Feed Forward Neural Networks with Asymmetric Training

Archit Srivastava
University of Nebraska-Lincoln, asrivastava6@huskers.unl.edu

FEED FORWARD NEURAL NETWORKS WITH ASYMMETRIC TRAINING.

by

Archit Srivastava

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Vinod Variyam

Lincoln, Nebraska

August, 2022

FEED FORWARD NEURAL NETWORKS WITH ASYMMETRIC TRAINING.

Archit Srivastava, M.S.

University of Nebraska, 2022

Adviser: Vinod Variyam

Our work presents a new perspective on training feed-forward neural networks (FFNN). We introduce and formally define the notion of symmetry and asymmetry in the context of training of FFNN. We provide a mathematical definition to generalize the idea of sparsification and demonstrate how sparsification can induce asymmetric training in FFNN.

In FFNN, training consists of two phases, forward pass, and backward pass. We define symmetric training in FFNN as follows– If a neural network uses the same parameters for both forward pass and backward pass, then the training is said to be symmetric. The definition of asymmetric training in artificial neural networks follows naturally from the contrapositive of the definition of symmetric training. Training is asymmetric if the neural network uses different parameters for the forward and backward pass.

We conducted experiments to induce asymmetry during the training phase of the feed-forward neural network such that the network uses all the parameters during the forward pass, but only a subset of parameters are used in the backward pass to calculate the gradient of the loss function using sparsified backpropagation.

We explore three strategies to induce asymmetry in neural networks. The first method is somewhat analogous to drop-out because the sparsified backpropagation algorithm drops specific neurons along with associated parameters while calculating the gradient. The second method is excessive sparsification. It induces asymmetry by

dropping both neurons and connections, thus making the neural network behave as if it is partially connected while calculating the gradient in the backward pass. The third method is a refinement of the second method; it also induces asymmetry by dropping both neurons and connections while calculating the gradient in the backward pass.

In our experiments, the FFNN with asymmetric training reduced overfitting, had better accuracy, and reduced backpropagation time compared to the FFNN with symmetric training with drop-out.

DEDICATION

This thesis is dedicated to my family, maa, papa, my sister, mamu and the loving memory of my grandparents - mummy and nati.

ACKNOWLEDGMENTS

I was able to complete this thesis because of the tremendous support I received while working on it. First and foremost, I would like to thank my advisor, Dr. Vinod Variyam, for his time, patience, guidance, and all the constructive feedback that pushed me to refine my work. I also thank Dr. Stephen Scott and Dr. Ashok Samal for reviewing this thesis and for their valuable insights. I would also like to thank my parents for their unconditional support and my sister for always being there whenever I needed her. I want to acknowledge my friends Kshitij Srivastava and Shalini Rai, for often giving me good technical advice, Mrinal Rawool for giving me advice on LaTeX and how to be more productive while working on my thesis, and Arth Pundir for being my sympathetic ear. Last but not least, I would like to acknowledge Haruichi Furudate's and Yuki Hayashi's works which became a surprising source of inspiration and motivation during this arduous undertaking.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern artificial neural networks are significantly over-parameterized[8]; it is the reason they perform so well on large datasets. However, they are also prone to overfitting. Dropout[28] is a powerful technique to reduce overfitting. However, most popular implementations of dropout techniques in practice (in PyTorch and TensorFlow) do not reduce training time because dropout uses various sampling methods to drop the neuron's output after the computing the output.

Sparsified backpropagation[29] is a technique to compute the gradient of the loss function using a subset of parameters of the neural network. Since not all parameters are used in the computation of the gradient, the amount of computation reduces, which makes sparsified backpropagation faster. Additionally, experiments show that sparsified backpropagation also reduces overfitting.

We define a theoretical framework that generalizes the idea of sparsification and describes how sparsification techniques interact with the layers of the FFNN to construct the subset of parameters to be used for training. Using insights from our framework, we define the notion of symmetric and asymmetric training.

Definition 1 (Symmetric and Asymmetric training)**.** *Let* $\prod$ *be the set of all trainable* *parameters of a feed-forward neural network (FFNN). Let $\Theta \subseteq \prod$ be the subset of training parameters constructed by some sparsification method that is used in forward propagation and $\Phi \subseteq \prod$ be the subset of training parameters constructed by some sparsification method that is used in backpropagation. If $\Theta = \Phi$, then the training is said to be symmetric. If $\Theta \neq \Phi$, then the training is said to be asymmetric.*

In a FFNN without any sparsification method $\prod = \Theta = \Phi$. In this thesis, we explore asymmetry induced by sparsified backpropagation. [29].

1.1 Contributions

The main contributions of this thesis are as follows:

1. We develop a theoretical framework that generalizes the idea of sparsification and unifies dropout and sparsified backpropagation as members of the sparsification function class.

2. We develop rigorous notation to describe the interaction dynamics between sparsification function and neural network and how that leads to parameter subset construction when sparsification methods are used in FFNN.

3. Insights from our theoretical formulations offer a new perspective on the training dynamics of the neural network and define the notion of symmetric and asymmetric training.

4. We present a detailed symbolic description of the backpropagation algorithm for the FFNN.

5. From the first set of experiments, it seemed that asymmetric training using sparsifed backpropagation gives better accuracy than symmetric training. Further

experiments reveal that this is not the case. Out of the three asymmetric training methods we studied in this thesis, topk asymmetric training gave the best accuracy in 60% of the experiments. Partial connectivity asymmetry gave the best accuracy in 28% of the experiments. Models with symmetric training had the best accuracy in the remaining 12% of the experiments. The complete sparsified asymmetric training had the worse accuracy consistently in all the experiments.

Topk asymmetric training had up to 2% better accuracy than models with symmetric training. Partial connectivity asymmetric training had up to 5% better accuracy in an experiment than the models with symmetric training. When it came to reduction in backpropagation time,

1.2 Motivation and Challenges

The work in this thesis has both theoretical and practical motivations. On theoretical front, we invest time to come up with notation that rigorously defines sparsification and describes the interaction of sparsification functions with the layers of the neural network.

On applications front, sparsification has the potential to reduce training time, without adversely affecting the performance of the model on unseen data. In out experiments topk asymmetry and partial connectivity asymmetric reduces the back-propagation time by upto 50% without any reduction in accuracy. This makes sparsification a viable candidate for smaller devices with model training capabilities such smart sensors, raspberry pi, wearable and iot devices.

1.3 Outline

The remainder of the thesis is organized as follows. Chapter 2 discusses related works. Chapter 3 presents a brief background on machine learning. Chapter 4 establishes the notation of FFNN. Chapter 5 presents the backpropagation algorithm in detail. Chapter 6 introduces the definition of the sparsification functions and describes how sparsification functions interact with the layer of a FFNN to construct the subset of parameters to be used in training. Chapter 7 combines ideas presented in chapters 5 and 6 to describe sparsified backpropagation and its variants. Chapter 8 describes how different variations of sparsified backpropagation induce different types of asymmetric training. Chapter 9 provides information about the experiment-setup and results. Finally, Chapter 10 is the conclusion and discussion of future works.

Chapter 2

Related Works

In this section, we present related works that are relevant to this thesis.

Acceleration in backpropagation can be achieved without sparsification. As demonstrated by authors in their work, using adaptive methods accelerates learning by localized adaptations of the parameters based on the behavior of the error functions [26]. However, sparsified backpropagation has become very popular in the last few years as considerable progress has been made to develop sparsified backpropagation in different directions. Theoretical works like the convergence of sparsified stochastic gradient descent [2] explore convergence properties of using sparsified gradient in stochastic gradient descent. Meanwhile, works like SGD with memory, extents sparsified backpropagation to distributed systems, while also working to reduce communication time [1]. Sparsified backpropagation has also been extended to CNNs as well [16].

While working on the generalizing sparsification, we investigated various dropout techniques as well, such as guided dropout sampling [20] that defines a strength attribute for perceptrons using uniform sampling. Gaussian dropout [33] uses Gaussian sampling as an attribute for dropping neurons. There have been efforts to use Bayesian deep learning [14] with dropout. In annealed dropout [25] authors change the dropout rate during training and make it dynamic. Drop connect [32] is a modification to the standard dropout that drops connections in a neuron instead of neurons.

While looking for asymmetric training dynamics and partial connected neural networks, we found that genetic algorithms [11] are used to induce partial connectivity instead of sparsification methods.

A lot of the theory developed for describing properties of sparsification functions and how it interacts with FFNN uses multisets [10] extensively as they allow membership for multiple elements and have set operations and functions defined for them. Multisets have been used to represent input features to make neural network permutation invariant [35], Multisets have also been used in hypergraph neural networks [13].

The work in this thesis is different, as we use set theoretic to formalize the internal dynamics of sparsification functions which yields novel insights.

Chapter 3

Background

The Statistical learning framework [6] generalizes the structure and dynamics of multi-class classifiers such as, but not limited to, FFNN.

3.1 Statistical learning framework

We briefly go over the components of the statistical learning framework -

1. The learner, is usually a parameterized function approximator that learns the mapping between the inputs and output class.
2. The input set X, represented by a vector of features and is sampled from an arbitrary, unknown probability distribution
3. The output set of labels Y containing the correct label for each instance in the input set X.
4. Training dataset $T_S \subset X \times Y$ is of the form (x_i, y_i)
5. There exists an unknown function $f \colon X \to Y$ that maps each instance x_i in training set S to its correct label y_i.

6. The output of the learner is called the hypothesis and is a function $H\colon X \to Y$, that assigns labels to each instance x_i.

7. Measure of success provides a quantitative way to evaluate the output of a learner. It is usually the an error or loss function. For example, cross-entropy loss.

8. Empirical risk minimization (ERM) is the learning paradigm for a learner that searches for a hypothesis that minimizes the error or the loss function.

9. Learning rule is the algorithms that dictates how the parameters of the learner change during empirical risk minimization while training. In case of FFNN, the most commonly used learning rule is the stochastic gradient descent.

10. Hypothesis class is the set of all possible hypotheses for a learner. Training is searching for a hypothesis in the hypothesis class that minimizes the error or the loss function..

3.2 Overfitting and inductive bias

Overfitting is the condition when the learner ends up memorizing the training set instead of learning the underlying patterns, in order to prevent overfitting the hypothesis class is restricted during training the learner. This restriction is called inductive bias. Asymmetric training imposes a restriction that the set of parameters used during forward pass and backward pass are different, thus acting as an inductive bias.

Chapter 4

Feed forward neural networks

For this thesis, the learner is the feed-forward neural network (FFNN). This chapter establishes the notation to describe the FFNN. *Relu* [7] activation function is used throughout for the non-linear transformation.

4.1 Element sets

Definition 2. *Let $\boldsymbol{V}$ be a non-scalar entity; it can either be a vector or a matrix. Then the element set $\{\boldsymbol{V}\}$ of a vector or matrix $\boldsymbol{V}$ is a set defined as follows,*

$$\{\boldsymbol{V}\} = \{(i, v_i) | v_i \textit{ is an element of } \boldsymbol{V} \textit{ and } i \textit{ is the index of } v_i\} \tag{4.1}$$

Where (i, v_i) is an ordered pair.

4.2 Perceptron

[1] Perceptron [27] is the building block of the feed-forward neural network. We denote a perceptron by ρ.

[1]The term perceptron and neuron are used interchangeably in this document.

4.2.1 Structure of a perceptron

The parameters of the perceptron ρ consists of a weight vector $\overrightarrow{\mathbf{W}}$ and a bias scalar b, as displayed in Figure 4.1. The weight vector $\overrightarrow{\mathbf{W}}$ characterizes a hyperplane [4] that resides in the feature space of its input. Thus, for an n-dimensional input, $\vec{\mathbf{X}} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_m \end{bmatrix}$ the weight vector $\overrightarrow{\mathbf{W}}$ is represented by, $\overrightarrow{\mathbf{W}} = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_m \end{bmatrix}$.

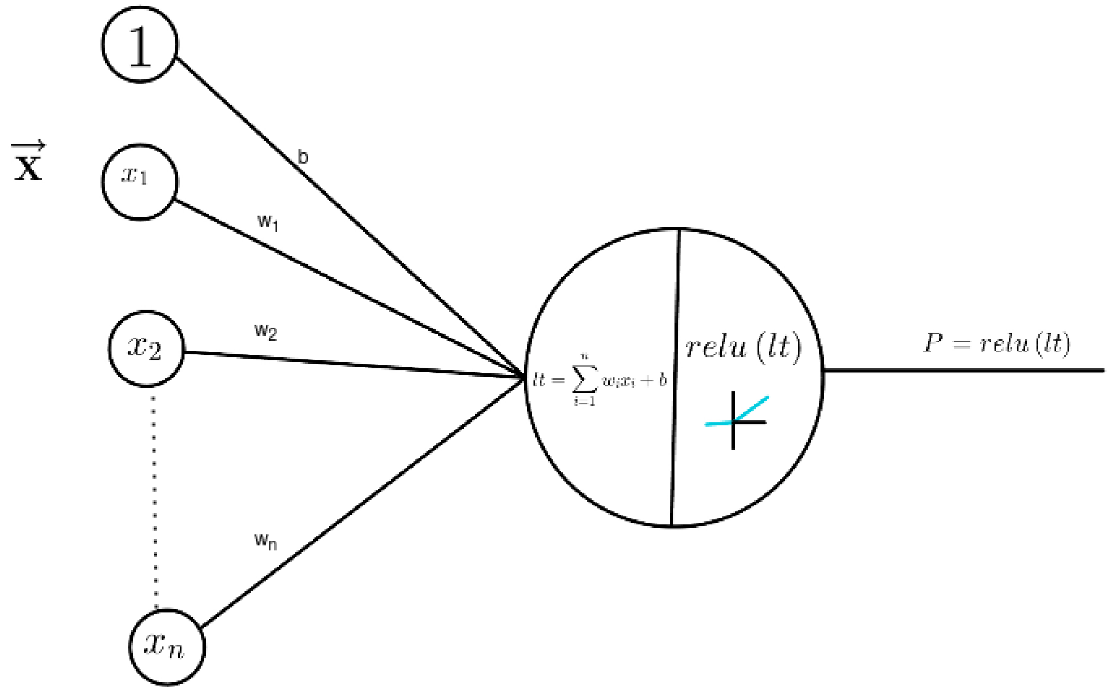
$\overrightarrow{\mathbf{X}}$
1
x_1
x_2
x_n
b
w_1
w_2
w_n
$lt = \sum_{i=1}^{n} w_i x_i + b$
$relu\,(lt)$
$P = relu\,(lt)$

Figure 4.1: Perceptron

4.2.2 Operation at a perceptron

A perceptron performs functional composition of linear transformation followed by non-linear transformation as displayed in Figure 4.2.

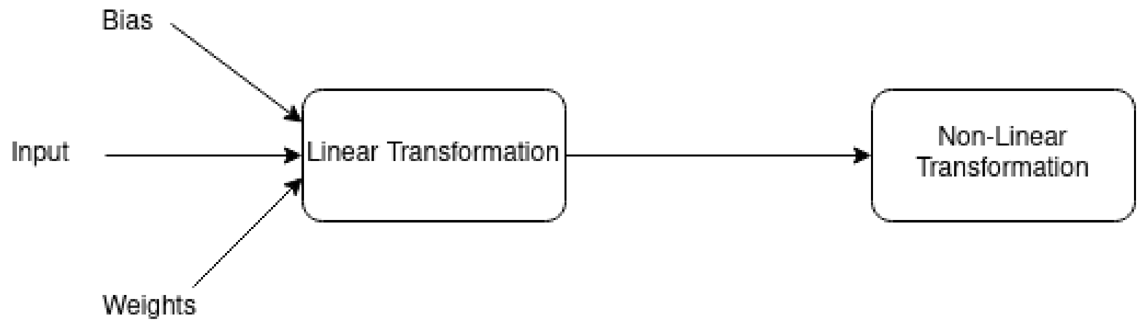

Figure 4.2: Operation at a perceptron

The linear transformation includes the inner product of the input vector $\vec{\mathbf{X}}$ with the weight vector $\vec{\mathbf{w}}$ followed by the addition of scalar b as shown by equation 4.2.

$$lt = \overrightarrow{\mathbf{W}}^{T}.\overrightarrow{\mathbf{X}} + b = \sum_{k=1}^{n} w_k x_k + b \tag{4.2}$$

The output of the linear transformation is the input for the non-linear transformation that applies the activation function to the linear transformation. The output of the non-linear transformation is considered the output of the perceptron and is denoted by P as shown in equation 4.3.

$$P = relu(lt) = \begin{cases} 0, & \text{if } lt \leq 0, \\ lt, & \text{if } lt > 0. \end{cases} \tag{4.3}$$

Where $lt \in \mathbb{R}$ and is the output of the linear transformation.

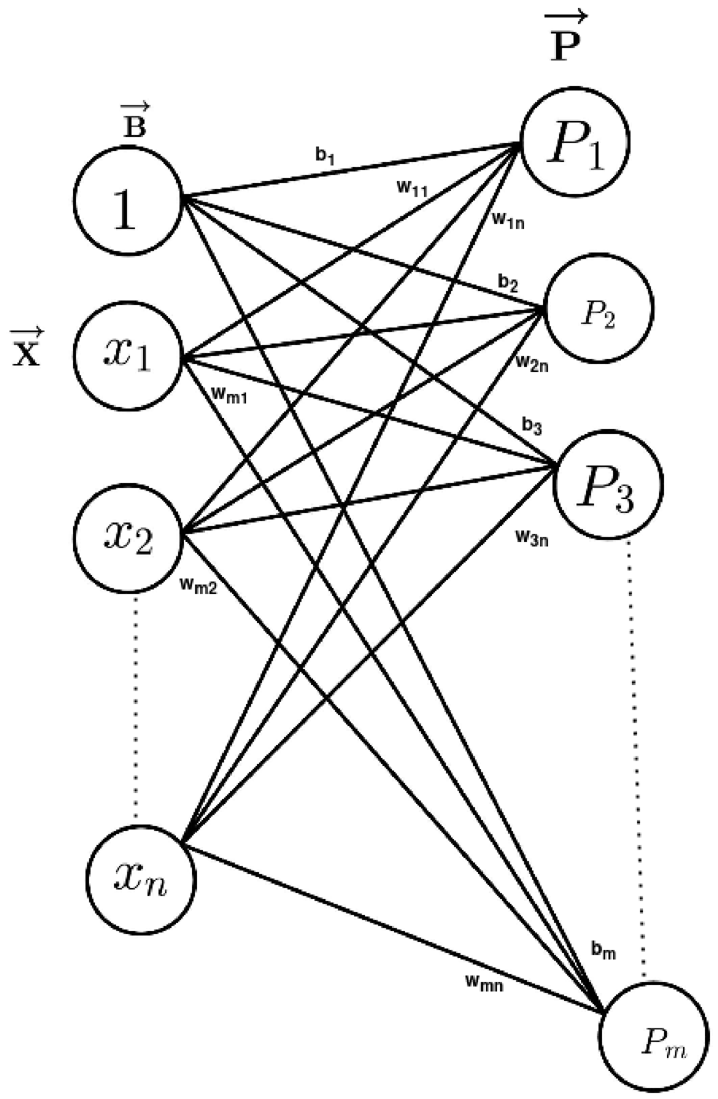
$\overrightarrow{\mathbf{P}}$
$\overrightarrow{\mathbf{B}}$
$\overrightarrow{\mathbf{X}}$
1
x_1
x_2
x_n
P_1
P_2
P_3
P_m
b1
w11
w1n
b2
w2n
b3
w3n
wm1
wm2
bm
wmn

Figure 4.3: layer

4.2.3 Parameters of a perceptron

We use set-theoretic notation to describe the parameters of a perceptron. That will help us understand how the sparsification function constructs a subset of parameters in Chapter 4 Let π be the set of all parameters of the perceptron. Then

$$\pi = \{\overrightarrow{\mathbf{W}}^T\} \bigcup \{b\} \bigcup \{\emptyset\} \tag{4.4}$$

where $\{\overrightarrow{\mathbf{W}}^T\}$ and $\{b\}$ are element sets of $\overrightarrow{\mathbf{W}}^T$ and b.

4.3 Layer

For a FFNN with n layers,(Figure 4.3) the set of perceptrons is denoted by $\mathbb{P}_i$, where

$$\mathbb{P}_i = \{\rho_{i,j} | \rho_{i,j} \text{ is the } j^{th} \text{ neuron in the } i^{th} \text{ layer and } 1 \leq j \leq |i|\} \bigcup \{\emptyset\} \tag{4.5}$$

The number of of perceptrons in the i^{th} layer is the cardinality of the set $\mathbb{P}_i$, denoted by $|\ \mathbb{P}_i|$. The input, weights, biases, output of linear transformation, and final output of i^{th} layer are denoted by $\overrightarrow{\mathbf{P}_{i-1}}, \mathbf{W}_i, \overrightarrow{\mathbf{B}}_i, \overrightarrow{\mathbf{Lt}}_i, \overrightarrow{\mathbf{P}}_i$ respectively,where $1 \leq i \leq n$, input layer is $\overrightarrow{\mathbf{X}} = \overrightarrow{\mathbf{P}}_0$, and the output of the last layer is $\overrightarrow{\mathbf{P}}_n$.

1. The weights of the i^{th} layer is represented by the weight matrix $\mathbf{W}_i$, where each row of the weight matrix is the transpose of the weight vector of the corresponding perceptron in the layer. $\mathbf{W}_i = \begin{bmatrix} \overrightarrow{\mathbf{W}}^T_{i,1} \\ \overrightarrow{\mathbf{W}}^T_{i,2} \\ . \\ . \\ \overrightarrow{\mathbf{W}}^T_{i,|i|} \end{bmatrix}$

2. The biases of the i^{th} layer are represented by an $|i|$ dimensional bias vector $\overrightarrow{\mathbf{B}}_i$. Where each element b_{ij} in $\overrightarrow{\mathbf{B}}_i$ represents the bias of the j^{th} perceptron in the i^{th} layer. $\overrightarrow{\mathbf{B}} = \begin{bmatrix} b_{i,1} \\ b_{i,2} \\ . \\ . \\ b_{i,|i|} \end{bmatrix}$

3. The output of the linear transformation at each neuron of the ith layer is represented by an $|i|$ dimensional vector $\overrightarrow{\mathbf{Lt}_i}$, where each element lt_{ij} in $\overrightarrow{\mathbf{Lt}_i}$ represents the output of the linear transformation of the jth neuron in the ith layer. Having a separate representation of

$$\overrightarrow{\mathbf{Lt}_i} = \mathbf{W}_i.\overrightarrow{\mathbf{P}_{i-1}} + \vec{\mathbf{B}}_i = \begin{bmatrix} lt_{i,1} \\ lt_{i,2} \\ . \\ . \\ lt_{i,m} \end{bmatrix} \text{ where } lt_{i,j} = \overrightarrow{\mathbf{W}}^T_{i,j}.\overrightarrow{\mathbf{P}}_{i-1} + b_{i,j}$$

$= \sum_{k=1}^{n} w_{i,j,k} P_{i-1,k} + b_{i,j}$

4. The output of the the ith layer are represented by an $|i|$ dimensional vector $\overrightarrow{\mathbf{P}_i}$. Where each element $P_{i,j}$ in $\overrightarrow{\mathbf{P}_i}$ represents the output of the jth neuron in the ith layer.

$$\overrightarrow{\mathbf{P}_i} = \begin{bmatrix} P_{i,1} \\ P_{i,2} \\ . \\ . \\ P_{i,|i|} \end{bmatrix}, \text{ where } P_{i,j} = relu(lt_{i,j}) = \begin{cases} 0, & \text{if } lt_{i,j} \leq 0, \\ lt_{i,j}, & \text{if } lt_{i,j} > 0. \end{cases}$$

4.3.1 Parameters of a layer

Let $\pi_{i,j}$ be the set of all parameters of the jth neuron in the ith layer. Then

$$\pi_{i,j} = \{\overrightarrow{\mathbf{W}}^T_{i,j}\} \bigcup \{b_{i,j}\} \bigcup \{\emptyset\} \tag{4.6}$$

where $\{\overrightarrow{\mathbf{W}}^T_{i,j}\}$ and $\{b_{i,j}\}$ are element sets of $\overrightarrow{\mathbf{W}}^T_{i,j}$ and $b_{i,j}$ respectively.

We define Π_i to be the set of all the parameters in the ith layer of the FFNN and the null element.

$$\Pi_i = \bigcup_{j=1}^{|\mathbb{P}_i|} \{\pi_{i,j}\} \bigcup \{\emptyset\} \tag{4.7}$$

4.3.2 Parameter mapping function

The set of neurons $\mathbb{P}_i$ is an abstract construct that facilitates the development of theoretical ideas. In this section, we associate each abstract representation of a neuron with its parameters. It is done by mapping a perceptron $\rho_{i,j} \in \mathbb{P}_i$ in layer i of the FFNN to its parameters $\pi_{i,j}$.

Definition 3 (Parameter mapping function)**.** *The Parameter mapping function $\mathscr{P}$ for layer i in the FFNN is a bijection defined as,*

$$\mathscr{P} : \mathbb{P}_i \to \Pi_i \textit{ such that } \mathscr{P}(\rho_{i,j}) = \pi_{i,j} = \{\vec{\boldsymbol{w}}_{i,j}^T\} \bigcup \{b_{i,j}\} \bigcup \{\emptyset\} \tag{4.8}$$

where $\pi_{i,j} \in \Pi_i$ and $\mathscr{P}(\{\emptyset\}) = \{\emptyset\}$.

Chapter 5

Backpropagation algorithm.

In the case of FFNN, gradient descent is the most commonly used learning rule.

5.1 Gradient Descent

Gradient descent is a finite sum optimization problem[3] of the form

$$min_x \frac{1}{n} \sum_{i=1}^{i=n} f_i(x) \tag{5.1}$$

where $f_i(x)$ is the metric that measures the performance of the output of the FFNN with the actual labeling function. However, stochastic gradient descent [12] constructs a stochastic approximation [5] of the gradient is used in practice.

The gradient descent algorithm finds the gradient of the loss function in the direction of the steepest descent. The parameters are updated according to the equation 5.2.

$$w_{i,j,k} := w_{i,j,k} - \eta \frac{\partial L}{\partial w_{i,j,k}} \tag{5.2}$$

5.2 Backpropagation

Backpropagation is a special case of reverse mode automatic differentiation [9] that finds the gradient of the loss function with respect to the network parameters.

5.2.1 Reverse mode Automatic differentiation

Automatic differentiation is a technique for computing the gradient of a function at a given point. Let $f\colon \mathbb{R}^n \to \mathbb{R}^m$, such that $y = f(x_1, .., x_n)$, where $x_1, .., x_n$ are the independent variables,whose derivative needs to be computed with respect to its input variables.

In automatic differentiation, the function to be differentiated-f is expressed as a recursive functional composition of atomic operations (or functions) whose derivatives are known beforehand; usually, these atomic operations include arithmetic operations, trigonometric, exponential, logarithmic functions.

Depending on the structure of f, the atomic functions are applied to the input variables, and their outputs are denoted by intermediate variables. The number of intermediate variables depends on the number of operations in f. If the function has o operations, then number of intermediate variables would also be o and they are denoted by u_i where $1 \leq i \leq o$. Thus,

$$y = u_i(u_j) \text{ where } u_j = \begin{cases} u_j(u_k) \\ \text{or} \\ u_j(x_i) \end{cases} \tag{5.3}$$

Using the chain rule from calculus,

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial u_i}\frac{\partial u_i}{\partial x_i} \tag{5.4}$$

Definition 4 (Sensitivity)**.** *The term $\frac{\partial y}{\partial u_i}$ is called sensitivity. It is the derivative of the output y with respect to any intermediate variable.*

Definition 5 (Adjoint)**.** *The term $\frac{\partial u_i}{\partial x_i}$ is referred to as adjoint. It is the derivative of the intermediate variable u_i with respect to another intermediate variable u_j or an independent variable x_i.*

Equation 5.4 establishes the derivative of the function with respect to an arbitrary input as the product of sensitivity and adjoint. Since f is a multidimensional vector function, each term in equation 5.4 is a non scalar entity (either a vector or a matrix). Usually, sensitivity is a matrix called the Jacobian matrix, and the adjoint is a vector. Their product is called the Jacobian vector product or JVP.

5.2.2 Backpropagation as Reverse Automatic differentiation

In this section, we express the FFNN loss function in the form mentioned in section 5.2.1 and see how the backpropagation algorithm is a special case of the reverse mode automatic differentiation.

5.2.2.1 Loss function of FFNN in recursive functional composition form.

Equation 5.5 expresses loss L as a function of vectors $\overrightarrow{\mathbf{P}_n}$ and $\overrightarrow{t}$.

$$L = L(\overrightarrow{\mathbf{P}_n}, \overrightarrow{t}) \tag{5.5}$$

Where $\overrightarrow{\mathbf{P}_n}$ is the output of the last layer as specified in section 4.3 and $\overrightarrow{t}$ is the true output in one-hot encoding. Similarly, section 4.3 also establishes that

$$\overrightarrow{\mathbf{P}_n} = P_n(\overrightarrow{\mathbf{Lt}_n}) \tag{5.6}$$

and

$$\overrightarrow{\mathbf{Lt}_n} = Lt_n(\overrightarrow{\mathbf{P}_{n-1}}, \mathbf{W}_n, \overrightarrow{\mathbf{B}_n}) \tag{5.7}$$

5.2.3 Gradient of the last layer

The backpropagation algorithm starts by calculating the gradient of the loss function with respect to the parameters of the last layer, and it cascades from the last layer to the preceding layers, all the way to the first layer. From the application of chain-rule to equations 5.5, 5.6,5.7 we get,

$$\frac{\partial L}{\partial \boldsymbol{W_n}} = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} \frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \boldsymbol{W_n}} \tag{5.8}$$

- By definition 4 $\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}}$ is the sensitivity of layer n denoted by δ_n and
- $\frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \boldsymbol{W_n}}$ is the adjoint. For clarity, this term is called the weight adjoint.

5.2.3.1 Sensitivity of a layer

Application of the chain rule to the sensitivity of $n^{th} layer$ $\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}}$ gives us equation 5.9

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} = \frac{\partial \overrightarrow{\boldsymbol{P_n}}}{\partial \overrightarrow{\boldsymbol{Lt_n}}} \frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_n}}} \tag{5.9}$$

Since $\overrightarrow{\mathbf{P}_{n-1}}, \mathbf{W}_n, \overrightarrow{\mathbf{B}_n}$ are non scalar entities (vectors or matrices) the product of the sensitivity and weight adjoint will be the Jacobian-Vector product (JVP) as mentioned in 5.2.1.

Equations 5.10, 5.11,5.12 and 5.13 were worked out by following the ideas on matrix calculus for deep learning outlined in [23]

$$\frac{\partial \overrightarrow{\boldsymbol{P_n}}}{\partial \overrightarrow{\boldsymbol{Lt_n}}} = \begin{bmatrix} \frac{\partial P_{n1}}{\partial Lt_{n1}} & 0 & \cdots & 0 \\ 0 & \frac{\partial P_{n2}}{\partial Lt_{n2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \cdots & \frac{\partial P_{n1}}{\partial Lt_{n|n|}} \end{bmatrix} \tag{5.10}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_n}}} = \begin{bmatrix} \frac{\partial L}{\partial P_{n,1}} \\ \frac{\partial L}{\partial P_{n2,}} \\ . \\ . \\ \frac{\partial L}{\partial P_{n,|n|}} \end{bmatrix} \tag{5.11}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} = \begin{bmatrix} \frac{\partial P_{n1}}{\partial Lt_{n1}} & 0 & \cdots & 0 \\ 0 & \frac{\partial P_{n2}}{\partial Lt_{n2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \cdots & \frac{\partial P_{n1}}{\partial Lt_{n|n|}} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial P_{n,1}} \\ \frac{\partial L}{\partial P_{n2,}} \\ . \\ . \\ \frac{\partial L}{\partial P_{n,|n|}} \end{bmatrix} \tag{5.12}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} = \begin{bmatrix} \frac{\partial P_{n1}}{\partial Lt_{n1}} \frac{\partial L}{\partial P_{n1}} \\ \frac{\partial P_{n2}}{\partial Lt_{n2}} \frac{\partial L}{\partial P_{n2}} \\ . \\ . \\ \frac{\partial P_{n|n|}}{\partial Lt_{n|n|}} \frac{\partial L}{\partial P_{n|n|}} \end{bmatrix} \tag{5.13}$$

For this thesis, the *relu* non-linear activation function has been used throughout for non-linear transformation.

$$\overrightarrow{\mathbf{P}_n} = relu(\overrightarrow{\mathbf{L}t_n}) \tag{5.14}$$

where $\overrightarrow{\mathbf{L}t_n}$ has been defined in 2 Thus,

$$\frac{\partial P_{ni}}{\partial Lt_{ni}} = \begin{cases} 0, & \text{if } P_{ni} \leq 0, \\ 1, & \text{if } P_{ni} > 0. \end{cases} \tag{5.15}$$

5.2.3.2 Weight adjoint

The weight adjoint in equation 5.8. Equation 5.16 also follows from [23]

$$\frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \boldsymbol{W_n}} = \overrightarrow{\boldsymbol{P}}^{\boldsymbol{T}}_{\boldsymbol{n-1}} = \begin{bmatrix} P_{n-1,1} & P_{n-1,2} & .. & P_{n-1,|n|} \end{bmatrix} \tag{5.16}$$

5.2.3.3 Gradient with respect to the bias

Equation 5.17 is the gradient of the loss function with respect to the bias vector of the last layer using the chain rule.

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{B_n}}} = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} . \frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \overrightarrow{\boldsymbol{B_n}}} \tag{5.17}$$

We have equation 5.18 from chapter 2.

$$\overrightarrow{\mathbf{Lt}_n} = \mathbf{W}_n \overrightarrow{\mathbf{P}_{n-1}} + \overrightarrow{\mathbf{B}_n} \tag{5.18}$$

Equation 5.19 follows from [23]

$$\frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \overrightarrow{\boldsymbol{B_n}}} = I_n \tag{5.19}$$

Thus,

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{B_n}}} = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} I_n = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} \tag{5.20}$$

5.2.3.4 Sensitivity of preceding layer

From equations 5.9 and 5.17, we see that in order to calculate the gradient of the loss functions with respect to the parameters of the preceding layer, we require the sensitivity of the layer and the weight adjoint of the layer.

We get the weight adjoint for the layer from equation 5.16. For sensitivity, we require $\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_{n-1}}}}$, the derivative of the loss function with respect to the output of each perceptron in the preceding layer (that is also the input for the current layer) that is given by the equation 5.21.

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_{n-1}}}} = \frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \overrightarrow{\boldsymbol{P_{n-1}}}} \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_n}}} \tag{5.21}$$

5.2.3.5 Perceptron (or neuron) adjoint

We refer to the term $\frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \overrightarrow{\boldsymbol{P}}_{\boldsymbol{n-1}}}$ as perceptron (or neuron) adjoint. It is given by equation 5.22. The proof could be found in [23].

$$\frac{\partial \overrightarrow{\boldsymbol{Lt_n}}}{\partial \overrightarrow{\boldsymbol{P}}_{\boldsymbol{n-1}}} = \boldsymbol{W_n^T} = \begin{bmatrix} \vec{w_{n1}} & \vec{w_{n2}} & .. & \vec{w_{n|n|}} \end{bmatrix} \tag{5.22}$$

5.2.3.6 Gradient of an arbitrary layer i

For layer i, the backpropagation algorithm calculates the following for the JVP.

- the sensitivity of the layer,
- the weight adjoint,
- the perceptron adjoint and
- the derivative of the loss, with respect to the input of the layer to get the sensitivity of the preceding layer.

And the gradient is given by,

$$\frac{\partial L}{\partial \boldsymbol{W_i}} = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}} \frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}} \tag{5.23}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}} = \frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}} \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}} \tag{5.24}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{B_i}}} = \frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}} . \frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \overrightarrow{\boldsymbol{B_i}}} \tag{5.25}$$

Chapter 6

Sparsification

Definition 6 (Sparsification functions)**.** *We define a class of functions $\mathbb{S}$, as sparsification functions if for any arbitrary function S in $\mathbb{S}$, the following properties hold true.*

1. *$S\colon \mathbb{R}^m \to \mathbb{R}^m$, is an element-wise function, let $\vec{x}, \vec{y} \in \mathbb{R}^m$ such that $\vec{y} = S(\vec{x})$, then $y_i = S(x_i)$.*
2. *For every element of x_i of vector $\vec{x}$, $S(x_i)$ may return either 0 or x_i.*

$$y_i = S(x_i) = \begin{cases} 0 & \textit{selection criterion: false} \\ x_i & \textit{selection criterion: true} \end{cases} \tag{6.1}$$

3. *Where selection criteria is an intrinsic property of each member function S in $\mathbb{S}$ that dictates how sparsification is carried out.*

Dropout [28] and the *topk* [29] sparsification function used in backpropagation are member functions of the sparsification function class $\mathbb{S}$.

6.1 Dropout and topk as sparsification functions

Let $\vec{x}$ be a vector in $\mathbb{R}^m$.

6.1.1 Dropout

Let dropout rate $= d$. Then,

$$dropout(x_i) = \begin{cases} 0, & \text{with probability } d \\ x_i, & \text{with probability } (1-d). \end{cases} \tag{6.2}$$

Example, let $\overrightarrow{x} = \begin{bmatrix} 10 \\ -20 \\ 30 \\ -40 \end{bmatrix}$, and dropout rate$= 0.5$, then, $dropout(\overrightarrow{x}) = \begin{bmatrix} 0 \\ -20 \\ 30 \\ 0 \end{bmatrix}$

6.1.2 Topk

$$topk(x_i) = \begin{cases} 0, & \text{when } x_i \text{ is not the k largest element of } \vec{x} \text{ irrespective of the sign} \\ x_i, & \text{when } x_i \text{ is k largest element of } \vec{x} \text{ irrespective of the sign.} \end{cases} \tag{6.3}$$

Example, let $\overrightarrow{x} = \begin{bmatrix} 10 \\ -20 \\ 30 \\ -40 \end{bmatrix}$, and k$= 2$, then, $topk(\overrightarrow{x}) = \begin{bmatrix} 0 \\ 0 \\ 30 \\ -40 \end{bmatrix}$

6.2 Sparsification in FFNN

Most variations of dropout and the topk sparsification function in sparsified back-propagation select a subset of neurons in layer i. However, sparsification functions topk and dropout can construct the subset of parameters directly, without constructing the subset of neurons.

For a FFNN with n layers, the set of parameters in the i^{th} layer is Π_i and the set of neurons in the i^{th} layer is $\mathbb{P}_i$. We can apply the sparsification function on either Π_i or $\mathbb{P}_i$ to construct the subsets $\Pi spars_i$ or $\mathbb{P}spars_i$ respectively.

The sparsification function does not directly interact with any neuron or parameter. To explicitly describe the interaction of the sparsification function with a neuron or a parameter, we define a numerical metric for each neuron or parameter called the sparsification attribute. The sparsification function acts on the sparsification attribute of a neuron or a parameter to decide whether that neuron or parameter would participate in training or not.

