BAYESIAN TRANSFER LEARNING FOR DEEP NETWORKS

J. Wohlert, A. M. Munk, S. Sengupta, F. Laumann

Technical University of Denmark
Department of Applied Mathematics and Computer Science
2800 Lyngby, Denmark

ABSTRACT

We propose a method for transfer learning for deep networks through Bayesian inference, where an approximate posterior distribution $q(\boldsymbol{w}|\theta)$ of model parameters $\boldsymbol{w}$ is learned through variational approximation. Utilizing Bayes by Backprop we optimize the parameters θ associated with the approximate distribution. When performing transfer learning we consider two tasks; A and B. Firstly, an approximate posterior $q_A(\boldsymbol{w}|\theta)$ is learned from task A which is afterwards transferred as a prior $p(\boldsymbol{w}) \to q_A(\boldsymbol{w}|\theta)$ when learning the approximate posterior distribution $q_B(\boldsymbol{w}|\theta)$ for task B. Initially, we consider a multivariate normal distribution $q(\boldsymbol{w}|\theta) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with diagonal covariance matrix $\boldsymbol{\Sigma}$. Secondly, we consider the prospects of introducing more expressive approximate distributions - specifically those known as normalizing flows. By investigating these concepts on the MNIST data set we conclude that utilizing normalizing flows does not improve Bayesian inference in the context presented here. Further, we show that transfer learning is not feasible using our proposed architecture and our definition of task A and task B, but no general conclusion regarding rejecting a Bayesian approach to transfer learning can be made.

1. INTRODUCTION

When optimally trained, deep neural networks (DNN) excel at solving a variety of complex and specialized tasks, such as style transfer [1] and speech recognition [2], but suffer from what is known as *catastrophic forgetting* [3]. The term describes the inability for classically trained DNNs to remember previous tasks when exposed to and trained to solve new ones. Most current methods use what is known as the *multitask learning* paradigm [4] in order to alleviate the issue of forgetting. The method involves interleaving data associated with different tasks before training. Thus, network weights are optimized to solve the joint set of tasks. If, on the other hand, tasks are presented sequentially, training requires task-associated data to be remembered and reused every time a new task is presented. This, however, is often undesirable for large amounts of tasks as memory requirements and number of retraining sessions is proportional to number of tasks. Secondly, this approach arguably does not reflect true learning of previous learned tasks. The model is simply retrained to map increasingly complex systems, rather than dynamically adapt when confronted with a new task. A new recently proposed strategy allows models to continue learning through what they present as *elastic weight consolidation* (EWC) [5]. The method considers data from two different tasks as independent, and by inferring the posterior weight distribution for task A $p(\boldsymbol{w}|\mathcal{D}_A)$, they seek to train a model, able to solve the second task B when exposed to new data $\mathcal{D}_B$, while remembering task A. Specifically, $p(\boldsymbol{w}|\mathcal{D}_A)$ acts as a prior to $p(\boldsymbol{w}|\mathcal{D})$, such that $p(\boldsymbol{w}|\mathcal{D}_A)$ enforces a regularizer. Consequently, the final weight distribution $p(\boldsymbol{w}|\mathcal{D})$, where $\mathcal{D} = \mathcal{D}_A \cup \mathcal{D}_B$, is learned such as to recognize and be able to solve both task A and B. In other words, learning was transferred from task A to B to formulate a complete model.

In this work, transfer learning in conjunction with Bayesian inference and DNN will be investigated. Thus, a marginalization over the weights $\boldsymbol{w}$ is performed, which in turn requires a posterior distribution - e.g. $p(\boldsymbol{w}|\mathcal{D})$. Drawing inference through a posterior distribution allows for transfer learning. The concept of transfer learning, as defined in this project, will be largely inspired by the main idea from [5]. However, while [5] tries to address catastrophic forgetting, this project will not. The difference between the two approaches is subtle but immensely important. In this project, knowledge between two different tasks will be transferred, but not in order to maintain the ability to solve task A, but rather to aid and/or improve the ability to solve task B. The idea is that task B is assumed to be similar to task A. Thus, it is hypothesized that using $p(\boldsymbol{w}|\mathcal{D}_A)$ as a prior, i.e. $p(\boldsymbol{w}) \to p(\boldsymbol{w}|\mathcal{D}_A)$ when learning to solve task B results in faster convergence and potentially achieving a higher accuracy than using a more standard prior - such as a standard multivariate normal distribution.

