Eötvös Loránd University
Faculty of Informatics

LEVERAGING ZERO-KNOWLEDGE SUCCINCT ARGUMENTS OF KNOWLEDGE FOR EFFICIENT VERIFICATION OF OUTSOURCED TRAINING OF ARTIFICIAL NEURAL NETWORKS

P. Burcsi
Dr. Ir. at ELTE
A. Peter
Dr. Ir. at UTwente
A. Szabo
Msc. at E-Group

M.J. van de Zande
Computer Science

Budapest, 2019.

ABSTRACT

Incremental innovations have led to practical implementations of both Artificial Neural Networks and Succinct Non-interactive ARGuments, this thesis explores the practicality of using these SNARGs to verify outsourced training of an ANN. This training algorithm presents a particular case of Verifiable Computation as it relies on large quantities of input data, floating point arithmetic and parallel computation. Decomposition of the core concepts ANNs and SNARGs to their elementary building blocks allows the identification of imposed constraints mapped to the case of outsourced training and existing practical implementations of proof systems. The verdict separates the two computations and postulates how the Linear Probabilistically Checkable Proofs fit the inference computation and how the Interactive Oracle Proofs, specifically STARKs, fit the training computation.
This work builds on the the works of Chabenne et al. and Ghodsi et al. that studied the verification of the inference computation and the work of Wu et al. that studies a similar training algorithm in the context of DIstributed Zero-Knowledge.

It is my experience that proofs involving matrices can be shortened by 50% if one throws the matrices out.

— Emil Artin [4]

ACKNOWLEDGMENTS

Many thanks to all of my supervisors; Andreas Peter, Aron Szabo and in particular Péter Burcsi. I thank you for your time and valuable feedback. Our bi-weekly meetings were often filled with endless discussions on the theory and most definitely led to new and valuable insights that make this thesis today.

In addition I would like to thank the originator of the topic, István Seres. Eventhough our approaches do not always align your endless optimism is what made our discussions constructive and fruitful.

Thank you, E-Group. In particular, Aron, Antal and Zoltán for the scientific freedom and the experience of working in a Hungarian company that you have given me. This includes the occasional rooftop barbecue.

Last but not least, I want to thank those who supported me during the full four months; my friends back home in the Netherlands, my parents for the support and valuable feedback and ultimately Mareike for the endless stream of support, the occasional push in the right direction and love. Thank you.

CONTENTS

LIST OF TABLES

ACRONYMS

AGM	Algebraic Group Model
AIR	Algebraic Intermediate Representation
ANN	Artificial Neural Network
APR	Algebraic Placement and Routing
CE	Cross Entropy
CI	Computational Integrity
CNN	Convolutional Neural Network
CRS	Common Reference String
DIZK	DIstributed Zero-Knowledge
DMZ	De-Militarized Zone
ECC	Error Correcting Code
FRI	Fast Reed-solomon IOP
GAN	Generative Adversarial Network
GD	Gradient Descent
GPU	Graphical Processing Unit
HC	Homomorphic Commitment
IOP	Interactive Oracle Proof
IOPP	Interactive Oracle Proof of Proximity
IP	Interactive Proof

ITM Interactive Turing Machine

LDE Low Degree Extension

LIP Linear Interactive Proof

LPCP Linear Probabilistically Checkable Proof

LSTM Long Short-Term Memory

MATMUL matrix multiplication

MB-GD Mini-Batch Gradient Descent

MIP Multi-prover Interactive Proof

MLA Levenberg–Marquardt Algorithm

MLaaS Machine Learning as a Service

MLE Multilinear Extension

MPC Secure Multi Party Computation

MSE Mean Squared Error

NIROP Non-Interactive Random Oracle Proof

ROM Random Oracle Model

PCP Probabilistically Checkable Proof

PCIP Probabilistically Checkable Interactive Proof

PCPP Probabilistically Checkable Proof of Proximity

PoK Proof of Knowledge

QAP Quadratic Arithmetic Program

QSP Quadratic Span Program

RAM Random Access Machine

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

R1CS Rank-one Constraint System

SA Simulated Annealing

SGD Stochastic Gradient Descent

SNARG Succinct Non-interactive ARGument

SNARK Succinct Non-interactive ARgument of Knowledge

SRS Structured Reference String

SSP Square Span Program

STARK Scalable, Transparant ARguments of Knowledge

VAE Variational Auto-Encoder

VC Verifiable Computation

1 INTRODUCTION

As our society is capable of gathering increasing amounts of data, to analyse this data is a natural subsequent step. The algorithms used for this analysis require increasingly more computational and memory resources to manage and utilise these growing flows of data. A consequence of this trend is that only a limited number of players in the field have access to this data and likewise, only a limited number of players have the resources, the computational power, required to effectively run these algorithms. Inherent to this trend is that most of the other, smaller, players are not able to profit from the progress and innovation made in the fields of Data Science and Artifical Intelligence.

The first of the two challenges, the fair data availability of data, is a challenge out of scope of this thesis. Primarily because we believe that the solution does not lie within the field of cryptography, but requires a more social approach. For example, comprehensive regulations to determine data ownership, preserve privacy and enforce public data availability.

Different from the first, the field of cryptography might offer solutions for the second challenge. We believe that a market in computational power and corresponding tasks, Data Science specific or not, might balance the distribution in analytical wealth. Products and services like Machine Learning as a Service (MLaaS)[1] have come to rise and speak for the interest in outsourcing machine learning tasks. But these solutions rely completely on trust of the outsourcing party in the computation providing party. Especially when it comes to machine learning problems, optimisation problems that do not always converge, it is close to impossible to verify correct computation without re-executing it.

We consider the outsourcing party to be $\mathcal{V}$, the verifier that requires some form of proof the task was done correctly, and the providing party to $\mathcal{P}$, the prover that will need to convince $\mathcal{V}$ that it has done the task correctly. There are basically two ways to outsource machine learning jobs: the first option for outsourcing a machine learning task is for $\mathcal{V}$ to ask $\mathcal{P}$ to keep training on some specific training set and algorithm until the model converges and the performance satisfies

1 A selection of MLaas providers:
Amazon: https://aws.amazon.com/machine-learning/
Google: https://cloud.google.com/products/ai/
Microsoft: https://azure.microsoft.com/en-us/overview/ai-platform/

some minimum threshold. This requires $\mathcal{P}$ and $\mathcal{V}$ to agree on the edge cases, what happens if the model does never converge, when to stop, a price per iteration, how does $\mathcal{P}$ quantify its work and proof this to $\mathcal{V}$. Following the alternative approach, $\mathcal{V}$ asks $\mathcal{P}$ to execute some specific training algorithm for a predefined number of steps or iterations. This solves the issue with non-converging machine learning algorithms, but $\mathcal{V}$ will still require some proof that the answer is correct and that $\mathcal{P}$ actually did perform the number of executions. The only way for $\mathcal{V}$ to find any inconsistencies is by re-executing the algorithm, which is often impossible considering the limited resources of such parties. The latter describes the current situation most accurate. MLaaS providers, similar to most other cloud solutions, offer their services, keep their own metrics and ultimately charge by these same metrics. It is merely the image, the name, brand and the absence or lack of publicly disclosed scandals, that creates the required trust. Or the absence of an alternative.

In recent years, in particular the past five, have innovations in the fields of Information Theory, Complexity Theory and Cryptography led to practical implementations of a potential alternative: succinct arguments. The notion of Verifiable Computation (VC) and succinct arguments itself exist for a longer time and term the proof systems that allow a prover $\mathcal{P}$ to proof correct execution of some task to a verifier $\mathcal{V}$, such that the running time of verification is significantly shorter than the running time of naive execution of the algorithm.

Theoretical study of these arguments has resulted in proof systems that have the expressiveness that allows provers to prove statements of all problems in $NEXP$, succinctly. In practice, most real use case live in the realm of distributed ledgers proving valid transactions or solvency.

This thesis explores the real practicality of state of the art argument systems in VC, specifically on the use case of verification of outsourced training of an Artificial Neural Network (ANN). This computation is not only chosen due to its modularity and importance to the field of Deep Learning. But in particular because it incorporates a variety of characteristics typical to many computations susceptible to outsourcing, to name a few: most of the arithmetic is done over floating points in contrast to the the finite fields in cryptography, the computation is highly parallelisable and relies greatly on the use of specialised hardware e. g.Graphical Processing Unit (GPU) and finally it processes vast amounts of input data.
In short, this problem defines a typical computation one would like to delegate to an external party but runs, if constraint by a network description and number of iterations, in polynomial time.

1.1 RESEARCH QUESTIONS

To formalise and structure the aim and goal of this research we re-instate the research questions that define this research. The main research question we aim to answer:

MAIN RESEARCH QUESTION :

Can Zero-Knowledge ARguments of Knowledge verify correct training of Artifical Neural Networks fulfilling the completeness, soundness, succinct and zero-knowledge properties?

To achieve this, the following sub-questions will be answered:

SQ 1 What are the essential building blocks of the training algorithm of an Artificial Neural Network and what are the constraints imposed?

SQ 2 What are the essential building blocks of succinct arguments and what constraints do these impose on efficient verification of the computation of Artificial Neural Networks?

1.2 APPROACH

This thesis aims to explore the practicality of state of the art solutions on a particular use case. Through definition of our interpretation of *practical* as part of our problem statement in section 1.4 we aim to exclude further subjective decisions and constraints. From this problem statement this research starts with decomposition of both concepts ANN and proof systems to identify their essential building blocks and consequently the theoretical and/or practical allowances and constraints these blocks impose on a potential solution. These theoretical and practical possibility and impossibility-results are then combined to construct the intersection of practical argument systems that allow for succinct verification of outsourced training of an ANN.

1.3 OUTLINE

The next section, the problem statement, defines our definition of practicality and the constraints imposed by this definition of the case study. Then chapter 2 elaborates on related work. Chapter 3 answers sub-question 1 through the introduction and decomposition of ANNs. Starting from an intuition on how machines learn, a definition of the fundamental algorithms that make the model and the explicit definition of the computation that requires verification with its constraints. Chapters 4 and 5 answer sub question 2 through the introduction and decomposition of the concept of succinct arguments. The first starts with an intuition of proving, introduces the formal definition of

essential properties and defines the three information theoretical proof systems that form the core of practical implementations. The second starts with a remark on the proposed categorisation followed by the introduction of the conceptual frontend and backend of practical proof systems. The former reduces generic problems in for example high level code to circuits and constraint satisfaction problems that serve as a generic representation of computation and as such as the input for the backend. The backend combines the cryptographic primitives and the information theoretical proof systems to create practical proof systems to prove computation on presented problem. In particular we outline some of the fundamental and state of the art backend implementations and their properties to finally collate to a preselection on potentially suitable proof systems. Chapter 7 combines the findings on Artificial Neural Network and proof systems, elaborates on the conflicting constraints to ultimately answer the main research question. We conclude with a recommendation for those in need of a practical solution in this use case, elaborate on potential future works.

1.4 PROBLEM STATEMENT

In this thesis we consider the verification of outsourced training of an ANN as a prime example of a computation that is not particularly complex in terms of Complexity Theory but nonetheless requires significant amounts of *parallel* computation, additionaly it processes floating points and typically significant amounts of data. A typical computation to outsource one could say.

We are aware of the fact that this problem is strategically chosen primarily in the interest of its properties and therefore point out that there exist a variety of other computational problems that naturally match the constraints imposed by these succinct argument systems. A prime example of such a computation is the transaction verification in ZCash[2].

This particular use case comes with a context and use in practice and also the term ”practical” is rather subjective, given both of these pliable characterisations of constraints a more strict definition of the case and practicality is given by:

1. The verifier runs the verification procedure in less time than it would run the naive computation.
2. The prover runs the computation and proof generation with reasonable low overhead, in quasilinear time with respect to the naive computation.

2 For more information on this: https://z.cash

3. The prover can use the parallelisation of GPUs for the ANN computation and proof generation.

The first requirement follows directly from the definition of VC and is a straightforward requirement for an outsourced computation in practice. The second is not as self-evident as the first as in most VC-schemes the prover is considered to have unbounded computational power. We limit the additional cost of proof generation to one that is approximately a constant factor larger than the computation itself. Additionally the prover has, inherent to the computation of these networks, GPUs at its disposal which it should be able to use for the ANN related computation and, to fulfil the second requirement, the proof generation as well.

2

RELATED WORK

A multitude of prior research have suggested the use of proof systems to verify outsourced computation. One prime example is the work by Babai et al. 1991[6] that describes how a two-prover interactive proof systems allows for computation to be outsourced to a cluster of untrusted machines but also for the result to be verified efficiently: *"In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware"* .

