# Chemical Property Prediction with Stochastic Neural Networks

**Soren Dunn**
sorend2@illinois.edu

## Abstract

Kernel-expanded stochastic neural networks (K-StoNets) have several useful theoretical properties including (under certain assumptions) not having any local minima and having universal approximation ability, even in small networks. However, little work has yet been conducted on evaluating the empirical importance of each of the model components of K-StoNets in the context of a real-world prediction problem. In this project, I investigated the importance of the support vector regression (SVR) layer, hidden layer(s), added noise, and SVR hyperparameter choice in K-StoNets for model convergence and performance in the context of predicting activity of chemicals in inhibiting HIV replication using extended connectivity fingerprints. Code for this project is made publically available at https://github.com/sorendunn/Chemical-Property-Prediction-with-KStoNet

## 1 Introduction

Deep neural networks in recent years have shown remarkable performance in tasks ranging from image classification Krizhevsky et al. [2012] to bioinformatics Sapoval et al. [2022] to robotics Dong et al. [2021] due to their flexibility and potential for high performance given sufficient compute. However, state-of-the-art neural networks often require a highly complex model structure with millions or even billions of parameters. Additionally, successfully training deep neural networks can be finicky as they are prone to becoming stuck in local minima.

To work towards solving these problems Sun and Liang [2022] proposed a novel model architecture termed kernel-expanded stochastic neural network (K-StoNet). This architecture was inspired by previous work by Gori et al. [1992] and Nguyen and Hein [2017] who showed that given particular regularity conditions, a pyramidal network structure, and linearly independent samples every critical point on a feed-forward neural network will be a global minima. Specifically they showed that for a feed forward neural network with $h$ hidden layers given:

1. **Input vector:** $Z_0 = X \in \mathbb{R}^{m_0}$ represents the input to the neural network, where $m_0$ is the input dimension.
2. **Hidden and output layers:** For $i$ from 1 to $h+1$, the output of layer $i$, $Z_i \in \mathbb{R}^{m_i}$, is computed as
$$Z_i = \Psi(w_i Z_{i-1} + b_i),$$
where $w_i \in \mathbb{R}^{m_i \times m_{i-1}}$ are the weights, $b_i \in \mathbb{R}^{m_i}$ the biases, and
$$\Psi(s) = (\psi(s_1), \ldots, \psi(s_{m_i}))^T$$
applies the activation function component-wise.
3. **Activation function:** $\psi(\cdot)$ is real analytic, strictly monotonically increasing, and satisfies either a bounded condition or a growth condition given by certain positive constants $\rho_1$, $\rho_2$, $\rho_3$, and $\rho_4$.

4. **Loss function:** $U(\theta)$ represents the loss function of the neural network. It is given by the negative log-likelihood or log-probability of the target output $Y$ averaged over the $n$ training samples (index $i$ references the training sample):

$$U(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log \pi(Y^{(i)}|\theta, X^{(i)}) \equiv \frac{1}{n}\sum_{i=1}^{n} l(Z_{h+1}^{(i)}),$$

where $\pi(\cdot)$ denotes the density/mass function, $n$ is the sample size, and $l(\cdot)$ is a continuously differentiable function.

5. **Regularity Conditions:** Training samples are linearly independent, the activation function $\psi$ is real analytic, strictly increasing, and satisfies the bounded or growth conditions, and the loss function $l$ is continuously differentiable and if its derivative evaluated at 'a' is zero, then 'a' is a global optimum.

every critical point of the loss function $U(\theta)$ is a global minimum.

Such a result is possible because distinct, linearly independent training samples ensure that the input information is sufficiently diverse, full row rank of weight matrices implies that the layer transformations are capable of capturing the necessary complexity, and the activation function being real analytic and strictly increasing ensures well-behaved gradients work together to ensure that there are no stationary points or local minima. The problem with this result is that it requires the data to be linearly independent which inherently requires the sample size not be larger than the number of features. For most deep learning problems this is widely unrealistic as a datasets of thousands or millions of training samples is required to achieve state-of-the-art performance. The K-StoNet archictecture was proposed to remove this constraint.

The way K-StoNets attempted to circumvent the required linear independence of the data was by adding a radial basis function (RBF) support vector regression (SVR) on the first layer of the network to map the input data into an infinite dimensional space. The intuition behind why this can help satisfy the linear independence constraint is if the data has an arbitrary number of dimensions then the number of samples will never be larger than the feature dimension, ensuring the data's linear independence. The technical explanation is that in the infinite dimensional feature space the Gram matrix is of full rank which means the samples are linearly independent Sun and Liang [2022].

To see how the RBF kernel can be considered to map the input data into an infinite number of dimensions consider the RBF equation:

$$K(x, y) = \exp(-\gamma\|x - y\|^2). \tag{1}$$

Here, the parameter $\gamma > 0$ controls the width of the Gaussian function, dictating how quickly the similarity measure decreases with distance. The norm $\|x - y\|^2$ represents the squared Euclidean distance between the two feature vectors $x$ and $y$. This can be expanded using the dot product:

$$\|x - y\|^2 = (x - y) \cdot (x - y) = \|x\|^2 - 2x \cdot y + \|y\|^2. \tag{2}$$

Thus, substituting this back into the definition of the RBF kernel, we get:

$$K(x, y) = \exp(-\gamma\|x\|^2 + 2\gamma x \cdot y - \gamma\|y\|^2). \tag{3}$$

This reformulates the RBF kernel in terms of the squared norms of $x$ and $y$ and their dot product.

