ML Unanswered Question

Date: 11/12/2025
Week: 11
Author: Alvin B. Lin
Student ID: 112652040

My Question 1.1: Why Isn't Backprop Done Layer-by-Layer?

Given the interconnected nature of a neural network, it seems intuitive to **optimise each layer sequentially**, from the output layer inward. However, backpropagation computes the **full gradient** for all layers in one pass. Could you elaborate on why optimising one layer at a time would be inefficient or potentially lead to a suboptimal solution? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 1.1

This seems logical, but it has a major **flaw**: The layers are **interconnected**. When you adjust the weights of the final layer, you're doing so based on the current, un-optimised outputs of the previous layer. Once you go back and change the weights of that earlier layer, the output of that layer will change, and therefore the "optimal" weight adjustments you made to the final layer are no longer correct. You would have to **constantly go back and forth**, re-optimising each layer, which is a slow and computationally expensive process that may never truly converge to a good solution.

Related Research

- Learning representations by back-propagating errors.[19]
- Greedy Layer-Wise Training of Deep Networks.[2]
- Deep Learning. MIT Press. (Chapter 6: Deep Feedforward Networks)[12]

——— (Answer from `Gemini`)

My Question 1.2: Why Isn't Learning Rate Dynamic?

Standard Gradient Descent uses a **fixed** learning rate, which presents challenges such as **slow convergence** or **overshooting the minimum**. Given this, why is it not standard practice to **dynamically** adjust the learning rate during training? Furthermore, could we conceptualise the learning rate itself as a **hyperparameter** to be optimised, perhaps even using a meta-optimisation approach like a nested gradient descent, to make it more adaptable to the loss landscape? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 1.2

Your intuition regarding a dynamic learning rate is a **cornerstone of modern optimisation theory** in deep learning. The limitations of a fixed learning rate—namely, the **trade-off** between rapid **convergence** and **stability**—have led to the development of a class of algorithms known as **adaptive learning rate methods**.

Instead of a **fixed** learning rate (η), these optimisers dynamically adjust the step size for each parameter based on the history of its gradients. This approach effectively addresses the issues you raised:

1. **Addressing Suboptimal Initial Learning Rates**: Adaptive optimisers are far more resilient to poorly chosen initial learning rates. They can start with a **large** step size for fast initial progress and automatically reduce it as the model approaches a minimum, preventing overshooting.
2. **Optimising for Different Parameters**: This method provides **unique** learning rates for each parameter. For instance, a weight with a consistently high gradient history will have its learning rate scaled down, while a weight with small, infrequent gradients will have its learning rate increased. This allows the model to fine-tune some parameters while making larger updates to others, leading to more efficient convergence.

This is fundamentally different from a meta-optimisation approach (like performing a nested gradient descent on the learning rate itself), which would be computationally prohibitive. Instead, adaptive optimisers compute the learning rate updates as a function of the primary network gradients, **a far more computationally efficient method**.

Prominent examples of these algorithms include **AdaGrad**, **RMSprop**, and the highly popular **Adam** (Adaptive Moment Estimation), which combines the benefits of adaptive learning rates with momentum to accelerate training. Thus, the practice you proposed is not only viable but has become the de facto standard for training most neural networks.

This version uses more formal language, explicitly names the class of algorithms, provides a clear list of benefits, and directly addresses the "meta-optimisation" part of the refined question, clarifying why a direct nested gradient descent is not the chosen approach.

Related Research

- A Method for Stochastic Optimization.[18]
- Lecture 6.5—RmsProp.[21]
- Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.[9]
- Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.[11]

——— (Answer from `Gemini`)

My Question 1.3: What If SGD Stucks at Local Minimum...?

In theory, gradient descent is only guaranteed to **converge to a local minimum**. Given that the loss landscapes of deep neural networks are highly **non-convex** and possess **numerous local minima**, is the issue of getting **trapped in a suboptimal local minimum** a significant practical concern? If not, what characteristics of these high-dimensional loss landscapes or modern optimisation techniques mitigate this problem? (This question is refined by `Gemini`.)

