Predictive Coding and Biologically Plausible Neural Networks

Bachelor Project

Anders Bredgaard Thuesen
s183926@student.dtu.dk

January 2022

Abstract

Contents

1 Introduction

1.1 Motivation

In the recent years, deep learning has shown impressive results due to the availability of massive parallel compute and huge amounts of data. From the biological inspiration of the neuron to the perceptron where data inputs are weighted, summed together and thresholded, several new modern architectures, like recurrent, residual and transformer neural networks have pushed the limits and achieved state of the art results in speech recognition, computer vision and natural language understanding. Despite of these networks being originally inspired by the brain, the backpropagation (backprop) algorithm for learning the weights and deep learning in general has been criticized for being biologically implausible [1]. This project will primarily be dealing with on of them: The weight transport problem which arises from the way backprop uses the connection weights in both the forward pass (inference) and the backwards parse (calculating the gradients), requiring that both forward and backward connections have symmetric weights and that information is able to flow backwards through the weights.
Besides having both philosophical as well scientific interest, studying the computational aspects of how the human brain processes sensory input might lead to great improvements in deep learning and artificial intelligence.

1.2 Related work

Several attempts has been made to make modern deep learning more biologically plausible. These can be divided into two types of categories. The first category consists of methods that try to optimize the inference on low-powered neuromorphic hardware such as the Intel Loihi or IBM TrueNorth chips, by converting existing neural network architectures into their spiking counterparts. The other category consists of methods that aim to make the learning phase biologically plausible by only relying on local weight updates in order to optimize for some objective. This is in alignment with the Hebbian learning theory from the neuroscience litterature which states that the synaptic plasticity is only dependent on the pre- and post-synaptic activity, possibly modulated through some global signaling mechasnism (ex. dopamine). One example hereof is the work by Bengio et al. on Continual Equilibrium Propagation [4].

1.3 Research questions

The project will address the following three research questions:

- According to the current literature, what are the biological constraints of biological learning?
- To what extent can predictive coding be used to approximate backpropagation under the above mentioned biological constraints?
- In what ways can modern deep learning benefit from biological plausible learning algorithms?

2 Classical deep learning

2.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is method of finding the optimal parameters, θ, of some assumed probability distribution or model, P, in order to best describe the observed data, $\mathbf{X}$. Statistically this can be formulated as $P(\mathbf{X};\theta)$ or more concisely $P_\theta(\mathbf{X})$. Assuming that the observed data is independent and identically distributed, the likelihood estimator function can be defined as:

$$\mathcal{L}_\theta(\mathbf{X}) = \prod_{i=1}^{N} P_\theta(x_i) \tag{1}$$

where $\mathbf{X} = (x_i, \ldots, x_N)$. MLE can then be formulated as the optimization problem:

$$\theta^\star = \underset{\theta}{\operatorname{argmax}} \;\; \mathcal{L}_\theta(\mathbf{X}) \tag{2}$$

where the goal is to find the optimal set of model parameters (maximum likelihood estimate), $\theta^\star$, that maximizes the probability of the observed data (the likelihood). In the case of supervised learning it is often more interesting to model the conditional probability function $P_\theta(\mathbf{Y}|\mathbf{X})$ where $\mathbf{Y}$ might be some label of $\mathbf{X}$. Fortunately, MLE can be generalized to work with conditional probabilities [6] using the conditional likelihood estimator:

$$\mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{N} P_\theta(y_i \mid x_i) \tag{3}$$

where $\mathbf{Y} = (y_i, \ldots, y_N)$. The MLE task is then to find $\theta^\star = \operatorname{argmax}_\theta \;\; \mathcal{L}_\theta(\mathbf{Y}|\mathbf{X})$. Often, it is easier to instead minimize the negative log-probability as taking the logarithm of a product yields a sum of logs:

$$-\log \mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) = -\sum_{i=1}^{N} \log P_\theta\left(y_i \mid x_i\right). \tag{4}$$

As the logarithm is a monotonic increasing function, minimizing the log-probability is essentially the same as maximizing the probability itself. By assuming that $\mathbf{Y}$ follows a Gaussian distribution $P_\theta(\mathbf{Y} \mid \mathbf{X}) = \mathcal{N}(\hat{\mathbf{Y}}(\mathbf{X}), \sigma)$ with a mean of $\hat{\mathbf{Y}}$ and a fixed variance of σ^2 the log likelihood function can be written as:

$$\begin{aligned} -\log \mathcal{L}_\theta(\mathbf{Y} \mid \mathbf{X}) &= -\sum_{i=1}^{N} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i-\hat{y}_i)^2}{2\sigma^2}} \\ &= \sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}_i)^2 \frac{1}{\sigma^2} + \log\left(\sigma\sqrt{2\pi}\right). \end{aligned} \tag{5}$$

If all constant terms are dropped it becomes apparent that minimizing the mean squared error, $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, performs maximum likelihood estimation under the assumption that the data is normally distributed.

2.2 Dataset

We define our dataset, $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ for $i = 1 \ldots N$ consisting of N pairs of datapoints, $\mathbf{x}_i \in \mathbb{R}^k$ and $\mathbf{y}_i \in \mathbb{R}^l$ where N is the size of the dataset. We notice, that both $\mathbf{x}_i$ and $\mathbf{y}_i$ can

be vectors of possibly differently dimensions k and l respectively. $\mathbf{x}_i$ might represent eg. an image as it is the case with the MNIST dataset used later in this project that consists of 60.000 training examples and 10.000 test examples of 28x28 images depicting handwritten digits and their corresponding labels [12]. As the images in the MNIST dataset are two-dimensional, the image has to be flattened into a one-dimensional vector. In the case of supervised learning, the objective is from $\mathbf{x}_i$ to predict the corresponding label, $\mathbf{y}_i$ which would amount to a single scalar number from 0 to 9 in the MNIST dataset.