Definition 7 (Sparsification attribute)**.** *The sparsification attribute is an m dimension vector $\overrightarrow{\boldsymbol{A}}$. where $\overrightarrow{\boldsymbol{A}} \in \mathbb{R}^m$., Where m can either be the number of neurons in the layer when the sparsification function is applied to $\mathbb{P}_i$ or the number of connections in a neuron when the sparsification function is applied to Π_i.*

1. *If sparsification function S is used to construct the subset of neurons , $\mathbb{P}spars_i$ in layer i , then $m = |\mathbb{P}_i| =$ number of neurons in layer i. The **sparsification attribute for neuron** selection is represented as $\overrightarrow{\boldsymbol{An}_i}$. The element $an_{i,j}$ in $\overrightarrow{\boldsymbol{An}_i}$ is a metric for the neuron $\rho_{i,j}$, and $S(a_{i,j})$ dictates whether neuron $\rho_{i,j}$ should be kept or dropped.*

2. *If sparsification function is used to construct the subset of parameters $\pi spars_{i,j}$ in the j^{th} neuron in layer i , then $m = |\mathbb{P}_{i-1}| =$ number of connections in neuron $\rho_{i,j}$. The **sparsification attribute for parameter** selection is represented as $\overrightarrow{\boldsymbol{Ap}_{ij}}$ The element $ap_{i,j,k}$ in $\overrightarrow{\boldsymbol{Ap}_{ij}}$ is a metric for the parameters $\pi_{i,j}$, and $S(ap_{i,j,k})$ specifies which parameters in $\pi_{i,j}$ should be kept.*

Definition 8 (Sparsified attribute)**.** *Sparsified attribute $\overrightarrow{\boldsymbol{A}spars}$ is defined as, $\overrightarrow{\boldsymbol{A}spars} = S(\overrightarrow{\boldsymbol{A}})$, where $\overrightarrow{\boldsymbol{A}}$ is the sparsification attribute and S is the sparsification* *function.*

6.2.1 Neuron selection using sparsification function

The set of neurons in layer i is denoted by $\mathbb{P}_i$, from equation 4.5. Let $\mathbb{P}spars_i$ be the set of neurons selected by the dropout or topk sparsification function. Then $\mathbb{P}spars_i \subset \mathbb{P}_i$.

Only the parameters of the neurons in $\mathbb{P}spars_i$ are used in training. This set of parameters is

$$\Pi spars_i = \mathscr{P}(\mathbb{P}spars_i) \tag{6.4}$$

Where $\mathscr{P}$ is the parameter mapping function defined in 3

6.3 Subset construction using sparsification function

6.3.1 Attribute-neuron mapping function and construction of the neuron subset

Here we introduce the notation that associates a neuron's sparsification attribute $(j, a_{i,j}) \in \{\overrightarrow{\mathbf{An}_i}\}$ to the neuron $\rho_{i,j}$ in $\mathbb{P}_i$.

Definition 9 (Attribute-neuron mapping function)**.** *The attribute-neuron mapping function $\mathscr{A}_n$ for layer i in the FFNN is a bijection defined as,*

$$\mathscr{A}_n \colon \{\overrightarrow{\boldsymbol{An}spars_i}\} \to \mathbb{P}_i \textit{ such that } \mathscr{A}_n(a_{i,j}) = \begin{cases} \rho_{i,j} & \textit{If } a_{i,j} \neq 0 \\ \{\emptyset\} & \textit{If } a_{i,j} = 0 \end{cases} \tag{6.5}$$

6.3.2 Attribute-parameter mapping function and construction of parameter subset

Here we introduce the notation that associates a parameter's sparsification attribute $(k, a_{i,j,k}) \in \{\overrightarrow{\mathbf{Ap}_{i,j}}\}$ to the parameter $w_{i,j,k}$ or $b_{i,j}$ in $\pi_{i,j}$.

Definition 10 (Attribute-parameter mapping function)**.** *The attribute-parameter mapping function $\mathscr{A}_p$ for the neuron layer i in the FFNN is a bijection defined as,*

$$\mathscr{A}_p \colon \{\overrightarrow{\boldsymbol{Ap}spars_{i,j}}\} \to \pi_{i,j} \ \textit{such that} \ \mathscr{A}_p(a_{i,j,k}) = \begin{cases} w_{i,j,k} \ \textit{or} \ b_{i,j} & \textit{If } a_{i,j,k} \neq 0 \\ \{\emptyset\} & \textit{If } a_{i,j,k} = 0 \end{cases} \tag{6.6}$$

Theorem 1. *If layer i in FFNN, has the set of perceptrons $\mathbb{P}_i$, set of parameters Π_i, parameter mapping function $\mathscr{P}$, sparsification function S,*

(a) Neuron sparsification attribute $\overrightarrow{\boldsymbol{An}_i}$, sparsified attribute $\overrightarrow{\boldsymbol{An}spars_i} = S(\overrightarrow{\boldsymbol{An}})$, attribute-neuron mapping function $\mathscr{A}_n$, $\mathbb{P}spars_i = \mathscr{A}_n(\{\overrightarrow{\boldsymbol{An}spars_i}\})$ and, $\Pi spars_i = \mathscr{P}(\mathbb{P}spars_i)$, then $\mathbb{P}spars_i \subseteq \mathbb{P}_i$, and $\Pi spars_i \subseteq \Pi_i$.

(b) Parameter sparsification attribute $\overrightarrow{\boldsymbol{Ap}_{i,j}}$, sparsified attribute $\overrightarrow{\boldsymbol{Ap}spars_{i,j}} = S(\overrightarrow{\boldsymbol{Ap}_{i,j}})$, attribute-parameter mapping function $\mathscr{A}_p$, $\pi spars_{i,j} = \mathscr{A}_p(\{\overrightarrow{\boldsymbol{Ap}spars_{i,j}}\})$, then $\bigcup_{j=1}^{|\mathbb{P}_i|} \pi spars_{i,j} \subseteq \bigcup_{j=1}^{|\mathbb{P}_i|} \pi_i$.

Chapter 7

Sparsified Backpropagation

Now that we have established the notation for backpropagation in chapter 3 and sparsification in chapter 4, we can define sparsified backpropagation. Sparsified backpropagation uses a sparsification function to construct a subset of the layer's parameters to calculate the gradient of the loss function. The resulting gradient is also sparsified and is a subset of the full gradient computed using all the neural network parameters. We use the *topk* sparsification function that has already been defined in chapter 4.

7.1 Topk sparsified backpropagation

The *topk* sparsification function [29] is applied to the sensitivity vector of each layer. This modifies backpropagation equations 5.8, 5.21 and 5.17 as follows

$$\frac{\partial L}{\partial \boldsymbol{W_i}} topk = topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right)\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}} \tag{7.1}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}} topk = \frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}} topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right) \tag{7.2}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{B_i}}} topk = topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right)\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \overrightarrow{\boldsymbol{B_i}}} \tag{7.3}$$

Thus, instead of using the entire sensitivity vector for a layer, only the k largest elements of the sensitivity, irrespective of their sign, are used in the JVP.

7.2 Complete sparsified backpropagation

For complete sparsified backpropagation, we introduced variation to the sparsified gradient by applying the *topk* sparsification function to the weight adjoint and the perceptron adjoint in the JVP for calculating the gradient. This modifies backpropagation equations 5.8 and 5.21 as follows

$$\frac{\partial L}{\partial \boldsymbol{W_i}}csa = topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right) topk\left(\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}}\right) \tag{7.4}$$

$$\frac{\partial L}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}}csa = topk\left(\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}}\right) topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right) \tag{7.5}$$

The remaining equation is the same as described in 7.1.

7.3 Partial connectivity sparsified backpropagation

For partial connectivity sparsified backpropagation, we apply the $topk_1$ sparsification function to the sensitivity of the layer and $topk_2$ sparsification function to the weight adjoint of the layer in the JVP. This modifies backpropagation equations 5.8 as follows

$$\frac{\partial L}{\partial \boldsymbol{W_i}}pca = topk_1\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right) topk_2\left(\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}}\right) \tag{7.6}$$

The remaining equations are the same as in 7.1.

Chapter 8

Asymmetric training

This chapter discusses how the *topk* sparsification function in sparsified backpropagation constructs the subset of parameters for calculating the gradient in the back-propagation phase of the training, thus inducing asymmetry.

Since no sparsification technique is used during the forward pass in any of our experiments with the sparsified backpropagation or its variations, the parameters used during forward propagation will remain the same.

Let θ_i be the set of parameters of the FFNN used for forward propagation for layer i, then $\theta_i = \bigcup_{j=1}^{|\mathbb{P}_i|} \pi_{i,j}$. Let Θ be the set of all parameters of the FFNN used for forward propagation, then $\Theta = \bigcup_{i=1}^{i=n} \theta_i$

8.1 Topk asymmetry

We refer to the asymmetry induced by topk sparsified backpropagation as topk asymmetry. In *topk* asymmetry for layer i in the FFNN, the sensitivity of layer i is the sparsification attribute for neuron selection, $\overrightarrow{\mathbf{A}_n}$ as defined in 9. Let $\phi_i topk$ be the set of parameters of layer i that is used in the backpropagation, then $\phi_i topk$ is given by theorem 1,

$$\phi_i topk = \mathscr{P}\left(\mathscr{A}_n\left(\left\{topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right)\right\}\right)\right) \tag{8.1}$$

Where $\mathscr{P}$ is the parameter mapping function 3 and $\mathscr{A}_n$ is the attribute neuron mapping function 9

Let the set of all parameters used in *topk* sparsified backpropagation for a FFNN with n layers be $\Phi topk$, then,

$$\Phi topk = \bigcup_{i=1}^{n} \phi_i topk \tag{8.2}$$

By theorem 1, $\Phi topk \subset \Theta$. Thus, sparsified backpropagation with *topk* induces asymmetry in the training of the FFNN.

8.1.1 Complete sparsification asymmetry

We refer to the asymmetry induced by complete sparsified backpropagation as complete sparsification asymmetry. In complete sparsification asymmetry, for layer i in the FFNN, the topk sparsification function is applied to the sensitivity of layer i, the weight adjoint of layer i, and the perceptron adjoint of layer i. The sensitivity and perceptron adjoint are the sparsification attributes for neuron selection in layer i. We computed the union of neuron subset constructed by both the sensitivity and the perceptron adjoint to find all the neurons that are selected to be used for backpropagation in layer i. It is given by the equation 8.3

$$\mathbb{P}csa_i = \mathscr{A}_n \left(\left\{ topk \left(\frac{\partial \overrightarrow{\mathbf{Lt}_i}}{\partial \overrightarrow{\boldsymbol{P_{i-1}}}} \right) \right\} \bigcup \left\{ topk \left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}} \right) \right\} \right) \tag{8.3}$$

The corresponding set of parameters for neuron subset $\mathbb{P}csa_i$ is given by $\mathscr{P}(\mathbb{P}csa_i)$

We also applied the sparsification function to the weight adjoint of layer i, so the weight adjoint is the sparsification attribute for parameter selection for each neuron in layer i. The set of parameters selected by the application of topk sparsification function on weight adjoint is denoted by Πcsa_i, where

$$\Pi csa_i = \mathscr{A}_p\left(\left\{topk\left(\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}}\right)\right\}\right) \tag{8.4}$$

Let $\phi_i csa$ be the set of parameters selected by complete sparsified backpropagation, then

$$\phi_i csa = \mathscr{P}(\mathbb{P}csa_i)\bigcap \Pi csa_i \tag{8.5}$$

Let the set of all parameters used in complete sparsified backpropagation for a FFNN with n layers be Φcsa, then,

$$\Phi csa = \bigcup_{i=1}^{n} \phi_i csa \tag{8.6}$$

Where, $\Phi csa \subset \Theta$. Thus, complete sparsified backpropagation induces asymmetry in the training of the FFNN.

8.1.2 Partial connectivity asymmetry

For layer i in the FFNN, complete sparsification asymmetry has redundancy in neuron selection, as we applied the topk sparsification function to both sensitivity and the neuron adjoint. So for partial connectivity asymmetry, we do not apply the *topk* sparsification method to the neuron adjoint of the layer; instead, we only apply the *topk* sparsification method to the sensitivity and the weight adjoint of the layer. We also vary the value of k for the sensitivity and weight adjoint. The sensitivity is the sparsification attribute for neuron selection, and the weight adjoint is the sparsification attribute for parameter selection. Equations 8.7 and 8.8 represent the neurons and parameters after the application of the $topk_1$ and $topk_2$ sparsification functions.

$$\mathbb{P}pca_i = \mathscr{A}_n\left(\left\{topk\left(\frac{\partial L}{\partial \overrightarrow{\boldsymbol{Lt_i}}}\right)\right\}\right) \tag{8.7}$$

$$\Pi pca_i = \mathscr{A}_p \left(\left\{ topk_2 \left(\frac{\partial \overrightarrow{\boldsymbol{Lt_i}}}{\partial \boldsymbol{W_i}} \right) \right\} \right) \tag{8.8}$$

Let $\phi_i pca$ be the set of parameters selected by partial connectivity backpropagation, then

$$\phi_i pca = \mathscr{P}(\mathbb{P}pca_i) \bigcap \Pi pca_i \tag{8.9}$$

Let the set of all parameters used in partial connectivity backpropagation for a FFNN with n layers be Φpca, then,

$$\Phi pca = \bigcup_{i=1}^{n} \phi_i pca \tag{8.10}$$

Where, $\Phi pca \subset \Theta$. Thus, partial connectivity backpropagation induces asymmetry in the training of the FFNN.

Chapter 9

Experiments

We conduct two set of experiments.

1. The first set of experiments were conducted on the MNIST dataset to explore the sparsified backpropagation and how it can induce asymmetric training in context of multi-layer perceptron.

2. The second set of experiments were conducted on the Fashion MNIST dataset, we consider different architectures and study how varying the width and depth of the FFNN effects the behaviour of asymmetric training.

9.1 Setup

The experiments were conducted using python [30], PyTorch [24]. Additional libraries like numpy[17] and scipy [31] were also used. Plotly [19] was used for the visualization of results.

9.1.1 MNIST experiments setup

The first set of the experiments were conducted on the MNIST [22] dataset consisting of $60,000$ training images and $10,000$ test images. The size of each image is 28×28

pixels. The experimental setup for the first set of experiments is identical to the experimental setup as described in [29], this is the reason why our experiments on mnist dataset were conducted on a FFNN with one hidden layer containing 512 units and the training was done for 17 epochs. Each image contains a single digit from 0 to 9.

9.1.2 Fashion MNIST experiments setup

The second set of experiments were conducted on the Fashion mnist [34] dataset. The fashion mnist dataset if very similar to the mnist dataset, as it also consists of 60,000 training images and 10,000 test images. The images are 28 $\times$28 pixels of photographs of different clothes in grayscale. We conducted around 100 experiments over 25 different architectures of FFNN. Each architecture is represented as li-j, where i is the number of hidden layers and j is the number of neurons per layer. We list the architectures used for the experiments.

1. l1-64- One hidden layer with 64 neurons.
2. l2-64- Two hidden layers with 64 neurons each.
3. l3-64- Three hidden layers with 64 neurons each.
4. l4-64- Four hidden layers with 64 neurons each.
5. l5-64- Five hidden layers with 64 neurons each.
6. l1-128- One hidden layer with 128 neurons.
7. l2-128- Two hidden layers with 128 neurons each.
8. l3-128- Three hidden layers with 128 neurons each.

9. l4-128- Four hidden layers with 128 neurons each.
10. l5-128- Five hidden layers with 128 neurons each.
11. l1-256- One hidden layer with 256 neurons.
12. l2-256- Two hidden layers with 256 neurons each.
13. l3-256- Three hidden layers with 256 neurons each.
14. l4-256- Four hidden layers with 256 neurons each.
15. l5-256- Five hidden layers with 256 neurons each.
16. l1-512- One hidden layer with 512 neurons.
17. l2-512- Two hidden layers with 512 neurons each.
18. l3-512- Three hidden layers with 512 neurons each.
19. l4-512- Four hidden layers with 512 neurons each.
20. l5-512- Five hidden layers with 512 neurons each.
21. l1-1024- One hidden layer with 1024 neurons.
22. l2-1024- Two hidden layers with 1024 neurons each.
23. l3-1024- Three hidden layers with 1024 neurons each.
24. l4-1024- Four hidden layers with 1024 neurons each.
25. l5-1024- Five hidden layers with 1024 neurons each.

For the fashion mnist experiments, all models were trained for 20 epochs. This was determined using early stopping, with the stopping criteria being a reduction in validation accuracy.

The first $5,000$ training images in both the datasets were set aside and used for validation. Xavier Glorot's initialization [15] method was used to initialize all the parameters of the network. ReLU activation function was used for non-linear transformation. Adam [21] optimizer was used with the following hyper-parameters, learning rate, $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

All experiments were conducted on four variations of backpropagation that are described in chapter 5.

1. Symmetric training with dropout, with the dropout rate of 0.4, was used to establish the baseline for all comparisons. The complete gradient was calculated with normal backpropagation.

2. For asymmetric training, selected the following values for the top k function because they gave better accuracy on the mnist dataset as compared to FFNN with symmetric training. We decided to use the same parameters for the fashion mnist experiments as well since both the datasets are similar.

   a) Topk asymmetry: $k = 40$.

   b) Complete Sparsified asymmetry: $k = 170$.

   c) Partial connectivity: $k_1 = 40$ and $k_2 = 200$.

9.2 MNIST Results

We present the following data for each of our experiments.

1. Validation accuracy

2. Test accuracy

3. Backpropagation time

Tables 9.2 and 9.1 display the time and accuracy for different backpropagation methods.

9.2.1 Symmetric training with drop-out

The FFNN with symmetric training with drop-out had the test accuracy (Figure 9.1) of 97.98%, in the 95% confidence interval of $(97.9\%, 98.06\%)$, and the validation accuracy (Figure 9.2) of 99.4%, in the 95% confidence interval of $(99.29\%, 99.87\%)$.

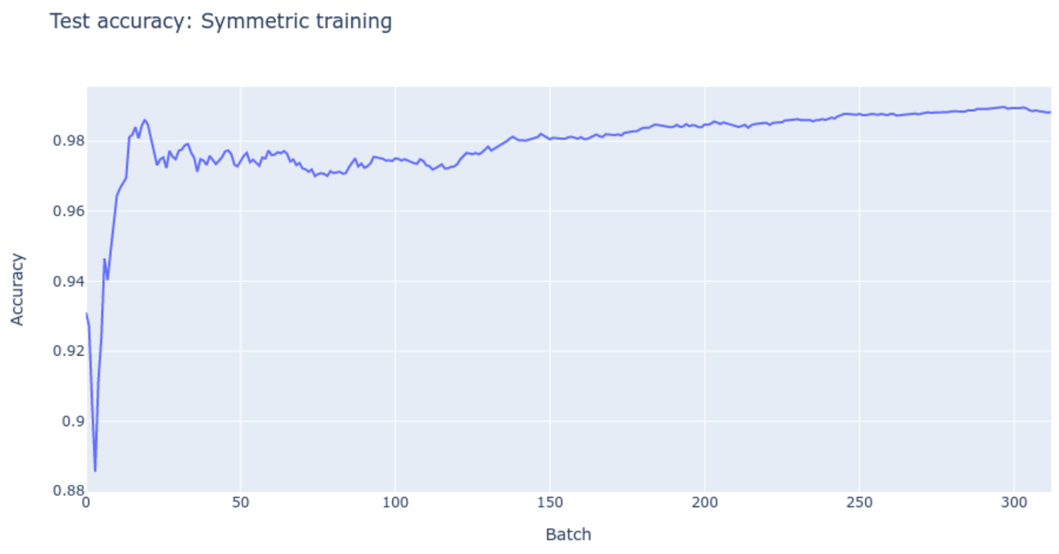

Figure 9.1: Test accuracy for Symmetric training

9.2.2 Asymmetric training with topk sparsification

The FFNN with asymmetric training with topk sparsification had the test accuracy (Figure 9.3) of 98.19%, in the 95% confidence interval of $(98.13\%, 98.25\%)$. And

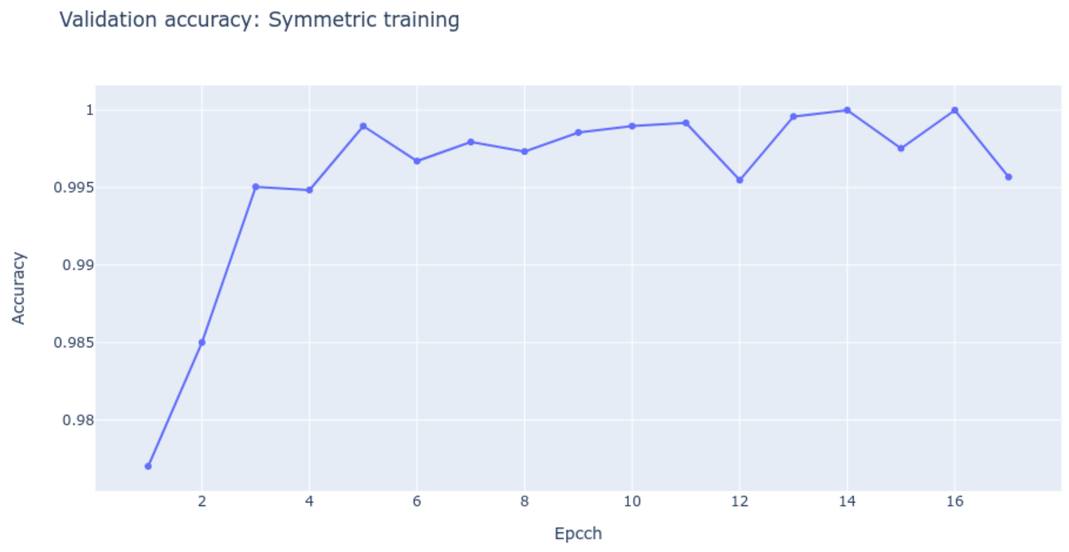

Figure 9.2: Validation accuracy for Symmetric training

the validation accuracy (Figure 9.4) of 98.30%, in the 95% confidence interval of (98.02%, 98.58%).

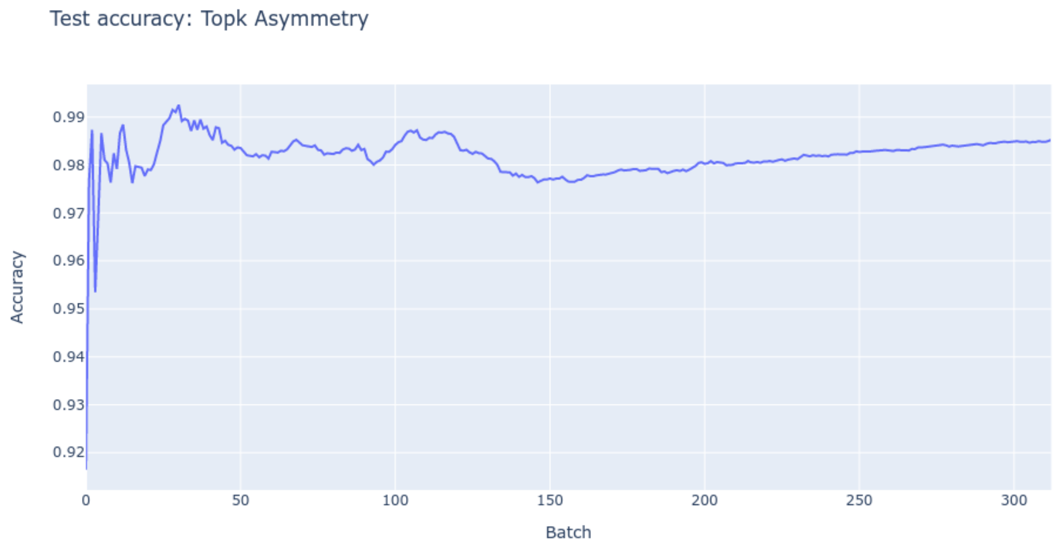

Figure 9.3: Test accuracy for Topk Asymmetric training

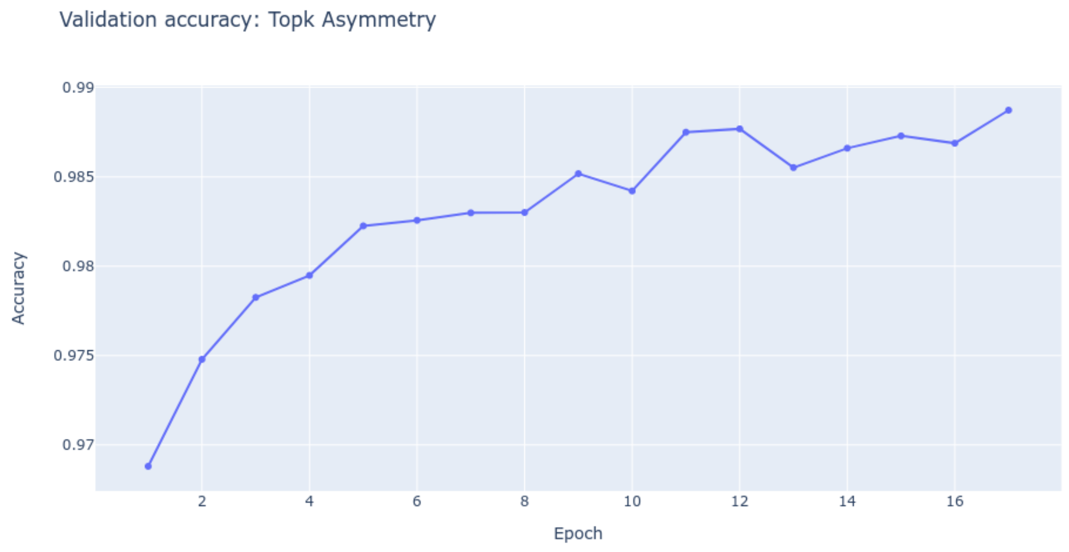

Figure 9.4: Validation accuracy for Topk Asymmetric training

9.2.3 Asymmetric training with complete sparsification

The FFNN with complete sparsification asymmetric training using topk had the test accuracy (Figure 9.5) of 98.002%, in the 95% confidence interval of (97.97%, 98.034%). And the validation accuracy (Figure 9.6) of 98.58%, in the 95% confidence interval of (98.40%, 98.76%). Higher values of k were used for complete sparsification asymmetric training to get comparable results which also increased the backpropagation time. The complete sparsification asymmetry method has an intrinsic redundancy 8.1.1 because we use the union of sensitivity and perceptron adjoint to select the neurons for the layer.

These findings of the complete sparsification asymmetric training led to the third method, which uses different values of k for the sensitivity and the weight adjoint to decouple the neuron selection and parameter selection from each other.

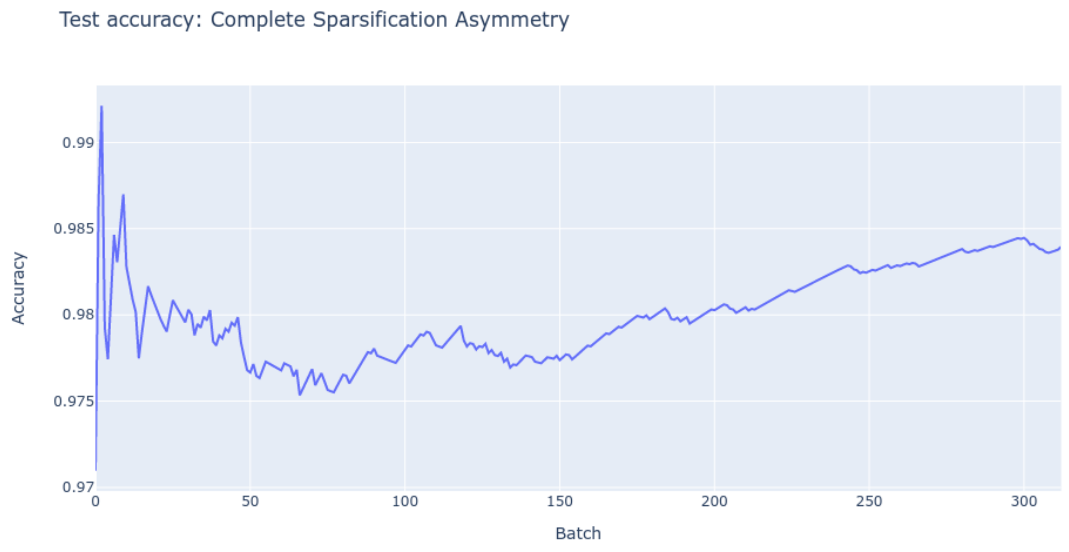

Figure 9.5: Test accuracy for Complete Sparsification Asymmetric training

9.2.4 Asymmetric training with partial connectivity

The FFNN with asymmetric training with partial connectivity using topk had the test accuracy (Figure 9.7) of 98.10%, in the 95% confidence interval of (98.01%, 98.19%). And the validation accuracy (Figure 9.8) of 98.28%, in the 95% confidence interval of (98.10%, 98.46%)

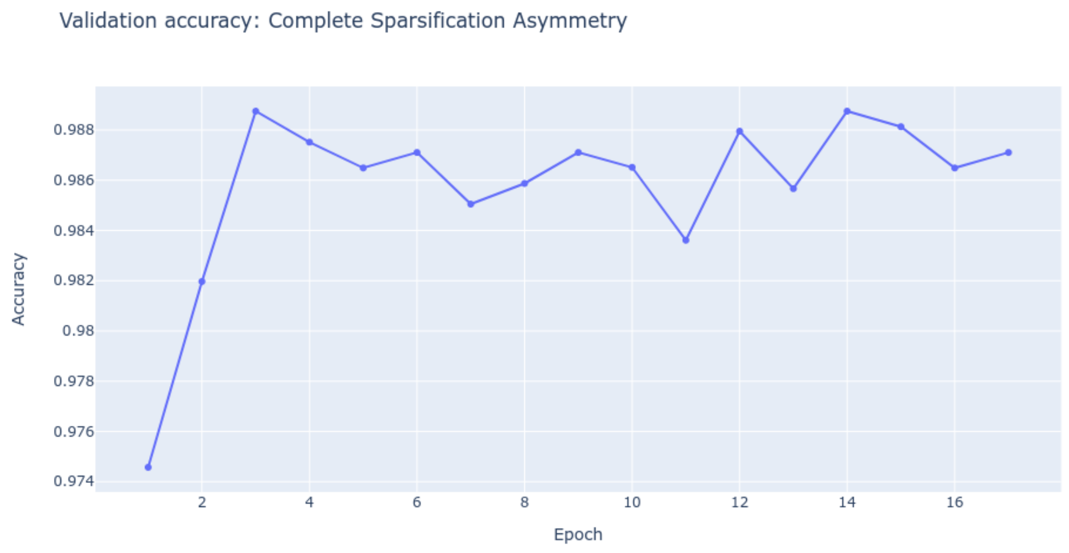

Figure 9.6: Validation accuracy for Complete Sparsification Asymmetric training

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.994 (0.9929, 0.9987)	0.9798 (0.979, 0.9806)
Topk k=40	0.9830 (0.9802, 0.9858)	0.9819 (0.9813 0.9825)
Partial Connectivity k1=40, k2=200	0.98580 (0.9840, 0.9876)	0.98002 (0.9797, 0.98034)
Complete Sparsification Asymmetry k=170	0.9828 (0.9810, 0.9846)	0.9810 (0.9801, 0.9819)

Table 9.1: Accuracy table for different methods for mnist

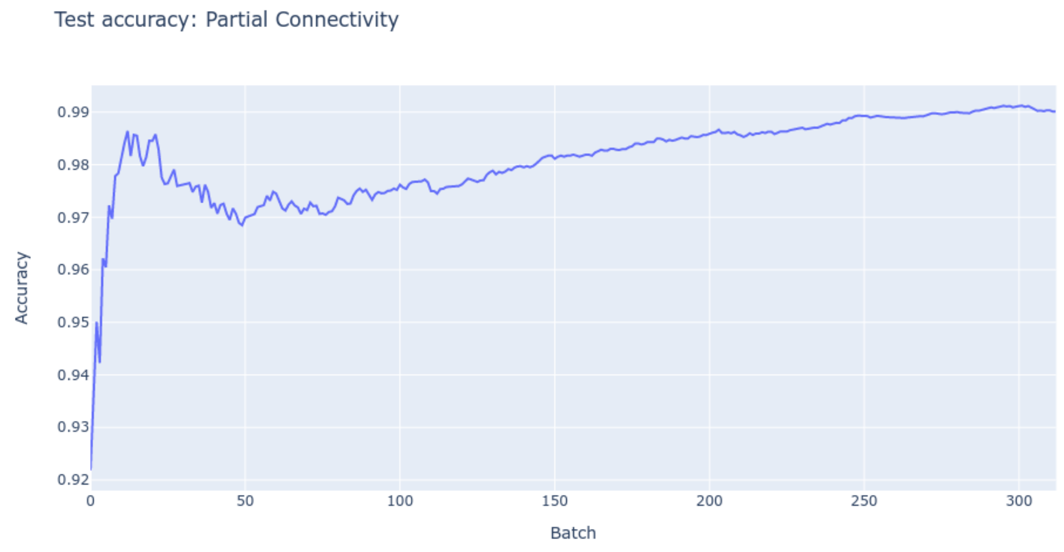

Figure 9.7: Test accuracy for Partial Connectivity Asymmetric training

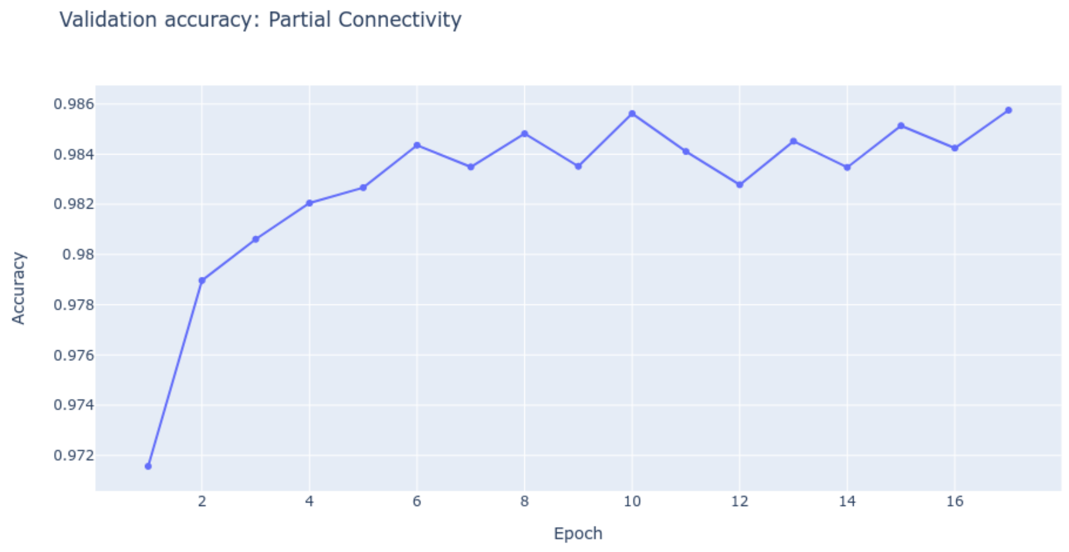

Figure 9.8: Validation accuracy for Partial Connectivity Asymmetric training

9.3 Time

In backpropagation, the calculation of JVP takes the most time. Sparsified backpropagation significantly reduces the time taken for the calculation of JVP. However, there is an additional overhead of application of the sparsification function to the sparsification attributes for subset construction. The max-heap time refers to the time the *topk* sparsification function takes to select the neurons and parameters with the *k* largest sparsification attribute regardless of magnitude. The total backpropagation time is the sum of the time needed to calculate the JVP and the max-heap. We also observe that applying the sparsification function multiple times to different sparsification attributes increases the max-heap time in the backpropagation. Therefore, complete sparsification asymmetry has the maximum max-heap time, followed by partial connectivity asymmetry, and the top k asymmetry has the least max-heap time.

We find that the total backpropagation time in asymmetric training is still less than the backpropagation time of symmetric training with dropout, as shown in Figure 9.9

Table 9.2 aggregates the total backpropagation time for each training method.