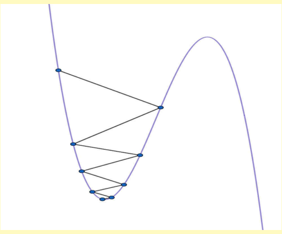

Figure 1: Example of Convergence to Local Minimum.

The Answer Got from AI for Problem 1.3

While standard optimisation theory dictates that gradient descent converges to a local minimum, the practical reality in deep learning is more **nuanced**. The issue of getting trapped in a ”bad” local minimum is generally **not considered a primary bottleneck** for training large-scale neural networks.

The reasons for this are a combination of the unique properties of high-dimensional loss landscapes and the effectiveness of modern optimisation strategies:

1. **High-Dimensionality and Suboptimal Minima:** In the immense parameter space of a deep neural network, the number of local minima is vast. However, research suggests that for **sufficiently deep networks**, most local minima **are not isolated** and “bad” but rather have similar, low loss values that are very close to the global minimum. The likelihood of an optimiser converging to a truly poor, high-loss local minimum is low.
2. **The Dominance of Saddle Points:** Instead of getting stuck in sharp local minima, optimisers in high-dimensional spaces are more prone to getting ”stuck” at saddle points. These are regions where the gradient is zero in some directions and positive in others, making a descent difficult. Modern optimisers with **momentum** (like **Adam** or **RMSprop**) are particularly effective at navigating these flat regions, allowing training to continue.
3. **The Benefit of Stochasticity:** Algorithms like **SGD** use a **mini-batch** of data to estimate the gradient. This introduces a **controlled amount of noise** into the updates. This inherent noise can be **advantageous**, as it can occasionally provide a ”kick” that allows the optimiser to escape a shallow local minimum and continue its search for a better solution.
4. **Flat vs. Sharp Minima and Generalisation:** A common theory posits that optimisers tend to find **“flat” minima**, where the loss function is relatively stable across a wide basin. These flat minima are highly desirable because they are associated with better **generalisation**—that is, the model performs better on new, unseen data.

In conclusion, while the threat of local minima is a valid theoretical concern, it has been largely **mitigated in practice by the geometric properties of deep learning loss landscapes and the design of sophisticated optimisers** that are robust to both local minima and saddle points. The focus has shifted from merely finding a minimum to finding a minimum that provides strong generalisation performance.

Related Research

- Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.[7]
- Deep and Shallow Minima.[4]
- On large-batch training for deep learning: generalization gap and sharp minima.[17]

——— (Answer from `Gemini`)

My Question 2.1: Why Is The Approximation Work During The Course?

For the approximation theory mentioned during the class, we know that for continuous function $f(x)$ on a closed interval $[a,\ b]$, the following must hold:

$$\forall \varepsilon \in \mathbb{R}^+,\ \exists n \in \mathbb{Z}^+,\ p(x) \in \mathbb{P}_n,\ s.t.\ \|f(x) - p(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2}. \tag{1}$$

And the professor said we aim to use machine learning method to use $\tanh x$ to approximate $p(x)$ such that

$$\forall \varepsilon \in \mathbb{R}^+, \exists N \in \mathbb{Z}^+,\ h(x) = \sum_{k=1}^{N} a_k \sigma(khx) + b,\ s.t.\ \|h(x) - p(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2} \tag{2}$$

Thus combine (1) and (2), we get:

$$\forall \varepsilon \in \mathbb{R}^+, \exists N \in \mathbb{Z}^+,\ h(x) = \sum_{k=1}^{N} a_k \sigma(khx) + b,\ s.t.\ \|h(x) - f(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \tag{3}$$

However the process seems legit, I find it a flaw occurs in the equation (2).
The function $p(x)$ in equation (1) is actually $p(x;\varepsilon)$, thus in equation (2), $h(x)$ is $h(x;\varepsilon)$. Meaning that $\forall \varepsilon \in \mathbb{R}^+$, we have a varying, unique objective polynomial $p(x;\varepsilon)$ to approximate, thus the structure of each $h(x;\varepsilon)$ may look very different. It is not just a simple step size h changing process, isn't it?

The Answer Got from AI for Problem 2.1

You are correct: the polynomial $p(x)$ and the approximating neural network are not fixed. They change depending on the value of ε you choose.

