1).$$

$$k = 1$$

$$\implies a^{(k)} = a^{(1)} = W_1 X_0 + B_1.$$

$$g \leftarrow (W_2^\top (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)) \odot f(a^{(1)})(f(a^{(1)}) - 1))$$

$$\implies \nabla_{B_1} J =$$

$$(W_2^\top (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)) \odot f(a^{(1)})(f(a^{(1)}) - 1)),$$

$$\nabla_{W_1} J =$$

$$((W_2^\top (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)) \odot f(a^{(1)})(f(a^{(1)}) - 1)) f(a^{(0)})^\top,$$

$$g \leftarrow W_1^\top ((W_2^\top (W_3^\top (2y - 2\hat{y}, 2\hat{y} - 2y) \cdot \hat{y} \odot f(a^{(3)})(f(a^{(3)}) - 1)) \odot f(a^{(2)})(f(a^{(2)}) - 1)) \odot f(a^{(1)})(f(a^{(1)}) - 1))).$$

The only part missing from the computations above are the initial values of the weights and biases. Selecting these properly is not a trivial problem, and is often an important part of ensuring that a neural network actually converges during training[GBA16, p.301]. In most practical applications, where data sets and the dimensionality of it's datapoints are large, it becomes impractical from a computational standpoint to do (GD) computations across entire training sets. *Stochastic Gradient Descent* (SGD) allows for the training of an MLP with a fewer computations by using estimates of its gradient instead of the gradient itself:

Definition 1.19. Consider a function $f : \mathbb{R}^d \to \mathbb{R}$:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

where the functions $f_i : \mathbb{R}^d \to \mathbb{R}$ are smooth and convex. Let $x^0 \in \mathbb{R}^d$, and let $\gamma_t > 0$ be a sequence of step sizes. The Stochastic Gradient Descent (SGD) algorithm is given by the iterates $(x^t)_{t\in\mathbb{N}}$ where:

$$i_t \in \{1, ...n\} \quad \text{Sampled with probability} \quad \frac{1}{n},$$

$$x^{t+1} = x^t - \gamma_t \nabla f_{i_t}(x^t).$$

Theorem 5.5 of [CH23] also proves that (SGD) converges under conditions similar to those for (GD):

Theorem 1.20. *Let f be a sum of convex and smooth functions f_i on a compact set $U \subset \mathbb{R}^d$ under the assumptions of (SGD), where L_{max} is the the largest Lipschits constant for some set of Lipschitz constants of the functions f_i. Consider $(x^t)_{t\in\mathbb{N}}$ a sequence generated by the (SGD) algorithm with a stepsize $\gamma_t > 0$.*

1. *If $\gamma_t = \gamma < \frac{1}{2L_{max}}$, then for every $t \geq 1$ we have that*

$$\mathbb{E}[f(\overline{x}^t) - f(x^*)] \leq \frac{||x^0 - x^*||^2}{2\gamma(1 - 2\gamma L_{max})}\frac{1}{t} + \frac{\gamma}{1 - 2\gamma L_{max}}\sigma_f^*$$

   where $\overline{x}^t = \frac{1}{t}\sum_{k=0}^{t-1} x^k$.

2. *If $\gamma_t = \gamma\sqrt{t+1}$ with $\gamma \leq \frac{1}{2L_{max}}$, then for every t we have that*

$$\mathbb{E}[f(\overline{x}^k) - f(x_*)] \leq \frac{||x^0 - x^*||^2}{2\gamma\sqrt{k}} + \frac{\gamma^2 \log(k)}{\gamma\sqrt{k}}\sigma_f^* = \mathcal{O}(\frac{\log(k)}{\sqrt{k}}).$$

1.4 The convolution layer

Successful training of a neural network for a given task is commonly known as *convergence*. How the distance function d is chosen and the sample datasets X and Y are constructed have an enormous impact on the conditions under which a neural network converges. For instance,describing the distance between the predicted number in an image of a number and the actual number depicted using subtraction may be enough for a relatively simple MLP to converge. Assuming that the a sample set consists of small, digitally drawn images of white numbers on a black background paired with the numbers written, and that all numbers it is expected to classify follow this pattern. An example of an MLP with a slightly different loss function that achieves 99% accuracy on the foundational MNIST dataset can be found at https://www.kaggle.com/code/devanshbesain/deep-mlp-on-mnist-dataset. It may however at the same time be impossible for any MLP at all to converge using the same distance metric when trying to train it to automatically classify arbitrary images of numbers in real life, in any number of orientation, drawn on any number of surfaces or using any number of objects, in all kinds of illumination levels. One issue lies in the fact that MLPs process all parts of their input in largly the same way, thus the weights and biases necessary for classifying two seemingly similar images, e.g various rotations of the number 7, can differ significantly. Exploiting various spatial symmetries, for instance that numbers are numbers regardless of their orientation, when constructing a distance function, or a neural network more broadly, can however turn an intractable problem into a tractable one. *Convolutional neural networks* are such an example. A convolutional neural network is a neural network that employs convolution in place of general matrix multiplication in at least one of it's layers[GBA16, p.330]. Convolution is a method widely used in the context of signal processing, and can be viewed as a method of averaging two time continuous measurements $x(t)$ and $w(t)$, or signals, over some domain $D \in \mathbb{R}$ in the following manner:

$$s(t) = \int_D x(a)w(t-a) \quad da. \tag{2}$$

The operation is typically represented using an asterisk:

$$s(t) = (x * w)(t)$$

It is however not a true weighted average unless w is a valid probability density function[GBA16, p.331]. If w is a probability density function of D, then the convolution above corresponds with the expected value of $x(t)$ as a random variable with a probability distribution equal to D. For instance, for some square shaped greyscale image $I \in [255]^{r \times r} \subset \mathbb{Z}^{r \times r}$ of resolution $r \in \mathbb{N}$, we may use $w(t) = \frac{1}{r^2}$ of the uniform distribution and see an example of this. As the input space for a given neural network used practically is most often finite, and most functions are defined as zero valued outside them, the integration step of the convolution operation can be assumed to be representable as a sum.