Termed Verifiable Computation, Delegation of Computation, Execution or Computational Integrity or Certified Computation - each of these definition describes the same concept of a two party system; having a powerful prover and an efficient verifier that with the help of randomness, interactivity or proof structure come to correct result of the computation and corresponding proof of correctness. The verifier verifies the proof efficiently, in less time than the time required to execute the computation herself to ultimately accept a valid proo with overwhelming probability and accept a false proof with negligible probability.

Parallel to the incremental innovations in proof systems, the field of data science or more specifically artificial neural networks experienced similar progress. The advances in this field resulted in new and more efficient algorithms to predict and classify that require significant amounts of resources still.

In recent years MLaaS has gained popularity and is now supported by most cloud service providers (sources). But the trust in a party to present correct results is currently primarily based on the name of the company of a company and their assessment of the financial impact a potential scandal might have.

To the best of our knowledge there exist only three works that have effectively looked into the computational verification of artificial neural networks via proof systems. Because we distinguish two components in the ANN algorithms we categorise related work following the two categories.

INFERENCE , also known as feed-forward or classification, uses the weights and topology of a prior trained ANN to label input to the algorithm.

TRAINING , also known as back-propagation, consists of two steps that follow inference, starting with the evaluation of an error or

the cost of the result and consecutively propagate the error back through the network to update its internal state.

Chabenne et al. 2017[19] focus on the inference part alone. The work constructs a system that uses proof composition to separate general purpose VC from specialised embedded efficient VC. In the context of ANN inference the matrix multiplication is delegated to the proving and verification of the more efficient *Sum-Check*[40, 51] protocol, made non-interactive with an algebraic hash function and the Fiat Shamir Heuristic. All other computations including the verification of the Sum-Check are proven and verified by Pinocchio[43], a general purpose proof system.

Ghodsi et al. 2017[27] focus on the inference part as well. This work describes a specialised proof system called SafetyNets that builds on Interactive Proof (IP) system GKR[29] and uses efficient Sum-Check[51] for matrix multiplication as well. The system limits itself to proving and verification of deep neural networks that can be represented as arithmetic circuits over finite fields. Besides the dense network structure of ANNs this work also implements pooling layers to reduce the network size and to help prevent overfitting. Instead of the common ReLu activation function this system uses x^2 demanding finite fields with large primes. Through scaling and quantisation a similar network trained on floating point values can be transformed to a feed-forward network that allows for verification.

Wu et al. 2018[56] focus on the training part but with respect to a different algorithm and thus a different computation, in particular linear regression and covariance matrix multiplication. This work describes a new system DIstributed Zero-Knowledge (DIZK) that leverages parallelisation in the proving algorithm of the Succinct Non-interactive ARgument of Knowledge (SNARK) by Groth[32] to reduce the limitiations of this, and similar, monolithic systems. They manage to improve the 1 ms per gate to 10 μs per gate and expand the upper bound of previously 10-20 million gates per statement to multiple billions.

3

ARTIFICIAL NEURAL NETWORKS

3.1 HOW MACHINES LEARN

Machine learning is the field of study that aims to find algorithms that make machines learn from prior experience. In this context learning means to reduce the error with respect to a specific goal, an optimisation problem. One particular algorithm that has gained significant attention and led to a multitude of applications in recent years, is the ANN introduced by McCulloch and Pitts in 1943 [46].

This early work describes how the mathematician and neuroscientist aim to mimic neurons of the human brain in a mathematically model. This model consisted of multiple layers of artificial neurons that based on some binary input compute an aggregation $g(x)$ and activation $f(g(x))$ successively to come to some binary output defining one of the inputs for the neurons in the next layer. These artificial neurons, although not accurate, mimic the biological neurons with respect to the dendrites (input), soma (aggregation), axon (activation) and synapse (output). In the following decennia algorithms like back-propagation and stochastic gradient descent, but also hardware improvements that allow for parallelism e.g. GPUs, led to successful training of these networks. Networks that are now progressively used in applications in everyday life.

3.2 ARTIFICIAL NEURAL NETWORKS

More formally, an ANN tries to approximate an unknown function t that represents the "truth" on some input x and does this by testing an hypothesis h on the inner state of the network, prior experience, and that same input x. The learning happens in three steps; first an evaluation of the hypothesis $h(x)$, also known as feed forward, results in an output $\hat{y}$, secondly an evaluation of some error function E that compares the ground truth $t(x)$ and the output $\hat{y}$ which results in an error and finally an update of the internal state through back-propagation of this error. This final step is what improves future hypothesis and represents the learning of an ANN.

This process can be split up in two categories based on their application; inference and training, step 1. The former feeds the input to the network forward allowing for classification and prediction while the

latter calculates the error of the inference result and propagates this error back through the network directly updating the internal state to improve future outputs, step 1, 2 and 3.

The artificial neural network itself consists of a set of layers with an aggregation and activation function per node, a set of weights that connect the previous layer to the next layer and finally an error function to determine the deviation from correct results.

3.3 DENSE NEURAL NETWORK

A simple example of a network is a fully connected three layer network with one input layer, a hidden layer and an output layer respectively. This network is often used as an example to classify the written digits 0-9 of the MNIST data set. With 784 nodes in the input layer namely the 26 x 26 pixels of the written digits, 300 nodes in the hidden layer an arbitrary number in between 10-784 and 10 nodes in the output layer each representing one of the digits 0-9.

An image of a written digit assigns a byte value to each of the 784 input nodes which in turn feed these values multiplied with their corresponding weights to each node in the hidden layer. For each of the nodes in the hidden layer computes the aggregation function on the multiple outputs of the previous layers including an input independent value, called the bias, and computes a single result through the non-linear activation function. Similar to the previous layer, inputs to the output layer are the products of outputs and their corresponding weights of the previous layer and aggregated to come to a single value per node. Often the label associated with the node with the maximal value is chosen as the prediction or classification result.

In the context of supervised learning, the learning situation where we have a correct label for each of the inputs, the error function computes the error based on the difference between the correct and the predicted label. This error value or cost is then propagated back through the network updating the weights proportional to its significance in a misclassification and a predetermined learning rate. These algorithms that define the significance based on the aggregation, activation and error functions generally use some form of derivative to move towards an optimum, a prime example is called gradient descent.

3.4 COMPLEX NETWORKS IN PRACTICE

This example gives a general idea on the inference and training algorithms and a basic design and topology of an artificial neural network. But in practice the flexibility in choice of functions, layers and topology of networks have lead to significantly more complex constructs. Following the overview of common network structures by Salvaris et al.[47] and a generalisation of the works described by Schmidhuber[48] we describe four general constructs that have led to applications in image classification, object detection, speech recognition and natural language processing. We distinguish Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Generative Adversarial Network (GAN) and Variational Auto-Encoder (VAE).

3.4.1 *Convolutional Neural Network*

Convolutional Neural Networks[25] have been proven to be successful in the use cases of a.o. image classification and object detection. These networks are similar to the simple example but generally consist of more layers, of which at least one implements a convolutional layer. Convolutional layers take advantage of the context, surrounding pixels, of each pixel similar to how we classify furry animals when we see a furred paw. Via a range of convolutional filters, kernels, and limited perceptive fields, sliding windows, this network can cross-correlate the pixel values of (parts of) some image and reduce them to features that are in turn used for classification. To reduce the complexity of the network so called pooling layers are introduced to reduce the dimensionality. A pooling layer reduces a cluster of outputs to a single maximal or average value.

A CNN can be seen as a composition of two separate networks:

FEATURE EXTRACTOR a dense network using the convolutional and pooling layers to detect and aggregate features, a more comprehensive representation of the input.

CLASSIFIER another dense network but similar to the simple network example that classifies input based on the extracted features of the feature extractor.

3.4.2 *Recurrent Neural Network*

RNNs[45] have been proven to be effective in use cases that have recurring or sequential input data. Good examples are applications such as natural language processing and time series analysis. What differentiates this network structure from the simple example or the

other networks is that the inference algorithm of the network is not feed-forward but allows for feed-back loops. This results in a network that uses a previous state with the current feature vector to create some effect that is often referred to as "memory".

3.4.3 *Generative Adversarial Network*

GANs[31] have been proven in a variety of translation problems such as text-to-image creation, image-to-image creation or enhancement. The adversarial aspect of its name originates from the two competing networks that make up the system. We recognise two networks; a generative network that aims to generate some output or label based on some input and a discriminative network that aims to distinguish between the generated input and the real input. The success of one or the other is again propagated back through the network depending on either error function. An interesting and recent result is the StyleGAN[1][37] that is trained to generate images fulfilling specific characteristics.

Similar to the CNN, the GAN can be seen as the composition of two seperate networks:

GENERATOR a dense network that is trained to generate realistic output from a random input.

DISCRIMINATOR another dense network but similar to the simple network example that is trained to classify whether its input is a generated or a real input.

3.4.4 *Variational Auto-Encoder*

VAEs[39] have proved their effectiveness in use cases like anomaly detection or recommender systems but also as an essential part of other networks as method for dimensionality reduction. What distinguishes this type of network from the other networks is to be found in the approach to the training of the network. Instead of input data and labels these networks are tasked to generate the input from its input. The networks are designed with a restriction that prevents simply copying the input data to the output layer. This restriction is generally realised through a lower dimension hidden layer that has to encode the essential independent features in separate neurons to be able to recreate the output.

1 A prime example of an implementation can be found here: https://www.thispersondoesnotexist.com and shows a randomly generated face at each page load.

Combinations of these networks exist as well, a prime example of such a composition are Long Short-Term Memory (LSTM) networks[34] that are a combination of a CNN and a RNN. This combination creates a network suitable for object detection and image classification on single images but also on videos, essentially image streams, as it uses both new and prior input to infer the next output.

3.5 DEFINITION OF COMPUTATION

Each of the previous complex networks relies to a great extent on the composition of multiple dense layers or more abstract, multiple networks. An ANN G takes a single input x, vector $\vec{x}$ or batch X and evaluates to a prediction or classification $\vec{y}$, inference. This evaluation is then compared with some label $\hat{\vec{y}}$ and results in a cost, also error, that is then propagated back through the layer and network to update the inner state, training. The updates of each element are proportional to the influence of each element on the eventual error of the inference computation.

We define a network G to be the set of L connected layers and the cost function C to determine the error. Each of the layers is defined by the tuple of weights and the activation function of that layer, the weights define the influence of the $m^{(i-1)}$ outputs of the previous layer on the $m^{(i)}$ artifical neurons in this layer. Here $m^{(i)}$ denotes the number of neurons in layer i.

$$G = \left(C, \left\{L^{(1)}, \ldots, L^{(L)}\right\}\right) \tag{3.1}$$

$$L^{(i)} = \left(W^{(i)}, \sigma^{(i)}\right) \tag{3.2}$$

Where the input, output, label and state are defined by:

$\vec{x} \in \mathbb{R}^{m^{(0)}}$	the input
$X \in \mathbb{R}^{m^{(0)} \times n}$	batch input with batch size n
$\vec{y} \in \mathbb{R}^{m^{(L)}}$	the output
$\hat{\vec{y}} \in \mathbb{R}^{m^{(L)}}$	the output label
$W^{(i)} \in \mathbb{R}^{m^{(i-1)} \times m^{(i)}}$	the weight matrix of layer i
η	the learning rate

And the thus the corresponding cost and activiation functions are defined by:

$C : \mathbb{R}^{m^{(L)}} \times \mathbb{R}^{m^{(L)}} \to \mathbb{R}^{m^{(L)}}$	cost function
$\sigma : \mathbb{R}^{m^{(i)} \times n} \to \mathbb{R}^{m^{(i)}}$	activation function

The exact composition of the layers including the specific cost and activation function determine for a large part what the computation looks like under the hood. For this reason we will have a closer look into the layer composition and network composition and define their

respective computations in terms of matrix multiplications, both dot products "$\cdot$" and the element-wise Hadamard product "$\circ$".

3.5.1 *Layer Composition*

Each layer $L^{(i)}$ is defined by its weights and activation function. The weights connect this layer to the previous layer and the activation introduces non-linearity to the network which transforms it in a model that is able to describe more complex relations over a linear regression model. There is a wide variety of activation functions available therefore we restrict the selection to the the most generic and common functions used in practice. We distinguish five different activation functions: the sigmoidal function (equation 3.3), softmax (equation 3.4), hyperbolic tangent function (equation 3.5), Rectified Linear Unit (equation 3.6) and the unconventional x squared (equation 3.7). We will denote the activation function $f(x)$ with σ if there is no need for further specification.

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

$$f(x) = \sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \quad \text{for } i = 1, \ldots, k \text{ with } k = m^{(L)} \tag{3.4}$$

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{3.5}$$

$$f(x) = ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{3.6}$$

$$f(x) = x^2 \tag{3.7}$$

Also we will not consider any aggregation functions $h(x)$ different from the weighted sum inherent to the dot product between the weights and output.

