a (theoretical!) dumpster fire

projectx

nov 2023

[Annotations/additional explanatory comments made in blue.]
[Questions/points of confusion made in red.]
[Areas of further exploration in violet.]

1 Finite Sample Expressivity of ReLU Networks

1.1 Expressivity and Depth of ReLU Networks

This subsection contains the main relevant results from Yun et al., 2019:

Main Results We first state the main theorems on shallow FNNs showing tight lower and upper bounds on memorization capacity. Detailed discussion will follow in the next subsection.

[Note: When they refer to a k-layer net, they are including the output layer. Our current(?) FFN has 3 *hidden* layers, so it would be a 4-layer net for this paper. For notation: throughout this paper, d_i refers to the number of nodes in the i^{th} hidden layer of a fully connected net. d_y is the number of nodes in the classification layer of the net. For n-class classification tasks, $n = d_y$]

Assumption 3.1. In the dataset $\{(x_i, y_i)\}_{i=1}^N$ assume that all x_i's are distinct and all $y_i \in [-1, 1]^{d_y}$. [Essentially just that there are no repeats and that the labels are properly dimensioned. Also note that this is **regression, not classification** afaik since the labels are continuous vectors.]

Theorem 3.1. Consider any dataset $\{(x_i, y_i)\}_{i=1}^N$ that satisfies Assumption 3.1. If

- a 3-layer hard-tanh FNN f_θ satisfies $4\left\lfloor \frac{d_1}{2} \right\rfloor \left\lfloor \frac{d_2}{2d_y} \right\rfloor \geq N$; or
- a 3-layer ReLU-like FNN f_θ satisfies $4\left\lfloor \frac{d_1}{4} \right\rfloor \left\lfloor \frac{d_2}{4d_y} \right\rfloor \geq N$

then there exists a parameter θ such that $y_i = f_\theta(x_i)$ for all $i \in [N]$.

Theorem 3.1 shows that if $d_1 d_2 = \Omega(N d_y)$ then we can memorize arbitrary datasets; this means that $\Omega(\sqrt{Ndy})$ hidden nodes are sufficient for memorization, in contrary to $\Omega(Ndy)$ requirements of recent results. [Currently our FFN is really tiny? Like absurdly tiny? We don't even come anywhere close to satisfying the 2nd condition of the Theorem statement (LHS would be literally

just be 0 I think) so this may be a little trivial.] By adding one more hidden layer, the next theorem shows that we can perfectly memorize any classification dataset using $\Omega(\sqrt{N} + dy)$ hidden nodes.

Proposition 3.2. Consider any dataset $\{(x_i, y_i)\}_{i=1}^N$ that satisfies Assumption 3.1. Assume that $y_i \in \{0,1\}^{dy}$ [this *is* **multiclass classification**] is the one-hot encoding of dy classes. Suppose one of the following holds:

- a 4-layer hard-tanh FNN f_θ satisfies $4\lfloor \frac{d_1}{2} \rfloor \lfloor \frac{d_2}{2} \rfloor \geq N$, and $d_3 \geq 2d_y$; or
- a 4-layer ReLU-like FNN f_θ satisfies $4\lfloor \frac{d_1}{4} \rfloor \lfloor \frac{d_2}{4} \rfloor \geq N$, and $d_3 \geq 4d_y$.

Then there exists a parameter θ such that $y_i = f_\theta(x_i)$ for all $i \in [N]$.

Theorem 3.3. Consider FNNs with $d_y = 1$ and piecewise linear activation σ with p pieces. If

- a 2-layer FNN f_θ satisfies $(p-1)d_1 + 2 < N$; or
- a 3-layer FNN f_θ satisfies $p(p-1)d_1 d_2 + (p-1)d_2 + 2 < N$,

then there exists a dataset $\{(x_i, y_i)\}_{i=1}^N$ satisfying Assumption 3.1 such that for all θ, there exists $i \in [N]$ such that $y_i \neq f_\theta(x_i)$.

