Thompson Sampling via Local Uncertainty: Appendix

A. Model Details

In the implementation, we all use ReLU activation function, except for using the exponential link function for parameterizing the standard deviations of the Gaussian distributions. We set prior distribution $p(\boldsymbol{z}_t)$ as a normal distribution with mean $\mathbf{0}$ and standard deviation $\sigma\mathbf{I}$, where σ is a point estimation with initial value $\sigma = 1.25$. Denote the latent dimension as H, the number of actions as A, and context dimension as C.

For LU-Gauss: we set the latent dimension of $\boldsymbol{z}$ as $H = 50$. $\mathcal{T}_{\boldsymbol{\theta}}$ is a neural network composed of input layer $[\boldsymbol{x}, \boldsymbol{z}]$ in dimension $H + C$, one hidden layer [50], and output layer in dimension A. We use point estimate on $\boldsymbol{\Sigma}_{\boldsymbol{r}}$. $\mathcal{T}_{\boldsymbol{\phi}_0}$ is a neural network composed of input layer $[\boldsymbol{x}, \boldsymbol{\epsilon}]$ in dimension $C + C$, one hidden layer [100], and output layer in dimension 50, where $\boldsymbol{\epsilon}$ has the same dimension as context dimension. $\mathcal{T}_{\boldsymbol{\phi}_1}$ is a neural network composed of input layer in dimension 50, one hidden layer [50], and output layer in dimension H. $\mathcal{T}_{\boldsymbol{\phi}_2}$ is a neural network composed of input layer in dimension 50, one hidden layer [50], and output layer in dimension H.

For LU-SIVI: we set the latent dimension of $\boldsymbol{z}$ as $H = 50$ and the number of noise $K = 50$. $\mathcal{T}_{\boldsymbol{\theta}}$ is a neural network composed of input layer $[\boldsymbol{x}, \boldsymbol{z}]$ in dimension $H + C$, one hidden layer [50], and output layer in dimension A. We use point estimate on $\boldsymbol{\Sigma}_{\boldsymbol{r}}$. $\mathcal{T}_{\boldsymbol{\phi}_1}$ is a neural network composed of input layer $[\boldsymbol{x}, \boldsymbol{\epsilon}]$ in dimension $C + C$, one hidden layer [100], and output layer in dimension H, where $\boldsymbol{\epsilon}$ has the same dimension as context dimension. $\mathcal{T}_{\boldsymbol{\phi}_2}$ is a neural network composed of input layer in dimension C, one hidden layer [50], and output layer in dimension H.