Feed-Forward

The feed-forward computation consists of the weighted sum and the elementwise activation per layer. So when we consider a a single neuron this is defined by;

$$\begin{aligned} a^{(l)} &= \sigma\Big(\sum_{i=0}^{k} w_i^{(l)} \cdot a^{(l-1)}\Big) \quad \text{with } k = m^{(l-1)} \\ &= \sigma(\vec{w}^{(l)} \cdot \vec{a}^{(l-1)}) \end{aligned}$$

and generalises to a dot product and elementwise activation with respect to full layer. For completeness we define the activation of this layer based on the previous layer;

$$A^{(l)} = \sigma(W^{(l)} \cdot A^{(l-1)})$$

Back-Propagation

Similar to the feed-forward algorithm the back-propagation algorithm can be reduced to a sequence of tensor products as well. Individually each of these products represents, informally, the proportion to which each of the feed-forward steps contributed to the consequent error. The algorithms used to determine this proportion are called optimisation algorithms. There exist a wide variety of these optimisation algorithms of which Gradient Descent (GD), Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent (MB-GD) are most common[2] and will be considered in this thesis.

All of the Gradient Descent-based methods use the gradient to account for this proportion or influence in each of the subsequent steps. This means that, for the final layer, we compute the gradient of the error function evaluation towards the final activation (defined by equation 3.10) to determine the difference in input that was required for a better result. The resulting error is then propagated further back as the product of the error and the gradient of the activation towards the aggregation (defined by equation 3.11). Finally we propagate the error further back to the activation of the previous layer $A^{(l-1)}$ (defined by equations 3.8 and 3.12) and to the weights of this layer $W^{(l)}$ (defined by equations 3.9 and 3.13).

$$\frac{\partial E}{\partial A^{(l-1)}} = \frac{\partial Z^{(l)}}{\partial A^{(l-1)}} \cdot \frac{\partial A^{(l)}}{\partial Z^{(l)}} \cdot \frac{\partial E}{\partial A^{(l)}} = \delta^{(l-1)} \tag{3.8}$$

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial Z^{(l)}}{\partial W^{(l)}} \cdot \frac{\partial A^{(l)}}{\partial Z^{(l)}} \cdot \frac{\partial E}{\partial A^{(l)}} \tag{3.9}$$

2 It is important to note that there exist alternative optimisation algorithms, to name a few for the interested reader; Simulated Annealing (SA) and Levenberg–Marquardt Algorithm (MLA)

Where the partial derivatives are defined by:

$$\frac{\partial E}{\partial A^{(L)}} = C'\left(y, \hat{y}\right) \tag{3.10}$$

$$\frac{\partial A^{(l)}}{\partial Z^{(l)}} = \sigma'\left(Z^{(l)}\right) \tag{3.11}$$

$$\frac{\partial Z^{(l)}}{\partial A^{(l-1)}} = W^{(l)} \tag{3.12}$$

$$\frac{\partial Z^{(l)}}{\partial W^{(l)}} = A^{(l-1)} \tag{3.13}$$

As such, the composition of these equations results in the delta towards the previous layer and to the internal state of the model, the latter is essential to the learning of these ANN models. Updating the internal state, the weights, of the model to reduce the error on future predictions. The weights are updated corresponding to the proportional error scaled by a learning parameter called the learning rate.

$$W^{(l)} = W^{(l)} - \eta \frac{\partial E}{\partial W^{(l)}} \tag{3.14}$$

3.5.2 *Network Composition*

Each network G is defined by its cost function and its layers. Similar to the activation functions for each layer each network has a cost function and again there exists a wide variety of functions to choose from. Also in this case we limit our scope to the most generic and common cost functions in practice. We consider both the Mean Squared Error (equation 3.15) and the Cross Entropy (equation 3.16) cost functions.

$$C_{MSE}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}(y_n - \hat{y_n})^2 \tag{3.15}$$

$$C_{CE}(y, \hat{y}) = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}} \quad \text{for } i = 1, \ldots, k \text{ with } k = m^{(L)} \tag{3.16}$$

Feed-Forward

Feed-forward generalises well to a full network as only the input and the weights contribute to the final result. This means that the inference computation can be defined by the following sequence of tensor products and element-wise activations.

$$\vec{y} = \sigma(W^{(L)} \cdot \sigma(W^{(L-1)} \cdot \ldots \sigma(W^{(2)} \cdot \sigma(W^{(1)} \cdot \vec{x})) \ldots))$$

Back-Propagation

Back-propagation generalises to a full network as well but at a greater cost. Where in the feed-forward computation the weight tensors are the only data stored in memory the back-propagation computation requires the $A^{(l)}$ and $Z^{(l)}$ to be stored or re-computed for each of the layers. The computation itself does generalise well and is defined by the following sequence of tensor products and element-wise function evaluation, specifically the derivative of the cost and activation functions.

$$\delta^{(1)} = \left(W^{(1)} \cdot \ldots \left(W^{(L)} \cdot C'\left(y, \hat{y}\right)\right) \circ \sigma'\left(Z^{(L)}\right) \ldots\right) \circ \sigma'\left(Z^{(1)}\right)$$

Similar to this computation is the computation that updates the weights for each layer. On the way propagating back, the algorithm can update the weights of each layer using the yet computed gradients of this layer and the activations computed in the feed-forward computation, prior to this back-propagation.

$$\frac{\partial E}{\partial W^{(l)}} = \left(A^{(l-1)} \cdot \delta^{(l)}\right) \circ \sigma'\left(Z^{(l)}\right) \tag{3.17}$$

3.6 COMPUTATIONAL COMPLEXITY

Aside from the exact definition of the computation and its composing elements for implementation, the complexity and inherent cost of the computation are of particular interest in the context of Verifiable Computation. An understanding of the computational complexity of the inference and training algorithms allows for a comparison between the expected cost of native computation and the cost of verification.

Although the optimisation problems core to machine learning algorithms, including ANNs, are assumend to be NP-hard[17] both the inference and training algorithms are, as described in the sections 3.5.2 and 3.5.2, a sequence of dot and Hadamard tensor products. This means that the verification of the inference and training computations, subject to this thesis, are in Pin contrast to the underlying NP-hardoptimisation problem. Specifically, for a simple dense network the complexity of inference the computational complexity is polynomial and bound by $k \cdot \mathcal{O}\left(n^3\right) + k \cdot \mathcal{O}\left(n\right) = \mathcal{O}\left(n^4\right)$ considering naive matrix multiplication, n neurons per layer over k layers. The

training or back-propation is an order of magnitude larger due to the gradient computation in each of the layers, resulting in $\mathcal{O}\left(n^5\right)$. Convolutional layers will increase the complexity of the network and are therefore often used in combination with pooling layers, to compensate this additional computational complexity. Both inference and training of each of these networks are in P.

A direct consequence of this characterisation is that there does exist a polynomial time algorithm that verifies the computation with a complexity lower than that of the computation itself. This means that a VC system to be of use, will need to have a verification time sublinear to native execution. In all other cases the cost for the verifier to execute the computation herself is more attractive as it is more efficient and is trustless.

3.7 CONCLUSION

In this chapter we abstracted the core computations for both the inference and training algorithms to the extend that the mathematical representation describes most common scenarios of artificial neural networks used in practise. This abstraction helped in the identification of characteristics of named algorithms in the context of their use cases. We give a summary of these characteristics:

1. Practical efficiency of both training and inference rely to a great extent on parallel computations and elementary GPU operations such as tensor products.

2. The input, output and internal state are generally represented as floating points and so is the arithmetic used in computation.

3. The activation functions are generally of high degree or even exponential to ensure the required non-linearity.

4. Both inference and training are in $P$and therefore require sublinear verification with respect to the computation.

With respect to the case subject to this thesis and its practicality we distinguish the sequential and parallel nature of the to be outsourced combination and whether it relies on maintaining intermediate states or not. Specifically:

1. Outsourcing inference is primarily of value in a parallel fashion, specifically on a large number of different inputs and on a pre-trained network because of the static weights. An example of such a use case is the labelling of a large collection of pictures.

2. Outsourcing training is inherently a sequential procedure as the algorithm updates its own state based on new inputs. This results in a computation that, in terms of efficiency, relies on

storing intermediate values of the computation for reuse later in the computation.

4

INFORMATION THEORETICAL PROOF SYSTEMS

Core to the design and implementation of argument systems for Verifiable Computation are information theoretical proof systems. These systems integrate a prover and verifier algorithm and use randomness, interactivity or a proof structure to convince the verifier to accept a valid proof or, with non-negligible probability, to decline an invalid proof. This chapter will recall the notion of formal proofs, extend this notion to probabilistic proofs and elaborate on how the randomness, interactivity and proof structure contribute in the construction of succinct and efficient proofs, in particular; Interactive Proof, Probabilistically Checkable Proof, Interactive Oracle Proof and their non-interactivity counterparts in the Random Oracle Model.

4.1 INTUITION ON PROVING

To present an intuition and to elaborate on the properties we require from mathematical proofs we introduce an informal example: Consider a professor and a class of students. At the end of the semester the professor is required to verify whether she was successful in transferring the knowledge to the students and whether her students are capable of reproducing the knowledge. To conduct this test the professor considers four forms of examination that will allow the students to prove to the professor that they possess the required knowledge and the skill to reproduce this later in time. We start with the naive approach and will come back to the three other strategies later in this chapter, when succinctness and efficiency comes to play.

The naive way for the professor to check sufficient skill and knowledge of her students is to ask the students to write a report. This report should contain all the knowledge subject to this course. The resulting report then serves as a proof for the student of being able to reproduce the appropriate knowledge, as was intended by the professor.

This test fulfils two important properties that we require of proof systems:

COMPLETENESS : A student with the appropriate knowledge and skill will most certainly pass the test.

SOUNDNESS : A student without the appropriate knowledge or skill will not pass this test with high probability.

It also shows the impracticality of a naive approach and hints on other favourable properties we will require from a proof system to

make it suitable in the context of verification of correct computation on a artificial neural network. A non-exhaustive list of Informal examples of such properties:

SUCCINCTNESS : A reduction in proof length, one can imagine that the full report detailing all knowledge will most likely be of considerable size and therefore demand a similar portion of, let's say, the inbox or available printing budget.

PROVER EFFICIENCY : The student has to write the full report which takes considerable time and resources.

VERIFIER EFFICIENCY : The professor has to read each of the reports in full to ensure that not one of the students cheated by replacing one of the paragraphs with some Lorum Ipsum.

ZERO KNOWLEDGE : A little far-fetched, but imagine a PhD candidate that includes some additional information, some of his own ideas, in his work. A zero-knowledge proof system ensures the professor does learn that bit of information that tells her whether the candidate has the appropriate knowledge and skill but nothing more than this one bit of information.

4.2 FORMAL PROOFS

In contrast to the student report mentioned in the previous section we consider formal proofs. These mathematical proofs represent an inferential argument that combines the input, axioms and rules of inference to show the validity of a mathematical statement. This inferential argument can be represented as a tree that combines the (inputs and)axioms via inference to the statement, the root. A proof string π of such a proof could be the transcript of the inferential arguments from the root of the tree to the previous layer, recursively, all the way to the input and axioms. A prime example of such a tree is a boolean circuit with boolean inputs and predicate logic, AND, OR and NOT, that evaluates these inputs to a final output that represents the validity of the statement.

Because of our particular interest in the verification of outsourced computation, but also to allow for the zero knowledge property to hold, we focus on decision problems alone. These computational theoretical problems evaluate to either true or false, one bit of information, depending on the input to the problem. Note that the verification of computation is inherently a computational decision problem as we only accept the results true and false. Also note that the verification of the correct output can easily be transformed to a non-deterministic computational decision problem, e.g. $f(x) = y$ via the arithmetic equa-

tion $\exists\, y : g(x,y) = 0$ given $\exists\, y : g(x,y) = f(x) - y$.

More formally: the combination of parameters that define a specific computational problem are referred to as the instance $\mathbb{x}$ and the knowledge or transcript that accounts for the non-deterministic choices in the algorithm is referred to as the witness $\mathbb{w}$, or certificate in complexity theory. The binary witness relation R is defined to be the set of instance-witness tuples that are polynomial time decidable. The formal language L is defined to be the set of instances that have a corresponding witness in the relation R, specifically: $L \triangleq \{\mathbb{x} \mid \exists \mathbb{w}\ (\mathbb{x}, \mathbb{w}) \in R\}$.

In the context of Verifiable Computation is the $NEXP$-complete Computational Integrity language of particular interest. A first variation of this language got introduced as the ''Computationally Sound'' language by Micali[42] and refined to the ''Universal'' language by Barak and Goldreich[7] and finally the Computational Integrity (CI) language by Ben-Sasson et al.[10, 14]. The language, following the definition of the latter works, is defined by the instance-witness tuples under the relation R_{CI} where:

INSTANCE $\mathbb{x} = (M, x, y, T, S)$ is a quintuple that consist of a non-deterministic Turing Machine, an input, an output, a time bound in terms of the number of steps and a space bound in terms of the number of bits respectively.