Theorems 3.1 and 3.3 together show tight lower and upper bounds $\Theta(d_1 d_2)$ on memorization capacity of 3-layer FNNs, which differ only in constant factors. [Theorem 3.1 shows that you need at most [some hidden nodes] to memorize arbitrary datasets. Theorem 3.3 shows that you need at least [some hidden nodes].] Theorem 3.3 and the existing result on 2-layer FNNs [53, Theorem 1] also show that the memorization capacity of 2-layer FNNs is $\Theta(d_1)$. [Have not looked too closely at this yet, maybe there's something worth looking at here]

[**Missing proofs???** They cite some appendices but they are nowhere to be found.]

Proof ideas. The proof of Theorem 3.1 is based on an intricate construction of parameters. Roughly speaking, we construct parameters that make each data point have its unique activation pattern in the hidden layers; more details are in Appendix B. The proof of Proposition 3.2 is largely based on Theorem 3.1. By assigning each class j a unique real number ρ_j (which is similar to the trick in Hardt and Ma [20]), we modify the dataset into a 1-D regression dataset; we then fit this dataset using the techniques in Theorem 3.1, and use the extra layer to recover the one-hot representation of the original y_i. Please see Appendix C for the full proof. The main proof idea of Theorem 3.3 is based on counting the number of "pieces" in the network output $f_\theta(x)$ (as a function of x), inspired by Telgarsky [44]. For the proof, please see Appendix D.

Discussion: Depth-Width Tradeoffs for Finite Samples. Theorem 3.1 shows that if the two ReLU hidden layers satisfy $d_1 = d_2 = \sqrt{2Ndy}$, then the network can fit a given dataset perfectly. Proposition 3.2 is an improvement for classification which shows that a 4-layer ReLU FNN can memorize any dy-class classification data if $d_1 = d_2 = 2\sqrt{N}$ and $d_3 = 4dy$.

1.2 Application to CLS + FFNN Architecture

The expressivity of ReLU networks, particularly in finite sample settings, provides a theoretical underpinning for understanding the performance of FFNNs when applied to CLS token embeddings derived from transformer models for the CoLA task.

2 Intrinsic Dimensionality

2.1 Definition and Theoretical Results

The concept of intrinsic dimensionality (ID) pertains to the minimum number of parameters required to achieve satisfactory solutions for a given objective. It can also be viewed as the dimensionality of the lowest subspace in which the original objective function can be optimized to a specified approximation error level. Although computing the exact intrinsic dimensionality is intractable, heuristic methods allow us to approximate an upper bound.

Given a set of parameters $\theta_D = [\theta_0, \theta_1, \ldots, \theta_m]$ that parameterize a model $f(\cdot; \theta)$, the approach re-parametrizes the model in a lower-dimensional subspace as follows:

$$\theta_D = \theta_0^D + P(\theta_d), \tag{1}$$

where $P : \mathbb{R}^d \to \mathbb{R}^D$ projects parameters from a lower-dimensional space d to a higher-dimensional space D. This subspace methodology facilitates optimization in a reduced parameter space and identifies the intrinsic dimensionality if a satisfactory solution is reached [**?**]. [I'm not actually sure if there is a precise mathematical definition of intrinsic dimension; this is just one of many formulations]

The Fastfood transform, used to implement this methodology, is defined by:

$$\theta_D = \theta_0^D + \theta_d M, \quad M = HG\Phi HB, \tag{2}$$

where H denotes a Hadamard matrix, G is a diagonal matrix with standard normal entries, B is a diagonal matrix with entries of ± 1 with equal probability, and Φ is a permutation matrix. The use of the Fastfood transform allows the re-parametrization to be computed efficiently, particularly important for large models.

2.2 Methods

There are a number of ways to estimate intrinsic dimensionality; this section outlines a few of the most common ones

2.2.1 Principal Component Analysis (PCA)

PCA is a statistical procedure that utilizes an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

1. **Standardization (optional, but commonly used):**

$$X_{\text{std}} = \frac{X - \mu}{\sigma}$$

   Here, X is the original data, μ is the mean, and σ is the standard deviation. Standardization is applied to each feature.

2. **Covariance Matrix Computation:**

$$\Sigma = \frac{1}{n-1}(X_{\text{std}})^T X_{\text{std}}$$

   Where Σ is the covariance matrix, X_{std} is the standardized data, and n is the number of data points.

3. **Eigenvalue Decomposition:**

$$\Sigma V = V\Lambda$$

   Here, V is the matrix of eigenvectors, and Λ is the diagonal matrix of eigenvalues of Σ.

