Supplementary Material for Bayesian Nonparametric Federated Learning of Neural Networks

Mikhail Yurochkin [1 2] **Mayank Agarwal** [1 2] **Soumya Ghosh** [1 2 3] **Kristjan Greenewald** [1 2] **Trong Nghia Hoang** [1 2] **Yasaman Khazaeni** [1 2]

1. Single Hidden Layer Inference

The goal of maximum a posteriori (MAP) estimation is to maximize posterior probability of the latent variables: global atoms $\{\theta_i\}_{i=1}^{\infty}$ and assignments of observed neural network weight estimates to global atoms $\{\boldsymbol{B}^j\}_{j=1}^{J}$, given estimates of the batch weights $\{\mathbf{v}_{jl} \text{ for } l = 1, \dots, L_j\}_{j=1}^{J}$:

$$\underset{\{\theta_i\},\{\boldsymbol{B}_j\}}{\arg\max}\, P(\{\theta_i\},\{\boldsymbol{B}^j\}|\{\mathbf{v}_{jl}\}) \propto P(\{\mathbf{v}_{jl}\}|\{\theta_i\},\{\boldsymbol{B}^j\})P(\{\boldsymbol{B}^j\})P(\{\theta_i\}). \tag{1}$$

MAP estimates given matching. First we note that given $\{\boldsymbol{B}^j\}$ it is straightforward to find MAP estimates of $\{\theta_i\}$ based on Gaussian-Gaussian conjugacy:

$$\hat{\theta}_i = \frac{\sum_{j,l} B^j_{i,l}\mathbf{v}_{jl}/\sigma_j^2}{1/\sigma_0^2 + \sum_{j,l} B^j_{i,l}/\sigma_j^2} \text{ for } i = 1, \dots, L, \tag{2}$$

where $L = \max\{i : B^j_{i,l} = 1 \text{ for } l = 1, \dots, L_j,\ j = 1, \dots, J\}$ is the number of active global atoms, which is an (unknown) latent random variable identified by $\{\boldsymbol{B}^j\}$. For simplicity we assume $\boldsymbol{\Sigma}_0 = \boldsymbol{I}\sigma_0^2$, $\boldsymbol{\Sigma}_j = \boldsymbol{I}\sigma_j^2$ and $\boldsymbol{\mu}_0 = 0$.

Inference of atom assignments (Proposition 2 of the main text). We can now cast optimization corresponding to (1) with respect to only $\{\boldsymbol{B}^j\}_{j=1}^{J}$. Taking natural logarithm we obtain:

$$-\frac{1}{2}\sum_i \left(\frac{\|\hat{\theta}_i\|^2}{\sigma_0^2} + D\log(2\pi\sigma_0^2) + \sum_{j,l} B^j_{i,l}\frac{\|\mathbf{v}_{jl} - \hat{\theta}_i\|^2}{\sigma_j^2} \right) + \log(P(\{\boldsymbol{B}^j\}). \tag{3}$$

We now simplify the first term of (3) (in this and subsequent derivations we use $\cong$ to say that two objective functions are equivalent up to terms independent of the variables of interest):