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	4818.70	2978.8	3212.9	4647.8

Table 9.2: Backpropagation time table for different training methods for mnist

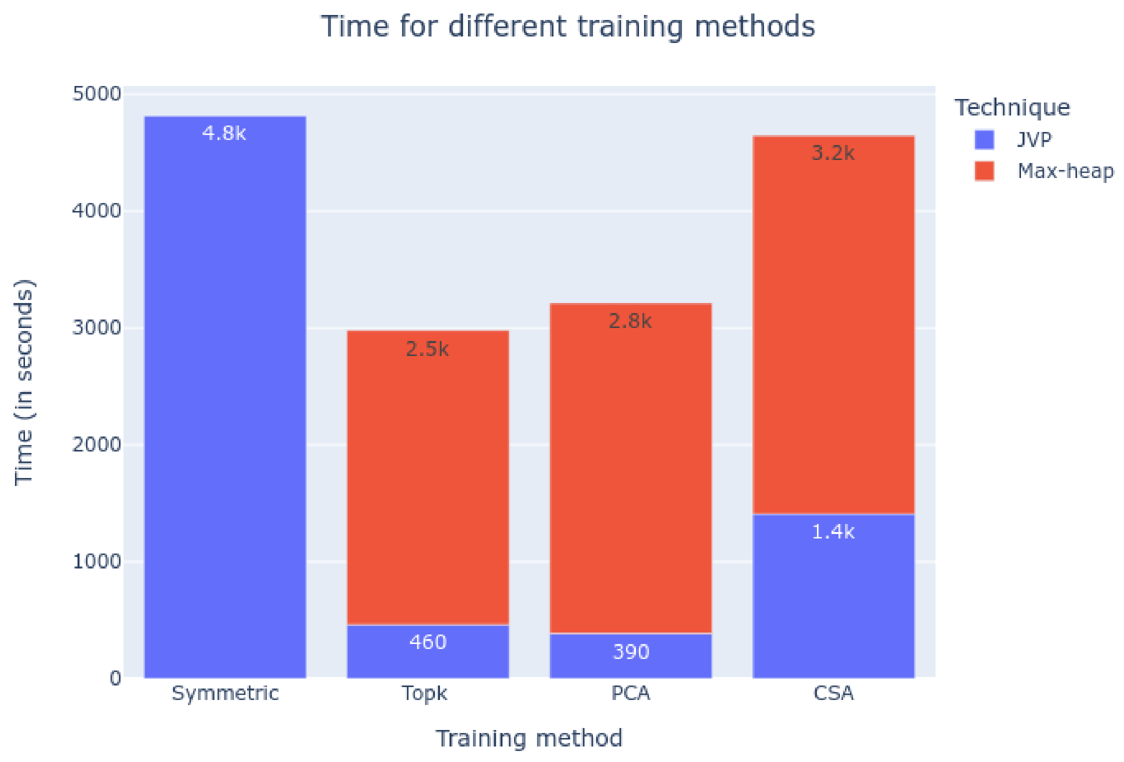

Figure 9.9: Backpropagation time for different training methods for mnist

9.4 Fashion MNIST Results

We report the results of the experiments on the architectures mentioned in 9.1.2

1. One hidden layer with 64 neurons.

    a) Accuracy

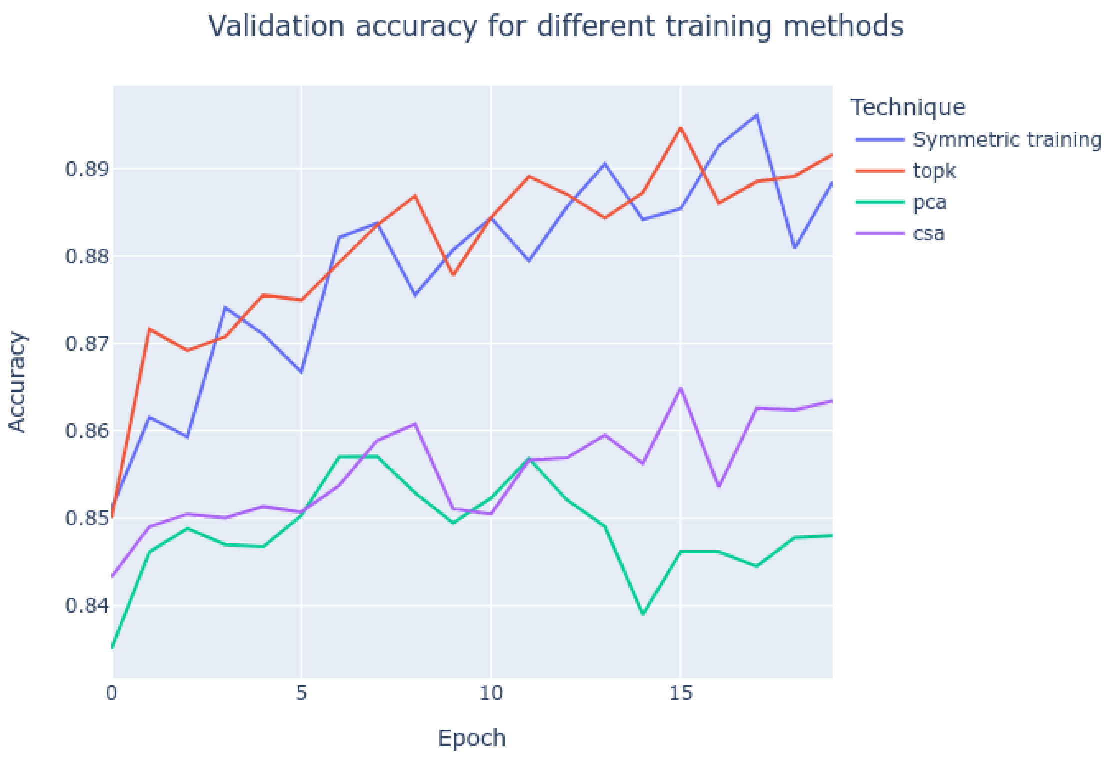

Figure 9.10: Validation accuracy for different training methods for l1-64

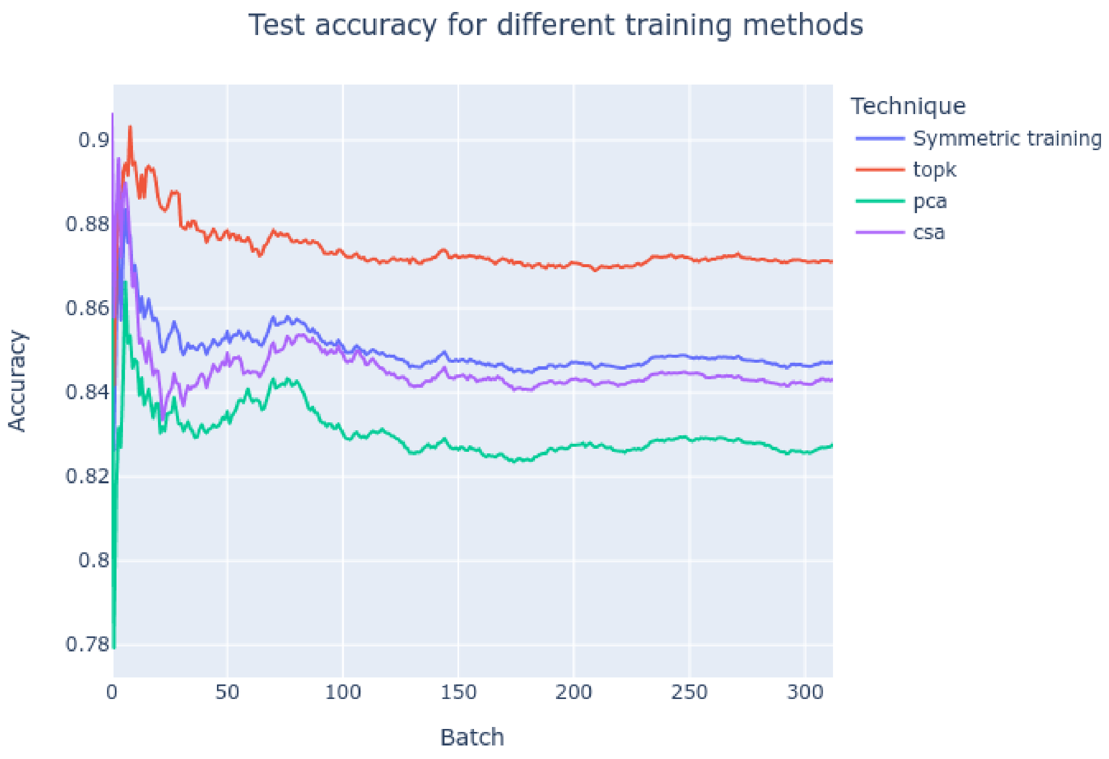

Figure 9.11: Test accuracy for different training methods for l1-64

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8786 (0.8731, 0.8841)	0.8500 (0.8493, 0.8506)
Topk Asymmetry k=40	0.8810 (0.8762, 0.8859)	0.8740 (0.8733, 0.8746)
Partial Connectivity Asymmetry k1=40, k2=200	0.8486 (0.8460, 0.8512)	0.8202 (0.8195, 0.8210)
Complete Sparsification Asymmetry k=170	0.8553 (0.8525, 0.8580)	0.8457 (0.8448, 0.8466)

Table 9.3: Accuracy table for l1-64

b) Time

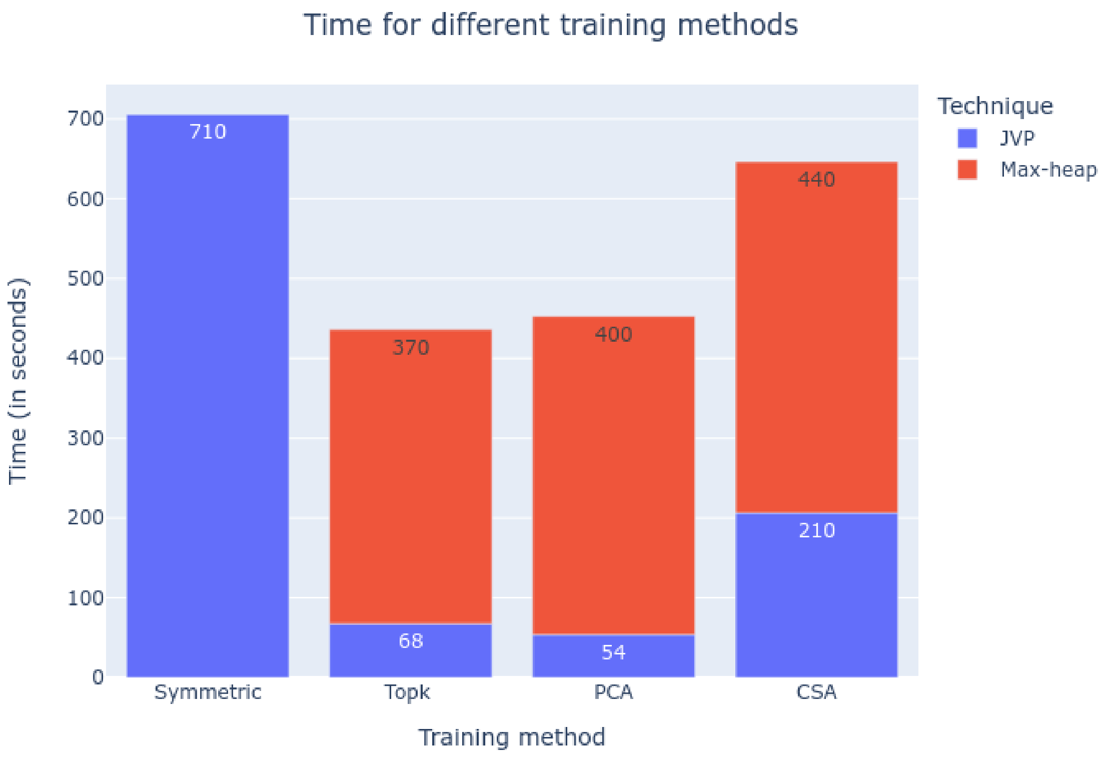

Figure 9.12: Backpropagation time for different training methods for l1-64

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	706.04	435.96	452.66	646.07

Table 9.4: Backpropagation time table for different training methods for l1-64

2. Two hidden layers with 64 neurons each.

   a) Accuracy

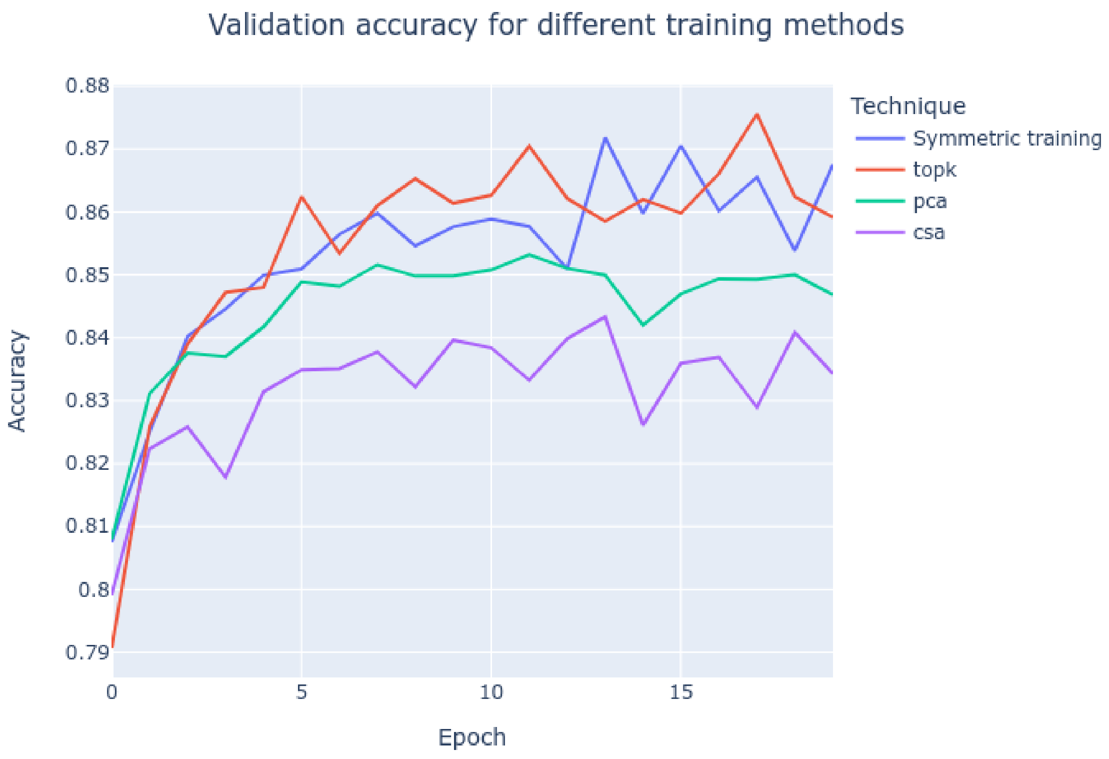

Figure 9.13: Validation accuracy for different training methods for l2-64

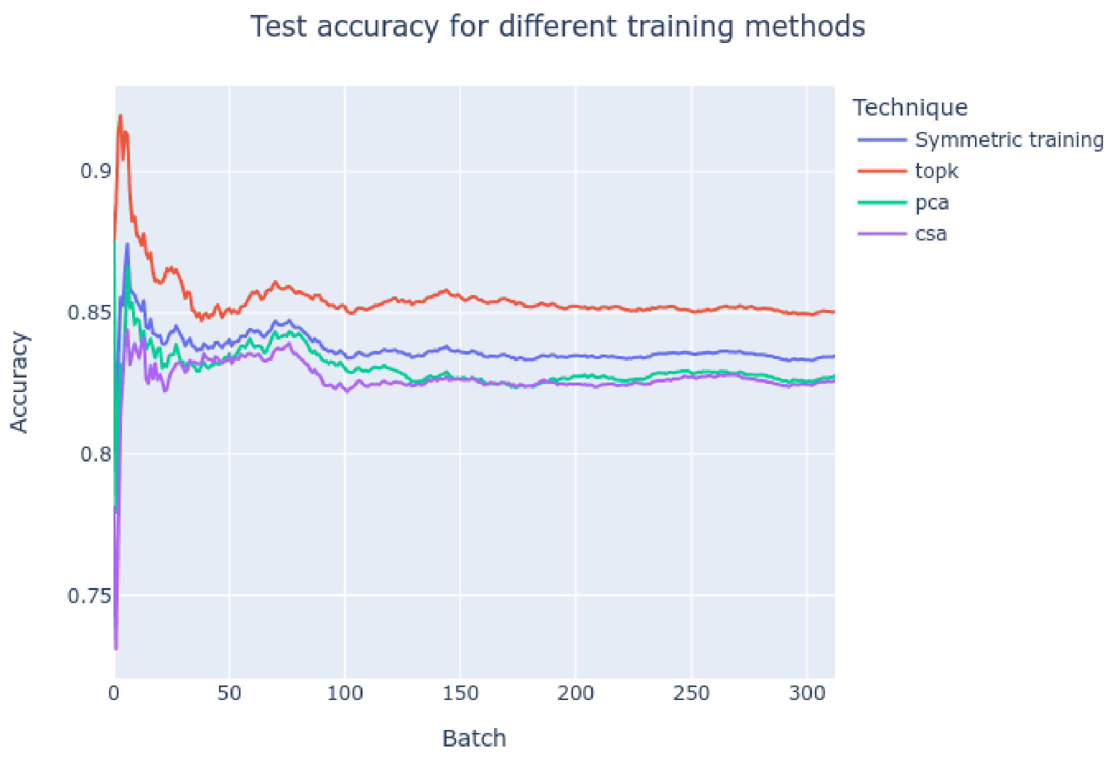

Figure 9.14: Test accuracy for different training methods for l2-64

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8531 (0.8460, 0.8602)	0.8374 (0.8368, 0.8380)
Topk Asymmetry k=40	0.8546 (0.8459, 0.8634)	0.8552 (0.8541, 0.8562)
Partial Connectivity Asymmetry k1=40, k2=200	0.8446 (0.8397, 0.8495)	0.8298 (0.8287, 0.8310)
Complete Sparsification Asymmetry k=170	0.8316 (0.8269, 0.8364)	0.8268 (0.8259, 0.8277)

Table 9.5: Accuracy table for l2-64

b) Time

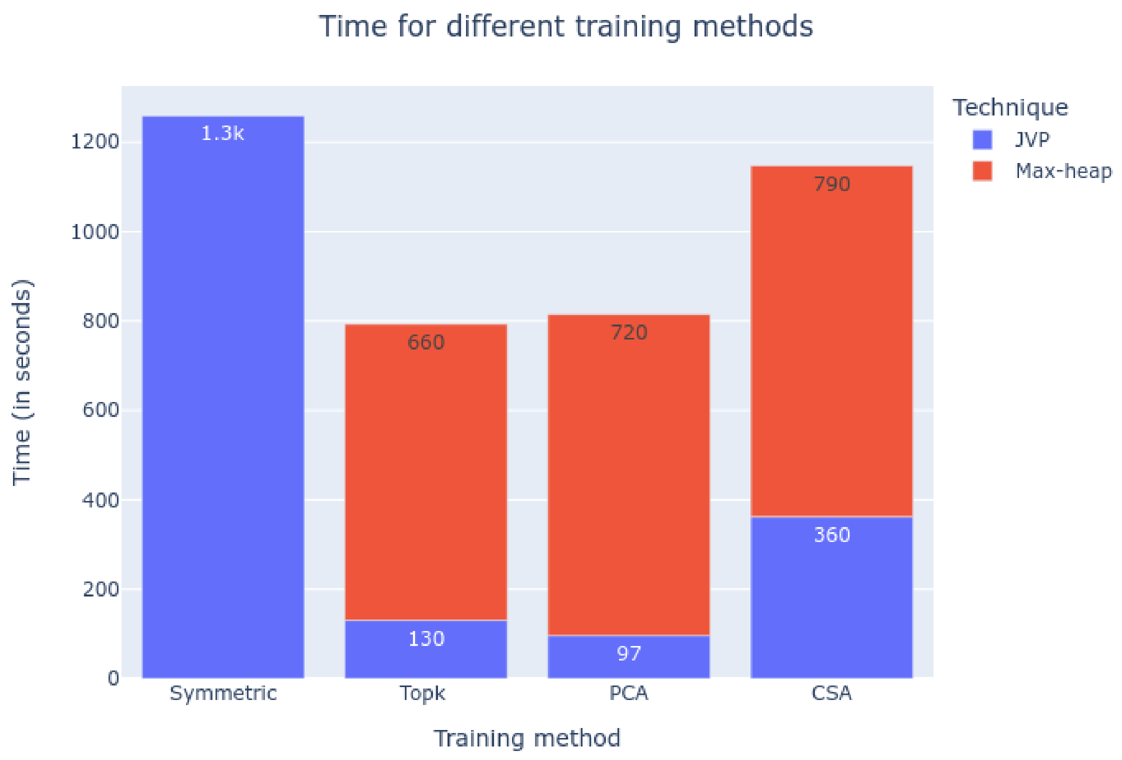

Figure 9.15: Backpropagation time for different training methods for l2-64

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	1259.81	792.91	815.31	1147.31

Table 9.6: Backpropagation time table for different training methods for l2-64

3. Three hidden layers with 64 neurons each.

   a) Accuracy

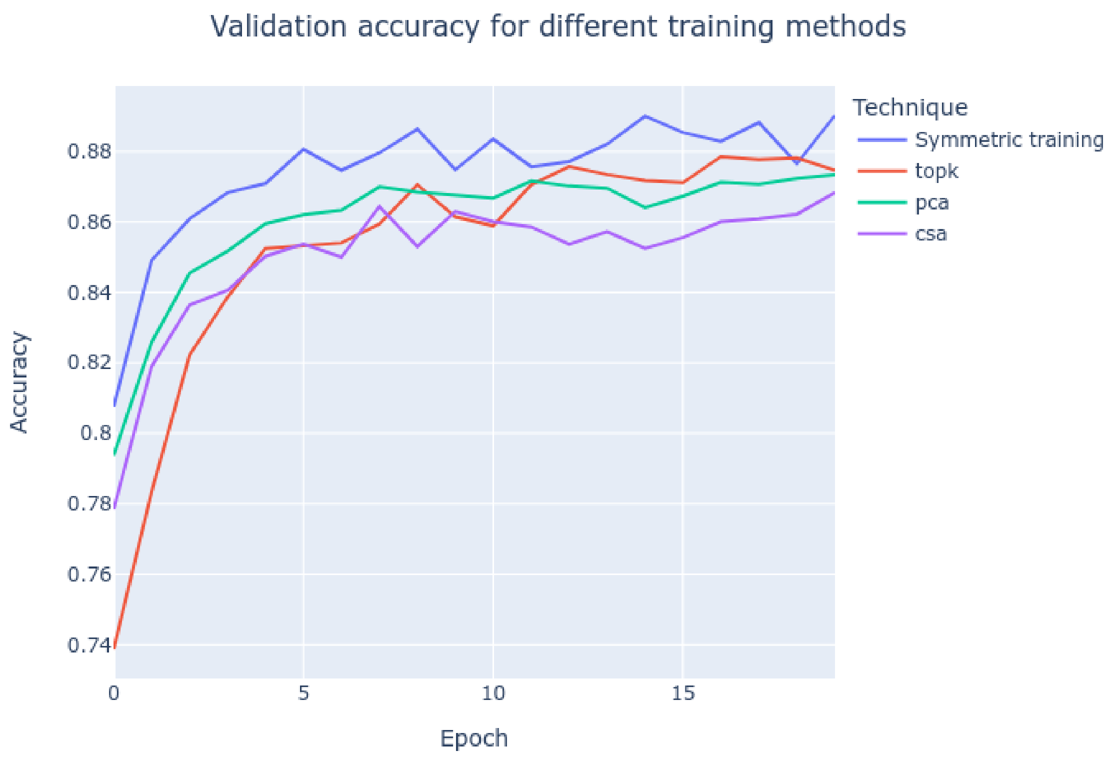

Figure 9.16: Validation accuracy for different training methods for l3-64

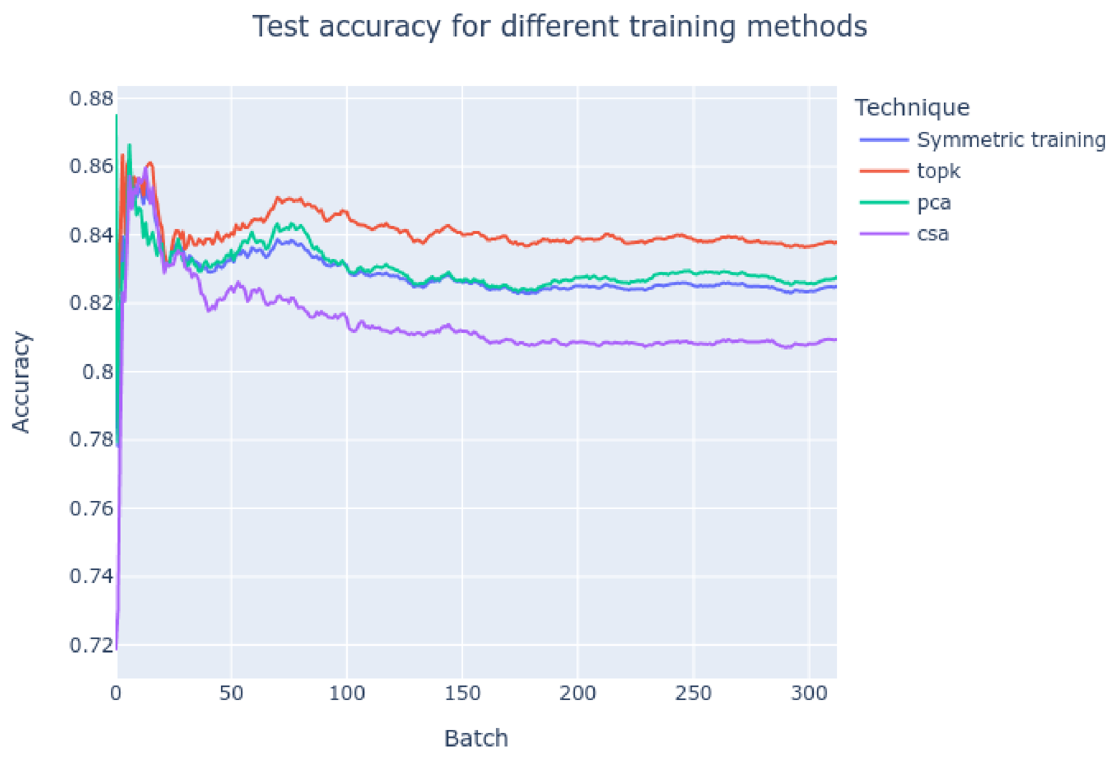

Figure 9.17: Test accuracy for different training methods for l3-64

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8742 (0.8655, 0.8829)	0.8284 (0.8276, 0.8293)
Topk Asymmetry k=40	0.8532 (0.8366, 0.8698)	0.8409 (0.8403, 0.8415)
Partial Connectivity Asymmetry k1=40, k2=200	0.8602 (0.8512, 0.8693)	0.8307 (0.8295, 0.8320)
Complete Sparsification Asymmetry k=170	0.8499 (0.8404, 0.8593)	0.8142 (0.8128, 0.8156)

Table 9.7: Accuracy table for l3-64

b) Time

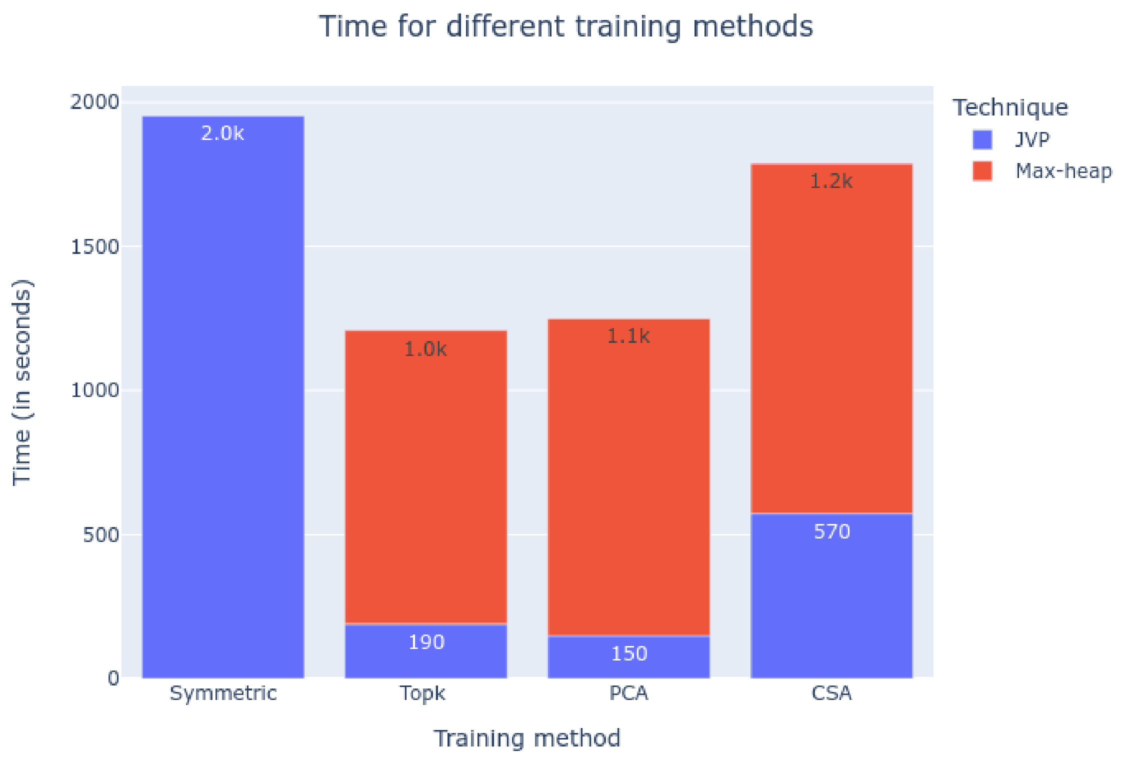

Figure 9.18: Backpropagation time for different training methods for l3-64

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	1954.64	1208.12	1249.29	1788.04

Table 9.8: Backpropagation time table for different training methods for l3-64

4. Four hidden layers with 64 neurons each.

    a) Accuracy

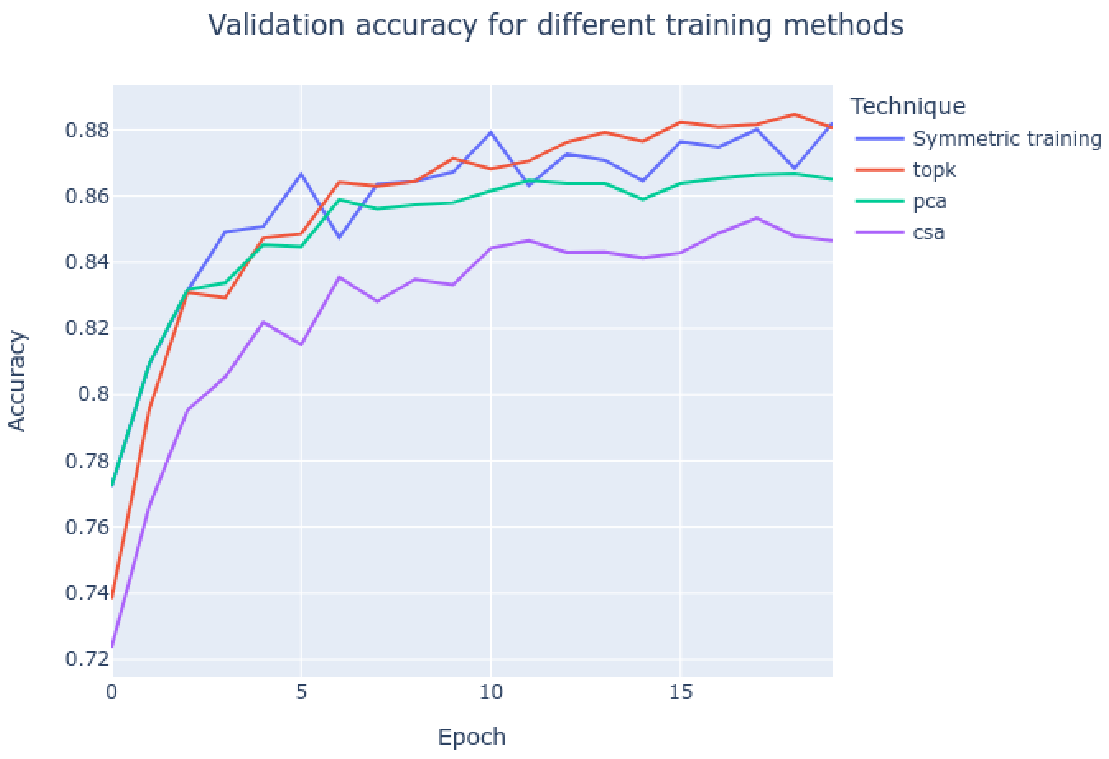

Figure 9.19: Validation accuracy for different training methods for l4-64

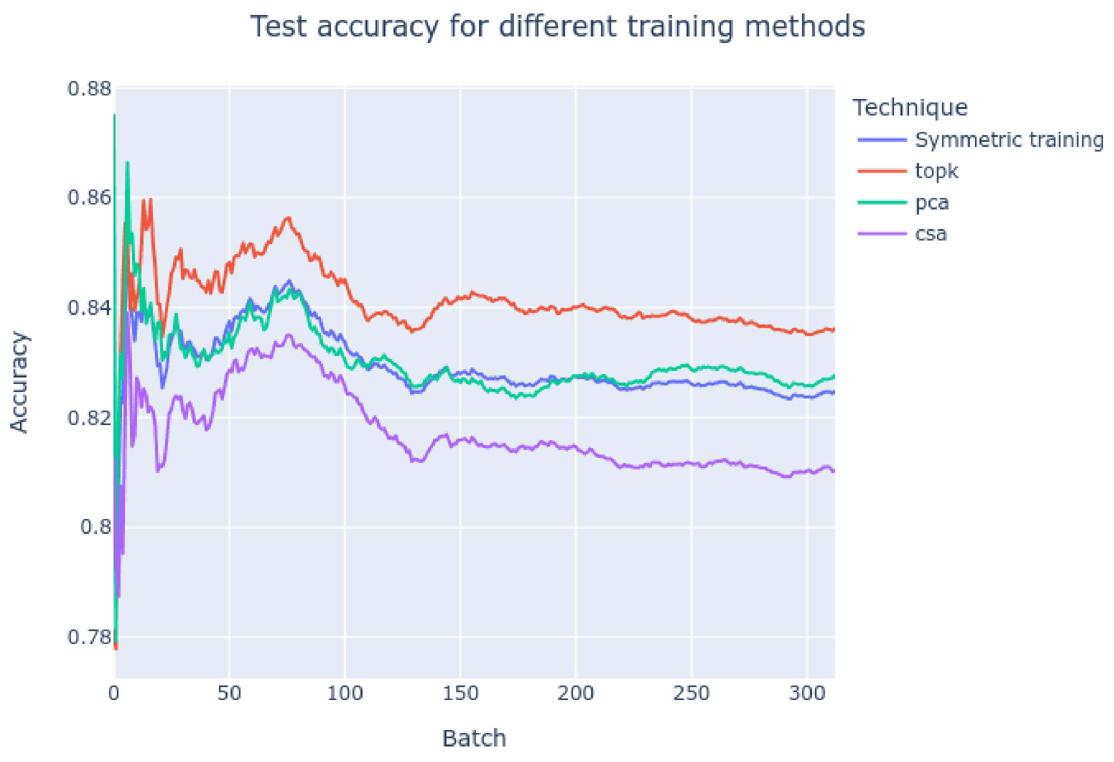

Figure 9.20: Test accuracy for different training methods for l4-64

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8477 (0.8351, 0.8602)	0.8295 (0.8288, 0.8302)
Topk Asymmetry k=40	0.8466 (0.8298, 0.8634)	0.8412 (0.8403, 0.8420)
Partial Connectivity Asymmetry k1=40, k2=200	0.8403 (0.8293, 0.8514)	0.8299 (0.8289, 0.8310)
Complete Sparsification Asymmetry k=170	0.8158 (0.8007, 0.8309)	0.8172 (0.8163, 0.8180)

Table 9.9: Accuracy table for l4-64

b) Time

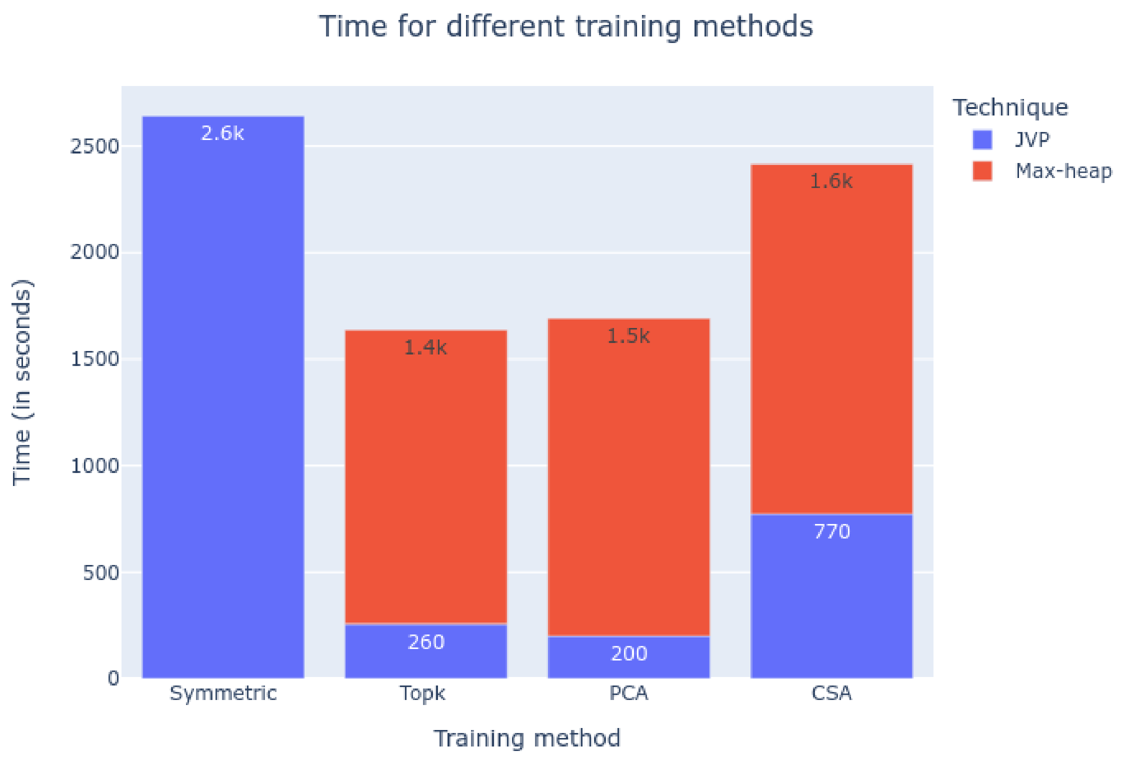

Figure 9.21: Backpropagation time for different training methods for l4-64

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	2643.79	1636.47	1691.16	2416.27

Table 9.10: Backpropagation time table for different training methods for l4-64

5. Five hidden layers with 64 neurons each.

   a) Accuracy

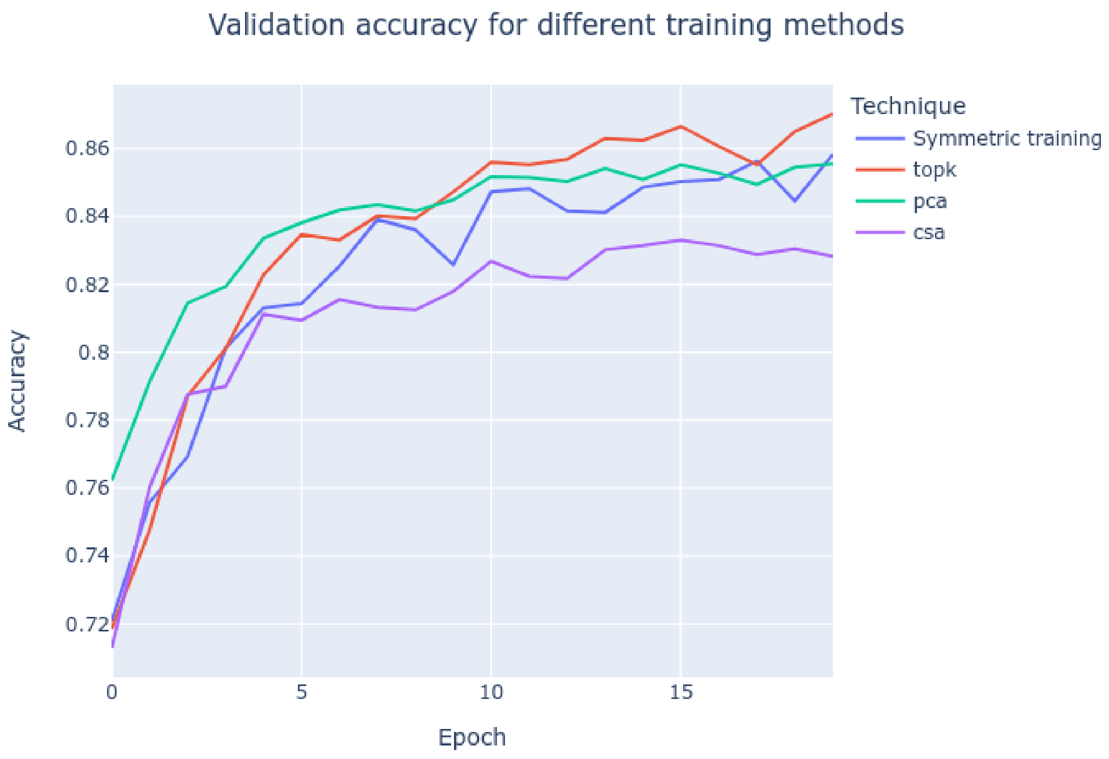

Figure 9.22: Validation accuracy for different training methods for l5-64

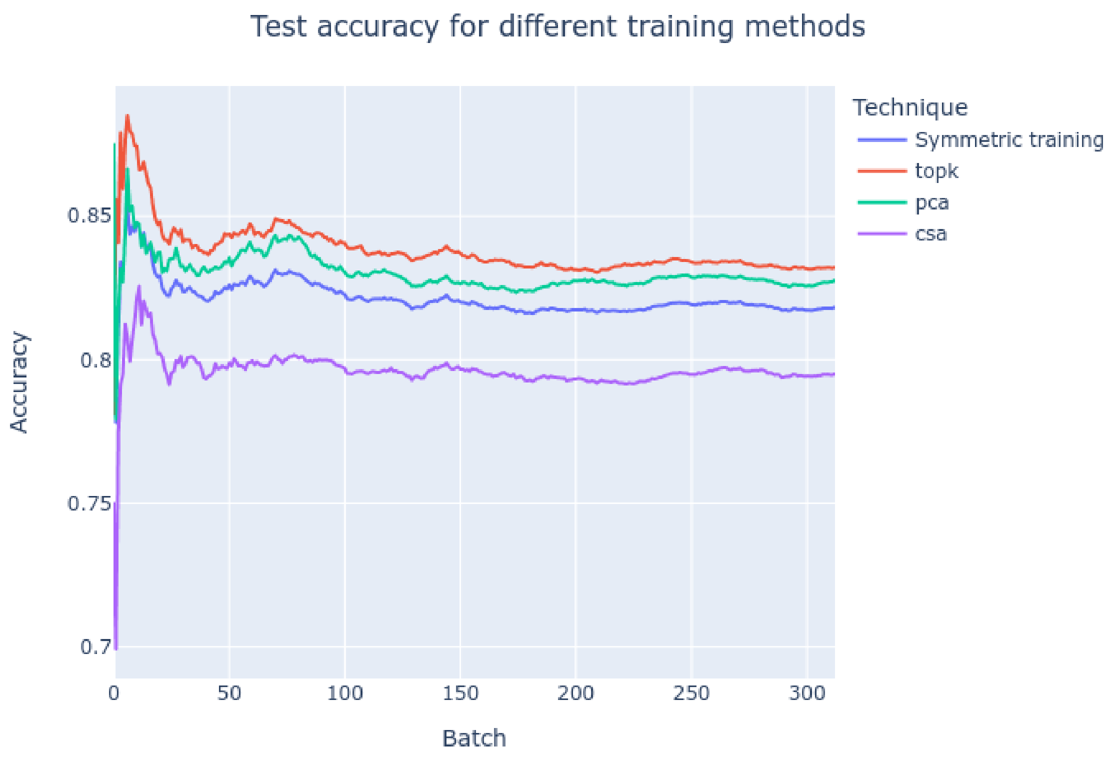

Figure 9.23: Test accuracy for different training methods for l5-64

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8243 (0.8071, 0.8416)	0.8215 (0.8207, 0.8222)
Topk Asymmetry k=40	0.8451 (0.8269, 0.8633)	0.8381 (0.8371, 0.8392)
Partial Connectivity Asymmetry k1=40, k2=200	0.8378 (0.8265, 0.8491)	0.8302 (0.8295, 0.8310)
Complete Sparsification Asymmetry k=170	0.8107 (0.7969, 0.8245)	0.7961 (0.7952, 0.7969)