The exponential function can be expanded using the Taylor series:

$$\exp(z) = \sum_{n=0}^{\infty} \frac{z^n}{n!}, \tag{4}$$

which applies to any real or complex number $z$. Taking $z = 2\gamma x \cdot y$, the Taylor series for the central term of our RBF kernel becomes:

$$\exp(2\gamma x \cdot y) = \sum_{n=0}^{\infty} \frac{(2\gamma x \cdot y)^n}{n!}. \tag{5}$$

We can now substitute this expansion into our RBF kernel formulation, separating the part that depends on $x$ and $y$ alone:

$$K(x, y) = \exp(-\gamma \|x\|^2) \cdot \left( \sum_{n=0}^{\infty} \frac{(2\gamma x \cdot y)^n}{n!} \right) \cdot \exp(-\gamma \|y\|^2). \tag{6}$$

This final expression shows that the RBF kernel can be understood as an infinite series expansion of the weighted dot products of $x$ and $y$. Each term of this series represents an interaction of a different degree between $x$ and $y$. This derivation is one way to see that the RBF kernel maps the data into an infinite dimensional embedding and thus ensures samples are linearly independent.

However, applying this transformation means that the model can no longer be trained using a gradient-based algorithm since the transformed input features aren't explicitly calculated. One approach to circumventing this difficulty would be to use the kernel representer theorem to try to train the model based on the kernel of corresponding to the transformed data:

$$\tilde{Y}_1 = b_1 + \sum_{i=1}^{n} w_1^{(i)} K(\mathbf{X}^{(i)}, \mathbf{X}) \tag{7}$$

where $w_1^{(i)} \in \mathbb{R}^{m_1}$ and $K(\mathbf{X}^{(i)}, \mathbf{X}) = \phi(\mathbf{X}^{(i)})^T \phi(\mathbf{X})$ is explicitly available. The authors term the above model a kernel-expanded neural network (KNN). The issue with this model specification though is that for a nonlinear kernel regression, the kernel representer theorem may not hold. This is due to the fact that depending on the chosen activation function (which always involves a nonlinear mapping), the solution cannot be easily expressed as a linear combination of kernel functions based on training data. Instead, the mapping $g(\cdot)$ could potentially create a much more complex boundary that can not be represented as a linear combination of kernels.

### 1.1 Model Specification

To avoid this issue Sun and Liang [2022] propose the K-StoNet model:

$$Y_1 = b_1 + \beta\phi(X) + e_1 \tag{8}$$

$$Y_i = b_i + w_i \Psi(\tilde{Y}_{i-1}) + e_i \; for \; i = 2, 3, ..., h \tag{9}$$

$$Y = \Psi'(b_{h+1} + w_{h+1}\Psi(Y_h) + e_{h+1}) \tag{10}$$

The variables in the K-StoNet model given above can be split into 3 types:

1. **Data and Predictions**: $X$ is the input data, $Y$ is the model predictions.
2. **Hyperparameters**: $h$: Number of hidden layers, $\psi$: Radial basis function (RBF) kernel, $e_1, \ldots, e_{h+1}$: Random noise, $\Psi$: Activation function, $\Psi'$: Logit function, $\varepsilon_i$ are random noise added with distribution $N(0, \sigma_i^2 I_{m_i})$ where $I_{m_i}$ is an identity matrix with dimensions equal to the number of units on layer $i$ and $\sigma_i$ is the chosen noise level for layer $i$, $g$ is a link function to convert the output to a format suitable for classification.
3. **Learned Parameters**: $b_1, \ldots, b_{h+1}$ are learned parameters (the biases), $\beta$ is the learned weight for the first layer, $w_1, \ldots, w_{h+1}$ are learned parameters (the weights), $Y_1, \ldots, Y_h$ are latent (unobserved) variables defined as shown in the equations above.

The model is thus composed of an RBF kernel SVR, a lasso for each of the hidden layers (a penalty is added to ensure convexity) and a generalized linear model for the output layer (in order to allow for classification).

### 1.2 K-StoNet as an Approximation of KNN

To establish that $K$-StoNet is a valid approximator to KNN, the authors make a series of assumptions to establish the validity of the K-StoNet including:

(i) $\Theta$ is compact. This means the set of all parameter vectors $\theta$ is bounded and closed.

(ii) The expectation of the logarithm squared of the conditional probability is finite, ensuring reasonable behavior of the loss.

(iii) The activation function $\psi(\cdot)$ is Lipschitz continuous, indicating a bounded rate of change.

(iv) The network can grow with the size of training data.

(v) A constraint on the noise $\sigma_n$ that ensures the noise does not overwhelm the signal in the network's predictions.

They use these assumptions to show that K-StoNet and KNN will have matching loss functions in the limit as the sample size grows without bound.

They next make the following assumptions:

- $Q^*(\theta)$ is continuous in $\theta$ and has a unique maximum.
- There's a positive gap between the maximum value of $Q^*(\theta)$ and any values outside a certain bound.

to ensure that the objective function has a well-defined maximum and is not too 'flat' or discrete.

They use these additional assumptions to prove that the estimated parameter vector $\hat{\theta}_n$ will converge in probability to the true parameter $\theta^*$, meaning that the K-StoNet consistently approximates the true model as the number of samples increases.

### 1.3 Training Algorithm

The authors next propose an approach for training the K-StoNet utilizing the Imputation-Regularized Optimization (IRO) algorithm introduced by Liang et al. [2018] to address the challenge of missing data in the context of statistical learning. $Z_{obs}$ is defined as the observed data, $Z_{mis}$ as the missing data, and $\theta$ as the model parameter. The primary objective of the IRO algorithm is to obtain a consistent estimate of $\theta$ by maximizing the expected log-likelihood $E \log \pi(Z_{obs}, Z_{mis}|\theta)$, with the expectation taken over the joint distribution of $(Z_{obs}, Z_{mis})$.

The IRO algorithm shares similarities with the Expectation-Maximization (EM) algorithm and the Stochastic EM algorithm, which focus on maximizing the marginal likelihood $\pi(Z_{obs}|\theta)$. However, IRO differentiates itself by incorporating a regularization term in the optimization step to ensure convergence, especially in high-dimensional settings.

For the K-StoNet, the goal is to estimate $\theta$ by maximizing $E \log \pi(Y, Y_{mis}|X, \theta)$. The IRO algorithm entails the following iterative steps starting from an initial guess $\hat{\theta}_n^{(0)}$:

**I-step** For each sample $(X^{(i)}, Y^{(i)})$, impute $Y_{mis}^{(i)}$ from the predictive distribution $g(Y_{mis}|Y^{(i)}, X^{(i)}, \hat{\theta}^{(t)})$.