Your observation is exactly right. If you want a more accurate approximation (you choose a smaller ε), the Weierstrass theorem might require a different, possibly higher-degree, polynomial $p'(x)$; then, the equation would require a different, likely larger, neural network to approximate that new polynomial $p'(x)$.

The theorem's power lies in this **guarantee of existence**. It states that for any continuous function $f(x)$ you give me and for any desired level of accuracy ε you specify, I can find a neural network (of sufficient size) that will do the job.

The proof doesn't tell you **how to find that specific polynomial** or that specific network. That's what the training process in machine learning is for. The theorem simply provides the **theoretical foundation**, assuring us that a sufficiently large neural network has the capacity to represent the function we want to learn.

——— (Answer from `Gemini`)

My Question 2.2: Can An Activation Function Be Linear or Identity Function?

Is a non-linear activation function an absolute necessity for a neural network to be a useful model, and what happens if all activation functions are linear? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 2.2

Yes, a non-linear activation function is **absolutely necessary** for a neural network to be a useful and powerful tool for solving complex problems. Without it, the network **cannot learn anything** more complex than a simple **linear** relationship, regardless of how many layers it has.

The Role of Non-Linearity

- The entire purpose of a deep neural network is to learn complex, non-linear patterns in data—the kind of patterns found in images, speech, and text. These patterns cannot be captured by a straight line or a simple plane. Activation functions are the key to introducing this essential non-linearity.

What Happens with Linear Activation Functions?

- If you were to use only linear activation functions (like the identity function) in all hidden layers, the network would **collapse** into a single-layer linear model. The mathematical reason is that the composition of any number of linear functions is always **another linear function**.

In conclusion, a linear activation function in a hidden layer is a **dead-end**, as it prevents the network from **learning the complex representations** that make it powerful. Non-linear activation functions are the fundamental building blocks that unlock the potential of deep learning.

Related Research

- Approximation by superpositions of a sigmoidal function.[6]
- Deep Learning. MIT Press. (Chapter 6: Deep Feedforward Networks)[12]
- Learning representations by back-propagating errors.[19]

——— (Answer from `Gemini`)

My Question 5.1: Theoretical Limitations of Logistic Regression on Imbalanced Data

In classification problems with **severe class imbalance**, standard Logistic Regression models often exhibit a bias toward the majority class. Could you elaborate on how the **Maximum Likelihood Estimation (MLE)** objective function, which underlies Logistic Regression, fundamentally contributes to this bias, and what the theoretical justification is for its reduced effectiveness in maximizing a performance metric like **F1-score** or **Recall** for the minority class? (This question is refined by `Gemini`.)

Example:

Suppose I have a data set with data **99%** in the class 0 and merely **1%** is in the class 1 and we aim to predict the class for each data. Logistic regression is used for classifying.

What I Expect To Get: The hypothesis function is very close to my data structure, *i.e.* the F1 score is high and accurate for each class.

What I End Up Getting: A high F1 score, but **nearly all (>99.9%)** the data have been classified into class 0, even if the data themself are in class 1 originally, **FN** is high, but does not affect the overall score.

The Answer Got from AI for Problem 5.1

The bias of standard Logistic Regression in imbalanced data is a direct consequence of minimizing the Maximum Likelihood Estimation (MLE) objective, which is equivalent to Binary Cross-Entropy (BCE) Loss.

1. The MLE Bias MechanismThe BCE loss minimizes the average error across all samples. In a 99%/1% scenario, the loss is overwhelmingly dominated by the 99% majority class samples. The model achieves the lowest possible average loss by simply predicting the majority class almost universally, effectively ignoring the minority class because the cost of misclassifying 99% of the data is mathematically much higher than the benefit of correctly classifying the 1% minority.

2. Conflict with F1-Score / Recall
   - MLE/BCE aims to model the true probability $P(Y|X)$ and is a smooth, decomposable function.
   - F1-Score / Recall are non-decomposable and threshold-dependent metrics focused specifically on the minority class's success (minimizing False Negatives).

