An Empirical Study of Neural Kernel Bandits

Michal Lisicki
University of Guelph
Vector Institute for AI
`mlisicki@uoguelph.ca`

Arash Afkanpour
Google
`arashaf@google.com`

Graham W. Taylor
University of Guelph
Vector Institute for AI
`gwtaylor@uoguelph.ca`

Abstract

Neural bandits have enabled practitioners to operate efficiently on problems with non-linear reward functions. While in general contextual bandits commonly utilize Gaussian process (GP) predictive distributions for decision making, the most successful neural variants use only the last layer parameters in the derivation. Research on neural kernels (NK) has recently established a correspondence between deep networks and GPs that takes into account all the parameters of a NN and can be trained more efficiently than most Bayesian NNs. We propose to directly apply NK-induced distributions to guide an upper confidence bound or Thompson sampling-based policy. We show that NK bandits achieve state-of-the-art performance on highly non-linear structured data. Furthermore, we analyze practical considerations such as training frequency and model partitioning. We believe our work will help better understand the impact of utilizing NKs in applied settings.

1 Introduction

Contextual bandit algorithms, like upper confidence bound (UCB) [5] or Thompson sampling (TS) [25], balance exploration and exploitation with two terms: (1) a frequentist reward estimate and (2) an uncertainty term [15]. From a Bayesian perspective those terms represent the first two moments of a predictive posterior distribution. If little prior knowledge is available about the problem, this distribution can be chosen to be a Gaussian process (GP). GP-UCB [24] and GP-TS [7] are special cases of kernel bandits [26], in which the ridge regularizer is set to Gaussian noise to form a GP. In practical applications bandits are traditionally combined with well-known kernels, like linear, RBF or Matérn [22]. More recently a new family of neural kernels (NK) has emerged, which directly approximates the behavior of deep neural networks (NN) [6, 10]. The neural network Gaussian process (NNGP) kernel is derived from a forward pass of a NN, and corresponds to a random NN at initialization when combined with a GP [16]. The finding of a direct GP correspondence was later followed by establishing a similar relation for random linearized NNs and the neural tangent kernel (NTK), where NTK represents a network's dynamics during gradient descent training [3, 13, 14, 17]. A review of all recent NK-GP models, with connections to other existing uncertainty models, like deep ensembles and randomized priors [21], can be found in [13]. NeuralUCB [29] and NeuralTS [28] are two related neural bandit policies that can be considered parameter space variants of NK bandits with a NTK-GP posterior [13], whose moments are defined by a standard NN estimate and the neural tangent features (NTF) covariance (Sec. B.5). These algorithms were shown to achieve a regret bound comparable to kernel bandits with the effective dimension induced by a corresponding NTK [29, 28]. The results presented in [29, 28], and later in [27, 19], however, show the algorithm to underperform in practical applications, which was linked to overparametrization and poor covariance approximation [27].

In this work we focus on the conditions in which NK bandits operating in the function space provide a competitive advantage over other neural bandit approaches. Even though NKs have been shown to lack the full representational power of the corresponding NNs, they outperform finite fully-connected networks in small data regimes [4], and combined with GPs, successfully solve simple reinforcement learning tasks [12]. NK-induced GPs provide a fully probabilistic treatment of infinite NNs, and therefore result in more accurate predictive distributions than those used in most of the state-of-the-art neural bandit models. We expected the NK approaches to outperform NNs in certain practical applications and the main purpose of this work is to uncover the characteristics of bandit problems that provide the necessary conditions. We present empirical assessment and practical considerations for NK bandits and show that NKs' advantages result in improved performance on tasks that require complex learned represenations and accurate exploration. We consider a variant of a kernel bandit specialized to the randomized prior approach represented through a NK-GP [13], and focus primarily on empirical analysis using the framework established by [23] and [19] [1].

2 Neural kernel bandits

Our algorithm (Alg. 1) is built using a similar structure to classic kernel and GP bandit approaches [7, 24, 26]. Instead of performing standard GP inference, we compute GP moments $\mu_{a,t}$ and $\sigma_{a,t}$ according to the randomized prior NK approach [13, 21], which was found to work best in practice through our preliminary experiments. We provide both UCB and TS variants of our algorithm, which use the moments or the predictive distribution sampling respectively to provide reward estimates for each arm. In each round an arm is chosen based on the highest estimated reward. We use the disjoint approach as our primary model, which means that the data $(\mathbf{X}_a, \mathbf{y}_a)$ and kernels (NNGP $\boldsymbol{\mathcal{K}}_{a,t}$ and NTK $\mathbf{\Theta}_{a,t}$) are collected and computed separately for each arm a. The disjoint strategy preserves memory, while ensuring that better performing arms, chosen more frequently, have larger associated datasets and thus more certainty over time. This approach is very much in line with classic optimism in face of uncertainty (OFU) methods. A common alternative is the joint model, which, in the case of GP, uses a single kernel and produces separate posteriors, albeit with a common uncertainty estimate, for each arm at prediction time. Using NKs in this manner for multi-class problems is common in the NNGP/NTK literature (e.g. [4, 20]. However, the approach is not feasible in the bandit setting as it assumes access to rewards associated with all the arms. A common solution (e.g. [29]) is to use zero-padding to create a separate context vector for each arm $\mathbf{x}_a = [\mathbf{0}, \ldots, \mathbf{0}, \mathbf{x}_{orig}, \mathbf{0}, \ldots, \mathbf{0}]$. In the parameter space view this has an effect of splitting the model's parameters per arm yielding separate uncertainty estimates. For further details and background please refer to Sec. B.