**RO-step** Update the estimate $\hat{\theta}_n^{(t+1)}$ by minimizing a penalized loss function

$$\hat{\theta}_n^{(t+1)} = \arg\min_{\theta} \left( -\frac{1}{n} \sum_{i=1}^{n} \log \pi(Y^{(i)}, Y_{mis}^{(i)}|X^{(i)}, \theta) + P_{\lambda_n}(\theta) \right),$$

where $P_{\lambda_n}(\theta)$ is a chosen penalty function to form a consistent estimate of $\theta$.

in pseudocode:

```
Algorithm 1: Outline of Training K-StoNet
1: Initialize parameters ...
2: Repeat until convergence {
```

```
3:     Perform I-step: Impute missing data
4:     Perform RO-step: Update parameters by minimizing penalized loss function
5:     Check convergence criteria
}
6: Output final model parameters
```

Despite the potential higher cost for iteratively solving convex optimization problems, IRO typically converges within tens of iteratoins, which may offset the high computational cost.

## 2 Dataset

For training the K-StoNets we utilized a dataset originally compiled by the Drug Therapeutics Program's (DTP) AIDS Antiviral Screen, which assessed the capacity of over 40,000 chemical compounds to inhibit the replication of the Human Immunodeficiency Virus (HIV). We accessed this dataset through the MoleculeNet benchmark created by Wu et al. [2018] which provides access to a variety of datasets for molecular analysis. The antiviral screen tested all the compounds in the dataset and their effectiveness was classified into one of three distinct categories, namely confirmed inactive (CI), confirmed active (CA), and confirmed moderately active (CM). For the purpose of our analysis, we merged the active (CA) and moderately active (CM) categories to simplify the task into a binary classification challenge, differentiating between inactive (CI) and active (CA/CM).

The raw dataset provided by MoleculeNet provided Simplified Molecular Input Line Entry System (SMILES) representation strings to identify each compound as well as the activity classification of each compound.

We converted the SMILES strings to Extended-Connectivity Fingerprints (ECFP) using RDKit (an open-source platform designed to support the chemical information processing) which is provided through the python package DeepChem. ECFP are a common way to represent chemical compounds in chemical informatics. Creating ECFP fingerprints works roughly as follows

1. **Molecule Decomposition:** A molecule is dissected into a series of submolecular fragments according to the connectivity of its atoms.

2. **Unique Atom Identification:** Every atom in the molecule is labeled with a distinctive identifier that reflects its type, chemical properties, and bonding context.

3. **Hashing:** The identifiers are hashed into a fixed-length binary array, or *fingerprint*, to facilitate efficient data handling and comparison.

The dataset automatically applied scaffold splitting (splitting which ensures compounds which share largely the same structure are split into the train or test set together). This provides a more stringent evaluation of the predictive model's generalizability. The dataset is composed of 32,901 training samples and 4,113 testing samples.

Unfortunately the molecular activies in the dataset are very imbalanced: only around 1,300 total molecules are classified as inhibiting HIV replication. We addressed the large imbalance in the dataset both in our approach to training and in our approach to model evaluation. On the training side, we employed under-sampling and over-sampling procedures to attempt to balance the classes in the dataset. In terms of model evaluation, we used the area under the receiving operator characteristic (AUC-ROC) as the evaluation metric as opposed to accuracy. The receiving operator characteristic provides a better evaluation of the true positive rate false positive rate trade-off provided by the model than simple accuracy. This is especially important in an imbalanced dataset since it is common for models to end up classifying all the data as either one class or another, resulting in either an extremely high false positive rate or an extremely high false negative rate while sacrificing the other. Since ROC curves are a good method for evaluating imbalanced datasets, we employed the AUC-ROC as a method of quantitatively comparing ROC curves.

## 2.1 Data Preprocessing

### 2.1.1 Sampling

To increase the model's training performance on active compounds (the minority class) we tried employing both undersampling and oversampling procedures to balance the dataset. For undersampling, a random selection of samples which were inactive were discarded until there was an even number of active and inactive compounds.

For oversampling the SMOTE (Synthetic Minority Over-sampling Technique) method was applied. The SMOTE algorithm has the following specification:

$$\mathbf{s} = \mathbf{x} + (\mathbf{n} - \mathbf{x}) \times \delta$$

where $\delta$ is a random number between 0 and 1. This operation will yield a point along the line segment between $\mathbf{x}$ and $\mathbf{n}$ but not necessarily at one of the endpoints, thus creating diversity in the sample.

By repeating this process, the SMOTE algorithm generates new, synthetic examples of the minority class, allowing models to have a more balanced view of the problem and potentially improving classification performance on imbalanced datasets.

After oversampling, the dataset comprised of over 60,000 compounds. Training a 1 hidden layer K-StoNet with 5 hidden units took over 10 minutes to train for a single epoch and a 20-unit K-StoNet took over an hour to train for a single epoch. Additionally the performance of these K-StoNets were lower than the same models trained on the undersampled dataset for only a tenth of the time they took to train a single epoch. Thus due to a combination of computational constraints and low performance no further samples were evaluated on the oversampled dataset.

### 2.1.2 Principle Component Analysis

Due to the high number of features in the ECFP, we also applied principle component analysis (PCA) to the features and then selected a subset of the principle components to use for prediction to lower the feature dimension in the dataset. This allowed for further testing of specific model configurations and training of larger models over more epochs due to the reduced size of the dataset.