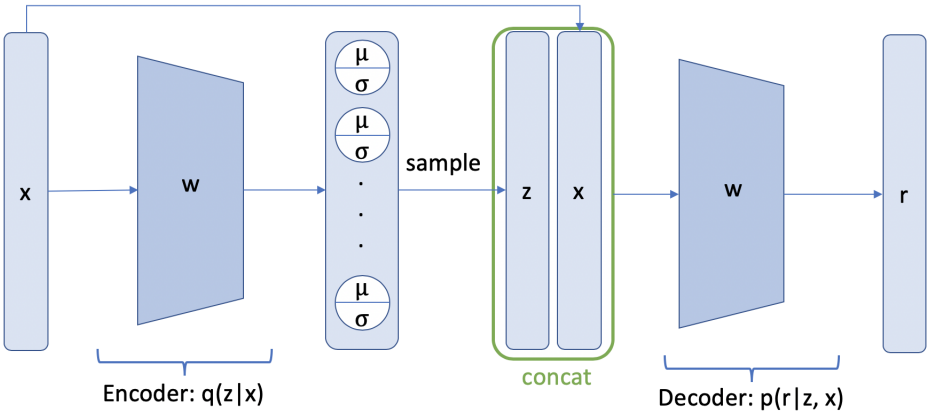

Figure 1: Model architecture

B. Comparison with FBNNs

We have tried to make comparison with FBNNs of Sun et al. (2019). The code for FBNNs, however, is too computationally expensive for us to run. For example, the smallest FBNN model (1x50) used by Sun et al. (2019) is already about 20 times slower than our proposed LU-Gauss and LU-SIVI, let along larger FBNN models. For this reason, we directly quote FBNNs' results reported in Sun et al. (2019), which were obtained based on as few as 10 random trials. Note the following results, which are quoted merely for reference, are not intended for a rigorous comparison as they are not obtained by averaging over the same set of 50 random sequences used by the other algorithms. The Mean Value is 46.0 for FBNN (1×50), 47.0 for (2×50), 48.9 for (3×50), 45.3 for (1×500), 44.2 for (2×500), and 44.6 for (3×500). Their best Mean Value over the eight datasets is 44.2, from model FBNN (2 x 500), which is considerably slower to run in comparison to both LU-Gauss and LU-SIVI.

C. Details for TS models via Global Uncertainty

Linear Method: This method uses Bayesian linear regression with closed-form Gibbs sampling update equations, which relies on the posterior distributions of regression coefficients for TS updates and maintains computational efficiency due to the use of conjugate priors. This method assumes that at time t, the reward y_t of an action given contextual input $\boldsymbol{x}_t$ is generated as $y_t = \boldsymbol{x}_t^T \boldsymbol{\beta} + \epsilon_t$, where $\boldsymbol{\beta}$ is the vector of regression coefficients and $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is the noise. Note to avoid cluttered notation, we omit the action index. This method places a normal prior on $\boldsymbol{\beta}$ and inverse gamma prior on σ^2. At time t, given $\boldsymbol{x}_t$ and the current random sample of $\boldsymbol{\beta}$, it takes the best action under TS and receives reward y_t; with $\boldsymbol{x}_{1:t}$ and $y_{1:t}$, it samples σ^2 from its inverse gamma distributed conditional posterior, and then samples $\boldsymbol{\beta}$ from its Gaussian distributed conditional posterior; it proceeds to the next time and repeats the same update scheme under TS.

While this linear method accurately captures the posterior uncertainty of the global parameters $\boldsymbol{\beta}$ and σ^2, its representation power is limited by both the linear mean and Gaussian distribution assumptions on reward y given context $\boldsymbol{x}$. In practice, the linear method often provides surprisingly competitive results, thanks to its ability to provide accurate uncertainty estimation. However, when its assumptions do not hold well in practice, such as when there are complex nonlinear dependencies between the rewards and contextual vectors, the linear method, even though with accurate posterior estimation, may not be able to converge to a good local optimal solution. Following Riquelme et al. (2018), we refer to this linear method as "LinFullPost."

Neural Linear: To enhance the representation power of LinFullPost while maintaining closed-form posterior sampling, Riquelme et al. (2018) propose the "Neural Linear" method, which feeds the representation of the last layer of a neural network as the covariates of a Bayesian linear regression model. It models the reward distribution of an action conditioning on $\boldsymbol{x}$ as $y \sim \mathcal{N}(\boldsymbol{\beta}^T \boldsymbol{z}_{\boldsymbol{x}}, \sigma^2)$, where $\boldsymbol{z}_{\boldsymbol{x}}$ is the output of the neural network given $\boldsymbol{x}$ as the input. It separates representation learning and uncertainty estimation into two parts. The neural network part is responsible for finding a good representation of $\boldsymbol{x}$, while the Bayesian linear regression part is responsible for obtaining uncertainty estimation on the regression coefficient vector $\boldsymbol{\beta}$, and making the decision on which action to choose under TS. The training for the two parts can be performed at different time-scales. It is reasonable to update the Bayesian linear regression part as soon as a new data arrives, while to update the neural network part only after collecting a number of new data points.