4. **Selecting Principal Components:** The eigenvectors are ordered by their corresponding eigenvalues in descending order. The first few eigenvectors correspond to the directions of maximum variance.

5. **Projection onto New Feature Space:**

$$X_{\text{pca}} = X_{\text{std}} V_{\text{reduced}}$$

   Here, V_{reduced} contains the selected eigenvectors, and X_{pca} is the data transformed into the new feature space.

PCA begins by standardizing the data. The covariance matrix Σ is then calculated to understand how each variable relates to one another. Eigenvalue decomposition of Σ yields eigenvalues and eigenvectors. The eigenvectors (principal components) are the directions of maximum variance, and eigenvalues indicate the magnitude of variance in these directions. By projecting the original data onto a few leading eigenvectors, PCA reduces the dimensionality while preserving as much variance as possible. [PCA reduces global variance, but in order to get "good" (as in informative) embeddings, I think you'd want to "locally" reduce variance (within clusters). Idk how to do this tho and this seems hard. PCA is pretty primitive anyways so it shouldn't matter too much.]

2.2.2 Mixture of Gaussians

[tbh GMMs aren't too relevant but I'm throwing this in here anyways since they can help in estimating the intrinsic dimensionality by analyzing the covariance matrices of distributions.]

1. **Probability Density Function:**

$$p(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i)$$

   Here, π_i are the mixing coefficients, $\mathcal{N}(x|\mu_i, \Sigma_i)$ is the normal distribution with mean μ_i and covariance Σ_i, and k is the number of Gaussian components.

2. **Expectation-Maximization (EM) Algorithm for Parameter Estimation:**

   - *E-step (Expectation step):*

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}$$

     $\gamma(z_{ik})$ is the posterior probability that x_i belongs to the k-th Gaussian component.

   - *M-step (Maximization step):*

$$\mu_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(x_i - \mu_k^{new})(x_i - \mu_k^{new})^T$$

$$\pi_k^{new} = \frac{N_k}{N}$$

     Where $N_k = \sum_{i=1}^{N} \gamma(z_{ik})$.

2.2.3 Isomap (Isometric Feature Mapping)

Isomap is a non-linear dimensionality reduction method that is an extension of PCA to non-linear manifolds. [Isomap seems* (*educated guess) to be one of the more reasonable approaches to finding the ID of our datasets]

1. **Constructing the Neighborhood Graph:** Determine the neighbors of each point x_i by either:

   - k-nearest neighbors, or

- Fixed radius ϵ neighborhood.

2. **Compute Shortest Paths:** Calculate the shortest path distance $d_{ij}^{(G)}$ between all pairs of points x_i and x_j in the graph.

3. **Multidimensional Scaling (MDS):**

$$\min_{Y} \sum_{i<j} (\|y_i - y_j\| - d_{ij}^{(G)})^2$$

Here, Y is the low-dimensional representation, and $\|y_i - y_j\|$ is the Euclidean distance between points in the low-dimensional space.

2.2.4 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a non-linear dimensionality reduction technique particularly well suited for embedding high-dimensional data into a space of two or three dimensions, typically for visualization purposes.

1. **Similarity Computation in High-Dimensional Space:**

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

Here, x_i and x_j are data points in the high-dimensional space, and σ_i is the variance of the Gaussian that is centered on data point i. This equation computes the conditional probability $p_{j|i}$ that i would pick j as its neighbor.

2. **Symmetrization:**

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

This step symmetrizes the probabilities for a pair of points i and j, where N is the total number of data points.

3. **Similarity Computation in Low-Dimensional Space:**

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}}$$

y_i and y_j are the representations of points i and j in the low-dimensional space. This equation computes the similarity q_{ij} in the low-dimensional space, using a Student-t distribution.

4. **Cost Function (KL Divergence):**

$$C = \mathrm{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Kullback-Leibler divergence is used as a cost function to measure the difference between the two distributions P (in high-dimensional space) and Q (in low-dimensional space).