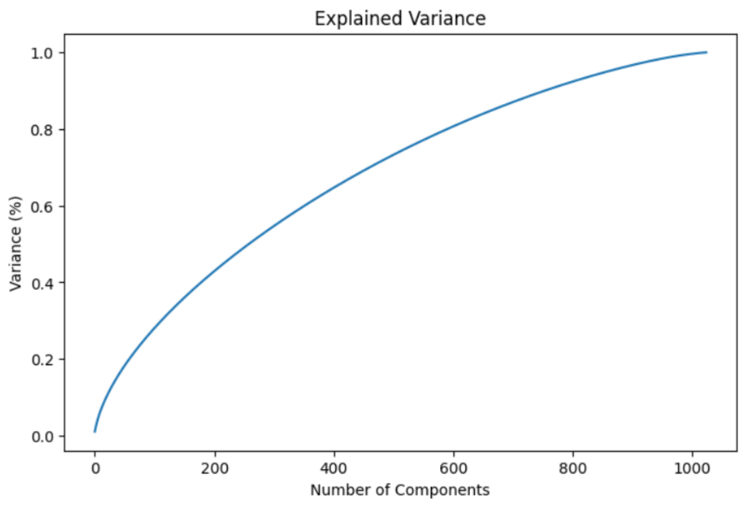

Figure 1: PCA Cumulative variance across all principle components

The percent of variance explained by the principle components increased approximately logarithmically with the number of components, resulting in there being no obvious cutoff for selection of principle components to use. Thus the top 100 principle components were chosen as over the top 100 components the increase in percent variance as the number of component increase is still slightly greater than linear and selecting the first 100 components decreases the size of the dataset by a factor of 10, which allows for a dramatic increase in computational speed of models trained on the dataset.

The loadings of the principle components were determined based on the testing set and the same transformation was applied to both the training and testing set. Two versions of the PCA subset dataset were created: one with the undersampled dataset and the other with the full dataset.

As a litmus test for whether subsetting the dataset to the top principle components effected trained model performance, we trained a simple 100 estimator gradient boosting classifier on the full dataset and the feature-selected version of the dataset only containing the first 100 principle componenets. The model trained on the full dataset achieved 0.71 test AUC-ROC and the model trained on the data subset achieved 0.70 test AUC-ROC. Additionally the model trained on the PCA subset actually achieved higher test set accuracy (0.942) than the model trained on the full dataset (0.94). These tests indictate that the subsetted version of the dataset is potentially able to train models to similar levels of performance while requiring much less computational time to run. For this reason, most of the model inference training runs are performed on a dataset subsetted to only the first 100 principle components and undersampled to acheive class balance. This approach greatly increases the speed of training runs but has a potential cost in decreased model performance.

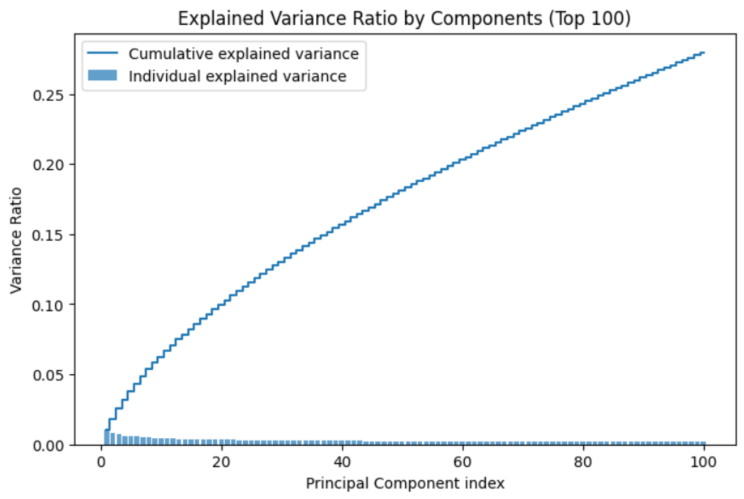

Figure 2: Plot of first 100 principle components which were selected for the feature subset dataset. All together the first 100 principle components make up around 25% of the total variance

## 3 Results

Five sets of analyses were conducted. The most basic of these was a basic performance comparison between the best-performing K-StoNet model, the best-performing traditional neural network, and hyper-parameter tuned shallow machine learning baselines from the literature. The remaining analyses were aimed at performing inference on the model structure of the K-StoNets to determine the impact of each of the major model choices made in their specification. Namely the choice of hyperparameters in the SVR layer, the inclusion of an initial SVR layer, inclusion of subsequent hidden layers, and addition of noise to each layer of the model. For each of these analyses, we analyzed how varying these model specifications effect the convergence and performance of these models in comparison with their theoretical guarantees.

The choice of hyperparameters in the initial SVR layer was tested through grid search of the initial SVR hyperparameters in a small K-StoNet without any hidden layers (which increased the computational efficiency of training so a wide range of combinations were able to be tested). The importance of the SVR layer was tested by training models with the same number of hidden dimensions but with either solely the SVR removed or the SVR removed combined with an additional dimension being added in order to counteract the loss of model complexity from simply removing the SVR layer. The importance of the subsequent hidden layers were evaluated by training versions of the initial layer with and without the subsequent hidden layer. The importance of the added noise to the model was evaluated by varying the added noise in the hidden layers of a small K-StoNet.

All models were trained on a T4 GPU with 15GB RAM in Google Colab. Note that all trained convention neural networks in the following results used the same L1 regularization as was applied in the hidden layers of the K-StoNets.

### 3.1 Performance Compared to Baselines

The AUC-ROC between the best performing of our K-StoNets, the best performing of our feedforward neural networks, and previous baselines Wu et al. [2018] are plotted in Figure 3. In terms of raw performance, even the shallow K-StoNet trained here (with only a single hidden layer) are competitive with the previously established baselines.

In these baselines, the feed-forward neural network, K-StoNet, and KernelSVM all display large amounts of overfitting (with training values between 0.2 and 0.25 higher than testing values). Disturbingly the K-StoNet seems just as prone to overfitting as the feedforward neural network (with training AUC-ROC 0.23 and 0.25 points larger than testing values respectively). This suggests that the K-StoNet may be converging to a minimum different from the underlying data-generating process which means it might be converging to a local minima instead of the global minima. If this is the case it is possible it is because the sample size is not large enough (since the proof that the K-StoNet is equivalent to the KNN only holds in the limit as n goes to infinity) or one of the other more technical conditions in the theoretical derivation may not be valid.