Figure 1: First 10 examples of the MNIST training dataset.

2.2.1 MNIST

2.2.2 CIFAR10

2.3 Feedforward neural networks

Feedforward neural networks (FNN) are considered the simplest kind of neural network where the connections between the nodes does not allow for any cycles or recurrent connections. Feedforward neural networks are divided into several layers, where input data from the first layer is "feed forward" through the so-called hidden layers to the final output layer of the network, considered the prediction of the network. FNNs are typically fully connected networks which entails that every node of the network is connected to every node in the previous layer. One special case of FNNs is that of when there are no hidden layers in the network and the input layer is linearly transformed to the output layer and "activated" through a softmax or sigmoid function, depending on the output of output nodes. In that case, the FNN will correspond to (multinomial) logistic regression. This correspondance incentivises the use of activation functions after each linear transformation of the layers, as the network would otherwise not be able to describe non-linearities in the data. Historically, the sigmoid activation function $\sigma(z) = (1 + \exp(-z))^{-1}$ has been the go-to activation function, but recently the rectified linear unit, $\text{ReLU}(z) = \max(0, z)$, has become the de facto standard.

A feedforward neural network with $L-2$ hidden layers is parametarized by the weight matrices $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times n}$ and biases $\mathbf{b}^{(l)} \in \mathbb{R}^m$ for $l = 2 \dots L$ where n is the input dimension of the layer and m the output dimension. A feedforward pass from layer $l-1$ to layer l is given by $\mathbf{a}^{(l)} = \sigma(\mathbf{z}^l)$ where $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$ and σ is the chosen activation function. By letting the initial activation $\mathbf{a}^{(1)} = \mathbf{x}_i$ one can consider the final activation of the network the prediction of the network $\hat{\mathbf{y}}_i = \mathbf{a}^{(L)}$.
test

2.4 Weight initialization

We initialize the weight matrices using Kaiming He initialization, where the entries of the matrix $W_{ij}^{(l)}$ are drawn from a normal distribution with zero mean and $\sqrt{2/n}$ standard deviation as this will help reduce vanishing and exploding gradient problem by keeping the variance in each layer equal when using ReLU activation. [7]

2.5 Back-propagation

The working horse of allmost all modern deep learning models is the back-propagation (aka. backprop) algorithm first popularized for training neural networks by Rumelhart, Hinton & Williams in 1986 [16]. The algorithms solves what is referred to as the *credit-assignment problem*. When learning the parameters of an artificial neural network we would like to know how changing a weight in the network contributes to the total loss, in order to change it in the direction that minimizes the loss. One naive way to do this would be simply to adjust a single random weight slightly, evaluate the new neural network on the dataset and observe the effect on the model loss. If the change leads to a decrease in loss, keep the change, otherwise repeat from the beginning. This would however be very computationally expensive, since the network would have to be evaluated on the entire dataset for each weight in the network. Fortunately, the backprop algorithm achieves this much more efficiently as we will see in the following section.

Back-propagation is an efficient method for calculating the weight updates that minimizes some loss function, $\mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i)$, which measures the difference between the predicted output of the network, $\hat{\mathbf{y}}_i$, and the true output, $\mathbf{y}_i$. Examples of loss functions are squared error $\sum \frac{1}{2}(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$, typically used for regression and categorical cross entropy $-\sum \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i)$ for classification. At its core, the back-propagation algorithm is simply applying the chain rule on the partial derivate of the loss function with respect to the parameters and realizing that a lot of computation can be reused or "back propagated" in order to calculate the weight and bias updates for earlier layers. It is therefore necessary that the loss function is differentiable with respect to the prediction variable. Some alternatives to back-propagation exist such as Direct Feedback Alignment [14], but is not commonly used in practice.

To demonstrate the efficiency of the back-propagation algorithm, one can consider the last and second to last layers of the network. For now, the demonstration will only consider the weights as the bias terms can integrated into the weight matrices by extending the output dimension n by 1 and appending a 1 to the $\mathrm{a}^{(l)}$ vectors resulting which will give an equivalent result. Applying the chain rule on the partial derivative of the loss function wrt. $\mathbf{W}^{(L)}$ yields

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\delta^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{W}^{(L)}} = \underbrace{\mathcal{L}'(\hat{\mathbf{y}}_i)\sigma'(\mathbf{z}^{(L)})}_{\delta^{(L)}} \mathbf{a}^{(L-1)} \tag{6}$$

whose factors can be divided into the error term, $\delta^{(L)}$, and activation in the previous layer, $\mathbf{a}^{(L-1)}$. Yet again, applying the chain rule on the partial derivative of the loss function, but this time wrt. $\mathbf{W}^{(L-1)}$ hints at the source of its efficiency:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L-1)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\delta^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L-1)}} \frac{\partial \mathbf{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}} \frac{\partial \mathbf{z}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}} = \underbrace{\delta^{(L)} \mathbf{W}^{(L)} \sigma'(\mathbf{z}^{(L-1)})}_{\delta^{(L-1)}} \mathbf{a}^{(L-2)} \tag{7}$$

Though the derivation above only considers $\mathbf{W}^{(L)}$ and $\mathbf{W}^{(L-1)}$ it is the general case that

$$\delta^{(L)} = \mathcal{L}'(\hat{\mathbf{y}}_i)\sigma'(\mathbf{z}^{(L)}), \quad \delta^{(l)} = \delta^{(l+1)}\mathbf{W}^{(l+1)}\sigma'(\mathbf{z}^{(l)}) \tag{8}$$

which can be computed using a dynamic programming approach to update the weight parameters using gradient descent with learning rate α:

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha\delta^{(l)}\mathbf{a}^{(l-1)}. \tag{9}$$

The weight is updated in the opposite direction (hence the negation) of the gradient as the aim is to minimize the loss function. By reusing the computation of the error terms, $\delta^{(l)}$, backprop is able to very efficiently compute the weight updates to minimize the global loss function. In theory the weights should be updated according to all datapoints in the dataset to maximize the likelihood, however in practice stochastic gradient descent is used with mini-batches of datapoints that stochastically resembles the overall distribution of the dataset.