WITNESS $\mathbb{w}$ is a transcript of the computation, a description of the non-deterministic steps of M to get from x to y.

The Computational Integrity language: $L_{CI} \triangleq \{\mathbb{x} \mid \exists \mathbb{w}\ (\mathbb{x}, \mathbb{w}) \in R_{CI}\}$.

4.3 PROOF SYSTEMS

Up until now we considered static proofs where a verification algorithm verifies the integrity of the proof independent of its origin. These static proofs are inherently transferable, in the sense that the proof can be used by anyone to prove this exact same statement. To argue about transferability and the efficiency of proving and verification we follow the traditional complexity theoretical notion of proofs analogue too that of NP. We define a two-party proof system consisting of a polynomial bound time Interactive Turing Machine (ITM) that runs the proof verification procedure, the verifier $\mathcal{V}$, and an unbound ITM that runs the proof generation procedure, the prover $\mathcal{P}$. This system with the separate prover and verifier creates a model that is closer to the way these proof systems will be implemented in practice. Additionally this allows us to argue about the computational and memory costs for both the prover and verifier as well as the communication between the two parties.

Following the conventions of complexity theory: a language L belongs to the complexity class NP if and only if there exists an efficient algorithm such that $\mathcal{V}$ in the system $\Pi = (\mathcal{P}, \mathcal{V})$ where $\mathcal{P}$ and $\mathcal{V}$ are two, respectively unbound and probabilistic polynomial bount ITMs, such that the following properties hold:

COMPLETENESS $\forall \mathbb{x} \in L : Pr[\mathcal{V}(\mathbb{x}, \pi) \; accepts] = 1$

SOUNDNESS $\forall \mathbb{x} \notin L, \pi^* : Pr[\mathcal{V}(\mathbb{x}, \pi^*) \; accepts] = 0$

The communication, or interaction, in proof systems has first been introduced by Goldwasser, Micali and Rackoff[30] parallel to and independent of Babai[5] in 1985. These new proof systems, coined Interactive Proofs and Arthur-Merlin protocols, are not only interactive but also probabilistic in nature, meaning that there exists a chance the verifier algorithm accepts a false proof. Via interactivity and random challenges, the verifier can query each prover with unique set of challenges resulting in a transcript that is non-transferable due to the random challenges. Furthermore the randomness in the challenges raised the question on what amount of additional information is required to prove the validity of a statement, in other words, what is the knowledge complexity of a proof. The answer to this question appeared to be that to prove a decision problem statement zero additional bits of information are required to make an honest verifier accept the proof, leading to the notion of zero-knowledge proofs.

4.4 EFFICIENT AND SUCCINCT PROOF SYSTEMS

The introduction of interactivity resulted in a variety of new properties such as adaptivity of proofs, non-transferability of proofs, zero-knowledge proofs but most important for this research probabilistic and more efficient proofs. The years after the introduction of IPs, research identified three ingredients to efficient and succinct proof systems: *randomness*, *interactivity* and *proof structure*. Composition of these ingredients lead to the Interactive Proof, the Probabilistically Checkable Proof and the Interactive Oracle Proof that serve as the core information theoretical proof systems of practical implementations. For each of the information theoretical proof systems we introduce:

1. an intuition of the idea behind the system via an example in line with the student-professor example.
2. the essential properties and metrics of each theoretical system with respect to Verifiable Computation.
3. variations on these systems and how these influence the expressiveness and efficiency of the system.
4. transformation of the public coin proof system to a non-interactive variant in the Random Oracle Model.

In order to compare the different proof systems, for example on expressiveness, we denote the appropriate parameters of a proof system $\Pi[r, q, k, a, l, \varepsilon, \delta, tv, tp]$ such that:

$r(\mathbb{x})$	randomness
$q(\mathbb{x})$	query complexity
$k(\mathbb{x})$	round complexity
$a(\mathbb{x})$	answer alphabet size
$l(\mathbb{x})$	proof length
$\varepsilon(\mathbb{x})$	soundness error
$\rho(\mathbb{x})$	robustness parameter
$\delta(\mathbb{x})$	proximity parameter
$tv(\mathbb{x})$	verifier time
$tp(\mathbb{x})$	prover time

represent the bounds of Π for a particular problem instance $\mathbb{x}$. This allows us to argue about the implications and applicability of such a system in reality. An example of a natural question to ask about such a system is whether a proof system has the expressiveness to prove outsourced computation with negligable error and realistic prover and verifier times.

4.4.1 *Interactive Proofs*

An Interactive Proof[5, 30] uses both randomness and interactivity to come to a verdict on the validity of a statement. A high level analogue of this proof system would be an oral exam: The professor asks the student targeted or, to stick with the proof system, random questions and the student has to answer each and every one of them correctly. Note that in this situation, a student with the required knowledge and skill will always pass the test whereas a student lacking the knowledge is likely to fail this form of examination. The chance of a slacking student failing the test increases with the number of questions asked by the professor. Also note that, although the overall proof length, question and answer, is again similar for both parties the total length is considerably shorter for long proofs compared to the naive approach at the cost of a small chance that a slacking student might pass the test.

The formal definition of an Interactive Proof system $\Pi = (\mathcal{P}, \mathcal{V})$ is given by a tuple of two ITMs $\mathcal{P}$ and $\mathcal{V}$ respectively unbound and probabilistic polynomial time bound. such that the following properties hold:

COMPLETENESS $\forall \mathbb{x} \in L : Pr[\mathcal{V}(\langle \mathcal{P}, \mathcal{V} \rangle(\mathbb{x})) \; accepts] \geq \frac{2}{3}$

SOUNDNESS $\forall \mathbb{x} \notin L, \mathcal{P}^* : Pr[\mathcal{V}(\langle \mathcal{P}^*, \mathcal{V} \rangle(\mathbb{x})) \; accepts] \leq \frac{1}{3}$

Where $\langle \cdot, \cdot \rangle(\mathbb{x})$ denotes the interaction between two entities over some instance $\mathbb{x}$.

After the invention of a two-party interactive proof system adding extra provers and exploring the consequences is an evident follow-up step. Interestingly, adding multiple provers and prevention of communication among provers results in non-adaptivity for interactive proofs. Non-adaptivity in the sense that a prover can not effectively adapt its response strategy based on prior challenges. This resulted in Multi-prover Interactive Proof[8] that extend the expressiveness of traditional $IP = PSPACE$[40, 50] to $MIP = NEXP$[6].

A public coin IP can be transformed to a Non-Interactive Random Oracle Proof (NIROP) using the Fiat-Shamir heuristic[22]. In this case the prover executes the interactive protocol and uses a random oracle to determine the verifier's challenges by himself. The randomness is based on the transcript of prior interaction using the instance $\mathbb{x}$. In turn the verifier checks whether the simulated verifier computed the response messages correctly and whether the challenges correspond with the transcript and the random oracle.

4.4.2 *Probabilistically Checkable Proofs*

A Probabilistically Checkable Proof[2, 3], first introduced as the oracle model[23] uses randomness and the structure of a proof to make the verification of the proof more efficient. The structure of a proof can be compared to the abstract, introduction and conclusion of academic papers that adds redundancy and ensures consistency but allows for the reader to get convinced by reading only parts of the work. The professor asks the students to write their reports following these requirements with respect to the structure. Similar to how we verify related work by reading the abstract, introduction, results and conclusion and some figures, the professor can use a similar strategy to verify the knowledge and skill of each student by probabilistically checking parts of the reports. Note that again a student with the appropriate knowledge will pass this form of examination and depending on the interdependency of the report structure a slacking student gets caught with non-negligible probability. Another note, the time it takes for the student to write the report increased slightly due to the redundant parts of the proof and the constraints opposed by the structure. The professor, on the other hand, does not have to read the full report as she can read only parts of the proof and check whether the argument is consistent. The cost to get the proof from the student to the professor increased over the naive approach again due to the redundancy.

The formal definition of the Probabilistically Checkable Proof system $PCP_{\alpha,\beta}[r,q]$ [1] is given by a tuple of two ITMs $(\mathcal{P},\mathcal{V})$, a prover and verifier respectively unbound and probabilistic polynomial time bound. The prover $\mathcal{P}(\mathbb{x},\mathbb{w})$ computes a samplable proof string π such that the verifier $\mathcal{V}(\mathbb{x},\pi)$ accepts or rejects the proof by querying only $q(n)$ bits of the proof using $r(n)$ bits of randomness such that the following properties hold:

COMPLETENESS $\forall \mathbb{x} \in L : Pr[\mathcal{V}^{\pi}(\mathbb{x}) \; accepts] \geq \alpha$

SOUNDNESS $\forall \mathbb{x} \notin L : Pr[\mathcal{V}^{\pi}(\mathbb{x}) \; accepts] \leq \beta$

A public coin Probabilistically Checkable Proof (PCP) can be transformed to a Non-Interactive Random Oracle Proof following the approach of ''CS-proofs" by Micali[42] based on the earlier Merkle proof[41] and an extension of the non-interactive arguments by Kilian[38]. Following the respective approach the prover computes the PCP proof string π, stores this proof in samplable chunks and constructs a Merkle tree, this results in the root of the tree that with the help of the random oracle results in the indices of the chunks that will be queried in the verification process. Finally, the prover sends one proof to the verifier that composes the root of the Merkle tree, the queried chunks and the authentication paths to these chunks. The verifier, who has access to the same random oracle, can efficiently verify the indices with a the Merkle root and a query to the oracle and the Merkle inclusion proofs for respective chunks consecutively.

4.4.3 *Interactive Oracle Proofs*

An Interactive Oracle Proof [9] or Probabilistically Checkable Interactive Proof (PCIP)[44] uses all three, randomness, interactivity and the proof structure and basically generalizes the IP and PCP to create an efficient system. This form of examination can be compared to a combination on the previous two. The professor asks the students to write a small report with the summary, like in with the PCP, on one topic of her choice every week and she verifies the each of the reports by checking this summary. One can compare the appearing random choice of topics by the professor with the random questions of an IP and analogous the report sent back to her as the answer to this question. Note that a student with the required knowledge will pass this examination procedure and depending on the number of topic report iterations and the inter-dependency of the summaries a slacking student gets caught with non-negligible probability. Also note that this does require the student to write this additional section

1 In contrast to the notion specified in 4.4 PCPs have the convention to denote the completeness and soundness error explicitly via α and β respectively

for every report but if not all topics are chosen less than the naive or the PCP. The professor on the other hand only has to choose the topics and assess the reports by reading only part of it.

The formal definition of the Interactive Oracle Proof system $IOP[k, a, l, r, q, \varepsilon]$ is given by a tuple of two ITMs $(\mathcal{P}, \mathcal{V})$, a prover and verifier respectively unbound and probabilitc polynomialtime bound. The protocol consists of multiple rounds of interaction k, each of those starts with a random challenge by the verifier on which the prover responds with a proof π_i. Finally the verifier queries $q(n)$ bits of each proof using the same $r(n)$ bits of randomness for each proof and accepts or rejects the proof, such that the following properties hold:

COMPLETENESS $\forall \mathbb{x} \in L : Pr[\mathcal{V}(\langle \mathcal{P}, \mathcal{V} \rangle(\mathbb{x})) \; accepts] = 1$

SOUNDNESS $\forall \mathbb{x} \notin L : Pr[\mathcal{V}(\langle \mathcal{P}^*, \mathcal{V} \rangle(\mathbb{x})) \; accepts] \leq \varepsilon$

Where $\langle \cdot, \cdot \rangle(\mathbb{x})$ denotes the interaction or transcript between the two entities over some instance $\mathbb{x}$.

A public coin Interactive Oracle Proof (IOP) can be transformed into a Non-Interactive Random Oracle Proof following a combination of the approaches used to transform IP and PCP to their non-interactive variants. Again, the prover simulates the interaction of the protocol and uses the random oracle as the source of randomness for the verifiers queries. For the PCP part, the prover computes the initial proof π_0 based on the randomness returned from a query to the random oracle with the instance as input. From each of these proofs, the prover constructs a Merkle tree of which the Merkle root and the previous proof form, with the help of the random oracle, the challenge for the next round, the IP part. As such the consecutive proofs are linked via the Merkle-Damgard construction and result in a random output that serves as the index for the samplable chunks to be queried in the proofs $\pi_0, \ldots, \pi_k$. For each of the samples the prover provides the Merkle roots and the authentication paths. The verifier can efficiently check whether the instance and the Merkle roots compose to the queried index and can subsequently verify the relevant chunks via their respective Merkle roots and authentication paths.