Algorithm 1 Neural kernel bandit

```
Require: Number of arms $k$, number of rounds $T$, kernel regularization parameter $\gamma$, exploration
  parameter $\eta$, NNGP kernel function $\mathcal{K}(\cdot,\cdot)$, neural tangent kernel function $\Theta(\cdot,\cdot)$, initial number
  of steps $\iota$, policy $\pi$

  Play each arm sequentially for $\iota$ steps to accumulate data $\mathbf{X}_a \in \mathbb{R}^{\iota \times d}, \mathbf{y}_a \in \mathbb{R}^{\iota}\ \forall a$
  for round $t = 1, 2, ..., T$ do
      Observe context $\mathbf{x}_t$
      for arm $a = 1, 2, ..., k$ do
          $\boldsymbol{\mathcal{K}}_{a,t} \leftarrow \mathcal{K}(\mathbf{X}_a, \mathbf{X}_a)$
          $\mathbf{\Theta}_{a,t} \leftarrow \Theta(\mathbf{X}_a, \mathbf{X}_a) + \gamma \mathbf{I}$
          // Calculate predictive distribution moments for all arms
          $\mu_{a,t} \leftarrow \Theta(\mathbf{x}_t, \mathbf{X}_a)\mathbf{\Theta}_{a,t}^{-1}\mathbf{y}_a$
          $\sigma_{a,t} \leftarrow \sqrt{\mathcal{K}(\mathbf{x}_t,\mathbf{x}_t)+\Theta(\mathbf{x}_t,\mathbf{X}_a)\mathbf{\Theta}_{a,t}^{-1}\boldsymbol{\mathcal{K}}_{a,t}\mathbf{\Theta}_{a,t}^{-1}\Theta(\mathbf{X}_a,\mathbf{x}_t)-2\Theta(\mathbf{x}_t,\mathbf{X}_a)\mathbf{\Theta}_{a,t}^{-1}\mathcal{K}(\mathbf{X}_a,\mathbf{x}_t)}$
          if $\pi$ is UCB then
              $p_{a,t} \leftarrow \mu_{a,t} + \frac{\eta}{\gamma^{1/2}}\sigma_{a,t}$
          else if $\pi$ is TS then
              $p_{a,t} \sim \mathcal{N}(\mu_{a,t}, \frac{\eta}{\gamma^{1/2}}\sigma_{a,t})$
          end if
      end for
      Choose $a_t \leftarrow \arg\max_a p_{a,t}$ and obtain reward $y_t$
      Update $(\mathbf{X}_{a_t}, \mathbf{y}_{a_t})$ with $(\mathbf{x}_t, y_t)$
  end for
```

[1] We make the code available at https://github.com/mlisicki/NeuralKernelBandits

3 Empirical Results and Discussion

While NK bandits were conceptually proposed in the past [29, 28], they are not yet considered a reliable method for sequential decision making. We conduct experiments to show that NKs provide a competitive approach in bandit settings. We begin by evaluating our method within a Bayesian bandit framework [23] and present our main result w.r.t. performance of related approaches. We commit the subsequent subsections to measure the implications of practical implementation considerations.

3.1 NK bandits outperform neural-linear and NTF bandits on complex datasets

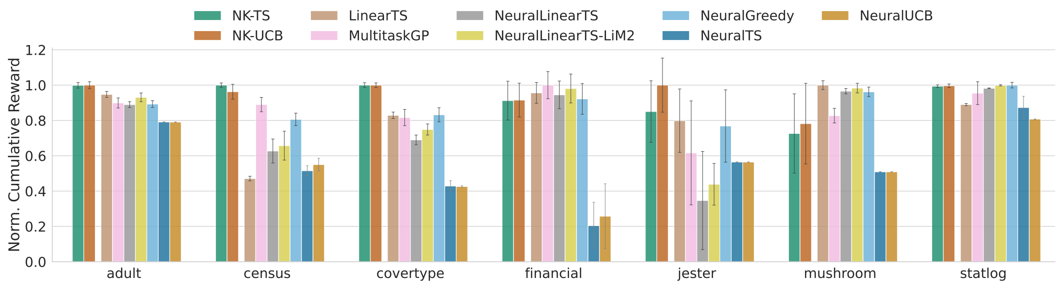

Figure 1: Final normalized cumulative rewards (y-axis) of NK bandits compared against all other methods after 5000 steps rollout on the respective UCI datasets (x-axis). Each bar shows the mean and standard deviation over 10 rollouts.

We tested the performance of NK bandits on a set of UCI datasets [11] commonly used in the bandits literature [23]. The details of our experimental setup can be found in Sec. A.2. NK bandits tend to achieve higher performance for higher-dimensional (over 50 features) non-linear datasets that correspond to classification tasks (Fig. 1). We attribute the increase in performance to the high accuracy of exploitation, achieved through the application of NTK to non-linear structured data problems [4], and exploration, achieved through the utilization of GP predictive distribution linked directly to the behavior of a deep neural network [17].

3.2 Training frequency

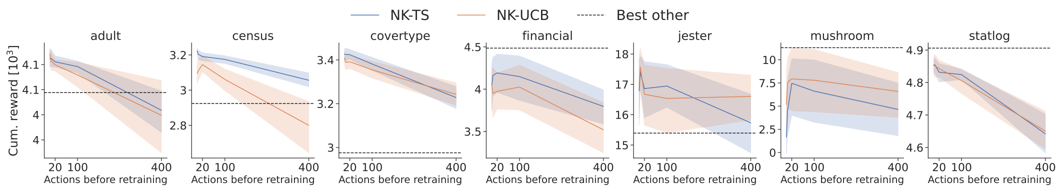

Figure 2: Performance w.r.t. $\{1, 5, 20, 100, 400\}$ actions before retraining.

Frequent updates of complex models in bandit settings put high demand on compute resources. We measure the sensitivity of our method to the number of actions taken between subsequent model updates. We ran our NK bandit for numbers of actions specified in the recent literature [19, 23]: $\{1, 5, 20, 100, 400\}$, and report their respective cumulative rewards on all the datasets after 5000 steps. Fig. 2 shows a smooth degradation in performance as the number of actions increases, which is expected. We note, however, that the degradation is relatively small and preserves competitive performance on all tested datasets. As large numbers of actions help to save significant amounts of computational resources, we believe that the sustained performance provides a strong argument for applicability to real world scenarios. It also positions our method as a good candidate for a limited memory approach, which we leave for future work. For some datasets, we observe a significant spike in performance at 5 actions before retraining. The result suggests an additional role the training frequency plays in the exploration-exploitation trade-off. In Fig. 2 we also show the results of the best performing models, other than our method, as reported in Fig. 1. We observe that for some datasets of higher complexity (e.g. covertype), our method's rank w.r.t. other algorithms is not affected, even with large numbers of actions before retraining, which additionally confirms its robustness.