Another use case of convolution beyond signal processing is for image processing. For instance to detect the edges of an object in an image. If a black and white image consists of a written number, as shown below:

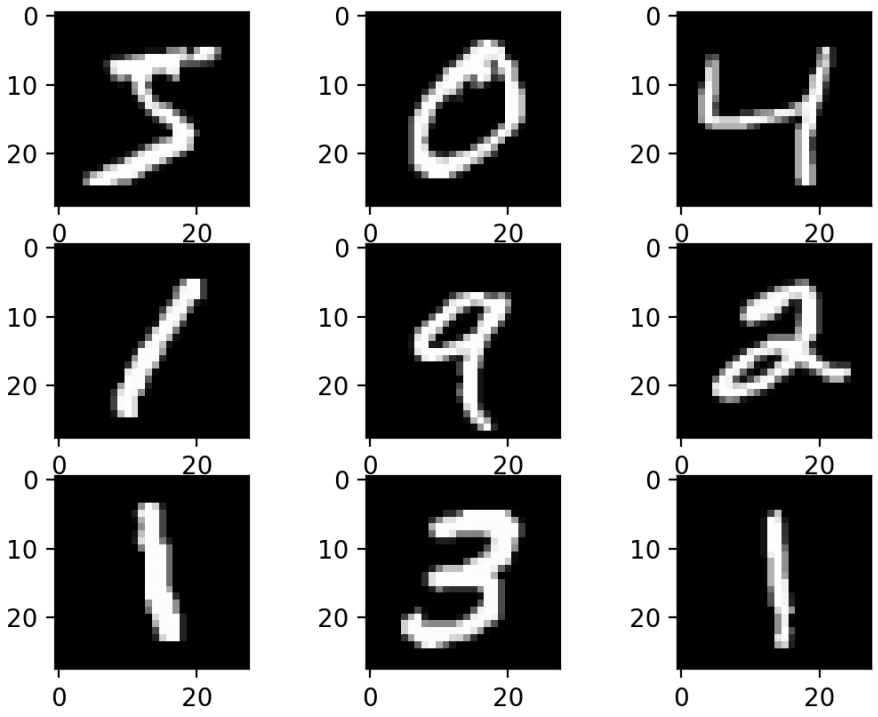


Figure 3: Excerpt from the MNIST dataset

It is reasonable to assume that the pixels on the edges of the written number may be characterized by a sharp increase/decrease in opacity. This increase/decrease can be detected using an appropriate *convolution matrix*. The convolution matrix

is multiplied peicewise with the matrix of pixels representing the image, allowing for only the edges to be visually identifiable afterwards. For instance, using the following matrix as a convolution matrix:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

and by considering that a triangular matrix represents an edge in an image:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 255 \\ 0 & 255 & 255 \\ 255 & 255 & 255 \end{bmatrix}$$

$$= \begin{bmatrix} -255 & -1020 & 510 \\ -1020 & 765 & 255 \\ 510 & 255 & 0 \end{bmatrix}$$

while a matrix without zeroes represents a part of an image without edges:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} 255 & 255 & 255 \\ 255 & 255 & 255 \\ 255 & 255 & 255 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

we can see how the outputs generated by the convolution matrix differs based on the presence of edges in an image. Using the ImageConvolve function in Mathematica, we can see some visual examples of applying the convolution matrix below:

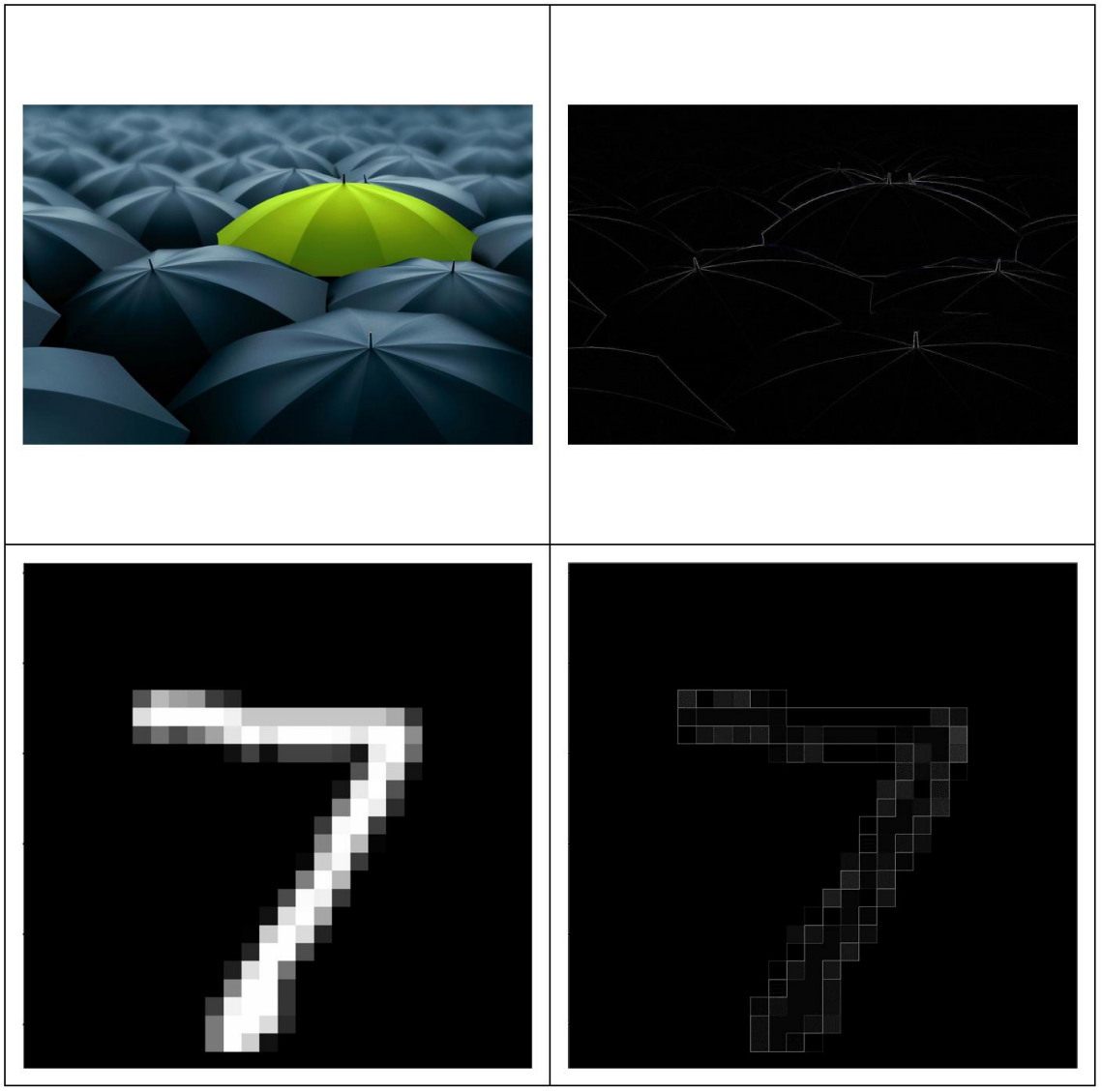


Figure 4: Images before and after processing using the convolution matrix

2

Group convolutional neural networks

A central notion when discussing disirable properties of neural networks is *equivariance*:

Definition 2.1. A function $F : X \to Y$ is *equivariant* to some group action by G on X and Y if:

$$\forall x \in X : F(gx) = gF(x).$$

Analogously to the previously described situation with arbitrary convex sets, we can now consider ways of classifying datasets that are equivariant to some group action. If we consider the datasets $C = C_- \cup C_+ \subset \mathbb{R}^2$ consisting of an infinite amount discs of radius 1 centered at $x = -3$ and $x = 3$ separated by a distance of 3 along the y-axis and the dataset A consisting of an infinite amount of similar discs centered at the origin of the same distance between each other. We can see that both sets are equivariant, in fact even *invariant*, to reflection about the y-axis. This can be seen as an action of Z_2. The datasets could not be classified using a finite amount of MPNs whose weights represent lines not equivariant to this action, in the form of e.g a zigzag pattern, but could be classified by using only two MPNs f_1, f_2 with their respective weights and biases being

$$W_1 = (1, 0), B_2 = (1.5, 0)$$

$$W_2 = (-1, 0), B_2 = (-1.5, 0)$$

as the lines they represent are equivariant to the action. We can see it visually below:

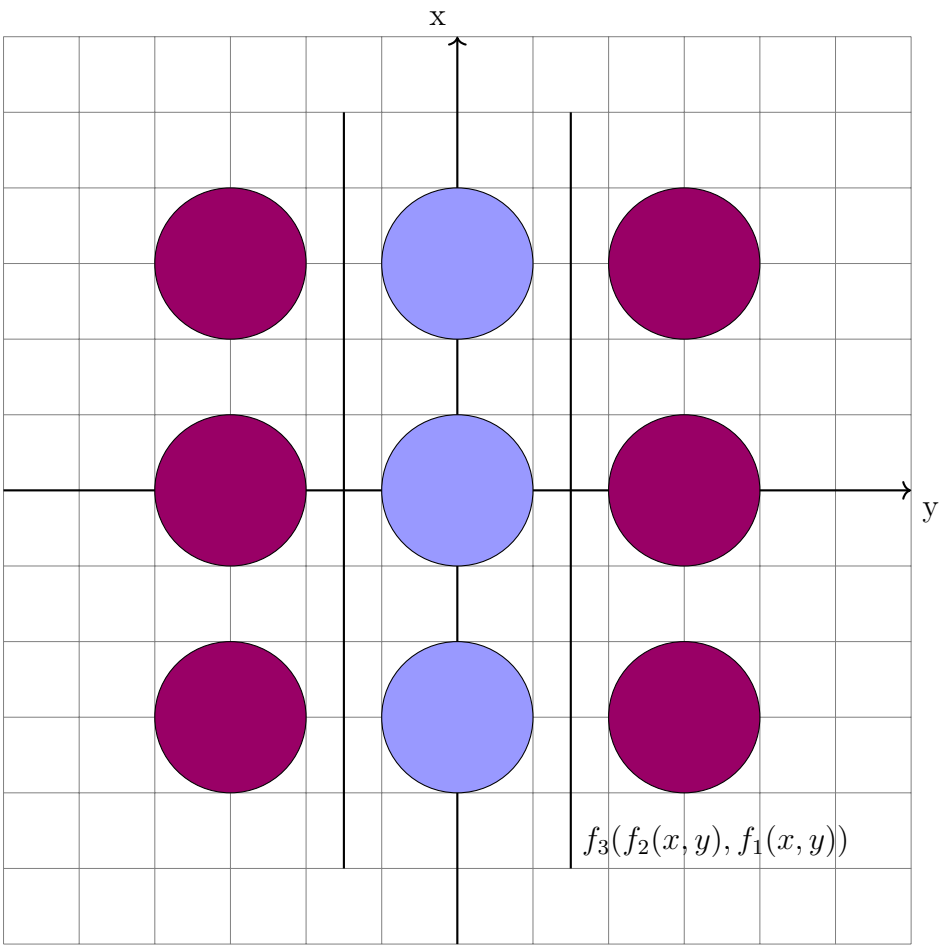


Figure 5: Sets and artificial neuron invariant to reflection

Where f_4 is an AN with bias $\mathbf{0}$, the weights $(1,1)$ and the following activation function:

$$\alpha(x) = \begin{cases} 1 & \text{if} \quad x \geq 2 \\ 0 & \text{if} \quad x < 2 \end{cases}$$

It can be shown that convolutional neural networks, most famously used to great success in the domain of object recognition[KSH12], are equivariant under spatial translation. Spatial translation of images can be represented by actions of $\mathbb{Z}/n_1\mathbb{Z}\times\mathbb{Z}/n_2\mathbb{Z}$, where n_1, n_2 represent the dimensions of the image being translated. They are a special case of what is known as *group convolutional neural networks*, for which the equivariant property will be proven in one direction below. There are however distinct operations beyond purely convolution that are necessary in practice, and that allow for the property of equivariance to be generalized beyond finite group actions and to new types of data entirely[RT18]. Specifically, the notion of G-convolution

and G-pooling, along with a nonlinearity layer.

