ECS170 Final Project: Deep Learning for Meta-Statistical Inference

Brandon Bayquen, Sophia Brady, Adrian Javier, Luuk Janssen, James Moran, Radish Paik, Angela Traugott, Nadav Weinberger

Introduction

Our project investigates the ability of supervised machine learning algorithms to identify the probability distribution underlying a given data sample. We have implemented a feedforward neural network which is trained with synthetically generated data and evaluated on a combination of classification and regression metrics. Using this problem as a sandbox, we explore how multitask networks can form shared representations. This project was chosen both for the advantages provided by synthetic data and because artificial intelligence–driven statistical tools appear to be underexplored.

Background

Many single-variable probability measures are uniquely defined by two parameters, which can be transformed into mean and standard deviation; our network predicts these in addition to a class of distribution based on a random sample of 30 points. Defining distributions using mean and standard deviation allows for a meaningful relationship between samples and the target labels. Additionally, the network identifies the support, the subset of $\mathbb{R}$ on which the probability density is positive.

Calculating the mean of a sample is a linear operation and can thus be carried out by a neural network with no hidden layers. In contrast, sample standard deviation is nonlinear, and thus potentially challenging for neural networks. Statistical methods such as Q-Q plots can estimate whether a sample is drawn from a given distribution, but are fallible and require an a priori hypothesis. Since the space of probability measures is uncountable, the general case of this problem is computationally intractable. We therefore simplified the problem to nine classes: Beta, Gamma, Gumbel, Laplace, Logistic, Lognormal, Normal, Rayleigh, and Wald. These distributions were chosen to span a variety of probability distributions, as well as to contain sufficiently similar distributions as to pose a challenge, given the complexity of goodness-of-fit tests. A naive approximation of intuitive human performance on our classification task estimates that accuracy should be no higher than ⅓. In virtue of their respective difficulties, regression tasks were treated as stepping stones to the main challenge of classification.

Methodology

Our synthetic data was generated using NumPy. For each piece of data, a distribution family was first uniformly randomly selected. If that distribution had a support of $\mathbb{R}$, the mean was randomly drawn from the standard normal distribution. If the support was strictly positive, then the mean was randomly drawn from a lognormal distribution with mean and standard deviation of 1. Exceptions to this were the Beta distribution, which has a support of [0,1], and Rayleigh, which is determined by only the mean. Generating distribution parameters in this way ensured the parameters of our data were unbounded, enabling generalization to real-world datasets. Synthetic data also meant that class imbalance was not a concern and that the train-validation loop could persist ad infinitum without risk of overfitting, since fresh data could always be generated.

Each piece of data was labelled with a one-hot vector representing the distribution class, concatenated with the mean and standard deviation. The “features” associated with the label were thirty points randomly drawn from the distribution defined by the label, the choice of thirty following the classic rule

of thumb for application of the central limit theorem. Multidimensional data was generated by repeatedly choosing labels as before, drawing tuples from the set of distributions defined by the labels, and concatenating the labels and points respectively. Since the input to the network was featureless in the sense that all input nodes are conceptually identical, the network needed to first extract features from the input before making its predictions. We later present evidence that the network can be forced to extract specific features from the data, forming more robust representations and increasing accuracy.

We chose to implement our model in PyTorch for fine control of our model's parameters and architecture. Specifically, PyTorch is capable of automatically calculating gradients for custom loss functions, and its Module class allows for branching network structures, both of which are crucial for multitask networks. The network's loss function was an average of root mean squared error for the regression tasks and cross-entropy for the classification task. Research uncovered the surprising fact that optimizing mean average error over a dataset converges to the median, whereas optimizing mean squared error converges to the mean. Empirically, ReLU activations gave the best results.

We experimented with a number of small tweaks. Label smoothing increased classification accuracy, but techniques like class weighting based on difficulty were ineffective. Choice of optimizer had a large impact on performance: stochastic gradient descent and other algorithms which use only first-order gradients converged far too slowly to be viable, but some second-order algorithms like Adam displayed spurious early-stopping behavior. Adamax is designed to handle extreme gradients, such as the space of probability measures, and performed the best for this task. Adding warmup epochs allowed for a higher average learning rate without early destabilization, and cosine annealing in conjunction with early stopping made the model less sensitive to epoch count; training was stopped early if validation loss failed to improve by more than 10E-4 for three consecutive epochs. During training, models were evaluated on regression R^2 scores and classification accuracy. After each run, MAE and RMSE were calculated for regression tasks, and precision, recall, and F1-scores were calculated for classification tasks.

