TECHNICAL UNIVERSITY DRESDEN

FACULTY OF COMPUTER SCIENCE
INSTITUTE OF SOFTWARE AND MULTIMEDIA TECHNOLOGY
CHAIR OF COMPUTER GRAPHICS AND VISUALIZATION
PROF. DR. STEFAN GUMHOLD

Bachelor's Thesis

for obtaining the academic degree
Bachelor of Science

Generatively Learned Feed Forward Networks

Jakob Klinger
(Born 19.06.1999 in Erfurt, Mat.-No.: 4684272)

Tutor: Dr. D. Schlesinger

Dresden, July 19, 2021

Task Description

Mandatory goals:

- Literature research on discriminative and generative modeling.
- Implementation of the proposed learning scheme in PyTorch (a template solution is provided) for FFNs of moderate complexity.
- Exhaustive evaluation on several simple benchmark data sets (like e.g. MNIST, FMNIST, CIFAR-10, SVHN) for a classification problem. Studying the dependencies of the performance on meta-parameters, especially with regard to the batch size.
- Calculation of the expected calibration error and evaluation with regard to it.
- Check the suitability of the proposed approach for out-of-distribution detection and uncertainty quantification.

Optional goals:

- Implementation and evaluation for further problems, like e.g. object detection and recognition, etc.
- Investigate the applicability of the proposed approach for data generation.
- Choice of either a regression or segmentation problem as well as the corresponding benchmark data set, implementation and evaluation of the proposed approach for the chosen use-case. (swapped with out-of-distribution detection)

Declaration of authorship

I hereby declare that I wrote this thesis on the subject

Generatively Learned Feed Forward Networks

independently. I did not use any other aids, sources, figures or resources than those stated in the references. I clearly marked all passages that were taken from other sources and cited them correctly.

Furthermore I declare that – to my best knowledge – this work or parts of it have never before been submitted by me or somebody else at this or any other university.

Dresden, July 19, 2021

Jakob Klinger

Kurzfassung

Obwohl sich die Forschung auf dem Gebiet der generativen Modelle hauptsächlich auf die Generierung von hochqualitativen samples konzentriert, haben diese auch eine Menge Potential für diskriminative Aufgaben. Unter anderem sollen sie zu einem Regularisierungseffekt für kleinere Datensätze führen, die Kalibrierungseigenschaften von Ungewissheiten verbessern und neue Möglichkeiten zur Erkennung von Beispielen erlauben , die nicht der gleichen Verteilung wie die Trainingsbeispiele entstammen.
In dieser Arbeit wird untersucht, inwiefern sich diese angestrebten Eigenschaften auf Modelle für diskriminative Aufgaben übertragen lassen. Dazu werden die Ausgaben eines traditionellen Feed Forward Netzwerkes neuinterpretiert, um unnormalisierte Werte für eine Verbund- statt einer bedingten Wahrscheinlichkeit zu repräsentieren.
Das resultierende Modell wird mit einem normalen Feed Forward Netzwerk mit unterschiedlichen Trainingsbatch Größen auf verschiedenen einfachen Datensätzen zur Bilderkennung verglichen. Die Ergebnisse zeigen, dass der neue Ansatz zu einer verbesserten Regularisierung und teilweise auch zu verbessertem Umgang mit Unsicherheiten führt. Das Modell verbessert jedoch nicht die Erkennung von verteilungsfernen Eingaben.
Insgesamt kann das Einbauen eines generativen Modells in ein standard Feed Forward Netzwerk vorteilhaft sein. Trotzdem braucht es passende Trainingsumstände, um das volle Potential der Generativen Feed Foreward Networks zu entfalten.

Abstract

While research regarding generative models primarily focuses on the generation of high quality samples generative models also hold a lot of potential for discriminative tasks. Amongst other things, they are said to have a regularization effect for smaller data sets, improve the calibration of uncertainty and allow for new out-of-distribution detection possibilities.
This work investigates whether those desired attributes can be included in models for discriminative tasks. Therefore the logit outputs of a standard feed forward network are reinterpreted to represent unnormalized values of a joint instead of a conditional probability distribution.

The resulting model is compared to a standard feed forward network for different batch sizes on multiple simple data sets for image classification. Results show that the new approach leads to improved regularization and can improve the calibration of uncertainty. However the new model does not improve out of distribution detection.

Overall incorporating a generative model in a standard feed forward network can be beneficial. Nevertheless there is still the need for fitting training schemes to unlock the whole potential of the generatively learned feed forward networks.

Contents

1 Introduction

Generative models learning a joint probability distribution are said to have a regularization effect for small and medium data sets compared to standard discriminative models learning a conditional probability distribution [NJ02]. Furthermore they can be beneficial for solving problems like calibration of uncertainty, imputation of missing data, out-of-distribution detection, and more. However research on deep generative models mainly focuses on generating high quality samples.

Nonetheless those attributes are also desirable for problems such as classification or regression. Usually because of their discriminative nature discriminative models outperform their generative counterparts on those problems. Any other improvements from the generative aspect of the model are however useless if the discriminate performance worsens.

Furthermore architectures for state of the art generative models like generative adversarial networks are build to achieve totally different goals and can not be used for supervised learning. However there exists some approaches of using a generative model for discriminative tasks (e.g. [CBDJ19]).

The most relevant one is the generative model of Grathwohl et al. [GWJ+19] which can be trained to match state of the art discriminative performance. However their work used energy-based models which are, in spite of recent improvements of scaling them to high dimensional data, challenging to work with. Therefore the training of their models was still very unstable.

During this work a similar but simpler approach is followed. The logit outputs of a standard feed forward network which is a classical discriminative model can be interpreted as unnormalized values of a conditional probability distribution. Theoretically the logit outputs can be reinterpreted to resemble unnormalized values of a joint probability distribution instead. Under the assumption that the data in a large enough mini-batch is distributed similarly to the data generating distribution a gradient can be computed approximately. Therefore a generative model can be trained on any standard feed forward network architecture by slightly altering the gradients.

The question of this bachelor thesis is whether this newly defined generatively learned feed forward network can incorporate the desired attributes. Particularly the regularization effect, the improved handling of uncertainty and possibilities for out-of-distribution detection are examined.

To evaluate the performance of the generatively learned feed forward network it is compared to a standard feed forward network on the same architecture. Also as the generative model depends on the batch

size all experiments are made for a variety of different batch sizes.

This bachelor thesis will first cover some of the theoretical background of the probability distributions and feed forward networks. It is also shown how standard feed forward network outputs are connected to a conditional probability distribution. Afterwards an overview of the state of the art is given for popular generative models as well as hybrid models. The next chapter explains why and under which assumptions the proposed training approach works. The difference between the new and standard training of feed forward networks as well as the similarities and differences to other works on the field will be discussed. Finally the experimental setups and the methodology are explained and the results are discussed and evaluated.

2 Background

2.1 Probability

2.1.1 Describing Probability Distributions

Probability distributions describe the likelihood of random variables on each of their possible states. They are described differently depending on whether the random variable is discrete or continuous. Probability distributions over discrete random variables may be described using probability mass functions while probability density functions are used for distributions over continuous random variables.

For a discrete random variable x a function $P(x)$ has to fulfill the following properties to be a probability mass function over the variable x:

- The domain of $P(x)$ must be defined for all possible values of x.
- $\forall x, 0 \leq P(x) \leq 1$. All probabilities have to lie between 0 and 1 as an impossible event has a probability of 0 and a guaranteed event has a probability of 1.
- $\sum_x P(x) = 1$. All probabilities have to add up to 1. This is called a *normalized* probability distribution.

Probability density functions $p(x)$ on the other hand have to satisfy the following similar properties:

- The domain of $p(x)$ must again be the set of possible states of x.
- $\forall x, 0 \leq p(x)$. Formally $p(x) \leq 1$ is not required, instead p(x) just needs to be not negative for all possible values of x.
- $\int p(x)dx = 1$. This time the integral has to be 1 for the function to be normalized.

Unlike probability mass functions a probability density function $p(x)$ does not give a probability for a specific state directly, but instead it describes the probability of landing in a particular range of values. The probability of falling in such an interval of values between $\mathrm{x} = a$ and $\mathrm{x} = b$ is given by $\int_a^b p(x)$.

2.1.2 Common Probability Distributions

In machine learning a lot of simple probability distributions are used, one of them is the *Bernoulli* distribution. It is a distribution over a single binary random variable using only a single parameter $\phi \in [0, 1]$, which gives the probability of the random variable being equal to 1. An event described by a Bernoulli distribution can only be either a success $x = 1$ or a failure $x = 0$. For example, it can be used to describe the outcome of a not necessarily unbiased coin toss where either heads or tails, depending on the definition, would represent success while the other outcome would be considered the failure. It has the following properties:

$$\begin{aligned} P(x = 1) &= \phi \\ P(x = 0) &= 1 - \phi \\ \mathbb{E}_x[x] &= \phi \\ \mathrm{Var}_x(x) &= \phi(1 - \phi) \end{aligned} \tag{2.1}$$

Another important discrete probability distribution is the *categorical* or *multinoulli* distribution that describes the probability of each of the k categories a random variable can take on. It can be seen as a generalization of the Bernoulli distribution for variables that can take on more than two different outcomes. An example would be the roll of a dice, where again the outcomes do not have to be equally likely. The categorical distribution is characterized by a vector $p \in [0, 1]^k$, where p_i gives the probability of the i-th state. As all the probabilities have to add up to 1, a vector $p \in [0, 1]^{k-1}$ would in fact be sufficient as the k-th state's probability is already given as $1 - \sum_i^{k-1} p_i$.

