Optimisation of Overparametrized Sum-Product Networks

Martin Trapp [1 2] **Robert Peharz** [3] **Franz Pernkopf** [2]

Abstract

It seems to be a pearl of conventional wisdom that parameter learning in deep sum-product networks is surprisingly fast compared to shallow mixture models. This paper examines the effects of overparameterization in sum-product networks on the speed of parameter optimisation. Using theoretical analysis and empirical experiments, we show that deep sum-product networks exhibit an implicit acceleration compared to their shallow counterpart. In fact, gradient-based optimisation in deep sum-product networks is equal to gradient ascend with adaptive and time-varying learning rates and additional momentum terms.

1. Introduction

Tractable probabilistic models, in which exact inference can be tackled efficiently, have gained increasing popularity within and outside the machine learning community. In particular Sum-Product Networks (SPNs) (Poon & Domingos, 2011), - which subsume existing approaches such as latent tree models and deep mixture models (Jaini et al., 2018) - have shown to perform well on various tasks, e.g. image classification (Gens & Domingos, 2012; Peharz et al., 2018), action recognition (Amer & Todorovic, 2016), bandwidth extension (Peharz et al., 2014), language modelling (Cheng et al., 2014), spatial modelling (Pronobis & Rao, 2017) and non-linear regression (Trapp et al., 2018). Much of their success is due to their flexibility, tractability and efficient representation of complex function approximations.

Thus in recent years various ways to perform parameter learning in SPNs and build their structure, e.g. (Gens & Domingos, 2013; Vergari et al., 2015), have been proposed. For example, using a latent variable interpretation of SPNs (Poon & Domingos, 2011; Peharz et al., 2017) one can derive classic expectation-maximization as introduced in (Peharz et al., 2017). Moreover, parameter learning can also be tackled within the Bayesian framework using, e.g. variational inference (Zhao et al., 2016) or Bayesian moment-matching (Rashwan et al., 2016). Optimising a discriminative objective can be achieved using gradient-based optimisation (Gens & Domingos, 2012) and recently Peharz *et al.* (Peharz et al., 2018) introduced a hybrid objective also optimised using gradient-based optimisation. Semi-supervised learning can be tacked likewise, as shown by Trapp *et al.* (Trapp et al., 2017). However, even though many approaches utilise gradient-based optimisation for parameter learning it is not clear if and to which extend the depth of an SPNs has an effect on the speed of optimisation.

On the other hand, analysing the dynamics of optimisation for linear neural networks, see (Baldi & Hornik, 1995) for a survey on linear neural networks, has recently gained increasing interest, e.g. (Saxe et al., 2014; Arora et al., 2018). In particular, Arora *et al.* (Arora et al., 2018) have shown that increasing the depth in linear neural networks can speed up the optimisation. In fact, they showed that the acceleration effect of overparameterization in linear neural networks cannot be achieved by *any* regulariser.

This work discusses the implicit acceleration effects of depth and overparameterization in SPNs, which are a multi-linear function in their weights. As SPNs have been used for non-linear classification tasks, e.g. (Gens & Domingos, 2012; Trapp et al., 2017; Peharz et al., 2018), implicit acceleration effects due to their depth are of particular relevance.

The remainder of this paper is structured as follows: In Section 2 we review recent work on overparameterization in linear neural networks and briefly introduce SPNs. We discuss acceleration effect in SPNs in Section 3 and conclude the work in Section 4.

2. Background and Related Work

2.1. Overparameterization in Linear Networks

Recent work has shown that increasing depth in linear neural networks can speed up the optimisation (Arora et al., 2018). In fact, even gradient-based optimisation of linear regression benefits by moving from a convex objective function to a

[1]Austrian Research Institute for AI, Austria [2]SPSC Lab, Graz University of Technology, Austria [3]CBL Lab, Department of Engineering, University of Cambridge, United Kingdom. Correspondence to: Martin Trapp <martin.trapp@tugraz.at>.