3.3 Joint model

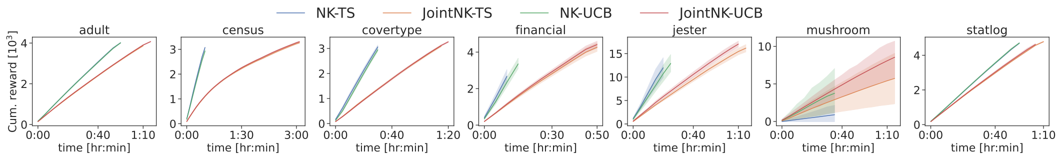

Figure 3: Cumulative rewards w.r.t. running times of a joint and a disjoint variant of our method.

Following the zero-padding approach (Sec. 2), we compared the performance of joint and disjoint models and noticed a considerable improvement in the last epoch performance for linear and stochastic datasets (financial and mushroom), slight improvement for datasets of medium complexity, and a large deterioration for some of the most difficult datasets — covertype and jester. When examined from a resource usage perspective, however, the joint model underperforms considerably. This result is significant, as most neural bandits use a joint (NeuralUCB, NeuralTS) or partially joint (neural-linear) model. It is important to note that in functional space a disjoint model reduces the overall number of kernel entries, while in the case of parameter space models we observe the opposite effect. We can only create a disjoint model at the expense of introducing additional $(k-1)|\boldsymbol{\theta}|$ number of parameters, which puts a burden on resources. Our result highlights this difference, and can help practitioners to assess the impact of choosing joint or disjoint models when using NK bandits.

Acknowledgments and Disclosure of Funding

The authors would like to thank Carlos Riquelme, Blair Bilodeau, Angus Galloway, and Magdalena Sobol for discussions and support in reviewing this manuscript. The authors thank the Canada Foundation for Innovation and Compute Canada for computing resources.

References

[1] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pages 127–135. PMLR, 2013.

[2] Robin Allesiardo, Raphael Feraud, and Djallel Bouneffouf. A Neural Networks Committee for the Contextual Bandit Problem. *arXiv:1409.8191 [cs]*, September 2014. URL `http://arxiv.org/abs/1409.8191`. arXiv: 1409.8191.

[3] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On Exact Computation with an Infinitely Wide Neural Net. *arXiv:1904.11955 [cs, stat]*, November 2019. URL `http://arxiv.org/abs/1904.11955`. arXiv: 1904.11955.

[4] Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks. *arXiv:1910.01663 [cs, stat]*, October 2019. URL `http://arxiv.org/abs/1910.01663`. arXiv: 1910.01663.

[5] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, May 2002. ISSN 1573-0565. doi: 10.1023/A:1013689704352. URL `https://doi.org/10.1023/A:1013689704352`.

[6] Youngmin Cho and Lawrence Saul. Kernel Methods for Deep Learning. In *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL `https://papers.nips.cc/paper/2009/hash/5751ec3e9a4feab575962e78e006250d-Abstract.html`.

[7] Sayak Ray Chowdhury and Aditya Gopalan. On Kernelized Multi-armed Bandits. *arXiv:1704.00445 [cs]*, May 2017. URL `http://arxiv.org/abs/1704.00445`. arXiv: 1704.00445.

[8] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

[9] Mark Collier and Hector Urdiales Llorens. Deep Contextual Multi-armed Bandits. *arXiv:1807.09809 [cs, stat]*, July 2018. URL http://arxiv.org/abs/1807.09809. arXiv: 1807.09809.

[10] Amit Daniely, Roy Frostig, and Yoram Singer. Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity. *arXiv:1602.05897 [cs, stat]*, May 2017. URL http://arxiv.org/abs/1602.05897. arXiv: 1602.05897.

[11] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2019. URL http://archive.ics.uci.edu/ml.

[12] Imène R. Goumiri, Benjamin W. Priest, and Michael D. Schneider. Reinforcement Learning via Gaussian Processes with Neural Network Dual Kernels. *arXiv:2004.05198 [cs, eess, stat]*, April 2020. URL http://arxiv.org/abs/2004.05198. arXiv: 2004.05198.

[13] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. Bayesian Deep Ensembles via the Neural Tangent Kernel. *arXiv:2007.05864 [cs, stat]*, October 2020. URL http://arxiv.org/abs/2007.05864. arXiv: 2007.05864.

[14] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. *arXiv:1806.07572 [cs, math, stat]*, February 2020. URL http://arxiv.org/abs/1806.07572. arXiv: 1806.07572.

[15] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020. URL https://tor-lattimore.com/downloads/book/.

[16] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. *arXiv:1711.00165 [cs, stat]*, March 2018. URL http://arxiv.org/abs/1711.00165. arXiv: 1711.00165.

[17] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124002, December 2020. ISSN 1742-5468. doi: 10.1088/1742-5468/abc62b. URL http://arxiv.org/abs/1902.06720. arXiv: 1902.06720.

[18] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. *Proceedings of the 19th international conference on World wide web - WWW '10*, page 661, 2010. doi: 10.1145/1772690.1772758. URL http://arxiv.org/abs/1003.0146. arXiv: 1003.0146.

[19] Ofir Nabati, Tom Zahavy, and Shie Mannor. Online Limited Memory Neural-Linear Bandits with Likelihood Matching. *arXiv:2102.03799 [cs]*, June 2021. URL http://arxiv.org/abs/2102.03799. arXiv: 2102.03799.

[20] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803*, 2019.

[21] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018.

[22] C. E. Rasmussen and C. Williams. *Gaussian processes for machine learning*. The MIT Press, 2006. ISBN 0-262-18253-X. URL http://gaussianprocess.org/gpml/.

[23] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling. *arXiv:1802.09127 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1802.09127. arXiv: 1802.09127.

[24] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *arXiv:0912.3995 [cs]*, June 2010. doi: 10.1109/TIT.2011.2182033. URL http://arxiv.org/abs/0912.3995. arXiv: 0912.3995.

[25] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. Publisher: JSTOR.

[26] Michal Valko, Nathaniel Korda, Remi Munos, Ilias Flaounas, and Nelo Cristianini. Finite-Time Analysis of Kernelised Contextual Bandits. *arXiv:1309.6869 [cs, stat]*, September 2013. URL http://arxiv.org/abs/1309.6869. arXiv: 1309.6869.

[27] Pan Xu, Zheng Wen, Handong Zhao, and Quanquan Gu. Neural Contextual Bandits with Deep Representation and Shallow Exploration. *arXiv:2012.01780 [cs, stat]*, December 2020. URL `http://arxiv.org/abs/2012.01780`. arXiv: 2012.01780.