Table 9.11: Accuracy table for l5-64

b) Time

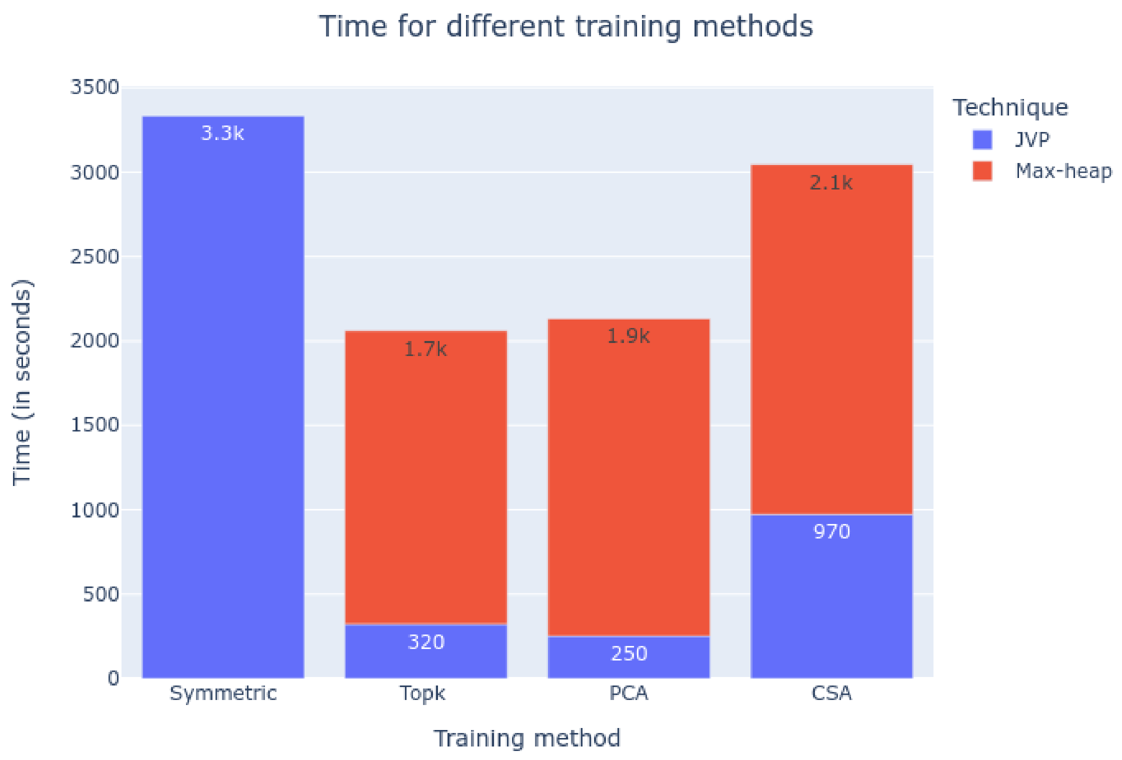

Figure 9.24: Backpropagation time for different training methods for l5-64

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	3335.98	2060.71	2131.4	3046.87

Table 9.12: Backpropagation time table for different training methods for l5-64

6. One hidden layer with 128 neurons.

    a) Accuracy

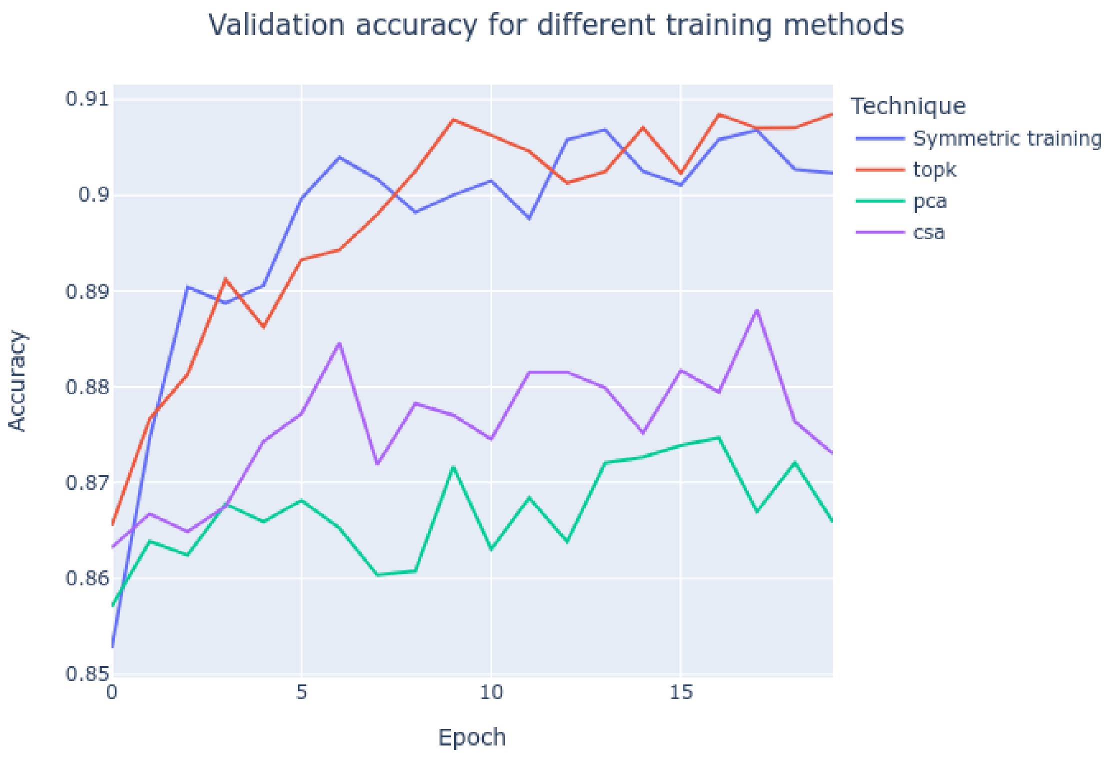

Figure 9.25: Validation accuracy for different training methods for l1-128

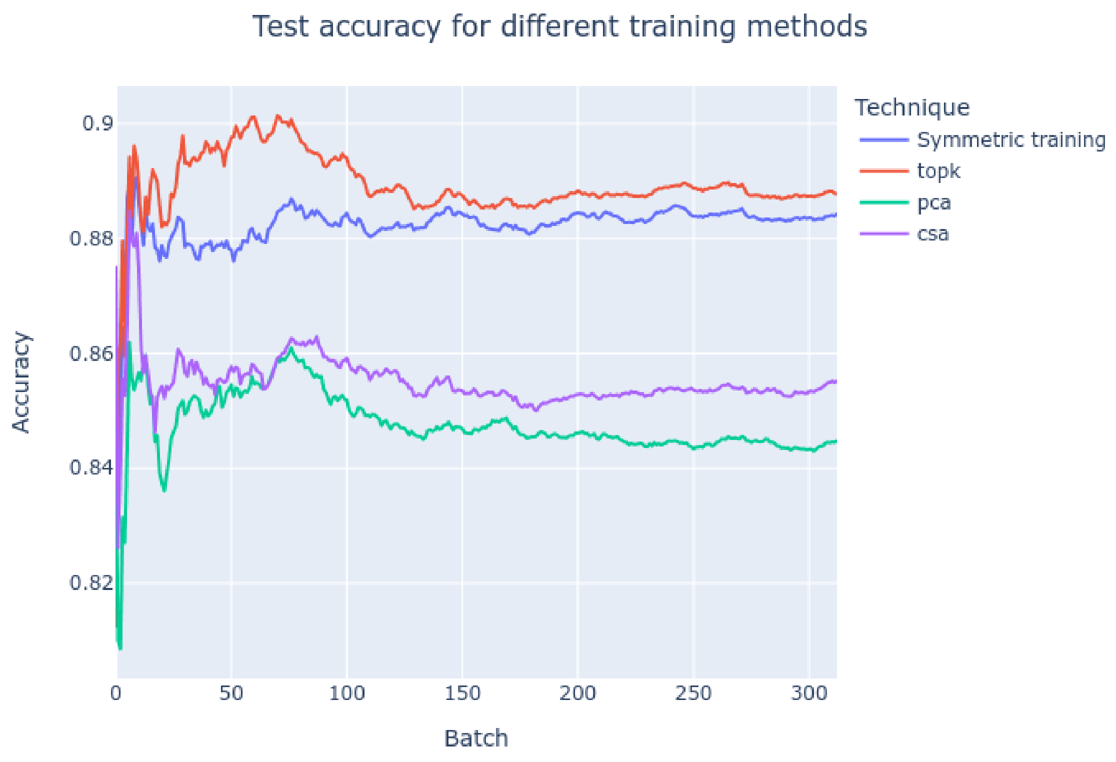

Figure 9.26: Test accuracy for different training methods for l1-128

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8966 (0.8906, 0.9027)	0.8823 (0.8819, 0.8828)
Topk Asymmetry k=40	0.8975 (0.8919, 0.9032)	0.8891 (0.8883, 0.8898)
Partial Connectivity Asymmetry k1=40, k2=200	0.8668 (0.8645, 0.8691)	0.8475 (0.8469, 0.8482)
Complete Sparsification Asymmetry k=170	0.8758 (0.8727, 0.8789)	0.8550 (0.8545, 0.8556)

Table 9.13: Accuracy table for l1-128

b) Time

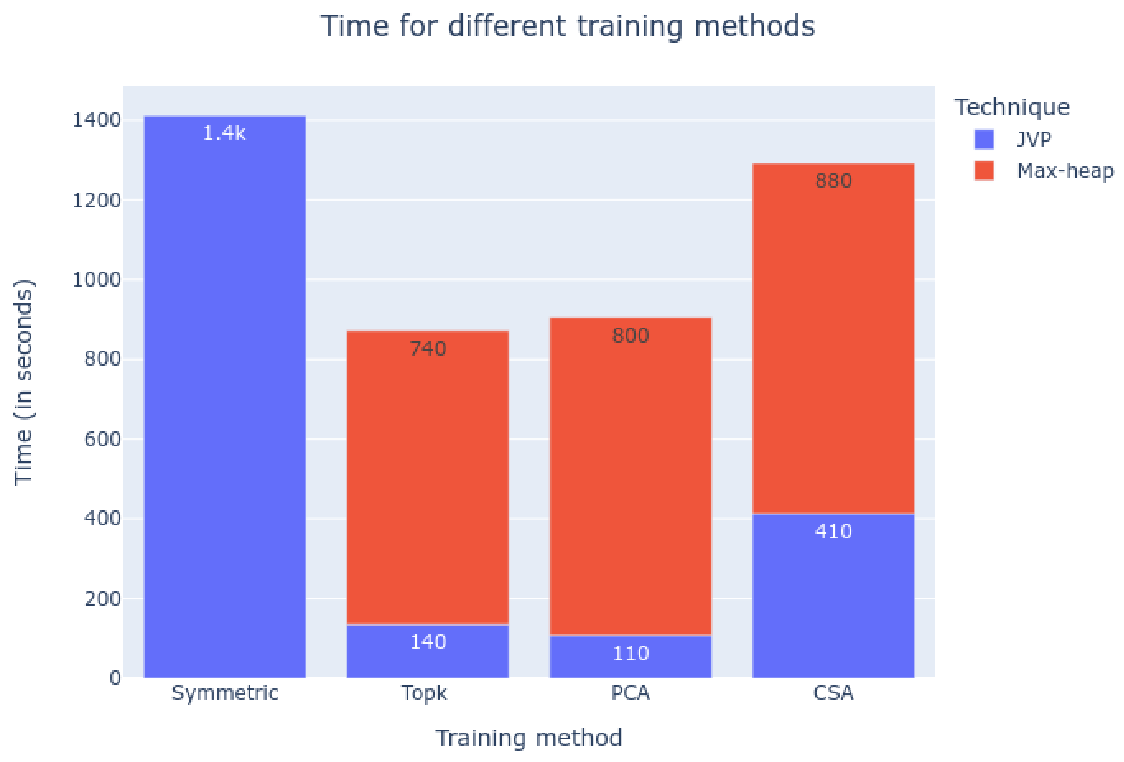

Figure 9.27: Backpropagation time for different training methods for l1-128

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	1412.07	871.91	905.32	1292.13

Table 9.14: Backpropagation time table for different training methods for l1-128

7. Two hidden layers with 128 neurons each.

   a) Accuracy

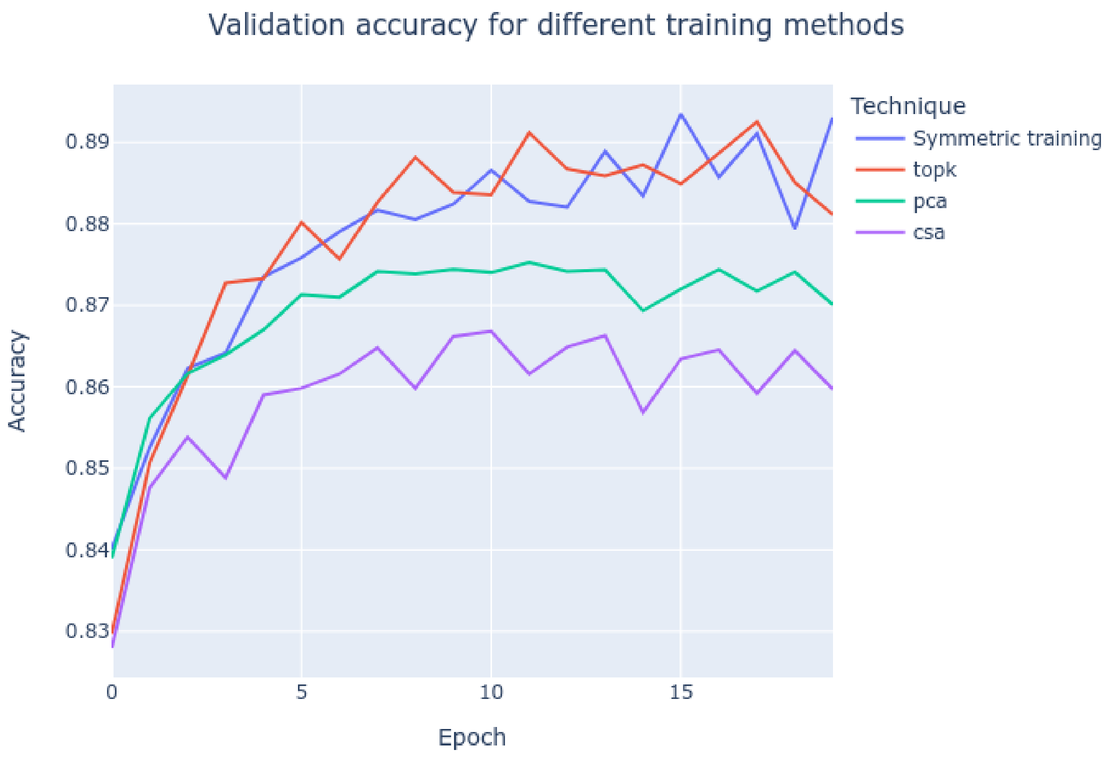

Figure 9.28: Validation accuracy for different training methods for l2-128

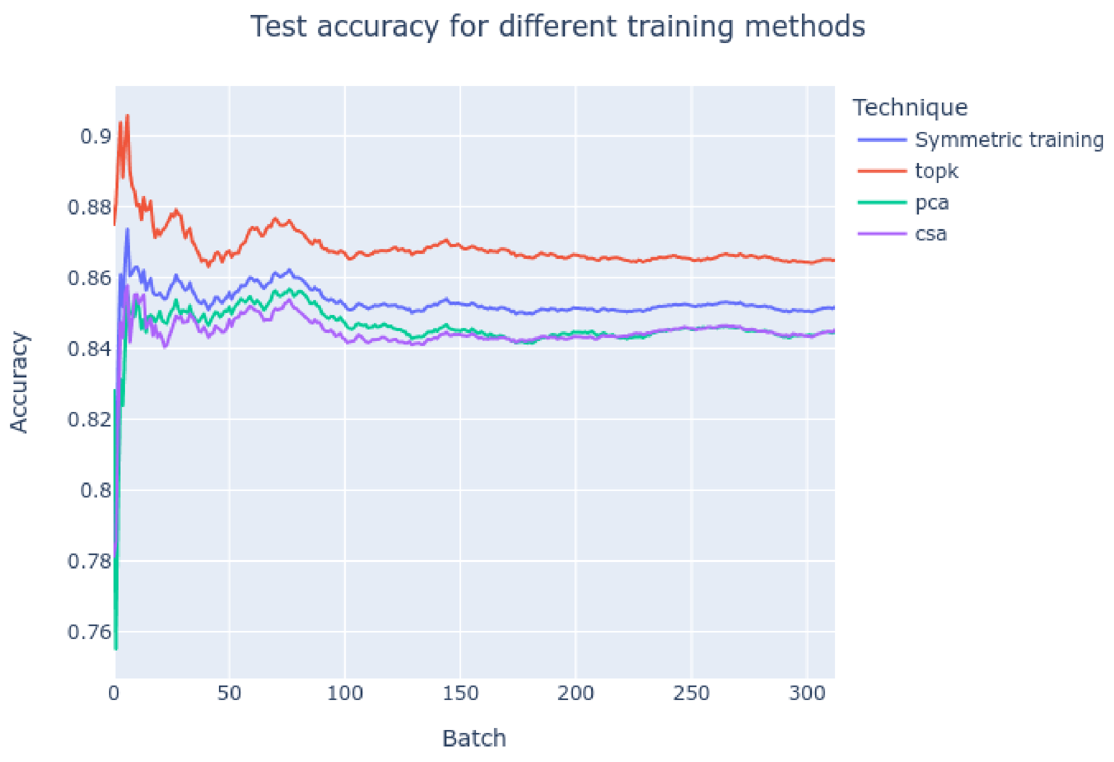

Figure 9.29: Testing accuracy for different training methods for l2-128

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8779 (0.8715, 0.8843)	0.8530 (0.8525, 0.8535)
Topk Asymmetry k=40	0.8782 (0.8711, 0.8854)	0.8686 (0.8679, 0.8692)
Partial Connectivity Asymmetry k1=40, k2=200	0.8691 (0.8650, 0.8731)	0.8459 (0.8451, 0.8466)
Complete Sparsification Asymmetry k=170	0.8588 (0.8546, 0.8630)	0.8445 (0.8439, 0.8452)

Table 9.15: Accuracy table for l2-128

b) Time

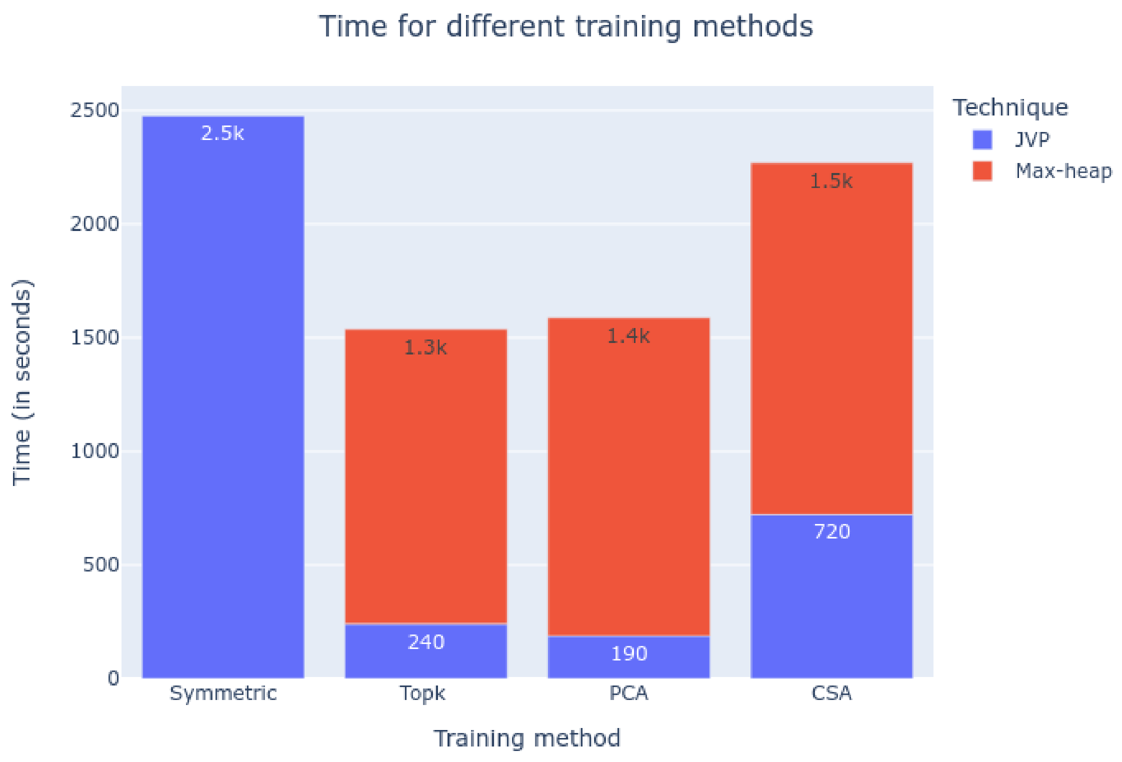

Figure 9.30: Backpropagation time for different training methods for l2-128

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	2477.62	1536.79	1587.64	2268.53

Table 9.16: Backpropagation time table for different training methods for l2-128

8. Three hidden layers with 128 neurons each.

   a) Accuracy

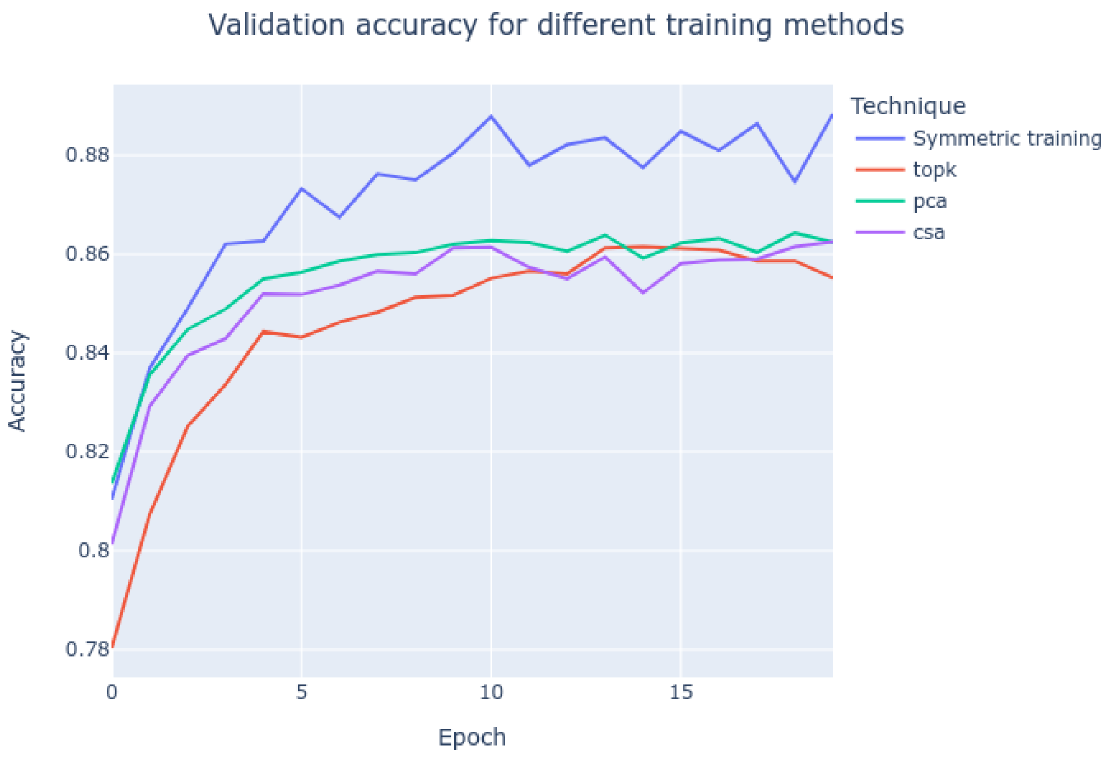

Figure 9.31: Validation accuracy for different training methods for l3-128

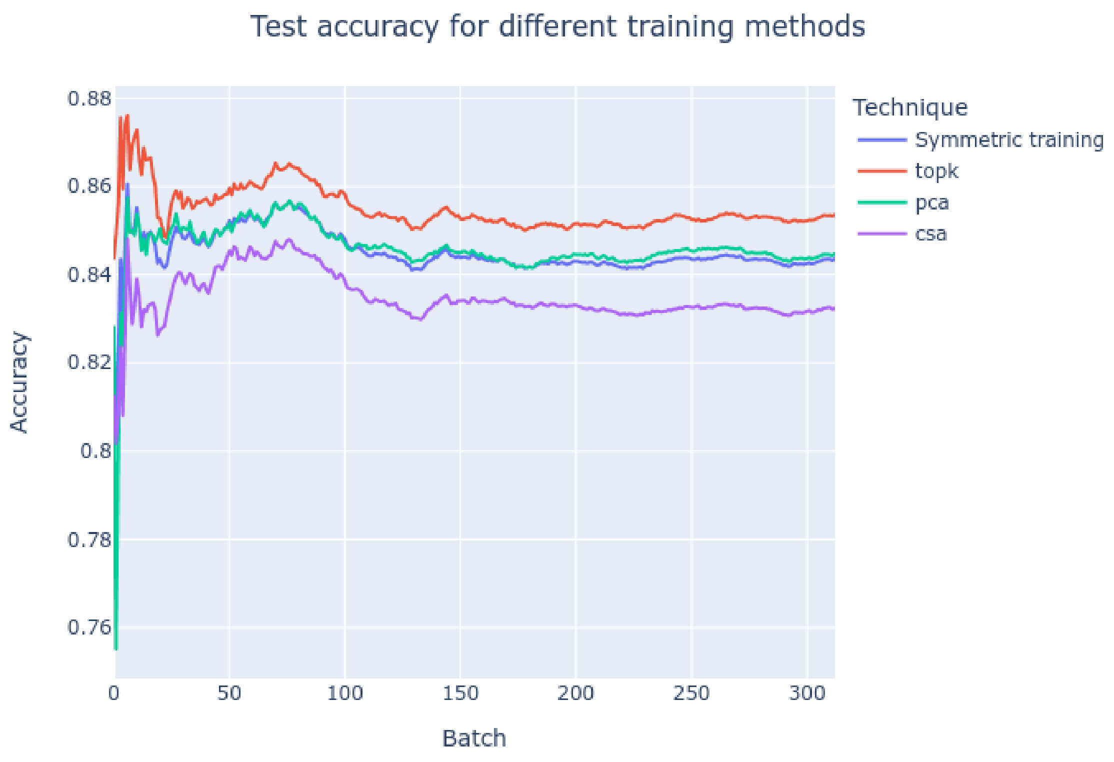

Figure 9.32: Test accuracy for different training methods for l3-128

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8708 (0.861, 0.879)	0.8451 (0.8446, 0.8457)
Topk Asymmetry k=40	0.8458 (0.8361, 0.8554)	0.8548 (0.8543, 0.8554)
Partial Connectivity Asymmetry k1=40, k2=200	0.8558 (0.8500, 0.8615)	0.8460 (0.8451, 0.8470)
Complete Sparsification Asymmetry k=170	0.8514 (0.8447, 0.8582)	0.8347 (0.8340, 0.8353)

Table 9.17: Accuracy table for l3-128

b) Time

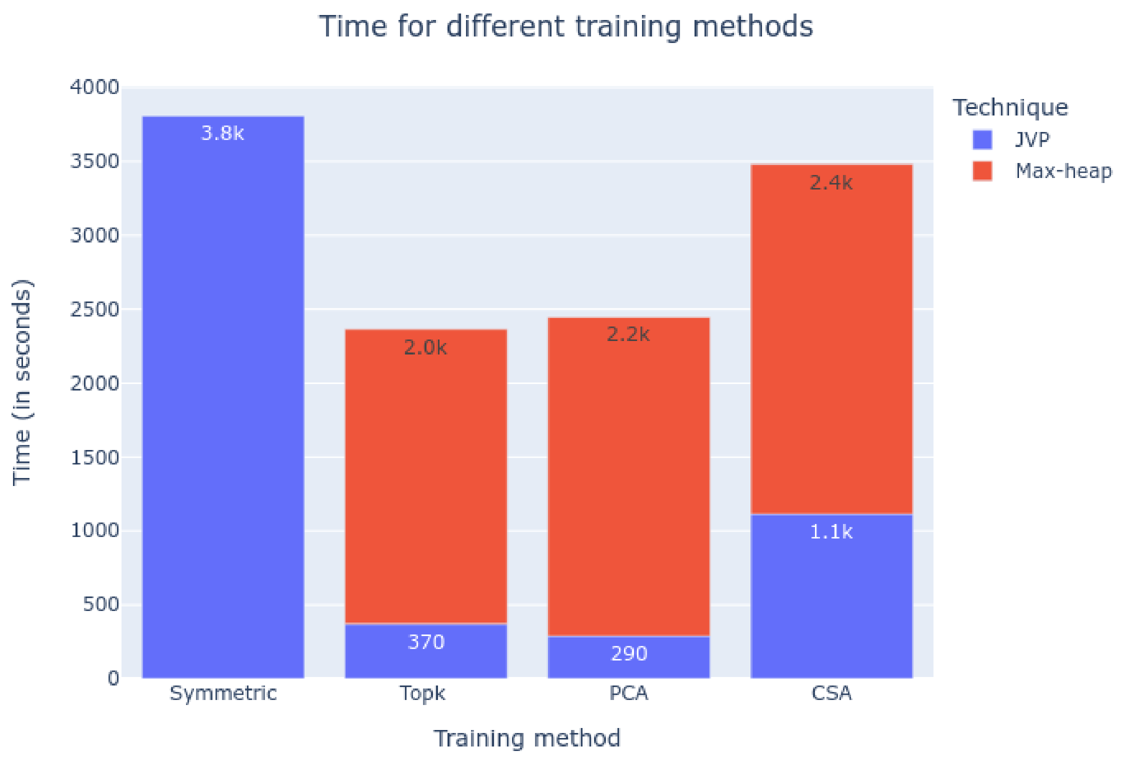

Figure 9.33: Backpropagation time for different training methods for l3-128

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	3811.12	2364.23	2446.35	3481.34

Table 9.18: Backpropagation time table for different training methods for l3-128

9. Four hidden layers with 128 neurons each.

   a) Accuracy

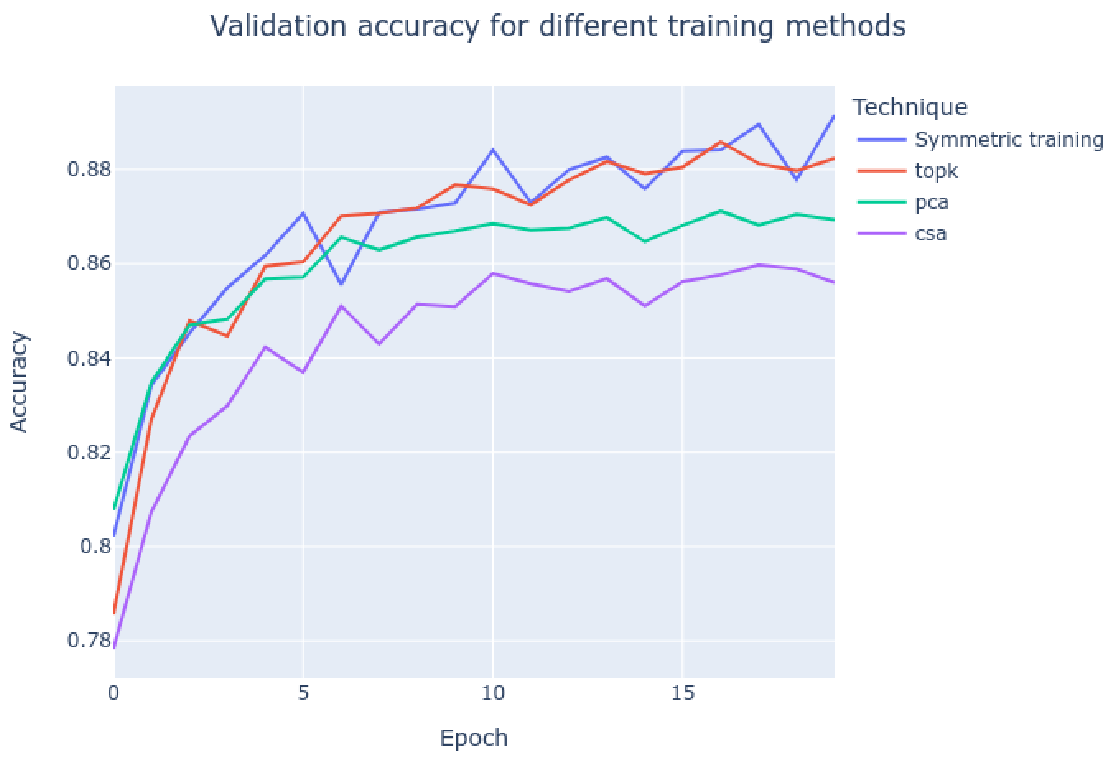

Figure 9.34: Validation accuracy for different training methods for l4-128

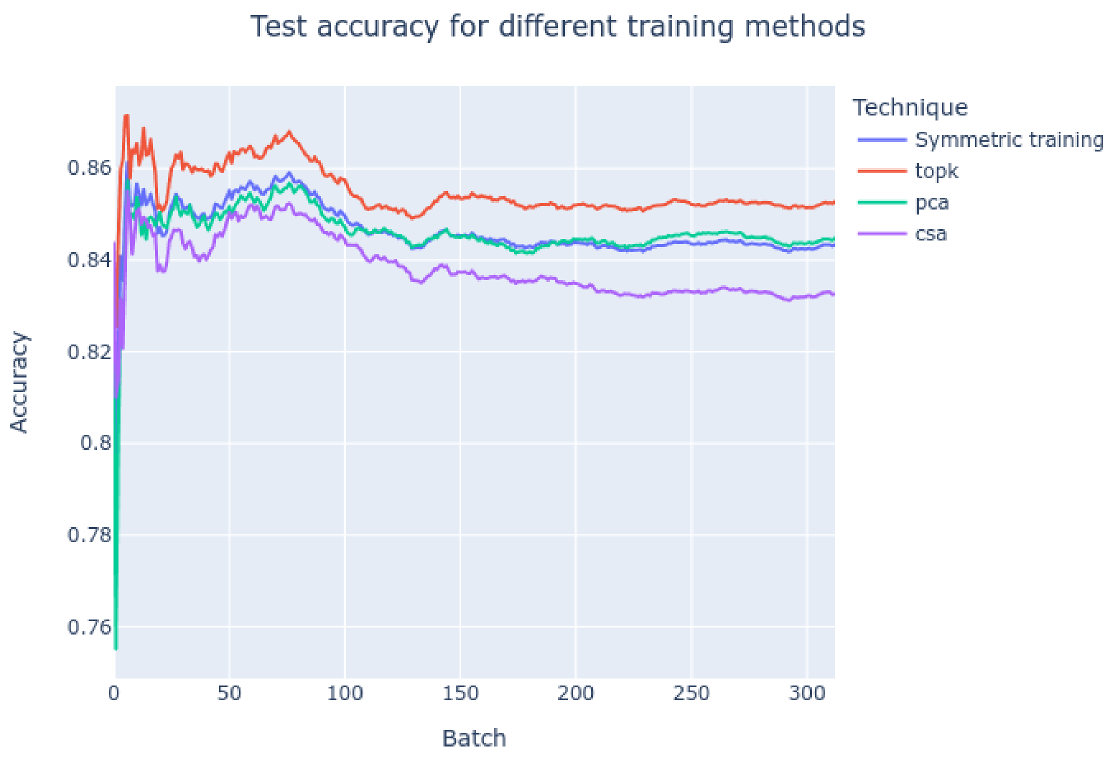

Figure 9.35: Test accuracy for different training methods for l4-128

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8681 (0.8580, 0.8781)	0.8464 (0.8458, 0.8471)
Topk Asymmetry k=40	0.8655 (0.8542, 0.8768)	0.8549 (0.8543, 0.8556)
Partial Connectivity Asymmetry k1=40, k2=200	0.8598 (0.8526, 0.8671)	0.8461 (0.8447, 0.8474)
Complete Sparsification Asymmetry k=170	0.8439 (0.8342, 0.8535)	0.8384 (0.8377, 0.8392)