As Neural Linear transforms context $\boldsymbol{x}$ into latent space $\boldsymbol{z}$ via a deterministic neural network, the model uncertainty still all comes from sampling the global parameters $\boldsymbol{\beta}$ and σ from their posteriors under the Bayesian linear regression part. Hence, this method relies on the uncertainty of global model parameters to perform TS.

Bayesian By Backprop (BBB): This method uses variational inference to perform uncertainty estimation on the neural network weights (Blundell et al., 2015). In order to exploit the reparameterization trick for tractable variational inference (Kingma & Welling, 2013), it models the neural network weights with independent Gaussian distributions, whose means and variances become the network parameters to be optimized. However, the fully factorized mean-field variational inference used by BBB is well-known to have the tendency to underestimate posterior uncertainty (Jordan et al., 1999; Blei et al., 2017). Moreover, it is also questionable whether the weight uncertainty can be effectively translated into reward uncertainty given context $\boldsymbol{x}$ (Bishop, 2006; Sun et al., 2019), especially considering that BBB makes both the independent and Gaussian assumptions on its network weights. For TS, underestimating uncertainty often leads to under exploration. As the neural network weights are shared across all observations, BBB also relies on the uncertainty of global parameters to perform TS.

Particle-Interactive TS via Discrete Gradient Flow (π-TS-DGF): The π-TS-DGF method of Zhang et al. (2019) casts posterior approximation as a distribution optimization problem under the Wasserstein-gradient-flow framework. In this setting, posterior sampling in TS can be considered as a convex optimization problem on the space of probability measures. For tractability, it maintains a set of particles that interact with each other and evolve over time to approximate the posterior. For the contextual bandit problem, each particle corresponds to a set of neural network weights, and the algorithm uniformly at random chooses one particle at each time and uses it as a posterior sample of the neural network weights. A benefit of π-TS-DGF is that it imposes no explicit parametric assumption on the posterior distribution. However, it faces an uneasy choice of setting the number of particles. Maintaining a large number of particles means training many sets of neural network weights at the same time, which is considerably expensive in computation, while a small number of particles might lead to bad uncertainty estimation due to inaccurate posterior approximation. The computational cost prevents π-TS-DGF from using large-size neural networks. Similar to BBB, π-TS-DGF also relies on the uncertainty of global parameters to perform exploration.

D. Results in Simple Regret Metric

We report the performance in terms of Simple Regret in Table 1. Simple Regret is approximated by averaging the regrets over the last 500 steps.

Table 1: Comparison of Normalized Simple Regret between various methods, with the normalization performed with respect to the Simple Regret of Uniform. For each dataset, the same set of 50 random contextual sequences are used for all algorithms. For each algorithm on a given dataset, we report its mean and standard error over these 50 independent random trials.

Algorithms	Mean Rank	Mean Value	Mushroom	Financial	Statlog	Jester	Wheel	Covertype	Adult	Census
Uniform	7	100.00	100.00 ± 9.70	100.00 ± 12.62	100.00 ± 1.76	100.00 ± 8.66	100.00 ± 13.20	100.00 ± 1.88	100.00 ± 1.38	100.00 ± 1.48
BBB	5.125	52.77	19.36 ± 8.58	30.25 ± 19.04	30.91 ± 8.80	71.54 ± 5.28	60.62 ± 25.12	62.32 ± 7.74	95.73 ± 2.63	51.41 ± 6.31
Neural Linear	4.0	41.30	7.10 ± 7.51	2.60 ± 0.54	3.03 ± 0.91	74.88 ± 4.85	33.05 ± 16.12	48.99 ± 2.96	93.93 ± 2.07	66.84 ± 3.65
LinFullPost	3.125	41.06	**4.14 ± 3.06**	**0.67 ± 0.24**	11.28 ± 1.90	69.64 ± 4.87	**20.62 ± 13.68**	44.26 ± 3.13	91.56 ± 2.10	86.32 ± 2.12
π-TS-DGF	3.125	42.15	6.37 ± 2.91	2.73 ± 3.31	**1.01 ± 1.79**	74.93 ± 4.11	78.00 ± 27.72	40.75 ± 3.20	89.27 ± 2.80	**44.16 ± 2.65**
LU-Gauss	**2.625**	**38.83**	27.79 ± 11.04	3.73 ± 2.96	1.99 ± 3.13	**67.91 ± 5.38**	38.33 ± 22.76	**38.51 ± 3.25**	**85.85 ± 3.33**	46.51 ± 3.20
LU-SIVI	3.0	40.08	7.07 ± 4.06	1.96 ± 3.09	3.70 ± 7.07	68.43 ± 5.27	55.64 ± 25.79	42.76 ± 3.60	87.08 ± 2.74	54.02 ± 11.00