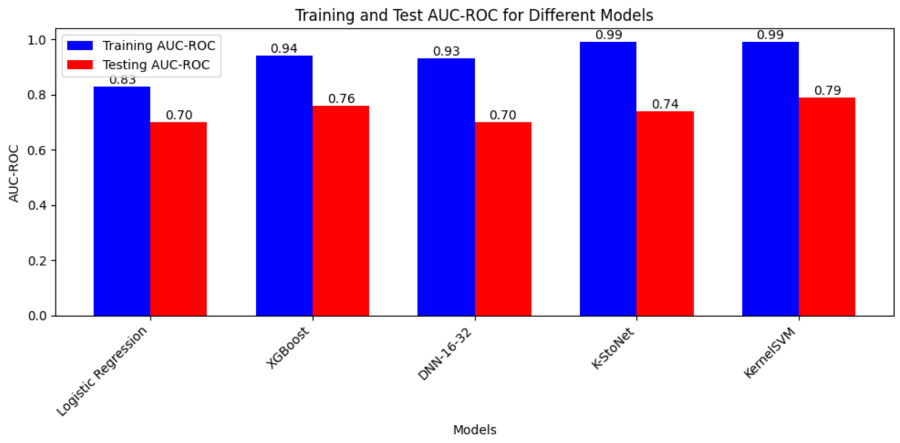

Figure 3: AUC-ROC comparison between K-StoNets and various machine learning baselines. DNN 16-32 refers to a feed forward neural network with two hidden layers of sizes 16 and 32. The K-StoNet is ours and the rest are previous baseline results from MoleculeNet

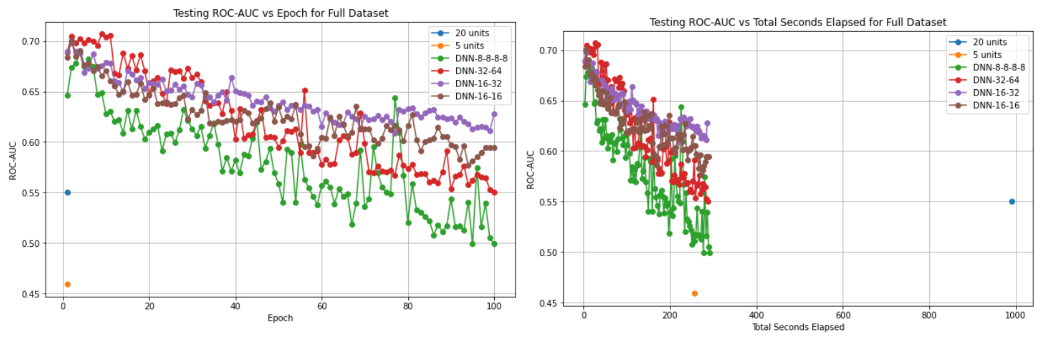

Figure 4: Testing AUC-ROC over the course of training for the full dataset. DNN refers to fully connected feed-forward networks with the specified number of hidden units. Lines which are not labeled as DNN refer to K-StoNets. Left shows results versus epochs while the right shows results over time

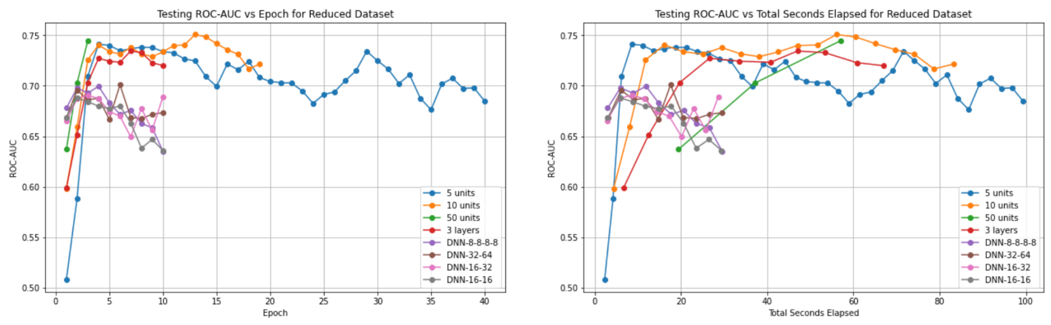

Figure 5: Testing AUC-ROC over the course of training for the undersampled dataset. DNN refers to fully connected feed-forward networks with the specified number of hidden units. Lines which are not labeled as DNN refer to K-StoNets. Left shows results versus epochs while the right shows results over time

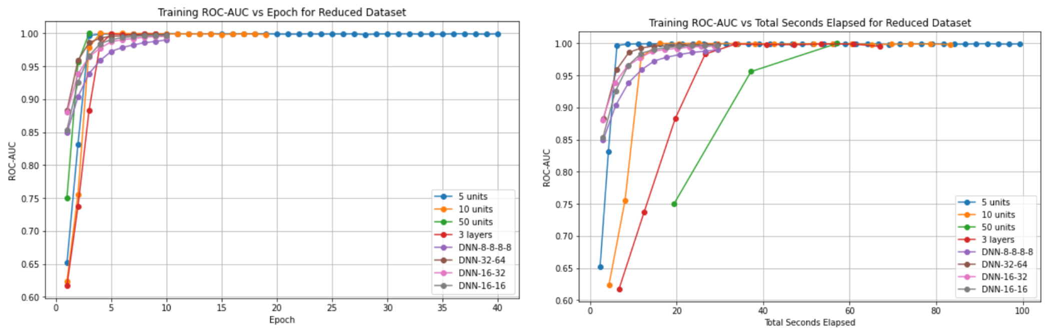

Figure 6: Training AUC-ROC over the course of training for the undersampled dataset. DNN refers to fully connected feed-forward networks with the specified number of hidden units. Lines which are not labeled as DNN refer to K-StoNets. Left shows results versus epochs while the right shows results over time

The lack of convergence to global minima is also supported by looking at the graphs of training and testing AUC-ROC over the course of training. The variation in the performance of the K-StoNets over the course of training appears very similar to the conventional feed forward networks. Additionally, the training performance of these models quickly increases to nearly one in less than ten epochs just as the regular feed forward networks do. However, as can be seen in the right side of Figure 6, this actually represents a longer total time to overfit the training data than the conventional feed forward networks. However overall based on this analysis the K-StoNets seem to have similar performance and convergence properties and feed forward neural networks trained over similar amounts of time. However these comparisons do not directly compare K-StoNet architecture to equivalent conventional neural network architecture, merely compare training K-StoNets and conventional neural networks over similar amounts of time/epochs. We performed a more direct comparison of K-StoNets and neural networks with similar architectures in Section 3.4.