Since exact Bayesian inference is generally intractable we shall utilize variational approximation in order to approximate the posterior distributions of the weights. To this end *Bayes by Backprop* [6] is implemented, which provides a principled way of performing variational inference on deep networks.

Finally, due to the approximate distribution often not being expressive enough, i.e. assuming independence between weights, we consider the prospects of employing a more expressive approximate distribution. Specifically, we shall investigate the possibilities of using variational inference with normalizing flows [7].

2. METHODS

Bayesian inference draws inference about the weight distribution $p(\boldsymbol{w}|\mathcal{D})$, where $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ defines a data set. Here $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ and $\boldsymbol{y} \in \mathbb{R}^{N}$ defines known observations and labels respectively, N is the number of observations and D is the number of features defining an observation. Denoting a new observation with a correspondong label we try to predict by $\hat{x}$ and $\hat{y}$ respectively, a conditional predictive distribution of $\hat{y}$ given observation $\hat{x}$ is found by means of marginalization of the weights $p(\hat{y}|\hat{x}, \mathcal{D}) = \int_{\boldsymbol{w}} p(\hat{y}|\hat{x}, \boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})\mathrm{d}\boldsymbol{w}$. However, this integration is intractable for deep neural networks, as e.g. [8] shows, and so following Blundell et al. [6] we employ a variational approximation scheme in order to approximate $p(\boldsymbol{w}|\mathcal{D})$. In general, we can consider the log distribution of any observed data expressed through the following decomposition [9, pp. 462–473],

$$\ln p(\mathcal{D}) = \mathcal{L}(q) + KL(q||p), \tag{1a}$$

$$\mathcal{L}(q) = \int_{\boldsymbol{w}} q(\boldsymbol{w}|\theta) \ln \frac{p(\mathcal{D}, \boldsymbol{w})}{q(\boldsymbol{w}|\theta)} \mathrm{d}\boldsymbol{w}, \tag{1b}$$

$$KL(q||p) = -\int_{\boldsymbol{w}} q(\boldsymbol{w}|\theta) \ln \frac{p(\boldsymbol{w}|\mathcal{D})}{q(\boldsymbol{w}|\theta)} \mathrm{d}\boldsymbol{w}, \tag{1c}$$

where $KL(q||p)$ is the Kullback-Leibler divergence, $\mathcal{L}(q)$ is a lower bound for $\ln p(\mathcal{D})$ and $q(\boldsymbol{w}|\theta)$ is the variational distribution to $p(\boldsymbol{w}|\mathcal{D})$. We can optimize the parameters θ by minimizing Eq. (1c). This minimization drives $q(\boldsymbol{w}|\theta)$ towards $p(\boldsymbol{w}|\mathcal{D})$, and so minimizes the additional information associated with the imposed approximation. Denoting the optimal parameters as θ^*, we write

$$\begin{aligned}\theta^* &= \arg\min_{\theta} KL(q(\boldsymbol{w}|\theta)||p(\boldsymbol{w}|\mathcal{D})), \\ &= \arg\min_{\theta} KL(q(\boldsymbol{w}|\theta)||p(\boldsymbol{w})) - \mathbb{E}_{q(\boldsymbol{w}|\theta)}\left[\ln p(\mathcal{D}|\boldsymbol{w})\right],\end{aligned} \tag{2}$$

where $p(\boldsymbol{w})$ is a fixed prior distribution of the weights and $p(\mathcal{D}|\boldsymbol{w})$ is the likelihood function. Thus, the cost function is defined as,

$$\mathcal{F}(\mathcal{D}, \theta) = KL(q(\boldsymbol{w}|\theta)||p(\boldsymbol{w})) - \mathbb{E}_{q(\boldsymbol{w}|\theta)}\left[\ln p(\mathcal{D}|\boldsymbol{w})\right]. \tag{3}$$

As in [6], we refer to the prior-dependent term and the data-dependent term of Eq. (3) as the complexity cost and likelihood cost respectively.