To start with, it is useful to introduce some of the mathematical machinery necessary to describe equivariance more generally. An object known as a topological group that lies in the intersection of two areas, topology and algebra, is necessary. A useful introduction to general topology can be found at [TWT20]. First we define a topological space and it's topology:

Definition 2.2. A topology $\mathcal{T}$ of the topological space $(X, \mathcal{T})$ is a set of subsets of the set X that is closed under arbitrary unions and finite intersections. The empty set $\emptyset$ and X must also be contained in $\mathcal{T}$.

Elements of $\mathcal{T}$ are known as open sets, and their complements in X as closed sets. It is common to refer to X as the topological space itself if it's topology is not ambiguous. Let $[n] = \{1, 2, 3...n\}$ for $n \in \mathbb{N}$. Then we can see:

Proposition 2.3. *The set of all subsets of* $[n]$*, also known as the power set* $\mathcal{P}([n])$*, is a topology on* $[n]$ *known as the discrete topology.*

Proof. By convention, $[n]$ and the empty set $\emptyset$ are elements of $\mathcal{P}([n])$. As $\mathcal{P}([n])$ is finite, any set of unions or intersections can at most be finite. Assuming an index set I, we can see that for any family of subsets $(a_i)_{i \in I}$ of $[n]$ that

$$\bigcup_{i \in I} a_i \subseteq [n] \implies \bigcup_{i \in I} a_i \in \mathcal{P}([n])$$

and

$$\bigcap_{i \in I} a_i \subseteq [n] \implies \bigcap_{i \in I} a_i \in \mathcal{P}([n])$$

thus $\mathcal{P}([n])$ satisfies the definition of a topology on $[n]$. □

Much as with the vectors of a vector space, it may in some cases be quite cumbersome to give an explicit description of all open sets of a topological space. Therefore there is use in defining a *basis* of a topological space.

Definition 2.4. A *basis* $\mathcal{B}$ of a topological space X is a set such that every open set of X can be written as a union of members of $\mathcal{B}$.

For instance, in the case of the previously mentioned topology of $[n]$, it can be seen that the so called *singleton sets* $\{i\}$ for $i \in [n]$ forms a basis for $\mathcal{P}([n])$. A well

known topology on the set of real numbers $\mathbb{R}$ is known as the *euclidean topology*. It has the set of open intervals $\{(r_1, r_2) | r_1 < r_2 \wedge r_i \in \mathbb{R}\}$ as its basis. There are multiple properties of functions between topological spaces that signify that they in some sense preserve the topological structure of the respective spaces. One such property is known as continuity:

Definition 2.5. A function $f : X_1 \to X_2$ between topological spaces is *continuous* if the inverse image $f^{-1}(U)$ of an open set $U \subset X_1$ is open in X_2.

Continuity of a real function $f : \mathbb{R} \to \mathbb{R}$ is usually defined as the property that for any $x_0 \in \mathbb{R}$ and $\epsilon > 0$ there exists some $\delta > 0$ such that

$$|x - x_0| < \delta \implies |f(x) - f(x_0)| < \epsilon.$$

By the following propositions:

Proposition 2.6. *A function $f : \mathbb{R} \to \mathbb{R}$ is continuous if and only if for each $a \in \mathbb{R}$ and each open set U containing $f(a)$, there exists and open set V containing a such that $f(V) \subseteq U$.*

Proof. Assume that f is continuous. Let $a \in \mathbb{R}$ and let U be any open set containing $f(a)$. Then there exist real numbers c and d such that $f(a) \in (c, d) \subseteq U$. Put ϵ equal to the smaller of the two numbers $d - f(a)$ and $f(a) - c$, so that

$$(f(a) - \epsilon, f(a) + \epsilon) \subseteq U.$$

As the mapping f is continuous there exists a $\delta > 0$ such that $f(x) \in (f(a) - \epsilon, f(a) + \epsilon)$ for all $x \in (a - \delta, a + \delta)$. Let V be the open set $(a - \delta, a + \delta)$. Then $a \in V$ and $f(V) \subseteq U$, as required. Conversely assume that for each $a \in \mathbb{R}$ and each open set U containing $f(a)$ there exists an open set V containing a such that $f(V) \subseteq U$. We have to show that f is continuous. Let $a \in \mathbb{R}$ and ϵ be any positive real number. Put $U = (f(a) - \epsilon, f(a) + \epsilon)$. So U is an open set containing $f(a)$. Therefore there exists an open set V containing a such that $f(V) \subseteq U$. As V is an open set containing a, there exist real numbers c and d such that $a \in (c, d) \subseteq V$. Put δ equal to the smaller of the two numbers $d - a$ and $a - c$, so that $(a - \delta, a + \delta) \subseteq V$. Then for all $x \in (a - \delta, a + \delta)$, $f(x) \in f(V) \subseteq U$, as required. So f is continuous. □

Proposition 2.7. *Let f be a mapping of a topological space $(X, \mathcal{T})$ into a topological space $(Y, \mathcal{T}')$. Then the following two conditions are equivalent:*

1. *for each* $U \in \mathcal{T}'$, $f^{-1}(U) \in \mathcal{T}$

2. *for each* $a \in X$ *and each* $U \in \mathcal{T}'$ *with* $f(a) \in U$, *there exists a* $V \in \mathcal{T}$ *such that* $a \in V$ *and* $f(V) \subseteq U$.