## 3.2 SVR Parameter Search

The next analysis conducted was to identify how important the specification of the initial SVR was on the performance of the K-StoNet. Since the SVR is the most complicated component of the K-StoNet (and creates a whole new embedding of the data to feed into the next layer of the network) it is intuitively plausible that the exact specification of the SVR layer might greatly effect the model's results. However, as shown in Figure 7 the range of the maximum testing accuracy across all epochs of the trained SVR models over 3-4 orders of magnitude for each of their parameters only had around a 2% change in test set performance. This suggests that the exact specification of the SVR model likely does not have too dramatic of an effect on overall model performance. As shown in Figure 7, the best performing hyperparameter combinations were for low values of C regardless of epsilon.

In the SVR, C controls the strength of the L2 regularization where higher values of C correspond to lower regularization. Epsilon specifies the region where no penalty is incurred in the training loss function by points predicted within epsilon of their actual value. Thus the SVR performing best with small values of C indicates that regularization of the SVR is increasing its performance (up until around $10^-2$ where performance slightly drops off) but the exact specification of the no-penalty region is not particularly significant for model performance. Thus for the remaining analysis the value of C was chosen to be relatively small but otherwise little effort was put into exact hyperparameter configuration for the SVR layer since the model performance seemed decently robust to the hyperparameter specification.

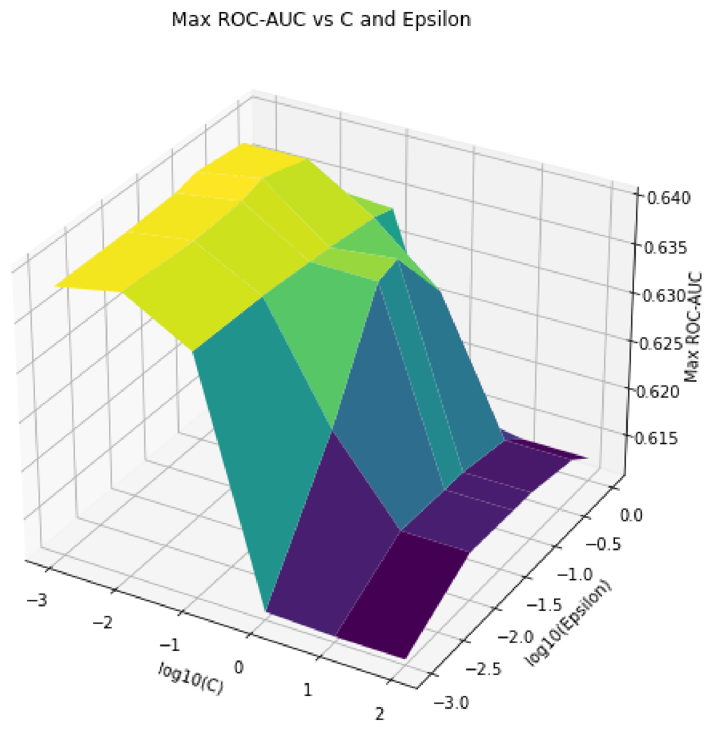

Figure 7: Maximum AUC-ROC from SVR models trained over 100 epochs plotted against their penalization coefficients trained on the PCA subsetted and undersampled dataset

## 3.3 K-StoNet Compared to SVR

The third analysis conducted was to test whether the additional model layers beyond the SVR significantly increased performance. This was especially of interest after seeing that a pure SVM baseline approach outperformed the best-performing K-StoNet on the testing data.

Additionally without the hidden layer the models seem to never really "learn" in the sense they never display significantly higher performance than their initial epoch. These results indicate that the hidden layer of the K-StoNet is essential both for its performance and convergence properties as was suggested by the earlier theoretical analysis.

## 3.4 K-StoNet Compared to Equivalent Neural Network Configuration

Since the hidden layers were previously shown to be central to both convergence and general performance of the K-StoNet the natural next step was to test whether the SVR layer was empirically useful to the model specification. To do this 1-layer 5-unit, 1-layer 10-unit, 1-layer 50-unit, and a 3-layer 15-unit-10-unit-5-unit K-StoNets were trained on the undersampled dataset over 50 epochs as shown in 10. Additionally, traditional neural networks with the same number of hidden layers and ones with an additional hidden layer of 5 units were also trained in the same way. Only testing

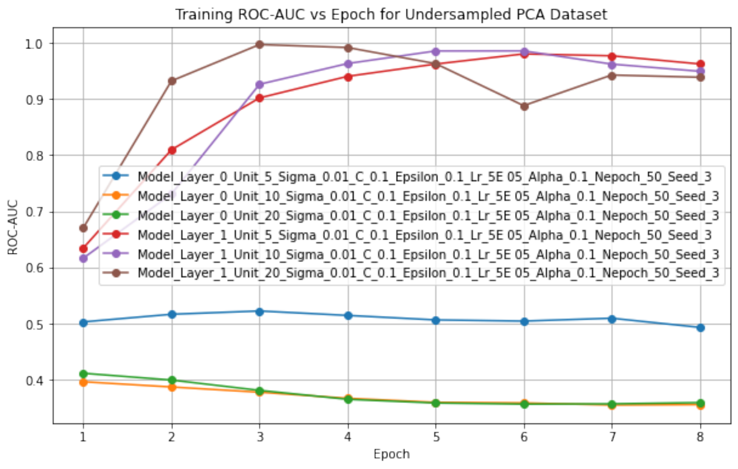

Figure 8: Training ROC-AUC for several K-StoNets either with a hidden layer or with their hidden layer removed evaluated on the undersampled PCA dataset