Real valued random variables whose distribution is not known yet are commonly described using the *Gaussian* or *normal* distribution. The central limit theorem establishes that the sum of many independent random variables is often approximately normally distributed. There are two parameters used to define a normal distribution: the mean $\mu \in \mathbb{R}$ and the standard deviation $\sigma \in (0, \infty)$. It is then given as:

$$N(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(x - \mu)^2 \Big). \tag{2.2}$$

The parameter μ gives the coordinate for the central peak and because of the symmetry of the $Gaussian$ is also the mean of the distribution: $\mathbb{E}[x] = \mu$. The variance is simply given by $\mathrm{Var}(x) = \sigma^2$.

In its simplest case also known as the *standard normal distribution* the mean is $\mu = 0$ and the standard deviation is $\sigma = 1$.

2.1.3 Conditional and Joint Probability Distributions

Probability mass functions can act on many random variables at the same time. Given two random variables the probability of the event of both $\mathrm{x} = x$ and $\mathrm{y} = y$ happening is defined as the joint probability $P(x, y)$. If the random variables are independent the joint probability distribution can be calculated as:

$$P(x, y) = P(x) \cdot P(y) \tag{2.3}$$

Considering three random variables where x and y are *conditionally independent* given another random variable z. The conditional probability distribution over x and y factorizes for every z as:

$$P(x, y|z) = P(x|z) \cdot P(y|z) \tag{2.4}$$

The probability of an event $\mathrm{y} = y$ given that another event $\mathrm{x} = x$ has happened is called a conditional probability $P(y|x)$. It can be calculated as:

$$P(y|x) = \frac{P(x, y)}{P(x)}. \tag{2.5}$$

It is only defined if $P(x) > 0$.

As a consequence of 2.5 any joint probability distribution over multiple random variables can be expressed with conditional probability distributions over a single variable as:

$$P(x_1, x_2, \ldots, x_n) = P(x_1) \prod_{i=2}^{n} P(x_i|x_1, \ldots, x_{i-1}) \tag{2.6}$$

This is called the *chain rule of conditional probabilities*.

2.2 Feed Forward Networks

Feed forward neural networks (FFNs), loosely inspired by actual neurons of the brain, are the most commonly used machine learning models. They can learn how to approximate a function $f^*(x)$. For example in case of a classifier the function $y = f^*(x)$ maps an input x to a categorical output y. The networks task is to learn the values of the parameters θ that will result in the best approximation of the function $y = f(x; \theta)$.

Sometimes FFNs are also referred to as *multilayer perceptrons* (MLPs) as they are organized in multiple layers of multiple single perceptrons per layer. The first layer of a neural network is called the input

layer as the data is simply fed into it. From there the information is passed forward through a number of hidden layers until it reaches an output layer y.

A single perceptron is nothing more than a simple mapping of its inputs to a single output like this:

$$y = \phi(\sum_i (x_i \cdot w_i) + b) \tag{2.7}$$

where x_i are the elements of an input vector x, y is the single output of the perceptron and ϕ is a nonlinear function called the $activation function$. The weights w_i and the bias b are the parameters θ of the perceptron that are trained to give the best possible approximation of the function that is to be learned.

Considering multiple perceptrons per layer a more compact notation can be used:

$$\vec{y} = \phi(W\vec{x} + \vec{b}) \tag{2.8}$$

where $\vec{x}$ and $\vec{y}$ are the input and output vector of the layer. Again W are the weights but this time in form of a matrix and $\vec{b}$ is the vector of biases. $\phi(W\vec{x} + \vec{b})$ is supposed to imply that the activation function is applied to every element of the vector seperatly. Many different activation functions are used the most common of which are the sigmoid or logistic function:

$$\mathrm{sigm}(x) = \frac{1}{1 + \exp(-x)} \tag{2.9}$$

and the Rectified Linear Unit (ReLU) [NH10] activation function:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.10}$$

.

A whole network is created by using the output of a first layer as the input for a second $f(x;\theta) = f_2(f_1(x;\theta_1);\theta_2)$. Adding more layers to this chain increased the depth of the model. Therefore neural networks with many hidden layers are called *deep neural networks*.

During training the aim of the network is to match the output of the final layer with the label of the given input example for all examples in a training set. The hope is that the parameters learned to correctly classify the examples in the training set generalize well to examples the network has never seen before. In order to train the parameters of the model a *loss* is calculated between the expected result and the present output of the network. This loss becomes larger the bigger the difference between desired and

actual output is. Depending on the application many different loss functions are imaginable. However commonly used loss functions include e.g. the cross-entropy loss based on the principle of maximum likelihood:

$$L(\theta) = -\sum_i y_i \cdot \ln y_i^* \tag{2.11}$$

where y_i^* is the i-th element of the output vector and y_i is the corresponding desired value. This function is a good measure of how distinguishable two discrete probability distributions are.

To get the best constellation of parameters for the training data the loss function has to be minimized with regards to the model parameters θ. As neural networks tend to have thousands or even millions of parameters the minimum of the loss function can not be simply calculated explicitly. Instead a local minimum has to be found using some form of gradient descent algorithm. Here the derivative of the loss function with regards to θ is calculated. To make calculating the gradient with regards to every single weight in the net easier the *back-propagation* algorithm is used. Iterating backwards from the output layer it uses the chain rule of calculus to compute the gradients for each layer at a time, taking into account the previously calculated gradients of the previous layers. This avoids redundant calculations and makes the calculation feasible. The gradient than gives the direction of steepest increase of the loss function. This information is used to make a small update step towards a local minimum of the loss function using the negative gradient:

$$\theta \leftarrow \theta - \epsilon \nabla L(\theta) \tag{2.12}$$

where ϵ is the *learning rate*, a small positive scalar that determines the sizes of the update step. This procedure is iterated multiple times until the model has adapted the weights and biases to make correct predictions for the training data. Theoretically the loss should be calculated as an average over all the training examples. As training sets can be huge and memory is limited the training set is split into mini-batches instead. The calculations are then done over each mini-batch separately to get an approximation of the gradients for the whole data set.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) introduced by LeCun et al. [LBD$^+$89], initially applied to the classification of handwritten zip codes, proven to be incredibly useful for processing data with a grid-like topology like for example images. CNNs are a special kind of feed forward neural network that use

the convolution operation in one or more layers instead of standard matrix multiplication. In fact the convolution can be viewed as a matrix multiplication, however, several matrix entries are constrained to be equal to other entries and the matrix typically contains a lot of zeros.

The convolution operation used in machine learning slightly differs from the convolution used in math or engineering. The core idea is to apply a certain filter or *kernel* ,which is much smaller than the input and is supposed to detect a specific type of feature in the input, systematically to each filter sized patch of the input data. To do so the filter sized patch of the input data is element-wise multiplied by the weights of the filter. The results are then summed up into a single value. Those single values from applying the kernel multiple times will still be arranged in the so called feature map.

For example consider the two dimensional case of an image as in input. The kernel would than be a two dimensional array that is applied to every possible part of pixels from left to right, top to bottom. The result is a two dimensional feature map array.

It is worth noting that the convolution operation shrinks the size of the feature map compared to the original input by the size of the kernel minus one, per dimension. The following problem being that larger kernels would rapidly shrink the feature map which limits the number of possible layers within the network. In order to avoid this problem *zero padding* is used. When using it the input is implicitly padded with zeros on the edges so that the size of the feature map does not shrink anymore.

A convolutional layer typically performs three steps: first multiple convolutions are performed, then a nonlinear activation function is applied to each element of the feature map. Finally a *pooling function* is used. The pooling function replaces multiple outputs of the layer with a single statistic summary of them which results in a reduced size of the representation. The most common pooling functions are max pooling, which uses the maximum of a local cluster as the output, and mean pooling, which uses the average.

While looking quite different to a fully connected standard FFN a convolutional neural network can still be trained using the back propagation algorithm and a stochastic gradient descent algorithm.

CNNs have a variety of advantages compared to fully connected architectures that makes them appealing. First of all the number of weights is drastically reduced because the kernel is much smaller than the input. Leading to a drastically reduced memory allocation. This effect further increases as the weights of a kernel only have to be stored once even though they are applied multiple times. This allows CNNs to be easily scaled to very large sizes. Especially when working with images a lot of features can be obtained from a small local area of the image. As fully connected FFNs reshape the input to be a vector some of those connections get harder to detect.

2.4 FFNs Learn a Conditional Probability Distribution

This section shows how the softmax output of a feed forward network trained with the cross entropy loss function can be interpreted as a conditional probability distribution.

Let $X \in \mathcal{X}$ be a random variable which is also used as the network input. While its type does not matter for this consideration in case of image classification its an image. Let $y \in \mathcal{B}^n$ be the binary vector with the components $y_i \in \{0, 1\}, i = 1 \dots n$. The function $f(x; \theta)$ maps each data point to K real-valued numbers called logits. The parameters θ of this function should be learned.

those logits can be used to get the conditional probability distribution with the softmax function:

$$p(y|x; \theta) = \frac{1}{Z(x; \theta)} \exp\langle y, f(x; \theta)\rangle \tag{2.13}$$

where the normalization constant or partition function is:

$$Z(x; \theta) = \sum_{y} \exp\langle y, f(x; \theta)\rangle \tag{2.14}$$

which ensures that the outputs are a valid probability distribution with values between 0 and 1 that add up to 1 in total.