TPM workshop at the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

non-convex objective. In particular, let

$$\ell_p(\boldsymbol{w}) = \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\frac{1}{p}(\boldsymbol{x}^\top\boldsymbol{w} - y)^p\right],$$

be the ℓ_p loss function of a linear regression model parametrised with $\boldsymbol{w} \in \mathbb{R}^D$ and let $\boldsymbol{x} \in \mathbb{R}^D$ be the explanatory variables and $y \in \mathbb{R}$ their respective dependent variable. Note that we use bold font to indicate vectors. Suppose that we now artificially overparametrized this linear model as follows:

$$\boldsymbol{w} = \boldsymbol{w}_1 \cdot w_2\,,$$

for which $\boldsymbol{w}_1 \in \mathbb{R}^D$ and $w_2 \in \mathbb{R}$. Thus, our overparameterization results in the following non-convex loss function:

$$\ell_p(\boldsymbol{w}_1, w_2) = \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[\frac{1}{p}(\boldsymbol{x}^\top\boldsymbol{w}_1 w_2 - y)^p\right].$$

Now let us consider the respective gradients, i.e.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} &:= \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(\boldsymbol{x}^\top\boldsymbol{w} - y)^{p-1}\boldsymbol{x}\right]\\
\nabla_{\boldsymbol{w}_1} &:= \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(\boldsymbol{x}^\top\boldsymbol{w}_1 w_2 - y)^{p-1} w_2\boldsymbol{x}\right]\\
\nabla_{w_2} &:= \mathop{\mathbb{E}}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(\boldsymbol{x}^\top\boldsymbol{w}_1 w_2 - y)^{p-1}\boldsymbol{w}_1^\top\boldsymbol{x}\right]
\end{aligned}$$

allowing to compute the updated parameters at time $t+1$, e.g.

$$\boldsymbol{w}_1^{(t+1)} \leftarrow \boldsymbol{w}_1^{(t)} - \eta\nabla_{\boldsymbol{w}_1^{(t)}}\,,$$

where η is a fixed learning rate.

To understand the dynamics of the underlying parameter $\boldsymbol{w} = \boldsymbol{w}_1 w_2$, Arora *et al.* (Arora et al., 2018) show that

$$\begin{aligned}
\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}_1^{(t+1)} \cdot w_2^{(t+1)}\\
&= \boldsymbol{w}^{(t)} - \eta(w_2^{(t)})^2\nabla_{\boldsymbol{w}^{(t)}} - \eta(w_2^{(t)})^{-1}\nabla_{w_2^{(t)}}\boldsymbol{w}^{(t)}\\
&= \boldsymbol{w}^{(t)} - \rho^{(t)}\nabla_{\boldsymbol{w}^{(t)}} - \lambda^{(t)}\boldsymbol{w}^{(t)}\,.
\end{aligned}$$

Initialising all weights close to zero allows arriving at an update rule that is similar to momentum optimisation (Nesterov, 1983) with an adaptive learning rate. Arora *et al.* further show that overparameterizing fully connected linear neural networks, i.e. increasing the number of layers, leads to an adaptive learning rate and gradient projection amplification that can be thought of as momentum optimisation.

2.2. Sum-Product Networks

A Sum-Product Network (SPN) (Poon & Domingos, 2011) is a rooted directed acyclic graph composed of sum nodes (S), product nodes (P) and leaf nodes (L), i.e. indicators or arbitrary distributions (Peharz et al., 2015). Let $\{X_d\}_{d=1}^D$ be a set of random variables (RVs), then an SPN represents the joint $p(X_1, \ldots, X_D)$ using a recursive structure in which internal nodes (N) either compute a weighted sum, i.e. $\mathsf{S}(\mathbf{x}) = \sum_{\mathsf{N}\in\mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}}\,\mathsf{N}(\mathbf{x})$, or compute a product of the values of their children, i.e. $\mathsf{P}(\mathbf{x}) = \prod_{\mathsf{N}\in\mathbf{ch}(\mathsf{P})}\mathsf{N}(\mathbf{x})$, where $\mathbf{ch}(\mathsf{N})$ denotes the children of node N. Each edge (S,N) emanating from a sum node S has a non-negative weight $w_{\mathsf{S},\mathsf{N}}$. The sum node is said to be normalised if the weights sum to one.