Proof. Assume that condition 1. is satisfied. Let $a \in X$ and $U \in \mathcal{T}'$ with $f(a) \in U$. Then $f^{-1}(U) \in \mathcal{T}$. Put $V = f^{-1}(U)$, and we have that $a \in V, V \in \mathcal{T}$, and $f(V) \subseteq U$. So condition 2. is satisfied. Conversely, assume that condition 2. is satisfied. Let $U \in \mathcal{T}'$. If $f^{-1}(U) = \emptyset$ then clearly $f^{-1}(U) \in \mathcal{T}$. If $f^{-1}(U) \neq \emptyset$, let $a \in f^{-1}(U)$. Then $f(a) \in U$. Therefore there exists a $V \in \mathcal{T}$ such that $a \in V$ and $f(V) \subseteq U$. So for each $a \in f^{-1}(U)$ there exists a $V \in \mathcal{T}$ such that $a \in V \subseteq f^{-1}(U)$. This implies that $f^{-1}(U) \in \mathcal{T}$. So condition 1. is satisfied. □

We can see that the inverse of any non-constant f on an open interval in $\mathbb{R}$ is an open interval in $\mathbb{R}$. Open intervals in $\mathbb{R}$ are open sets with regards to the euclidean topology. Since the inverse of a constant function on an open interval is either X or $\emptyset$, both open sets in the euclidean topology, continuity in the general topological sense with regards to the euclidean topology can be seen as a strict generalization of continuity as commonly defined for real functions. By first defining the product topology:

Definition 2.8. The *product topology* $\mathcal{T}_{X\times Y}$ on $X \times Y$ is defined as the topology having the set $\{U_1 \times U_2 \mid U_1 \in \mathcal{T}_X, U_2 \in \mathcal{T}_Y\}$ as it's basis, where $\mathcal{T}_X$,$\mathcal{T}_Y$ are the topologies on X and Y respectively.

we can now define a topological group:

Definition 2.9. A *topological group* is a group G with a topology and a continuous group operation, as seen as a map from the product topology $G \times G$ to G, and that has a continuous inverse function $i : g \to g^{-1}$.

It is worth noting that the topology constructed on $[n]$ where every subset is open works for any set, and is then also known as the *discrete topology.* One useful fact is that:

Proposition 2.10. *An arbitrary group* G *can be made into a topological group via the discrete topology.*

Proof. Note that $\mathcal{T}_{G\times G}$ is also discrete, so all subsets are open. The inverse image U of an element g consists of all pairs (g_1, g_2) such that $g_1 g_2 = g$. As the inverse

image of a subset $H \subset G$ is the union of the inverse images of it's elements, the group operations is shown to be continuous. The inverse operation $i : g \to g^{-1}$ is necessarily continuous as the axioms of a group forces inverse elements for each element to be part of the same group, thus $i^{-1}(H) \subset G$ is an open set. □

The *metric topology* on a metric space(Definition 1.2) is defined as the topology with the set of open balls $B_{a,r} = \{x \in X | d(a, x) < r\}$ for points in $a \in X$ as its basis. Using the standard notion of distance in $\mathbb{R}^n$, referred to in topological settings as the *euclidean metric*, it can be seen that this basis is the same as the basis for the euclidean topology given earlier for $\mathbb{R}$, but now generalized to arbitrary dimensions. As the set of real numbers has been shown above to be a topological space, it can be worth asking if real analysis can be done on topological spaces. To a certain extent, and under specific circumstances, it can. Using the concept of a homeomorphism:

Definition 2.11. A *homeomorphism* $f : X_1 \to X_2$ is a bijective continuous function between topological spaces whose inverse is also continuous.

one can define a *manifold*:

Definition 2.12. A *manifold* is a topological space X for which all elements $x \in X$ there exists in an open set $x \in U \subseteq X$ that is homeomorphic to an open subset of euclidean n-space.

For any function f defined on a topological space X that is a manifold, composing it with the homeomorphisms from X to $\mathbb{R}^n$ would allow for integration and derivation in $\mathbb{R}^n$. There are however compatibility conditions that need to be in place for these operations to be well-defined. Thus it is necessary to define an *atlas* on a manifold:

Definition 2.13. An *atlas* A of a manifold X is a set of pairs (φ, U) of maps $\varphi : U \to \mathbb{R}^n$ and open sets U that cover X. On $U_1 \cap U_2$ we have two maps $\varphi_1 : U_1 \cap U_2 \to \mathbb{R}^n$, $\varphi_2 : U_1 \cap U_2 \to \mathbb{R}^n$. The *transition map* $\sqcup_{21} : \varphi_1(U_1 \cap U_2) \to \varphi_2(U_1 \cap U_2)$ is defined as $\varphi_2 \circ \varphi_1^{-1}$, and is thus a map from an open subset of $\mathbb{R}\acute{I}n$ to an open subset of $\mathbb{R}^n$. Transition maps are also homeomorphisms.

The elements of A are known as *charts*. A manifold can thus be said to be differentiable if the transition maps of it's charts are differentiable. A differentiable manifold where the transition maps of its charts are C^∞ is known as a *smooth* manifold. A good source for further information on differentiable manifolds is [MS99]. Combining the notion of a smooth manifold with that of a topological group, we arrive at what is known as a Lie group:

Definition 2.14. A Lie group is a topological group that is also a smooth manifold, whose group operation and inverse operation are both smooth.

A well-known example of a Lie group is the subset $SO(3) \subset GL_3(\mathbb{R})$ consisting of orthogonal matrices of determinant 1.