As the probabilities for each class are conditionally independent from each other the probability distribution can be rewritten as:

$$p(y|x) = \prod_{i} p(y_i|x) \tag{2.15}$$

The partition function can then be rewritten as:

$$\sum_{y} \exp\langle y, f(x; \theta)\rangle = \prod_{i} \sum_{y_i \in \{0,1\}} \exp(y_i \cdot f_i(x; \theta)) \tag{2.16}$$

.

This can be further simplified as for $\exp(0 \cdot f_i(x; \theta)) = 1$ and $\exp(1 \cdot f_i(x; \theta)) = \exp(f_i(x; \theta))$:

$$\prod_{i} \sum_{y_i \in \{0,1\}} \exp(y_i \cdot f_i(x; \theta)) = \prod_{i} (1 + \exp(f_i(x; \theta))) \tag{2.17}$$

Therefore the whole probability distribution (2.15) can be rewritten with the sigmoid function as:

$$p(y|x;\theta) = \prod_i \frac{\exp(y_i \cdot f_i(x;\theta))}{1+\exp(f_i(x;\theta))} = \prod_i \mathrm{sigm}(y_i \cdot f_i(x;\theta)) \tag{2.18}$$

Considering the learning of the conditional likelihood for a single training example (x^*, y^*) the aim is to maximize the conditional log-likelihood under the parameters θ.

$$\ln p(y^*|x^*;\theta) \to \max_\theta \tag{2.19}$$

Using (2.18) results in:

$$\sum_i \Big[y_i^* \cdot f_i(x^*;\theta) - \ln(1+\exp(f_i(x^*;\theta))) \Big] \to \max_\theta \tag{2.20}$$

.

This is binary cross-entropy with logits. The gradient of which averaged over the training data $T = ((x^l, y^l), l = 1 \dots L)$ reads as:

$$\frac{1}{L} \sum_l \Big[\Big\langle y^l, \frac{\partial f(x;\theta)}{\partial \theta} \Big\rangle - \frac{\partial \ln Z(x^l;\theta)}{\partial \theta} \Big] \tag{2.21}$$

$$H = \sum_y p^*(y) \log p(y) \tag{2.22}$$

3 Related Work

3.1 Discriminative and Generative Models

In machine learning models can be roughly grouped into different categories such as discriminative and generative models. Usually a model is called generative if data can be sampled from it while a discriminative model allows to distinguish between different types or classes of data.
A generative model can generate new data instances such as e.g. pictures of handwritten digits that look similar to the real handwritten digits the model was trained with. In this example a discriminative model on the other hand could tell which digit is shown on an given image of a handwritten digit. The key difference however is not whether samples are generated or not but rather which kind of probability distribution is learned from the training data. Discriminative models learn a conditional probability distribution $P(y|x)$ for the output under the input. They ignore the likelihood of a given example but solely focus on how likely a label or output is to apply to this data instance. On the other hand generative models deal with joint probability distribution $P(x, y)$ of the input belonging to an output or simply the distribution of data $P(x)$ if there are no labels. Generative models therefore have to model a more complex representation. Discriminative models draw boundaries in the data space to differentiate between different types of data while generative models try to model how the data is spread throughout the data space. Throughout this work generative models will refer to models learning a joint probability distribution whether or not samples can directly be drawn from it.

3.2 State of the Art Generative Models

In the recent years deep generative models have gained a lot of attention due to their ability to produce highly realistic samples. Especially two approaches have become popular: Variational Auto-encoders (VAE)[KW14] and Generative Adversarial Networks (GAN) [GPAM$^+$14].

VAEs are a kind of probabilistic version of a standard auto-encoder. Standard auto-encoders are neural networks used for feature encoding of unlabeled data and therefore can be trained in an unsupervised way. Their architecture is kind of unusual as the width of the layers gets smaller and smaller until it

reaches a bottleneck from where it gets wider again until it reaches the original input size. The part of the network before the bottleneck is called the encoder while the part behind is called the decoder. The idea behind this is that a high dimensional input is fed into the network which is then forced to reduce the dimensionality of the input towards the bottleneck and from there reconstruct the original image towards the output. If a model is well trained to do this the bottleneck will contain a very dense encoding of the features of the input. Otherwise the model would not be able to reconstruct the original input towards the output.

Variational Auto-encoders improve on this idea for the generation of samples. Instead of mapping an input on a feature representation of a vector, the input will get mapped onto a distribution usually a Gaussian. This is achieved by exchanging the bottleneck vector with two vectors: one representing the mean and the other one representing the standard deviation of the distribution. For decoding a latent vector is sampled from this distribution. For training this arises the problem that back-propagation can not simply push the gradients through the sampling operation. To allow for both the encoder and the decoder to get trained by back-propagation the so called reparameterization trick is used. The latent vector that gets sampled can be seen as $z = \mu + \sigma \odot \epsilon$ where $\epsilon \sim \mathrm{Normal}(0, 1)$ and σ as well as μ are learned parameters. As ϵ is always the same normal distribution containing all the stochasticity the gradients can be calculated for σ and μ. This results in a stochastic part that does not have to be trained and a part of the parameters where back-propagation can be applied.

On a trained VAE a sample from the training distribution can get generated in two simple steps. First a latent feature vector is sampled from the distribution in the bottleneck (e.g. the Gaussian) which is then fed into the decoder where the sample is generated.

Higgins et al. [HMG+16] have proposed the Disentangled Variational Auto-encoder that makes sure that each element of the latent representation vector is uncorrelated. This can be achieved by adding a single hyper parameter to the loss function. By using this method changes to the latent feature representation become much more interpretable.

Generative Adversarial Networks take a very different spin on the subject. They do not work with an explicit density function, but take a kind of game-theoretic approach. The model learns to generate samples from the training distribution through a two player game. those players are the generator network which tries to generate realistic looking samples and the discriminator network which tries to distinguish between real data from the training set and fake samples created by the generator. The generator similarly to the decoder of in a VAE takes a vector of random noise z, sampled from a simple distribution like a Gaussian, as an input which can be seen as latent features of the generated sample. Intuitively what happens is that both parts of the network improve by working against each other. If the discriminator

can confidently distinguish fake from real data the generator needs to improve. On the other hand the discriminator will have to improve its ability to detect the more realistic fake data.

Even though there are no labels for the data in the training set by itself marking the real data as real and the fake data as fake allows to define a loss function for the output of the discriminator. Therefore the models can be trained using the back-propagation algorithm. However because the generator wants to increase the loss of the discriminator gradient ascent is used to train it. The discriminator on the other hand is simply trained using standard back-propagation with gradient descent. Usually training alternates between the generator and the discriminator to make training more stable [LJY].

3.3 Hybrid Models

As shown State of the art generative models look very different than discriminative models. GANs and VAEs are unsupervised training approaches built to generate high quality samples. However some approaches exist that try to combine both approaches in a single model.

Xie et al. [XLZW16] proposed the "generative ConvNet" which can be derived from a standard discriminative convolutional neural network by reinterpreting the logit outputs to define a class-conditional energy-energy based model $p(x|y)$. While this approach allows them to generate realistic texture and object patterns a classifier and an unconditional model need additional parameters to be derived.

Grathwohl et al. [GWJ+19] proposed to reinterpret a common discriminative classifier of a conditional probability distribution as a joint energy-based model. Using the logit outputs of a standard feed forward network like in section xx the joint probability distribution can be defined as:

$$p(x, y; \theta) = \frac{\exp\langle f(x; \theta), y\rangle}{Z(\theta)} \tag{3.1}$$

where $Z(\theta)$ again is the normalization constant but this time it is unknown and can not be computed. The energy function reads:

$$E(x, y; \theta) = -\langle f(x; \theta), y\rangle \tag{3.2}$$

Furthermore by marginalizing out y unnormalized values for $p(x)$ can be obtained:

$$p(x; \theta) = \sum_y p(x, y; \theta) = \frac{\sum_y \exp\langle f(x; \theta), y\rangle}{Z(\theta)} \tag{3.3}$$

The logits of a classifier can therefore be used to also define the energy function of a data point x:

$$E(x;\theta) = -\ln \sum_{y} \exp\langle f(x;\theta), y\rangle \tag{3.4}$$

Their model is trained using a factorization of:

$$\ln p(x, y;\theta) = \ln p(x;\theta) + \ln p(y|x;\theta). \tag{3.5}$$

Their model allows them to rival state of the art discriminative models while simultaneously adding the benefits of generative modeling approaches. those benefits include improved calibration, out-of-distribution detection and robustness compared to a baseline model without EBM training. Furthermore they can generate samples rivaling state of the art generative approaches. However energy-based models are hard to work with especially for high dimensional data. The training of their models was unstable and needed to be improved on.

Another approach are invertible architectures like the Invertible Residual Networks by Behrmann et al. [BDJ18] or Residual Flows by Chen et al.[CBDJ19]. They can improve the discriminative performance of generative models but do not quite match the performance of purely discriminative models.