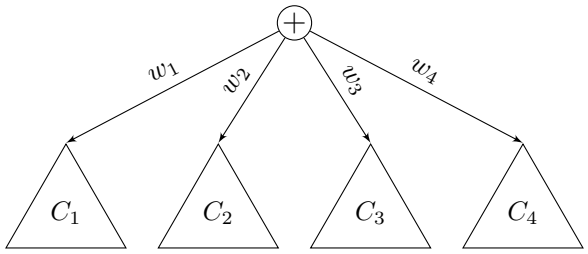

Figure 1. Illustration of a shallow SPN. Triangles denote sub-SPNs or distributions under the root of the SPN.

In the context of this work, we require SPNs to be *complete* and *decomposable* (Poon & Domingos, 2011; Darwiche, 2003). Both concepts can be expressed in terms of the scope of the nodes. The scope of a node in an SPN is the set of variables that appear in it, i.e. the node models a joint over those RV's. An SPN is *complete* if for each sum node S the scopes of all children of S are equal. Further, an SPN is *decomposable*, if all children of a product node have non-overlapping scopes. The scope of each internal node is defined as the union of its children's scopes. Due to those two conditions, SPNs allow for efficient marginalisation, conditioning, sampling, and exact inference.

3. Overparameterization in Sum-Product Networks

For SPNs there seems to be a pearl of conventional wisdom that parameter learning is surprisingly fast. Following the approach by Arora *et al.* (Arora et al., 2018) we will now discuss the implicit dynamics of gradient-based optimisation in SPNs. To simplify the discussion, consider the network structure illustrated in Figure 1, i.e. an SPN with a root node that computes a weighted sum over sub-networks $C_1, \ldots, C_K$.

We can denote the log-likelihood function of this network as

$$\mathcal{L}(\theta \mid \mathcal{X}) = \sum_{n=1}^{N} \log \mathrm{f}(\mathbf{x}_\mathrm{n} \mid \theta) - \log \mathrm{f}(* \mid \theta)\,, \qquad (1)$$

where $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ is a set of observed data and θ contains

all parameters of our model. Further we have

$$\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta) = \sum_{k=1}^{K} w_k C_k(\mathbf{x}_n)\,,$$

where $C_k(\boldsymbol{x})$ denotes the value of child k for observation $\mathbf{x}_n$. Note that $\mathrm{f}(* \,|\, \theta)$ is the partition function[1], see (Poon & Domingos, 2011) for details. The normalisation is only necessary if the network is not normalised, i.e. $\mathrm{f}(* \,|\, \theta) \neq 1$. For deriving our results we will assume that all w_k are initialised as $w_k \approx 0$, i.e. the network is not normalised.

To maximise Equation 1 we can use vanilla gradient ascend with the following gradients:

$$\nabla_{w_k} := \frac{1}{\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta)} C_k(\boldsymbol{x}) - \frac{1}{\mathrm{f}(* \,|\, \theta)} C_k(*)\,, \tag{2}$$

where $C_k(*)$ denotes the partition function of sub-network C_k. Thus optimising Equation 1 can be achieved by updating w_k accordingly at each iteration t.

Let us now consider an overparametrized version of this network by introducing additional sum nodes into the model. For this purpose, let each weight w_k be decomposed into multiple independent weights. See Figure 2 for an illustration of an overparametrized version of our initial network. Note that this SPN is in its expressiveness equivalent to our initial model and only introduces additional parameters. Therefore, let $w_j^{[l]}$ denote the j^{th} weight of layer l. We can define the decomposition of w_k by L sum node layers as

$$w_k = \prod_{l=0}^{L} w_{\phi(k,l)}^{[l]}\,, \tag{3}$$

where $\phi(k,l)$ maps the index k onto the respective index at layer l. For example, in the case of our model in Figure 2 we can define $\phi(k,l)$ as

$$\phi(k,l) = \begin{cases} \lceil \frac{k}{2} \rceil, & \text{if } l = 0 \\ k, & \text{otherwise} \end{cases}\,,$$

where $\lceil\, \rceil$ denotes the ceiling operator. Note that the definition of $\phi(k,l)$ generally depends on the network structure of the SPN.

Similarly, we can denote the log-likelihood function of an overparametrized network as

$$\mathcal{L}(\theta \,|\, \mathcal{X}) = \sum_{n=1}^{N} \log \mathrm{g}(\mathbf{x}_\mathrm{n} \,|\, \theta) - \log \mathrm{f}(* \,|\, \theta)\,, \tag{4}$$

where

$$\mathrm{g}(\mathbf{x}_\mathrm{n} \,|\, \theta) = \sum_{k=1}^{K} \prod_{l=0}^{L} w_{\phi(k,l)}^{[l]} C_k(\mathbf{x}_\mathrm{n})\,.$$

[1]The partition function is obtained by setting all indicators to one.