5. **Gradient Descent:**

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

   The positions y_i of points in the map are determined by minimizing the cost function C using gradient descent.

t-SNE starts by converting high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities. The probability $p_{j|i}$ is high if x_j is close to x_i. After symmetrization, a similar process is applied in the low-dimensional space to calculate q_{ij}. The Kullback-Leibler divergence between the two distributions P and Q is minimized using gradient descent. This process embeds the high-dimensional data into a lower-dimensional space in a way that preserves local structures and relationships between data points.

2.2.5 Hessian Eigenmaps

Hessian Eigenmaps is an unsupervised learning algorithm that computes a lower-dimensional representation of high-dimensional data, focusing on preserving local properties and is particularly adept at unfolding manifolds.

1. **Local Covariance Matrix:** For each data point x_i, identify its k nearest neighbors and compute the local covariance matrix C_i.
2. **Hessian Estimation:** The Hessian matrix is estimated locally for each data point. It is approximated by the local covariance matrix C_i in the neighborhood of each point.
3. **Eigenvalue Decomposition:**

$$H_i V = V \Lambda$$

   For each local Hessian matrix H_i, compute the eigenvalue decomposition. Here, V is the matrix of eigenvectors, and Λ is the diagonal matrix of eigenvalues.
4. **Global Optimization:** The algorithm seeks a global coordinate system in which the sum of the local Hessian matrices is minimized.
5. **Dimensionality Reduction:** The lower-dimensional representation is obtained by selecting eigenvectors corresponding to the smallest eigenvalues (excluding the zero eigenvalues) across all local Hessian matrices.

Hessian Eigenmaps algorithm starts by estimating the Hessian matrix locally for each data point using the local covariance matrix. This approach allows the algorithm to capture the curvature of the manifold at each point. The eigenvalue decomposition of these local Hessians and the subsequent selection of eigenvectors corresponding to the smallest non-zero eigenvalues enable the unfolding and flattening of the manifold into a lower-dimensional space.

2.2.6 Local Linear Embedding (LLE)

Local Linear Embedding (LLE) is a nonlinear dimensionality reduction method effective at unfolding twisted or curved manifolds.

1. **Neighborhoods:** For each data point x_i, find the K nearest neighbors.
2. **Linear Coefficients:**
$$\min_W \sum_i ||x_i - \sum_j W_{ij} x_j||^2$$
Compute the weights W_{ij} that best reconstruct each data point x_i from its neighbors, subject to the constraint that each row of W sums to one.
3. **Reconstruction in Lower-Dimensional Space:**
$$\min_Y \sum_i ||y_i - \sum_j W_{ij} y_j||^2$$
Find a lower-dimensional representation y_i of the data points that best preserves the local geometry represented by the weights W_{ij}.
4. **Eigenvalue Problem:** The minimization can be reformulated as an eigenvalue problem where the goal is to find the bottom $d+1$ eigenvectors of the matrix $(M = (I - W)^T(I - W))$.

LLE starts by finding the nearest neighbors for each data point and computing linear coefficients that best reconstruct each point from its neighbors. The key idea is to find a lower-dimensional representation of the data that preserves these local relationships. This is achieved by minimizing the reconstruction error in the lower-dimensional space, leading to an eigenvalue problem that provides the desired embedding.

2.3 Application to CLS + FFNN Architecture

In the context of the FAST model, where the CLS token embeddings from a transformer model are used, the intrinsic dimensionality helps in identifying the minimal set of parameters that can still capture the complexity of the embeddings and achieve high performance on tasks like CoLA.

Given a FFNN with architecture defined by:

$$f_{\text{FFNN}}(x) = W_2 \cdot \text{ReLU}(W_1 \cdot x + b_1) + b_2, \tag{3}$$

where W_1, W_2, b_1, b_2 represent the weights and biases of the network, and x represents the CLS token embeddings. The intrinsic dimensionality can be explored by considering how the re-parametrization techniques can compress W_1 and W_2 while preserving performance metrics such as Matthews correlation coefficient (MCC) on CoLA.

By identifying a lower-dimensional subspace for W_1 and W_2, we can hypothesize that if the FAST model retains high MCC with this reduced parametrization, then the intrinsic dimensionality of the task is less than the dimensionality of the embeddings, suggesting efficiency in the model architecture.

2.4 Pros/Cons

The strengths of applying intrinsic dimensionality to the FAST model include potential reduction in the number of parameters, leading to less memory usage and faster computation, which are critical for deploying NLP models in production. This reduction could also potentially mitigate overfitting by constraining the model's capacity.