4 Joint Probability Training of Feed Forward Networks

This section is going to show how the logit outputs of a feed forward network can be reinterpreted to represent a joint probability distribution instead of a standard conditional probability distribution following the derivation of an approximated gradient by Schlesinger [Sch21][1]. Therefore a generative feed forward network (GFFN) is trained. Again as in section 2.4 consider labeled data where x is the input and $y \in B^n$ is a binary vector of the class labels with the components denoted as $y_i \in \{0, 1\}, i = 1...n$. The logit outputs of a neural network will again be denoted as $f(x; \theta)$ for an input x with the parameters θ. The difference is that this time they are supposed to be interpreted as unnormalized logarithms of a joint probability distribution similar to the work of Grathwohl et al. [GWJ+19].
For generative models the probability distribution reads as follows:

$$p(x, y; \theta) = \frac{1}{Z(\theta)} \exp\langle y, f(x; \theta)\rangle \tag{4.1}$$

where the partition function is:

$$Z(\theta) = \sum_{xy} \exp\langle y, f(x; \theta)\rangle \tag{4.2}$$

In contrast to the partition function of a conditional probability model shown in (2.14) where a specific x was given, the partition function is now obtained by summation over the whole (x,y) space. The partition function is now simply a number that ensures that all joint probabilities add up to 1. Though unlike in a discriminative model the partition function can now not longer be computed as summation over the whole (x,y) space is infeasible. Nevertheless an approximation of the gradient with regard to θ is still possible.

Considering a single training example (x', y') the learning task of maximizing the joint log-likelihood is:

$$L(\theta) = \ln p(x', y'; \theta) = \langle y', f(x'; \theta)\rangle - \ln Z(\theta) \rightarrow \max_{\theta} \tag{4.3}$$

the gradient of which with regards to θ reads as:

[1] unpublished

$$\frac{\partial L(\theta)}{\partial \theta} = \langle y', \frac{\partial f(x';\theta)}{\partial \theta} \rangle - \frac{\partial \ln Z(\theta)}{\partial \theta}. \tag{4.4}$$

While the first part can be easily calculated by a standard error back-propagation algorithm the second part remains problematic for now. Using the chain rule results in:

$$\frac{\partial \ln Z(\theta)}{\partial \theta} = \frac{1}{Z(\theta)} \cdot \frac{\partial Z(\theta)}{\partial \theta}. \tag{4.5}$$

Considering that the partition function can be written as $Z(\theta) = \sum_x Z(x;\theta)$:

$$\frac{1}{Z(\theta)} \cdot \frac{\partial Z(\theta)}{\partial \theta} = \frac{1}{Z(\theta)} \cdot \sum_x \frac{\partial Z(x;\theta)}{\partial \theta}. \tag{4.6}$$

Here the "log-trick":

$$\frac{\partial Z(x;\theta)}{\partial \theta} = Z(x;\theta) \frac{\partial \ln Z(x;\theta)}{\partial \theta} \tag{4.7}$$

is used to get:

$$\frac{1}{Z(\theta)} \cdot \sum_x \frac{\partial Z(x;\theta)}{\partial \theta} = \frac{1}{Z(\theta)} \cdot \sum_x Z(x;\theta) \cdot \frac{\partial \ln Z(x;\theta)}{\partial \theta} = \sum_x \frac{Z(x;\theta)}{Z(\theta)} \cdot \frac{\partial \ln Z(x;\theta)}{\partial \theta}. \tag{4.8}$$

The probability of an input x can be marginalized over the joint probabilities. Furthermore this probability describes the the ratio between the joint and conditional probability $p(x) = \frac{p(x,y)}{p(y|x)}$.

$$p(x;\theta) = \sum_y p(x,y;\theta) = \frac{Z(x;\theta)}{Z(\theta)} \tag{4.9}$$

Therefore (4.8) can be rewritten as:

$$\sum_x \frac{Z(x;\theta)}{Z(\theta)} \cdot \frac{\partial \ln Z(x;\theta)}{\partial \theta} = \sum_x p(x;\theta) \cdot \frac{\partial \ln Z(x;\theta)}{\partial \theta} \tag{4.10}$$

While each element of the sum is easy to compute, the summation over the whole x space is still infeasible. However the core assumption of this thesis is that it is possible to approximate this gradient by computing an average over a large enough mini-batch instead of the average over the whole x space. Let $T = (x^1, x^2, x^3...)$ be examples x in the training data;

$$Z'(\theta) = \sum_{x \in T} Z(x;\theta) \tag{4.11}$$

under the assumption that $Z(\theta) \approx Z'(\theta) \cdot const$:

$$\frac{\partial \ln Z(\theta)}{\partial \theta} \approx \frac{\partial \ln Z'(\theta)}{\partial \theta} \tag{4.12}$$

$$p'(x;\theta) = \frac{Z(x;\theta)}{Z'(\theta)} \tag{4.13}$$

The approximated gradient that is now easy to compute reads:

$$\frac{\partial \ln Z(\theta)}{\partial \theta} \approx \sum_{x \in T} p'(x;\theta) \cdot \frac{\partial \ln Z(x;\theta)}{\partial \theta} \tag{4.14}$$

Let the training data $T = ((x^l, y^l), l = 1...L)$ be given. The final approximated gradient of the joint likelihood is:

$$\frac{1}{L} \sum_l \left\langle y^l, \frac{\partial f(x^l;\theta)}{\partial \theta} \right\rangle - \sum_l p'(x^l;\theta) \frac{\partial \ln Z(x^l;\theta)}{\partial \theta} \tag{4.15}$$

with:

$$p'(x^l;\theta) = \frac{Z(x^l;\theta)}{\sum_{l'} Z(x^{l'};\theta)} \tag{4.16}$$

In contrast the gradient of the conditional likelihood:

$$\frac{1}{L} \sum_l \left[\left\langle y^l, \frac{\partial f(x;\theta)}{\partial \theta} \right\rangle - \frac{\partial \ln Z(x^l;\theta)}{\partial \theta} \right] \tag{4.17}$$

only differs in the handling of the partition function. In the standard case it is simply an average over the training data, in the GFFN case it becomes a weighted average. The probability p' can be calculated using a single softmax operation in addition to a single multiplication for the whole weighted average. The proposed approach therefore does not lead to a relevant computational overhead as compared to the standard loss for a conditional likelihood.

It is worth noting that both approaches are equal when only considering a single training example or a training batch with one element.

5 Experiments

5.1 Experiments on Validation Accuracy with Different Batch Sizes

Even though GFFN is called generative and learns a joint probability distribution like other generative models the primary focus of this work is not to generate samples with it. While it would certainly be interesting whether quality samples can be generated using it first and foremost the GFFN is trained as a classifier. Therefore its performance should be compared to a standard FFN classifier. To do so models were trained for both approaches on multiple simple standard benchmark data sets for image classification (MNIST, FMNIST, CIFAR-10, SVHN) using the Python framework PyTorch. All experiments have been run on the same network architecture. A convolutional architecture was chosen as CNNs are the state of the art for image classification. The difference in input sizes of the images between the data sets is compensated by zero padding in the first convolutional layers. As CIFAR-10 and SVHN use RGB-images the network used three input channels on the first convolutional layer compared to the single channel inputs for the black and white images in the MNIST and FMNIST data set.
No dropout [SHK+14] or batch-normalization [IS15] was used to ensure that the output of the model is a deterministic function of the input. It is not necessary by any means but instead removes stochasticity to allow for a better comparison. While Batch Normalization primarily increased training speed also report that Batch Norm has a small regularization effect. [LWSP18] investigated this effect in more detail and have shown that the regularization decreases for larger batch sizes which makes evaluation of the performance with regard to different batch sizes harder.

Adam [KB15] was used as the optimizer as it benefits the convergence rate of stochastic gradient descent based optimization.

It has been shown [KMN+16] that larger training batch sizes lead to a decreased generalization performance. Experiments by Keskar et al. support the theory that this is a consequence of increased noise in the gradient estimation for smaller batches. They, amongst other things like data augmentation, proposed to use *dynamic sampling* where the batch size is slowly increased as the iteration progresses to correct the problem. This is however not an option for experiments with regards to the batch size. Hoffer et al. [HHS17] have also investigated this "generalization gap" phenomenon. They propose to adapt the

learning rate with batch size using a square root scaling and a decreasing learning rate schedule with a sufficient number of high learning rate training iterations. The learning rate for a larger batch can be calculated as:

$$\epsilon_L = \sqrt{\frac{|B_L|}{|B_S|}}\epsilon_S \tag{5.1}$$

where ϵ_S is the original learning rate used for a small batch, ϵ_L is the adapted learning rate and $|B_L|$, $|B_S|$ are the large and small batch sizes, respectively. This already drastically reduced the generalization gap between small and large batch sizes. Also they introduced "ghost batch-norm" which completely closed the generalization gap in most cases, but was not used in this work.

However applying the learning rate adaption and training for a constant amount of iterations is not feasible with batch sizes differing that much. Smaller batch sizes need a much higher number of iterations to finish a single training epoch. As a consequence larger batch sizes can reach training accuracies of close to 100% in a low number of iterations, where the smallest batch sizes did not even finish the first epoch. Increasing the learning rate for the larger batch sizes would further speed up the training process.

For a small number of update iterations the models with very small batch sizes could not finish training. Additionally training the large batch size models with the same number of update steps is also not feasible as the largest batch size is over 500 times larger than the smallest batch size. As a result those models would have to be trained for 500 times the number of epochs of the small batch size models. Even considering the better use of parallelism training for so many epochs takes way to long.