2.6 Classification with softmax and cross entropy

It is often of interest in computer vision to classify images into several categories. Examples hereof could be monitoring for explicit content on social media platforms, making images accessible for visually impaired by detecting and reading aloud the image contents or detecting pedestrians in self-driving cars. Although the last two examples are more advanced usecases of computer vision, namely scene recognition and image segmentation, they all rely on the ability to classify images or objects therein.
It is possible to use artificial neural networks in classification tasks by transforming the output logits, $\mathbf{z} = (z_1, \dots z_K) \in \mathbb{R}^K$, into a discrete probability distribution using the softmax vector function:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \dots, K \tag{10}$$

The softmax function scales every z_i to lie in the range from 0 to 1 and makes the sum of all output probabilities equal to 1. It thereby satisfies the formal definition of probability mass functions. An important property of the softmax function is that the ordering of logits and their respective output probabilities stays the same. Eg. the largest value element of $\mathbf{z}$ also has the largest output probability.
If the label is one-hot encoded, meaning that the element in the vector with index corresponding to the correct class set to one and the rest to zero, $\mathbf{Y}$ would have distribution:

$$p(y_i \mid x_i)_k = \begin{cases} 1 & \text{if } c_i = k \\ 0 & \text{otherwise} \end{cases} \quad \text{for } k = 1, \dots, K \tag{11}$$

where c_i is the correct class for label x_i. Neural networks with softmax activation in the last layer can be trained by minimizing the cross-entropy between the one-hot encoded label, p, and the output distribution of the network, q, given by:

$$H(p, q) = -\sum_{i=1}^{N}\sum_{k=1}^{K} p(y_i \mid x_i)_k \log q(y_i \mid x_i)_k. \tag{12}$$

Since $p(y_i \mid x_i)_k$ is only 1 when $k = c_i$, the inner summation over k can be compressed to:

$$H(p, q) = -\sum_{i=1}^{N} \log q(y_{i,c_i} \mid x_i). \tag{13}$$

By comparison with equation 4, minimizing the cross-entropy can be seen as performing maximum likelihood estimation. To actually minimize the cross-entropy using gradient based methods such as back-propagation, both functions have to be differentiable. The Jacobian of the softmax function is a K by K matrix:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial s_1}{\partial z_1} & \cdots & \frac{\partial s_1}{\partial z_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial s_K}{\partial z_1} & \cdots & \frac{\partial s_K}{\partial z_K} \end{pmatrix} \tag{14}$$

where each s_i is is the i'th entry of the softmax output. As each entry is positive, the partial derivative of the logarithm of s_i wrt. z_i can be written as:

$$\frac{\partial}{\partial z_j} \log s_i = \frac{1}{s_i} \frac{\partial s_i}{\partial z_j} \Rightarrow \frac{\partial s_i}{\partial z_j} = s_i \cdot \frac{\partial}{\partial z_j} \log s_i. \tag{15}$$

From the softmax function definition in 10 it follows that:

$$\log s_i = \log \left(\frac{e^{z_i}}{\sum_{l=1}^{K} e^{z_l}} \right) = z_i - \log \sum_{l=1}^{K} e^{z_l} \tag{16}$$

such that the partial derivative of the above becomes:

$$\frac{\partial}{\partial z_j} \log s_i = \frac{\partial z_i}{\partial z_j} - \frac{1}{\sum_{l=1}^{K} e^{z_l}} \cdot \frac{\partial}{\partial z_j} \sum_{l=1}^{K} e^{z_l}. \tag{17}$$

Here, $\frac{\partial z_i}{\partial z_j}$ will be equal to 1 if $i = j$ and otherwise 0, which can be denoted by $1\{i = j\}$. Because e^{z_l} in the sum only depends on z_l, the partial derivative of terms where $l \neq j$ is 0. The above can now be rewritten as:

$$\frac{\partial}{\partial z_j} \log s_i = 1\{i = j\} - \frac{e^{z_j}}{\sum_{l=1}^{K} e^{z_l}} = 1\{i = j\} - s_j \tag{18}$$

in order to finally reveal:

$$\frac{\partial s_i}{\partial z_j} = s_i \cdot (1\{i = j\} - s_j). \tag{19}$$

If one now considers the partial derivative of the cross-entropy function:

$$\frac{\partial}{\partial z_j} H(p, q) = -\sum_{i=1}^{K} y_i \cdot \frac{\partial}{\partial z_j} \log s_i \tag{20}$$

and substitutes in $\frac{\partial}{\partial z_j} \log s_i$ from equation 18 to get:

$$= -\sum_{i=1}^{K} y_i \cdot (1\{i = j\} - s_j) \tag{21}$$

which can be split into the two sums:

$$= \sum_{i=1}^{K} y_i \cdot s_j - \sum_{i=1}^{K} y_i \cdot 1\{i = j\}. \tag{22}$$