$$\begin{aligned}
&-\frac{1}{2}\sum_i \left(\frac{\|\hat{\theta}_i\|^2}{\sigma_0^2} + D\log(2\pi\sigma_0^2) + \sum_{j,l} B^j_{i,l}\frac{\|\mathbf{v}_{jl} - \hat{\theta}_i\|^2}{\sigma_j^2} \right) \\
&= -\frac{1}{2}\sum_i \left(\frac{\langle\hat{\theta}_i,\hat{\theta}_i\rangle}{\sigma_0^2} + D\log(2\pi\sigma_0^2) + \sum_{j,l} B^j_{i,l}\frac{\langle\mathbf{v}_{jl},\mathbf{v}_{jl}\rangle - 2\langle\mathbf{v}_{jl},\hat{\theta}_i\rangle + \langle\hat{\theta}_i,\hat{\theta}_i\rangle}{\sigma_{jl}^2} \right) \\
&\cong -\frac{1}{2}\sum_i \left(\langle\hat{\theta}_i,\hat{\theta}_i\rangle \left(\frac{1}{\sigma_0^2} + \sum_{j,l}\frac{B^j_{i,l}}{\sigma_j^2}\right) + D\log(2\pi\sigma_0^2) - 2\langle\hat{\theta}_i, \sum_{j,l} B^j_{i,l}\frac{\mathbf{v}_{jl}}{\sigma_j^2}\rangle \right) \\
&= \frac{1}{2}\sum_i \left(\langle\hat{\theta}_i,\hat{\theta}_i\rangle \left(\frac{1}{\sigma_0^2} + \sum_{j,l}\frac{B^j_{i,l}}{\sigma_j^2}\right) - D\log(2\pi\sigma_0^2) \right) \\
&= \frac{1}{2}\sum_i \left(\frac{\|\sum_{j,l} B^j_{i,l}\mathbf{v}_{jl}/\sigma_j^2\|^2}{1/\sigma_0^2 + \sum_{j,l} B^j_{i,l}/\sigma_j^2} - D\log(2\pi\sigma_0^2) \right).
\end{aligned} \tag{4}$$

We consider an iterative optimization approach: fixing all but one $\boldsymbol{B}^j$ we find the corresponding optimal assignment, then pick a new j at random and repeat until convergence. We define notation $-j$ to denote "all but j", and let $L_{-j} = \max\{i : B^{-j}_{i,l} = 1\}$ denote number of active global weights outside of group j. We partition (4) between $i = 1, \dots, L_{-j}$ and $i = L_{-j} + 1, \dots, L_{-j} + L_j$, and since we are solving

*Equal contribution [1]IBM Research, Cambridge [2]MIT-IBM Watson AI Lab [3]Center for Computational Health. Correspondence to: Mikhail Yurochkin <mikhail.yurochkin@ibm.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

for $\boldsymbol{B}^j$, we subtract terms independent of $\boldsymbol{B}^j$:

$$\sum_i \left(\frac{\| \sum_{j,l} B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + \sum_{j,l} B^j_{i,l}/\sigma_j^2} - D \log(2\pi\sigma_0^2) \right)$$
$$\cong \sum_{i=1}^{L_{-j}} \left(\frac{\| \sum_l B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 + \sum_{-j,l} B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + \sum_l B^j_{i,l}/\sigma_j^2 + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} - \frac{\| \sum_{-j,l} B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} \right)$$
$$+ \sum_{i=L_{-j}+1}^{L_{-j}+L_j} \left(\frac{\| \sum_l B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + \sum_l B^j_{i,l}/\sigma_j^2} \right). \tag{5}$$

Now observe that $\sum_l B^j_{i,l} \in \{0, 1\}$, i.e. it is 1 if some neuron from batch j is matched to global neuron i and 0 otherwise. Due to this we can rewrite (5) as a linear sum assignment problem:

$$\sum_{i=1}^{L_{-j}} \sum_{l=1}^{L_j} B^j_{i,l} \left(\frac{\|\mathbf{v}_{jl}/\sigma_j^2 + \sum_{-j,l} B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + 1/\sigma_j^2 + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} - \frac{\| \sum_{-j,l} B^j_{i,l} \mathbf{v}_{jl}/\sigma_j^2 \|^2}{1/\sigma_0^2 + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} \right)$$
$$+ \sum_{i=L_{-j}+1}^{L_{-j}+L_j} \sum_{l=1}^{L_j} B^j_{i,l} \left(\frac{\|\mathbf{v}_{jl}/\sigma_j^2\|^2}{1/\sigma_0^2 + 1/\sigma_j^2} \right). \tag{6}$$

Now we consider second term of (3):

$$\log P(\{\boldsymbol{B}^j\}) = \log P(\boldsymbol{B}^j | \boldsymbol{B}^{-j}) + \log P(\boldsymbol{B}^{-j}).$$