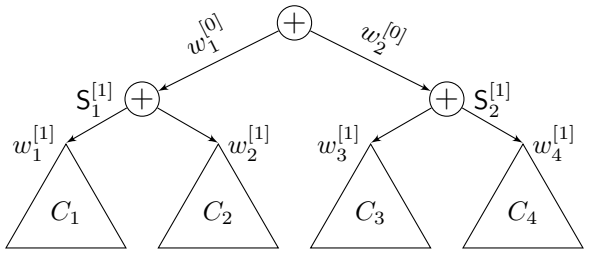

Figure 2. Illustration of an overparametrized SPN. Triangles denote sub-SPNs or distributions.

Note that $\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta) = \mathrm{g}(\mathbf{x}_\mathrm{n} \,|\, \theta)$ as we defined $w_k = \prod_{l=0}^{L} w_{\phi(k,l)}^{[l]}$. Let $\gamma := \lceil \frac{k}{2} \rceil$ and let us assume the SPN illustrated in Figure 2. Therefore, we have the decomposition

$$w_k = w_\gamma^{[0]} \cdot w_k^{[1]}$$

for each weight k. Thus we define the gradients of $w_\gamma^{[0]}$ and $w_k^{[1]}$ as

$$\begin{aligned} \nabla_{w_k^{[1]}} &:= \frac{w_\gamma^{[0]}}{\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta)} C_k(\mathbf{x}_n) - \frac{w_\gamma^{[0]}}{\mathrm{f}(* \,|\, \theta)} C_k(*) \\ &= w_\gamma^{[0]} \left[\frac{1}{\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta)} C_k(\mathbf{x}_n) - \frac{1}{\mathrm{f}(* \,|\, \theta)} C_k(*) \right] \\ &= w_\gamma^{[0]} \nabla_{w_k}\,, \end{aligned} \tag{5}$$

and

$$\begin{aligned} \nabla_{w_\gamma^{[0]}} &:= \sum_{j \in \mathbf{ch}(S^{[1]}_{\phi(k,0)})} \frac{w_j^{[1]}}{\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta)} C_j(\mathbf{x}_n) - \frac{w_j^{[1]}}{\mathrm{f}(* \,|\, \theta)} C_j(*) \\ &= \sum_{j \in \mathbf{ch}(S^{[1]}_{\phi(k,0)})} w_j^{[1]} \nabla_{w_j}\,, \end{aligned} \tag{6}$$

where $S^{[1]}_{\phi(k,0)}$ denotes the sum node at the first layer which is connected to the edge with weight $w_\gamma^{[0]}$. In more general terms we can say that the gradient of $w_{\phi(k,l)}^{[l]}$ at layer l is defined by:

$$\nabla w_{\phi(k,l)}^{[l]} := \sum_{w_{\phi(k,l)}^{[l]} \rightsquigarrow j} (w_{\phi(k,l)}^{[l]})^{-1} w_j \nabla w_j \tag{7}$$

where we use $w_{\phi(k,l)}^{[l]} \rightsquigarrow j$ to denote the set of all weights $w_j := \prod_{l=0}^{L} w_{\phi(j,l)}^{[l]}$ for which $w_{\phi(k,l)}^{[l]}$ is included in the decomposition of w_j.

Similar to Arora *et al.* (Arora et al., 2018), we can now examine the dynamics of w_k by assuming a small learning

rate η and by assuming that all weights are initialised near zero. For this let $w_k^{(t+1)}$ denote w_k at time $t+1$. Therefore, we obtain

$$w_k^{(t+1)} = w_\gamma^{[0](t+1)} \cdot w_k^{[1](t+1)} \tag{8}$$

$$= \left[w_\gamma^{[0](t)} + \eta \nabla_{w_\gamma^{[0](t)}} \right] \left[w_k^{[1](t)} + \eta \nabla_{w_k^{[1](t)}} \right] \tag{9}$$