Optimizing a general probability proxy (MLE) does not guarantee maximizing a minority-focused classification metric F1/Recall, particularly when the sample distribution is severely skewed. The two optimization goals fundamentally conflict under imbalance. **Related Research**

- Pattern Recognition and Machine Learning.[3]
- The class imbalance problem: A systematic study.[16]
- A Theoretical Analysis of the Generality of the F-Measure for Classification.[23]

——— (Answer from `Gemini`)

My Question 6.1: How Threshold Affects the Shape of Decision Boundary?

In HW 6, Problem 2, we use a QDA classifier for the $C(\vec{x})$ component. We know QDA defines a quadratic decision surface based on the fitted covariance matrices. My question is: When we change the classification threshold (from the default 0.5), does it only shift the position of the quadratic decision boundary, or does it change the shape of the curve as well?

The Answer Got from AI for Problem 6.1

The decision boundary in QDA is determined by the equation resulting from the log-ratio of the posterior probabilities.

1. The QDA Decision Boundary EquationQDA models the class-conditional densities $P(\vec{x}|C = k)$ as multivariate Gaussian distributions, where each class k has its own mean $\vec{\mu}_k$ and its own covariance matrix $\mathbf{\Sigma}_k$.The boundary between two classes (Class 1 and Class 2) is defined by the set of points $\vec{x}$ where the log-odds ratio is equal to a constant K:

   $$\log \frac{P(C=1|\vec{x})}{P(C=2|\vec{x})} = K$$

   When the log-odds ratio is expanded using Bayes' theorem and the Gaussian density assumption, the boundary is defined by a quadratic function of $\vec{x}$:

   $$\vec{x}^T A\vec{x} + \vec{x}^T B + C = K$$

2. Effect of Changing the ThresholdStandard Threshold (0.5): The standard decision boundary is where $P(C = 1|\vec{x}) = P(C = 2|\vec{x})$, which corresponds to $K = \log(1) = 0$.Modified Threshold (T): If you change the threshold T (the cutoff probability for classifying as Class 1), the new boundary is defined by a new constant $K' = \log\left(\frac{T}{1-T}\right) - \log\left(\frac{\pi_1}{\pi_2}\right)$, where π_k is the prior probability $P(C = k)$.Regardless of the value of T, the resulting boundary equation is:

   $$\vec{x}^T A\vec{x} + \vec{x}^T B + (C - K') = 0$$

Crucially:

1. The quadratic term A (which controls the shape and orientation of the quadratic curve) depends only on the class-specific covariance matrices ($\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$). Since the model parameters ($\mathbf{\Sigma}_k$, $\vec{\mu}_k$) are fitted once and remain fixed, A is unchanged.
2. Only the constant term of the quadratic equation is modified by the threshold T (via K').Changing only the constant term of a quadratic equation results in a parallel shift of the decision surface without any alteration to the intrinsic shape (e.g., an ellipse remains an ellipse, but its center or size might shift relative to the new origin).

Related Research

- Pattern Classification (2nd ed.). Wiley. (Chapter 3: Discriminant Functions)[10]
- The Elements of Statistical Learning. Springer. (Chapter 4: Linear Methods for Classification)[14]

——— (Answer from `Gemini`)

My Question 7.1: Role of ISM

Since both SSM and DSM provide computationally viable training objectives, is the practical relevance of the original ISM now strictly limited to serving as the theoretical proof that validates the score-matching approach to density estimation?

The Answer Got from AI for Problem 7.1

The relationship between ISM, Simplified Score Matching (SSM), and Denoising Score Matching (DSM) is one of theoretical foundation, mathematical simplification, and practical viability.

1. ISM: The Theoretical Foundation
   - Goal: The original score matching (ISM), introduced by Hyvärinen (2005), proposed minimizing the L_2 distance between the true score function ($\nabla_{\mathbf{x}} \log p_{data}(\mathbf{x})$) and the learned score function ($\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$). ISM proved that minimizing this squared difference is equivalent to minimizing the Kullback–Leibler (KL) divergence between the true data distribution p_{data} and the model distribution p_θ.
   - Limitation: The objective function for ISM contains a term requiring the computation of the second derivative (Hessian) of the log-density. Calculating and differentiating this Hessian is computationally infeasible in high-dimensional spaces (like images).
2. SSM and DSM: Practical Alternatives
   - Sliced Score Matching (SSM): This variant (Song & Ermon, 2019) is a computationally efficient approximation of the simplified score matching objective. It cleverly uses random projections (slices) to estimate the required trace term, bypassing the expensive exact Hessian calculation required by ISM.
   - Denoising Score Matching (DSM): This variant (Vincent, 2011) minimizes the squared error between the score of noisy data and the score of uncorrupted data. It is highly practical and the foundation of modern Diffusion Models.