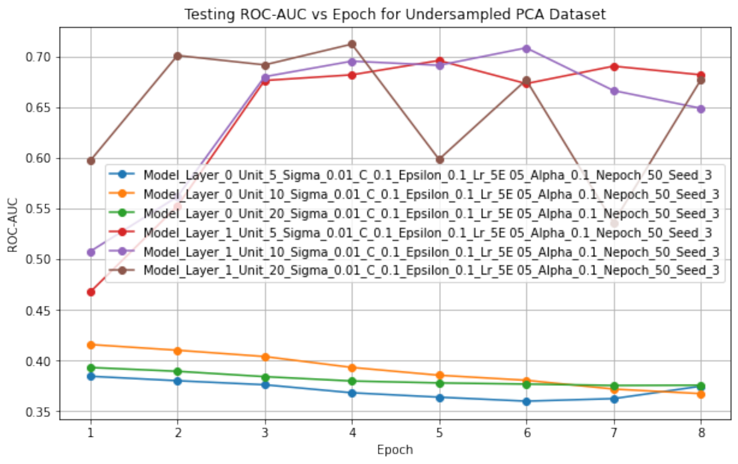

Figure 9: Test ROC-AUC for several K-StoNets either with a hidden layer or with their hidden layer removed evaluated on the undersampled PCA dataset

performance is shown since the training epochs mirror those of 6 (quickly converging to near perfect performance).

Comparing the K-StoNet performance with neural networks with the same number of hidden layers shows the K-StoNet's having uniformly greater performance, but with only around an average decrease in performance around 0.02 when removing the SVR, nearly an order of magnitude less of a drop in performance than from removing the hidden layer. Additionally comparing both the K-StoNets and the neural networks to their counterparts with an additional layer shows the networks with the additional layer achieving around the same performance as the traditional networks without the additional layer and in some cases they perform significantly worse. This is likely due to overfittig of the training data due to the increased model complexity.

One reason that this analysis might not be comparable with the previous one is that in this no errors were added to the hidden layers with the SVR removed while errors were still added in the case of removing the hidden layer form the SVR. However, the SVR analysis was also run with sigma at a negigile level and reproduced almost exactly the same results as shown in the previous section. Thus, these results indicate that the K-StoNet is more robust to removal of the SVR layer than to removal of the hidden unit layer(s).

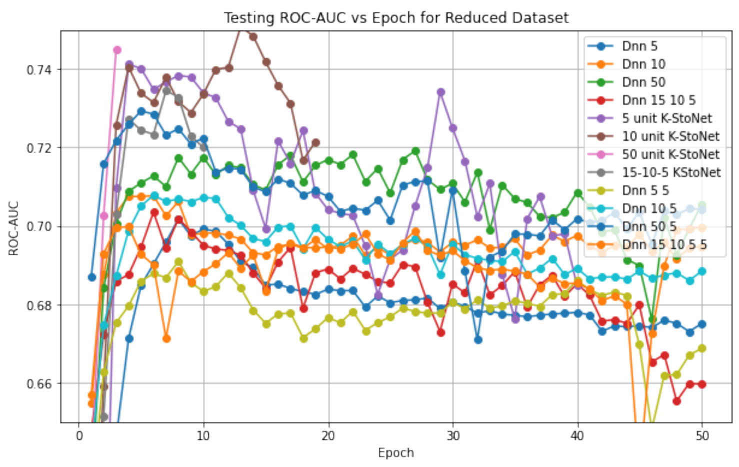

Figure 10: Testing performance on the undersampled dataset comparing K-StoNets with conventional neural networks with either the same hidden dimensions or the same hidden dimensions with an additional five unit hidden dimension to help counteract the decrease in model complexity from removing the SVR layer

## 3.5 K-StoNet Noise Specification

1-layer 5-hidden-unit K-StoNets were trained for ten epochs on the PCA subsetted and undersampled version of the dataset with varying levels of noise.

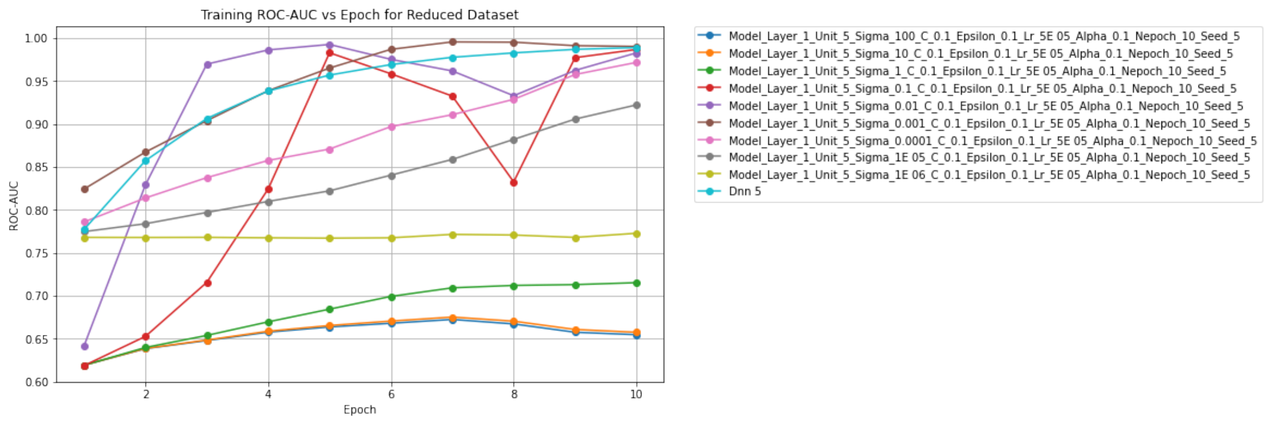

Figure 11: Training AUC-ROC for 1-layer five hidden unit K-StoNets with sigma varying from 100 to $10^{-6}$

The specific parameter that varied here was $\sigma_i$ which controls the random noise added to each layer (which has distribution $N(0, \sigma_i^2 I_{m_i})$). Thus larger $\sigma$ values imply greater noise added at each layer. Predictably, the training AUC-ROC is quite low when $\sigma$ is large (greater than 1). This is to be expected because in these cases the added noise is preventing the model from learning the underlying distribution. $\sigma$ values around 0.001 to 0.0001 achieve training performance comparable with a feed-forward neural network with the same number of hidden layers as the K-StoNet. As one continues to decrease $\sigma$ from 0.0001, the training performance starts to decrease again and for extremely small values ($10^{-6}$) training performance appears to never improve suggesting that for these extremely low values of the errors the network fails to converge.