$$= w_\gamma^{[0](t)} w_k^{[1](t)} + \eta w_\gamma^{[0](t)} \nabla_{w_k^{[1](t)}} + \eta w_k^{[1](t)} \nabla_{w_\gamma^{[0](t)}} + \mathcal{O}(\eta^2)\,, \tag{10}$$

where we can drop $\mathcal{O}(\eta^2)$ as we assume η to be small. Thus we can reformulate Equation 10 as follows:

$$w_k^{(t+1)} \approx w_k^{(t)} + \eta (w_\gamma^{[0](t)})^2 \nabla_{w_k^{(t)}} + \eta \nabla_{w_\gamma^{[0](t)}} (w_\gamma^{[0](t)})^{-1} w_k^{(t)} \tag{11}$$

$$= w_k^{(t)} + \rho^{(t)} \nabla_{w_k^{(t)}} + \lambda^{(t)} w_k^{(t)}\,. \tag{12}$$

We can find the solution for any decomposition, c.f. Equation 3, by considering the gradients in Equation 7. Thus the update rule of the weights for any overparametrized SPNs is

$$w_k^{(t+1)} \approx w_k^{(t)} + \eta (w_{\phi(k,0)}^{[0]})^2 \nabla_{w_k^{(t)}} + \left[\sum_{l=0}^{L-1} \eta \nabla_{w_{\phi(k,l)}^{[l]}} (w_{\phi(k,l)}^{[l]})^{-1} \right] w_k^{(t)}\,, \tag{13}$$

where we drop terms in which we have η to the power of two or larger as we assume η to be small. We can now define the adaptive and time-varying learning rate

$$\rho^{(t)} := \eta (w_{\phi(k,0)}^{[0]})^2\,,$$

and let

$$\lambda^{(t)} := \sum_{l=0}^{L-1} \eta \nabla_{w_{\phi(k,l)}^{[l]}} (w_{\phi(k,l)}^{[l]})^{-1}\,.$$

Note that we pulled out $l=0$ to obtain the gradient of w_k, c.f. second term in Equation 13, thus resulting in the weight $\lambda^{(t)}$ to be a summation over $L-1$ terms. Therefore, we can see that the gradient updates of w_k are directly influenced by the depth of the network.

Let all weights be initialisation near zero, then $w_k^{(t)}$ can be understood as a weighted combination of past gradients and thus there exists a $\mu^{(t,\tau)} \in \mathbb{R}$ such that the dynamics of $w_k^{(t)}$ correspond to gradient optimisation with momentum, i.e.

$$w_k^{(t)} \approx w_k^{(t)} + \rho^{(t)} \nabla_{w_k^{(t)}} + \sum_{\tau=1}^{t-1} \mu^{(t,\tau)} \nabla_{w_k^{(\tau)}}\,. \tag{14}$$

Observation 1. *Gradient-based optimisation of an overparametrized sum-product network with small (fixed) learning rate and near zero initialisation of the weights is equivalent to gradient-based optimisation with adaptive and time-varying learning rate and momentum terms.*

So far we have only considered the case in which we artificially introduce additional sum nodes to the network. However, an important question is if the depth of an SPN also implicitly accelerates parameter learning. As shown by (Zhao et al., 2016) the density function of an SPN can be written as a mixture over induced trees, i.e.

$$\mathrm{f}(\mathbf{x}_\mathrm{n} \,|\, \theta) = \sum_{k=1}^{\kappa} \prod_{w_{S,C} \in \mathcal{T}_k} w_{S,C} \prod_{\mathsf{L} \in \mathcal{T}_k} p(\mathbf{x}_n \,|\, \theta_\mathsf{L})\,,$$

where $\mathcal{T}_k$ denotes the k^{th} induced tree, κ is the number of induced trees and θ_L indicates the parameter of the leaf L which is included in $\mathcal{T}_k$.