[28] Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural Thompson Sampling. *arXiv:2010.00827 [cs, stat]*, October 2020. URL `http://arxiv.org/abs/2010.00827`. arXiv: 2010.00827.

[29] Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural Contextual Bandits with UCB-based Exploration. *arXiv:1911.04462 [cs, stat]*, July 2020. URL `http://arxiv.org/abs/1911.04462`. arXiv: 1911.04462.

A Experimental setup

A.1 Datasets

Table 1 provides an summary of the key properties associated with each of the UCI datasets we considered. While [23] and [19] used the same datasets, there are some subtle differences in the way that they were implemented. More details are in Sec. A.2.

Table 1: Summary of the UCI datasets used in the experiments.

Dataset	**Description**	**Distinguishing factor(s)**	d^2	k	$\sim n$	**Reward**	**Context**
Adult	predict income based on personal information	binary classification	13	2	50k	binary	categorical / integer
Census	predict occupation based on personal information	multivariate classification; large number of features	67	9	250k	binary	categorical / integer
Covertype	type of forest coverage in various areas	multivariate classification; requires non-linear models [23]	54	7	150k	binary	categorical / integer
Financial	stock prices from NYSE and NASDAQ	linear problem; easy to over-explore	21	8	4k	continuous	continuous
Jester	joke recommender system	inputs and outputs in the same domain	32	8	20k	continuous	continuous
Mushroom	poisonous vs. safe mushrooms	imbalanced stochastic rewards; captures aspects of classification and regression	117	2	10k	continuous	categorical
Statlog	space shuttle flight indicators	multivariate classification; tests exploration; one arm optimal 80% of the time; requires non-linear models [23]	9	7	45k	binary	integer

A.2 Experimentation details

We primarily compare our method against the state-of-the-art neural-linear [23] and neural-linear LiM2 [19]. We followed the experimental setup of [23] and [19] and report the average and standard deviation of the cumulative reward over 10 runs. We also include important baselines reported in [23], linear TS and multitask GP, due to their significant performance on simpler datasets. Whenever applicable, we used the hyperparameters reported in [23]. Due to differences in number of layers, and units per layer, between [23] and [19], in Tab. 2 we report the results for both settings. We used the neural-tangents library [20] to obtain the NTK, NNGP kernel, and the mean and covariance of the predictive posterior. We computed the NKs based on a two-layer fully-connected architecture with ReLU activations and regularizer $\gamma = 0.2$. NKs were then used to compute a GP, as presented in [17], later referred to in [13] as the randomized prior approach [21]. We tested both UCB and TS policies with the exploration parameter η set to 0.1. Even though we compare primarily against TS approaches, we add UCB for reference. We note that removing UCB does not change the overall rank of NK approaches with respect to other methods in places where NK achieved the highest performance. As established in [19], normalized cumulative rewards are computed w.r.t. baseline uniform sampling and the best algorithm for each dataset: $\text{norm\_cum\_rew}_{\text{alg}} = \frac{\text{cum\_rew}_{\text{alg}} - \text{cum\_rew}_{\text{uniform}}}{\text{cum\_rew}_{\text{best}} - \text{cum\_rew}_{\text{uniform}}}$.

We complement our results by reporting the average (over 10 runs) times per round that our algorithms took w.r.t. the neural-linear approach (Tab. 3). As the times grow with increasing number of collected data points, we report the shortest, median, and the longest average round. Both TS and UCB variants of our method tend to be about 5 times slower than the LiM2 approach, taking around 2 seconds for

[2]For adult and census datasets the categorical features are subsequently encoded with one hot vectors, resulting in ~ 80 features for adult and ~ 370 features for census. The numbers vary due to subsampling.

Table 2: Neural-linear methods compared over two significant hyperparameter settings.

	adult	census	covertype	financial	jester	mushroom	statlog
NK-TS $L = 1$ $\gamma = 0.2$	**4119 ± 16**	3152 ± 41	3402 ± 41	4407 ± 344	16779 ± 1234	4052 ± 4974	4720 ± 132
NK-TS $L = 2$ $\gamma = 0.2$	4113 ± 19	**3186 ± 29**	**3441 ± 36**	4162 ± 276	**16856 ± 1212**	2016 ± 3677	4820 ± 32
NeuralLinearTS $L = 1$ $n_l = 50$	3970 ± 36	2557 ± 54	2698 ± 76	4358 ± 363	12552 ± 1875	**11111 ± 336**	4771 ± 13
NeuralLinearTS $L = 2$ $n_l = 100$	3927 ± 26	2223 ± 181	2588 ± 74	4196 ± 350	11542 ± 2365	10516 ± 345	4801 ± 7
NeuralLinearTS-LiM2 $L = 1$ $n_l = 50$	4044 ± 17	2726 ± 36	2745 ± 141	**4485 ± 360**	14333 ± 1339	10962 ± 1070	4816 ± 66
NeuralLinearTS-LiM2 $L = 2$ $n_l = 100$	3993 ± 39	2304 ± 217	2748 ± 85	4357 ± 366	12325 ± 1005	10915 ± 631	**4872 ± 14**

the longest round. The total times can be further reduced for some applications by performing less frequent updates, as described in Sec. 3.2. However, additional study would need to be conducted to check for a similar effect in neural-linear approaches.