Thus different training schemes were tested. The results for two of them are discussed. First the models were trained until validation accuracy started to flatten and did not improve anymore for multiple epochs. Secondly the models were trained longer following the observation of Hoffer et al. that, contrary to common practices, models can still drastically improve validation accuracy even after reaching training accuracies of close to 100% for multiple epochs. While they were trained far longer the large batch size models were not trained for the same number of update iterations but rather until a given time limit ran out. The results are summarized in Fig. 5.1 and Fig. 5.2.

5.1.1 Results

Most models show the expected generalization gap effect between small and large batch sizes. Exceptions are only the MNIST and FMNIST models trained until the time limit ran out. In the MNIST models the validation accuracy is mostly constant apart from a statistical outlier at the beginning. The FMNIST models is the only one were the validation accuracy first improved and then decreased later. The GFFNs in the longer trained CIFAR-10 models remained mostly constant in contrast to the standard FFN.

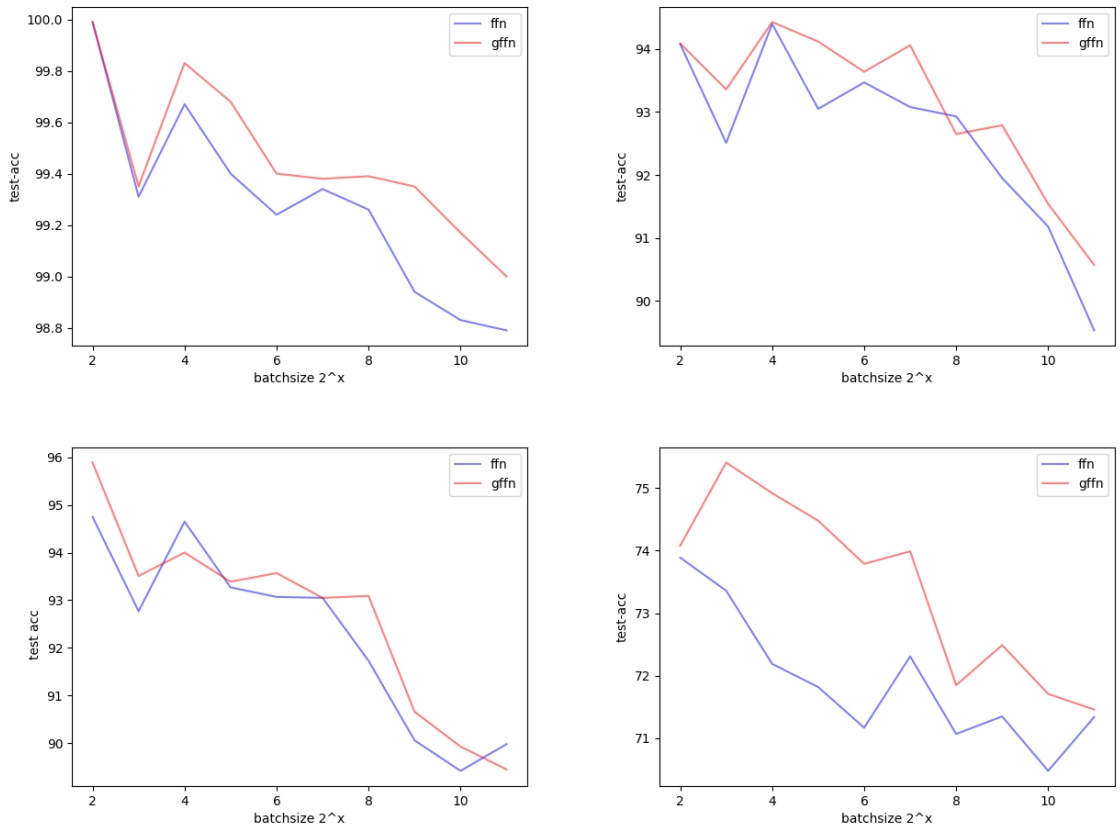

Figure 5.1: Test accuracies for both approaches with different batch sizes, trained until validation accuracy stopped improving for multiple epochs.top left: MNIST, top right: FMNIST, bottom left: SVHN, bottom right: CIFAR-10

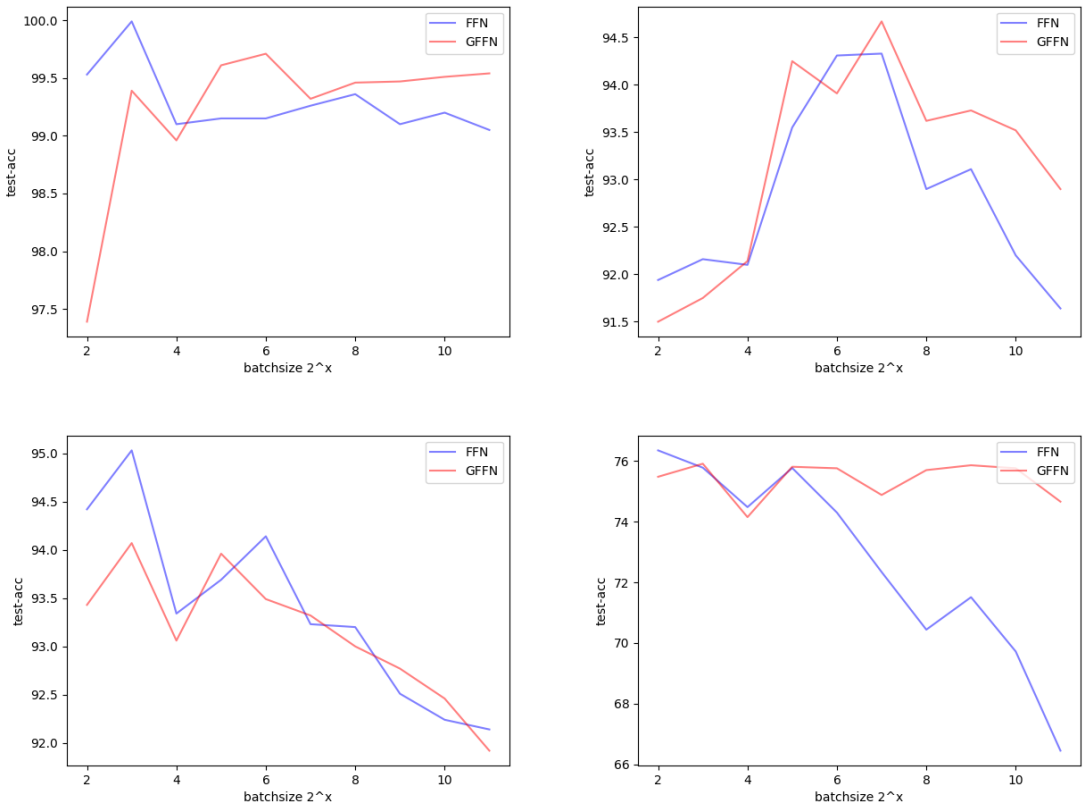

Figure 5.2: Test accuracies for both approaches with different batch sizes, trained longer. top left: MNIST, top right: FMNIST, bottom left: SVHN, bottom right: CIFAR-10

Furthermore the GFFN is mostly better or at least about equal to the classical FFN in all experiments. The biggest difference can be seen on the CIFAR-10 data set while the difference is smallest for SVHN. In some experiments the performance difference between the GFFN and standard FFN seem to increase with the batch size. This can be observed in the longer trained MNIST model as well as in the shorter trained CIFAR-10 and to some extend even in the FMNIST experiments. The longer trained CIFAR-10 experiments suggests the opposite while for the rest of the experiments the difference does not seem to depend on the batch size at all.

An expected result would be that the performance difference between the GFFN and the FFN increases with the batch size. The thought behind that is that for smaller batch sizes both approaches do not really differ a lot and are even equal for training with a batch containing only one example. For larger batch sizes however the examples are similarly distributed to the data generating distribution and therefore the model can actually start learning a good approximation of this distribution. Some experiments show this behavior but in general it is not observable. A possible explanation is that the results for small batch sizes are very noisy. This can also be seen as the accuracies for small batch sizes are heavily fluctuating. It might be a result of the increased noise that is introduced by training with smaller batch sizes.

Even though the GFFN approach performs better overall the results show that the absolute best test-accuracy can be reached by using a small batch. For small batches however the GFFN approach does not really make any difference to the point where for a batch size of one both approaches are identical. In other words if the best results arise for small batch sizes than there is no point in using the proposed approach at all. In practice FFNs are trained using high batch sizes to effectively use the parallelization of GPUs and speed up the training process. This does however not come with the cost of a reduced performance. Therefore applying regularization techniques such as dropout, batch normalization or adversarial training in addition to the methods by Hoffer et al. seem to hold the potential for completely closing the gap. If this can be done while retaining the additional regularization effect of the GFFN it presents a valuable upgrade on the standard training approach. It is not given though that training a GFFN with all those techniques will result in the larger batch sizes bringing up the absolute best validation accuracies.

5.2 Calibration

5.2.1 Discussion of Calibration Metrics

To be useful machine learning models need to be trustworthy, not only do they need to have a high accuracy but for many important real life applications the model should also indicate how certain it is.

Therefore a model should provide a calibrated confidence measure for its prediction. For example, given enough outputs of a well calibrated neural network each with a confidence of 70% one could expect 70% of them to be correct. Theoretically if this could hold for every probability the model would be perfectly calibrated. However this is not possible in practice as the confidence prediction is a continuous variable and only limited samples are given.

