Unanswered Questions

1. What's the use of the relation between sigmoid function and hyperbolic function in Machine Learning?

- The sigmoid and tanh functions are mathematically connected:
  $tanh(x) = 2\sigma(2x) - 1$ where $\sigma(x) = \frac{1}{1+e^{-x}}$

- Why this matters in Machine Learning:
  - a. Choice of Activation Function:
    Because sigmoid and tanh behave similarly, you can switch between them depending on performance needs.
  - b. Initialization and Scaling:
    Since tanh is just a scaled sigmoid, weight initialization strategies depend on keeping outputs in effective ranges.
  - c. Rewriting Models:
    Some models historically used sigmoid but can be rewritten using tanh for performance.
    Example: Recurrent Neural Networks (RNNs) originally used sigmoid activations.
  - d. Because the derivative of sigmoid relates to the function itself, understanding their connection helps analyze training dynamics like vanishing gradients in deep networks.

- Corresponding research:
  Bengio, Simard & Frasconi (1994) — Learning long-term dependencies with gradient descent is difficult. Classic analysis that identifies the vanishing (and exploding) gradient problem in recurrent nets — the reason why activation shape (sigmoid vs tanh) matters when gradients are repeatedly multiplied across time steps.
  link: `https://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf`

2. Is there any specific example about approximation theory?

- Example function: $f(x) = |x|$ on $[-1, 1]$
  Although $|x|$ is not differentiable at 0, we can approximate it using Chebyshev polynomials or Taylor-like expansions. One famous sequence is:
  $p_n(x) = \Sigma_{k=0}^{n} \frac{(-1)^k(2k)!}{(1-2k)(k!)^2 4^k} x^{2k}$ as $n \to \infty$, $p_n(x) \to |x|$ uniformly.

3. Is there any good neural network to let the approximation of both the Runge function and its derivative are precise?

- Yes. The best neural networks for precise approximation of both the Runge function and its derivative are:
  - SIREN (sin-activation networks): A special neural network using sin activations, which excel at approximating functions that have high curvature or fine structure.
  - PINN-style networks with derivative loss: Train a neural network $N(x)$ by optimizing $\mathcal{L} = ||N(x) - f(x)||^2 + \lambda||N'(x) - f'(x)||^2$.
  - Neural ODE networks: $\frac{dy}{dx} = g(x, y)$ Train a neural network g such that integrating it reproduces f. If $g(x, y) \approx f'(x)$ then solving the ODE ensures $y(x) \approx f(x)$

- Corresponding research:
  SIREN — Implicit Neural Representations with Periodic Activation Functions (Sitzmann et al., 2020).
  Shows that networks with sinusoidal activations (SIRENs) can represent signals and their derivatives very accurately, and gives the initialization recipe that makes training stable.

link: `https://arxiv.org/abs/2006.09661`

PINNs — Physics-informed neural networks (Raissi, Perdikaris & Karniadakis, 2017/2019).
Introduces training NNs by adding PDE/derivative residuals to the loss; the same idea applied to supervised regression (add derivative-matching terms) improves derivative accuracy drastically.
link: `https://arxiv.org/abs/1711.10561`

Neural ODEs — Neural Ordinary Differential Equations (Chen et al., 2018).
Casts models as continuous-depth networks (parametrize derivatives with a NN). Useful when you want the model to explicitly model derivatives / dynamics — leads to very smooth derivative estimates and principled extrapolation.
link: `https://arxiv.org/abs/1806.07366`

4. Is there any good way to find a good hypothesis function before training?

- We can use Prior Knowledge About the Data:
  - Smooth function: Polynomial regression (low degree), splines, kernel methods
  - Periodic behavior: Fourier basis, sinusoidal model, RNN with periodic kernels
  - Local discontinuities: Piecewise models, decision trees, ReLU networks
  - Physical or causal knowledge: Physics-informed NN, constrained models
- Note that you can't choose the exact best function without training
- Corresponding research:
  Jonathan Baxter (2000): "A Model of Inductive Bias Learning" — This is a foundational paper. Baxter formalizes how a learner, embedded in an "environment" of related tasks, can learn the hypothesis space that works well across tasks.
  link: `https://aima.eecs.berkeley.edu/~russell/classes/cs294/f05/papers/baxter-2000.pdf`