Table 3: Times per epoch [sec] (min / median / max) rounded to 2 significant figures.

	adult	census	covertype	financial	jester	mushroom	statlog
NTK-TS	0.0 / 0.8 / 2.25	0.0 / 0.07 / 1.93	0.0 / 0.64 / 2.47	0.0 / 0.02 / 1.79	0.0 / 0.48 / 1.62	0.0 / 0.82 / 2.66	0.0 / 0.85 / 2.42
NTK-UCB	0.0 / 0.8 / 1.85	0.0 / 0.07 / 2.04	0.0 / 0.71 / 2.01	0.0 / 0.02 / 2.04	0.0 / 0.44 / 3.15	0.0 / 0.82 / 4.81	0.0 / 0.85 / 2.81
JointNTK-TS	0.67 / 0.9 / 5.68	0.0 / 2.0 / 5.5	0.0 / 0.97 / 2.43	0.0 / 0.83 / 2.51	0.0 / 0.92 / 2.87	0.68 / 0.94 / 2.47	0.0 / 0.83 / 4.13
JointNTK-UCB	0.67 / 0.91 / 3.16	0.0 / 2.0 / 5.49	0.0 / 0.97 / 3.56	0.0 / 0.83 / 2.27	0.0 / 0.89 / 2.38	0.67 / 0.94 / 2.45	0.0 / 0.81 / 2.82
NeuralLinearTS	0.0 / 0.0 / 1.2	0.0 / 0.0 / 1.3	0.0 / 0.0 / 1.24	0.0 / 0.0 / 0.96	0.0 / 0.0 / 1.22	0.0 / 0.0 / 1.19	0.0 / 0.0 / 1.21
NeuralLinearTS-LiM2	0.02 / 0.03 / 0.33	0.04 / 0.1 / 0.36	0.03 / 0.06 / 0.36	0.04 / 0.08 / 0.38	0.03 / 0.09 / 0.37	0.02 / 0.03 / 0.32	0.03 / 0.08 / 0.36

B Theoretical background

B.1 Mathematical notation

We denote the training data by $(\mathbf{X}, \mathbf{y}) \in (\mathcal{X}, \mathcal{Y})$, and the test inputs with predictions by $(\mathbf{x}_*, y_*)$. The input data is composed of $(\mathbf{x}_i, y_i)$, where $i \in [n]$ when we talk about a predetermined set. Alternatively the data points can be indexed by time $t \in [T]$ in the context of bandits, where the data is collected sequentially. Parameter vectors are denoted by $\boldsymbol{\theta}$ and the reward estimation models by $f : \mathcal{X} \to \mathcal{Y}$. The dimensionality of both the feature vector and the parameter vector is denoted by p, i.e. $\boldsymbol{\theta} \in \mathbb{R}^p$ and $\phi(\mathbf{x}) \in \mathbb{R}^p$. In the linear case $\phi(\mathbf{x}) = \mathbf{x}$, so also $\mathbf{x} \in \mathbb{R}^p$. We denote a generic kernel matrix by $\mathbf{K}$ and the associated kernel function by $K(\cdot, \cdot)$. We use $\mathbf{x}_*$ for a general test input in the context of Bayesian inference, or $\mathbf{x}_t$ when the test input is given at a specific time t. We denote test predictions by y_* or y_t.

B.2 Contextual bandits from a Bayesian perspective

Contextual multi-armed bandits are probabilistic models in which at each round $t \in [0, T]$ of a sequential decision process, we observe a set of k arms and a global context $\mathbf{x}_t$. The role of a *policy* π is then to select an action $a \in [k]$ and observe the associated reward $y_{t,a}$. We name the triplet $(\mathbf{x}_{t,a_t}, a_t, y_{t,a_t})$ an *observation* at time t. The theoretical objective is to minimize (instantaneous) regret $R_T = \mathbb{E}\big[\sum_{t=1}^{T} \max_a y_{t,a} - \sum_{t=1}^{T} y_{t,a_t}\big]$. In practical scenarios, however, where no information is available about the optimal reward, we often resort to reporting just the cumulative reward $\sum_{t=1}^{T} y_{t,a_t}$. Thompson sampling (TS) is a policy that operates on the Bayesian principle. The reward estimates for each arm are computed in terms of a predictive distribution $p(y_*|a, D_a, \boldsymbol{\theta}) = \int p(y_*|\boldsymbol{\theta})p(\boldsymbol{\theta}|D_a)d\boldsymbol{\theta}$. Depending on the parameters and the nature of the data, this integral can be difficult to compute. Therefore in practice we often resort to sampling (hence "Thompson *sampling*"). At each step the TS policy calculates the posterior over the parameters $p(\boldsymbol{\theta}|D_a)$ and then collects a single parameter sample $\boldsymbol{\theta}_a \sim p(\boldsymbol{\theta}|D_a)$ per action. Then, instead of computing a full predictive posterior, the algorithm chooses actions according to the expected value of a posterior parametrized by $\boldsymbol{\theta}_a$, i.e. $\hat{a} \leftarrow \arg\max_a \mathbb{E}[y_*|\boldsymbol{\theta}_a]$. In practice $p(\boldsymbol{\theta}|D_a)$ can be difficult to derive analytically. For this reason, the prior and the likelihood are usually chosen to be Gaussian, which results in a closed form solution for $p(\boldsymbol{\theta}|D_a)$ that also becomes Gaussian. Linear TS [1] (Alg. 2) is a contextual variant of TS, in which $\mathbb{E}[y_*|\boldsymbol{\theta}_a] = \mathbf{x}_t^T \boldsymbol{\theta}_a$. A Gaussian formulation allows for straightforward computation of its parameters, covariance $\boldsymbol{\Sigma} = \sum_t \mathbf{x}_t \mathbf{x}_t^T$ and the response vector $b = \sum_t \mathbf{x}_t y_t$, by aggregating over the past epochs.

Instead of sampling parameters, TS can also be formulated such that rewards are sampled directly from the predictive distribution [7], which can be defined through Bayesian inference, or formulated

Algorithm 2 Linear Thompson sampling

```
Set $\mathbf{\Sigma}_a = \mathbf{I}_d, \mathbf{b}_a = \mathbf{0}_d \quad \forall a$
for round $t = 1, 2, \ldots, T$ do
    Observe context $\mathbf{x}_t$
    for arm $a = 1, 2, \ldots, k$ do
        $\hat{\boldsymbol{\mu}}_a \leftarrow \mathbf{\Sigma}_a^{-1}\mathbf{b}_a$
        Sample parameter $\tilde{\boldsymbol{\theta}}_a \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_a, \sigma_y^2\mathbf{\Sigma}_a^{-1})$
    end for
    Choose $a_t \leftarrow \arg\max_a \mathbf{x}_t^T\tilde{\boldsymbol{\theta}}_a$ and get reward $y_t$
    Update posterior:
        $\mathbf{\Sigma}_{a_t} \leftarrow \mathbf{\Sigma}_{a_t} + \mathbf{x}_t\mathbf{x}_t^T$
        $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + \mathbf{x}_t y_t$
end for
```