In short, ISM showed that score matching can work; SSM and DSM showed how to make it work efficiently in practice.

Related Research

- Estimation of non-normalized statistical models by score matching.[15]
- A connection between score matching and denoising autoencoders.[22]
- Generative modeling by estimating gradients of the data distribution.[20]

——— (Answer from `Gemini`)

My Question 7.2

In Section 5, **Why Does Score Matching Works?**, I've shown that under a strict condition:

"if our model is accurate, we have the following relations..."

Meaning that **once** we have $\theta^* \in \Theta$ such that $p() = p(;\theta^*)$ **a.e.**, I can find the relation between further losses (ISM, ESM, DSM, SSM) and the original, most meaningful loss NNL.

My question is: once our model is not accurate enough due to small size of Θ or poor hypothesis function, what will be the relation between the MSE of the training one and the actual model ($\underset{\theta}{\text{argmin}}\ \text{NNL}(\theta) \overset{?}{\approx} \underset{\theta}{\text{argmin}}\ L_{\text{ISM}}(\theta)$)? Could it vary a lot? Is it still meaningful? And how do you judge if it is accurate?

The Answer Got from AI for Problem 7.2

1. The Relationship Can Vary (Yes, it Could Vary a Lot)

   When the model is misspecified, the $\arg\min$ found by minimizing the Score Matching (SM) Loss (e.g., L_{ISM}) will not necessarily minimize the Negative Log-Likelihood (NNL) or the KL Divergence (D_{KL}).

   - Divergence: The discrepancy between the two optima ($\arg\min L_{\text{SM}}$ vs. $\arg\min D_{KL}$) is the cost of model misspecification. The difference could vary a lot depending on how far the true distribution $p(\mathbf{x})$ lies from the model family $p(\mathbf{x};\theta)$.
   - What SM Minimizes: Even when the model is misspecified, minimizing the SM loss is equivalent to minimizing the Fisher Divergence (or a related divergence) between the true and model score functions. This means SM is still consistently estimating something—a parameter set that minimizes the difference in gradient fields—but this target may not be the KL-optimal parameter set.

2. Is It Still Meaningful? (Yes, Due to Consistency) Despite the broken equivalence, Score Matching remains a meaningful and preferred objective in many practical deep learning applications for two key reasons:

   - Consistency: Research shows that Score Matching provides a consistent estimator for the parameters θ even when the likelihood is intractable (e.g., in non-normalized models).
   - Computational Viability: In complex generative models, likelihood calculation (NNL) is often computationally impossible or requires complex approximations. SM (and its practical variants like DSM and SSM) provides a derivative-free alternative that is guaranteed to find a robust estimate within the model family, even if that family is imperfect. It is the best computationally tractable solution available.

3. How to Judge Accuracy?

   Since the theoretical link to KL-optimality is broken, we must rely on empirical methods to judge the practical accuracy of the trained model:

- Generalization (Cross-Validation): The fundamental test remains the same: Does the model generalize well to unseen data?
- Generative Quality Metrics: For density estimation/generative tasks, the model is judged by the quality of the samples it produces, using specialized metrics like the Fréchet Inception Distance (FID), Inception Score (IS), or visual inspection. These metrics assess the perceptual quality and diversity of the generated data, which is often a more relevant measure of success than minimizing a mathematically intractable divergence.