As the indicator function $1\{i = j\}$ is only 1 when $i = j$ the above can be simplified as:

$$= \sum_{i=1}^{K} (y_i \cdot s_j) - y_j \tag{23}$$

and as s_j does not depend on i, it can be pulled out of the sum:

$$= s_j \sum_{i=1}^{K} y_i - y_j. \tag{24}$$

Finally, since y_i sums to 1, the partial derivative becomes:

$$\frac{\partial}{\partial z_j} H(p, q) = s_j - y_j. \tag{25}$$

2.7 Batch normalization

When training deep neural networks, it is often necessary to normalize the inputs to the network in obtain good results. By normalizing the input variables to the network the gradient of the loss function is more likely to point in the direction of the local minima resulting in more efficient and stable training. According to the authors [11], batch normalization has the effect of reducing "internal covariate shift" of the networks intermediate representations. As the weights and biases of each layer in the network is updated, the distribution of the internal representations tends to change aswell. This has the effect of slowing down learning and decrease the chance of converging to a good local minima. By normalizing each mini-batch, batch normalization is able to accelerate deep learning by enabling higher learning rates and fewer training steps. Batch normalization works by using a combination of the mean and variance of the current mini-batch and a running mean and variance. At training time, batch normalization standardizes the mini-batch according to

$$\hat{\mathbf{x}}_i^{(l)} = \frac{\mathbf{x}_i^{(l)} - \mu_B^{(l)}}{\sqrt{\sigma^{2}{}_B^{(l)} + \epsilon}} \tag{26}$$

where μ_B is the mean of the batch, σ_B^2 is the variance of the batch and ϵ is some small value for numerical stability (typically 10^{-5}). Batch norm then outputs an affine transformation using learned parameters γ and β

$$\mathbf{y}_i = \gamma \hat{\mathbf{x}}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(\mathbf{x}_i\right). \tag{27}$$

During the training fase, the mean and variance of the dataset is estimated by the means of an exponential moving average

$$\begin{aligned} \mu_{\text{mov}} &\leftarrow \alpha \mu_{\text{mov}} + (1 - \alpha)\mu_B \\ \sigma^2_{\text{mov}} &\leftarrow \alpha \sigma^2_{\text{mov}} + (1 - \alpha)\sigma_B^2 \end{aligned} \tag{28}$$

where $\alpha = 0.1$ is the default value. At inference time, batch normalization uses the dataset mean and variance estimations and affine scaling and translation parameters to normalize the intermediate representations

$$\mathbf{y}_i = \frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \cdot \mathbf{x}_i + \left(\beta - \frac{\gamma \mu_{\text{mov}}}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \right). \tag{29}$$

3 Biologically plausible deep learning

The following section will focus on biologically plausible deep learning and what that entails. There are many aspects of being biological plausible, however this project will only be considering those that are computationally relevant for deep learning. Initially to gain an understanding of biology a brief summary of the neuron is presented. From this biological constraints are defined and compared with current deep learning approaches. Subsequent sections will describe alternative, more biologically plausible methods for training and evaluating artificial neural networks. This project will primarily focus on methods presented in the two papers *Spiking Deep Networks with LIF Neurons* by Eric Hunsberger et al. [9] and *Can the Brain Do Backpropagation? - Exact Implementation of Backpropagation in Predictive Coding Networks* by Yuhang Song et al. [17]. The endgoal of this project is to compare these methods with a unification of the two on the supervised task of predicting the label of MNIST digits. Hopefully, the combination of these methods will shine some light on the intersection between artificial and biological neural networks and provide a foundation for further research.

3.1 Biological neurons

This section aims to briefly describe the biological neuron from a highlevel perspective to give an intuition of how processing is happening in the human brain. Hopefully this will lead to a better understanding of what biologically plausibility encapsulates. It is by no means fully comprehensive nor capturing all the nuanced details of the biological processes.

A neuron is an electrically excitable cell capable of communicating with other neurons by sending electrical signals called action potentials or spikes. Most often, multiple spikes are generated in sequence, called a spike train. When spikes are generated the neuron is said to be *firing*. The neuron has a cell body called the soma which is the central element in processing. From the soma root like structures called dendrites extrude, which capture signals from other nearby neurons. Receiving spikes from other presynaptic neurons will lead to polerization of the cell and trigger the neuron to spike itself if the input spikes exceed a certain threshold. As the neuron spikes an action potential will travel from the soma, through the axon and to the synaptic terminal where it will cause the release of neurotransmitter molecules to the postsynaptic neurons dendrites. Neurons are usually in computational models described by their membrane potential or voltage which arises from the fact that neurons maintain a voltage across its membrane. It achieves this by actively regulating the intracellular concentration of charged potassium, sodium and calcium ions through ion pumps until a resting potential is reached (around -70mV). When neurotransmitter molecules bind to receptors on the dendrites, it will cause various types of ion channels to open. This will result in a change in the ionic concentration and thereby membrane potential of the neuron. Once the membrane voltage reaches a certain level called the

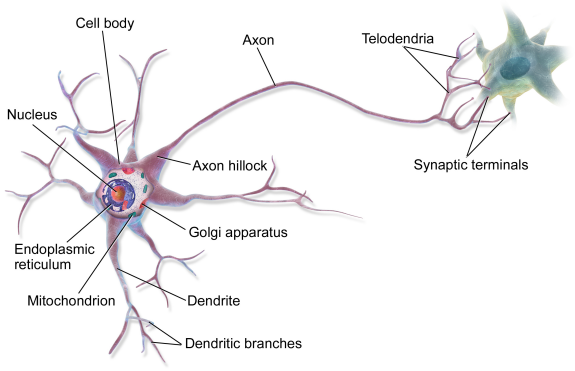


Figure 2: Illustration of biological neuron by BruceBlaus - Own work, CC BY 3.0 [1]

threshold voltage (around -55mV) voltage-gated ion channels open and ions flood in to the cell, further increasing the membrane voltage to form a chain reaction. This reaction causes neurotransmitter to be released at the synaptic terminal. In other words, the neuron will spike. After spiking a phase of hyperpolerization called the refractory period might occur. In this phase, ion pumps decrease the membrane potential beyond the resting potential and prevents the neurons from spiking.