4.5 ARGUMENT SYSTEMS

Each of the information theoretical proof systems has improved efficiency for either the prover, the verifier, communication or a combination of the prior mentioned. In 1986 Brassard, Chaum, and Crépeau[18] introduced the concept of an argument system that considers both the verifier and the prover to be probabilistic polynomial time bound, in contrast to the all powerful Merlin and the unbound prover in NP.

This alteration of the resource assumptions of the prover and indirectly the malicious prover and adversary, allows for cryptographic primitives to hold that would not be secure against unbound adversaries. Implicitly this means that such an argument system does no longer retain its perfect soundness but computational soundness instead. As a super-polynomial prover will still be able to cheat with non-negligible probability. The introduction of cryptographic primitives in proof systems did not only allow for better efficiency but also introduced novel concepts such as reusability of proofs and public verifiability.

4.6 PROOFS AND ARGUMENTS OF KNOWLEDGE

Proofs or arguments of knowledge are systems that satisfy the knowledge soundness property, this property ensures that a convincing prover also knows the witness. How to test whether an entity "knows" something and what the definition of knowledge is, is a topic on its own. In the context of proof systems knowledge is defined to be information that is stored or can efficiently be deduced or computed from the information available. This deduction, load from memory or computation is called the knowledge extractor E, a polynomial time bound procedure that extracts the designated information from the information available. A prover "knows" the witness and presents a Proof of Knowledge (PoK) if there exists an extractor that can derive $\mathbb{w}$ from the inner state and randomness of a prover if the prover convinces the verifier.

More formally: a proof system is ε-knowledge sound if there exists a probabilistic polynomial time bound extractor with oracle access to a possible malicious prover $E^{\tilde{\mathcal{P}}}$ such that the probability of the extractor extracting the witness from the transcript is greater than the probability of this same prover convincing the verifier in normal execution.

KNOWLEDGE SOUNDNESS : $\exists\ E, \forall\ \mathbb{x} \in L, \tilde{\mathcal{P}} : Pr[(\mathbb{x}, \mathbb{w}) \in R | \mathbb{w} \leftarrow E^{\tilde{\mathcal{P}}(\mathbb{x})}(\mathbb{x})] \geq Pr[\mathcal{V}(\mathbb{x}, \tilde{\mathcal{P}}(\mathbb{x}))\ accepts] - \varepsilon$

4.7 ZERO KNOWLEDGE

When we discussed the invention of IPs we hinted on the notion of zero-knowledge and what this informally means for a proof system. In the context of Verifiable Computation and more specifically with respect to the use case of this thesis this would mean that a prover could classify, for example a set of images, using a secret pre-trained network and proof that the classification was done correctly without disclosing the network.

The formal notion of zero-knowledge is similar to the more intuitive; Everything a malicious verifier learns from its interaction with the proof system could have been deduced without this interaction. Formally: if there exits a probabilistic polynomial time simulator $S(\mathbb{x}, \mathbb{z})$ that with the available public information, the instance and context, can generate a transcript indistinguishable from a transcript between an honest prover and verifier.

$$\text{ZERO KNOWLEDGE} : \exists\, S, \forall\, \mathbb{x} \in L, \mathbb{z}, \mathcal{V}^* : \langle S(\mathbb{x}, \mathbb{z}) \rangle \cong \langle \mathcal{P}(\mathbb{w}), \mathcal{V}^*(\mathbb{z}) \rangle (\mathbb{x})$$

This definition describes the strongest notion of zero-knowledge, specifically Perfect Zero-Knowledge with respect to auxiliary input. Most of the implementations discussed in this thesis satisfy the perfect zero knowledge property hence we do not elaborate on alternative notions that for example, relax indistinguishability to statistical or computational indistinguishability, consider an honest verifier or no auxiliary input at all.

4.8 CONCLUSION

In this chapter, we introduced the basic concepts and definitions of informational theoretical proof systems. Specifically, we introduced and formally defined the notions of completeness, soundness, succinctness, zero-knowledge and what it means for a proof system to be scalable and doubly-efficient. In terms of succinctness and efficiency we discussed the effects of interactivity, randomness and proof structure on the immediate costs for proof generation, proof verification and communication. This resulted in the categorisation of the following three probabilistic proof systems:

INTERACTIVE PROOFS : use interactivity and randomness for random challenges to the prover which results in non-transferable proof transcripts.

PROBABILISTICALLY CHECKABLE PROOFS : use randomness and the structure of proofs to reduce the verification cost significantly at the cost of communication and prover complexity.

INTERACTIVE ORACLE PROOFS : use interactivity, randomness and the structure of proofs to balance the reduction in complexity for verification and proof generation.

For each of these information theoretical proof systems there exists a transformation to a non-interactive variant in the Random Oracle Model, resulting in Non-Interactive Random Oracle Proofs. And finally, we formally define the language of computational integrity that, if proven and verified, ensures correct execution of a particular algorithm.

5 PRACTICAL PROOF SYSTEMS

This chapter aims to elaborate on the exact implementation of the argument systems used today and their relation to the fundamental information theoretical proof systems introduced in chapter 4. We will follow the literature in the separation of the so called frontend and backend. The frontend reduces the problem, for example written in code, to a more generic representation of computation such as boolean or arithmetic circuits. Depending on the backend more structured representations might be required such as Quadratic Span Program (QSP), Quadratic Arithmetic Program (QAP), Square Span Program (SSP) or TinyRAM. The backend uses this input together with an implementation of the information theoretical proof systems to create a proof with the required properties. In some cases additional cryptography is used to enhance or add properties in the final implementation.

Multiple works have hinted at but in particular the work by Bitansky et al.[16] have advocated for a more modular approach specifically separating the information theoretical proof systems from the cryptographic primitives, we endorse the stance on the modular approach but we take a different approach and will use the afore mentioned characteristics of proof systems, *randomness*, *interactivity* and *proof structure* to differentiate categories over the use of primitives.

Via this new and stricter categorisation we elaborate on a selection of the important and novel proof systems that present the state of the art, realised in code. Finally we compare each of the systems on the input format, prover complexity and most important verifier complexity.

5.1 A NOVEL CATEGORISATION

Mentioned briefly is the alternative approach on the categorisation of proof systems in this thesis. The aim to introduce more modularity to the field of proof systems, or how Ishai describes this field; the ''ZK Zoo'' [1] does require more structure for obvious reasons. Simplicity is a first, this helps those new to the field to break down the complexity of systems that often combine concepts from pure complexity theory to cryptography, a pallete not every graduate student has at his or her disposal, let alone the industry. More simple or smaller building

1 Specifically the talk on Compilers for Zero-Knowledge: https://cyber.biu.ac.il/event/the-9th-biu-winter-school-on-cryptography/

blocks with clear properties will help in the focus of future research, for example proving lower bounds or the use of components in different and new contexts. Ultimately leading to more innovation in the field and more efficient proof systems.

An initial differentiation is made in the separation of information theoretical proof systems and the consecutive use of cryptographic primitives to enhance some or all of the properties. Although this proposed categorisation might appear as a natural categorisation, this turns out to be less the case than we would have liked. The challenge with this categorisation emanates in the definition of the categories; what is the information theoretical proof system and where does the cryptography start. The translation from each of the public coin proof systems in section 4.4 to its non-interactive counter part is fully information theoretical but makes use of an oracle, while some of the commit-reveal schemes make use of either information theoretical binding or hiding commitment schemes. Both are examples of systems where it is hard to draw a line between the information theoretical components and the cryptography.

We argue that a categorisation on an abstract level, that of the eventual use case, the implementation and the involved actors, results in a more descriptive and practical categorisation. The information theoretical proof systems in chapter 4 describe generic systems that have the expressiveness to prove certain problems based on the context. The context being; the actors involved and their resource bounds in terms of computation, memory and communication or access to private or public randomness. Given such a description of a proof system, we can argue about its efficiency but more importantly about generic transformations that, independent of the implementation of the proof system can enhance or add new properties. This means we can have two or more bounded parties exchanging, potentially structured, messages.

Among these conceptual systems the exact implementation might significantly deviate. For example for the classification and the context it does not matter whether a partially samplable proof is implemented through Secure Multi Party Computation, Quadratic Arithmetic Programs or Error Correcting Codes as long as it satisfies the constraints imposed by the context. We will see that exactly these three approaches to structured proofs with the help of commitment schemes and efficient reductions build the majority of today's implementations. We will see that all of the efficient systems, in the context of Verifiable Computation we will require both the proof structure and the interactivity to create succinct proofs resulting in a variety of implementations in the categories IP, PCP and IOP.

5.2 FRONTEND: CIRCUITS AND CONSTRAINT SYSTEMS

The definition of the frontend is a means to abstract the exact implementation and inner workings of the proof system from its user or application and present the core constraint satisfaction problem to the proof system, the backend. The frontend is presented with a problem in for example high level code which is then compiled to an intermediate representation that efficiently defines the problem at hand. This intermediate representation is then transformed to its equivalent NPor $NEXP$relation in the language specific to the backend.

These systems will be introduced later but for example, the backend of a QAP-based Linear Probabilistically Checkable Proof (LPCP) or Ligero take a set of 2-degree constraints in the NP-complete language Rank-one Constraint System (R1CS). The front end takes high level code and translates the problem in these constraints via an algebraic circuit. Alternatively the STARK frontend takes its problem defined as an execution trace in combination with the boundary constraints and the transition relation called the Algebraic Intermediate Representation (AIR). This AIR is then translated to the $NEXP$-complete ans chip architecture inspired language Algebraic Placement and Routing (APR). There exist a wide variety of languages and which are excepted by which proof system is dependent on its implementation. The core idea of the frontend is that it compiles a problem definition to it analogue constraint system in the apropriate language.

Algebraic circuits offer a natural way to describe computation and will therefore be used as a means to compare the relative complexity of the different procedures with respect to the computation. Circuit complexity is primarily depend on the number of gates in the circuit C and the depth d in number of layers. Due to specific representations en implementations some additional metrics have a direct influence on the complexity, specifically: the input size n, number of multiplication gates n^*, the number of wire conditions m and the length of the statement l

5.3 BACKEND: IMPLEMENTATIONS

The backend defines the proof system and its proving protocol. It takes a relation in a specific constraint system as an input and generates the proof according to this instance witness relation. The constraint system that defines the relation and the proof system specify the expressiveness of a backend. Ideally these frontends and backends should be interoparable but so far the state of the art follows the monolithic approach with hand tuned problem and proof compilers.
In the following three sub-sections we outline the consequential techniques and innovations that led to implementations in code and detail

a selection of these implementations for each of the categories described in chapter 4.

5.3.1 *Interactive Proofs*

In the context of this thesis matrix multiplication is a computation of particular interest but supplementary a prime example in the comparison of succinctness in non and interactive verification algorithms. Perhaps the most well known algorithm to verify matrix multiplication (MATMUL) is Freivald's algorithm[24].

In short; a prover $\mathcal{P}$ claims that $A \cdot B = C$ with $A, B, C \in \mathbb{F}_p^{n \times n}$. To verify this a verifier $\mathcal{V}$ selects a random vector $\vec{x} \in_R \mathbb{F}_p^n$ and computes $A \cdot (B\vec{x}) = C\vec{x}$ which is essentially three matrix vector products. This results in a proof system with a verifier complexity of $\mathcal{O}\left(n^2\right)$, prover complexity $\mathcal{O}\left(n^{2.3728639}\right)$[2] and a communication complexity of $\Omega\left(n^2\right)$ as the product matrices have to be send to the prover. This is perfectly fine for the "in the head" verifier simulation in proof composition, but is undesirable for practical prover verifier interaction.

An alternative for this is the protocol described by Thaler[51] that preserves the prover and verifier complexity of Freivald's algorithm but reduces the communication complexity to $\mathcal{O}\left(\log n\right)$. This protocol leverages the SumCheck protocol[40] and Multilinear Extensions to realise this improvement.