To get a visualization of the calibration properties of a neural network *reliability diagrams* can be used (e.g. Fig .5.5). Here samples are grouped into M different equally sized interval bins depending on the networks confidence of the output being correct. The diagram then shows the accuracy for the samples in each bin. All samples with a predicted confidence within the Interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$ would fall into the bin B_m. The accuracy of which would be calculated as the number of correctly predicted outputs divided by the number of samples in the bin:

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}(y_i^* = y_i) \tag{5.2}$$

where y_i^* is the predicted and y_i the true output of a sample i. The average confidence of the bin is denoted as:

$$\mathrm{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} p_i^* \tag{5.3}$$

where p_i^* is the confidence of the model for the output of sample i being correct. For all bins $m \in \{1, \ldots, M\}$ of a well calibrated model $\mathrm{acc}(B_m) \approx \mathrm{conf}(B_m)$. In case of a perfectly calibrated model $\mathrm{acc}(B_m) = \mathrm{conf}(B_m)$ would plot the identity function. The gap between the actual accuracy and the expected one on the identity function is shown as the gap in red.
The histogram below as well as the brightness of the red gap indicate the number of samples in a bin.
A more compact way of describing the calibration properties of a model is the *expected calibration error* (ECE; []) defined as the weighted average over all bins of the gap between a bins accuracy and confidence:

$$\mathrm{ECE} = \sum_m \frac{|B_m|}{l} \big|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\big| \tag{5.4}$$

where l is the total number of samples. Another similar score is the *maximum calibration error* (MCE;):

$$\mathrm{MCE} = \max_m \big|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\big| \tag{5.5}$$

.

It is a measure for the largest accuracy calibration gap of all bins. In the reliability diagram it is the largest red gap across all the bins. While it might be useful for some applications for the evaluation of the calibration properties the weighted average of the ECE is more relevant though.

Guo et al. [GPSW17] have shown that unlike old neural networks from 1998 modern neural networks from 2017 lack this property and are poorly calibrated. They propose temperature scaling as a simple but quite effective way of improving the calibration error. Nevertheless the problem still remains relevant. If the model is well calibrated it can be trusted in case of a high certainty. If the confidence of the network is low the output can not be trusted and if this is possible the task should be taken over by a human instead. For example, imagine a neural network that was trained on thousands of medical cases and is supposed to help with diagnostic decisions. As long as the model confidently predicts what kind of disease the patient is suffering from its output can be used to help the doctor. Should the neural network be unsure of what the problem is the doctor should instead focus on his own diagnosis.

Again experiments on the same architecture with the standard and the GFFN approach with different batch sizes have been made.

5.2.2 Results

The resulting ECEs for the longer trained models are plotted in Fig. 5.3. The models where training was stopped earlier seem to be strictly worse and are therefore not analysed further. An example of a reliability and confidence diagram is displayed in 5.5.
For all models the ECE of the GFFN models is lower and therefore better. Furthermore the difference increases with the batch size for MNIST, FMNIST and CIFAR-10. Again this would be expected as the two approaches are increasingly different for larger batches. But this time it is actually observable for MNIST, FMNIST and CIFAR-10. For SVHN the GFFN models are constantly just slightly better.
Additionally on all data sets the ECEs increase with the batch size, just slower for the GFFN models. This increase is probability partially a result of the validation accuracies decreasing with higher batch sizes. Moreover the data sets with the highest validation accuracies also have the lowest ECEs. This is pretty intuitive as models with high accuracy are easier to calibrate. For example a perfect model with 100% accuracy can simply have a confidence of 100% for every input and will have an expected calibration error of zero. Similarly the MNIST models all have about 99% accuracy and predict nearly all outputs with the highest confidence. However in this case no information about the handling of uncertainty are given, simply because there is no uncertainty for this simple data set (Fig. 5.4). The more complex the data sets get the lower the validation accuracy becomes and the more outputs have a lower confidence. This can be seen very well on the CIFAR-10 models that have very large ECEs. However

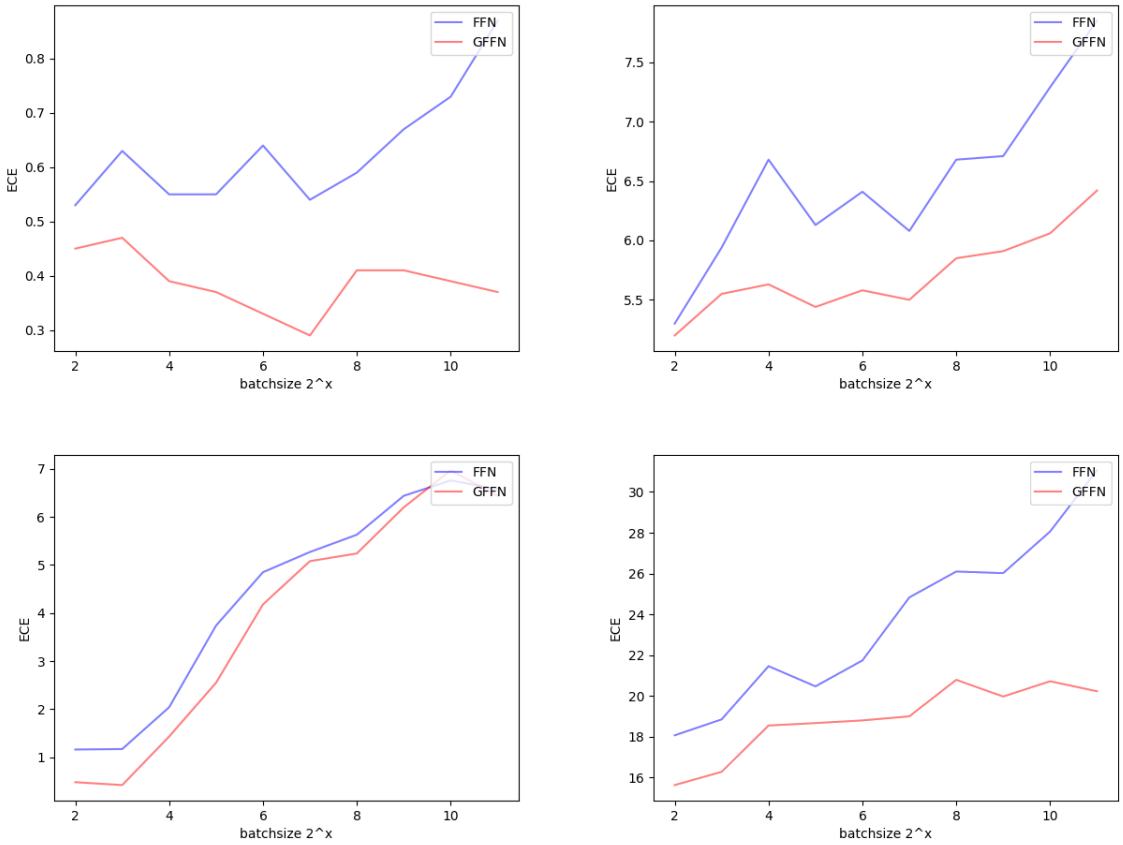

Figure 5.3: Expected calibration error for both approaches with different batch sizes, trained longer. top left: MNIST, top right: FMNIST, bottom left: SVHN, bottom right: CIFAR10

this is also were the GFFN models could show the most improvement over the standard FFN.

The reliability diagram in Fig. 5.5 shows an example of how the calibration properties improved using the GFFN approach. The diagram on the left shows large gaps between expected and actual accuracy. Moreover the the actual accuracies of the bins do not seem well calibrated at all. On the right though the accuracies of each bin are not following the identity function as well but however they increase steadily. While the confidence is still higher than the actual accuracy this at least allows for higher confidence outputs to be regarded as more accurate. In the standard model outputs with $0.5 - 0.6$ confidence have the same accuracy as outputs with $0.8 - 0.9$ confidence. It is also worth noting that the samples are more equally distributed and the accuracy for the highest confidence bin increased a lot.

Overall the GFFN model had significantly reduced ECEs compared to standard FFN. However the problem remains that the expected calibration error was best for small batch sizes which is not how the GFFN is supposed to be trained. Similarly to the validation accuracy it would be necessary to find training