First, because we are optimizing for $\boldsymbol{B}^j$, we can ignore $\log P(\boldsymbol{B}^{-j})$. Second, due to exchangeability of batches (i.e. customers of the IBP), we can always consider $\boldsymbol{B}^j$ to be the last batch (i.e. last customer of the IBP). Let $m_i^{-j} = \sum_{-j,l} B^j_{i,l}$ denote number of times batch weights were assigned to global atom i outside of group j. We now obtain the following:

$$\log P(\boldsymbol{B}^j | \boldsymbol{B}^{-j}) \cong$$
$$\sum_{i=1}^{L_{-j}} \left(\left(\sum_{l=1}^{L_j} B^j_{i,l} \right) \log \frac{m_i^{-j}}{J} + \left(1 - \sum_{l=1}^{L_j} B^j_{i,l} \right) \log \frac{J - m_i^{-j}}{J} \right)$$
$$- \log \left(\sum_{i=L_{-j}+1}^{L_{-j}+L_j} \sum_{l=1}^{L_j} B^j_{i,l} \right) + \left(\sum_{i=L_{-j}+1}^{L_{-j}+L_j} \sum_{l=1}^{L_j} B^j_{i,l} \right) \log \frac{\gamma_0}{J}. \tag{7}$$

We now rearrange (7) as linear sum assignment problem:

$$\sum_{i=1}^{L_{-j}} \sum_{l=1}^{L_j} B^j_{i,l} \log \frac{m_i^{-j}}{J - m_i^{-j}} \tag{8}$$
$$+ \sum_{i=L_{-j}+1}^{L_{-j}+L_j} \sum_{l=1}^{L_j} B^j_{i,l} \left(\log \frac{\gamma_0}{J} - \log(i - L_{-j}) \right).$$

Combining (6) and (8) we arrive at the cost specification for finding $\boldsymbol{B}^j$ as *minimizer* of $\sum_i \sum_l B^j_{i,l} C^j_{i,l}$, where:

$$C^j_{i,l} = \tag{9}$$
$$- \begin{cases} \frac{\left\| \frac{\mathbf{v}_{jl}}{\sigma_j^2} + \sum_{-j,l} B^j_{i,l} \frac{\mathbf{v}_{jl}}{\sigma_j^2} \right\|^2}{\frac{1}{\sigma_0^2} + \frac{1}{\sigma_j^2} + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} - \frac{\left\| \sum_{-j,l} B^j_{i,l} \frac{\mathbf{v}_{jl}}{\sigma_j^2} \right\|^2}{\frac{1}{\sigma_0^2} + \sum_{-j,l} B^j_{i,l}/\sigma_j^2} & \\ \qquad + 2 \log \frac{m_i^{-j}}{J - m_i^{-j}}, & i \le L_{-j} \\ \frac{\left\| \frac{\mathbf{v}_{jl}}{\sigma_j^2} \right\|^2}{\frac{1}{\sigma_0^2} + \frac{1}{\sigma_j^2}} - 2 \log \frac{i - L_{-j}}{\gamma_0 / J}, & L_{-j} < i \le L_{-j} + L_j. \end{cases}$$

This completes the proof of Proposition 2 in the main text.

2. Multilayer Inference Details

Figure 1 illustrates the overall multilayer inference procedure visually, and Algorithm 1 provides the details.

Algorithm 1 Multilayer PFNM

1: $L^{C+1} \leftarrow$ number of outputs
2: # Top down iteration through layers
3: **for** layers $c = C, C-1, \ldots, 2$ **do**
4: Collect hidden layer c from the J batches and form $\mathbf{v}^c_{jl}$.
5: Call Single Layer Neural Matching algorithm with output dimension L^{c+1} and input dimension 0 (since we do not use the weights connecting to lower layers here).
6: Form global neuron layer c from output of the single layer matching.
7: $L^c \leftarrow \text{card}(\cup_{j=1}^J \mathcal{T}^c_j)$ (greedy approach).
8: **end for**
9: # Match bottom layer using weights connecting to both the input and the layer above.
10: Call Single Layer Neural Matching algorithm with output dimension L^2 and input dimension equal to the number of inputs.
11: Return global assignments and form global mutltilayer model.