independently, as in [17]. In our experiments we use the former, two-stage sampling process, for weight-space view algorithms, like neural-linear or NeuralUCB/TS, and the latter for function space (kernelized) algorithms, like multitask GPs or our method. While TS offers an explicit Bayesian interpretation, other stochastic policies can take advantage of the inference process in a similar way to model the reward distributions of each arm. The upper confidence bounds (UCB) algorithm [5] becomes highly related to TS when examined from Bayesian perspective. In each step of the UCB policy, the actions are chosen according to the highest reward estimate $a_t \leftarrow \arg\max_a \mathrm{UCB}_{t,a}$. The reward estimates of each arm are updated at each round as $\mathrm{UCB}_a = \mu_a + \sigma_a$, resulting in an exploration-exploitation trade-off between picking arms that give the best results based on the history of collected rewards (large μ_a; e.g. $\mu_a = \bar{y}_a$) and the ones we are most uncertain about (large σ_a). From a Bayesian perspective the parameters μ_a and σ_a correspond to moments of the predictive posterior $p(y_*|a, D_a, \boldsymbol{\theta}_a)$. In this case it is advantageous to derive the predictive density in its full form. In fact, we can see in the linear UCB (aka LinUCB [8] [18]) algorithm (Alg. 3) below that UCBs for each action are calculated using exactly the mean $\mu_a = \mathbf{x}_t^T\mathbf{\Sigma}^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{x}_t^T\hat{\boldsymbol{\theta}}$ and the standard deviation $\sigma_a = \sqrt{\mathbf{x}_t^T\mathbf{\Sigma}^{-1}\mathbf{x}_t}$ of a normal distribution. It also includes the exploration parameter, denoted η, which can be optimized analytically w.r.t. a particular regret bound, or set by the user as a hyperparameter. In this work we distinguish between the exploration parameter η and a regularizer that captures the randomness coming from the data and the parameters $\gamma = \frac{\sigma_y^2}{\sigma_\theta^2}$. In practical applications, however, this is of little importance, as σ_y is often set to 1, in which case γ can be combined with η to form a single hyperparameter.

Algorithm 3 Linear UCB

```
Set $\mathbf{\Sigma}_a = \mathbf{I}_d, \mathbf{b}_a = \mathbf{0}_d \quad \forall a$
for round $t = 1, 2, \ldots, T$ do
    Observe context $\mathbf{x}_t$
    for arm $a = 1, 2, \ldots, k$ do
        $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{\Sigma}_a^{-1}\mathbf{b}_a$
        $\mathrm{UCB}_{t,a} \leftarrow \mathbf{x}_t^T\hat{\boldsymbol{\theta}}_a + \eta\sqrt{\mathbf{x}_t^T\mathbf{\Sigma}_a^{-1}\mathbf{x}_t}$
    end for
    Choose $a_t \leftarrow \arg\max_a \mathrm{UCB}_{t,a}$ and get reward $y_t$
    Update posterior:
        $\mathbf{\Sigma}_{a_t} \leftarrow \mathbf{\Sigma}_{a_t} + \mathbf{x}_t\mathbf{x}_t^T$
        $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + \mathbf{x}_t y_t$
end for
```

A contextual bandit can be either joint or disjoint. A joint model (e.g. [8]) uses a single model that makes predictions based on action-specific contexts, while a disjoint model (Alg. 2 and Alg. 3) uses a separate model per action, which is more suited for a global context scenario.

B.3 Non-linear bandits

KernelUCB [26] is a "kernelized" version of LinUCB [8], designed to perform efficiently in a $p >> n$ situation, where n is the number of collected data samples, and p is the dimensionality of data features. The derivation is based on kernel feature regression: $y = \phi(\mathbf{x})^T\boldsymbol{\theta}$, with $\phi(\mathbf{x})$ providing a mapping to high-dimensional (possibly infinite-dimensional) space, also known as the reproducing kernel Hilbert space (RKHS). Given the data $(\mathbf{X}, \mathbf{y})$ and a regularization parameter γ, the analytical solution to the feature regression model is given by: $\hat{\boldsymbol{\theta}} = (\phi(\mathbf{X})^T\phi(\mathbf{X}) + \gamma\mathbf{I})^{-1}\phi(\mathbf{X})^T\mathbf{y}$. If p is large, so is $\phi(\mathbf{X})^T\phi(\mathbf{X})$, and computing its inverse is computationally heavy or impossible (e.g. when $p = \infty$). The kernel trick is to use the identity $\left(\mathbf{\Phi}^T\mathbf{\Phi} + \gamma\mathbf{I}\right)^{-1}\mathbf{\Phi}^T = \mathbf{\Phi}^T\left(\mathbf{\Phi}\mathbf{\Phi}^T + \gamma\mathbf{I}\right)^{-1}$ to convert the $p \times p$ covariance matrix $\mathbf{\Sigma} = \Sigma(\mathbf{X}, \mathbf{X}) = \phi(\mathbf{X})^T\phi(\mathbf{X})$ into a smaller $n \times n$ kernel matrix $\mathbf{K} = K(\mathbf{X}, \mathbf{X}) = \phi(\mathbf{X})\phi(\mathbf{X})^T$. In the case where the regularizer represents the model's noise, $\gamma = \frac{\sigma_y^2}{\sigma_\theta^2}$, the resulting feature regression becomes a Gaussian process regression [26] [24], which has a predictive posterior given by:

$$p(y_*|\mathbf{x}_*, D) = \mathcal{N}\big(K(\mathbf{x}_*, \mathbf{X})(K(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}\mathbf{y}, \tag{1}$$

$$\sigma_\theta^2(K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})(K(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}K(\mathbf{X}, \mathbf{x}_*))\big). \tag{2}$$

UCB algorithms commonly use the predictive posterior parameters to derive a tight upper bound on the regret. We can, for example, specify UCB in terms of the mean and the standard deviation as follows:

$$\mu \leftarrow K(\mathbf{x}_*, \mathbf{X})(K(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}\mathbf{y} \tag{3}$$

$$\sigma \leftarrow \eta\sigma_\theta\sqrt{K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})(K(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}K(\mathbf{X}, \mathbf{x}_*)}. \tag{4}$$

B.4 Neural bandits

Neural bandits operate on the premise of feature regression. Even though any type of network is permissible, the prevalent architectures in the literature include: neural greedy, neural-linear, ensemble models [2], Bayesian neural networks, and their approximations [23, 9]. In all those approaches the input $\mathbf{x}$ is first propagated through a neural network $f(\mathbf{x})$ to yield learned features. The neural-greedy (e.g. the base model in [2]) approach uses predictions of a NN directly to choose the next action. It is considered a naive approach, tantamount to exploitation in feature space. An alternative is a full Bayesian neural network. A comparison of Bayesian NNs that can be used to model a predictive posterior in the Thompson sampling setup can be found in [23]. One of the key findings of [23] is that current Bayesian NNs, while providing better estimates than regular networks, are often too slow to operate efficiently as a subroutine for contextual bandits, while Bayesian approximation methods, like dropout [9], underperform in practical scenarios.