Definition 1 (Induced tree)**.** *Let $\mathcal{S}$ be a complete and decomposable (SPN). An induced tree $\mathcal{T}$ of $\mathcal{S}$ is a sub-tree of $\mathcal{S}$ for which i) the root of $\mathcal{T}$ is the root of $\mathcal{S}$, ii) each sum node S in $\mathcal{T}$ has only one child C and both S and C as well as $w_{\mathsf{S},\mathsf{C}}$ are in $\mathcal{S}$, iii) each product node P in $\mathcal{T}$ has the same children as in $\mathcal{S}$.*

Given a representation of an SPN by a mixture of κ many induced trees, for every component k the weight decomposition of its component is given by the respective induced tree, i.e.

$$w_k = \prod_{w_{S,C} \in \mathcal{T}_k} w_{S,C}\,.$$

Therefore, there exists an index function $\phi(k,l) \to (\mathsf{S},\mathsf{C})$ such that the decompositions of the components weights can be represented by an overparametrized SPN without changing the decompositions, the order of the weights or the weights itself, i.e.

$$\prod_{l=0}^{L_{\mathcal{T}_k}} w_{\phi(k,l)}^{[l]} := \prod_{w_{S,C} \in \mathcal{T}_k} w_{S,C}$$

where $L_{\mathcal{T}_k}$ is the depth of the induced tree.

Thus, we can say that gradient-based optimisation using $\prod_{w_{S,C} \in \mathcal{T}_k} w_{S,C}$ is equivalent to gradient-based optimisation using $\prod_{l=0}^{L_{\mathcal{T}_k}} w_{\phi(k,l)}^{[l]}$ and entails the same acceleration effects.

Claim 1. *Gradient-based optimisation of any deep sum-product network with small (fixed) learning rate and near zero initialisation of the weights is equivalent to gradient-based optimisation with adaptive and time-varying learning rate and momentum terms.*

Note that it is, therefore, not necessary to explicitely obtain an overparameterized SPN from a naturally deep SPN. Further, the same result can be shown for supervised and semi-supervised learning of SPNs with linear or non-linear leaves, e.g. as used in (Gens & Domingos, 2012; Trapp et al., 2017; Peharz et al., 2018). Thus, indicating that the

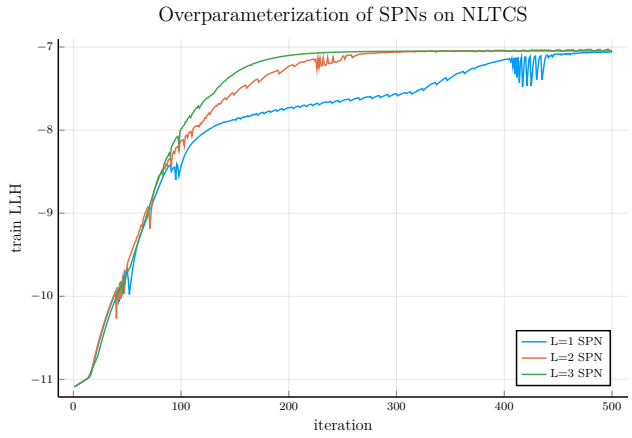

Figure 3. Empirical evaluation of overparameterization in sum-product networks (SPNs). L denotes the number of consecutive sum layers. We can see that adding additional parameters helps gradient-based optimisation and leads to an implicitly acceleration, higher LLH values are better. (Best seen in colour.)

depth of a network can help to accelerate parameter optimisation of non-linear classifiers if the non-linearities only occur at the terminal nodes.

3.1. Empirical Results

To empirically evaluate the effects of overparameterization in SPNs, we compared the speed of optimisation of a shallow SPN (similar to the one shown in Figure 1) to deep variants with the same expressiveness as the shallow model.[2] We used SPNs with 8 components $(C_1, \dots, C_8)$ and initialised all weights close to zero. Further we used vanilla gradient ascend for 500 iterations with a learning rate of $\eta = 0.01$. We used data extracted from the National Long Term Care Survey (NLTCS)[3] to examine the effects of overparameterization which contains 16 binary variables representing functional disability measures. We used only the training set, as processed by (Lowd & Davis, 2010), consisting of 16.181 observed samples to train each model. For each iteration we reported the log-likelihood on the training set (train LLH) resulting in the curves shown in Figure 3. Note that we estimated the train LLH curves over 2 independent re-runs to account for random effects of the structure generation.

We can see from the empirical experiment that increasing the number of sum layers, which is equivalent to increasing the depth of the network, leads to faster parameter optimisation. In fact, using $L = 3$ consecutive sum layers accelerates the optimisation in a way that a near optimal solution is found after only half of the number of iterations compared to the shallow construction, i.e. $L = 1$.

[2]The code to reproduce the experiment can be found on GitHub under https://github.com/trappmartin/TPM2019.