However, a weakness could be the introduction of underfitting if the reduced parameter space is too constrained to capture the necessary linguistic features for accurate predictions. Moreover, the process of identifying the intrinsic dimensionality requires careful empirical exploration, which can be computationally intensive.

- **Hypothesis 1:** The intrinsic dimensionality of CLS token embeddings is sufficiently low, allowing for a compact FFNN architecture without significant loss in task performance.
- **Hypothesis 2:** A reduced parametric space might lead to faster convergence during training due to a smaller search space, but may require careful tuning to avoid local minima that do not generalize well.

3 Manifold Learning

[Ignore this section, it's just a gpt hallucination] Manifold learning is a type of unsupervised learning that aims to uncover the low-dimensional structure hidden in high-dimensional data. It assumes that the high-dimensional data points are sampled from a low-dimensional manifold embedded within the high-dimensional space.

3.0.1 Locally Linear Embeddings (LLE)

Mathematical Underpinnings LLE assumes that each data point and its neighbors lie on a locally linear patch of the manifold. It computes the lower-dimensional embedding by preserving these local linear relationships.

Advantages LLE may be particularly advantageous for the FAST model if the CLS embeddings exhibit strong local linearity, allowing for accurate reconstruction in lower dimensions.

Disadvantages A potential disadvantage is that LLE might not scale well with the size of the dataset, which can be a concern when dealing with large-scale datasets such as those in GLUE benchmarks.

3.0.2 Isometric Mapping (Isomap)

Mathematical Underpinnings Isomap extends classical Multidimensional Scaling (MDS) by incorporating geodesic distances computed on the manifold. It aims to preserve the global geometry of the data by approximating the geodesic distances in the lower-dimensional space.

Advantages For the FAST model, Isomap could effectively maintain semantic distances between the embeddings, potentially enhancing performance on tasks requiring a global understanding of sentence relationships.

Disadvantages The computation of geodesic distances can be computationally intensive, possibly leading to inefficiencies in terms of training time for the FAST model.

3.0.3 Multidimensional Scaling (MDS)

Mathematical Underpinnings MDS seeks a low-dimensional space where the pairwise distances between points correspond to the distances in the original high-dimensional space, focusing on preserving the global structure.

Advantages MDS could help the FAST model to preserve distance relationships, which is crucial for tasks like STS-B in the GLUE benchmark that rely on semantic similarity measures.

Disadvantages The disadvantage is that MDS can be sensitive to noise and outliers, which may lead to a less robust model for certain NLP tasks.

3.0.4 Spectral Embedding

Mathematical Underpinnings Spectral Embedding, or Laplacian Eigenmaps, uses the spectral decomposition of the graph Laplacian to find a low-dimensional representation that preserves neighborhood relationships.

Advantages This method could benefit the FAST model by emphasizing clustering and continuity within the data, possibly improving performance on tasks like clustering similar sentences.

Disadvantages Spectral Embedding can be computationally expensive, and the choice of neighborhood size can significantly impact the quality of the embedding.

3.0.5 T-distributed Stochastic Neighbor Embedding (t-SNE)

Mathematical Underpinnings t-SNE focuses on converting the high-dimensional Euclidean distances between points into conditional probabilities that represent similarities. It minimizes the Kullback-Leibler divergence between these probabilities in both the high and low-dimensional spaces.

Advantages t-SNE may reveal clusters within the CLS embeddings in the FAST model, aiding in tasks that involve distinct semantic groupings.

Disadvantages t-SNE's non-convex cost function may lead to different results with different initializations, and its computational cost can be significant, especially for large datasets.

4 References

1. Zhang, C., et al. (2017). Understanding deep learning requires rethinking generalization.
2. Arpit, D., et al. (2017). A closer look at memorization in deep networks.
3. Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function.
4. Hornik, K., et al. (1989). Multilayer feedforward networks are universal approximators.
5. Delalleau, O., & Bengio, Y. (2011). Shallow vs. Deep Sum-Product Networks.
6. Telgarsky, M. (2015). Representation benefits of deep feedforward networks.
7. Eldan, R., & Shamir, O. (2016). The power of depth for feedforward neural networks.