We mainly compare our model to two baselines — neural-linear, given by [23], and its limited memory version, given by [19]. Neural-linear can be thought of as a mid-point between frequentist and Bayesian methods. The model operates on the basis of Bayesian feature regression, with features $\phi(\mathbf{x})$ provided by the penultimate layer of the neural network. As in linear Thompson sampling the parameters are sampled according to a posterior $p(\boldsymbol{\theta}|(\Phi, \mathbf{y}))$. Instead of setting the prior on likelihood parameters manually, those models use another classic approach to Thompson sampling and apply a conjugate prior for the parameters of the likelihood.

$$p(\boldsymbol{\theta}, \sigma_y^2) = p(\boldsymbol{\theta}|\sigma_y^2)p(\sigma_y^2) = \mathcal{N}(\tilde{\boldsymbol{\mu}}, \sigma_y^2\mathbf{\Sigma}^{-1})IG(a, b)$$

The neural-linear approach with conjugate priors was shown to obtain good empirical performance [23], most likely due to more aggressive initial exploration and higher final accuracy of the model. The limited memory version by [19] uses likelihood matching to preserve that performance, while greatly reducing resource usage.

As any contextual bandit, neural bandits can be disjoint or joint. A disjoint model uses a separate NN per action. A joint model is instead conditioned on an action, which can be achieved, for example, by means of modulation, input concatenation or input padding.

B.5 Neural kernels

In this work we define a neural kernel to be any compositional kernel [6, 10] that aims to mimic a state or behaviour of a neural network. A form of the compositional kernel was analytically derived in [6] for a family of one-sided polynomial activation functions. Most notably this family includes ReLU. The basis of the derivation is that the inner product $\sum_{i=1}^{p} \phi_{ReLU}(\mathbf{x})\phi_{ReLU}(\mathbf{x})^T$ becomes an integral when the number of output nodes p is taken to infinity, which in turn constitutes the arc-cosine kernel. The common practice in kernel regression is for the user to choose the kernel function based on certain assumptions about the output function, e.g. smoothness. The arc-cosine kernel, instead, constitutes an exact analytical solution by taking the output dimension to infinity: $\lim_{p\to\infty} \phi(\mathbf{x})\phi(\mathbf{x})^T$. Another useful property of the arc-cosine kernel is that it allows for compositional derivation: $K^{(l+1)}(\mathbf{x},\mathbf{x}) = \phi(\phi(\mathbf{x}))\phi(\phi(\mathbf{x}))^T$. The form of this composition includes the original kernel $K^{(l)}(\mathbf{x},\mathbf{x}) = \phi(\mathbf{x})\phi(\mathbf{x})^T$, and therefore can be applied recursively:

$$K_m^{(l+1)}(\mathbf{x},\mathbf{x}') = \frac{1}{\pi}\left[K_m^{(l)}(\mathbf{x},\mathbf{x})K_m^{(l)}(\mathbf{x}',\mathbf{x}')\right]^{m/2} J_m\big(\alpha_m^{(l)}\big), \tag{5}$$

$$J_m(\alpha) = (-1)^m(\sin\alpha)^{(2m+1)}\left(\frac{1}{\sin\alpha}\frac{\partial}{\partial\alpha}\right)^m\left(\frac{\pi-\alpha}{\sin\alpha}\right)$$

$$\alpha_m^{(l)} = \cos^{-1}\left(K_m^{(l)}(\mathbf{x},\mathbf{x}')\big[K_m^{(l)}(\mathbf{x},\mathbf{x})K_m^{(l)}(\mathbf{x}',\mathbf{x}')\big]^{-1/2}\right),$$

where m is the degree of the polynomial in the family considered by [6] ($m = 1$ for ReLU), $J_m(\alpha)$ is a function specifying the angular dependence, and α is the angle between inputs in RKHS.

The line of work on NKs was recently extended [16] by (1) placing it in a GP framework, (2) adding the parameters controlling the variances of weights and biases of the neural network, and (3) deriving a numerical method for approximating the neural kernel to any well-behaved activation function. In our experiments we used the neural-tangents library [20], which allows for full GP inference, following [16], but uses [6]'s analytical derivation for ReLU activation. The following general formulas apply for composing the NNGP (aka conjugate) kernel [16, 3]:

$$\Sigma^{(0)}(\mathbf{x},\mathbf{x}') = \mathbf{x}^T\mathbf{x}'$$

$$\Lambda^{(l)}(\mathbf{x},\mathbf{x}') = \begin{pmatrix}\Sigma^{(l-1)}(\mathbf{x},\mathbf{x}') & \Sigma^{(l-1)}(\mathbf{x},\mathbf{x}') \\ \Sigma^{(l-1)}(\mathbf{x}',\mathbf{x}) & \Sigma^{(l-1)}(\mathbf{x}',\mathbf{x}')\end{pmatrix}$$

$$\Sigma^{(l)}(\mathbf{x},\mathbf{x}') = \sigma\mathbb{E}_{(\mathbf{u},\mathbf{v})\sim GP(\mathbf{0},\Lambda^{(l)})}[\phi(\mathbf{u})^T\phi(\mathbf{v})].$$