It is believed that the brain learns both by forming new connections between neurons and by varying the amount of neurotransmitter receptors resulting in strengthening or weakening of existing connections. In the book, The *Organization of Behaviour* by Donald Hebb [8], he postulates that the pre- and postsynaptic activity drive this change, known as Hebb's postulate or Hebbian learning:

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

This postulate is congruent with the learning mechanism of spike-timing dependent plasticity (STDP) where changes in synaptic strength depend on the relative timings of the pre- and postsynaptic spikes, which has been observed to happen in the brain[2].

3.2 Biological constraints

To understand how the biological neurons process information and give rise to the emergent phenomenon of intelligence it is of interest to study the natural biological and physical limitations that they exhibit. One interesting question is to whether the perticular biological implementation of the neural networks in the brain are fundamental for the existense of intelligence or if it is rather a single instance in the set of possible configurations. For now, the focus will be on defining the constraints of the only tried-and-testet implementation we know of; the biological neuron. Neurons communicate using spikes of roughly the same amplitude, not known to encode any information, but is able to dynamically modulate the rate and timings of spikes depending on the presynaptic stimulation. This leads to our first **constraint of spikes** for communication. Due to the anatomical structure of neurons

[1] https://commons.wikimedia.org/w/index.php?curid=28761830

communication is unidirectional in the direction from the cell body, along the axon and to the synaptic terminal. No spiking signal is known to travel in the reverse direction from post-synaptic dendrites to pre-synaptic terminal. Thus we introduce a **constraint of unidirectionality**. Neurons primarily communicate with nearby neurons. Efficiency wise, it makes sense for the brain to place functionally dependent regions nearby to save energy in signal transmission. This principle of locality is also found in Hebb's postulate and STDP. We therefore say that biological learning is **constrained by locality**.

3.2.1 Biological violations of backprop and deep learning

In their 2016 paper, Bengio et al [1]. raises 6 problems regarding the biologically plausibility of back-propagation and artificial neural networks which are nicely summarized and expanded upon in the PhD thesis by Hunsberger [10] from which this project will use the same naming of the problems. To illustrate the problems more clearly in terms of dicussions in previous sections (2.5, 3.1) the problems have been reformulated as follows.

1. **Weight transport problem**: Back-propagation uses the same connection weights in both the forward pass for prediction and in the backwards pass when calculating the gradients as seen in equation 8. A well established fact of biological neurons is that of spikes travilling unidirectionally from the soma, through the axons and across the synapses as neurotransmitter chemicals. Backprop does not seem to comply with this constraint.

2. **The derivative problem**: The derivative problem problematises the need for the derivative of the activation function (ex. sigmoid or ReLU used in the forward pass) in order to back-propagate the error signal as it is also the case in equation 8. It is not known that biological neurons can do this.

3. **The linear feedback problem**: The linear feedback problem is due to the fact that neurons communicate in a non-linear fashion making the linear dependence on the feedback error term, $\delta^{(l)}$, in equation 9 difficult to implement for biological neurons.

4. **The spiking problem**: Biological neurons communicate with spikes whereas artificial neural networks use real values. It might be possible in a sense to communicate both positive and negative values stochastically by encoding the value in the spike rate with a combination of excitory and inhibitory neurons, however back-propagation depends on differentiable activation functions to work.

5. **The timing problem**: In the back-propagation algorithm, forwards and backwards passes are run alternatingly and sequantially, which is not known to be happening in the human brain. In constrast, neurons function asynchrously when activated by their inputs and spikes travel between neurons with some propagation delay, making it difficult for biological neurons to implement backprop as activations and their derivates should stay constant when performing weight updates.

6. **The target problem**: It is unclear how labels such as those in the MNIST dataset could be present in the brain. The brain seems to make sense of the world by itself without needing constant supervision. Humans generally only need a few examples in order to identify different objects. Though supervised learning has shown the most impressive results thus far, recent focus on unsupervised or self-supervised learning has startet to show its promise, which might better explain how the brain learns.

3.3 Spiking neural networks

Spiking neural networks (SNNs) are a perticular type of network that uses spikes for inter-neuron communication. In that way they more closeley mimic biological neural networks, but share their structure and dense connectivity with feedforward neural networks. Spiking neurons are temporal in their dynamics in the sense that they are integrating input spikes over time. When the threshold membrane potential is exceeded a spike is sent to the connected neurons in the next layer. The implementation of SNNs requires a perticular neuron model of the action potential often modelled as a first order differential equation describing the exact dynamics. Several different neuron models exists ranging from simple and compute efficient models like the Integrate-and-Fire (IF) model that only captures the coarse dynamics of biological neurons to more nuanced models like the Hodgkin-Huxley model. We will be using a variation of the IF neuron model which leaks membrane potential over time, called the Leaky-Integrate-and-Fire model. This particular model has the advantage of being relatively biologically plausible yet still being very efficient to compute.

3.3.1 Training spiking neural networks

There exist several methods for training SNNs such SpikeProp [3] which uses an algorithm akin to backprop directly on the generated spikes using surrogate gradient functions or modelling approaches where the spiking dynamics of neurons is defined using recurrent networks and trained using backprop through time, like snnTorch [5]. This project will use a method of shadow training where a "shadow" ANN will be trained using spike-rate coding and later run as spiking neural network at inference time using the learned weights from the ANN. This has the advantage of leveraging existing deep learning infrastructure and optimization techniques while being deployable on low-powered or analog hardware.