Visualizations in Matplotlib proved to be indispensable for tuning our model. Plotting the performance metrics across different values of a hyperparameter allowed for straightforward tuning and quantitative evaluation; graphing the actual distribution, the points drawn from it, and the model's prediction on the same plot allowed for qualitative evaluation. A crucial breakthrough in determining network architecture came from heatmaps and cumulative density plots of layer activations. Layers which were too wide displayed sparse activation, whereas layers which were too narrow (acting as an informational bottleneck) showed high levels of activation throughout. Well-fit cumulative activation density plots also displayed distinct plateaus – we hypothesize that these plateaus correspond to individual tasks.[1]

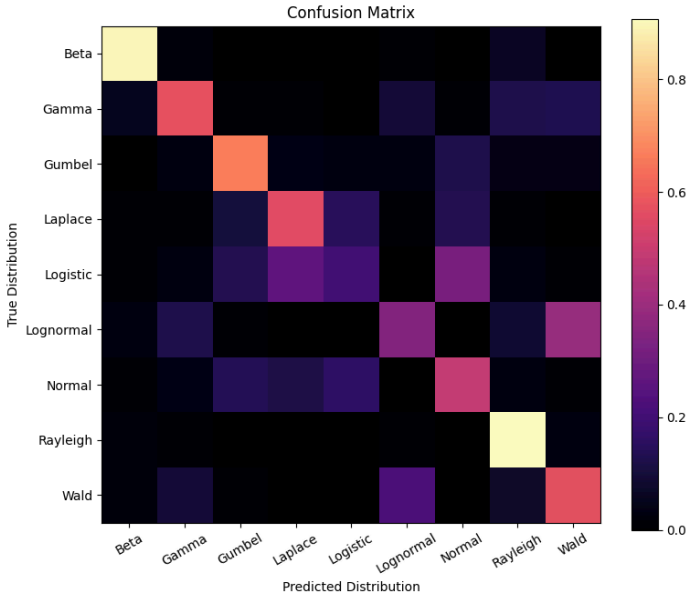
Results

Our network had mixed success. A neural network with no hidden layers can trivially calculate the mean of 30 sample points by setting each weight to 1/30, and the R^2 (coefficient of determination) between sample and population means with sample size 30 can be bounded below at 0.9667. Our model outperforms this, scoring 0.971, likely due to input from the classification task. However, our model's performance on standard deviation was disappointing, with R^2=0.775 significantly below the performance of mathematical methods. This can be explained by the nonlinear operations required to calculate standard deviation – a network with quadratic activation functions could trivially compute the

[1] Additional figures, 1.1-1.4

sample standard deviation, as a linear network can compute the sample mean. Finally, classification accuracy reached 0.576, which exceeds our estimate of human performance.

Performance between distributions varied. The model excels at identifying Beta and Rayleigh distributions, which are both unique: Beta is the only class with a support of [0,1], and Rayleigh is determined by only 1 parameter. Our model succeeds in distinguishing some cases where a human eye or Q-Q plot would fail, such as between Rayleigh and Gamma distributions. Sources of error are localized mostly in specific pairs of distributions, such as Logistic vs. Normal and Lognormal vs. Wald. Despite their prevalence, these *types* of errors indicate coherent performance: Logistic and Normal distributions are both symmetric, with similar curvature and identical support; Lognormal and Wald are similarly alike.