Due to the general computational infeasibility of performing the analytical integrations of Eq. (3), we shall turn to Monte Carlo approximation,

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \ln q(\boldsymbol{w}^{(i)}|\theta) - \ln p(\boldsymbol{w}^{(i)}) - \ln p(\mathcal{D}|\boldsymbol{w}^{(i)}), \tag{4}$$

here $\boldsymbol{w}^{(i)}$ is the i'th draw from the variational distribution $q(\boldsymbol{w}|\theta)$ and n is the number of draws. The minimization of Eq. (4) by use of backpropagation provides an unbiased estimate of the gradient [6] and is known as *Bayes by Backprop*. Further considering the application of neural networks in this work and thus also minibatch optimization, we may seek to weigh the complexity cost relative to the likelihood cost. Thus, by partitioning $\mathcal{D}$ into M parts, $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M\}$, we define an adjusted cost associated with a minibatch by,

$$\mathcal{F}_i(\mathcal{D}_i, \theta) = \beta_i KL(q(\boldsymbol{w}|\theta)||p(\boldsymbol{w})) - \mathbb{E}_{q(\boldsymbol{w}|\theta)}\left[\ln p(\mathcal{D}_i|\boldsymbol{w})\right],$$

where $i = 1, 2, \ldots, M-1, M$ and we consider $\beta_i = \frac{1}{M}$ or $\beta_i = \frac{2^{M-i}}{2^M-1}$ such that $\sum_{i=1}^{M} \beta_i = 1$. Both weights satisfy $\mathbb{E}_M\left[\sum_{i=1}^{M} \mathcal{F}_i(\mathcal{D}_i, \theta)\right] = \mathcal{F}(\mathcal{D}, \theta)$ as discussed by [6]. We shall later determine which form of β_i is most appropriate in this project.

It should be noted that the minimization of Eq. (2), is subject to the choice of the approximate distribution. Often exact minimization of Eq. (2) cannot be obtained, as $q(\boldsymbol{w}|\theta)$ is not expressive enough such that the true posterior is not in $q(\boldsymbol{w}|\theta)$. An example of a distribution which may lack sufficient complexity is the multivariate normal distribution where weights are assumed independent,

$$q(\boldsymbol{w}|\theta) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{5}$$

with $\boldsymbol{\mu}$ denoting the mean and $\boldsymbol{\Sigma}$ being a diagonal covariance matrix, which are optimized, i.e. $\theta = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$.

2.1. Introducing more complex distributions

One approach to introduce a more complex approximate distribution, would simply be to relax the assumption of independent sets of weights between layers - i.e. assuming dense covariance matrices (for each set of weights between layers),

$$p(\boldsymbol{w}) = \prod_{l=1}^{L} p(\boldsymbol{w}_l), \tag{6}$$

where $\boldsymbol{w}_l$ refers to weights joining layer $l-1$ with layer l for $l = 1, \ldots, L$ with layers L (not counting the input layer). However, optimizing Eq. (6) where $p(\boldsymbol{w}_l)$ are general Gaussian distributions requires $\mathcal{O}(n_l^3)$ computations for each layer, where n_l is the number of weights connecting layer $l-1$ to layer l. We therefore seek a method that introduces covariance and increased complexity, while limiting the number of computations. Normalizing flows [7] proposes to use a smooth invertible mapping $f : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_l}$, where f is constructed such that the determinant of its Jacobian is easily calculable. If we let $\boldsymbol{z}_0 \in \mathbb{R}^{n_l}$ be a stochastic variable, its probability density can be evaluated under the transformation