5. What's the use of cross-entropy loss, compare with ODE?

- Cross-entropy loss is a function used to quantify how different the predicted probability distribution is from the true (target) distribution.
- The connection: minimizing cross-entropy = learning a probability density, which can be evolved using an ODE.
  Cross-entropy minimization:Minimizing cross-entropy is equivalent to maximum likelihood estimation (MLE), meaning the model learns a distribution that matches the data distribution.
  $min\, H(p,q) \Leftrightarrow max\, log\, q(data)$
- Corresponding research:
  Zheng, Lu, Chen & Zhu (ICML 2023) — They propose new parameterizations ("velocity parameterization"), variance reduction, and a higher-order flow-matching objective to improve the log-likelihood (i.e., MLE) of diffusion ODEs.
  link: `https://proceedings.mlr.press/v202/zheng23c.html`

6. Explain more about QDA

- Quadratic Discriminant Analysis (QDA) is a classical machine learning method for classification, belonging to the family of discriminant analysis models. It is used when each class in the dataset is assumed to follow a multivariate normal (Gaussian) distribution, but with a different covariance matrix for each class.

- QDA models each class k by a Gaussian distribution:
  $p(x|y=k) = N(\mu_k, \Sigma_k)$. Where μ_k = mean vector of class k, Σ_k = covariance matrix of class k.
  Because each class has its own covariance matrix, the resulting decision boundaries become quadratic curves or surfaces—hence the name Quadratic Discriminant Analysis.

- QDA is better if classes have distinct spread/shape. LDA is better if data is limited or covariances similar.

7. When was the concept of score matching first proposed?

- The concept of score matching was first proposed by Aapo Hyvärinen in 2005, in his paper "Estimation of Non-Normalized Statistical Models by Score Matching."
  link: `https://jmlr.org/papers/volume6/hyvarinen05a/hyvarinen05a.pdf`

8. How to know E[x(t)] and Var[x(t)] in Stochastic differential equation?

- Consider SDE: $dx(t) = a(x(t), t)dt + b(x(t), t)dW$, where $a(x, t)$ is the drift, $b(x, t)$ is the diffusion, W_t is Brownian motion.

  Computing $E[x(t)]$:
  $\frac{d}{dt}E[x(t)] = E[a(x(t), t)]$ since $E[dW_t] = 0$, hence $E[x(t)] = x(0) + \int_0^t E[a(x(s), s)]\, ds$.

  Computing $\mathrm{Var}(x(t))$:
  Use the definition $\mathrm{Var}(x(t)) = E[x(t)^2] - (E[x(t)])^2$. Since $(dW_t)^2 = dt$, we get $d(x(t)^2) = 2x(t)a(x(t), t)\, dt + 2x(t)b(x(t), t)\, dW_t + b(x(t), t)^2\, dt$.
  Taking expectation: $\frac{d}{dt}E[x(t)^2] = E\left[2x(t)a(x(t), t) + b(x(t), t)^2\right]$. After that, $E[x(t)^2]$ and $E[x(t)]$ are known.

9. Can backward SDE actually back to initial data distribution in practice?

- In theory yes (under strong assumptions); In practice, not always, unless you solve it very accurately.

- Three sources of error appear:

  - Score estimation error: Neural network cannot perfectly approximate the score at all times, especially at low-density regions.
  - Numerical discretization error: Solving SDE backward uses small timesteps; finite-step solvers introduce errors, especially if steps are large.
  - Model misspecification: If the score network is too small or data too complex, approximation fails.

- Corresponding research:
  Score-based SDE framework (foundational theory + PC sampling). Song et al., Score-Based Generative Modeling through SDEs — shows the reverse-time SDE, ties score estimation to sampling, and introduces the predictor–corrector (PC) approach.
  link: `https://arxiv.org/abs/2011.13456`