3.3.2 Leaky-Integrate-and-Fire neurons

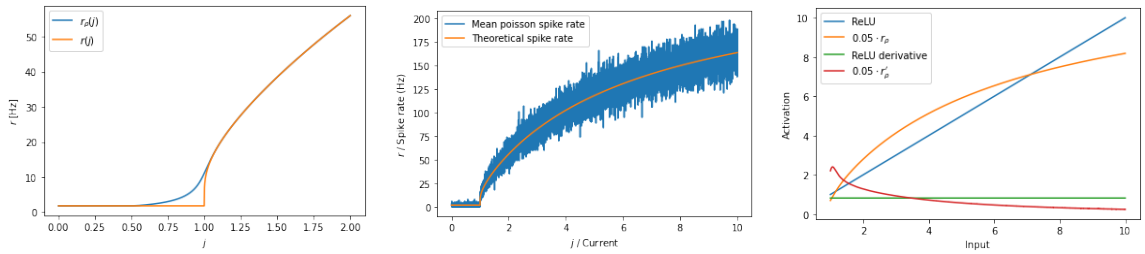
Leaky-Integrate-and-Fire neurons were inspired by the fact that biological neurons seem to lose voltage over time as ions leak through the membrane. Computationally this might be understood as a temporal filter that decreases the effect on the action potential of spikes happened longer time back in the past. The Leaky-Integrate-and-Fire (LIF) neuron model can be described by the following differential equation:

$$C\frac{dV}{dt} = J(t) - \frac{V}{R} \tag{30}$$

where V is the voltage across the membrane, R the membrane resistance and $J(t)$ the input current to the neuron at time t. The neuron will fire when the membrane voltage reaches a threshold of V_{th} and reset to V_{rest} in a refractory period of $t_{\text{ref}} = 0.004$. If the applied input current is not large enough to make the neuron spike, the membrane voltage will naturally decay to V_{rest}. Hunsberger [10] shows how the above can be normalized and rewritten in simpler terms as:

$$\tau_{RC}\frac{dv(t)}{dt} = j(t) - v(t) \tag{31}$$

where τ_{RC} is the membrane time constant and $v(t)$ and $j(t)$, resembling the membrane voltage and input current, are now unitless. Generally τ_{RC} is in the range in the tens of miliseconds for biological neurons. We use $\tau_{RC} = 0.02$. The implication of this rewrite is that the neuron now spikes at $v(t) = v_{\text{th}} = 1$ before resetting to $v_{\text{rest}} = 0$ and entering the



(a) Interleaved plot of original rate function, $r(j)$ (orange), and smoothed version, $r_\rho(j)$ (blue).

(b) Theoretical (orange) and simulated spike rates (blue) as functions of input current.

(c) Scaling of the rate function in the interval $[0; 10]$ to mimic ReLU.

Figure 3: Plots of smoothed, poisson coded and ReLU-like rate functions, respectively.

refractory period. Forcing the input current to be static by letting $j(t) = j$ and solving the differential equation allows us to describe the membrane voltage over time with constant input current:

$$v(t) = (v_0 - j)e^{-t/\tau_{RC}} + j \tag{32}$$

where v_0 is the voltage at $t = 0$ which most often is set $v_0 = V_{\text{rest}}$. It is now possible to derive the spike rate which for input current when the input current exceed the threshold $j > V_{\text{th}}$ by setting $v(t) = V_{\text{th}}$ and solving for $r = t^{-1}$ to get.

$$r(j) = \begin{cases} \left[t_{\text{ref}} - \tau_{RC} \log\left(1 - \frac{V_{\text{th}}}{j}\right)\right]^{-1} & \text{if } j > V_{\text{th}} \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

Notice that t_{ref} is added to the time-to-spike, to incorporate the effect of the refractory period. Using the rate function directly as activation function would lead to problems as its derivative goes to infinity when $j \to 0_+$. Hunsberger et al. therefore introduces a smoothed rate function using the softplus function $\rho(x) = \gamma \log(1 + \exp(x/\gamma))$ that works as a smooth max function and does in fact becomes equivilant when $\gamma \to 0$.

$$r_\rho(j) = \left[t_{\text{ref}} - \tau_{RC} \log\left(1 + \frac{V_{\text{th}}}{\rho(j - V_{\text{th}})}\right)\right]^{-1} \tag{34}$$

A plot of the original and smoothed rate functions can be seen in figure 3a.

3.3.3 Poisson rate coding

As SNNs operate on spiking inputs, continuous input variables have to be converted to stochastic spiking variables. Such can be accomplished with Possion rate coding where the value of the continuous variable is interpreted as the mean spike rate of the Poission distribution with the following parameters:

$$s_i^{(t)} \sim \text{Pois}\left(\gamma = \frac{x_i}{t_{\text{ref}}} dt\right) \tag{35}$$

where dt is the discrete timestep used for simulation. This formulation requires that input variables x_i are normalized to values between 0 and 1 as division with t_{ref} maps 1's to the maximum spike rate $r_{\max} = 1/t_{\text{ref}}$. Disadvantageously, poission rate coding causes the stochastic signal to contain a relativ large amount of noise in the signal as seen in figure 3b.

3.3.4 Low-pass alpha-filter

When modelling biologically realistic neural networks it is equally important to model the synapses as the neural dynamics. Synapses are known to act as low-pass filters on the transmitted spikes. Hunsberger and Eliasmith [9] proposes a simple exponential low-pass filter

$$\alpha(t) = \frac{t}{\tau_s} e^{-t/\tau_a} \tag{36}$$

where τ_s is some time-constant.