$f(\boldsymbol{z}_0)$ as

$$\boldsymbol{z}_1 = f(\boldsymbol{z}_0), \tag{7}$$

$$q(\boldsymbol{z}_1) = q(\boldsymbol{z}_0)\left|\det \frac{\partial f}{\partial \boldsymbol{z}_0}\right|^{-1}, \tag{8}$$

where Eq. (7) and Eq. (8) are defined as the flow and normalizing flow respectively. It is straightforward to apply such transformations sequentially.

$$\boldsymbol{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{z}_0), \tag{9}$$

$$\ln q(\boldsymbol{z}_K) = \ln q(\boldsymbol{z}_0) - \sum_{k=1}^{K} \ln \left|\det \frac{\partial f_k}{\partial \boldsymbol{z}_{k-1}}\right|, \tag{10}$$

Each transformation increases the complexity of $q(\boldsymbol{z}_K)$ and allows for a multimodal distribution.

Thus, the flow is an attempt to get closer to the true posterior distribution, i.e. $p(\boldsymbol{w}|\mathcal{D}) \approx q_K$, where $q_K = \prod_{l=1}^{L} q(\boldsymbol{z}_K^l)$ - i.e. a flow is made for each inter-layer set of weights, $\boldsymbol{z}_K^l$.

Additionally, the matrix determinant lemma allows us to construct mappings for which the computation of the determinant in Eq. (10) be done in $\mathcal{O}(n_l)$, and hence also the optimization. This family of transforms take the form

$$f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}h(\boldsymbol{a}^T\boldsymbol{z} + b),$$

where $\boldsymbol{u} \in \mathbb{R}^{n_l}$, $\boldsymbol{a} \in \mathbb{R}^{n_l}$ and b are parameters to be optimized between each layer and h is a smooth and invertible function with derivative h'. Thus the determinant can be computed as

$$\left|\det \frac{\partial f}{\partial \boldsymbol{z}}\right| = |1 + \boldsymbol{u}^T h'(\boldsymbol{a}^T\boldsymbol{z} + b)\boldsymbol{w}|. \tag{11}$$

In order to ensure $f(\boldsymbol{z})$ to be invertible, we shall use what is known as Planar flows [7]. Planar flows are defined as follows,

$$f(\boldsymbol{z}) = \boldsymbol{z} + \hat{\boldsymbol{u}}(\boldsymbol{u}, \boldsymbol{a})h(\boldsymbol{a}^T\boldsymbol{z} + b),$$

where $h(x) = \tanh(x)$ and

$$\hat{\boldsymbol{u}}(\boldsymbol{u}, \boldsymbol{a}) = \boldsymbol{u} + \left[m(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{u}) - \boldsymbol{a}^{\mathrm{T}}\boldsymbol{u}\right]\frac{\boldsymbol{a}}{\|\boldsymbol{a}\|^2},$$

where $m(x) = -1 + \log(1 + e^x)$. Normalizing flows were originally conceived for deep latent variable models, but in this project they are introduced in the context of weights as follows

$$\boldsymbol{z}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{12a}$$

$$\boldsymbol{w} = \boldsymbol{z}_K, \tag{12b}$$

$\boldsymbol{\Sigma}_0$ is again a diagonal covariance matrix, $\boldsymbol{\mu}_0$ is a mean vector (both to be optimized) and $\boldsymbol{z}_0$ is the initial stochastic variable, which is propagated through the flow Eq. (9).

2.2. Transfer learning

When seeking to perform transfer learning, we do so by considering a learned approximate posterior $q(\boldsymbol{w}|\mathcal{D}_A)$ as a prior when training a network on a data set $\mathcal{D}_B$. Note that the approximate distribution learned from data set A, is now explicitly written as a conditional distribution. As a short-hand notation any approximate posterior distribution conditioned on data set A will be denoted $q_A(\boldsymbol{w})$, and should be considered fixed.