3. Complexity Analysis

In this section we present a brief discussion of the complexity of our algorithms. The worst case complexity per layer is achieved when no neurons are matched and is equal to $\mathcal{O}(D(JL_j)^2)$ for building the cost matrix and $\mathcal{O}((JL_j)^3)$ for running the Hungarian algorithm, where L_j is the number of neurons per batch (here for simplicity we assume that each batch has same number of neurons) and J is the number of batches. The best case complexity per layer (i.e.

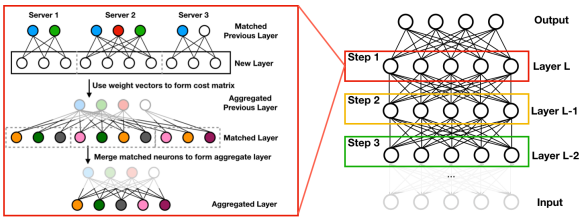


Figure 1: Probabilistic Federated Neural Matching algorithm showing matching of three multilayer MLPs. Nodes in the graphs indicate neurons, neurons of the same color have been matched. On the left, the individual layer matching approach is shown, consisting of using the matching assignments of the next highest layer to convert the neurons in each of the J batches to weight vectors referencing the global previous layer. These weight vectors are then used to form a cost matrix, which the Hungarian algorithm then uses to do the matching. Finally, the matched neurons are then aggregated and averaged to form the new layer of the global model. As shown on the right, in the multilayer setting the resulting global layer is then used to match the next lower layer, etc. until the bottom hidden layer is reached (Steps 1, 2, 3,... in order).

when all neurons are matched) is $\mathcal{O}(DL_j^2 + L_j^3)$, also note that complexity is independent of the data size. In practice the complexity is closer to the best case since global model size is moderate (i.e. $L \ll \sum_j L_j$). Actual timings with our code for the experiments in the main text are as follows - 40sec for Fig. 2a,b at $J = 30$ groups; 500sec for c,d at $J = 30$ (the DL_j^2 term is dominating as CIFAR10 dimension is much higher than MNIST); 60sec for e,f ($J = 10$) at $C = 6$ layers; 150sec for g,h ($J = 10$) at $C = 6$. The computations were done using 2 CPU cores and 4GB memory on a machine with 3.0 GHz core speed. We note that (i) this computation only needs to be performed once (ii) the cost matrix construction which appears to be dominating can be trivially sped up using GPUs (iii) recent work demonstrates impressive large scale running times for the Hungarian algorithm using GPUs (Date & Nagi, 2016).

4. Experimental Details and Additional Results

Data partitioning. In the federated learning setup, we analyze data from multiple sources, which we call batches. Data on the batches does not overlap and may have different distributions. To simulate federated learning scenario we consider two partition strategies of MNIST and CIFAR-10. For each pair of partition strategy and dataset we run 10 trials to obtain mean accuracies and standard deviations. The easier case is homogeneous partitioning, i.e. when class distributions on batches are approximately equal as well as batch sizes. To generate homogeneous partitioning with J batches we split examples for each of the classes into J approximately equal parts to form J batches. In the heterogeneous case, batches are allowed to have highly imbalanced class distributions as well as highly variable sizes. To simulate heterogeneous partition, for each class k, we sample $\mathbf{p}_k \sim \text{Dir}_J(0.5)$ and allocate $\mathbf{p}_{k,j}$ proportion of instances of class k of the complete dataset to batch j. Note that due to small concentration parameter, 0.5, of the Dirichlet distribution, some batches may entirely miss examples of a subset of classes.

Batch networks training. Our modeling framework and ensemble related methods operate on collection of weights of neural networks from all batches. Any optimization procedure and software can be used locally on batches for training neural networks. We used PyTorch (Paszke et al., 2017) to implement the networks and train these using the AMSGrad optimizer (Reddi et al., 2018) with default parameters unless otherwise specified. For reproducibility we summarize all parameter settings in Table 1.