Table 9.19: Accuracy table for l4-128

b) Time

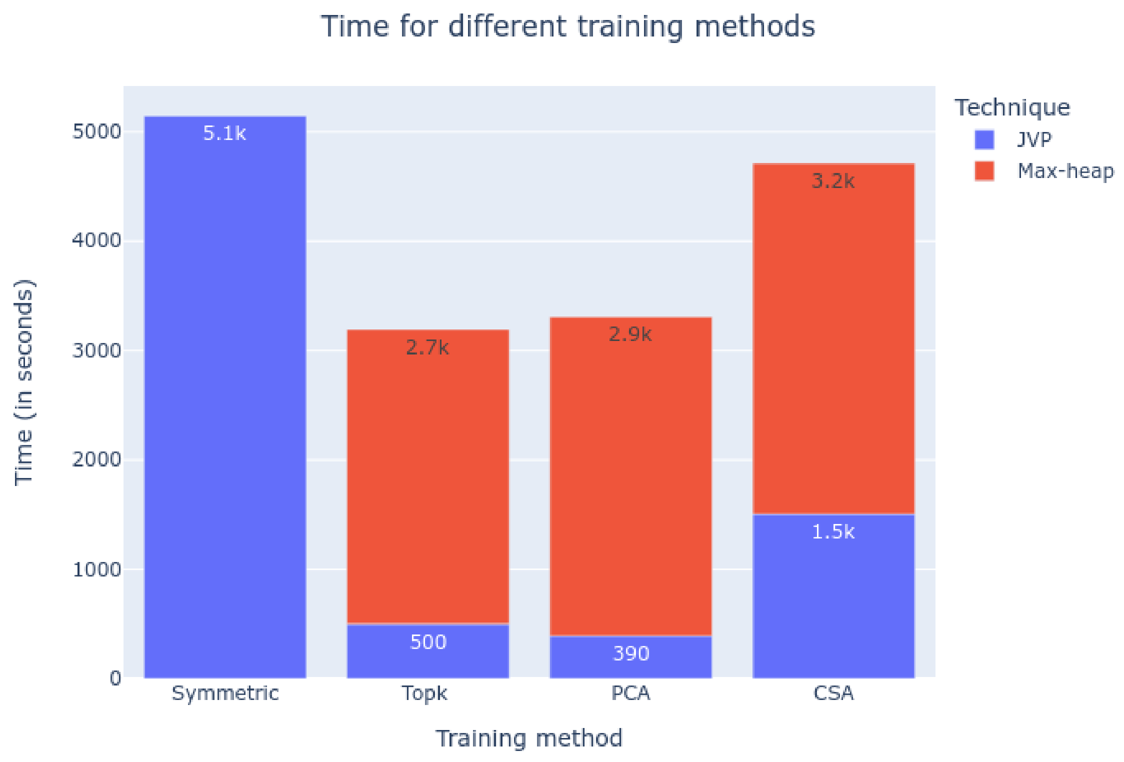

Figure 9.36: Backpropagation time for different training methods for l4-128

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	5148.06	3191.88	3308.3	4710.23

Table 9.20: Backpropagation time table for different training methods for l4-128

10. Five hidden layers with 128 neurons each.

    a) Accuracy

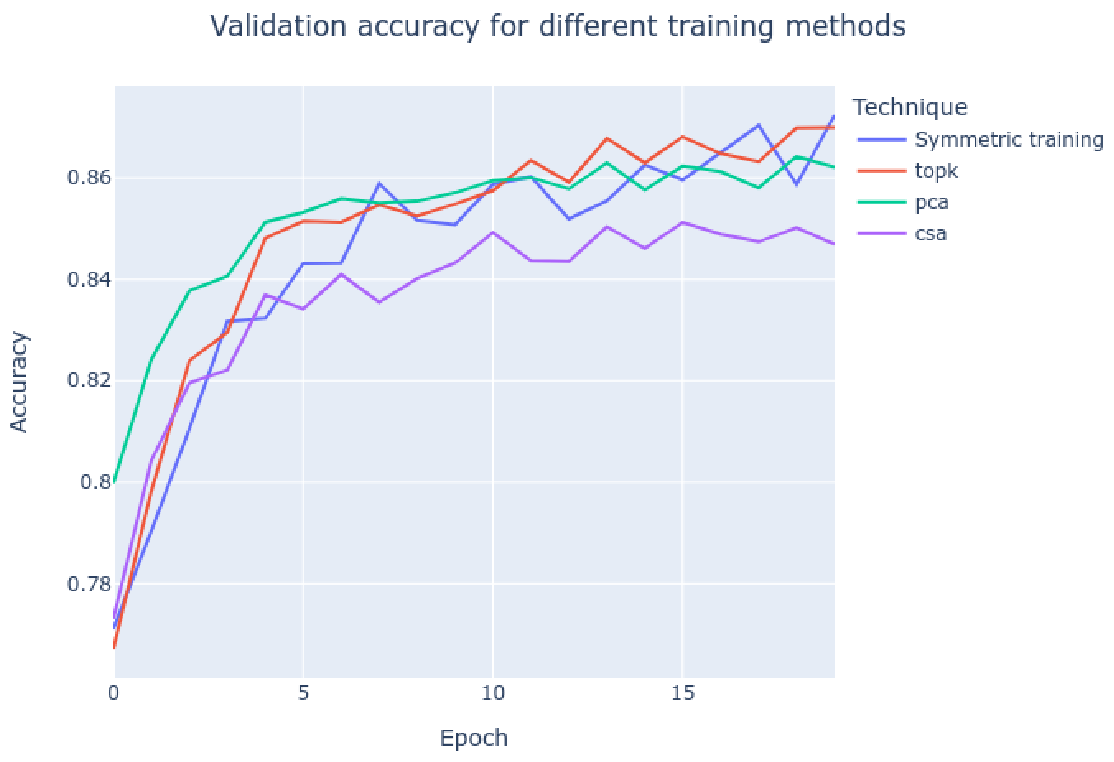

Figure 9.37: Validation accuracy for different training methods for l5-128

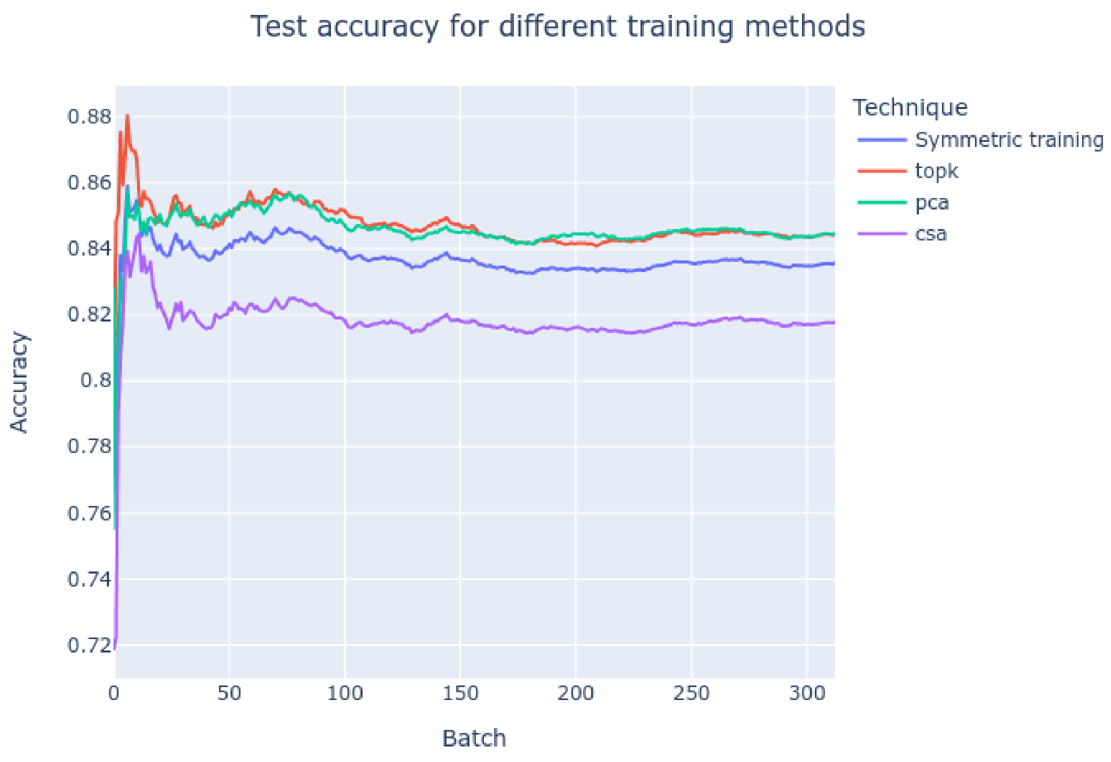

Figure 9.38: Test accuracy for different training methods for l5-128

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8649 (0.8525, 0.8773)	0.8530 (0.8525, 0.8535)
Topk Asymmetry k=40	0.8678 (0.8594, 0.8989)	0.8570 (0.8449, 0.8692)
Partial Connectivity Asymmetry k1=40, k2=200	0.8518 (0.8445, 0.8592)	0.8457 (0.8446, 0.8468)
Complete Sparsification Asymmetry k=170	0.8563 (0.8474, 0.8653)	0.8445 (0.8439, 0.8452)

Table 9.21: Accuracy table for l5-128

b) Time

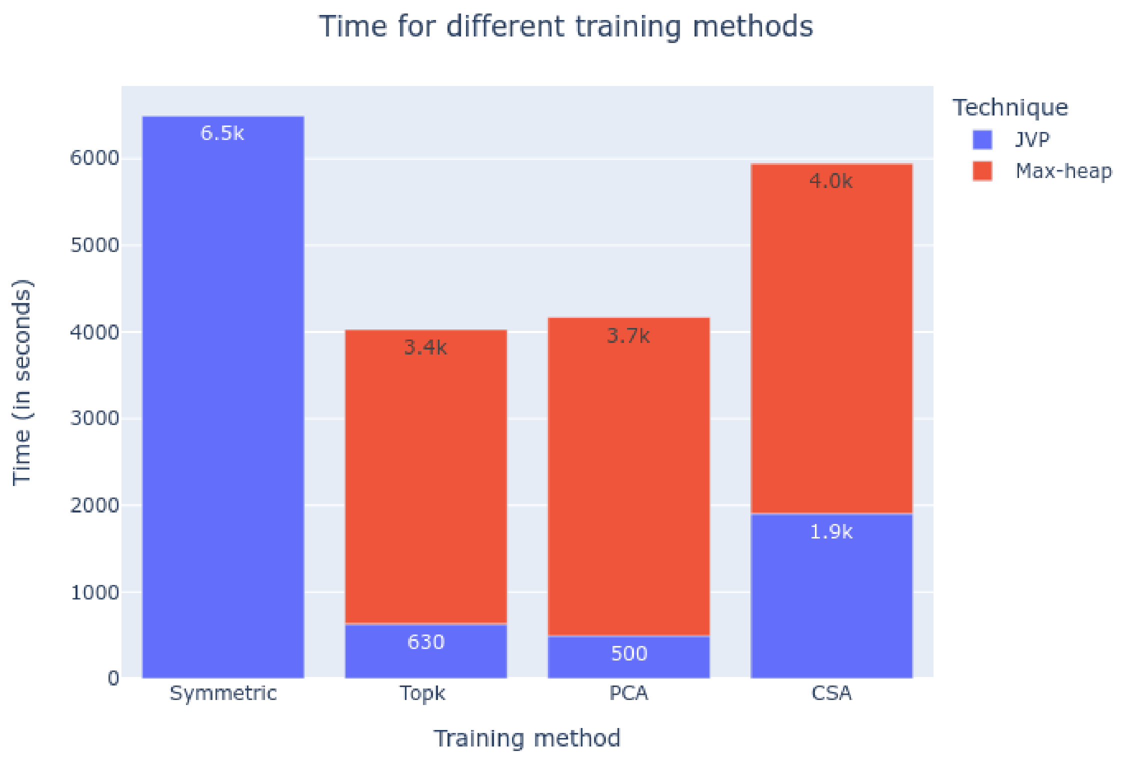

Figure 9.39: Backpropagation time for different training methods for l5-128

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	6498.47	4026.77	4170.24	5944.71

Table 9.22: Backpropagation time table for different training methods for l5-128

11. One hidden layer with 256 neurons.

    a) Accuracy

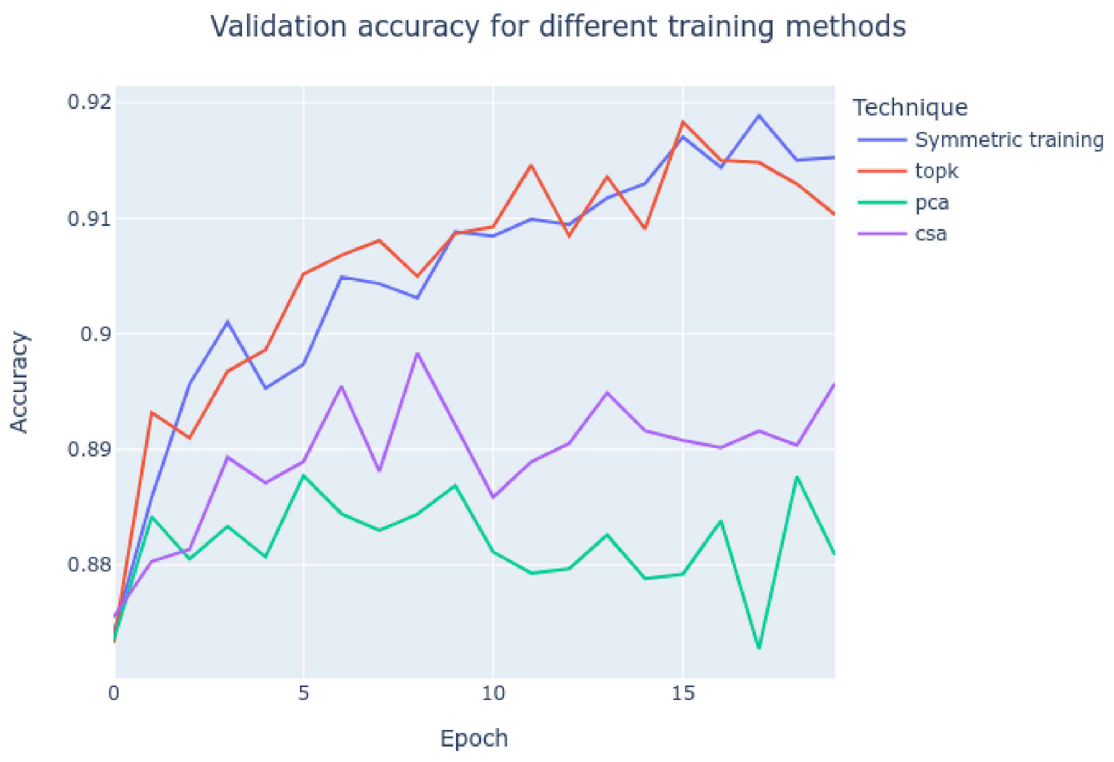

Figure 9.40: Validation accuracy for different training methods for l1-256

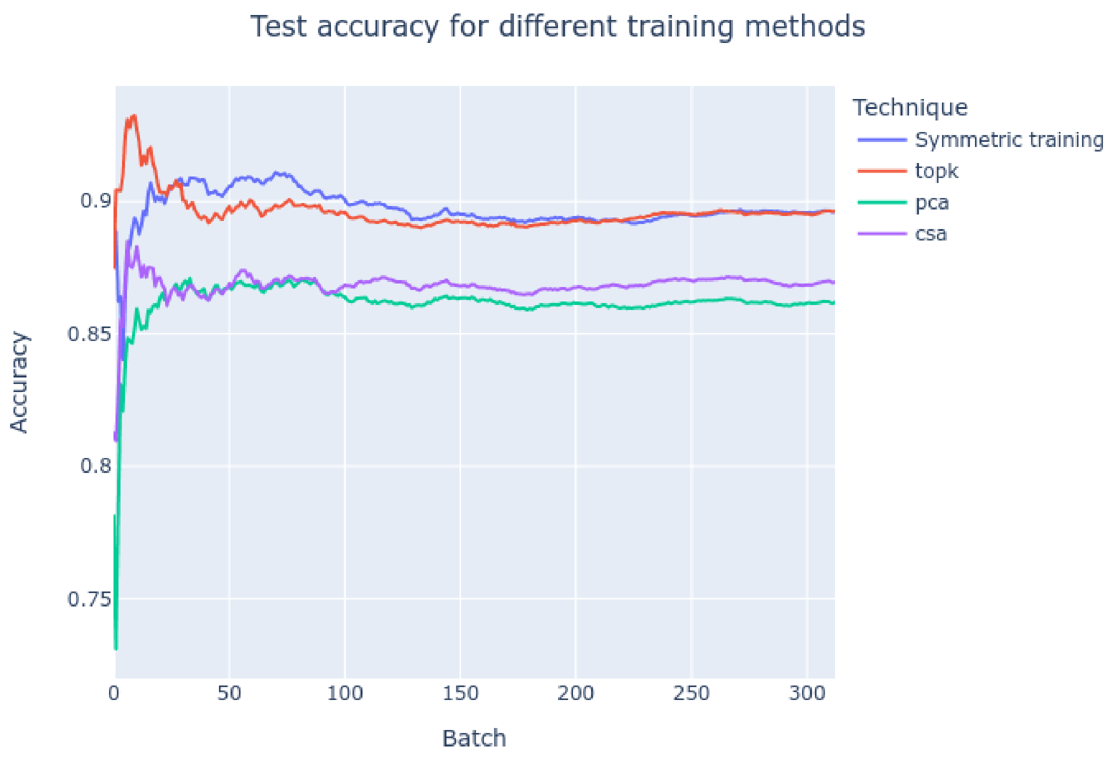

Figure 9.41: Test accuracy for different training methods for l1-256

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9051 (0.8999, 0.9104)	0.8971 (0.8963, 0.8979)
Topk Asymmetry k=40	0.9056 (0.9006, 0.9106)	0.8960 (0.8952, 0.8968)
Partial Connectivity Asymmetry k1=40, k2=200	0.8817 (0.8798, 0.8836)	0.8609 (0.8598, 0.8619)
Complete Sparsification Asymmetry k=170	0.8893 (0.8867, 0.8919)	0.8682 (0.8675, 0.8688)

Table 9.23: Accuracy table for l1-256

b) Time

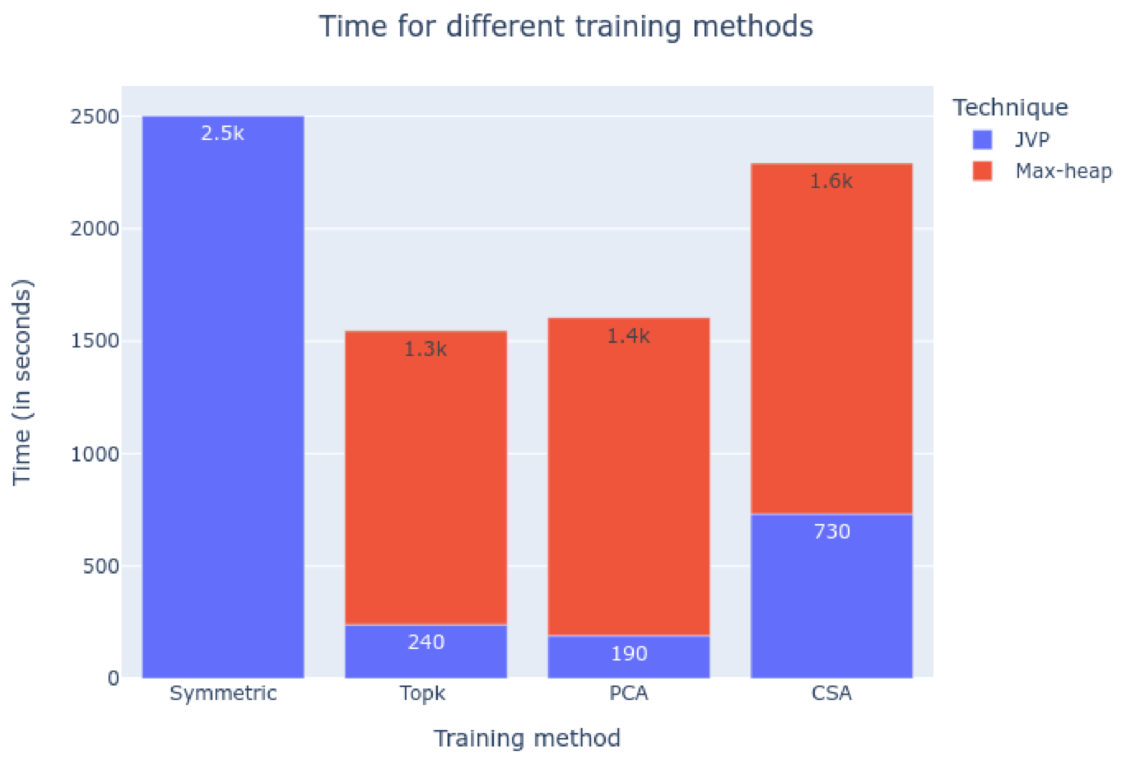

Figure 9.42: Backpropagation time for different training methods for l1-256

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	2503.22	1545.66	1604.88	2290.6

Table 9.24: Backpropagation time table for different training methods for l1-256

12. Two hidden layers with 256 neurons each.

    a) Accuracy

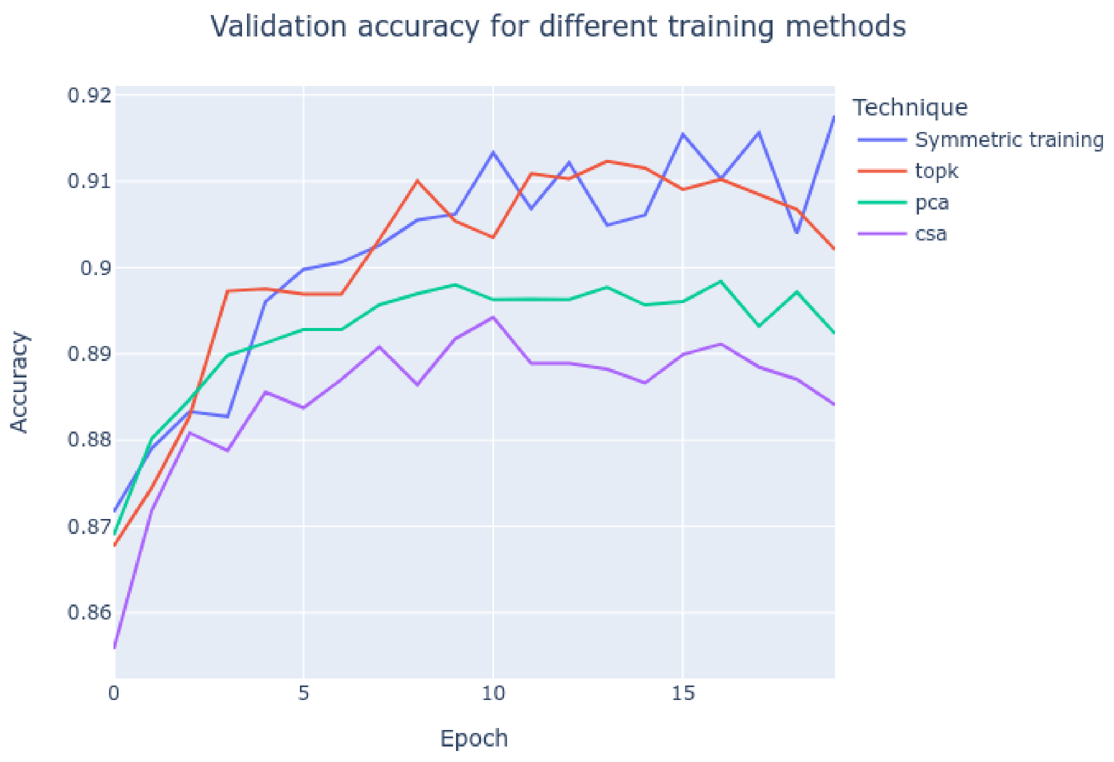

Figure 9.43: Validation accuracy for different training methods for l2-256

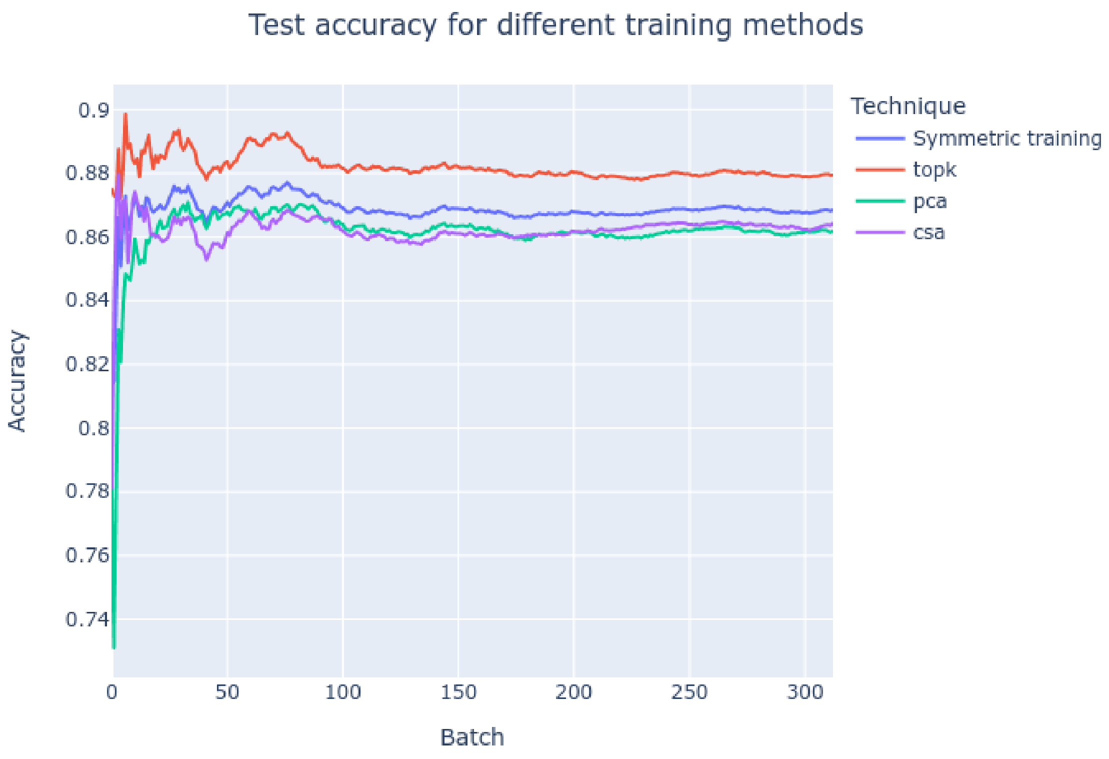

Figure 9.44: Test accuracy for different training methods for l2-256

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9016 (0.8956, 0.9077)	0.8686 (0.8680, 0.8692)
Topk Asymmetry k=40	0.9008 (0.8950, 0.9067)	0.8820 (0.8815, 0.8824)
Partial Connectivity Asymmetry k1=40, k2=200	0.8925 (0.8891, 0.8959)	0.8616 (0.8601, 0.8631)
Complete Sparsification Asymmetry k=170	0.8850 (0.8810, 0.8890)	0.8622 (0.8616, 0.8628)

Table 9.25: Accuracy table for l2-256

b) Time

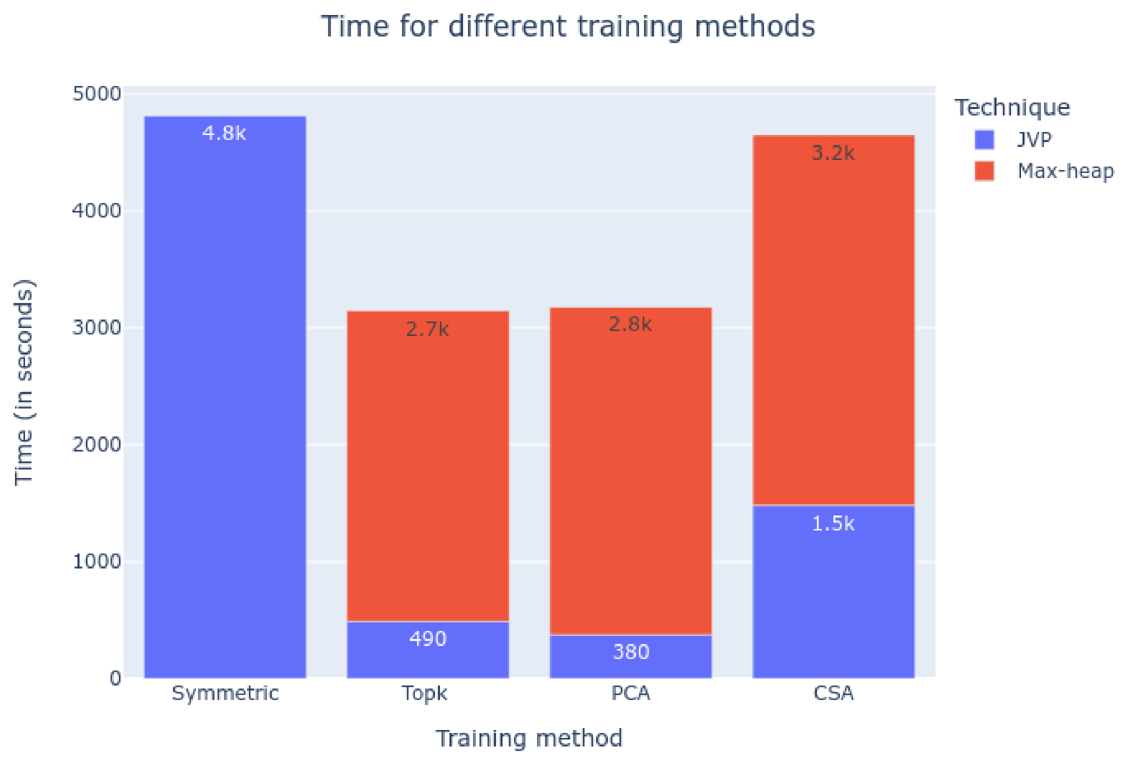

Figure 9.45: Backpropagation time for different training methods for l2-256

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	4815.55	3145.86	3175.61	4647.16

Table 9.26: Backpropagation time table for different training methods for l2-256

13. Three hidden layers with 256 neurons each.

a) Accuracy

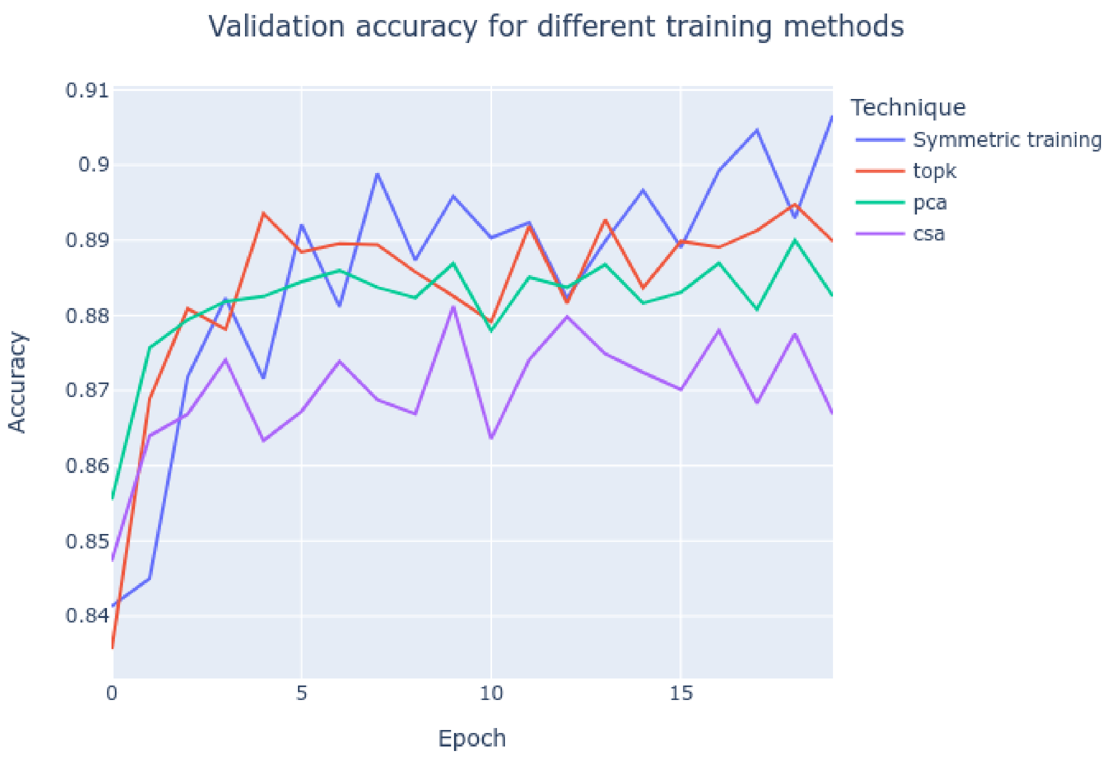

Figure 9.46: Validation accuracy for different training methods for l3-256

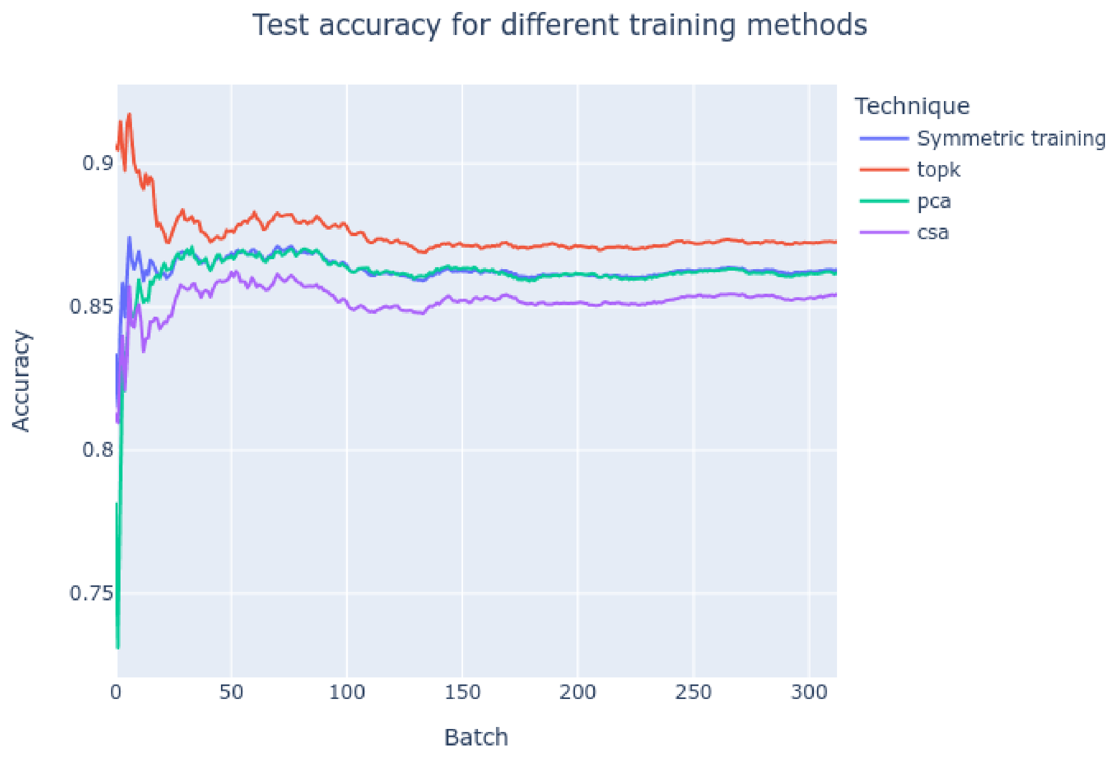

Figure 9.47: Test accuracy for different training methods for l3-256

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9040 (0.8981, 0.9099)	0.8630 (0.8625, 0.8635)
Topk Asymmetry k=40	0.8957 (0.8897, 0.9017)	0.8753 (0.8745, 0.8761)
Partial Connectivity Asymmetry k1=40, k2=200	0.8898 (0.8862, 0.8934)	0.8615 (0.8603, 0.8627)
Complete Sparsification Asymmetry k=170	0.8820 (0.8776, 0.8864)	0.8522 (0.8516, 0.8528)

Table 9.27: Accuracy table for l3-256

b) Time

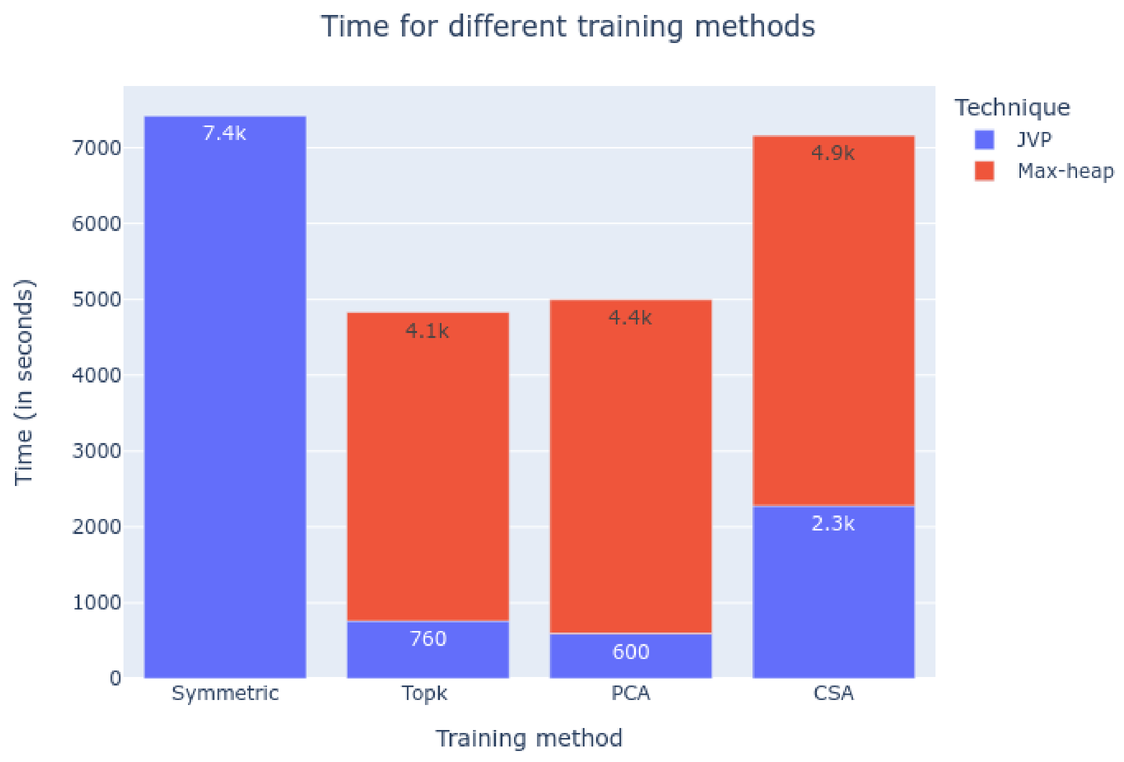

Figure 9.48: Backpropagation time for different training methods for l3-256

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	7424.35	4829.54	4995.79	7160.82