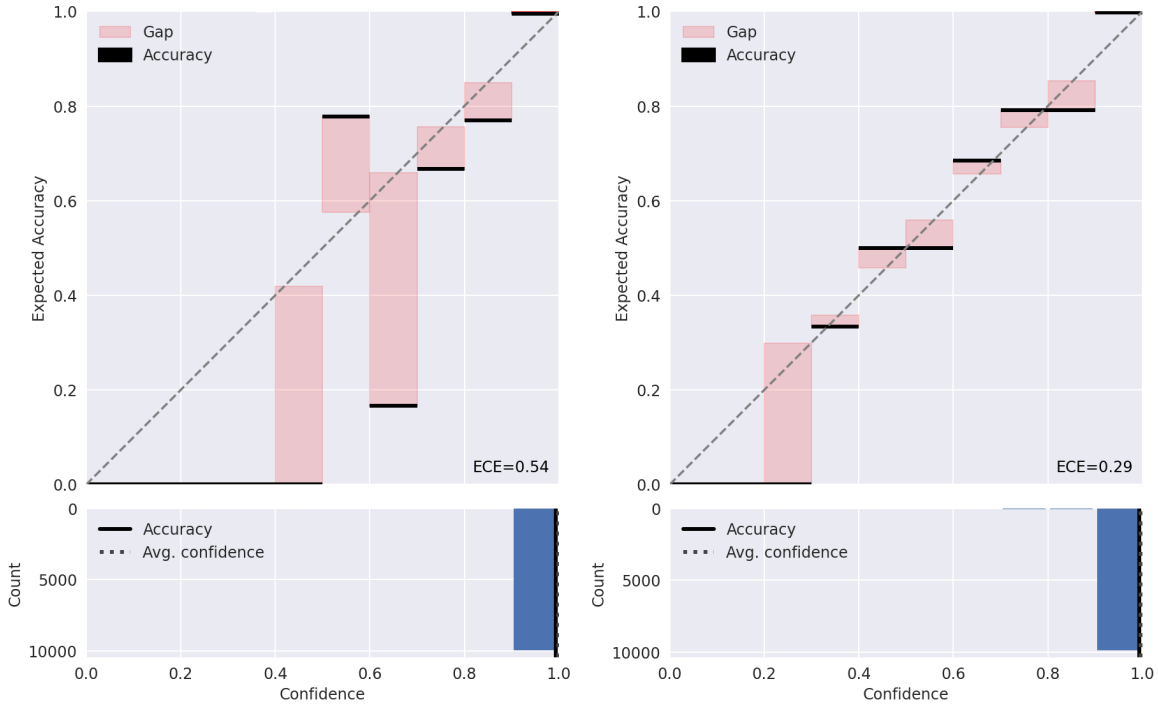

Figure 5.4: Reliability and confidence diagrams for MNIST with batch size 128. left: standard approach, right: GFFN approach

conditions under which those attributes reach their absolute best on higher batch sizes.

5.3 Out-Of-Distribution Detection

5.3.1 Methodology

Knowing whether or not a machine learning model is qualified to predict an output for a given input is of great importance to applying machine learning models to real life scenarios. If the model can not differentiate between data from the training distribution and other data it may predict outputs with high confidence for data that it is not accustomed to process. This is called the out-of-distribution (OOD) problem. OOD detection is the binary classification problem of deciding whether an input is sampled from the same distribution of data the model was trained on or not. Therefore a score $s(x;\theta) \in \mathbb{R}$ is calculated for an input x. The aim is to assign higher scores to in-distribution examples than to out-of-distribution examples. Therefore inputs with a score above a certain threshold will get treated as in-distribution while inputs with a smaller score are getting classified as out-of-distribution. It is unlikely that any approach can guarantee that all possible in-distribution examples get a higher score than any possible out-of-distribution example. The threshold therefore needs to be a trade-off between cor-

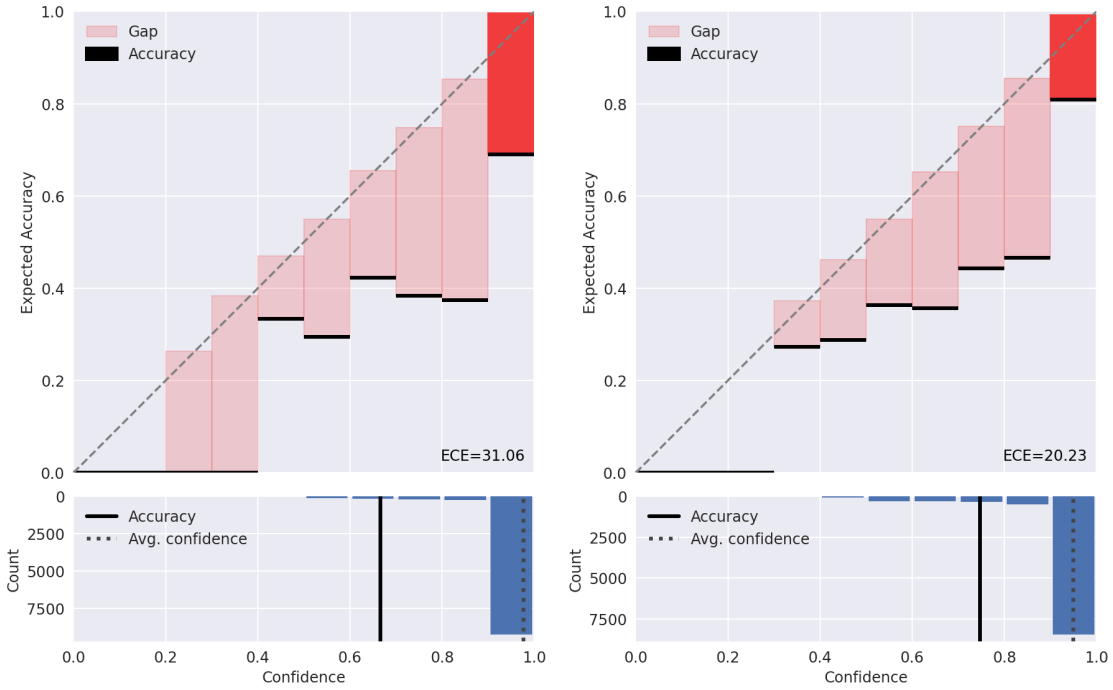

Figure 5.5: Reliability and confidence diagrams for CIFAR-10 with batch size 2048. left: standard approach, right: GFFN approach

rectly classifying every positive example and reducing the number of mistakenly correctly classified negative examples. This trade-off largely depends on the real world application but for evaluation purposes a threshold-free metric can be used such as the area under the receiver-operating curve (AUROC; [HG16]). This curve can be created by calculating the true positive rate and false positive rates for many different possible thresholds. The ROC then plots the true positive rate against the false positive rate. A larger AUROC means that a threshold exists that better differentiates between in-distribution and out-of-distribution. For a perfect classifier the AUROC would be equal to 1. A random classifier on the other hand has a AUROC value of 0.5. Hendrycks et al. have also established the maximum softmax output $\max_y p(y|x)$ as a baseline score for OOD detection. As the GFFN approach allows to calculate unnormalized values of the probability of examples $p(x)$ a logical idea is to simply use this score for OOD detection. Examples that are not drawn from the trained distribution are expected to have a far lower likelihood than examples sampled from the learned distribution. Therefore examples with low likelihood could just be considered as OOD. Determining what kind of examples should be used for OOD detection is actually not trivial as examples that are explicitly sampled from another distribution are not included in a data set. Furthermore every input that does not stem from the data generating distribution could be used, but there is no gain in simply detecting entirely nonsensical inputs. Instead the challenge is to detect inputs that are somewhat realistic but still clearly not sampled from the distribution.

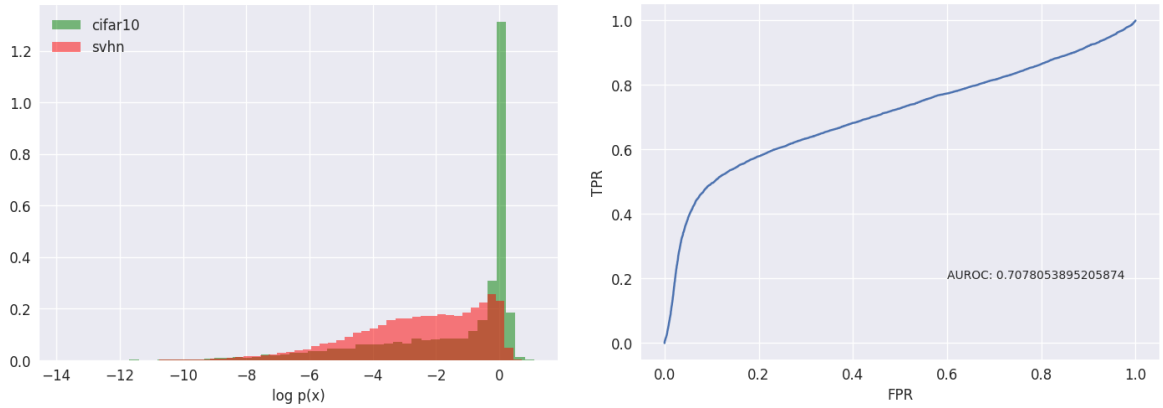

Figure 5.6: CIFAR-10 model with batch size 2048, left: histogram of the distribution for unnormalized probabilities of the inputs, right: receiver-operating curve

A common practice for analysing OOD detection is training a model on one data set and then feed samples from another data set, which is kind of similar but not related, into it. Then the scores for in-distribution examples can be compared to out-of-distribution examples. Out of the data sets used CIFAR-10 and SVHN are fitting as both data sets contain colored images of $32x32$ pixels . Furthermore for a human pictures of cats, dogs, plains etc. are easily distinguishable from house numbers. MNIST and FMNIST on the other hand are too different as the MNIST pictures contain only black and white pixels while FMNIST pictures also use different grey scales.
Accordingly AUROC scores for OOD detection using the maximum softmax output and the unnormalized probability of an input have been calculated on CIFAR-10 and SVHN models. Again only the longer trained models are evaluated here.

5.3.2 Results

The distribution of scores for the samples of different data sets can be visualized using a histogram like in Fig. 5.6. Here green marks the distribution of in-distribution examples that stem from the data set used to train the model, red are out-of-distribution examples from another data set.