E. Algorithms

Algorithm 1 Vanilla Thompson Sampling

Input: Prior distribution $p_0(\boldsymbol{\theta})$.
Output: Fine-tuned posterior distribution $p_T(\boldsymbol{\theta})$.
for $t = 1, \ldots, T$ **do**
  Observe context $\boldsymbol{x}_t$.
  Sample parameters $\boldsymbol{\theta}_t \sim p_{t-1}(\boldsymbol{\theta})$.
  Select action $a_t = \text{argmax}_a \, f(\boldsymbol{x}, a; \boldsymbol{\theta}_t)$.
  Observe reward r_t.
  Update posterior distribution $p_t(\boldsymbol{\theta})$ with $(\boldsymbol{x}_t, a_t, r_t)$.
end for

Algorithm 2 LU-Gauss: Thompson Sampling via Gaussian Local Uncertainty

Input: Likelihood $p(\boldsymbol{r}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{r}_t}, \boldsymbol{\Sigma}_{\boldsymbol{r}})$, prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{z}})$, and variational distribution $q(\boldsymbol{z}_t \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}_t}, \boldsymbol{\Sigma}_{\boldsymbol{z}_t})$; neural networks $\mathcal{T}_{\boldsymbol{\theta}}$, $\mathcal{T}_{\boldsymbol{\phi}_0}$, $\mathcal{T}_{\boldsymbol{\phi}_1}$, and $\mathcal{T}_{\boldsymbol{\phi}_2}$, $t_f = 20$ (training frequency), $t_s = 40$ (the number of mini-batches per training period).
Output: Inferred parameters $\boldsymbol{\Sigma}_{\boldsymbol{r}}$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}_0$, $\boldsymbol{\phi}_1$, and $\boldsymbol{\phi}_2$.
Initialize $\boldsymbol{\Sigma}_{\boldsymbol{r}}$
Initialize dataset $D_0 = \emptyset$
for $t = 1, \ldots, T$ **do**
  Observe context $\boldsymbol{x}_t$
  $\boldsymbol{\mu}_{\boldsymbol{z}_t} = \mathcal{T}_{\boldsymbol{\phi}_1}(\boldsymbol{h})$, $\boldsymbol{\Sigma}_{\boldsymbol{z}_t} = \mathcal{T}_{\boldsymbol{\phi}_2}(\boldsymbol{h})$, $\boldsymbol{h} = \mathcal{T}_{\boldsymbol{\phi}_0}(\boldsymbol{x}_t)$
  Sample $\boldsymbol{z}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{z}_t}, \boldsymbol{\Sigma}_{\boldsymbol{z}_t})$
  $\boldsymbol{\mu}_{\boldsymbol{r}_t} = \mathcal{T}_{\boldsymbol{\theta}}([\boldsymbol{x}_t, \boldsymbol{z}_t])$
  Select action $a_t = \mathrm{argmax}_a \, \boldsymbol{\mu}_{\boldsymbol{r}_t}$
  Observe reward r_t
  $D_t = D_{t-1} \cup (\boldsymbol{x}_t, a_t, r_t)$
  if t mod $t_f = 0$ **then**
    for iteration $= 1 : t_s$ **do**
      Draw a minibatch data $\{(\boldsymbol{x}_i, a_i, r_i)\}_{i=1}^N$
      Expand r_i to vector $\boldsymbol{r}_i$ with 0 at unobserved positions
      Create mask $\{\boldsymbol{m}_i\}_{i=1}^N$, where $\boldsymbol{m}_i$ are zeros only except $\boldsymbol{m}_i[a_i] = 1$
      Obtain the action space dimension as $|\mathcal{A}|$
      $\boldsymbol{h}_i = \mathcal{T}_{\boldsymbol{\phi}_0}(\boldsymbol{x}_i)$, $\boldsymbol{\mu}_{\boldsymbol{z}_i} = \mathcal{T}_{\boldsymbol{\phi}_1}(\boldsymbol{h}_i)$, $\boldsymbol{\Sigma}_{\boldsymbol{z}_i} = \mathcal{T}_{\boldsymbol{\phi}_2}(\boldsymbol{h}_i)$ for $i = 1 : N$
      Let $\boldsymbol{z}_i := \boldsymbol{\mu}_{\boldsymbol{z}_i} + \boldsymbol{\Sigma}_{\boldsymbol{z}_i} \odot \boldsymbol{\varepsilon}_i$, $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for $i = 1 : N$, where $\odot$ denotes element-wise product
      $\boldsymbol{\mu}_{\boldsymbol{r}_i} = \mathcal{T}_{\boldsymbol{\theta}}([\boldsymbol{x}_i, \boldsymbol{z}_i])$ for $i = 1 : N$
      Update $\boldsymbol{\Sigma}_{\boldsymbol{r}}$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}_0$, $\boldsymbol{\phi}_1$, and $\boldsymbol{\phi}_2$ by using the gradients of