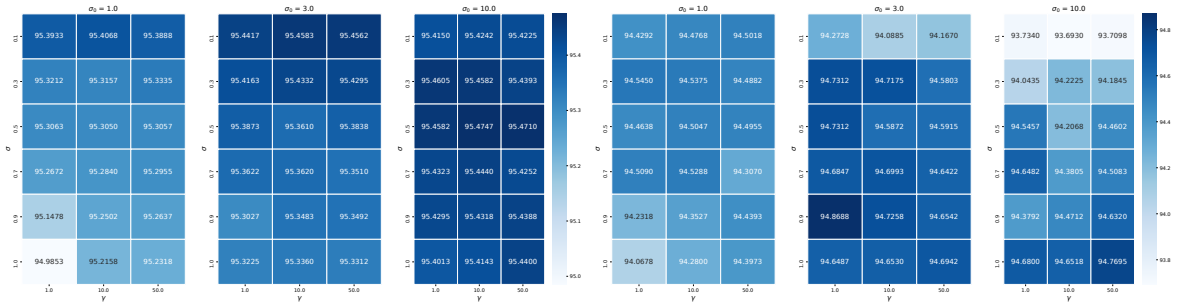
Table 1: Parameter settings for batch neural networks training

	MNIST	CIFAR-10
Neurons per layer	100	100
Learning rate	0.01	0.001
L_2 regularization	10^{-6}	10^{-5}
Minibatch size	32	32
Epochs	10	10
Weights initialization	$\mathcal{N}(0, 0.01)$	$\mathcal{N}(0, 0.01)$
Bias initialization	0.1	0.1

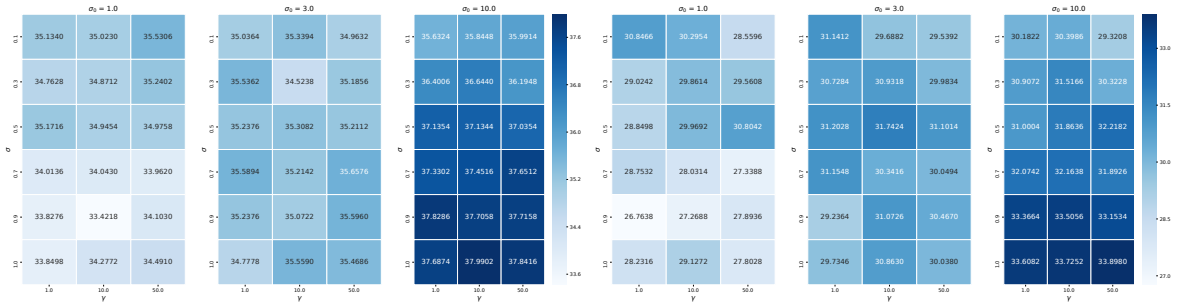
4.1. Parameter Settings for the Baselines

We first formally define the ensemble procedure. Let $\hat{y}_j \in \Delta^{K-1}$ denote the probability distribution over the K classes output by neural network trained on data from batch j for some test input x. Then ensemble prediction is $\arg\max_k \frac{1}{J} \sum_{j=1}^{J} \hat{y}_{j,k}$. In our experiments, we train each individual network on the specific batch dataset using the parameters listed in Table 1, and then compute the performance using the ensemble aggregation technique.