The AUROC values for models trained on CIFAR-10 and SVHN are plotted in Fig. 5.7. For both data sets and all batch sizes the $\log p(x)$ approach for OOD detection is worse than the baseline method. Even more the results are mostly not much better than a random classifier and in some cases even worse. This is surprising as an AUROC score of less than 0.5 means that OOD examples are assigned with higher likelihoods than in-distribution examples.
Even though for most models it is better than a random classifier the results are still very poor. The

images of CIFAR-10 are so visually different than SVHN images that higher scores and therefore more distinguishable curves in the histogram would have been expected.

But the phenomenon is also observed by for example Choi and Jang [CJ18] and Nalisnick et al. [NMT+19]. This is very interesting as generative models can in some cases assign high likelihoods to OOD examples but in practice still only generate samples from the distribution they were trained on. Multiple explanations have been suggested but no there does not seem to be one that is commonly agreed on. Serrà et al. [SÁG+19] suggest that generative models manifest a strong bias towards the complexity of the inputs. They report that complex images tend to produce lower likelihoods and simple images always get assigned the highest ones. The experiments with the GFFN however do not show this behavior. For the models trained on CIFAR-10 which is arguably the more complex data set the probabilities of in-distribution examples are higher. The models do not assign higher likelihoods to SVHN images.

Another idea is proposed by Ren et al. [RLF+19]. They describe an input as a combination of two things: a background component characterized by background statistics and a semantic component that is characterized by patterns from the in-distribution data. Images for example are simply backgrounds and objects. They show that generative models can get confounded by those background components which will lead to flawed likelihood estimations. Nalisnick et al. state similar observations where the flow based generative models used seemed to capture low level statistics rather than high level semantics. Overall the exact reason for this phenomenon remains unclear and no current explanation is completely satisfying. Whatever the reason might be it is not necessarily flaw of the training approach. Also when considering that the models trained with smaller batch sizes are also not really expected to work well for learning the distribution of the data their results are not that important. The larger batch sizes however at least work better than a random classifier which of course is not a huge achievement but it shows that OOD examples are at least sometimes assigned with lower likelihoods.

Nonetheless simply using the baseline method always works much better. The only exception is the CIFAR-10 model with the batch size of 2048 which at least gets close the the baseline method.

Furthermore the ability to learn $p(x)$ does not seem to depend on the batch size apart from a few outliers. However for small batch sizes the model can not really learn the underlying distribution of the data. Consequently it is questionable whether any learning of the distribution of the data is taking place at all. Answers to this can not be given until the phenomenon is understood better. Still the new approach can for sure not be used for improved OOD detection.

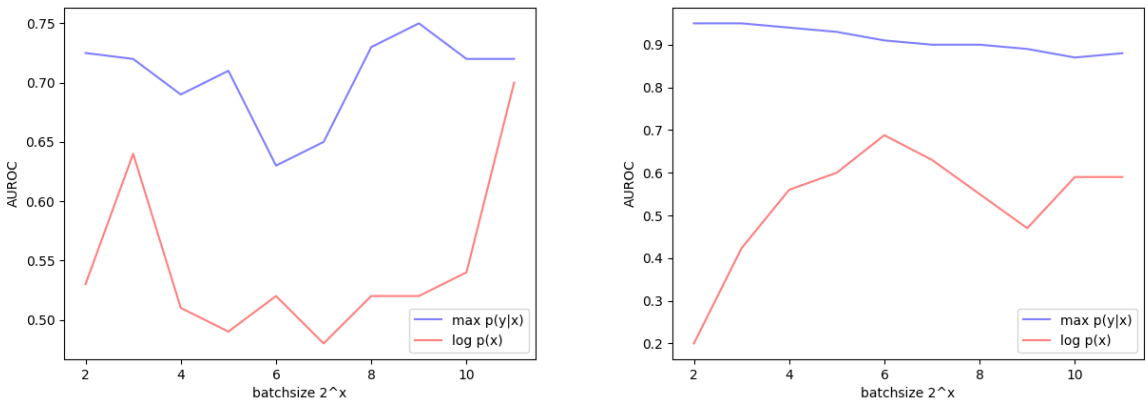

Figure 5.7: AUROC values when using $\max_y p(y|x)$ and $\log p(x)$, left: CIFAR-10, right: SVHN

6 Summary

It has been shown that generative models have a lot of desired attributes. The goal of this bachelor thesis was to evaluate whether a standard discriminative feed forward network can be trained in a way similar to generative models that allows it to incorporate the attributes. This is done be reinterpreting the logit outputs of a FFN as unnormalized logarithms of a joint probability distribution instead of the usual conditional probability distribution. This is however only possible under the assumption that the model is trained on a large enough mini batch. Under this assumption an approximated gradient can be derived. The resulting learning formulation does not lead to a relevant computational overhead.

Experiments on different simple benchmark data sets for image classification show that the generatively learned feed forward network (GFFN) achieves better training accuracies compared to a standard FFN. However in all experiments larger batch sizes lead to worse absolute results. This effect is known and different methods of reducing its impact exist. They were however not applied in this work yet. In order to fully make use of the potential of those generatively learned feed forward networks those methods need to be incorporated into the training setting. How those methods interact with the proposed training approach is not entirely clear and could be further investigated. Using them in addition to further regularization techniques will likely lead to even better and more applicable results.

Compared to the standard FFN the new approach can also drastically improve the calibration properties of a model.

An interesting idea is to use the estimated probability of a sample for out-of-distribution detection. Surprisingly the models can not reliably distinguish images from CIFAR-10 and SVHN in this way. Simply using the maximum soft max output works better. Furthermore out-of-distribution examples can even get assigned with higher likelihoods than in-distribution examples. In this case even a random classifier for out-of-distribution detection would perform better. As this phenomenon is also observed on entirely different and much more advanced models it is not necessarily a flaw of the training approach but at least makes it unusable for out-of-distribution detection. However it may also indicate that the model can not learn the underlying distribution of the data correctly.

It is therefore highly interesting and could prove to be relevant for other models as well to find the reason for it. While there is some research about it no explanation is commonly agreed on yet. Knowing why this happens and maybe even how to avoid it could open up the possibility to use the estimated likelihood

of an input for out-of-distribution detection.

Furthermore the approach should be tested on a larger variety of different applications other than image classification such as regression or segmentation problems to further verify its use. Similarly it could be examined whether the performance holds for larger and more complex data sets.

It could also get investigated whether further promising aspects of generative models can get incorporated for the discriminative tasks such as imputation of missing data or robustness to adversarial attacks.

On a different note it would be interesting to investigate whether samples can be generated using this approach. For now it seems to be difficult when even out-of-distribution examples can be assigned with higher likelihoods but other generative models that suffer from similar problems are also able to do it.

In summary generatively learned feed forward networks do profit from the generative training approach even when used for discriminative tasks. To make full use of its potential further work is required.

Bibliography

[BDJ18] Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *CoRR*, abs/1811.00995, 2018.

[CBDJ19] Ricky T. Q. Chen, Jens Behrmann, D. Duvenaud, and J. Jacobsen. Residual flows for invertible generative modeling. *ArXiv*, abs/1906.02735, 2019.

[CJ18] Hyun-Jae Choi and Eric Jang. Generative ensembles for robust anomaly detection. *ArXiv*, abs/1810.01392, 2018.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. *CoRR*, abs/1706.04599, 2017.

[GWJ+19] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *CoRR*, abs/1912.03263, 2019.

[HG16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, abs/1610.02136, 2016.

[HHS17] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[HMG+16] I. Higgins, Loïc Matthey, Xavier Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. *ArXiv*, abs/1606.05579, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network

training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[KW14] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[LBD+89] Yann LeCun, Bernhard Boser, John S. Denker, Richard E. Howard Donnie Henderson, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. In *Neural computation*, volume 1, pages 541–551. MIT Press, 1989.

[LJY] Fei-Fei Li, Justin Johnson, and Serena Yeung. Generative models.

[LWSP18] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. *CoRR*, abs/1809.00846, 2018.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[NJ02] Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002.

[NMT+19] Eric T. Nalisnick, Akihiro Matsukawa, Y. Teh, Dilan Görür, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *ArXiv*, abs/1810.09136, 2019.

[RLF+19] Jie Ren, Peter J. Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[SÁG+19] Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F. Núñez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. *CoRR*, abs/1909.11480, 2019.

[Sch21] Dmitrij Schlesinger. Generatively learned feed forward networks, 2021.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhut-

dinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[XLZW16] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative convnet. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2635–2644, New York, New York, USA, 20–22 Jun 2016. PMLR.

Acknowledgments

At this point I would like to thank everyone who supported and motivated me in preparing this bachelor thesis.

Special thanks go to my supervisor, Dr. Dmitrij Schlesinger. With his valuable tips and his helpful answers to my questions, he has contributed a large part to the completion of this work. Thank you for your time and effort.

I would like to thank the Centre for Information Services and High Performance Computing TU Dresden for the opportunity to use the computing power of the HPC. Without this computational power, it would not have been possible to create so many evaluations in such a short time.

I also thank my fellow students and friends, Tim Rieß and Jan Sommer, who assisted me a lot with their helpful stimulation.

Many thanks also to Mr. Jay Stamboolian, who (as a native speaker) has proofread my work in terms of language. So I was able to correct some word order, comma and spelling mistakes.

Finally, I would like to thank my parents for supporting me unconditionally throughout my studies and for motivating me while writing my bachelor thesis.

Dresden, July 19, 2021

Copyright Information

The code for the creation of the reliability diagram is based on an implementation by Matthijs Hollemans under the MIT license. The code can be found on `https://github.com/hollance/reliability-diagrams`