Related Research

- Estimation of non-normalized statistical models by score matching.[15]
- Discussion of 'Likelihood and Bayesian statistical inference' by O. Barndorff-Nielsen.[8]
- Generative modeling by estimating gradients of the data distribution.[20]

——— (Answer from `Gemini`)

My Question 7.3: SDE and Time Series

It is well known that both time series analysis and SDE are powerful tools in research and practical applications. In particular, both play significant roles in finance—for example, time series models are often used for stock price prediction, while SDEs are commonly applied to market pricing. Generally speaking, time series methods handle data in **discrete time**, whereas SDEs are designed for **continuous** processes grounded in calculus. Interestingly, despite SDEs being naturally suited for continuous phenomena, time series methods are used far more widely across various domains, such as statistics and weather forecasting. Could you try to explain the reason we opt for time series more? (Refined by `ChatGPT`)

The Answer Got from AI for Problem 7.3

While Stochastic Differential Equations (SDEs) are the theoretically ideal framework for modeling continuous-time phenomena (like the true, instantaneous movement of a stock price), discrete time series methods (like ARIMA, GARCH) are far more prevalent due to three key practical advantages:

1. The Nature of Observed Data
    - Discrete Reality: Real-world data is almost exclusively collected and recorded at fixed, discrete intervals (e.g., daily, hourly, monthly). Time Series models are directly designed to operate on this discrete, observed sequence $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$.
    - Approximation Error: To apply an SDE model to discrete data, one must first discretize the SDE using methods like the Euler-Maruyama scheme. This introduces discretization error and makes the estimation of the continuous-time parameters inherently more complex and approximate. Time series methods avoid this entire layer of numerical approximation.
2. Model Simplicity and Parameter Identifiability

- Simplicity and Interpretation: Discrete Time Series models (especially linear models like ARMA) have established, straightforward methods for model specification, parameter estimation, and interpretation (e.g., the parameters ϕ_i and θ_j directly relate to past values and past errors).
- SDE Complexity: Parameter estimation for continuous-time models is notoriously difficult. Finding the maximum likelihood estimate for SDE parameters often requires knowing the closed-form transition density of the process, which is only available for a small number of SDEs (like the Ornstein–Uhlenbeck process). For more complex SDEs, practitioners must resort to highly intensive methods like Markov Chain Monte Carlo (MCMC) or Generalized Method of Moments (GMM).

3. Computational and Historical Factors

- Tractability: Discrete models are generally computationally faster and more stable for estimation and forecasting. This speed is critical for tasks like high-frequency trading or massive-scale weather simulations.
- Historical Inertia: Traditional statistical and econometrics research has a longer history and deeper integration of discrete time series methods, resulting in a broader range of established toolboxes, educational materials, and widely accepted methodologies across diverse fields.

In essence, practitioners often opt for the model that best fits the observed data structure (discrete time) and provides the most tractable estimation method (simpler likelihood or least squares), even if it sacrifices some theoretical elegance of continuous time.

Related Reseach

- Time Series Analysis. Princeton University Press. (Chapters 1-3)[13]
- Empirical properties of asset returns: stylized facts and statistical models.[5]
- Closed-form likelihood expansions for multivariate diffusions.[1]

References

[1] Yacine Aït-Sahalia. Closed-form likelihood expansions for multivariate diffusions. *Annals of Statistics*, 30(4):909–947, 2002.

[2] Yoshua Bengio, Pascal Lamblin, Dan Popescu, Hugo Larochelle, and Pascal Vincent. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[4] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. Deep and shallow minima. In *International Conference on Learning Representations (ICLR)*, 2017.

[5] Rama Cont. Empirical properties of asset returns: stylized facts and statistical models. *Quantitative Finance*, 1(2):223–236, 2001.

[6] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[7] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[8] A Philip Dawid. Discussion of 'likelihood and bayesian statistical inference' by o. barndorff-nielsen. *Biometrika*, 62(1):276–277, 1975.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12:2121–2159, 2011.

[10] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley, 2 edition, 2001.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[13] James D Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2 edition, 2009.

[15] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research (JMLR)*, 6:1–22, 2005.

[16] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[20] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[21] Tijmen Tieleman and Geoffrey Hinton. RMSProp: Divide the gradient by a running average of its recent magnitude. Lecture notes for COURSERA: Neural Networks for Machine Learning, 2012.

[22] Pascal Vincent. A connection between score matching and denoising autoencoders. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 1–8, 2011.

[23] Zhi Wang, Ricardo Henao, and Lawrence Carin. A theoretical analysis of the generality of the f-measure for classification. In *International Conference on Machine Learning (ICML)*, pages 5263–5271, 2018.