In this project transfer learning is defined as follows,

$$p(\boldsymbol{w}|\mathcal{D}_B) \propto p(\mathcal{D}_B|\boldsymbol{w})p(\boldsymbol{w}), \;\; p(\boldsymbol{w}) = q_A(\tilde{\boldsymbol{w}})p(\bar{\boldsymbol{w}}), \tag{13}$$

where independence is assumed between the weights and $\boldsymbol{w} = \{\tilde{\boldsymbol{w}}, \bar{\boldsymbol{w}}\}$ - i.e. normalizing flows is not considered when doing transfer learning as independence cannot be assumed in this case. The partition $p(\boldsymbol{w}) = q_A(\tilde{\boldsymbol{w}})p(\bar{\boldsymbol{w}})$ in Eq. (13) allows complete control of which parts of the network should be shared across two domains. Considering the use of fully connected DNNs, $\tilde{\boldsymbol{w}}$ is considered the weights associated with the input layer through the last hidden layer and $\bar{\boldsymbol{w}}$ is considered associated with the last hidden layer and the output layer. This specific partition reflects the assumption that the initial bulk of the network captures general aspect or features associated with different domains, which could be transferred across these domains. On the other hand, the final connection is allowed to be adjusted to the specific data set. Thus, $\bar{\boldsymbol{w}}$ is considered independent of $\mathcal{D}_A$, and will be assumed to be a more typical prior - e.g. $p(\bar{\boldsymbol{w}}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, where $\boldsymbol{I}$ is the identity matrix.

The benefit of this partition, is that only part of the two models used to solve task A and B needs to be identical. For example, by only transferring the weights distribution of the first few layers, the number of outputs can be made task specific.

In terms of training, the prior distribution when learning the approximate distribution for task A is assumed to be a multivariate normal of the form, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. However, when learning the approximate distribution related to task B, only the weights from the final hidden layer to the output layer is a standard multivariate normal $p(\bar{\boldsymbol{w}}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, and the prior for the remaining weights is the approximate posterior distribution learned from task A, $p(\tilde{\boldsymbol{w}}) = q_A(\tilde{\boldsymbol{w}})$. The approximate distributions for both task A and B are assumed to be a multivarite Gaussian distributions, $q(\tilde{\boldsymbol{w}}_j|\mathcal{D}_j) = \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, with mean $\boldsymbol{\mu}_j$ and diagonal covariance matrix $\boldsymbol{\Sigma}_j$, both of which are optimized. Here index $j = \{\mathrm{A}, \mathrm{B}\}$ denotes the association with either task A or B.

3. EXPERIMENTS

The model architecture considered throughout this project is a DNN with two hidden layers, each with 400 hidden units.

The output layer was defined by whatever task in consideration. Throughout all experiments conducted in this report, optimization was performed using the Adam optimizer [10] with a learning rate of 10^{-5}. Also all integrals were approximated using 10 Monte Carlo samples. Our github repository[1] contains all necessary code needed to recreate our results.

In terms of priors, when not considering transfer learning the prior $p(\boldsymbol{w})$ was a Gaussian distribution such that $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Regardless of considering transfer learning or not the approximate distribution was initialized as $q(\boldsymbol{w}_{\text{init}}|\theta) = \mathcal{N}(\boldsymbol{\mu}_{\text{init}}, \boldsymbol{\Sigma}_{\text{init}})$, where each element in $\boldsymbol{\Sigma}_{\text{init}}$ and $\boldsymbol{\mu}$, denoted Σ_{ij} and μ_i respectively, is drawn from a uniform distribution such that

$$\mu_i \sim \mathcal{U}(0, 1), \tag{14a}$$

$$\log \Sigma_{ij} \sim \mathcal{U}(-\alpha - 8, \alpha - 8), \tag{14b}$$

where $\alpha = \frac{10}{\sqrt{D}}$.

Before evaluating our network on the ability to transfer knowledge, we wanted to investigate which of the previously mentioned values for β_i promised the highest accuracy. To this end, the model was trained and evaluated using the MNIST data set [11]. The data was split into a $60,000$ sized training set and a $10,000$ sized validation set - i.e. all digits (0-9) were considered. The data was standardised by dividing the pixel intensities by 126, similarly to [6]. The approximate distribution had the form Eq. (5), and the aim was to reproduce the results obtained in [6].