$$\frac{1}{N} \sum_{i=1}^{N} \left[\log p(\boldsymbol{r}_i \mid \boldsymbol{x}_i, \boldsymbol{z}_i) \cdot \boldsymbol{m}_i \cdot |\mathcal{A}| + \log \frac{p(\boldsymbol{z}_i)}{q(\boldsymbol{z}_i \mid \boldsymbol{x}_i)} \right]$$

    end for
  end if
end for

Algorithm 3 LU-SIVI: Thompson Sampling with Semi-Implicit Local Uncertainty

Input: Likelihood $p(\boldsymbol{r}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{r}_t}, \boldsymbol{\Sigma}_{\boldsymbol{r}})$; prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{z}})$; explicit variational distribution $q(\boldsymbol{z}_t \mid \boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\psi}_t, \boldsymbol{\Sigma}_{\boldsymbol{z}_t})$ with reparameterization $\boldsymbol{z}_t = \boldsymbol{\mu}_{\boldsymbol{z}_t} + \boldsymbol{\Sigma}_{\boldsymbol{z}_t} \odot \boldsymbol{\varepsilon}_t$, $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$; selected random noise distribution $q(\boldsymbol{\epsilon})$; neural network $\mathcal{T}_{\boldsymbol{\theta}}$, $\mathcal{T}_{\boldsymbol{\phi}_1}$, and $\mathcal{T}_{\boldsymbol{\phi}_2}$; $t_f = 20$ (training frequency), $t_s = 40$ (the number of mini-batches per training period)
Output: Fine-tuned parameters $\boldsymbol{\Sigma}_{\boldsymbol{r}}, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2$.
Initialize parameters $\boldsymbol{\Sigma}_{\boldsymbol{r}}, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2$.
Initialize dataset $D_0 = \emptyset$
for $t = 1, \ldots, T$ **do**
  Observe new context $\boldsymbol{x}_t$
  $\boldsymbol{\psi}_t = \mathcal{T}_{\boldsymbol{\phi}_1}([\boldsymbol{x}_t, \boldsymbol{\epsilon}_t])$, where $\boldsymbol{\epsilon}_t \sim q(\boldsymbol{\epsilon})$
  $\boldsymbol{\Sigma}_{\boldsymbol{z}_t} = \mathcal{T}_{\boldsymbol{\phi}_2}(\boldsymbol{x}_t)$
  Sample $\boldsymbol{z}_t \sim \mathcal{N}(\boldsymbol{\psi}_t, \boldsymbol{\Sigma}_{\boldsymbol{z}_t})$
  $\boldsymbol{\mu}_{\boldsymbol{r}_t} = \mathcal{T}_{\boldsymbol{\theta}}([\boldsymbol{x}_t, \boldsymbol{z}_t])$
  Select $a_t = \operatorname{argmax} \boldsymbol{\mu}_{\boldsymbol{r}_t}$
  Observe reward r_t
  $D_t = D_{t-1} \cup (\boldsymbol{x}_t, a_t, r_t)$