Proposition 2.15. *The group $SO(3)$ is a Lie group*

Proof. Multiplication and inversion of matrices are given by a number of polynomials in several variables: Let $X = (x_{ij])}$ and $Y = (y_{ij})$, then $XY = (\sum x_{ik}y_{kj})$ and the entries are polynomials $\sum x_{ik}y_{kj}$ that are clearly smooth. There is always an open subset of points for the set of solutions M to some set of equations, as in the definition of $SO(3)$, around M which is a manifold. The group action on $M = SO(3)$ ensures that any neighbourhood of such a point can be moved anywhere on the group, hence all points of the group have a neighbourhood that is a manifold. The transition maps for a smooth structure on $SO(3)$ can be constructed by composing the inversion function with the functions from any set of homeomorphisms that make $SO(3)$ a manifold. □

2.1 G-convolution layer

Black and white images, also known as greyscale images, can be represented as two dimensional grids of *pixels*. Each pixel in the grid is an integer in the interval $[0, 255]$, where a white pixel has the value 255, and a black pixel the value 0. The translation of a square greyscale image I in the x or y-axis can be viewed as a group action of the additive quotient group $\mathbb{Z}/256\mathbb{Z}$. We can capture the cyclical nature of image translations by deforming I into a cylinder and then a torus. Formally this can be done by sequentially constructing quotient spaces using the respective equivalences $(r, t) \sim (0, t)$ and $(t, r) \sim (t, 0)$ where r is the resolution of I and $0 \leq t \leq r$. The group action of $\mathbb{Z}/256\mathbb{Z}$ on I naturally induces a group action on the set of functions defined on I. In order to incorporate both the translation in the x and y-axis at the same time, we can instead look at the action of the group $G = (\mathbb{Z}/256\mathbb{Z}) \bigoplus (\mathbb{Z}/256\mathbb{Z})$ defined by $(g_1, g_2)(x, y) = (g_1x, g_2y)$, where g_ix, g_jy are calculated with regards to the group action from before. It can be shown that convolution is equivariant to this group action. In order to define convolution on compact groups, we need to generalize integration to compact groups. One way to of doing so is to define integration with regards to a *measure* on a set and then find

a suitable measure for compact groups. Before doing so, it is necessary to define the *σ-algebra*:

Definition 2.16. A *σ-algebra* on a set X is a set of subsets of X that are closed under complement, countable unions and countable intersections.

An example of a σ-algebra on an arbitrary topological group G is the σ-algebra generated by the open subsets of G, also known as a *Borel algebra*:

Definition 2.17. The σ-algebra generated by a family F of subsets of a space X is the set that contains all subsets of X that can be formed from elements of F using a countable number of unions, intersections and complement operations.

The elements of a Borel algebra are known as *Borel sets*. We can now define a measure:

Definition 2.18. A *measure* is a function $\mu : \Sigma \to \mathbb{R}^+ \cup \{+\infty\}$ for some σ-algebra Σ that is zero on the empty set and satisfies

$$\mu(\bigcup_{k=1}^{\infty} E_k) = \sum_{k=1}^{\infty} \mu(E_k)$$

for all countable collections $\{E_k\}_{k=1}^{\infty}$ of pairwise disjoint sets in Σ

An example of a measure on the power set $\mathcal{P}(X)$ of an arbitrary set X is the function:

$$\mu(A) = \begin{cases} |A| & \text{if} \quad A \quad \text{is finite} \\ +\infty & \text{if} \quad A \quad \text{is infinite} \end{cases}$$

commonly known as the *counting measure*. A measure defined on the Borel sets of a topological space is known as a *Borel measure*. We also need something known as a *haar measure*:

Definition 2.19. A (left) *haar measure* of a compact group G is a non-trivial measure on its Borel sets that satisfies the following criteria:

- Left-translation invariance: $\mu(gS) = \mu(S)$ for $g \in G$ and Borel sets $S \subseteq G$
- Finiteness on compact sets: $\mu(K) < \infty$ for compact $K \subseteq G$
- Outer regularity: $\mu(S) = \inf(\{\mu(U) : S \subseteq U, \quad U \quad \textit{open}\})$ for Borel sets $S \subseteq G$

- Inner regularity: $\mu(U) = \sup(\{\mu(K) : K \subseteq U, \quad K \quad compact\})$ for open sets $U \subseteq G$

Compact groups can be shown to always have a unique haar measure by what is known as Haar's theorem. Convolution of complex valued functions f, g on a compact group G can now be defined as follows:

$$(f * g)(u) = \int_G f(uv^{-1})g(v) \quad d\mu(v)$$

where μ is the Haar measure of G. Given that $(\mathbb{Z}/256\mathbb{Z}) \bigoplus (\mathbb{Z}/256\mathbb{Z})$ is finite, and thus necessarily a compact group under the discrete topology:

Proposition 2.20. *The counting measure is a haar measure on G*

Proof. Since the topology of G already contains all the Borel sets of G, and their cardinalites are unaffected by G acting on them, it is clear that the counting measure is left-translation invariant. Since G is finite, it is also clear that the measure is finite on all compact sets in G. Since all Borel sets in G are open sets in G, and vice versa, it is clear that $\mu(S) \in \{\mu(U) : S \subseteq U, \quad U \quad open\}$, and thus $\mu(S)$ must also be a greatest lower bound as S cannot be contained in a strict subset of itself. Since all subsets of G are compact by its finiteness, inner regularity follows by analogous reasoning. □

Since the counting measure of a discrete group satisfies the criteria of a Haar measure we can see that $\mu(v) = |[v]| = 1$. Group convolution of two functions $f : G \to \mathbb{R}$,$g : G \to \mathbb{R}$ using the counting measure implies

$$(f * g)(x) = \sum_{y \in G} f(xy^{-1})g(y)\mu(y)$$

$$= \sum_{y \in G} f(xy^{-1})g(y)$$

Thus we can see that group convolution over finite groups $H < \mathbb{R}$ with regards to the counting measure becomes regular convolution, as described by equation 2(page 22). This illustrates that group convolution, also known as G-convolution, can be seen as as a strict generalization of "regular" convolution.

2.2 G-CNN construction

For the purpose of constructing group convolutional neural networks, it may be easier to view MLPs in a way different from how it has been presented so far. By first defining:

Definition 2.21. Let X be an arbitrary set, and V a vector space. Then $L_V(X)$ contains the set of functions $f : X \to V$.