Table 9.28: Backpropagation time table for different training methods for l3-256

14. Four hidden layers with 256 neurons each.

a) Accuracy

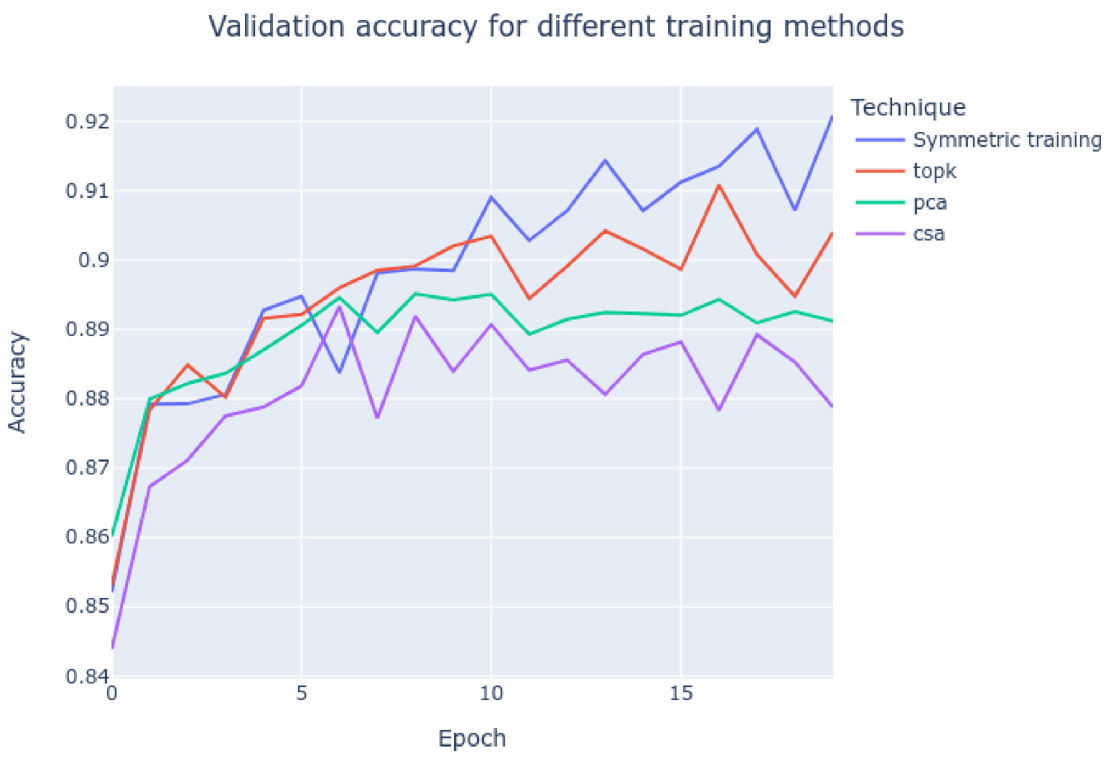

Figure 9.49: Validation accuracy for different training methods for l4-256

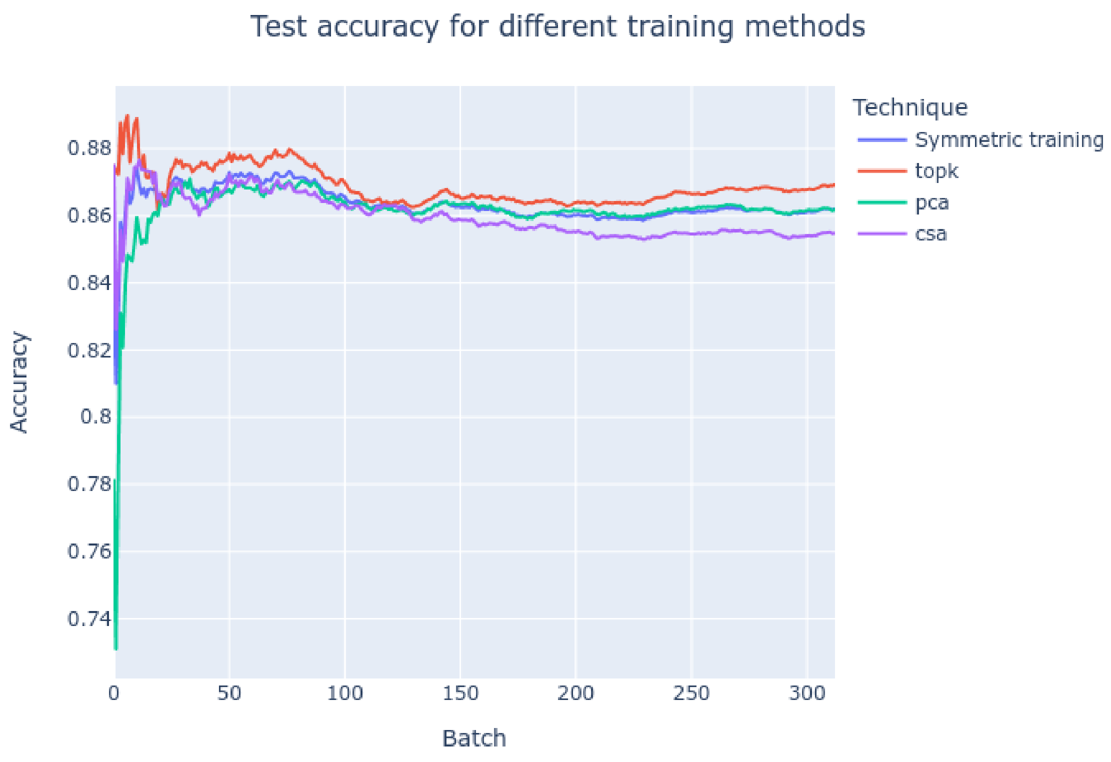

Figure 9.50: Test accuracy for different training methods for l4-256

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8985 (0.8906, 0.9064)	0.8633 (0.8627, 0.8639)
Topk Asymmetry k=40	0.8944 (0.8884, 0.9003)	0.8687 (0.8681, 0.8693)
Partial Connectivity Asymmetry k1=40, k2=200	0.8889 (0.8852, 0.8927)	0.8615 (0.8603, 0.8627)
Complete Sparsification Asymmetry k=170	0.8807 (0.8755, 0.8858)	0.8597 (0.8590, 0.8604)

Table 9.29: Accuracy table for l4-256

b) Time

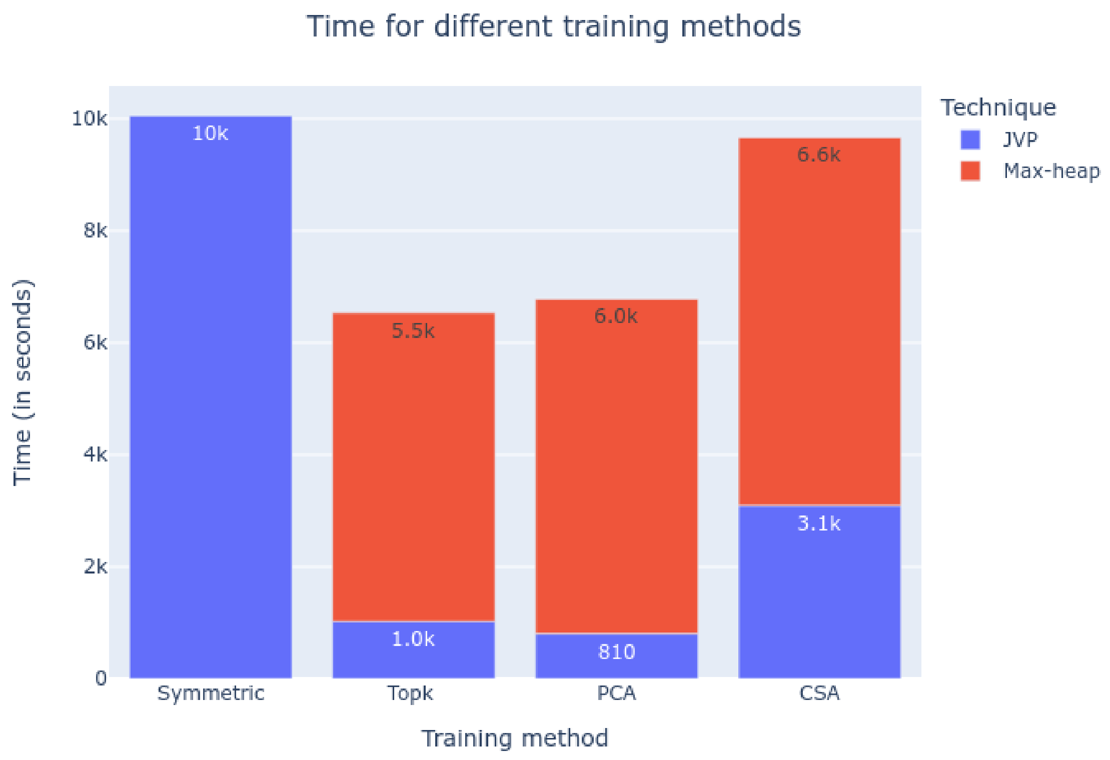

Figure 9.51: Backpropagation time for different training methods for l4-256

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	10057.21	6532.4	6780.03	9661.53

Table 9.30: Backpropagation time table for different training methods for l4-256

15. Five hidden layers with 256 neurons each.

    a) Accuracy

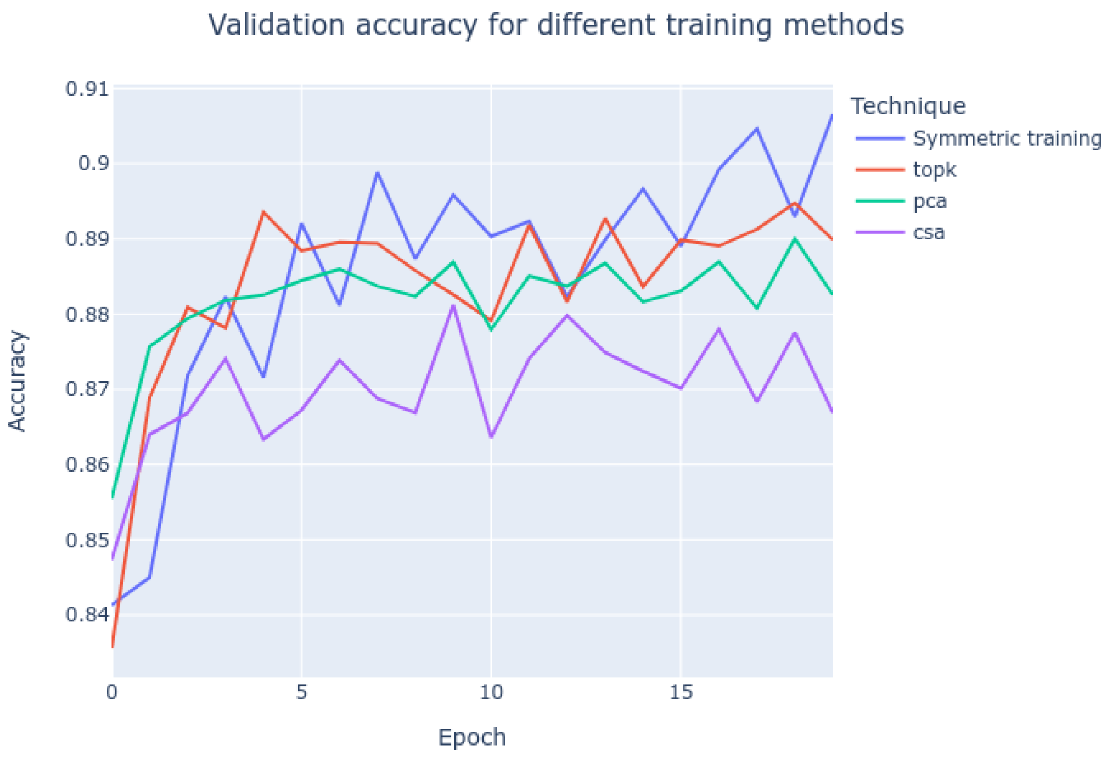

Figure 9.52: Validation accuracy for different training methods for l5-256

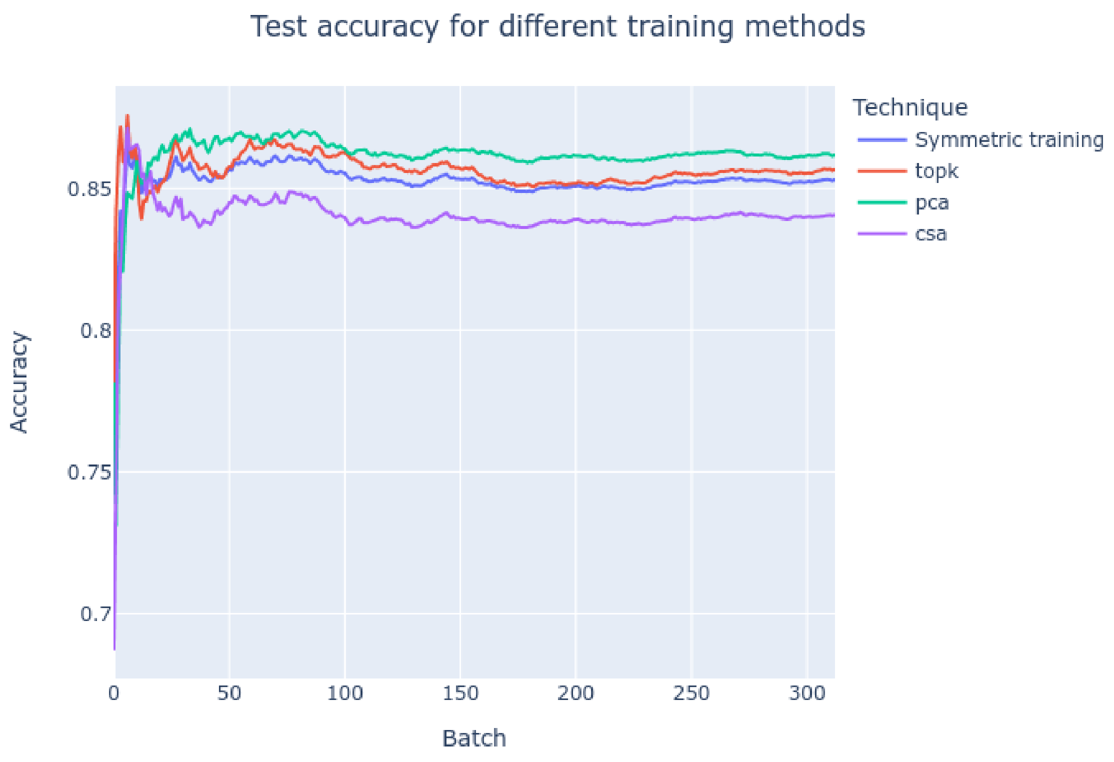

Figure 9.53: Test accuracy for different training methods for l5-256

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8855 (0.8775, 0.8936)	0.8526 (0.8517, 0.8535)
Topk Asymmetry k=40	0.8838 (0.8777, 0.8899)	0.8563 (0.8556, 0.8570)
Partial Connectivity Asymmetry k1=40, k2=200	0.8818 (0.8785, 0.8851)	0.8615 (0.8603, 0.8627)
Complete Sparsification Asymmetry k=170	0.8699 (0.8664, 0.8735)	0.8400 (0.8387, 0.8413)

Table 9.31: Accuracy table for l5-256

b) Time

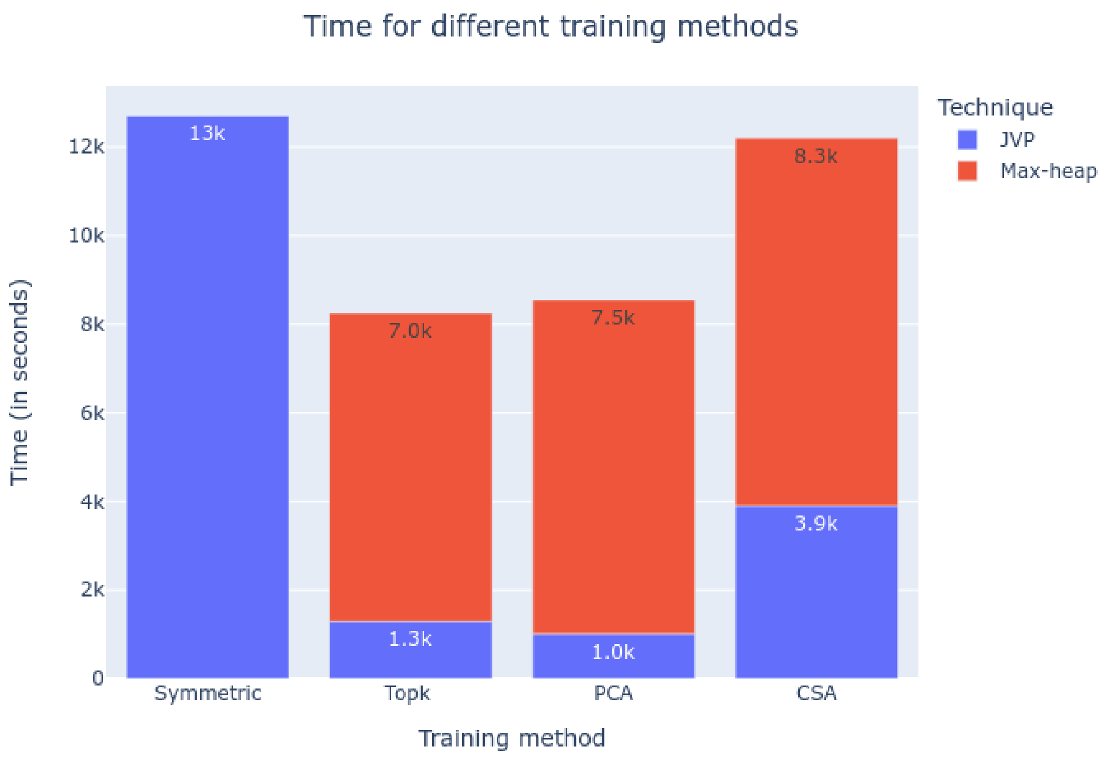

Figure 9.54: Backpropagation time for different training methods for l5-256

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	12710.38	8248.65	8547.62	12196.92

Table 9.32: Backpropagation time table for different training methods for l5-256

16. One hidden layer with 512 neurons.

a) Accuracy

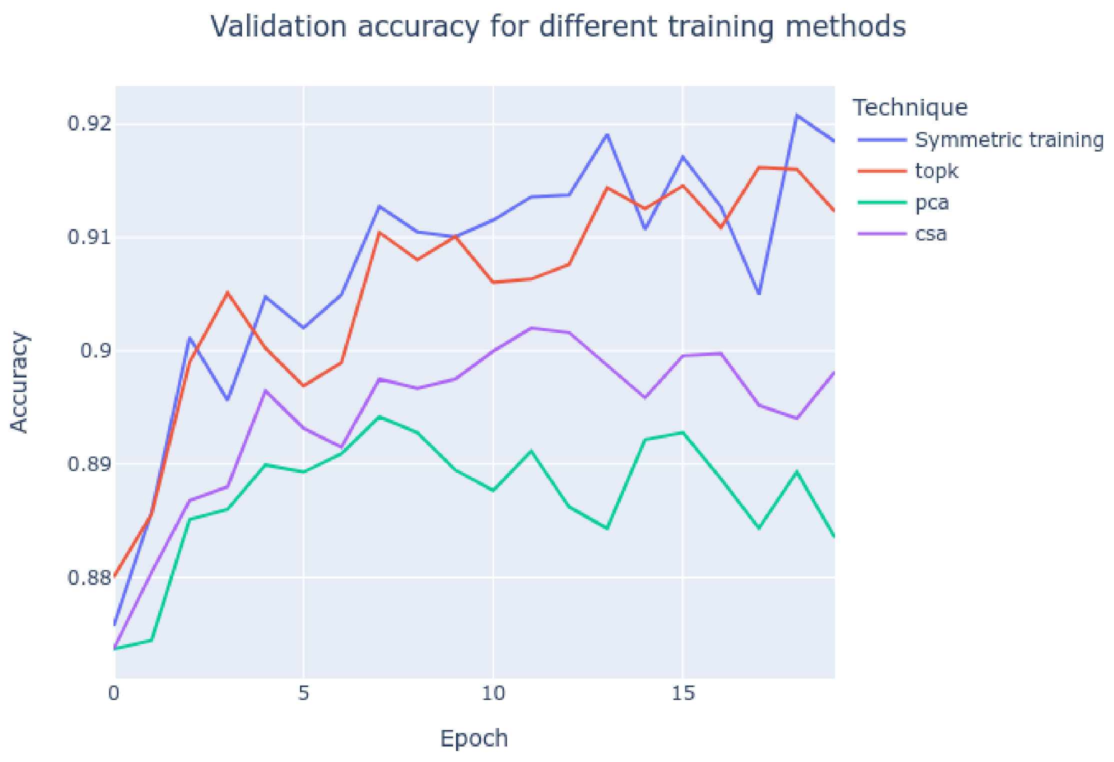

Figure 9.55: Validation accuracy for different training methods for l1-512

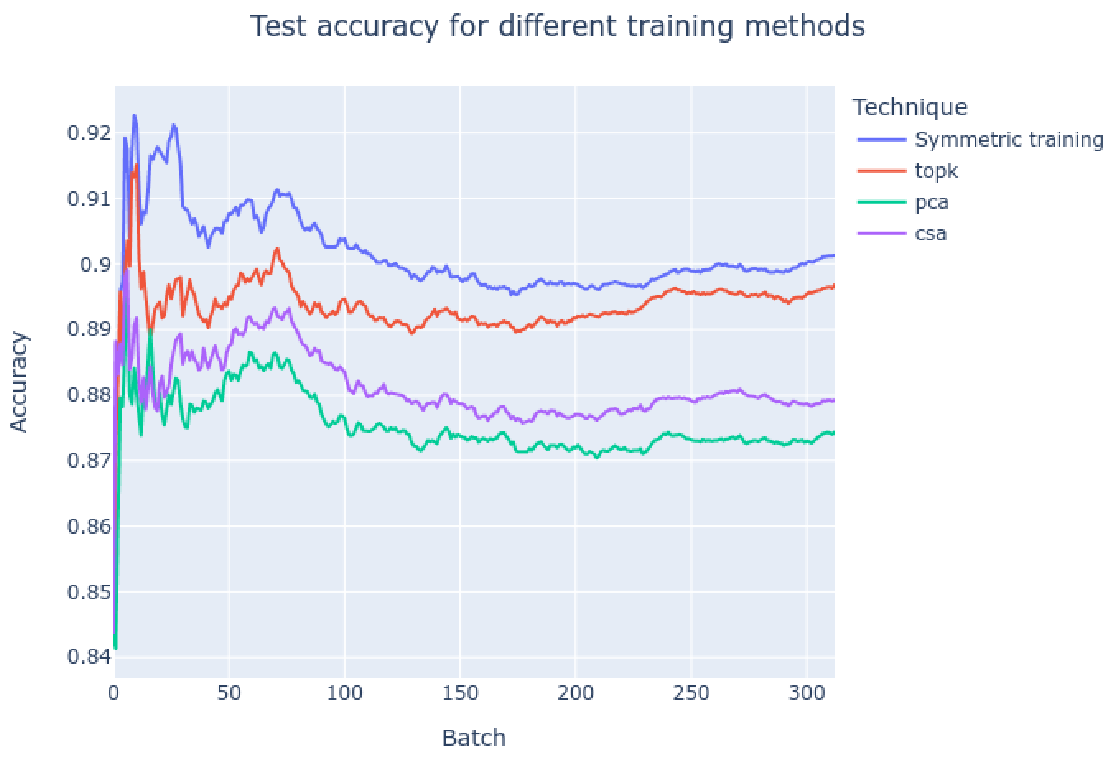

Figure 9.56: Test accuracy for different training methods for l1-512

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9072 (0.9020, 0.9125)	0.9013 (0.9006, 0.9021)
Topk Asymmetry k=40	0.9055 (0.9010, 0.9101)	0.8937 (0.8932, 0.8942)
Partial Connectivity Asymmetry k1=40, k2=200	0.8873 (0.8847, 0.8898)	0.8754 (0.8749, 0.8758)
Complete Sparsification Asymmetry k=170	0.8943 (0.8909, 0.8977)	0.8811 (0.8806, 0.8817)

Table 9.33: Accuracy table for l1-512

b) Time

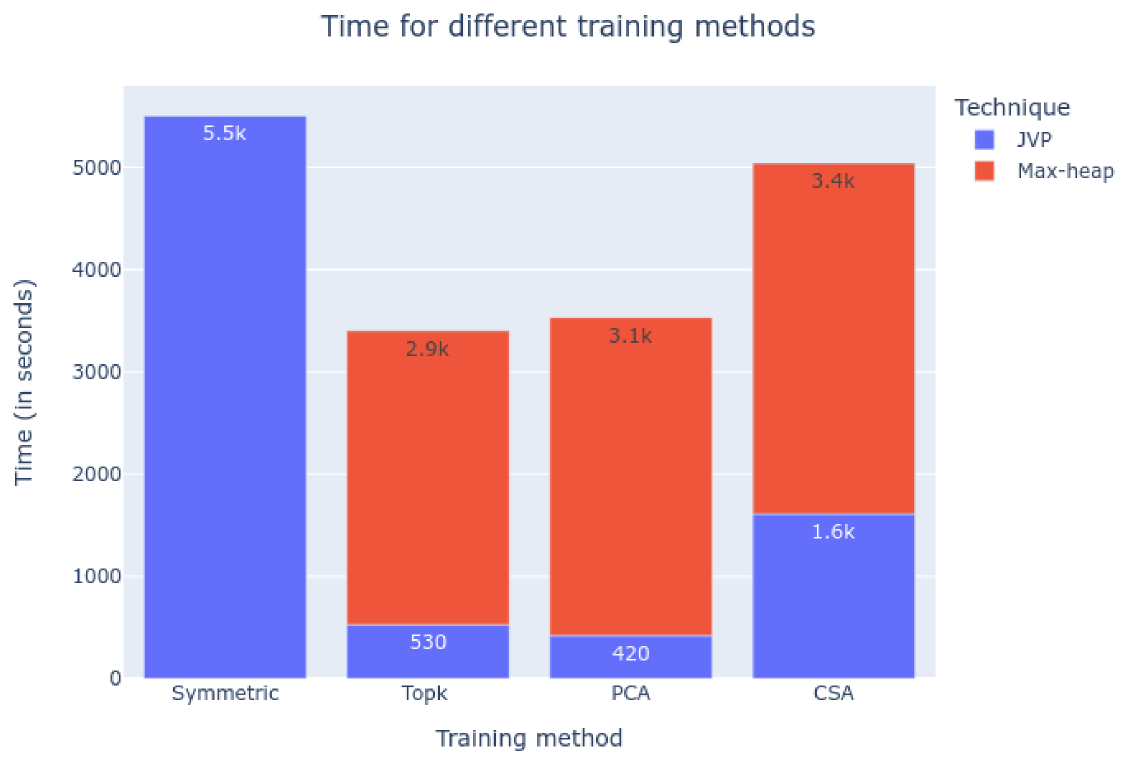

Figure 9.57: Backpropagation time for different training methods for l1-512

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	5507.09	3400.46	3530.74	5039.31

Table 9.34: Backpropagation time table for different training methods for l1-512

17. Two hidden layers with 512 neurons each.

   a) Accuracy

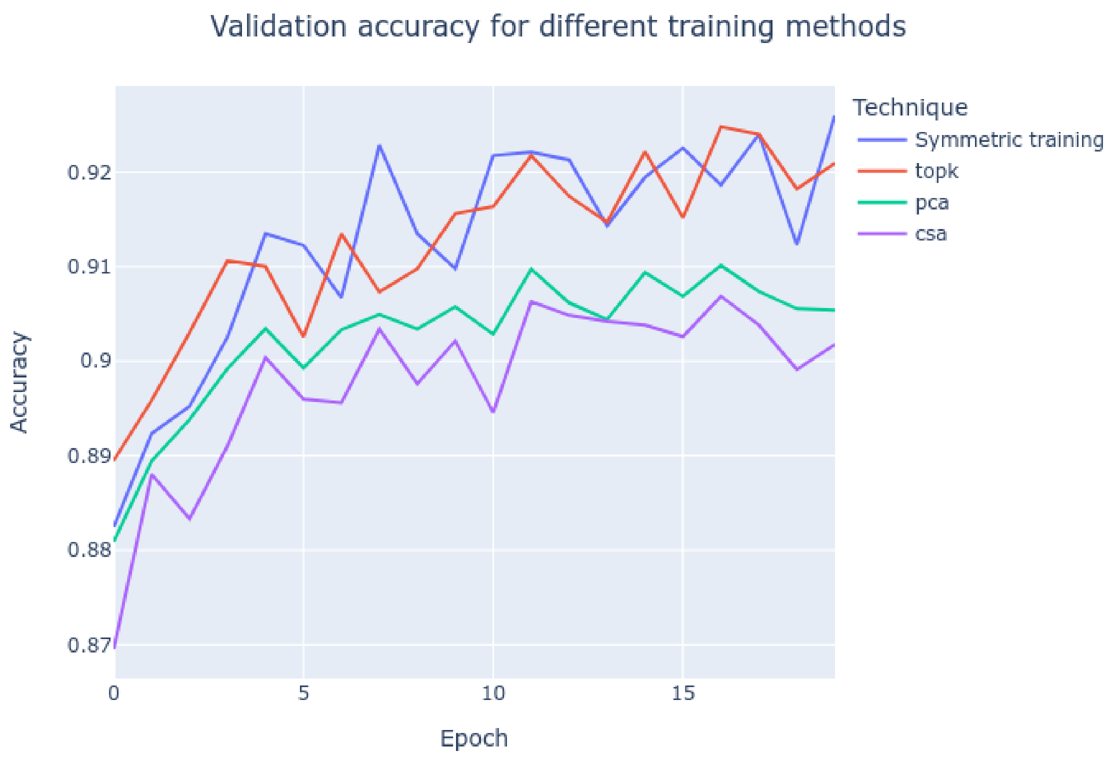

Figure 9.58: Validation accuracy for different training methods for l2-512

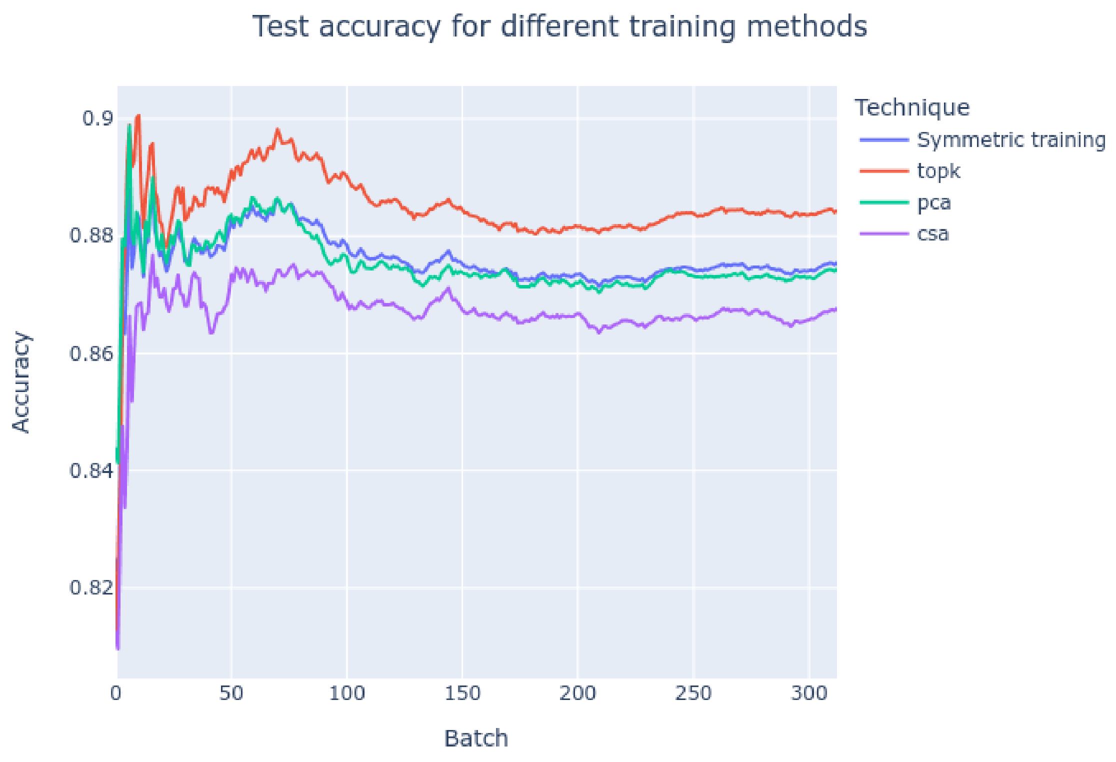

Figure 9.59: Test accuracy for different training methods for l2-512

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9126 (0.9072, 0.9181)	0.8757 (0.8751, 0.8764)
Topk Asymmetry k=40	0.9126 (0.9082, 0.9170)	0.8849 (0.8841, 0.8857)
Partial Connectivity Asymmetry k1=40, k2=200	0.9025 (0.8992, 0.9059)	0.8761 (0.8740 0.8781)
Complete Sparsification Asymmetry k=170	0.8977 (0.8934, 0.9020)	0.8671 (0.8664, 0.8678)

Table 9.35: Accuracy table for l2-512

b) Time

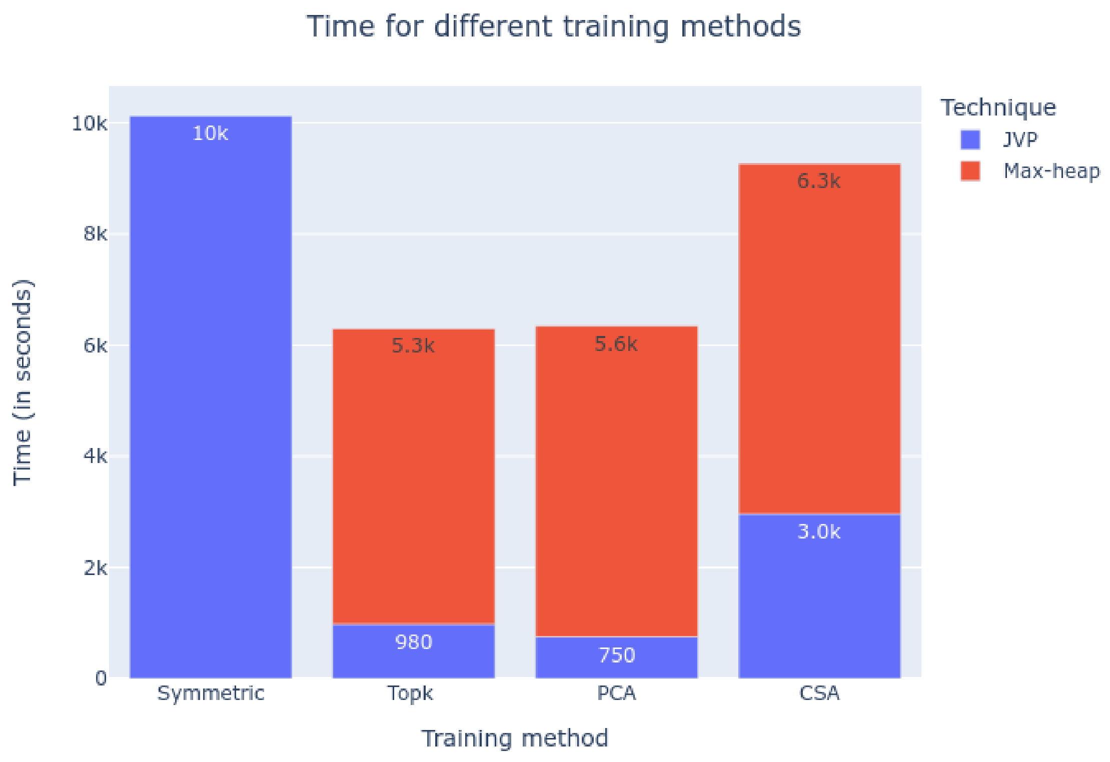

Figure 9.60: Backpropagation time for different training methods for l2-512

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	10132.71	6296.81	6351.22	9267.38

Table 9.36: Backpropagation time table for different training methods for l2-512

18. Three hidden layers with 512 neurons each.

    a) Accuracy

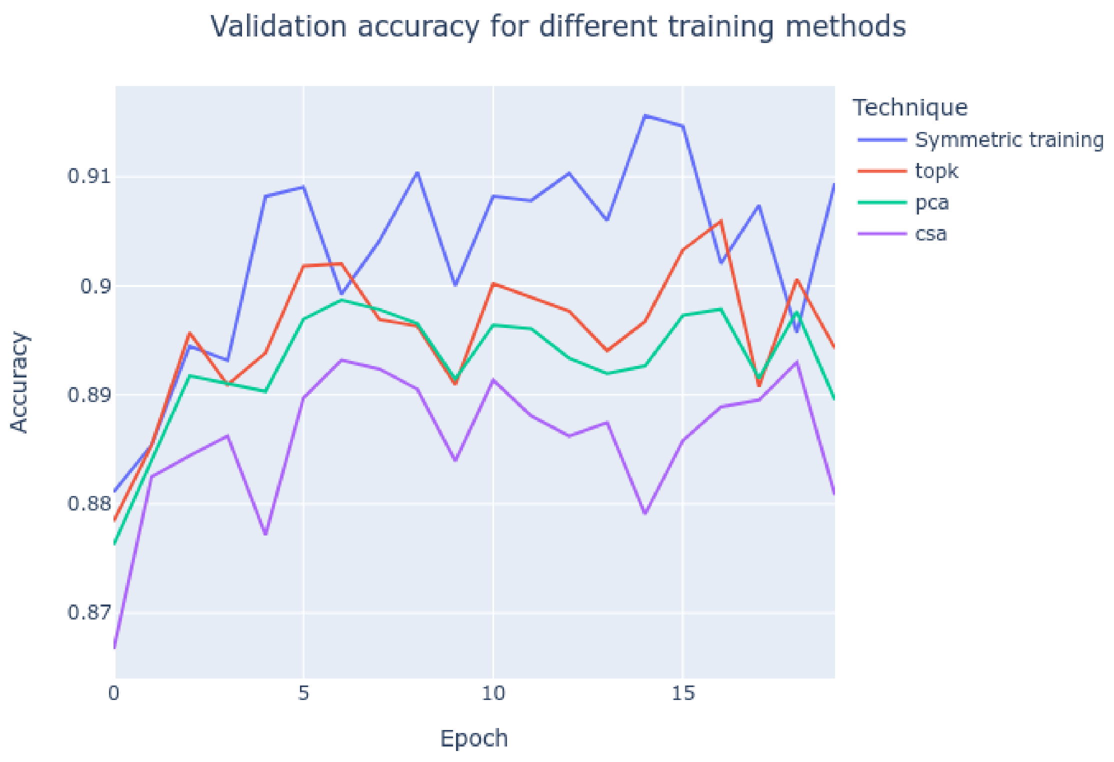

Figure 9.61: Validation accuracy for different training methods for l3-512

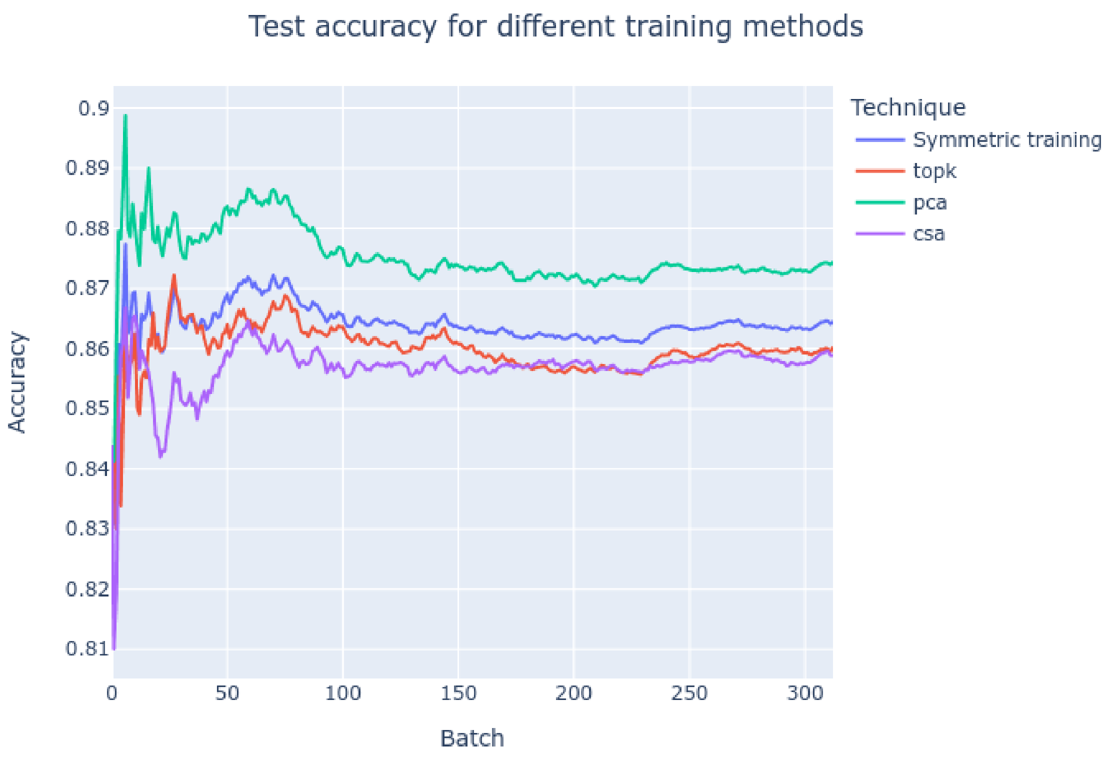

Figure 9.62: Test accuracy for different training methods for l3-512

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9031 (0.8987, 0.9074)	0.8640 (0.8636, 0.8644)
Topk Asymmetry k=40	0.8957 (0.8927, 0.8987)	0.8600(0.8595, 0.8604)
Partial Connectivity Asymmetry k1=40, k2=200	0.8929 (0.8904, 0.8954)	0.8762 (0.8747, 0.8778)
Complete Sparsification Asymmetry k=170	0.8858 (0.8828, 0.8888)	0.8569 (0.8563, 0.8574)