Using the optimal found form of β_i, experiments were conducted in order to assess the prospects of introducing more complex approximate distribution by use of normalizing flows. To this end, elements in $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\mu}_0$ from Eq. (12) were also initialized as given in Eq. (14).

Finally, experiments were undertaken to evaluate the transfer abilities of our model. Again, we turned to the MNIST data set which was divided into two disjoint sets of $35,000$ data points each ($30,000$ as training data and $5,000$ validation data) - $\mathcal{D}_A$ which contained the digits (0, 1, 2, 3, 4), and $\mathcal{D}_B$ which contained the digits (5, 6, 7, 8, 9). Thus, learning to differentiate between the digits in each set defines task A and task B. Furthermore, different fractions of the training data set associated with task B was considered. Namely 0.1, 0.5, and 1 were examined to discover any dependencies in this regard. When using a smaller training set the model would be more likely to over-fit and a good prior can help alleviate this issue through regularization. If the prior is close to the true distribution we would expect an introduction of such a prior to significantly help training a model using only small training sets. Hence, assuming the approximate posterior distribution learned from task A indeed could be transferred as a prior to task B, we hoped to see that the model associated with task B either converged faster and/or better than not performing transfer learning. Additionally, this effect (if present) might become less noticeable as the training set increases.

[1] https://github.com/ammunk/bayes_02456

4. RESULTS

Fig. 1 shows the validation accuracy after training three models for 600 epochs on the MNIST data set using different β_i values using the approximate distribution Eq. (5). Specifically we considered $\beta_i = \frac{2^{M-i}}{2^M-1}$, $\beta_i = \frac{1}{M}$ and $\beta_i = 0$ (included as a reference) which would result in maximizing the likelihood function averaged wrt. the approximate distribution. We shall refer to each validation accuracy as $\text{acc}_{\text{Graves}}$, $\text{acc}_{\text{Blundell}}$, acc_{ML} for $\beta_i = \frac{1}{M}$, $\beta_i = \frac{2^{M-i}}{2^M-1}$, $\beta_i = 0$ respectively.

Clearly, we found $\text{acc}_{\text{Blundell}}, \text{acc}_{\text{ML}} > \text{acc}_{\text{Graves}}$. It further appeared as if $\text{acc}_{\text{Blundell}} > \text{acc}_{\text{ML}}$ even though both models seemed to be learning even after 600 epochs and thus could be reaching a higher validation accuracy if allowed further training - but due to [6] only training for 600 epochs we did the same for the purpose of comparison. To this end it should further be noted that the accuracy achieved using $\beta_i = \frac{2^{M-i}}{2^M-1}$ is relatively close to the reported accuracy found in [6], which we denote $\text{acc}_{\text{baseline}} = 98.18\%$. Specifically we reached $\text{acc}_{\text{Blundell}} \approx 96.88\%$.

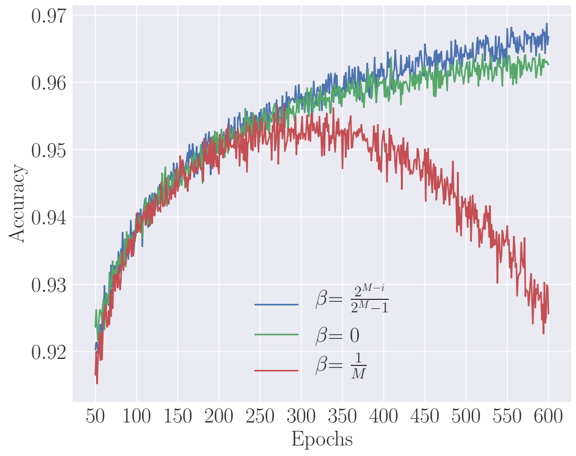

Fig. 1. Validation acccuracies as function of number of epochs. Each line corresponds to a different β_i value. The models are trained and validated using the MNIST data set and we considered all digits (0-9)

Due to these results all subsequent experiments were conducted using $\beta_i = \frac{2^{M-i}}{2^M-1}$.