[3]http://lib.stat.cmu.edu/datasets/

4. Conclusion

We have shown that overparameterization of sum-product networks (SPNs) has similar dynamics as observed in linear neural networks. In fact, we observe that the dynamics in SPNs correspond to gradient optimisation with adaptive and time-varying learning rate and momentum terms leading to an implicit acceleration of gradient-based optimisation. This observation is not completely surprising as SPNs are multi-linear functions in their weights which can be represented by sparsely connected linear-neural networks with potentially non-linear input units. As SPNs have been used for non-linear classification/modelling tasks frequently in recent years, the observation that the depth of such networks accelerates parameter learning is of particular relevance.

Acknowledgements

This research was partially funded by the Austrian Science Fund (FWF) under the project number I2706-N31.

References

Amer, M. R. and Todorovic, S. Sum product networks for activity recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(4):800–813, 2016.

Arora, S., Cohen, N., and Hazan, E. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the 35th International Conference on Machine Learning ICML*, pp. 244–253, 2018.

Baldi, P. and Hornik, K. Learning in linear neural networks: a survey. *IEEE Trans. Neural Networks*, 6(4):837–858, 1995.

Cheng, W., Kok, S., Pham, H. V., Chieu, H. L., and Chai, K. M. A. Language modeling with sum-product networks. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association INTERSPEECH*, pp. 2098–2102, 2014.

Darwiche, A. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

Gens, R. and Domingos, P. M. Discriminative learning of sum-product networks. In *Proceedings of the Advances in Neural Information Processing Systems NIPS*, pp. 3248–3256, 2012.

Gens, R. and Domingos, P. M. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning ICML*, pp. 873–880, 2013.

Jaini, P., Poupart, P., and Yu, Y. Deep homogeneous mixture models: Representation, separation, and approximation. In *Proceedings of the Advances in Neural Information Processing Systems NeurIPS*, pp. 7136–7145, 2018.

Lowd, D. and Davis, J. Learning markov network structure with decision trees. In *Proceedings of the 10th International Conference on Data Mining ICDM*, pp. 334–343, 2010.

Nesterov, Y. E. A method for solving the convex programming problem with convergence rate O(1/kˆ2). *Soviet Mathematics Doklady*, 269:543–547, 1983.

Peharz, R., Kapeller, G., Mowlaee, P., and Pernkopf, F. Modeling speech with sum-product networks: Application to bandwidth extension. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP*, pp. 3699–3703, 2014.

Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. M. On theoretical properties of sum-product networks. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics AISTATS*, 2015.

Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. M. On the latent variable interpretation in sum-product networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(10):2030–2044, 2017.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Kersting, K., and Ghahramani, Z. Probabilistic deep learning using random sum-product networks. *CoRR*, abs/1806.01910, 2018.

Poon, H. and Domingos, P. M. Sum-product networks: A new deep architecture. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence UAI*, pp. 337–346, 2011.

Pronobis, A. and Rao, R. P. N. Learning deep generative spatial models for mobile robots. In *Proceedings of the International Conference on Intelligent Robots and Systems IROS*, pp. 755–762, 2017.

Rashwan, A., Zhao, H., and Poupart, P. Online and distributed bayesian moment matching for parameter learning in sum-product networks. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics AISTATS*, pp. 1469–1477, 2016.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the 2nd International Conference on Learning Representations ICLR*, 2014.

Trapp, M., Madl, T., Peharz, R., Pernkopf, F., and Trappl, R. Safe semi-supervised learning of sum-product networks. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence UAI*, 2017.

Trapp, M., Peharz, R., Rasmussen, C. E., and Pernkopf, F. Learning deep mixtures of gaussian process experts using sum-product networks. *CoRR*, abs/1809.04400, 2018.

Vergari, A., Mauro, N. D., and Esposito, F. Simplifying, regularizing and strengthening sum-product network structure learning. In *Proceeding of the European Conference on Machine Learning and Knowledge Discovery in Databases ECML*, pp. 343–358, 2015.

Zhao, H., Adel, T., Gordon, G. J., and Amos, B. Collapsed variational inference for sum-product networks. In *Proceedings of the 33nd International Conference on Machine Learning ICML*, pp. 1310–1318, 2016.