For the downpour SGD (Dean et al., 2012) we used PyTorch, SGD optimizer and parameter settings as in Table 1 for the local learners. The master neural network was optimized with Adam and the same initial learning rate as in the Table 1. The local learners communicated the accumulated gradients back to the master network after every mini-batch update. This translates to the setting of Dean et al. (2012)

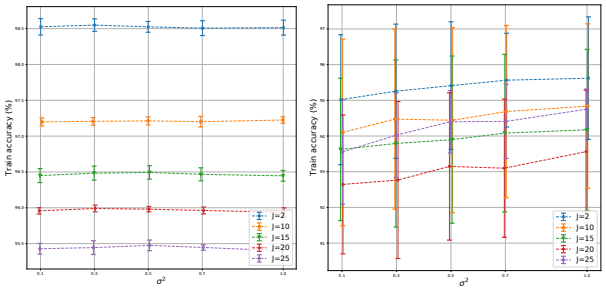


(a) MNIST homogeneous (b) MNIST heterogeneous

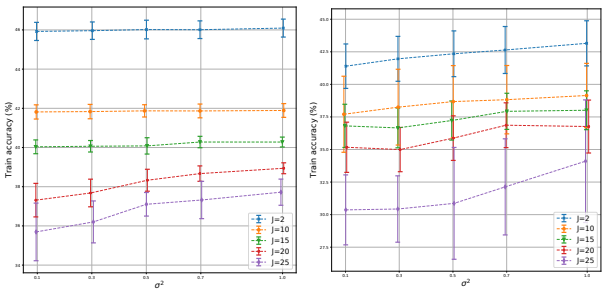


(c) CIFAR homogeneous (d) CIFAR heterogeneous

Figure 2: Parameter sensitivity analysis for $J = 25$.



(a) MNIST homogeneous (b) MNIST heterogeneous



(c) CIFAR homogeneous (d) CIFAR heterogeneous

Figure 3: Sensitivity analysis of σ^2 for fixed $\sigma_0^2 = 10$ and $\gamma_0 = 1$ for varying J.

with parameters $n_{push} = n_{fetch} = 1$. Note that with this approach the global network and networks for each of the batches are constrained to have identical number of neurons per layer, which is 100 in our experiments.

For Federated Averaging (McMahan et al., 2017), we use SGD optimizer for learning the local networks with the rest of the parameters as defined in Table 1. We initialize all the local networks with the same seed, and train these networks for 10 epochs initially and for 5 epochs after the first communication round. At each communication round, we utilize all the local networks ($C = 1$) for the central model update.

4.2. Parameter Settings for Matching with Additional Communications

For neural matching with additional communications, we train the local networks for 10 epochs for the first communication round, and 5 epochs thereafter. All the other parameters are as mentioned in Table 1. The local networks are trained using AMSGrad optimizer (Reddi et al., 2018), and the optimizer parameters are reset after every communication. We also found it useful to decay the initial learning rate by a factor of 0.99 after every communication.

4.3. Parameter Sensitivity Analysis for PFNM

Our models presented in Section 3 of the main text have three parameters σ_0^2, γ_0 and $\sigma^2 = \sigma_1^2 = \ldots = \sigma_J^2$. The first parameter, σ_0^2, is the prior variance of weights of the global neural network. Second parameter, γ_0, controls discovery of new neurons and correspondingly increasing γ_0

increases the size of the learned global network. The third parameter, σ^2, is the variance of the local neural network weights around corresponding global network weights. We empirically analyze the effect of these parameters on the accuracy for single hidden layer model with $J = 25$ batches in Figure 2. The heatmap indicates the accuracy on the training data - we see that for all parameter values considered performance doesn't not fluctuate significantly. PFNM appears to be robust to choices of σ_0^2 and γ_0, which we set to 10 and 1 respectively in the experiments with single communication round. Parameter σ^2 has slightly higher impact on the performance and we set it using training data during experiments. To quantify importance of σ^2 for fixed $\sigma_0^2 = 10$ and $\gamma_0 = 1$ we plot average train data accuracies for varying σ^2 in Figure 3. We see that for homogeneous partitioning and one hidden layer σ^2 has almost no effect on the performance (Fig. 3a and Fig. 3c). In the case of heterogeneous partitioning (Fig. 3b and Fig. 3d), effect of σ^2 is more noticeable, however all considered values result in competitive performance.

References

Date, K. and Nagi, R. Gpu-accelerated hungarian algorithms for the linear assignment problem. *Parallel Computing*, 57:52–72, 2016.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ryQu7f-RZ`.