In short; we consider the same problem, but this time we interpret A, B and C as functions mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n} \to \mathbb{F}$ and as such the product is defined by $\tilde{C}\left(\vec{x}, \vec{y}\right) = \sum_{\vec{z} \in \{0,1\}^{\log n}} \tilde{A}\left(\vec{x}, \vec{z}\right) \cdot \tilde{B}\left(\vec{z}, \vec{y}\right)$. Which directly results in the $\log n$-variate polynomial, input to the interactive SumCheck protocol, $g(\vec{z}) \triangleq \tilde{A}\left(\vec{x}, \vec{z}\right) \cdot \tilde{B}\left(\vec{z}, \vec{y}\right)$. This SumCheck protocol basically forces the prover to compute the equality sum for the verifier. The protocol consists of k rounds where k is equal to number of variables in the polynomial, in this case $\log n$. Each of these rounds the verifier challenges the prover with a random value r_i and subsequently the prover responds with a univariate polynomial that can be evaluated by the verifier. Either with the help of an oracle or via the prover who computes $g(r_1, \ldots, r_d)$ for the verifier. The first and last round acount for all input or all random variables as shown in equation 5.1 and 5.3 respectively, in every intermediate step the verifier forces the prover to compute the values with respect to one additional variable as shown in equation 5.2 such that the verifier comes to a decision in k steps, the exact number of variables. For each of these steps the verifier challenges the prover with a new random challenge and checks 1) whether she got an univariate polynomial of the appropriate

2 Williams algorithm as described in "Breaking the Coppersmith-Winograd barrier"

degree and 2) whether the equality $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$ holds. If the prover passes all k rounds the verifier accepts the proof.

$$g_1(X_1) = \sum_{(x_2,\ldots,x_d)\in\{0,1\}^{d-1}} g(X_1, x_2, \ldots, x_d) \tag{5.1}$$

$$g_i(X_i) = \sum_{(x_{i+1},\ldots,x_d)\in\{0,1\}^{d-1}} g(r_1, \ldots, X_i, \cdots x_d) \tag{5.2}$$

$$g_d(X_d) = g(r_1, \ldots, r_{d-1}, X_d) \tag{5.3}$$

This SumCheck protocol forms the basis for all efficient interactive proof systems, for the first sub-linear system, GKR, but also for the state of the art, Libra. We outline a selection of four works that define todays landmarks when it comes to interactive proof systems.

GKR

The proof system named after the inventors Golwasser, Kalai and Rothblum[29] was the first interactive proof that realised sub-linear verification times. The protocol takes as input a *logspace uniform* arithmetic circuit and verifies the claim via an approach similar to the SumCheck protocol. First it verifies the evaluation of the final layer via a Multilinear Extension and checks whether this satisfies the claimed output. As the verifier is not able to verify this yet it reduces this claim to a claim for the previous layer. Finally, after traversing all layers, the claim is reduced to the input of the computation which is available to the verifier and easy to verify.

This protocol does not have the zero-knowledge property but is highly efficient for data parallel circuits, computations in *NC*.

CMT

A second system named after its originators Cormode Mitzenmacher and Thaler[21] improved on the performance of GKR through the observation that the polynomials in the SumCheck are generally sparse and well structured. Making use of this structure they were able to reduce all of the communication, prover and verifier complexity. Most importantly the prover complexity from $poly(S)$ to $\mathcal{O}(S \log S)$.

Hyrax

Hyrax[55] is the last in a line of increasingly more efficient interactive systems and the first to add zero-knowledge to the acquired properties. The system itself is an improvement of the Giraffe[54] system which in turn is inspired by the Zebra[53], Allspice[52] and T13[51].

Libra

The final work, also the most recent and efficient, still builds upon the GKR SumCheck based system. Libra[57] improved the original GKR protocol, again based on the observation of the sparse polynomials that led to CMT, through the execution of two SumChecks in parallel. This results in a significant efficiency improvement and reduces the prover running time to $\mathcal{O}(S)$. Additionally this work introduces zero-knowledge through random masking polynomials.

SYSTEM	$\lvert\pi\rvert$	$\mathcal{P}$	$\mathcal{V}$
GKR	$d \cdot \text{polylog}(n)$	$\text{poly}(n)$	$(n+d) \cdot \text{polylog}(n)$
CMT [3]	$d \log(C)$	$\mathcal{O}(C \cdot \log C)$	$\mathcal{O}(n + d \cdot \log C)$
Hyrax	$\mathcal{O}(\sqrt{n} \cdot \log C)$	$\mathcal{O}(C \cdot \log C)$	$\mathcal{O}(\sqrt{n} \cdot \log C)$
Libra [4]	$\mathcal{O}(d \cdot \log C)$	$\mathcal{O}(C)$	$\mathcal{O}(d \cdot \log C)$

Table 5.1: A comparison of IP systems in terms of overall proof length $\lvert\pi\rvert$, prover compexity $\mathcal{P}$and verifier complexity $\mathcal{V}$based on the results published in the seminal works[21, 29, 55, 57]

5.3.2 *Probabilistically Checkable Proofs*

To realise sub-linear verification time there is no alternative to the probabilistic checking of structured proofs. This is inherent to the fact that reading the proof alone will already result in a linear time complexity. There are two common approaches to the creation of these structured proofs. The first and best known is *arithmetization*, the reduction of computational problems to an analogue problem in the algebraic domain. A suitable analogue for PCPs are polynomials, polynomials allow for efficient constraint testing and essentially Error Correcting Codes. A second approach is virtual Secure Multi Party Computation where the prover performs the computation virtually, ''in the head" with at least 3 parties. The consistency between the views of two parties ensures the complete and sound-ness properties while the hiding property of the scheme makes it a zero-knowledge -PCP.

With respect to the implementation of these systems we distinguish two types of PCPs and will consider arithmetization and accordingly polynomials to resemble the proof structure of the following PCPs.

The first category are the so called short-PCPs[5] that take a univariate polynomial of relatively low-degree and leverage this low-degree as an Error Correcting Code (ECC) or Low Degree Extension (LDE) to

5 "short" means quasilinear with respect to the circuit size

amplify a potential error. The verification of such a proof essentially consist of two steps;

1. One or a small number of queries to the proof to verify satisfiability
2. A low-degree test to ensure the proof is actually a low-degree polynomial

Although the idea of these systems is simple and clean the practical and especially efficient implementation of low-degree testing is more complex. Solutions to enable this second check to be conducted efficiently are the so called proofs of proximity that, although they can be made non-interactive through the transformations mentioned in section 4.4, are interactive. We will discuss these proximity proofs and respective implementations in more detail in the next subsection on IOPs.

We recall the challenge inherent to the separation of the information theoretical proof systems and the cryptographic primitives in section 5.1. A naive implementation of a system with these two checks would result in a linear proof system. Following the naive way degree test, by querying degree + 1 points, in this case approximately the number of gates in the circuit or the number of constraints. This system would not rely on any assumptions and would hold perfect zero-knowledge without any cryptography, or does it? The polynomial ECC and LDE are essentially an alternate form of Shamir's secret sharing[49]. This shows that a categorisation on related fields of study might not be as expressive and practical as a categorisation based on properties.

The second category is defined by the long-PCP, LPCP or Hadamard PCPs. This category of proof systems has most implementations and as such most practical applications today. Backed by strong communities in the area of crypto currencies and distributed ledgers also the line of research that dictates the course and terminology in the field. A good example of this terminology is the SNARK that implicitly refers to *pre-processing*-SNARKs, the range of works that build succinct arguments from the transformation introduced by IKO.

The first such system was proposed by and named after the originators Ishai, Kushilevitz, and Ostrovsky[35] and designed around the observation that the prover does not have to materialise the full proof, often of exponential size in the short-PCP structure, to commit to it. The protocol uses the fact that due to the highly structured nature of the proof a single encrypted query from the verifier would retain the prover from computation of, for example the full Merkle tree. This does mean that the verifier has to create this encrypted query of proof length resulting in significant additional complexity on the

verifier side and cancel the succinctness of the system. But because the query is only dependent on the computation and hidden due to the encryption it can be reused for multiple proofs that verify the same computation, but on different inputs. This initial constant query can be considered to be the structured Common Reference String (CRS), also Structured Reference String (SRS), that is set in the setup phase of the protocol, and therefore reducing the query complexity form two to one rounds, essentially making it a non-interactive protocol. We call this amortised computation the preprocessing of the protocol that defines the *preprocessing* Succinct Non-interactive ARGument (SNARG) or the knowledge sound, *preprocessing* SNARK.
The main challenges in this setup is how to ensure linearity of the the queries and consistency of the proof(s). Additive Homomorphic Commitment (HC) and bilinear pairings offer a solution for the former by cryptographically constraining the set of valid operations. The latter is can be solved with an interactive round as proposed by Bitansky et al.

The paradigm of information theoretical proof systems and the cryptographic primitives fits best in this category. The work of Bitansky et al. [16] introduces transformations from any PCP to a LPCP or directly to their final information theoretical Linear Interactive Proof (LIP) and in turn a cryptographic transformation that enforces the constraints of the LIP to a privately verifiable *preprocessing* SNARG. Parallel to this reduction they presented transformations to reduce any low-degree LPCP to a low-degree LIP and consecutively a public verifiable *preprocessing* SNARG.

Similar to the improvement of CMT on GKR, Genarro et al.[26] observe the structure in the constraints imposed by the program and were able to leverage this to improve the efficiency of the system significantly. Specifically they found that the constraints where all of the form $Q_i(W) = c_i - W_i = 0$, $Q_i(W) = \left(W_j + W_k\right) - W_i = 0$ and $Q_i(W) = \left(W_j \cdot W_k\right) - W_i = 0$ that allowed them to efficiently reduce the constraints to a single univariate polynomial that vanishes for a correct transcript $C(x, y, W)$. The univariate polynomial serves directly as the arithmetization for the proof itself. This compact encoding of arithmetic constraints has been termed QAP and QSP in the case of Boolean circuits and follows directly from a R1CS.

Pinocchio

Pinocchio[43] was in 2013, still is, the first general-purpose verifiable computation system with an implementation that had faster verification time than native execution of the program. The system comes with both a frontend and backend that translates high level code, e. g. C, to the corresponding arithmetic or boolean constraints, the QAP or QSP and the final ECC used in the proof.

BCTV14

BCTV14[11] proposes the first SNARK that is fully succinct and incrementally computable. It is based on the work of [15] and uses recursive proof composition together with a specification of a Random Access Machine (RAM) to prove and verify a sequence of machine steps incrementally. Each proof proving the correctness of the previous state, memory, and correctness of the next machine step.

Groth16

Groth16[32] describes the current most efficient SNARK with a proof of only three group elements and a verification time proportional to the statement. The system itself takes arithmetic circuits as an input is QAP based and can be instantiated with any type of bilinear pairing.

GM17

GM17[33] introduces the smallest Simulation Extractable-SNARK with the prove minimum of three group elements and two verification equations. This system is based on Groth16, has similar size is not as fast, but comes with better proven security.

SYSTEM	$\|\pi\|$	$\mathcal{P}$	$\mathcal{V}$	CRS
SNARK [6]	$3-8$	$\tilde{\mathcal{O}}(C)$	$\mathcal{O}(1)$	$\mathcal{O}(C)$
Pinocchio	$8\ \mathbb{G}$	$7m + n^* - 2l\ E$	$l\ E,\ 11\ P$	$7m + n^* - 2l\ \mathbb{G}$
BCTV14	$7\ \mathbb{G}_1,$	$6m + n^* - l\ E_1$	$l\ E_1,\ 12\ P$	$6m + n^* + l\ \mathbb{G}_1$
	$1\ \mathbb{G}_2$	$m\ E_2$		$m\ \mathbb{G}_2$
Groth16	$2\ \mathbb{G}_1$	$m + 3n^* - l\ E_1$	$l\ E_1,\ 3\ P$	$m + 2n^*\ \mathbb{G}_1$
	$1\ \mathbb{G}_2$	$n^*\ E_2$		$(n^*\ \mathbb{G}_2)$
GM17	$2\ \mathbb{G}_1$	$m + 4n^* - l\ E_1$	$l\ E_1,\ 5\ P$	$m + 4n^* + 5\ \mathbb{G}_1$
	$1\ \mathbb{G}_2$	$2n^*\ E_2$		$2n^* + 3\ \mathbb{G}_2$

Table 5.2: A comparison of PCP systems in terms of overall proof length $|\pi|$ in number of group elements, prover and verifier complexity, $\mathcal{P}$and $\mathcal{V}$, where E, M and P represent exponentiations, multiplications and pairings respectively and finally the size of the CRS based on the results published in the seminal works[11, 32, 33, 43] in particular the works by Groth

5.3.3 *Interactive Oracle Proofs*

The Interactive Oracle Proof is essentially a generalisation of IPs and PCPs, meaning that these generic proof systems can leverage that bit of interactivity or proof structure they need to be more expressive or

more succinct.

We differentiate between two variations of IOPs the first is based on the proofs of proximity discussed in the previous section. Both of these, Interactive Oracle Proof of Proximity (IOPP) and Probabilistically Checkable Proof of Proximity (PCPP), aim to test the proximity to some ECC like the Reed-Solomon code. The second variation is an example of the "MPC in the head" paradigm that is based on an Interactive PCP, a special case of IOP.

STARK

Scalable, Transparant ARguments of Knowledge (STARK)[14] introduces the first implementation in line with the checks for a short-PCP. It presents both an interactive and a non-interactive STARK. that takes the $NEXP$APR relation as an input to create a Reed-Solomon and verify the low degree via the novel Fast Reed-solomon IOP (FRI) protocol[13].