We can redefine MLPs as follows:

Definition 2.22. Let $X_0, ..., X_L$ be a sequence of index sets, $V_0, ..., V_L$ vector spaces, $\phi_1, ..., \phi_L$ linear maps

$$\phi_\ell : L_{V_{\ell-1}}(X_{\ell-1}) \to L_{V_\ell}(X_\ell),$$

and $\sigma_\ell : V_\ell \to V_\ell$ appropriate pointwise nonlinearities, such as the sigmoid function. The corresponding MLP is then a sequence of maps $f_0 \to f_1 \to f_2 \to ... \to f_L$, where $f_\ell(x) = \sigma_\ell(\phi_\ell(f_{\ell-1})(x))$.

In this definition, as opposed to definition 1.12, we treat an MLP as a sequence of functions rather than a series of function compositions. The index sets contain the neurons of each layer and the vector spaces represent the domain and codomains of the layers(assumed to be equivalent). The linear maps represent the operations applied to the outputs of each layer before passing it through an activation function and on to the next layer, in the case of definition 1.12 this has always been assumed to be matrix multiplication of weights and matrix addition of biases. By first generalizing convolution to quotient spaces of G:

Definition 2.23. Let G be a finite or countable group, X and Y be (left or right) quotient spaces of G, $f : X \to \mathbb{C}$, and $g : Y \to \mathbb{C}$. As X and Y are quotient spaces, there exist canonical maps $\eta_x : G \to X$ and $\eta_y : G \to Y$ that consist of reducing an element g modulo X and Y respectively. By defining $f \circ \eta_X = f \uparrow^G$ and $g \circ \eta_Y = g \uparrow^G$ we can then define the convolution of f with g as

$$(f * g)(u) = \sum_{v \in G} f \uparrow^G (uv^{-1}) g \uparrow^G (v), u \in G.$$

$f \uparrow^G$ and $g \uparrow^G$ are known as the *lifted* functions of f and g respectively. We may formally define a *G-Convolutional Neural Network* for some group G as follows:

Definition 2.24. Let G be a compact group and $\mathcal{N}$ an $L+1$ layer MLP in which the i'th index set is G/H_i for some subgroup H_i of G. We say that $\mathcal{N}$ is a G-convolutional neural network(G-CNN) if each of the linear maps $\phi_1 ... \phi_L$ in $\mathcal{N}$ is a generalized convolution as defined in Definition 2.23 of the form

$$\phi_\ell(f_{\ell-1}) = f_{\ell-1} * X_\ell$$

with some filter $X_\ell \in L_{V_{\ell-1} \times V_\ell}(H_{\ell-1} \backslash G / H_\ell)$.

By first defining G-equivariance with regards to MLPs:

Definition 2.25. Let $\mathcal{N}$ be an MLP as defined in Definition 2.22, and G be a group that acts on each index space $X_0, ..., X_L$. Let $\mathbb{T}_0, \mathbb{T}_1, ..., \mathbb{T}_L$ be the corresponding actions on $L_{V_0}(X_0), ..., L_{V_L}(X_L)$. We say that $\mathcal{N}$ is a G–equivariant feed-forward network if, when the inputs are transformed $f_0 \to \mathbb{T}_g^0(f_0)$ (for any $g \in G$), the activations of the other layers correspondingly transform as $f_\ell \to T_g^\ell(f_\ell)$.

We can see via theorem 1 of [RT18], that G-CNNs are equivariant to the action of G, although we only prove the forward implication:

Theorem 2.26. *Let G be a compact group and $\mathcal{N}$ be an $L+1$ layer MLP in which the ℓ'th index set is of the form $X_\ell = G/H_\ell$, where H_ℓ is some subgroup of G. Then $\mathcal{N}$ is equivariant to the action of G if and only if it is a G-CNN.*

Proof. The theorem can be proved in the forward direction as follows: Assume that we translate $f_{\ell-1}$ by some group element $g \in G$ and get $f'_{\ell-1}$, i.e., $f'_{\ell-1} = \mathbb{T}_g^{\ell-1}(f_{\ell-1})$, where $f'_{\ell-1}(x) = f_{\ell-1}(g^{-1}x)$. Then

$$\begin{aligned}
\theta_\ell(f'_{\ell-1})(u) &= (f'_{\ell-1} * X_\ell)(u) \\
&= \sum_{v \in G} f'_{\ell-1}(\eta_X(uv^{-1}))X_\ell(v) \\
&= \sum_{v \in G} f_{\ell-1}(g^{-1}(\eta_X(uv^{-1})))X_\ell(v).
\end{aligned}$$

By $g^{-1}(\eta_X(uv^{-1})) = \eta_X(g^{-1}uv^{-1})$ this is further equal to

$$\sum_{v \in G} f_{\ell-1}(\eta_X(g^{-1}uv^{-1}))X_\ell(v) = (f_{\ell-1} * X_\ell)(g^{-1}u) = \phi_\ell(f_{\ell-1})(g^{-1}u).$$

Therefore, $\phi_\ell(f_{\ell-1})$ is equivariant with $f_{\ell-1}$. Since σ_ℓ is a pointwise operator, so is $f_\ell = \sigma_\ell(\phi_\ell(f_{\ell-1}))$. By induction on ℓ, using the transitivity of equivariance, this

implies that every layer of $\mathcal{N}$ is equivariant with layer 0. Note that this proof holds not only in the base case, when each f_ℓ is a function $X \to \mathbb{C}$, but also in the more general case when $f_\ell : X_\ell \to V_\ell$ and the filters are $X_\ell : X_\ell \to V_{\ell-1} \times V_\ell$.

□

3

Linguistic convolution on finite language graphs

Some languages translate better between each other than others. For simplicity it would be best to restrict things to a set L of similar languages, e.g those of latin origin, and assume that there exists some perfect translation function between texts written in them. It would be good to further place the elements of L as nodes in a directed graph with the translation function describing it's edges. From there we choose a random cycle and identify it with the permutation $T : L \to L$ that the cycle represents. By composing T with the function ϕ_{rep} that maps a text document to it's token matrix, we can generate sets of graphs that may look very different in their representation, but that convey the same linguistic information. This allows us to develop theories about translational equivariance irrespective of the method used for representing the text graphs(tokens, bytes etc) or embedding of the text in graphs(sentence vs word level nodes). This abstraction also allows us to incorporate knowledge upwards from the representation space through the representation function(e.g explicit tokenization), and downward from the linguistic domain through the translation function(e.g a more refined, potentially probabilistic structure for L).