3.4 Shadow training spiking neural networks

The method of shadow training spiking neural networks amounts to the task of training a standard neural network using back-propagation, incorporating the rate-function of the chosen dynamical neuron model as activation function, whereby a spiking neural network can be constructed with the learned weight and bias parameters. However, doing so is in some cases not always quite as trivial. This section will go into the details of shadow training feedforward neural networks and the tricks for successfully doing so.

3.4.1 Activation scaling

When training neural networks using the analytical spiking rate as activation function, we found that down-scaling the activation function drastically improved the converge of the network. The reason for this result can be explained by the combination of the Kaiming He initialization of the weights which keeps the variance of the intermediate representations in the network close to 1 when used in conjunction with ReLU activation. By scaling the rate function with a factor of 0.05 the rate function and its derivative will more closely mimic that of the ReLU in the interval $[0; 10]$ as seen in figure 3c.

3.4.2 Rate estimation using alpha-filter

By using a low-pass alpha-filter as defined in equation 36 it is possible to adaptively estimate the underlaying firing rate of a sequence of spikes. This is useful as our rate function expects a current as input which is correlated with the instantatious pre-synaptic firing rate. We found that using an alpha filter with time-constant $\tau_s = 1.18 \cdot 10^{-3}$ most precisely determined the underlaying firing rate.

3.4.3 Fusing batch normalization

In the process of converting shadow trained ANNs to the SNNs, custom modules such as batch normalization layers has to be converted to their spiking counterparts. Fortunately, if used in between the linear layer and activation, it is possible to "fuse" batch normalization into the weight and bias paramters of the predecessing linear layer. By replacing $\mathbf{x}_i$ in equation 29 with the transformation from our linear layer $\mathbf{z}_i^{(l)} = \mathbf{W}^{(l)}\mathbf{a}_i^{(l-1)} + \mathbf{b}^{(l)}$

$$\mathbf{y}_i = \frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \cdot \left(\mathbf{W}^{(l)}\mathbf{a}_i^{(l-1)} + \mathbf{b}^{(l)} \right) + \left(\beta - \frac{\gamma \mu_{\text{mov}}}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}} \right) \tag{37}$$

it is possible to derive the "fused" weight and bias, parameters as follows:

$$\overline{\mathbf{W}}^{(l)} = \text{diag}\left(\frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right)\mathbf{W}^{(l)}, \quad \overline{\mathbf{b}}^{(l)} = \text{diag}\left(\frac{\gamma}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right)\mathbf{b}^{(l)} + \left(\beta - \frac{\gamma\mu_{\text{mov}}}{\sqrt{\sigma^2_{\text{mov}} + \epsilon}}\right). \tag{38}$$

3.4.4 Testing spiking neural networks

It is crucial and foundational to machine learning when constructing models to test how well they generalize on new data that the model has not been exposed to before. When testing on a well-balanced catagorical dataset, one might simply want to measure the accuracy of the model. For neural networks trained with softmax activation in the last layer, the output of the model can be seen as probability distribution of the input belonging to a certain class. Although this interpretation is useful, it is enough to take the argmax of the logits (inputs to the softmax) when trying to predict the most probable class. For spiking neural networks there are different options for determining the prediction of the network. One might predict the class of the first neuron that spikes if the time to prediction is critical. However, this method might not yield the best result as noise in the input signal might lead a neuron, not representative of the true class, to spike first. To evaluate our spiking network, we predict the class of the neuron that had the most spikes in a period of one second.

3.5 Predictive Coding

Predictive coding is hierarchial theory of perception in the brain where higher level cortical areas "explain away" lower-level visual input and only discrepancies hereof are propagated up for further processing. The theory sparked great interest in the neuroscientific community when Rao and Ballard published their seminal 1999 paper [15] showing how hierarchial predictive coding models were able explain neurological phenomenon like endstopping and other extra-classical receptive field effects where the surroundings modulate the response of a receptive field. In the predictive coding theory, the brain is seen as a generative model, able to restore the visual input of the retina by modelling the three dimensional structure of the world. This could also be formulated as inferring the underlaying hidden state of the world, s_i, causing the sensations, o_i, by modelling $p(o_i|s_i)$. Hence, predictive coding share core ideas with the bayesian brain theory. Though predictive coding was originally only ment to explain the perception, it has been suggested as unifying theory of the brain [13]. This section will focus on the correspondance between a special instance of predictive coding dubbed zero inference-learning (Z-IL) where the minimization of error units is equivalent to back-propagation as shown by Song et al [17].

3.5.1 Predictive coding networks and inference learning

Predictive coding networks are very similar to feedforward neural networks but include error nodes, $\epsilon^{(l)}_{i,t}$, between value nodes (as illustrated in figure 4) that encode the difference between a value node, $x^{(l)}_{i,t}$, and its prediction, $u^{(l)}_{i,t}$:

$$\epsilon^{(l)}_t = \mathbf{x}^{(l)}_t - \mathbf{u}^{(l)}_t, \quad \mathbf{u}^{(l)}_t = \mathbf{W}^{(l)}\sigma(\mathbf{x}^{(l-1)}_t) + \mathbf{b}^{(l)} \tag{39}$$

Inference learning (IL) is a particular algorithm introduced by Song et al. [17] to learn the parameters of predictive coding networks for both supervised and un-supervised learning