Aurora

Aurora[12] implements an IOP that takes the NPR1CS relation as an input and solves the univariate analogue of the SumCheck protocol using multiple Reed-Solomon encoded IOPPs.

Ligero

Ligero[1] introduces an implementation in line with ZKBoo[28] and ZKB++[20] that implement a paradigm called "MPC in the head"[36] that, as described in the previous section, allows the prover to virtually execute a Secure Multi Party Computation (MPC) scheme and subsequently commit to the views of two of the virtual participants.

SYSTEM	$\lvert\pi\rvert$	$\mathcal{P}$	$\mathcal{V}$	q
STARK	$\mathcal{O}(C \log C)$	$\mathcal{O}\left(C \log^2 C\right)$	$\mathcal{O}(C)$	$\mathcal{O}(\log C)$
Aurora	$\mathcal{O}(C)$	$\mathcal{O}(C \log C)$	$\mathcal{O}(C)$	$\mathcal{O}(\log C)$
Ligero	$\mathcal{O}(C)$	$\mathcal{O}(C \log C)$	$\mathcal{O}(C)$	$\mathcal{O}\left(\sqrt{C}\right)$

Table 5.3: A comparison of IOP systems in terms of overall proof length $\lvert\pi\rvert$ in number of field elements, prover and verifier complexity, $\mathcal{P}$and $\mathcal{V}$, in terms of field operations and the query complexity q based on the results published in the seminal works[1, 12, 14]

5.4 COLLATION

The previous section pre-selected the most fundamental and state of the art systems that have led to implementations in code. Because

almost all[7] of the systems satisfy the primary properties the research question requires from a VC system we make a more detailed selection. All systems in the selection are zero-knowledge, complete, sound and succinct. The most efficient systems are provided that there is at least one system representing each category and associated technologies and input types. In the case of LPCPs GM17 is chosen over Groth16 because it enjoys better proven security for relatively similar performance. Table 5.4 presents the relative complexity and table 5.5 presents an overview of the properties satisfied fro the selection of systems.

	STARK	AURORA	LIGERO	GM17	LIBRA
$\mathcal{P}$	$\mathcal{O}\left(C \log^2 C\right)$	$\mathcal{O}\left(C \log C\right)$	$\mathcal{O}\left(C \log C\right)$	$\tilde{\mathcal{O}}\left(C\right)$	$\mathcal{O}\left(C\right)$
$\mathcal{V}$	$\mathcal{O}\left(C\right)$	$\mathcal{O}\left(C\right)$	$\mathcal{O}\left(C\right)$	$\mathcal{O}\left(1\right)$	$\mathcal{O}\left(d \cdot \log C\right)$
q	$\mathcal{O}\left(\log C\right)$	$\mathcal{O}\left(\log C\right)$	$\mathcal{O}\left(\sqrt{C}\right)$	$\mathcal{O}\left(1\right)$	$\mathcal{O}\left(d \cdot \log C\right)$
S	-	-	-	$\mathcal{O}\left(C\right)$	$\mathcal{O}\left(n\right)$

Table 5.4: A comparison on asymptotic complexity, specifically prover and verifier complexity, $\mathcal{P}$and $\mathcal{V}$, and the query complexity q based on the actual query complexity in a PCP based system or the proof length, S depicts the setup costs

	STARK	AURORA	LIGERO	GM17	LIBRA
input	APR	R1CS	R1CS	AC	AC
complete	✓	✓	✓	✓	✓
sound	✓	✓	✓	✓	✓
succinct	✓	✓	✓	✓	✓
zero-know.	✓	✓	✓	✓	✓
setup	public	public	public	private	private
assumptions	standard	standard	standard	AGM	q-SBDH
post-quantum	✓	✓	✓	✗	✗

Table 5.5: An overview of the properties of the selected systems based on the seminal works[1, 12, 14, 33, 57]. On the assumptions; AGM and q-Strong Bilinear Diffie-Hellman (q-SBDH)

5.5 CONCLUSION

In this chapter we introduced the frontend and backend of proof systems to abstract the high level problem definition to constraint system reduction from the actual proof systems. Then we argued about

7 GKR and CMT do not fulfil the zero-knowledge property

a slight change in the categorisation of these actual proof systems and follow this novel categorisation to elaborate on the fundamental techniques that make the practical proof systems. Finally we compare a selection of the systems realised in code and find that the properties of interest for this thesis are satisfied in the majority of these systems by default and that therefore the prover and verifier efficiency and the trusted setup become key metrics in a final verdict.

IP and SumCheck-based systems tend to have the lowest prover efficiency and except from Libra do not require a trusted setup

PCP specifically LPCP has the fastest verification time but relies on public key cryptography and a trusted setup of considerable size

IOP presents framework for a variety of techniques to combine interactivity and proof structure that supports short-PCPs but also "MPC in the head". These systems are generally less efficient but are by default transparent, rely on standard assumptions and are plausibly post-quantum secure.

6 VERIFICATION OF ANN COMPUTATION

This chapter builds on chapter 3 that answers the first sub-question on the essential building blocks of Artificial Neural Network and what constraints these impose on a practical proofsystem and chapters 4 and 5 that answer the second sub-question on the essential building blocks of proof systems and what constraints their construction and expressiveness impose with respect to specific computations. Building on these findings this chapter maps these constraints of both constructs Artificial Neural Networks and proof systems together to ultimately answer the main research question by considering both the theoretical case, practical case and the actual practical case.

6.1 COMPUTATION REDUCTION

Both the inference and training algorithms can be reduced to a sequence of dot and Hadamard tensor products and the corresponding cost and activation functions. These computations can, both for fied as for floating point arithmetic, be implemented in an boolean circuit. This circuit takes the input, a label and weights and evaluates to the correctly updated weights or the correct prediction depending on the computation. This is inherently a circuit satisfiability problem and this problem can be reduced to a relation in the input language, specific to each of the selected proof systems, and as such this relation can be proven by each of the selected proof systems.
To ensure that the algorithms are executed on the appropriate data, the input to the model, the corresponding labels (in the case of training) but also the initial state of the model will be defined in the instance of the computational problem.

Randomness

Inherent to machine learning is a degree of randomness that shows up in the order of inputs, batch selection, drop out but in particular in the choice of initial weights. Similar to the free variables for the certificate or witness in NP problems proof systems we can use this freedom to define a variable input for the verification function. In the context of NP this additional input is generally unique and evaluation marks a correct or incorrect result. The discrepancy here is that; theoretically any random initialisation can be the basis for a correct computation and should therefore be accepted, but practically a prover is likely to choose an all zero initialisation which is, in the context of machine

learning, not in the interest of a verifier, the party outsourcing the training algorithm and relying on actual randomness.

A first and simple, naive, solution is to include the required randomness in the problem instance definition which effectively makes the computation deterministic. An alternative approach would be to do some sort of verification of randomness which ensures a close-to-uniform distribution and the appropriate complexity in a specific range. To the best of our knowledge there exist no such tests that run within polynomial time. Therefore we limit ourselves to the naive and in this particular case suitable solution.

A similar problem arises in the choice of the order of the inputs, or which inputs to batch together, as this order is generally random as well. Also in this case the naive solution will do the job in the absence of a better solution. As such, the best way to ensure proper randomness in an outsourced computation is for the verifier to generate it herself and make it part of the instance definition of the problem.

Input Selection

Ideally one would allow the prover to generate its own randomness and select the inputs randomly from a specified source. This means that in addition to a form of verification to guarantee randomness this would require a proof of inclusion, that specifies that this input or this batch of inputs is a (random) selection from the provided training set. Although proofs of inclusion exist, e. g. Merkle trees, we consider this type of proofs out of scope in this thesis as the nature of the outsourced computation dictates that the random selection is an essential part of proof. Specifically a proof of inclusion alone is not enough as it would still allow a prover to classify or train on one input of the provided data set. Therefore we again follow the naive approach and present the inputs to the inference and training algorithm, just like the randomness, in the problem instance definition. We consider both of the verification problems, randomness and indexed inclusion as open questions for future work.

6.1.1 *Computational Complexity*

All of the selected proof systems have sufficient expressiveness for both of the ANN computations as both of these are, if bound by a number of iterations, executable in polynomial time. The fact that both computations are in Palso means that the prover verification procedure is the execution of the computation itself. This in contrast to the polynomial evaluation of a problem in $NP$where the computa-

tion itself is different from the verification computation of the prover. More concretely solving the *NP*problem might require exponential complexity while the verification of the relation is by definition polynomial time bound. Effectively this defines a class of problems that are inherently efficient to verify. The computations studied in this thesis are, although in this class, not of this nature and do not separate a computation and verification algorithm. The computation itself is the one single approach to non-probabilistic verification of completeness and soundness.

The fact that both inference and training are polynomial procedures implies that the verification algorithm performed by the prover is of similar complexity compared to that of native execution. This means that succinctness and concrete verification efficiency are key to a solution for the case subject to this thesis. We require a potential solution to be either sub-linear or in the case of more significant computations, e. g. training, an order of magnitude smaller.

6.1.2 *System Computation Pairing*

At the end of chapter 3 a differentiation was made in the use cases of both the inference and training computation. Later, chapter 5 presents a selection of proof systems that all satisfy the zero-knowledgeând sub-linear verifier properties. The properties that are different for example inherent to the category of proof system makes a difference in the system to computation fit in terms of its use case.

In the case of inference, the computation consist of a large number of executions each with different input data but the exact same computation and therefore the same circuit. This means that any setup costs that can be shared over all the seperate executions reduces the relative cost per execution. This means that the creation of a CRS can be amortised. The LPCP class of problems, with the Groth16 and GM17 to be the most efficient protocols in its category, offers the best fit to this problem.

Training on the other hand is sequential and therefore represented by a circuit of significant depth and inherent complexity. But the repeated structure in the computation itself presents a natural fit to the execution trace and corresponding transition relation that defines the AIR that is taken as an input in STARKs. Although the system has considerable constant costs, long repeated computations enjoy good asymptotics that out perform the performance of other systems.

6.2 VERDICT

This leads us to the verification of this thesis by answering the mainr research question. We recall: Can Zero-Knowledge ARguments of Knowledge verify correct training of Artifical Neural Networks fulfilling the completeness, soundness, succinct and zero-knowledge properties?

6.2.1 *Theory*

Yes, theoretically both the training and inference computations of ANN fall within the class of problems P that, because $P \subseteq NP \subseteq NEXP$, can be proven and verified by the selection of proof systems in chapter 5. For example via a boolean circuits and a sufficiently large field it is possible to mimic all operations, including floating point arithmetic, exactly the same way a machine with bounded registers would physically. The important observation here is that both the prover and verifier run virtually, meaning that the prover cannot directly use the efficiency of its CPU nor GPU neither can the verifier.

These are significant constraints on the practicallity in the case of the verification of outsourced training of an ANN as this will impose an increase of computational cost of several orders of magnitude. Because of this gap between theory and practice it is only fair to refine and extend this answer and consider the practical case as well.

6.2.2 *Practice*

The difference between theory and practice is significant and it is clear that the computation in its exact native form will not allow any efficiency increase. As there are two key constructs here it is either one or both that has to relax to allow for an efficient match in constraints. True to the objective of this thesis identify the limiting constraints of the proof systems first and then consider how to alter the computation slightly to progress in a more practical implementation.

Because the inference and training computations are highly parallel and data intense in nature the underlying circuits will have many intermediate wire conditions that have to accounted for to generate the proof. This results in a large stream of data for all parallel computations, effectively rendering dedicated computation optimised hardware such as the GPU useless.
To mitigate this problem the prover will require more freedom, one way to do this is via am input problem in a different representation from a circuit. The frontend takes a high level representation of the computational problem and reduces this to a set satisfiable constraints

that do not require intermediate wire conditions. As this might not be possible an alternative approach is proof composition as was used in the work by Chabenne et al.[19] that used the SumCheck protocol to verify the matrix multiplication via subproofs and use a general proof system, Pinocchio, to verify the subproofs and activation functions. This approach has the added advantage that, for example when using Freivald's algorithm, the communication complexity and proof length of the subproofs are irrelevant as they are transferred virtually, "in the head", by the prover. Additionally this gives the freedom to the prover to use any MATMUL algorithm it pleases, e. g. naive, Strassen's or William's, on any hardware it pleases as only the relation between the input and output is verified with a probabilistic check.
An additional challenge presents itself in the nature of the finite fields at the core of proof system implementations. Because of the modulus in these fields the values do not have a relative relation, say a maximal or minimal value, one is greater than the other, a property that is essential to many of the delegatable computations among which machine learning and the training of ANNs. A solution for this is first of, choosing a large field. Second, a compiler that sets the value to the maximal accepted value when it should have done a modulus following algebra, this prevents the overflow. To actually use the order in the finite field is costly, but possible as well. Through the definition of a so called De-Militarized Zone (DMZ) and the apropriate arithmetisation it is possible to verify inequalities like $x < y$. This makes the go to efficient Rectified Linear Unit (ReLU) activation function a costly counterpart in the context of finite fields.