Table 9.37: Accuracy table for l3-512

b) Time

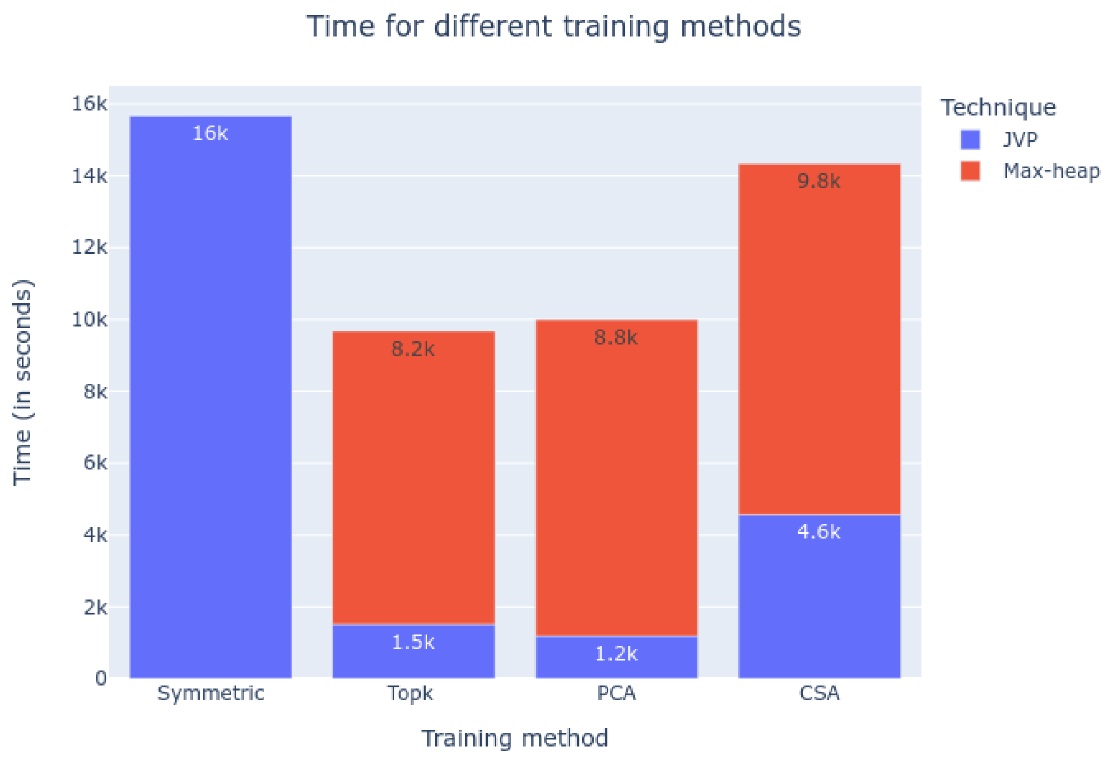

Figure 9.63: Backpropagation time for different training methods for l3-512

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	15679.24	9676.57	9989.58	14333.61

Table 9.38: Backpropagation time table for different training methods for l3-512

19. Four hidden layers with 512 neurons each.

    a) Accuracy

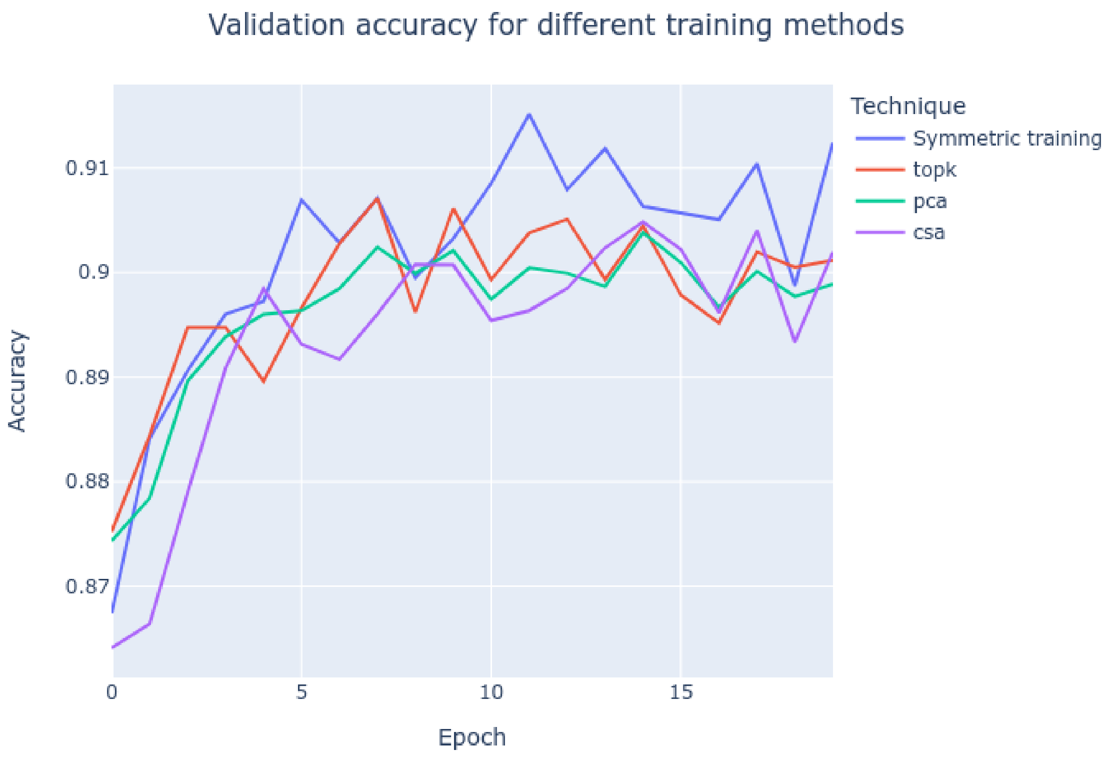

Figure 9.64: Validation accuracy for different training methods for l4-512

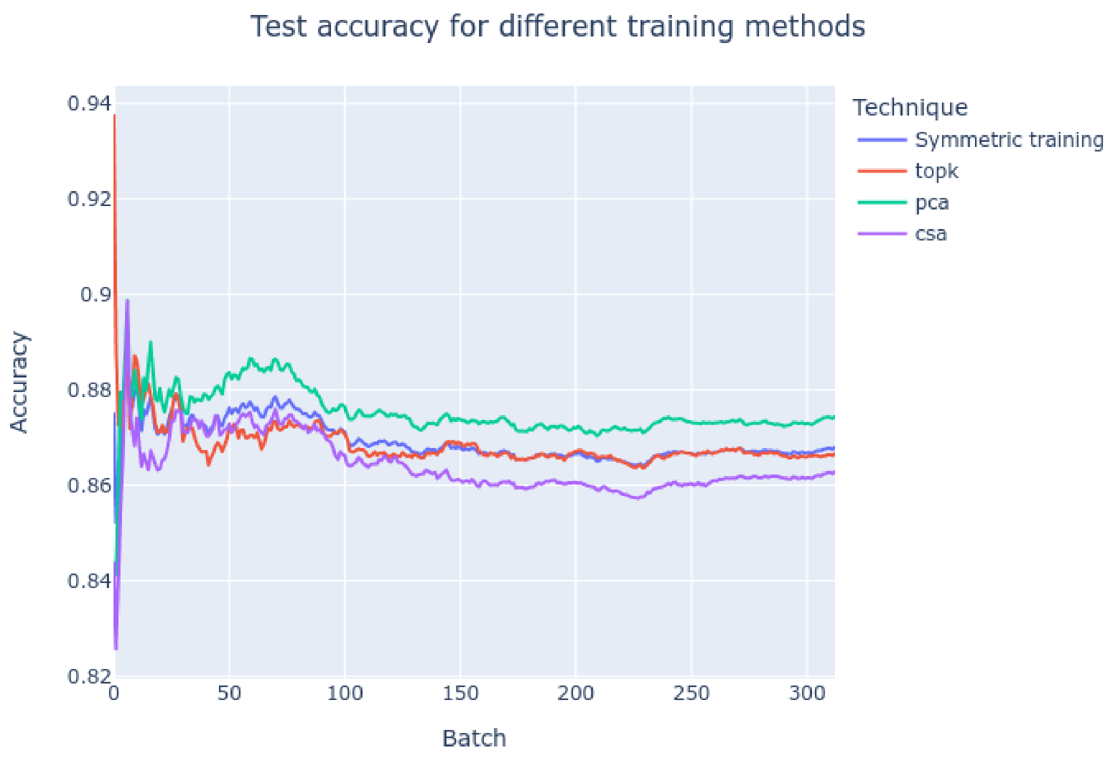

Figure 9.65: Test accuracy for different training methods for l4-512

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9018 (0.8966, 0.9070)	0.8693 (0.8688, 0.8698)
Topk Asymmetry k=40	0.8978 (0.8941, 0.9014)	0.8686 (0.8680, 0.8692)
Partial Connectivity Asymmetry k1=40, k2=200	0.8963 (0.8927, 0.8998)	0.8747 (0.8737, 0.8758)
Complete Sparsification Asymmetry k=170	0.8938 (0.8884, 0.8991)	0.8641(0.8634, 0.8648)

Table 9.39: Accuracy table for l4-512

b) Time

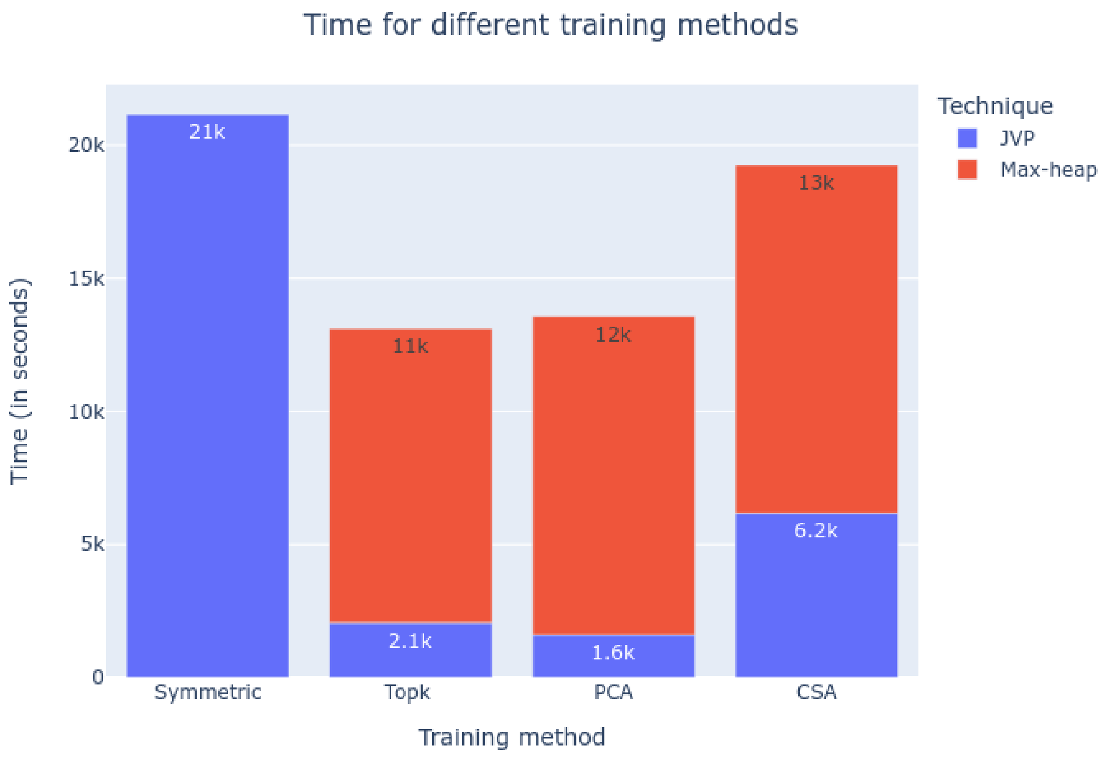

Figure 9.66: Backpropagation time for different training methods for l4-512

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	21171.1	13109.52	13573.99	19253.18

Table 9.40: Backpropagation time table for different training methods for l4-512

20. Five hidden layers with 512 neurons each.

a) Accuracy

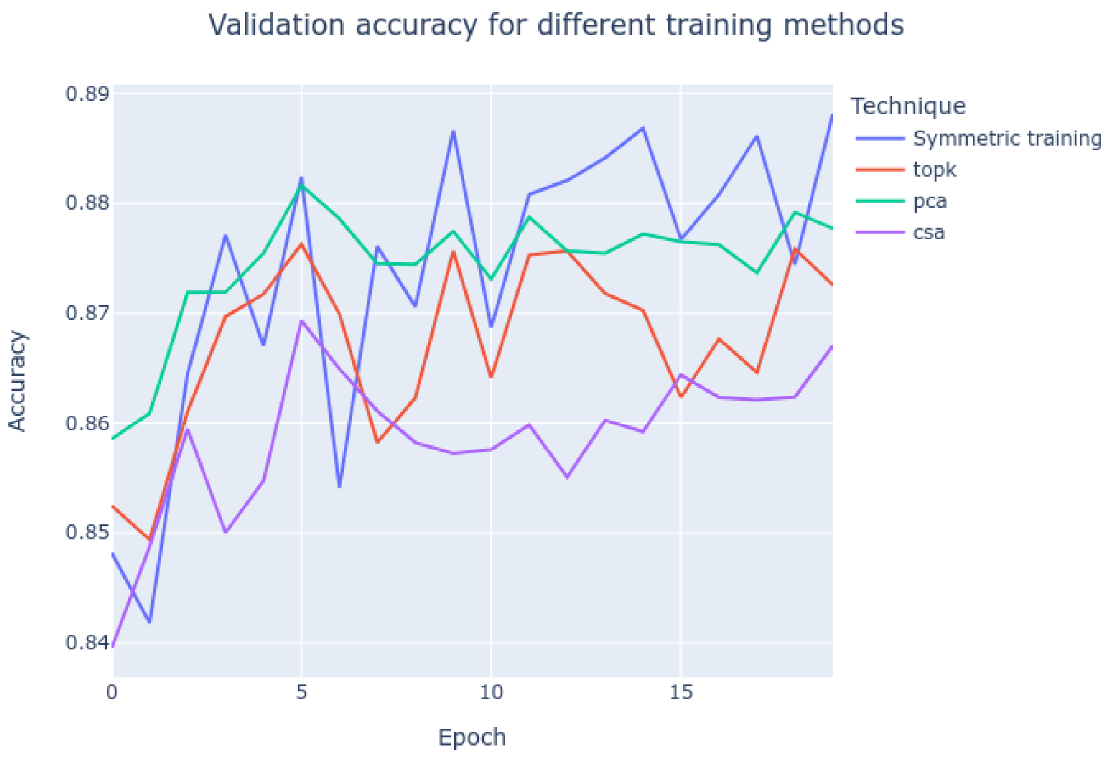

Figure 9.67: Validation accuracy for different training methods for l5-512

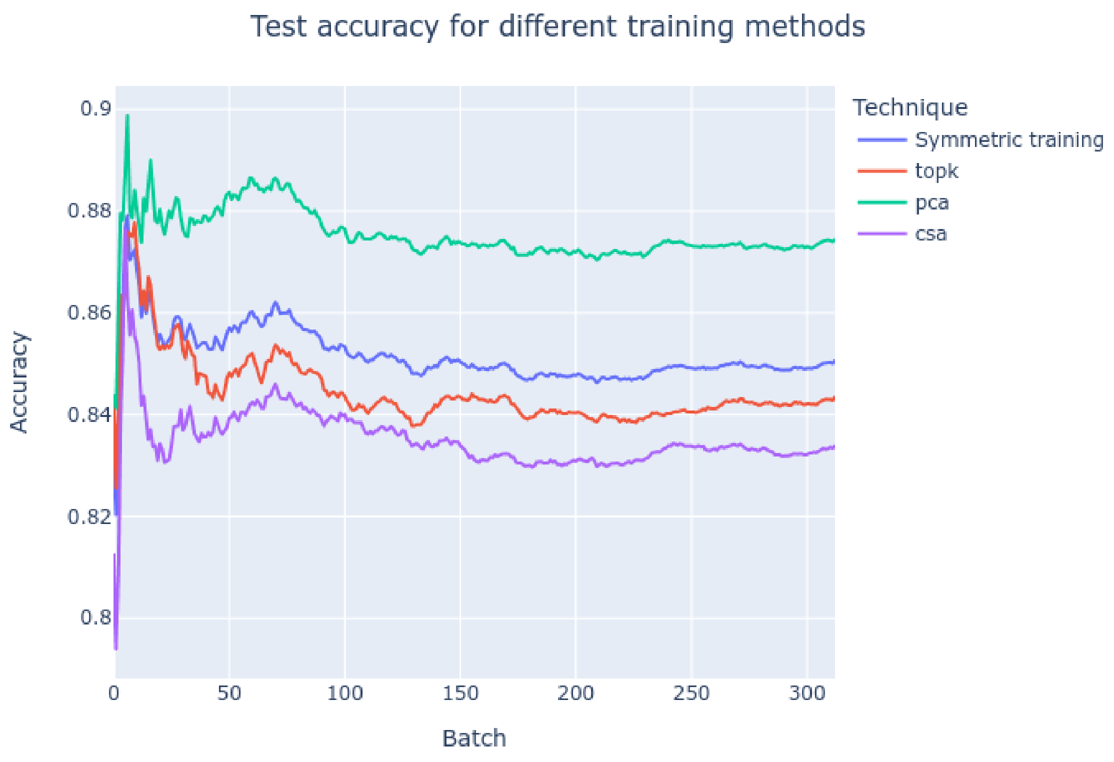

Figure 9.68: Test accuracy for different training methods for l5-512

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8738 (0.8676, 0.8800)	0.8516 (0.8510, 0.8522)
Topk Asymmetry k=40	0.8673 (0.8636, 0.8710)	0.8445 (0.8438, 0.8453)
Partial Connectivity Asymmetry k1=40, k2=200	0.8744 (0.8718, 0.8770)	0.8752 (0.8747, 0.8758)
Complete Sparsification Asymmetry k=170	0.8586 (0.8554, 0.8618)	0.8351 (0.8344, 0.8358)

Table 9.41: Accuracy table for l5-512

b) Time

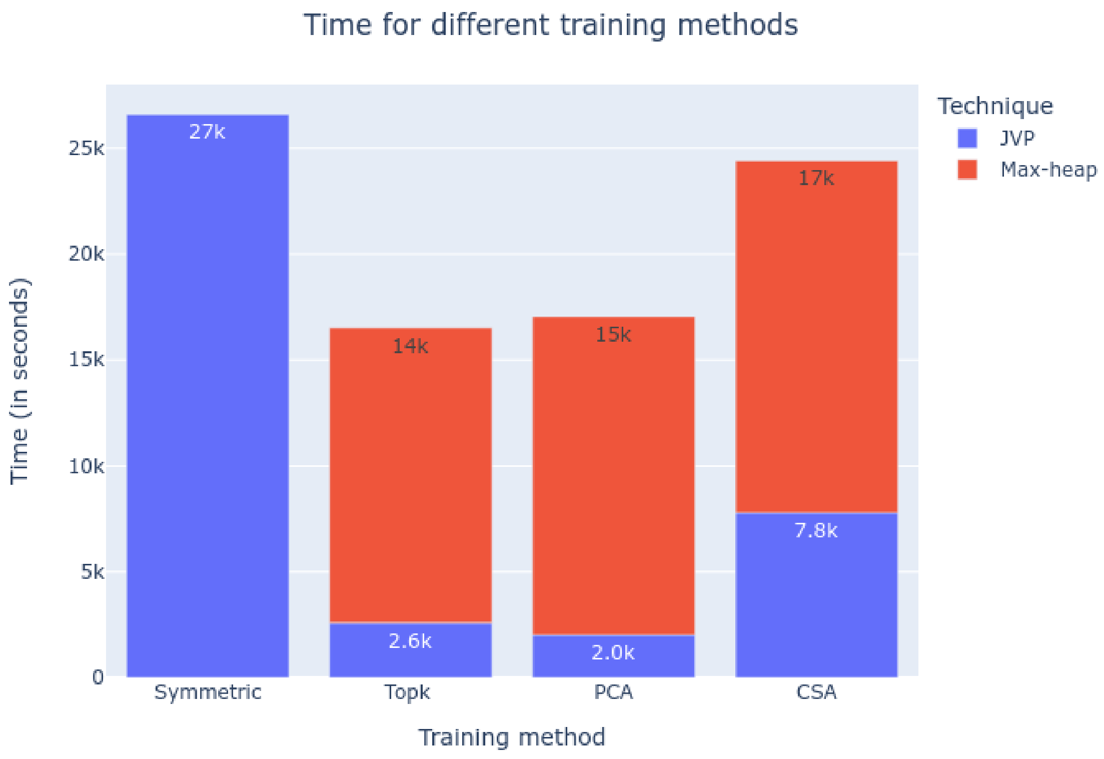

Figure 9.69: Backpropagation time for different training methods for l5-512

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	26615.25	16516.4	17047.5	24402.47

Table 9.42: Backpropagation time table for different training methods for l5-512

21. One hidden layer with 1024 neurons.

    a) Accuracy

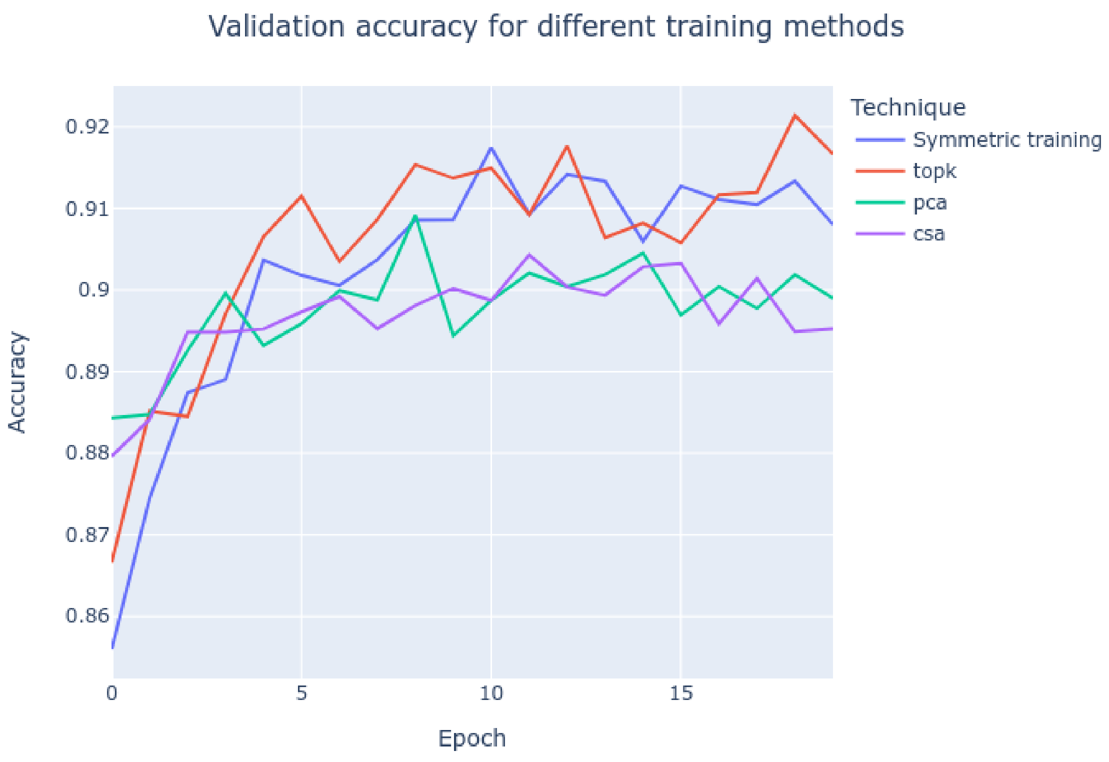

Figure 9.70: Validation accuracy for different training methods for l1-1024

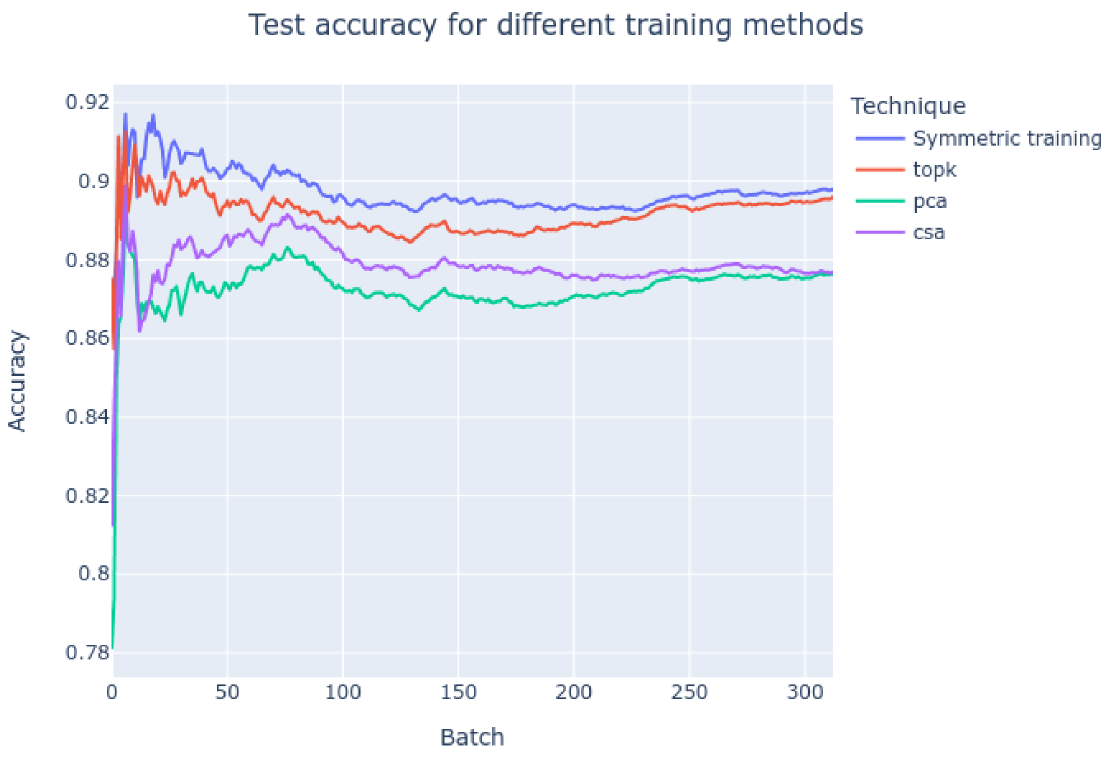

Figure 9.71: Test accuracy for different training methods for l1-1024

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9024 (0.8953, 0.9095)	0.8973 (0.8967, 0.8979)
Topk Asymmetry k=40	0.9058 (0.8996, 0.9120)	0.8916 (0.8911, 0.8922)
Partial Connectivity Asymmetry k1=40, k2=200	0.8978 (0.8950, 0.9005)	0.8724 (0.8715, 0.8733)
Complete Sparsification Asymmetry k=170	0.8967 (0.8939, 0.8995)	0.8788 (0.8780, 0.8795)

Table 9.43: Accuracy table for l1-1024

b) Time

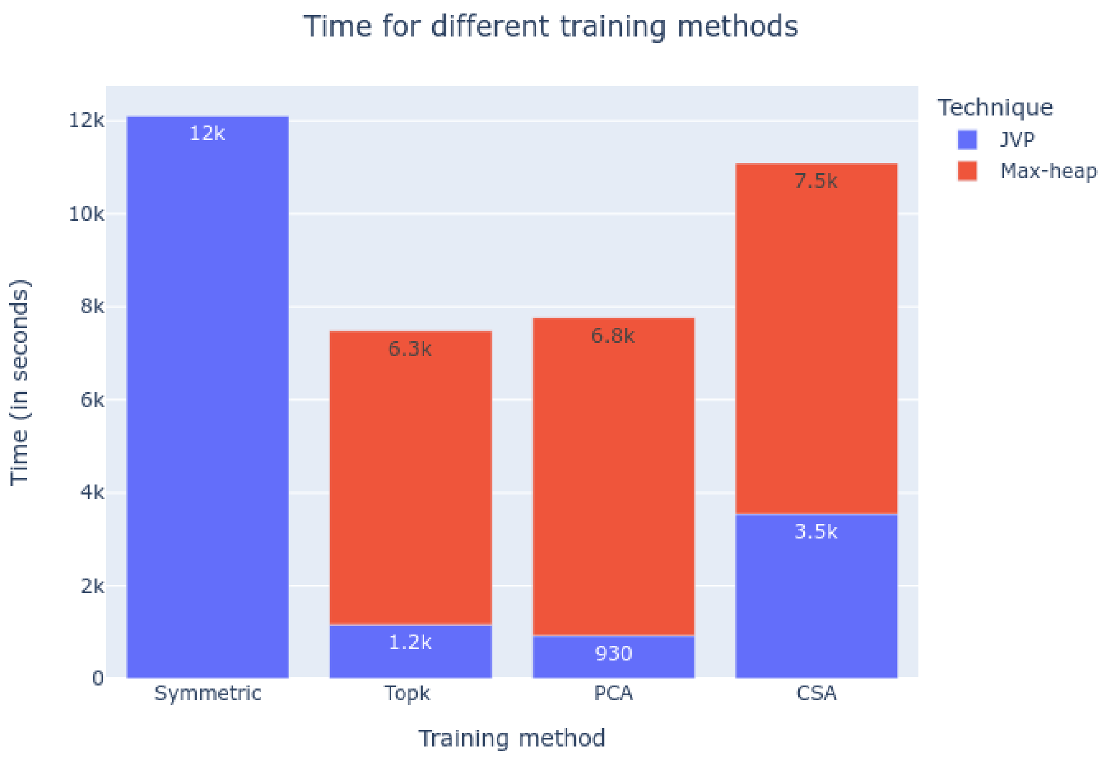

Figure 9.72: Backpropagation time for different training methods for l1-1024

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	12115.59	7481.01	7767.63	11086.49

Table 9.44: Backpropagation time table for different training methods for l1-1024

22. Two hidden layers with 1024 neurons each.

a) Accuracy

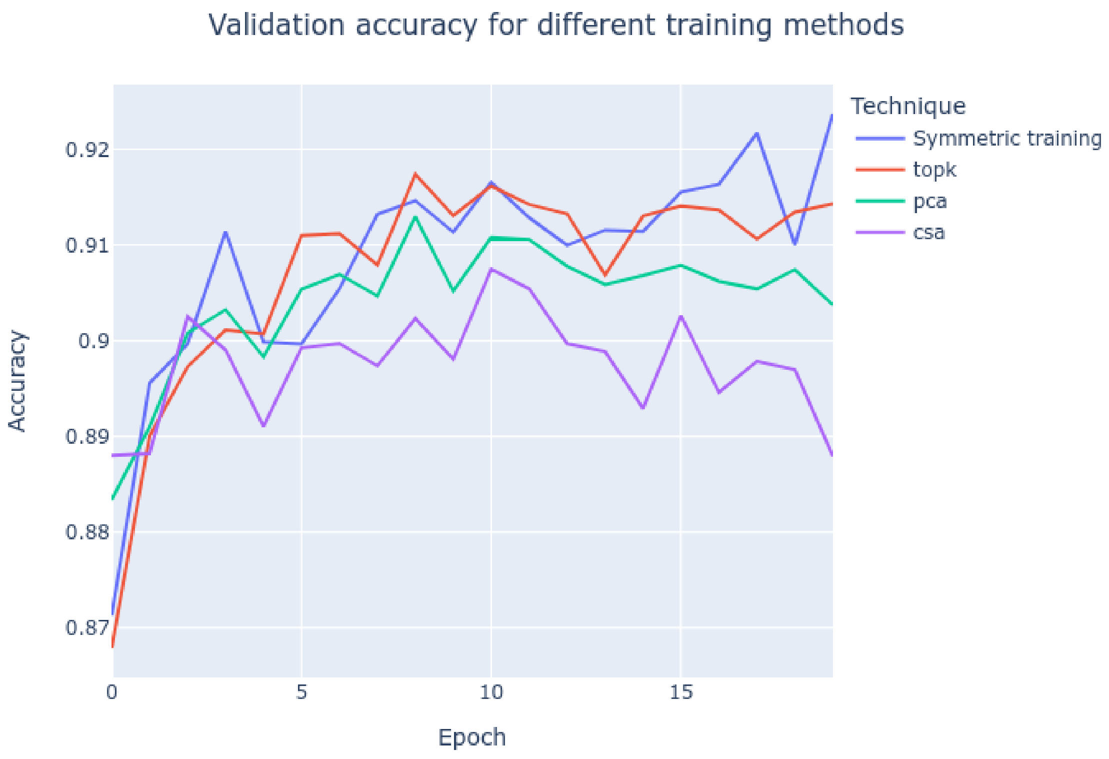

Figure 9.73: Validation accuracy for different training methods for l2-1024

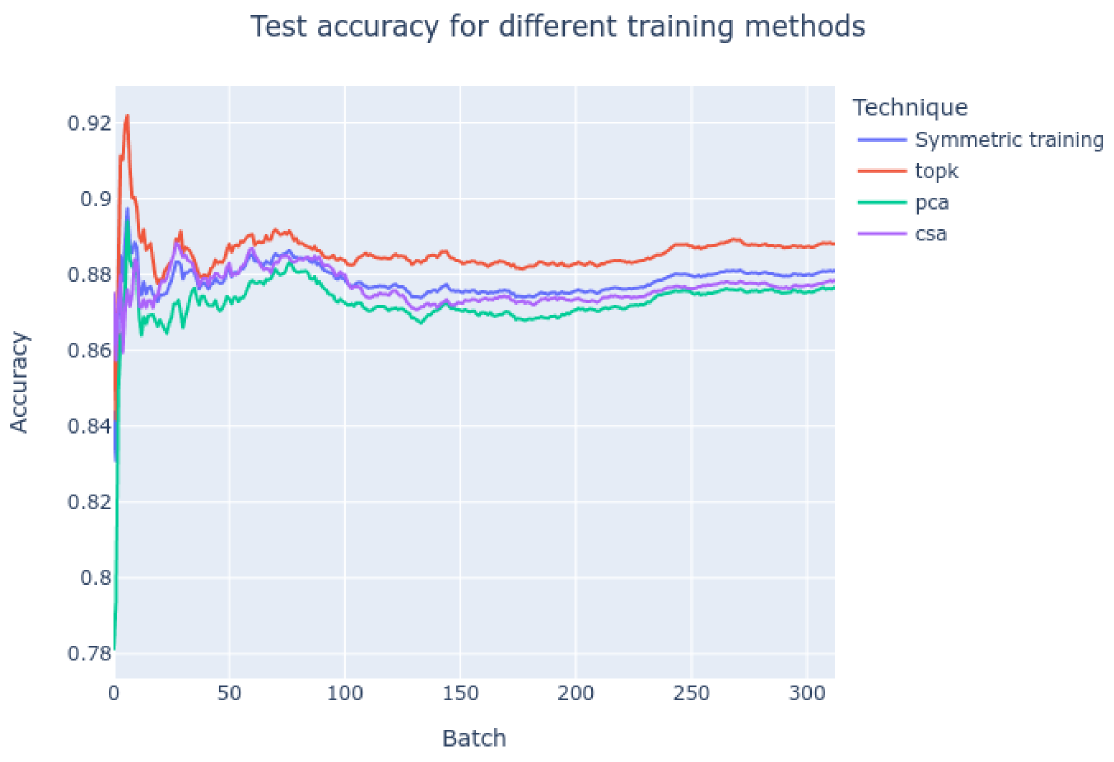

Figure 9.74: Test accuracy for different training methods for l2-1024

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.9085 (0.9032, 0.9139)	0.8783 (0.8777, 0.8788)
Topk Asymmetry k=40	0.9073 (0.9019, 0.9128)	0.8857 (0.8851, 0.8863)
Partial Connectivity Asymmetry k1=40, k2=200	0.9042 (0.9010, 0.9074)	0.8721 (0.8709, 0.8733)
Complete Sparsification Asymmetry k=170	0.8975 (0.8948, 0.9001)	0.8767 (0.8762, 0.8771)