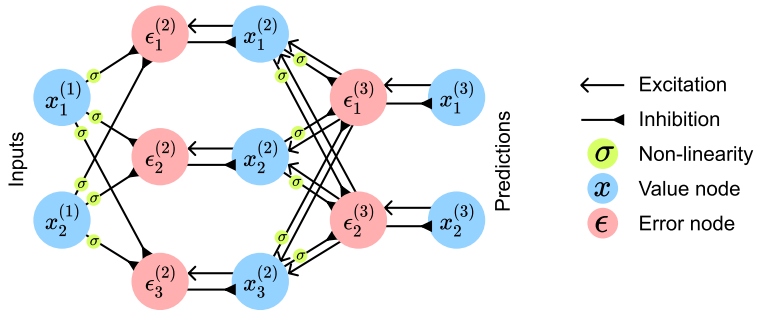


Figure 4: Predictive coding network with 2 input nodes, 3 hidden nodes and 2 output nodes.

tasks. The algorithm runs in two phases; an inference phase and a learning phase. To train the network supervised, the input and target nodes are clamped to the input and target variables that the network should learn a mapping between. Then in the inference phase, value nodes are inferred such that they minimize the total free energy of the network:

$$F = \sum_{l=2}^{L} \frac{1}{2} \epsilon^{(l)^T} \epsilon^{(l)}. \tag{40}$$

To minimize the error nodes, the value nodes are updating according to the following rules;

$$\dot{\mathbf{x}}_t^{(l)} = \begin{cases} 0 & \text{if } l = 1 \\ -\epsilon_t^{(l)} + \sigma'(\mathbf{x}_t^{(l)})(\epsilon_t^{(l+1)} \mathbf{W}^{(l+1)}) & \text{if } l = 2 \dots L-1 \\ -\epsilon_t^{(l)} & \text{if } l = L \text{ during prediction} \\ 0 & \text{if } l = L \text{ during learning} \end{cases} \tag{41}$$

using Eulers method such that $\mathbf{x}_{t+1}^{(l)} = \mathbf{x}_t^{(l)} + \gamma \dot{\mathbf{x}}_t^{(l)}$ where γ is the integration step size. Once value nodes have converged to an equilibrium at time $t = T$, the parameters are updated to yet again minimize the free energy:

$$\Delta \mathbf{W}^{(l)} = \alpha \epsilon_T^{(l)} \sigma(\mathbf{x}_T^{(l-1)}), \quad \Delta \mathbf{b}^{(l)} = \alpha \epsilon_T^{(l)} \tag{42}$$

where α is the learning rate. A possible source of confusion when coming from feedforward neural networks is that predictive coding networks do not propagate activity directly from value node to value node. Instead, value nodes should be considered free variables that are updated according to equation 41. Interestingly, after the inference phase comes to an equilibrium where $\dot{\mathbf{x}}_T^{(l)} = 0$ and all error nodes become zero, the value nodes will correspond to the activity of the neurons in forward prediction of an classic ANN; $0 = \epsilon_T^{(l)} = \mathbf{x}_T^{(l)} - \mathbf{u}_T^{(l)} \Rightarrow \mathbf{x}_T^{(l)} = \mathbf{u}_T^{(l)}$, if only the input node is clamped. Even more intriguing is the fact that the error nodes will become approximately equal to the error terms in the back-propagation algorithm (equation 8) when both input and output nodes are clamped

$$\epsilon_T^{(L-1)} = \sigma'(\mathbf{x}_T^{(L-1)}) \left(\mathbf{x}_T^{(L)} - \mathbf{u}_T^{(L)} \right), \quad \epsilon_T^{(l)} = \sigma'(\mathbf{x}_T^{(l)}) \left(\epsilon_T^{(l+1)} \mathbf{W}^{(l+1)} \right). \tag{43}$$

Here, $\mathbf{x}_T^{(L)} - \mathbf{u}_T^{(L)}$ is the derivative of the squared loss $\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$.

3.5.2 Softmax and cross-entropy loss in inference learning

How do we do this?? We have to redefine our errors to something smart. We have to use softmax and maybe change the weight updates.

3.5.3 Z-IL and equivalence to back-propagation

Song et al. takes this a step further by showing that a special variant of inference learning, zero-divergence inference learning (Z-IL) gives rise to the exact same weight updates as the back-propagation algorithm [17]. They show this by first assuming that every value node, $x_{i,t}^{(l)}$ and prediction $u_{i,t}^{(l)}$ is equal to the corresponding forward activity of the equivalent ANN. This is naturally satisfied when the input node is clamped to the input value of some training pair and the network is relaxed to an equilibrium. This corresponds to the forward propagation necessary in back-propagation algorithm. The output node is then fixed to the output value of the training pair after which each value node is updated to minimize F according to equation 41 with integration step size $\gamma = 1$. At last, the weights are scheduled to be updated according to equation 42 at time $t = l$.

3.5.4 Predictive coding networks for classification

3.5.5 Biological plausability

- Write about the biological plausability.
- Update rules are now local, but still require symmetric weights?
  - Maybe this could be solved with local random feedback alignment?

3.5.6 Variational free energy

Why is this relevant?

3.6 Variational Inference

Kriston paper about

3.7 Energy Based Models

- What can they be used for in general?
- Is predictive coding an instance of this?
- How do they handle adverserial attacks?

4 Results

5 Discussion

6 Conclusion

References

[1] Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015.

[2] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, 1998.

[3] Sander M. Bohté, Joost N. Kok, and Han La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN*, 2000.

[4] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *CoRR*, abs/2005.04168, 2020.

[5] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

[8] Donald O. Hebb. *The Organization of Behaviour - A neuropsychological theory*. John Wiley & Sons, Inc., 1949.

[9] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. 2015.

[10] Hunsberger, Eric. *Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition*. PhD thesis, University of Waterloo, 2018.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Beren Millidge, Anil Seth, and Christopher L Buckley. Predictive coding: a theoretical and experimental review, 2021.

[14] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks, 2016.

[15] Rajesh Rao and Dana Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2:79–87, 02 1999.

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[17] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579. Curran Associates, Inc., 2020.