As we use only ReLU in our work, Eq. 5 applies directly in kernel derivations for our experiments. Regularized, the resulting *NNGP* has the following moments:

$$\begin{aligned}\mathbb{E}[y_*] &= \mathcal{K}(\mathbf{x}_*,\mathbf{X})(\mathcal{K}(\mathbf{X},\mathbf{X})+\gamma\mathbf{I})^{-1}\mathbf{y} \\ \mathrm{Var}[y_*] &= \sigma_\theta^2(\mathcal{K}(\mathbf{x}_*,\mathbf{x}_*) - \mathcal{K}(\mathbf{x}_*,\mathbf{X})(\mathcal{K}(\mathbf{X},\mathbf{X})+\gamma\mathbf{I})^{-1}\mathcal{K}(\mathbf{X},\mathbf{x}_*)),\end{aligned}$$

where $\boldsymbol{\mathcal{K}} = \boldsymbol{\Sigma}^{(l)}$. The NNGP kernel models the network's covariance at random initialization, and allows for training by means of Bayesian inference. A related approach is to model the full dynamics of the network during GD training. Jacot et al. [14] showed that the gradient descent (GD) optimization process, when examined from functional perspective, can be captured by a kernel with RKHS corresponding to a network's gradients, further referred to as the neural tangent kernel (NTK). The general form of NTK is given by [14, 3]:

$$\dot{\Sigma}(\mathbf{x},\mathbf{x}') = \sigma\mathbb{E}_{(\mathbf{u},\mathbf{v})\sim GP(\mathbf{0},\Lambda^{(l)})}[\dot{\phi}(\mathbf{u})^T\dot{\phi}(\mathbf{v})]$$

$$\Theta(\mathbf{x},\mathbf{x}') = \sum_{l=1}^{L+1}\left(\Sigma^{(l-1)}(\mathbf{x},\mathbf{x}')\cdot\prod_{l'=l}^{L+1}\dot{\Sigma}^{(l')}(\mathbf{x},\mathbf{x}')\right),$$

where dot denotes gradients. NTK represents the tangent kernel of the form $\nabla_{\boldsymbol{\theta}} f(\mathbf{X}) \nabla_{\boldsymbol{\theta}} f(\mathbf{X})^T$ in the infinite width-limit. In this work we refer to its corresponding parameter space covariance $\mathbf{Z} = \nabla_{\boldsymbol{\theta}} f(\mathbf{X})^T \nabla_{\boldsymbol{\theta}} f(\mathbf{X})$ as neural tangent feature matrix (NTF). In the infinite width limit the parameters change so little throughout the training ($O(1/\sqrt{p})$, where p is the number of parameters) that the network can be approximated with a first order Taylor expansion around the initial parameters $\boldsymbol{\theta}_0$ [17]:

$$f_{\boldsymbol{\theta}_t}(\mathbf{x}_*) \approx f^{\text{lin}}_{\boldsymbol{\theta}_t}(\mathbf{x}_*) = f_{\boldsymbol{\theta}_0}(\mathbf{x}_*) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}_0}(\mathbf{x}_*)^T \boldsymbol{\omega}_t,$$

where $\boldsymbol{\omega}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$. The approximation $f^{\text{lin}}_{\boldsymbol{\theta}_t}(\mathbf{x}_*)$ is referred to as the "linearized" network.

Because Θ stays constant throughout training, the solution to the ODE above has a closed form and is straightforward to compute [17]. It yields the following model after training (when $t \to \infty$):

$$f^{\text{lin}}_{\boldsymbol{\theta}_\infty}(\mathbf{x}_*) = f_{\boldsymbol{\theta}_0}(\mathbf{x}_*) - \Theta(\mathbf{x}_*, \mathbf{X})\Theta(\mathbf{X}, \mathbf{X})^{-1}(f_{\boldsymbol{\theta}_0}(\mathbf{X}) - \mathbf{y}).$$

By putting a Gaussian prior of $\mathcal{N}(\mathbf{0}; \boldsymbol{\mathcal{K}})$ On the functional form of the neural network, the linearized model results in a GP predictive posterior with the following moments:

$$\begin{aligned}\mathbb{E}[y_*] &= \Theta(\mathbf{x}_*, \mathbf{X})\Theta(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y} \\ \text{Var}[y_*] &= \mathcal{K}(\mathbf{x}_*, \mathbf{x}_*) + \Theta(\mathbf{x}_*, \mathbf{X})\Theta(\mathbf{X}, \mathbf{X})^{-1}\mathcal{K}(\mathbf{X}, \mathbf{X})\Theta(\mathbf{X}, \mathbf{X})^{-1}\Theta(\mathbf{X}, \mathbf{x}_*) - \\ &\quad - 2\Theta(\mathbf{x}_*, \mathbf{X})\Theta(\mathbf{X}, \mathbf{X})^{-1}\mathcal{K}(\mathbf{X}, \mathbf{x}_*),\end{aligned} \tag{6}$$

This formulation of predictive distribution $p(y_*|\mathbf{x}_*, D)$ is given in [17] (eq 16). Lee et al. [17] points out, however, that their formulation does not admit the interpretation of the resulting GP as a full predictive *posterior*. Yet it can still be used as a predictive distribution. He et al. [13] attempted to remedy this by adding a missing variance (defined by a function $\delta(\cdot)$), of the parameters of all layers except the last one, to the linearized network formulation:

$$f^{\text{lin}}_{\boldsymbol{\theta}_t}(\mathbf{x}_*) = f_{\boldsymbol{\theta}_0}(\mathbf{x}_*) + \nabla_{\theta^{L+1}} f_{\boldsymbol{\theta}_0}(\mathbf{x}_*)\boldsymbol{\omega}_t^{L+1} + \delta(\mathbf{x}_*) + \nabla_{\theta^{\leq L}} f_{\boldsymbol{\theta}_0}(\mathbf{x}_*)\boldsymbol{\omega}_t^{\leq L}.$$

This results in the following predictive distribution, denoted *NTKGP*, that can be interpreted as a predictive posterior:

$$\begin{aligned}\mathbb{E}[y_*] &= \Theta(\mathbf{x}_*, \mathbf{X})(\Theta(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}\mathbf{y} \\ \text{Var}[y_*] &= \sigma_\theta^2(\Theta(\mathbf{x}_*, \mathbf{x}_*) - \Theta(\mathbf{x}_*, \mathbf{X})(\Theta(\mathbf{X}, \mathbf{X}) + \gamma\mathbf{I})^{-1}\Theta(\mathbf{X}, \mathbf{x}_*)).\end{aligned}$$

All mentioned predictive distributions were categorized in [13]. Notably linearized network GPs (Eq. 6) without regularization were related to deep ensembles, and with regularization to randomized priors [21]. Unless otherwise stated we use the randomized prior approach in all our experiments.