Table 9.45: Accuracy table for l2-1024

b) Time

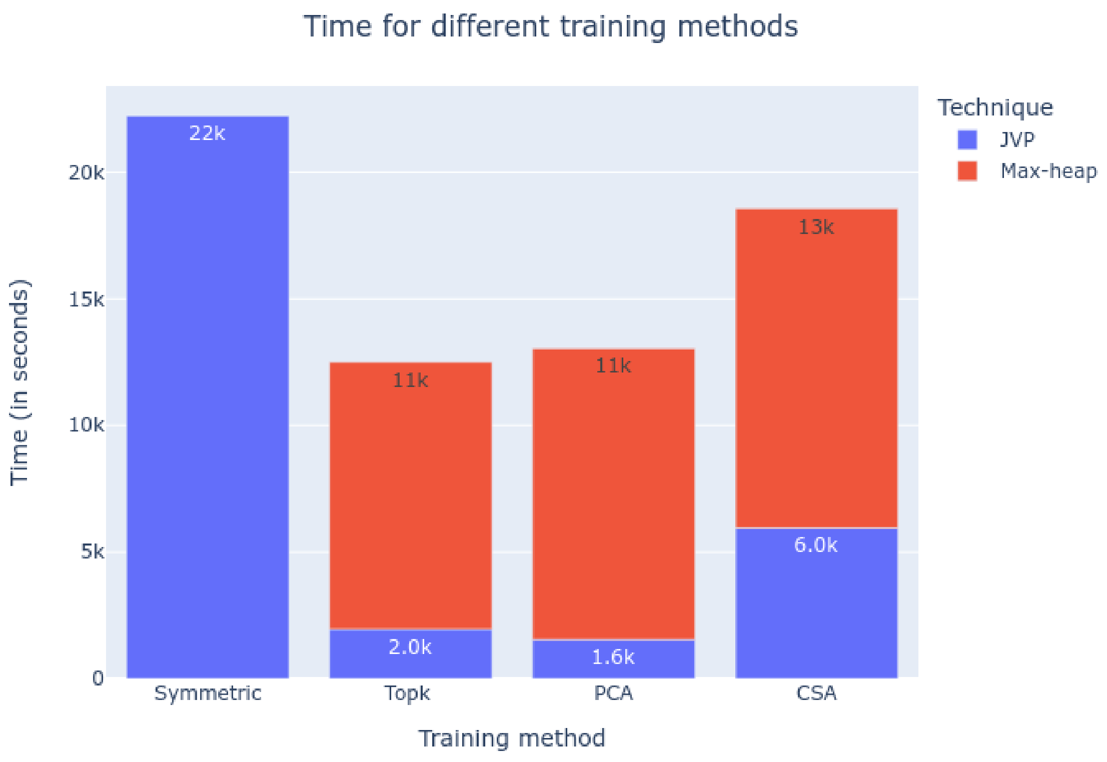

Figure 9.75: Backpropagation time for different training methods for l2-1024

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	22239.01	12497.18	13040.1	18562.63

Table 9.46: Backpropagation time table for different training methods for l2-1024

23. Three hidden layers with 1024 neurons each.

a) Accuracy

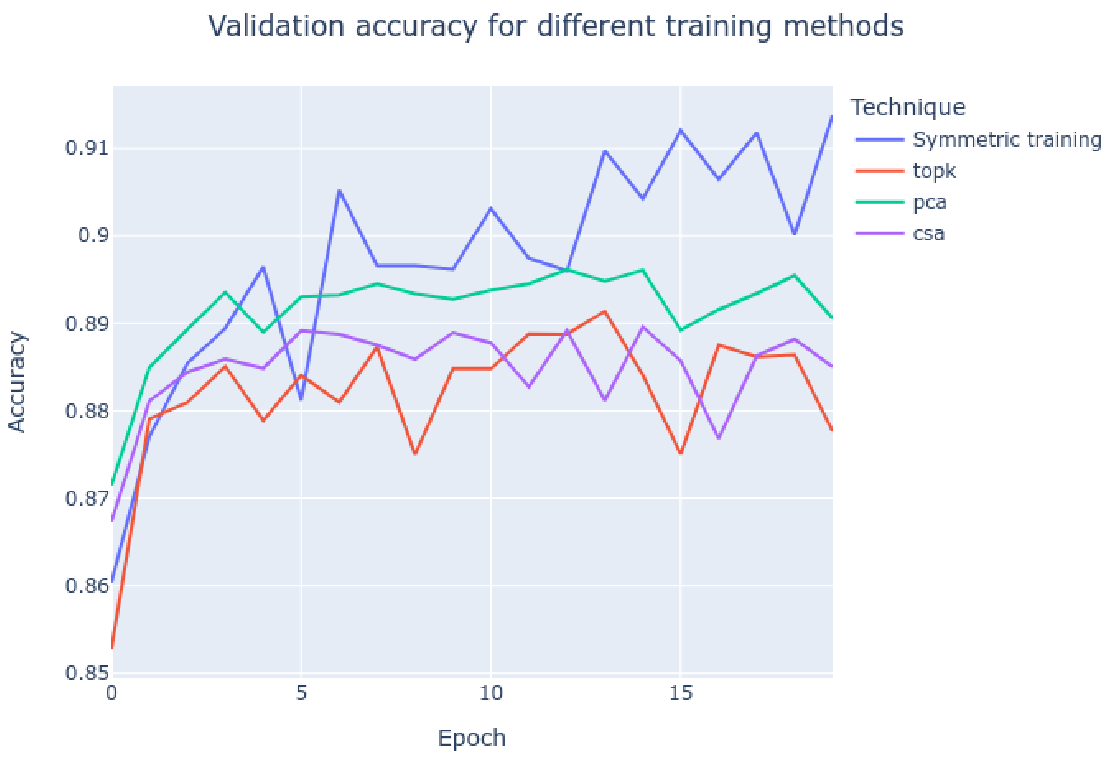

Figure 9.76: Validation accuracy for different training methods for l3-1024

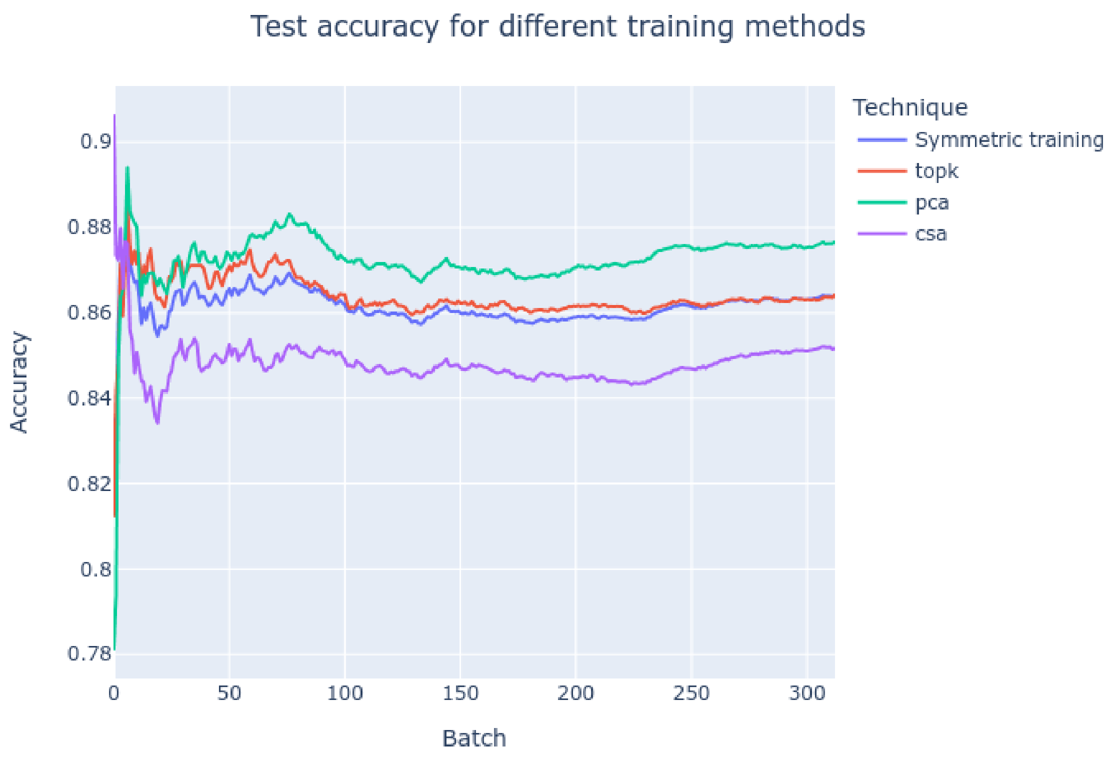

Figure 9.77: Test accuracy for different training methods for l3-1024

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8969 (0.8907, 0.9031)	0.8615 (0.8611, 0.8620)
Topk Asymmetry k=40	0.8819 (0.8780, 0.8858)	0.8639 (0.8633, 0.8644)
Partial Connectivity Asymmetry k1=40, k2=200	0.8915 (0.8889, 0.8940)	0.8726 (0.8715, 0.8734)
Complete Sparsification Asymmetry k=170	0.8848 (0.8823, 0.8872)	0.8483 (0.8476, 0.8489)

Table 9.47: Accuracy table for l3-1024

b) Time

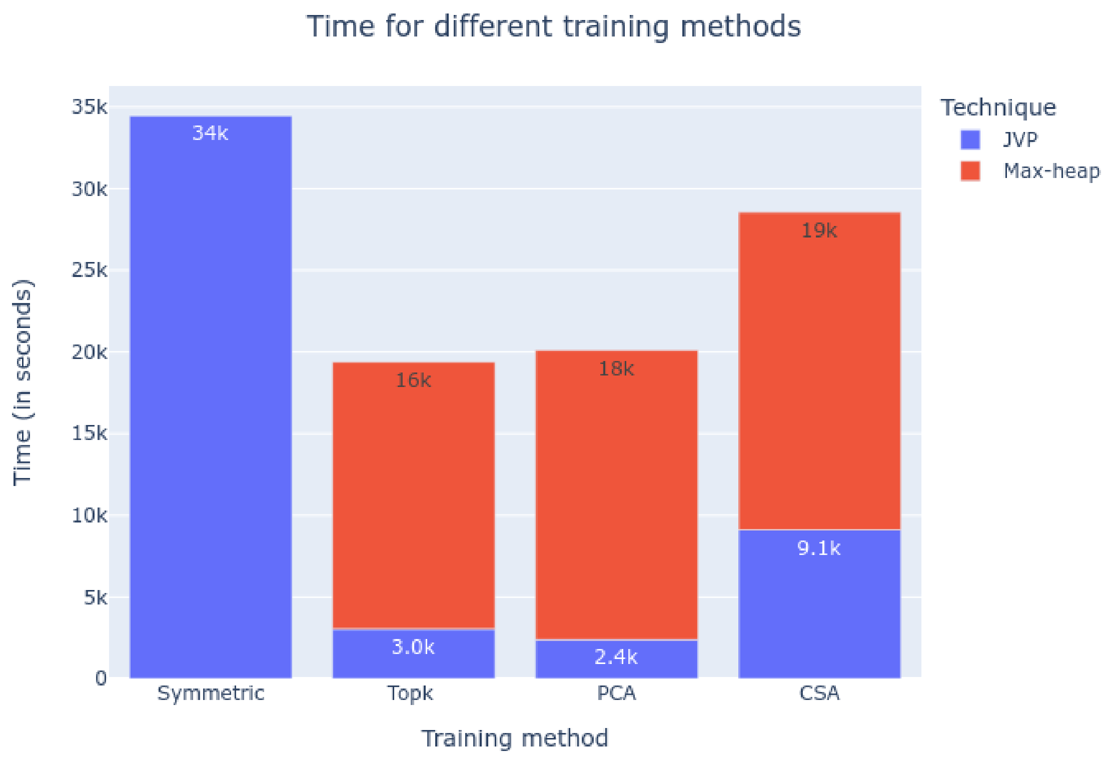

Figure 9.78: Backpropagation time for different training methods for l3-1024

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	34479.06	19383.62	20104.55	28547.29

Table 9.48: Backpropagation time table for different training methods for l3-1024

24. Four hidden layers with 1024 neurons each.

a) Accuracy

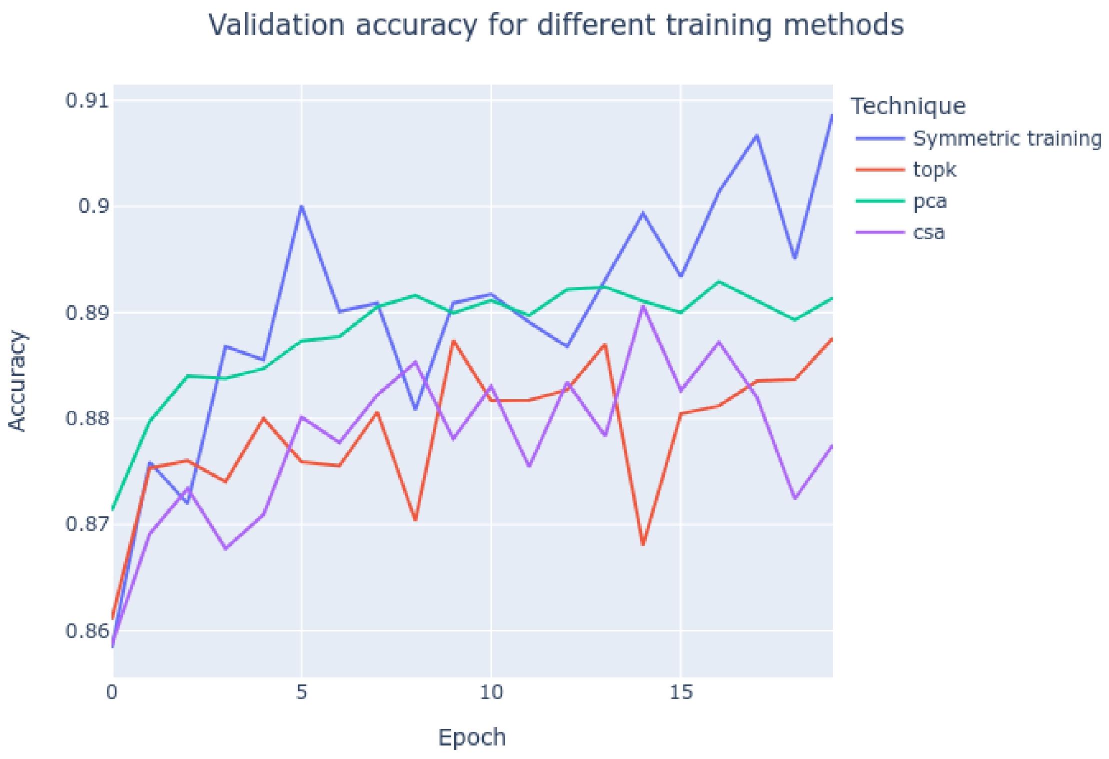

Figure 9.79: Validation accuracy for different training methods for l4-1024

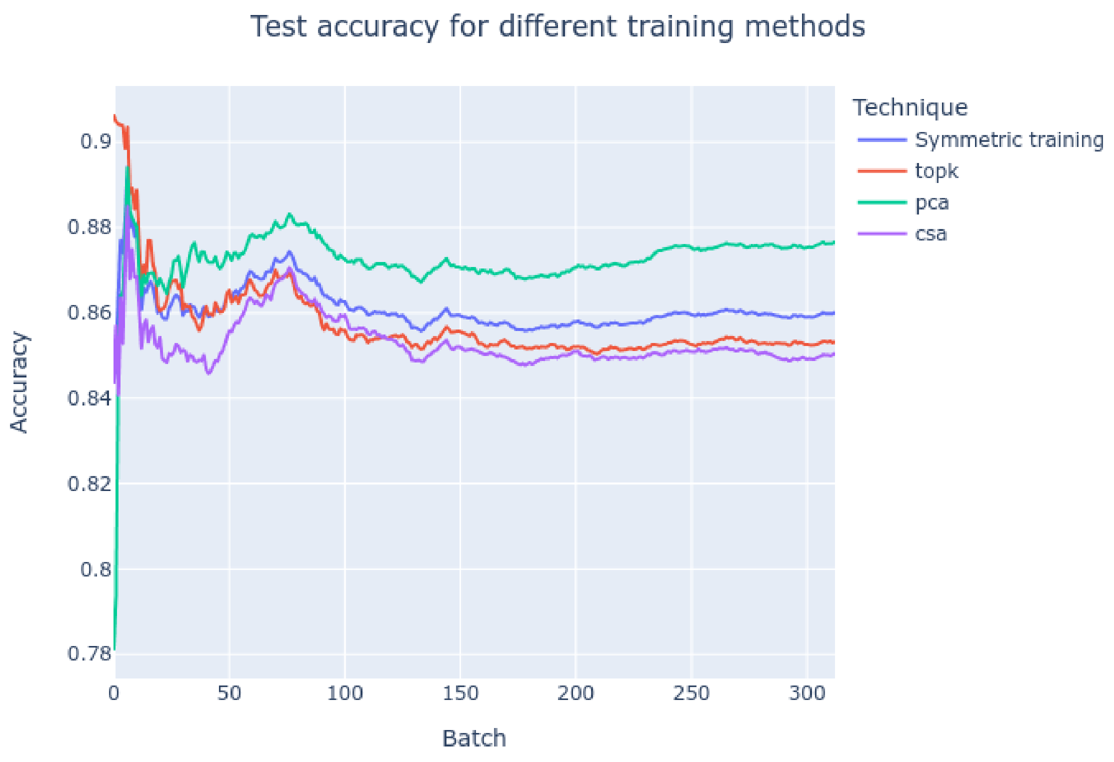

Figure 9.80: Test accuracy for different training methods for l4-1024

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8898 (0.8843, 0.8953)	0.8610 (0.8604, 0.8616)
Topk Asymmetry k=40	0.8786 (0.8755, 0.8818)	0.8575 (0.8564, 0.8586)
Partial Connectivity Asymmetry k1=40, k2=200	0.8881 (0.8856, 0.8905)	0.8724 (0.8715, 0.8733)
Complete Sparsification Asymmetry k=170	0.8778 (0.8742, 0.8813)	0.8531 (0.8525, 0.8538)

Table 9.49: Accuracy table for l4-1024

b) Time

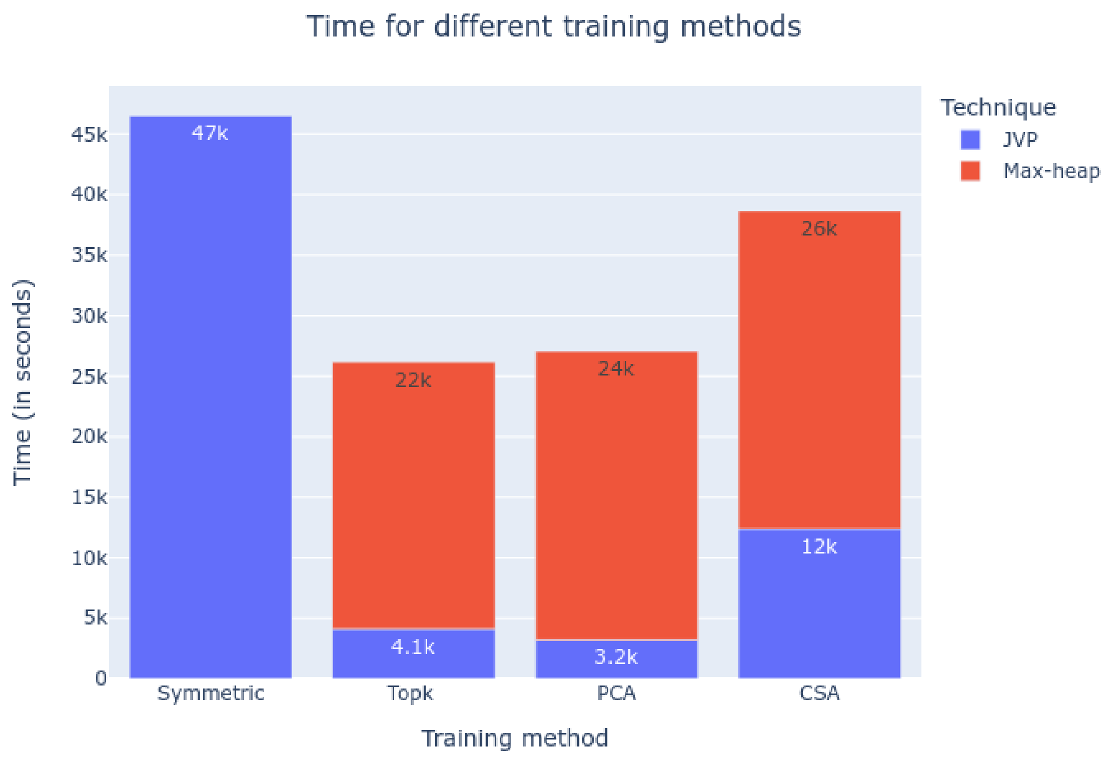

Figure 9.81: Backpropagation time for different training methods for l4-1024

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	46543.86	26153.92	27058.01	38650.6

Table 9.50: Backpropagation time table for different training methods for l4-1024

25. Five hidden layers with 1024 neurons each.

a) Accuracy

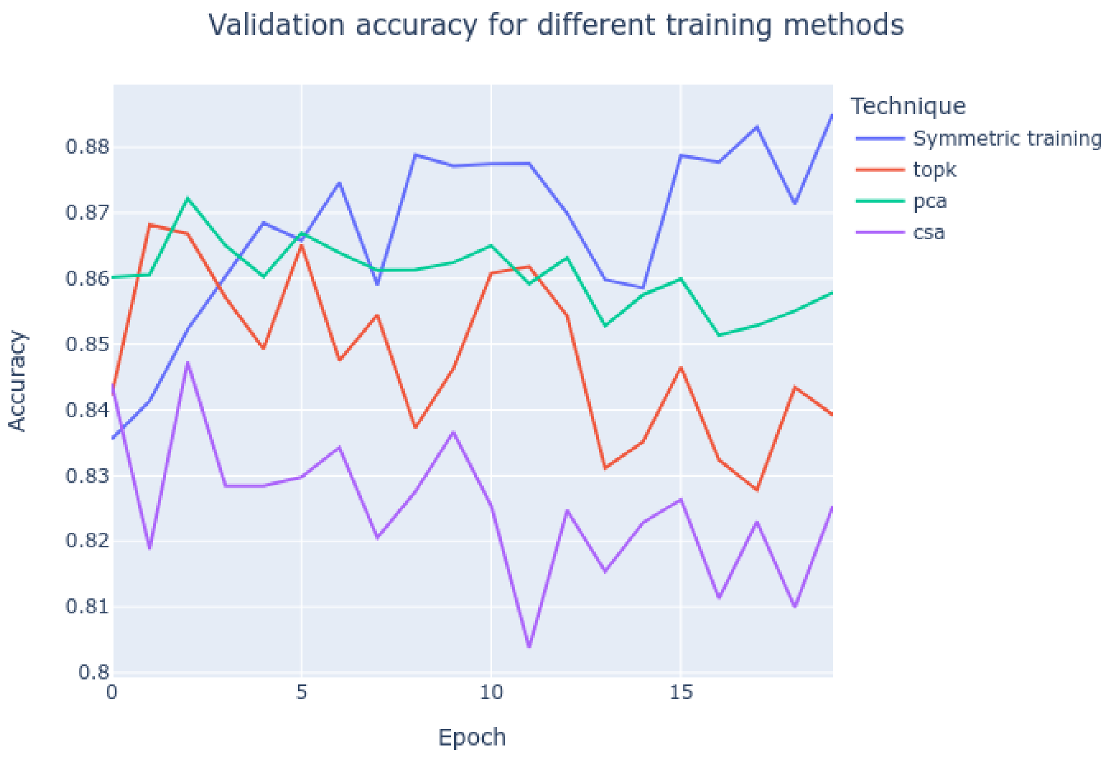

Figure 9.82: Validation accuracy for different training methods for l5-1024

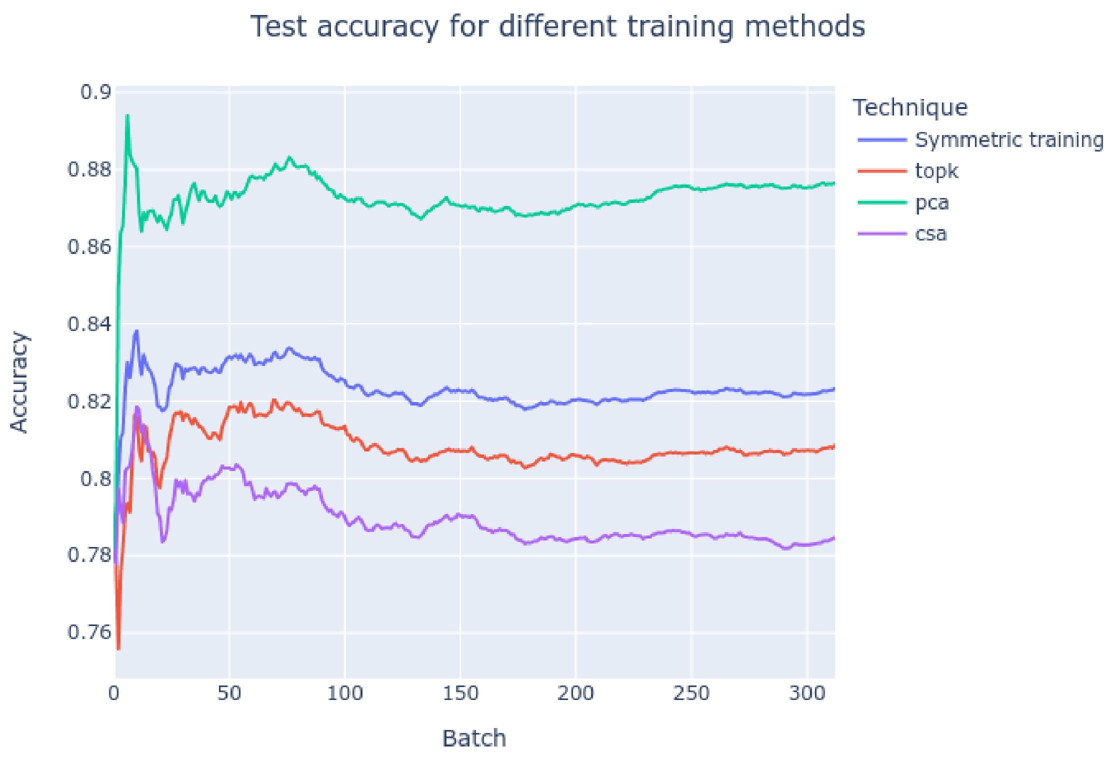

Figure 9.83: Test accuracy for different training methods for l5-1024

Accuracy in 95% CI	Validation accuracy	Test accuracy
Symmetric Dropout- 0.4	0.8676 (0.8612, 0.8739)	0.8233 (0.8226, 0.8239)
Topk Asymmetry k=40	0.8483 (0.8425, 0.8541)	0.8079 (0.8072, 0.8086)
Partial Connectivity Asymmetry k1=40, k2=200	0.8739 (0.8680, 0.8798)	0.8713 (0.8703, 0.8723)
Complete Sparsification Asymmetry k=170	0.8251 (0.8201, 0.8301)	0.7894 (0.7887, 0.7902)

Table 9.51: Accuracy table for l5-1024

b) Time

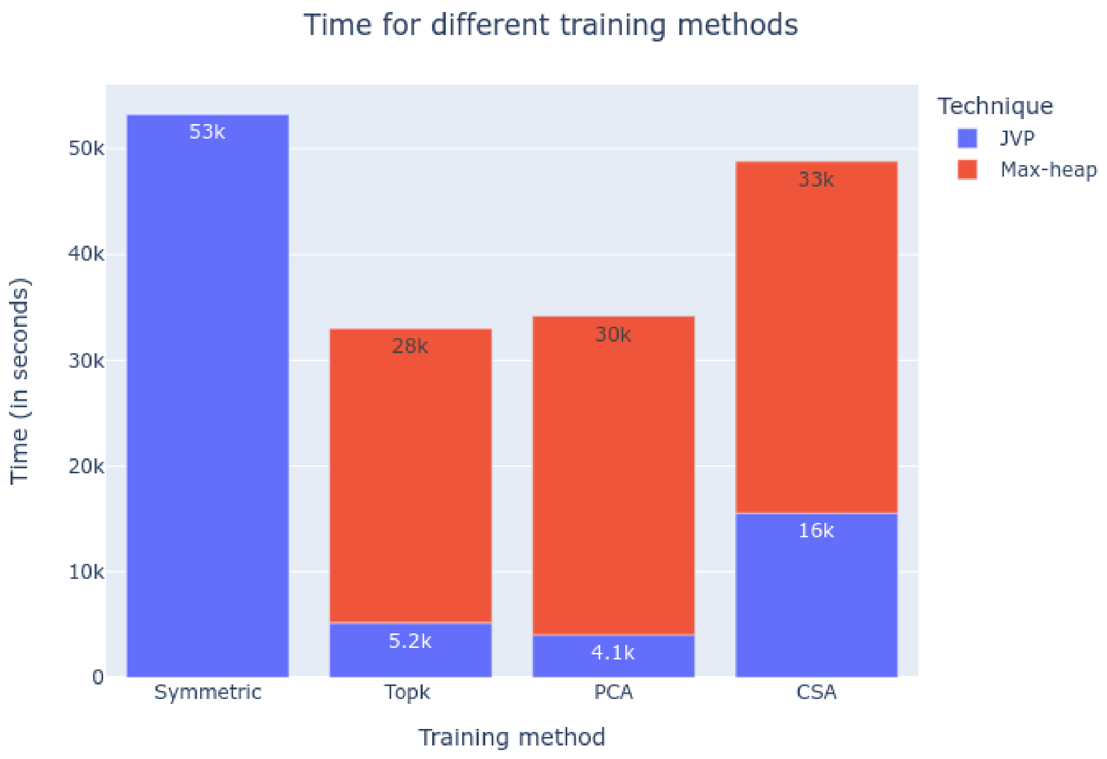

Figure 9.84: Backpropagation time for different training methods for l5-1024

	Symmetric Dropout- 0.4	Topk Asymmetry k=40	Partial Connectivity k1=40, k2=200	Complete Sparsification Asymmetry k=170
Time	53276.58	32982.28	34175.59	48782.31

Table 9.52: Backpropagation time table for different training methods for l5-1024

9.5 Asymptotic analysis

In this section we will present the time complexity and space complexity for the back-propagation time for each of training methods that were used in the experiments. For symmetric training, the backpropagation time is defined as the time required to calculate the JVP. For asymmetric training, the backpropagation time is defined as the time required to calculate the JVP and the time required by the sparsification function to construct the subset of parameters. For the i^{th} layer of the neural network, the number of neurons is the cardinality of the set of neurons $\mathbb{P}_i$ and is represented by $|\mathbb{P}_i|$. Let $\rho_{i,j}$ be the j^{th} neuron in the i^{th} layer of the FFNN, then, the number of incoming edges at $\rho_{i,j}$ from previous, $(i-1)^{th}$ layer, will be $|\mathbb{P}_{i-1}|$.

9.5.1 Time complexity

This section presents the time complexity of the backpropagation time for layer i of the FFNN.

1. Symmetric training with dropout: $O(|\mathbb{P}_i||\mathbb{P}_{i-1}|)$. Where $O(|\mathbb{P}_i||\mathbb{P}_{i-1}|)$ is time required to compute the JVP.

2. Asymmetric training with topk: $O(k|\mathbb{P}_{i-1}|) + O(|\mathbb{P}_i| \log k)$. Where $O(k|\mathbb{P}_{i-1}|)$ is the time required to compute the JVP and $O(|\mathbb{P}_i| \log k)$ is the time required by the sparsification function *topk* to find the k largest values irrespective of the sign using max-heap algorithm.

3. Asymmetric training with complete sparsified asymmetry:
   $O(k^2) + O(|\mathbb{P}_i| \log k + |\mathbb{P}_{i-1}| \log k + |\mathbb{P}_{i-1}||\mathbb{P}_i| \log k)$. Where $O(k^2)$ is the time required to compute the JVP and $O(|\mathbb{P}_i| \log k + |\mathbb{P}_{i-1}| \log k + |\mathbb{P}_{i-1}||\mathbb{P}_i| \log k)$ is the time required by the sparsification function *topk* to find the k largest values irrespective of the sign using max-heap algorithm.

4. Asymmetric training with partial connectivity asymmetry:
   $O(k_1 k_2) + O(|\mathbb{P}_i| \log k_1 + |\mathbb{P}_{i-1}| \log k_2)$. Where $O(k_1 k_2)$ is the time required to

compute the JVP and $O(|\mathbb{P}_i| \log k_1 + |\mathbb{P}_{i-1}| \log k_2)$ is the time required by the sparsification function *topk* to find the k largest values irrespective of the sign using max-heap algorithm.

The time complexity for backpropagation of the entire FFNN would be the summation of the time complexity of each layer.

9.5.2 Space complexity

Each layer of the FFNN using symmetric training requires memory of $O(|\mathbb{P}_i||\mathbb{P}_{i-1}|)$ for the computation of the gradient of the loss function. Each layer of the FFNN using topk and complete sparsified asymmetric training methods requires an additional $O(k)$ memory for storing the indices of the k largest elements irrespective of the sign of each sparsification attribute before computing the JVP. Partial connectivity asymmetry requires an additional $O(max(k_1, k_2))$ memory for storing the indices of the k_1 and k_2 largest elements irrespective of the sign of the sensitivity and the weight adjoint respectively.

9.5.3 Comparison of asymptotic time analysis with empirical results

In all the experiments we found that the empirical time was within the bounds suggested by asymptotic analysis. Empirically, topk sparsification reduces the time taken to calculate the JVP by almost a factor of 10. Empirically,partial connectivity asymmetry reduces the time taken to calculate the JVP even more so by almost a factor of 12; however, the sparsification function is used twice, once to construct a subset of neurons and then to construct a subset of connections for the selected neurons and that increases the max-heap component of the backpropagation time; as a result, the reduction in JVP time is canceled out by the additional max heap time.

Empirically, complete sparsifiction asymmetry takes the most time because not only it applies the sparsification function thrice but also uses redundant sparsifiction attribute for neuron selection, due to which it has to take the union of these different sets of neurons before calculating the JVP. All this makes the complete sparsification backpropagation almost as slow as normal backpropagation.

9.6 Discussion

From the first set of experiments, it seemed that asymmetric training using sparsififed backpropagation gives better accuracy than symmetric training. Further experiments reveal that this is not the case. Out of the three asymmetric training methods we studied in this thesis, topk asymmetric training gave the best accuracy in 60% of the experiments. Partial connectivity asymmetry gave the best accuracy in 28% of the experiments. Models with symmetric training had the best accuracy in the remaining 12% of the experiments. The complete sparsified asymmetric training had the worse accuracy consistently in all the experiments. Partial connectivity asymmetric training gave better accuracy in larger FFNN, such as l3-512 18.

both topk asymmetric training and partial connectivity asymmetric training had up to 2% better accuracy than models with symmetric training. While FFNN trained symmetrically performed poorly, the best accuracy among all the experiments was given by l1-512 9.33 with symmetric training with dropout. We performed over 100 experiments, and FFNN with asymmetric training, topk, and partial connectivity asymmetry had better results in 78% of the experiments. Moreover, both topk asymmetry and partial connectivity asymmetry displayed a consistent reduction in the backpropagation time by up to 50% as compared to symmetrical training. Topk sparsification reduces the time taken to calculate the JVP by almost a factor of 10.

Partial connectivity asymmetry reduces the time taken to calculate the JVP even more so by almost a factor of 12; however, the sparsification function is used twice, once to construct a subset of neurons and then to construct a subset of connections for the selected neurons and that increases the max-heap component of the backpropagation time; as a result, the reduction in JVP time is canceled out by the additional max heap time. Still, compared to symmetric training, both topk and partial connectivity asymmetry reduces the overall backpropagation time by around 50%.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

The thesis generalized the idea of sparsification. We rigorously described how sparsification interacts with neural networks, this led to insights into training dynamics, and we came up with the idea of asymmetric training. We induced three types of asymmetric training by varying sparsified backpropagation. We found out that there exists a trade-off between JVP time and the time taken to construct the parameter subset using the topk sparsification function. More sparsification reduces JVP however, increases the parameter subset construction time. Therefore, the topk asymmetric method had the least total backpropagation time; despite taking more time to calculate the JVP, while the partial connectivity asymmetric training had a higher total backpropagation time despite taking the least amount of time to calculate the JVP. From the first set of experiments, it seemed that asymmetric training using sparsififed backpropagation gives better accuracy than symmetric training. Further experiments reveal that this is not the case. We performed over 100 experiments, and FFNN with asymmetric training, topk, and partial connectivity asymmetry had better results in 78% of the experiments. Moreover, both topk asymmetry and partial connectivity asymmetry displayed a consistent reduction in the backpropagation time by up to 50% as compared to symmetrical training. Thus, asymmetric training could be adopted as a standard way for training models on smaller devices.

10.2 Future works

The reflective analysis of sparsification presented in the thesis is developed in the context of FFNN and considers only two classes of sparsification techniques, these ideas could be extended to other techniques as well, to determine if two seemingly unrelated techniques could be generalized. It would help with the systematic categorization of the vast zoo of techniques in deep learning. Different types of asymmetries could be explored by varying sparsification strategies. Asymmetric training could be extended to neural networks such as CNN or RNN by varying sparsification strategies. Our sparsification framework suggests the existence of more sparsification functions in the sparsification function class that could be discovered by carefully constructing different sparsification attributes. Since we do not have a theoretical description of sparsification, not much is understood about its inner workings [18] therefore; there are no explanations for the results. Moreover, future work exploring the hyperparameter space of different sparsification algorithms such as topk to determine the value of k that results in optimal accuracy and least training time for a given architecture could further enhance the performance of asymmetrically trained models.

Bibliography

[1] [1809.07599] Sparsified SGD with Memory.

[2] The convergence of sparsified gradient methods | Proceedings of the 32nd International Conference on Neural Information Processing Systems.

[3] Linear Algebra and Learning from Data.

[4] Perceptrons:An Introduction to Computational Geometry | Guide books.

[5] A Stochastic Approximation Method.

[6] Understanding Machine Learning - From Theory to Algorithms. | BibSonomy.

[7] A. F. Agarap. Deep Learning using Rectified Linear Units (ReLU). Technical Report arXiv:1803.08375, arXiv, Feb. 2019. arXiv:1803.08375 [cs, stat] type: article.

[8] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization, 2018.

[9] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. Technical Report arXiv:1502.05767, arXiv, Feb. 2018. arXiv:1502.05767 [cs, stat] type: article.

[10] W. D. Blizard. The development of multiset theory. *Modern Logic*, 1(4):319–352, Jan. 1991. Publisher: The Review of Modern Logic.

[11] S. Bornholdt and D. Graudenz. General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, 5(2):327–334, Jan. 1992.

[12] L. Bottou. Stochastic gradient descent tricks.

[13] E. Chien, C. Pan, J. Peng, and O. Milenkovic. You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks, Mar. 2022. arXiv:2106.13264 [cs].

[14] Y. Gal, J. Hron, and A. Kendall. Concrete Dropout. Technical Report arXiv:1705.07832, arXiv, May 2017. arXiv:1705.07832 [stat] type: article.

[15] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, Mar. 2010. ISSN: 1938-7228.

[16] N. Goli and T. M. Aamodt. ReSprop: Reuse Sparsified Backpropagation. pages 1548–1558, 2020.

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[18] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021.

[19] P. T. Inc. Collaborative data science, 2015.

[20] R. Keshari, R. Singh, and M. Vatsa. Guided Dropout. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4065–4072, July 2019. Number: 01.

[21] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. Technical Report arXiv:1412.6980, arXiv, Jan. 2017. arXiv:1412.6980 [cs] type: article.

[22] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6):141–142, Nov. 2012.

[23] T. Parr and J. Howard. The Matrix Calculus You Need For Deep Learning. Technical Report arXiv:1802.01528, arXiv, July 2018. arXiv:1802.01528 [cs, stat] type: article.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Technical Report arXiv:1912.01703, arXiv, Dec. 2019. arXiv:1912.01703 [cs, stat] type: article.

[25] S. J. Rennie, V. Goel, and S. Thomas. Annealed dropout training of deep networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 159–164, Dec. 2014.

[26] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, Mar. 1993.

[27] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. Place: US Publisher: American Psychological Association.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[29] X. Sun, X. Ren, S. Ma, and H. Wang. meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting. page 10.

[30] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[31] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[32] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of Neural Networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1058–1066. PMLR, May 2013. ISSN: 1938-7228.

[33] S. Wang and C. Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, pages 118–126. PMLR, May 2013. ISSN: 1938-7228.

[34] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

[35] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep Sets, Apr. 2018. arXiv:1703.06114 [cs, stat].