3.1 Sample dataset

For this text, we will define L as consisting of the languages english and swedish. Each language is defined as consisting of a finite set of phrases as shown below:

$P_{eng} = \{$"How long before the train arrives?", "Why are they striking?", "Should we get a taxi?"$\}$,

$P_{swe} = \{$"Hur lång tid är det tills tåget anländer?", "Varför strejkar dom?", "Ska vi ta en taxi?"$\}$.

Thus $|L| = |\{eng \cup swe\}| = 6$. Machine learning models can however not process language directly, they need an intermediary representation of it more suitable for doing computations. Transformer style models rely on tokenization. If we denote tiktoken by E, the tokenizer used in the transformer model GPT-4 by OpenAI[TT033], we can see that the languages can be represented as:

$$eng = E(P_{eng}),$$

$$= \{[4438, 1317, 1603, 279, 5542, 30782, 30],$$

$$[10445, 527, 814, 610, 10789, 30],$$

$$[15346, 584, 636, 264, 33605, 30]\},$$

$$swe = E(P_{swe}),$$

$$= \{[98428, 70469, 983, 13413, 19106, 3474, 259, 3385, 259, 3870, 456, 459, 44283, 910, 30],$$

$$[4050, 96061, 5527, 73, 29234, 4824, 30],$$

$$[50, 4657, 3355, 9637, 665, 33605, 30]\},$$

As the phrases of each language have such different meanings, the translation function T between them should be evident for those who speak both languages. Made explicit below:

$$T([4438, 1317, 1603, 279, 5542, 30782, 30])$$

$$= [98428, 70469, 983, 13413, 19106, 3474, 259, 3385, 259, 3870, 456, 459, 44283, 910, 30],$$

$$T([10445, 527, 814, 610, 10789, 30]) = [4050, 96061, 5527, 73, 29234, 4824, 30],$$

$$T([15346, 584, 636, 264, 33605, 30]) = [50, 4657, 3355, 9637, 665, 33605, 30].$$

As we have chosen to represent L in the same way that the Transformer represents text documents, we can see that ϕ_{rep} reduces to an identity function. Token level translation is however nonsensical for languages that have significant grammatical differences, as between english and japanese, but would be made less nonsensical by introducing a chain of translations between intermediary languages. It would be optimal for the translations to be done at a higher level with a more sophisticated framework, such as DisCoCat[DCC18] for instance. The rigidity of frameworks such as DisCoCat do however introduce challenges when trying to implement it across many sets of languages, therefore a learned component to L may need to be introduced.

3.2 Translation group

We can now see that T generates a subgroup $< T >= \{T^n| \quad 0 \leq n \leq |L|\}$ of the symmetry group S_L. As L is finite, S_L must be finite, and thus $T \leqslant S_L$ is compact with regards to the discrete topology. We can therefore conduct G-Convolution with regards to $< T >$.

4

Conclusions & next steps

In the context of natural language processing, neural networks operate on the languages themselves, not their translations. Given the above definition of language and translation, it is however possible to view languages as heterogeneous spaces of translation. Thus a way to apply G-convolution in existing neural networks focused on language is by convolving the lifted activations of the layers in the network using G-convolution equation.

Overall, there are many steps remaining in order to implement a G-CNN with regards to linguistic translation. Even beyond building appropriate models of language and translation. In terms of computational performance, it would likely also

be a challenge to find a good translation function that can be evaluated in real-time along with the remainder of the neural network, especially during training. I believe that the most viable solution in the short-term is to set up G-CNN layers inside of an existing pre-trained neural network, e.g a Transformer, for "finetuning" and when running it in production. Possibly as a way of allowing greater performance for neural networks built under data and compute restraints. Exploring language and translation model improvements as well as removing bottlenecks for efficient training, preferably from a mathematical perspective, is something I am personally interested in continuing to explore.

References

[GBA16] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press, 2016.

[RT18] R. Kondor and S. Trivedi. *On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups*. https://arxiv.org/abs/1802.03690, arXiv, 2018.

[CW16] T.S. Cohen and M. Trivedi. *Group Equivariant Convolutional Neural Networks*. https://arxiv.org/abs/1602.07576, arXiv, 2016.

[GDL] Bronstein, M. M., Bruna, J., Cohen, T., Veličković, P. (2021). *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*. arXiv preprint arXiv:2104.13478, Pages 80-82.

[DCC18] T. Bradley, M. Lewis, and B. Theilman. *Translating and Evolving: Towards a Model of Language Change in DisCoCat*. arXiv arXiv:1811.11041 https://arxiv.org/pdf/1811.11041.pdf, 2018

[TWT20] S.A. Morris *Topology without tears*. https://www.topologywithouttears.net/topbook.pdf, 2020

[KSH12] A. Krizhevsky, I. Sutskever and G.E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html, Advances in Neural Information Processing Systems 25, (NIPS 2012).

[MS99] M. Spivak. *A Comprehensive Introduction to Differential Geometry*. Vol.1, Publish or Perish, Inc; 3rd edition.

[TT033] OpenAI, *Tiktoken*. https://github.com/openai/tiktoken, GitHub, package version 0.3.3.

[PB2] A. Persson, L-C. Böiers. *Analys i flera variabler*. tredje upplagan, Studentlitteratur AB, 2005.

[ZZSL19] R. Kondor and S. Trivedi. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. https://arxiv.org/abs/1912.08777, arXiv, 2019.

[CH23] G. Garrigos *Handbook of Convergence Theorems for (Stochastic) Gradient Methods*. https://arxiv.org/pdf/2301.11235.pdf, arXiv, 2023.