Aside from the proof system there are slight addaptations possible in the definition of the computation as well. To support floating point arithmetic we turn to multiple gates per value in boolean circuits that are represented by multiple arithmetic gates before the final arithmetisation. Changing the computation from floating points to fixed point arithmetic will prevent this blow up in the number of gates and corresponding constraints accordingly. Additionally the circuits of the activation functions are generally high degree approximations that result in complex constraints and again a blow up in the number of gates in the circuit. Similar to how activation functions are now choosen for efficient computation on floating point arithmetic and clean derivatives this line of work will have to optimise activation and cost functions to have low degree polynomial representations for both the function as it derivative. A good example of such functions are the x^2 and Mean Squared Error (MSE) that are both low degree polynomials with derivatives suitable for finite fields.

7 CONCLUSION

This thesis explores the real practicality of state of the art arguments systems in the application of Verifiable Computation, specifically the verification of correct outsourced training of an Artificial Neural Network. This thesis decomposes the two concepts at the core of this challenge to identify the elementary building blocks that make both constructs, proof systems and Artificial Neural Network. From this decomposition the constraints are derived that one might impose on the other. Constraints that are ultimately mapped and incorporated to one verdict on the feasibility problem at the core of this thesis.

Starting from an intuition on machine learning and in particular Neural Networks, various topologies and common model designs are identified. The mathematics that makes these models is then trimmed to the most generalised but still practical mathematical definition that, in turn, forms the basis from where the elementary computation is derived. Fully generalised both the inference and training algorithm boil down to a sequence of dot and Hadamard tensor products and element-wise evaluation of activation and cost functions on floating point arithmetic. Although the underlying optimisation problem is presumed to be NP-hardthe bound inference and training computations are both in P. Based on the understanding of the case subject to this thesis and the nature of both computations we distinguish between the use cases of both ANN algorithms.
First of outsourcing inference has most value in a parallel fashion, specifically in the case of a large number of different inputs on a pre-trained network. In contrast to training, that is an inherent sequential computation that relies on memory access to store intermediate values and typically processes a large number of inputs as well.

Continuing with an intuition on proving, specifically completeness, soundness, (zero)-knowledge and proof systems a new categorisation is proposed that uses the traits *interactivity* and *proof structure* together with *randomness* to differentiate between three categories of proof systems. Each of these, Interactive Proof, Probabilistically Checkable Proof, Interactive Oracle Proof comes with its own characteristics in terms of trusted setup, cryptographic primitives. prover and verifier complexity and thus efficiency. For each of the categories the underlying techniques fundamental to many of the practical implementations are discussed as the elementary building blocks.
Based on the literature a selection of implementations is discussed and

compared in more detail on the metrics and characteristics described in the seminal works. As succinctness and both prover and verifier efficiency are key to a practical implementation a pre-selection is made deciding on the properties zero-knowledge, succinctness, cryptographic assumptions and at least one per category and input type.

Finally the constraints imposed by the implementations are mapped on the constraints imposed by the inference and training computations. This led to a variety of challenges and observations, in particular: the input data and randomness of the computation will have to be predefined as part of the problem instance. In the ideal case the verifier would point at a a data set on a local or remote location and the prover would randomly select input (batches) from this set. Both the randomness and proofs of inclusion exist but the combination that ensures random selection of a data set efficiently is not described in literature. Both inference and training are computations in Pwhich implies that non-probabilistic verification is of the same complexity as re-executing the algorithm. This means that the proof system will need to be succinct and have a verifier complexity that is sub-linear to the computation.

This leads to two-fold answer of the main research question: Can Zero-Knowledge ARguments of Knowledge verify correct training of Artifical Neural Networks fulfilling the completeness, soundness, succinct and zero-knowledge properties. Yes, theoretically there exist argument systems that have the expressiveness to prove and verify both the inference and training computations, including the floating point arithmetic, fulfilling the completeness, soundness, succinctness and zero-knowledge properties. The negative counter-part of this answer is that although theoretically possible this remains far from practical in a real use case for two reasons. First of, the prover will need to perform the computation virtually keeping track of the intermediate states that will later represent the witness of the computation. Secondly computations on ANNs have become feasible because of a combination of innovations in the field that led to more efficient algorithms, more data and access to dedicated hardware e. g. GPUs. Because the intermediate wire conditions are required in the proof generation this will require a potential GPU to stream a log of all the intermediate results of all parallel computations undermining the use of this dedicated hardware.

7.1 FINAL RECOMMENDATIONS

Even though not practically feasible there are argument systems better fit the computations, subject to this thesis, better than others. In the case of the inference computation we talk about effectively the same

computation and therefore the same circuit for a large number of inputs. This reduces the relative cost of amortisation, the creation of a CRS. The LPCP class of problems, with the Groth16 protocol most efficient and GM17 to be second most efficient but more secure, offers the best fit to this problem.

On the contrary the training algorithm is sequential and therefore resulting in a circuit of significant depth and inherent complexity. But due to the repeated structure in the computation itself it presents a natural fit to the execution trace and corresponding transition relation that defines the Algebraic Intermediate Representation that STARKs take as an input. Although the system has high constants over long repeated computations this system has good asymptotics that out perform other systems. An important note is that the influence of this amount of input data maintains this efficiency or defeats it purpose.

7.2 FUTURE WORK

The field of proof and argument systems has predominantly been theoretical and has only seen practical implementations in the past five years. These practical implementations have led to a wider interest in the field increasing the pace novel constructions and implementations enter the playing field. This has resulted in the so called "ZK Zoo" with a wide range of terms, definitions, constructions and metrics, both qualitative and quantitative but no clear structure on how these relate.

With respect to the verification of outsourced training of an ANN by argument systems the primary problem is in the requirement to have access to all wire conditions on order to be able to create a valid proof. This is inherent to the language taken as an input by the backend that is, within the scope of this thesis, a representation derived from arithmetic circuits or an internal representation of circuits that enforces the wire conditions.

There exist two approaches to mitigate this challenge. The first is to use proof composition as was done in the work by Chabenne et al. [19]. This allows creates the freedom for the prover to choose its own method for a subcomputation, in this case matrix multiplication, for which a generic prover computes a composite proof consolidating all computation specific subproofs to one final proof. In their work they opt for the SumCheckprotocol optimised to reduce communication. The advantage of proof composition is that the subproofs are generated "in the head" which makes communication cost irrelevant. This means that in a similar implementation to the one just mentioned with a newer more efficient generic proof system and the well known

and communication expensive Freival algorithm might lead to more efficient verification of both the inference and perhaps the training computations.
A second solution to the wire conditions is the search for a new language (and compilers to that language) that translates problems to higher order constraints that do not require the intermediate values.

BIBLIOGRAPHY

[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight sublinear arguments without a trusted setup." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 2087–2104.

[2] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: A new characterization of NP." In: *Journal of the ACM (JACM)* 45.1 (1998), pp. 70–122.

[3] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof verification and the hardness of approximation problems." In: *Journal of the ACM (JACM)* 45.3 (1998), pp. 501–555.

[4] Emil Artin. *Geometric algebra*. Courier Dover Publications, 2016.

[5] László Babai. "Trading group theory for randomness." In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. ACM. 1985, pp. 421–429.

[6] László Babai, Lance Fortnow, Leonid Levin, and Mario Szegedy. "Checking computations in polylogarithmic time." In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. 1991, pp. 21–31.

[7] Boaz Barak and Oded Goldreich. "Universal arguments and their applications." In: *SIAM Journal on Computing* 38.5 (2008), pp. 1661–1694.

[8] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. "Multi-prover interactive proofs: How to remove intractability assumptions." In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, pp. 113–131.

[9] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive oracle proofs." In: *Theory of Cryptography Conference*. Springer. 2016, pp. 31–60.

[10] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, et al. "Computational integrity with a public random string from quasi-linear PCPs." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 551–579.

[11] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable zero knowledge via cycles of elliptic curves." In: *Algorithmica* 79.4 (2017), pp. 1102–1160.

[12] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. "Aurora: Transparent succinct arguments for R1CS." In: *IACR ePrint* 828 (2018).

[13] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed-Solomon interactive oracle proofs of proximity." In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[14] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable, transparent, and post-quantum secure computational integrity." In: (2018).

[15] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive composition and bootstrapping for SNARKs and proof-carrying data." In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 111–120.

[16] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. "Succinct non-interactive arguments via linear interactive proofs." In: *Theory of Cryptography Conference*. Springer. 2013, pp. 315–333.

[17] Avrim Blum and Ronald L Rivest. "Training a 3-node neural network is NP-complete." In: *Advances in neural information processing systems*. 1989, pp. 494–501.

[18] Gilles Brassard, David Chaum, and Claude Crépeau. "Minimum disclosure proofs of knowledge." In: *Journal of computer and system sciences* 37.2 (1988), pp. 156–189.

[19] Hervé Chabanne, Julien Keuffer, and Refik Molva. "Embedded Proofs for Verifiable Neural Networks." In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 1038.

[20] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. "Post-quantum zero-knowledge and signatures from symmetric-key primitives." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1825–1842.

[21] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical verified computation with streaming interactive proofs." In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM. 2012, pp. 90–112.

[22] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems." In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1986, pp. 186–194.

[23] Lance Fortnow, John Rompel, and Michael Sipser. "On the power of multi-prover interactive protocols." In: *Theoretical Computer Science* 134.2 (1994), pp. 545–557.

[24] Rūsiņš Freivalds. "Fast probabilistic algorithms." In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 1979, pp. 57–69.

[25] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[26] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic span programs and succinct NIZKs without PCPs." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 626–645.

[27] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud." In: *Advances in Neural Information Processing Systems*. 2017, pp. 4672–4681.

[28] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. "Zkboo: Faster zero-knowledge for boolean circuits." In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 1069–1083.

[29] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. "Delegating computation: interactive proofs for muggles." In: *Journal of the ACM (JACM)* 62.4 (2015), p. 27.

[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The knowledge complexity of interactive proof systems." In: *SIAM Journal on computing* 18.1 (1989), pp. 186–208.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[32] Jens Groth. "On the size of pairing-based non-interactive arguments." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 305–326.

[33] Jens Groth and Mary Maller. "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks." In: *Annual International Cryptology Conference*. Springer. 2017, pp. 581–612.

[34] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[35] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. "Efficient arguments without short PCPs." In: *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*. IEEE. 2007, pp. 278–291.

[36] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Cryptography with constant computational overhead." In: *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM. 2008, pp. 433–442.

[37] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." In: *arXiv preprint arXiv:1812.04948* (2018).

[38] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments." In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. ACM. 1992, pp. 723–732.

[39] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (2013).

[40] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic methods for interactive proof systems." In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. IEEE. 1990, pp. 2–10.

[41] Ralph C Merkle. "A certified digital signature." In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, pp. 218–238.

[42] Silvio Micali. "Computationally sound proofs." In: *SIAM Journal on Computing* 30.4 (2000), pp. 1253–1298.

[43] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: Nearly practical verifiable computation." In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 238–252.

[44] Omer Reingold, Guy N Rothblum, and Ron D Rothblum. "Constant-round interactive proofs for delegating computation." In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM. 2016, pp. 49–62.

[45] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors." In: *Cognitive modeling* 5.3 (), p. 1.

[46] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Idea Immanent in Nervous Activity." In: *Bulletin of Mathematical Biology* 5 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259.

[47] Mathew Salvaris, Danielle Dean, and Wee Hyong Tok. "Overview of Deep Learning." In: *Deep Learning with Azure*. Springer, 2018, pp. 27–51.

[48] Jürgen Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural networks* 61 (2015), pp. 85–117.

[49] Adi Shamir. "How to share a secret." In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[50] Adi Shamir. "IP= PSPACE (interactive proof= polynomial space)." In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. IEEE. 1990, pp. 11–15.

[51] Justin Thaler. "Time-optimal interactive proofs for circuit evaluation." In: *Annual Cryptology Conference*. Springer. 2013, pp. 71–89.

[52] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. "A hybrid architecture for interactive verifiable computation." In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 223–237.

[53] Riad S Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. "Verifiable asics." In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2016, pp. 759–778.

[54] Riad S Wahby, Ye Ji, Andrew J Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. "Full accounting for verifiable outsourcing." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 2071–2086.

[55] Riad S Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. "Doubly-efficient zkSNARKs without trusted setup." In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 926–943.

[56] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. "{DIZK}: A Distributed Zero Knowledge Proof System." In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 675–692.

[57] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation." In: ().