  if $t \bmod t_f = 0$ **then**
    for iteration $= 1 : t_s$ **do**
      Draw a minibatch data $\{(\boldsymbol{x}_i, a_i, r_i)\}_{i=1}^N$
      Expand r_i to vector $\boldsymbol{r}_i$ with 0 at unobserved positions
      Create mask $\{\boldsymbol{m}_i\}_{i=1}^N$, where $\boldsymbol{m}_i$ are zeros only except $\boldsymbol{m}_i[a_i] = 1$
      Obtain the action space dimension as $|\mathcal{A}|$
      Let $\boldsymbol{\psi}_i^{(k)} := \mathcal{T}_{\boldsymbol{\phi}_1}([\boldsymbol{x}_i, \boldsymbol{\epsilon}_i^{(k)}])$, $\boldsymbol{\epsilon}_i^{(k)} \sim q(\boldsymbol{\epsilon})$ for $k = 0 : K$
      Compute $\boldsymbol{\Sigma}_{\boldsymbol{z}_i} = \mathcal{T}_{\boldsymbol{\phi}_2}(\boldsymbol{x}_i)$
      Let $\boldsymbol{z}_i := \boldsymbol{\psi}_i^{(0)} + \boldsymbol{\Sigma}_{\boldsymbol{z}_i} \odot \boldsymbol{\varepsilon}_i$, $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
      Compute $\boldsymbol{\mu}_{\boldsymbol{r}_i} = \mathcal{T}_{\boldsymbol{\theta}}([\boldsymbol{x}_i, \boldsymbol{z}_i])$
      Update $\boldsymbol{\Sigma}_{\boldsymbol{r}}, \boldsymbol{\theta}, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2$ by using the gradients of:

$$\frac{1}{N} \sum_{i=1}^{N} \left[\log p(\boldsymbol{r}_i \mid \boldsymbol{\mu}_{\boldsymbol{r}_i}, \boldsymbol{\Sigma}_{\boldsymbol{r}}) \cdot \boldsymbol{m}_i \cdot |\mathcal{A}| + \frac{\log p(\boldsymbol{z}_i)}{\log \frac{1}{K+1} \sum_{k=0}^{K} q(\boldsymbol{z}_i \mid \boldsymbol{\psi}_i^{(k)}, \boldsymbol{\Sigma}_{\boldsymbol{z}_i})} \right]$$

    end for
  end if
end for

References

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. In *International Conference on Learning Representations*, 2015.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Riquelme, C., Tucker, G., and Snoek, J. Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyYe6k-CW.

Sun, S., Zhang, G., Shi, J., and Grosse, R. Functional variational Bayesian neural networks. In *International Conference on Learning Representations*, 2019.

Zhang, R., Wen, Z., Chen, C., and Carin, L. Scalable Thompson sampling via optimal transport. In *Artificial Intelligence and Statistics*, 2019.