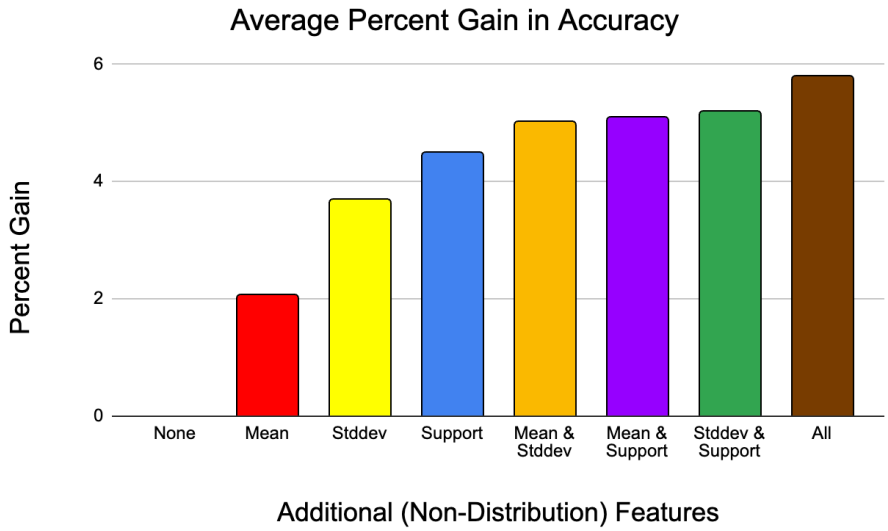
Normalized confusion matrix

Our final network had three hidden layers of sizes 256, 128, and 128, and was trained in under ten minutes on one million pieces of data with a batch size of 1000. Training lasted for 60 epochs, 5 of which were warmups to a max learning rate of 0.01. Keeping all other hyperparameters constant and doubling the size of each layer, the network generalized well to multiple dimensions. Classification accuracy decreased by 3% per dimension, while the regression tasks were largely unaffected.[2]

Discussion

Our findings demonstrate how neural networks can form and utilize shared representations. We discovered that training the model on the support of each distribution increased accuracy. We then trained the network with and without each feature, and found that each additional feature resulted in a modest but consistent accuracy improvement, showing that the network could extract meaningful relationships between features. It stands to reason that this trend would continue with the incorporation of more features.

[2] Additional figures, 2.2



Training the network on additional features increases classification accuracy.

We explored how task-specific hidden layers could increase performance, but found them to be suboptimal. We hypothesize that eliminating the hidden layers in task-specific heads forces the model to create good intermediate representations of the model in its backbone nodes.

Existing statistical techniques like maximum likelihood estimation and Q-Q plots can only determine how well a sample fits a given distribution, not which distribution best fits a sample. Our model identifies patterns in the data that indicate the underlying distribution family; this leads it to identify Rayleigh distributions even where Q-Q plots would fail.[3] Rayleigh distributions are defined uniquely by their mean, and neural nets take advantage of this reduced complexity where classical statistical methods could not.

Given more time, we would have asked humans to guess the distribution family and parameters of 30 sample points. This would have provided a more realistic expectation for model performance and allowed us to show how the model surpasses human ability. We also would have implemented additional features such as mode, kurtosis, skewness, and entropy to enhance the model's shared representations. Kurtosis in particular could have helped it distinguish between Normal and Logistic curves, which is currently one of its weakest points.

Conclusion

We have delivered a proof-of-concept that artificial intelligence can be an effective tool in statistics. While more training and optimization is possible, our model shows that neural networks are capable of inferences that traditional statistics struggle with. We have also presented evidence for the existence and efficacy of shared representations in multitask networks. We hope that with further improvements, similar models could be deployed into industry settings to assist researchers, data scientists, and students.

[3] Additional figures, 3.1

Contribution

- Sophia conducted early literature searches, assisted with hyperparameter tuning for learning rate, and did check-in and report outlining/drafting.
- Radish created the original prototypes and plotted the model's predicted distributions against the ground truth distributions for visual sanity checking; they also organized meetings.
- Angela worked on multidimensional data generation, classification performance metrics and plotting, and hyperparameter tuning for accuracy.
- Brandon provided starting code for Matplotlib visualizations and calculating classification metrics of the neural net as confusion matrices.
- Nadav helped with the design of the neural network, and with creating visualizations.
- Luuk contributed to various aspects of the code including testing and assisted with hyperparameter tuning and creating visualizations.
- James implemented code for data generation, model architecture, and optimizations.
- Adrian helped set up the presentation and did some research regarding parameter tuning.

Sources

Li, Chunyuan, et al. *Measuring the Intrinsic Dimension of Objective Landscapes*. arXiv, 2018. *DOI.org (Datacite)*, https://doi.org/10.48550/ARXIV.1804.08838.

Pennink, Ward. *WardPennink/Automatically_fitting_probability_distributions*. 2020. 7 June 2024. *GitHub*, https://github.com/WardPennink/Automatically_fitting_probability_distributions.

Nishiyama, Tomohiro. *Lower Bounds for the Total Variation Distance Given Means and Variances of Distributions*. arXiv:2212.05820, arXiv, 26 Dec. 2022. *arXiv.org*, https://doi.org/10.48550/arXiv.2212.05820.