The trends in the testing performance mirror that of the training performance. Very large values of sigma result in low testing performance while lower levels of noise are able to yield performance greater than the regular feed-forward network with the same number of hidden units and extremely small values result in the model failing to improve. This finding is supported by the fast that the

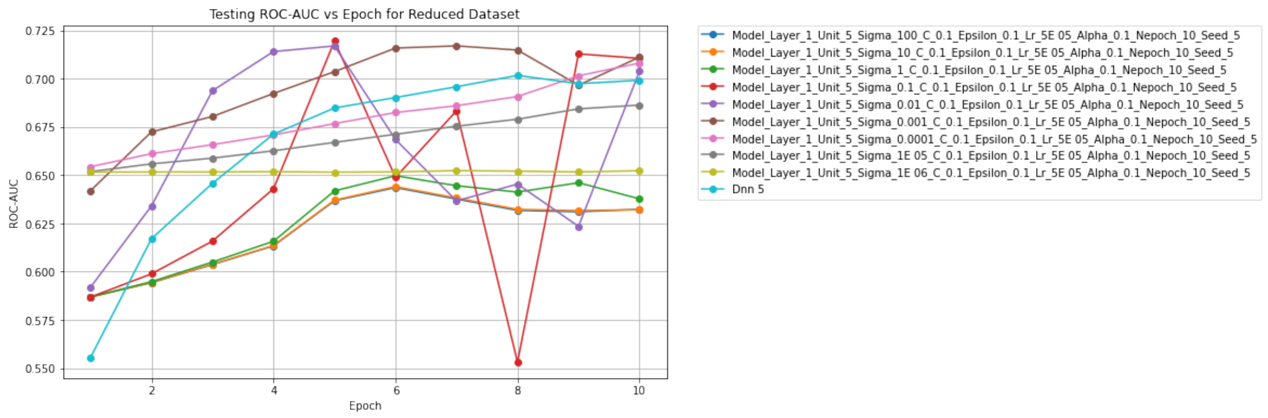

Figure 12: Testing AUC-ROC for 1-layer five hidden unit K-StoNets with sigma varying from 100 to $10^{-6}$

reason only 10 epochs are shown is that training began to be unstable after that for models with very low error levels.

These trends validate the theoretical result that the added noise is a necessary component of the K-StoNet architecture in order to allow for training but support the fact that the noise should be kept well below 1 to insure the model is still able to learn the underlying structure of the data. These findings support the fact that the K-StoNet is able to get comparable or even higher performance than conventional nerual networks when the errors are around 0.001 to 0.0001 for this dataset but for lower error levels begins to have trouble with training.

## 4 Conclusion

Kernel-expanded stochastic neural networks (K-StoNets) theoretically for large sample sizes include no local minnima and display universal approximating ability even with small networks as proven by Sun and Liang [2022]. However, K-StoNets display mixed empirical performance on chemical property prediction. Though they frequently converge in many fewer epochs than traditional networks each of those epochs takes correspondingly longer to train. They also in practice seem to display many of the issues of traditional neural networks including overfitting, converging to local minima, and hidden layer misspecification (with performance varying wildly depending on the exact model specifications including specification of the level of noise, number of hidden layers, ect. (though their performance seems robust to choice of SVR parameters)). Further analysis of the K-StoNet architecture shows that K-StoNets loose their performance and convergence properties when removing their SVR layer, but are resilient to removal of the SVR layer (which makes sense due to the well-known high performance of classic neural networks). Additionally, performance is maximized when setting the hidden-layer error values to around 0.001 with values too much larger than that leading to the signal in the data being drained out and values too much lower than that leading to issues in training.

These are several limitations of this analysis. On the theoretical end, one limitation of this analysis is that we were not able to evaluate the uncertainty quantification of the K-StoNets even though this was one of the main motivations for their use in the original paper. On the empirical end, the performance achieved lags behind current state-of-the-art baselines. An additional constraint is that though the dataset used in this project was drawn from a real-world prediction problem the combination of sampling and feature selection used to make the methods computationally feasible to run successive sets of runs on resulted in a relatively small dataset size (with only low thousands of samples). Additionally due to computational constraints the model sizes evaluated were relatively small in order to allow for comparisons between various variations of the models. Despite these limitations, these results provide important insights into the advantages, drawbacks, and practical training considerations when using K-StoNets on real-world datasets.

## References

Shi Dong, Ping Wang, and Khushnood Abbas. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021.

Marco Gori, Alberto Tesi, et al. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):76–86, 1992.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Faming Liang, Bochao Jia, Jingnan Xue, Qizhai Li, and Ye Luo. An imputation–regularized optimization algorithm for high dimensional missing data problems and beyond. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 80(5):899–926, 2018.

Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In *International conference on machine learning*, pages 2603–2612. PMLR, 2017.

Nicolae Sapoval, Amirali Aghazadeh, Michael G Nute, Dinler A Antunes, Advait Balaji, Richard Baraniuk, CJ Barberan, Ruth Dannenfelser, Chen Dun, Mohammadamin Edrisi, et al. Current progress and open challenges for applying deep learning across the biosciences. *Nature Communications*, 13(1):1728, 2022.

Yan Sun and Faming Liang. A kernel-expanded stochastic neural network, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning, 2018.