Next, we turned to the investigation of introducing a more complex approximate posterior distribution (which allows weight correlation and a multi-modal distribution), namely by using normalizing flows. Fig. 2 shows the validation accuracy using $K = 16$ flows of the so-called Planar flows. The data set used was the MNIST and we considered the digits (0-9). As a comparison we used the performance achieved using the approximate distribution of the form Eq. (5) from the previous experiment. Evidently, we did not achieve better results using normalizing flows, and we found that using the otherwise less expressive approximate distribution Eq. (5) performed better i.e. a relatively more consistent increase in validation accuracy is seen when not using normalizing flows, whereas using normalizing flows seemed to be stuck close to a local minimum.

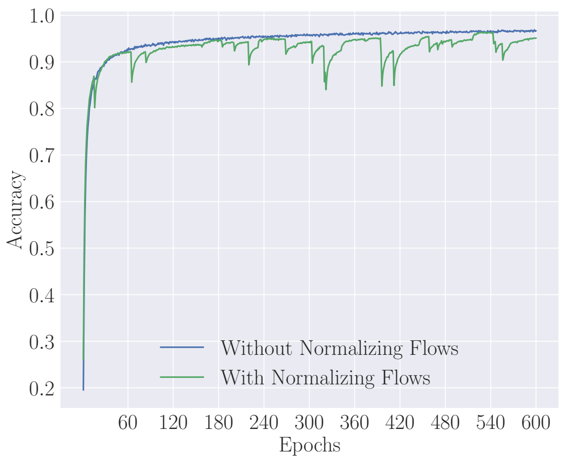

Fig. 2. Validation accuracies as function of epochs of two models trained on the MNIST data set using the digits (0-9). The green and blue lines corresponds to a trained model with and without normalizing flows.

Finally, we considered the prospects of performing transfer learning using Eq. (5) as the approximate posterior due to the previous results. Fig. 3 shows the validation accuracies of performing transfer learning, in which experiments were conducted with the model trained on 10%, 50%, and 100% of the total training data that is associated with task B - i.e. the digits (5-9). As a reference we trained a corresponding model on the same data sets, but without using transfer learning. Evidently, when using our model architecture and definition of task A and task B, transfer learning did not improve either convergence rate (i.e. at least for some significant amount of epochs the transfer learning model outperforms the non-transfer learning model) or final performance (after 600 epochs). In fact across all fractions we found that all models utilizing transfer learning significantly hurt the performance, and we were not able to achieve a validation accuracy $\gtrapprox 66.5\%$.

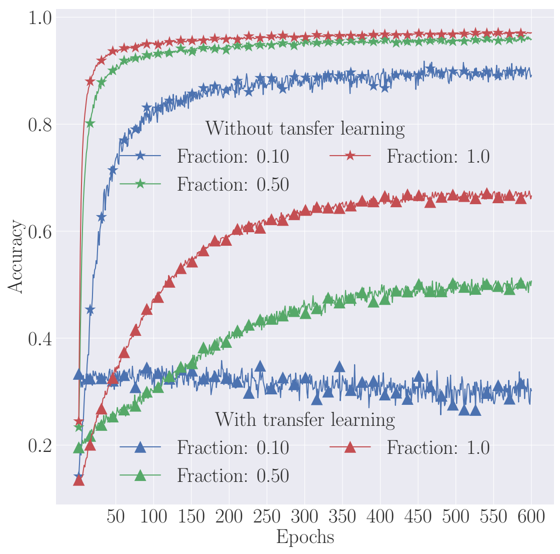

Fig. 3. Validation accuracies (using the digits (5-9) of the MNIST data set) as function of epochs with and without transfer learning using different fractions of the training data. The stars and triangles denotes models trained without and with transfer learning respectively. The line colors are associated with the fraction of the training data used for training.

5. DISCUSSION AND CONCLUSION

Before discarding our proposed method on how to perform transfer leaning, we shall discuss possible reasons which may explain why our results should perhaps only be considered preliminary.