Lu, Qihong, et al. *Shared Representational Geometry Across Neural Networks*. arXiv:1811.11684, arXiv, 16 Mar. 2019. *arXiv.org*, https://doi.org/10.48550/arXiv.1811.11684.

Shai, Adam S., et al. *Transformers Represent Belief State Geometry in Their Residual Stream*. arXiv:2405.15943, arXiv, 11 Nov. 2024. *arXiv.org*, https://doi.org/10.48550/arXiv.2405.15943.

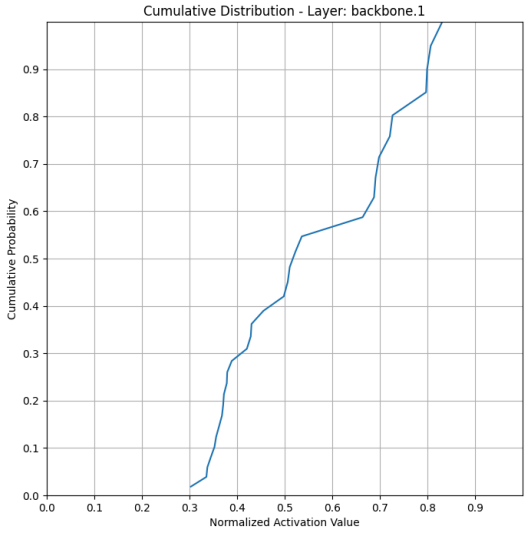
Cranmer, Miles, et al. *Discovering Symbolic Models from Deep Learning with Inductive Biases*. arXiv:2006.11287, arXiv, 18 Nov. 2020. *arXiv.org*, https://doi.org/10.48550/arXiv.2006.11287.

Liu, Ziming, et al. *Seeing Is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability*. arXiv:2305.08746, arXiv, 6 June 2023. *arXiv.org*, https://doi.org/10.48550/arXiv.2305.08746.

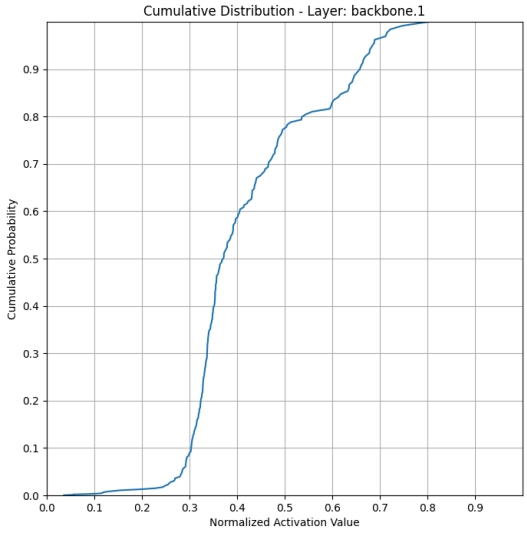
Swalin, Alvira. "Choosing the Right Metric for Evaluating Machine Learning Models — Part 1." *USF-Data Science*, 10 July 2018, https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4.

Additional Figures

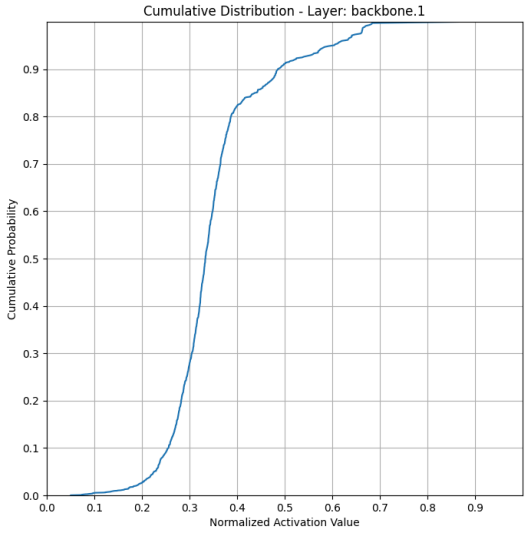
1. Layer activation density plots:



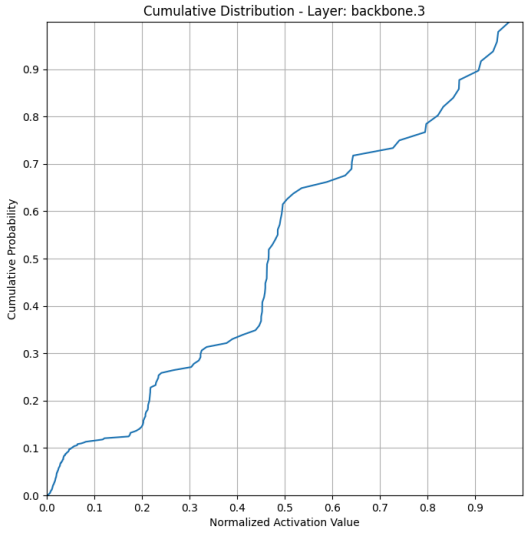
1.1 *A layer displaying information bottlenecking.*



1.2 A well-fit shallow layer with mild plateaus.

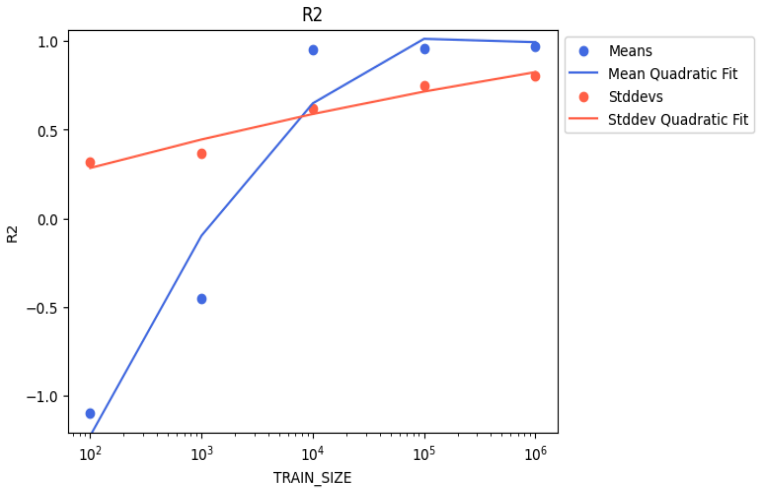


1.3 *A layer with overly sparse activation.*

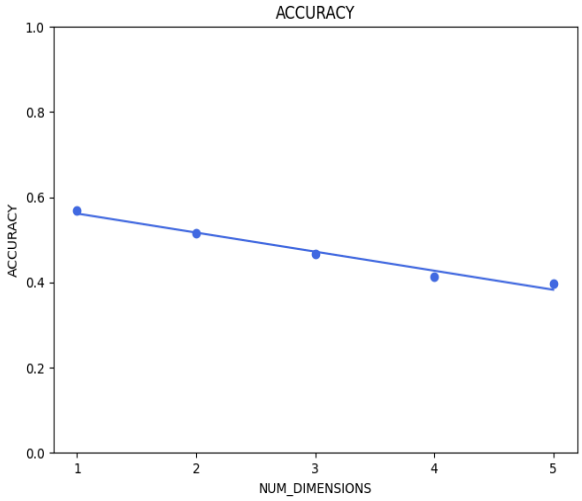


1.4 A well-fit deep layer with extreme plateaus

2. Performance:

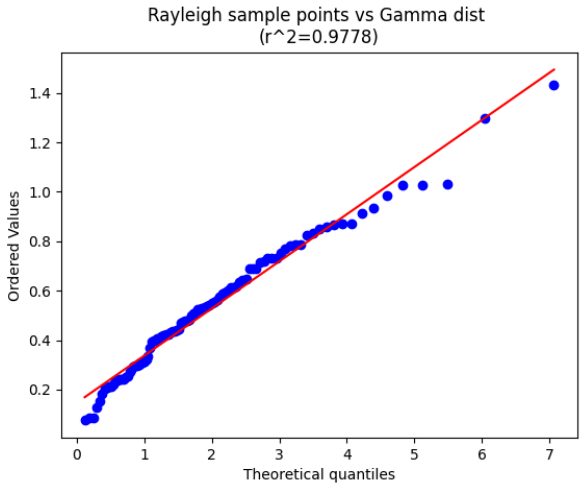


2.1 Performance on mean and standard deviation



2.2 Multidimensional performance

3. Failures of traditional statistical methods:



3.1 Q-Q plot exhibiting a Type I error