First of all, we did not perform a thorough exploration of the hyperparameters, i.e. learning rate, approximate posterior initialization (Eq. (14), Eq. (12)), the form of the prior, number of flows, number of hidden layers and units etc. Optimally, a cross-validation scheme should have been employed, however due to the significantly long duration spend on training every model ($\sim$ 6 h, $\sim$ 34 h for non-normalizing flows and normalizing flows respectively), true cross-validation was not feasible. Thus, we performed a manual selection of hyperparameters, which meant trying a variety of different parameter settings and training the models for only $\sim$ 50 epochs. The setting with the seemingly most promising validation performance was then picked. Consequently, we cannot ensure our results reflect the maximum potential of performing transfer learning. In fact this may explain why we were not able to quite reach the performance reported in [6], as we saw in Fig. 1. Specifically, [6] does not state how they initialized their approximate posterior, and we found that the performance of using Bayesian inference with variational approximation is highly sensitive to the initialization of the approximate posterior distribution - i.e. Eq. (14).

Secondly, and more specifically why our transfer learning scheme seemed to fail, one should consider the importance of the prior when performing Bayesian inference. While a good prior, which is not too far from the true (unknown) distribution of e.g. weights, can help find an optimal approximate posterior, a bad prior will do the opposite. If the prior is poorly chosen, the available data need to show strong evidence on how the weights ought to be distributed. Thus our results, as seen in Fig. 3 may just reflect the fact, that using the approximate posterior from task A as a prior when solving task B is simply a poor prior, and so we cannot expect good performances. This seems to be supported by the fact that when a larger fraction of the training data is used, we are provided with stronger data-driven evidence on how to distribute the weights. This would in turn provide a better performance, as seen.

Why then, is using the posterior as a prior a bad choice? First of all, the models considered have been simple fully connected deep networks. Our assumption that part of the networks captures features shared by both task A and B may simply be wrong for these features. Instead, one could consider using convolutional layers, which are known to capture general features in images (such as edges and color-opponency) [12]. In the context of transfer learning, where the data consists of images (like MNIST), it would seem more appropriate that such features are shared. In other words, in order for transfer learning to work, where a posterior distribution of weights are used as a prior, the specific architecture of the used models need to be taken into consideration.

As a final remark regarding as to why transfer learning appeared to be unsuccessful, one should consider if perhaps our definitions of task A and task B were too different. As an analogy, maybe what have been done in this project corresponds to trying to learn how to play soccer using ones ability to play chess.

For these reasons we do not believe transfer learning, where a posterior distribution is used as a prior, should be discarded. Rather, the results achieved in this project hints at potential shortcomings and thus could be used to narrow down further investigations.

In conclusion, transfer learning was not successful given our architecture, choice of hyperparameters and definition of task A and B. Nonetheless, further investigation is required in order to generally deem transfer learning, where a posterior distribution is used as a prior, ineffective or not.

6. REFERENCES

[1] Matthias Bethge Leon A. Gatys, Alexander S. Ecker, "A neural algorithm of artistic style," *arXiv*, 2015.

[2] Case C. Casper J. Catanzaro B. Diamos G. Elsen E. ... & Ng A. Y Hannun, A., "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint*, 2014.

[3] RM French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[4] Rich Caruana, "Multitask learning," in *Learning to learn*, pp. 95–133. Springer, 1998.

[5] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell, "Overcoming catastrophic forgetting in neural networks [arxiv]," *arXiv*, pp. 13 pp., 13 pp., 2016.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, Daan Wierstra, and Google Deepmind, "Weight uncertainty in neural networks," 2016.

[7] Danilo Jimenez Rezende and Shakir Mohamed, "Variational inference with normalizing flows," *arXiv preprint*, 2015.

[8] Yarin Gal, *Uncertainty in deep learning*, Ph.D. thesis, PhD thesis, University of Cambridge, 2016.

[9] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2009.

[10] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, 2014.

[11] Bottou L. Bengio Y. LeCun, Y. and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE, 86*, p. 22782324, 1998.

[12] Matthew D. Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8689, no. 1, pp. 818–833, 2014.