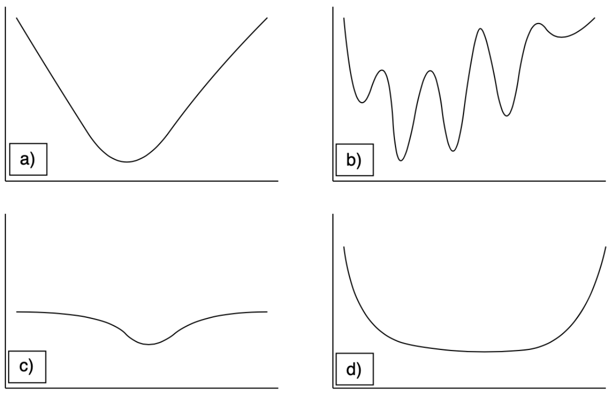
Deep Learning Theory and Applications
(CPSC/AMTH 452/552, CBB 663)
Final Exam
Due May 11, 2022
Wenxin Xu

Policy:

This exam is "open book, open notes" which means you are permitted to use any reading materials presented in class or suggested in the class syllabus. The exam must be taken completely alone. Showing it or discussing it with anyone is forbidden. You may not consult online forums, question-answering services, or course help sites. You may not consult with any other person regarding the exam. You may not check your exam answers with any person.

Multiple choice (2 points each):

1. In which of these four loss landscapes is SGD likely to do the best, i.e. the solution is best in terms of training convergence and generalization on the test set? d

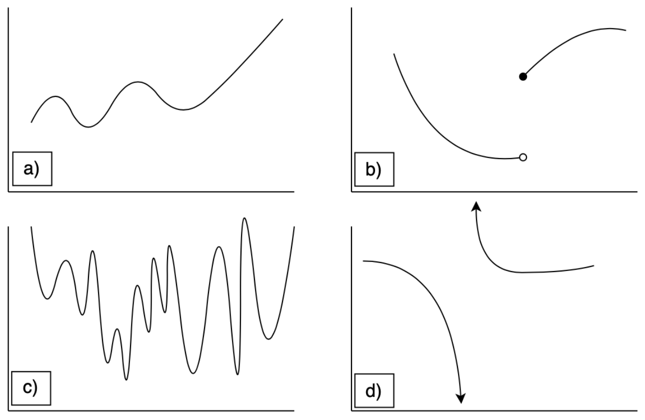


Loss landscapes

2. Suppose you wanted to train a neural network to predict the next note in a long musical composition. What kind of neural network is likely to produce the best performance? c
   (a) RNN
   (b) LSTM
   (c) Transformer
   (d) CNN

3. In which neural network(s) will SGD training reach the global minima? You may choose multiple answers. c
   (a) Linear neural network
   (b) Infinitely deep network
   (c) Infinitely wide network
   (d) Network with quadratic activations

4. Which of these functions can neural networks learn to approximate? You can choose multiple answers. ac



Functions

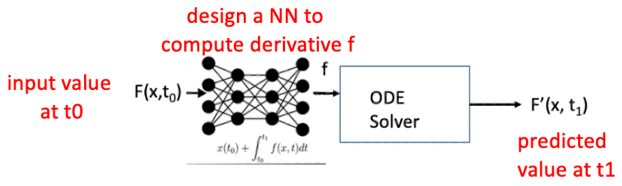
5. Suppose that given a large set of patient data features (labs, genomics, etc) , one can predict a disease diagnosis with high accuracy. Which of the following does this imply: a, d

(a) One or more features are correlated with the disease.
(b) One or more features cause the disease.
(c) One or more features is characteristic of the disease.
(d) One or more features have mutual information with the disease variable.
(e) All of the above.
(f) None of the above.

Short answer (5 points each):

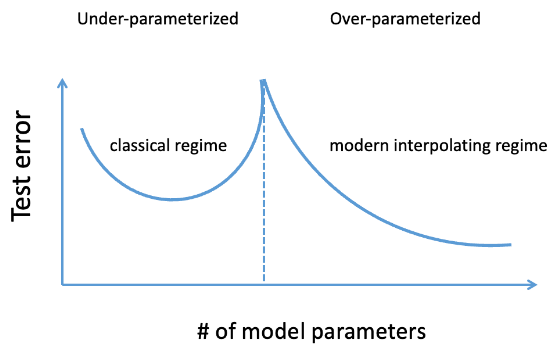
1. How do the neural network and ODE solver interact in a neural ODE network.

Input value of target function F at t_0 to a neural network to compute derivative f, the ODE integrals the derivative to predict value F at t_1, then compute loss between ground truth F(x, t_1) and predicted F'(x, t_1), backpropagate through operations of ODE-Solver and network via adjoint sensitivity which measures stability.


design a NN to compute derivative f
input value at t0
$F(x,t_0)$
f
ODE Solver
$F'(x, t_1)$
predicted value at t1
$x(t_0) + \int_{t_0}^{t_1} f(x,t)dt$

What does the "neural network" part of the neural ODE compute?
Compute the derivative of the target function f = dF/dt

2. Draw a conceptual graph of test error vs # of model parameters, in which regime of this graph do neural networks lie?
   Modern interpolating regime



3. Suppose you trained an n-layered autoencoder, and you got perfect reconstruction, i.e. 0 training error. Does this mean that one of the hidden layers of this network produces meaningful embeddings? Why or why not?

   No, the autoencoder may just memorize the correct answer to achieve 0 training error, not necessarily learn any meaningful pattern in data.

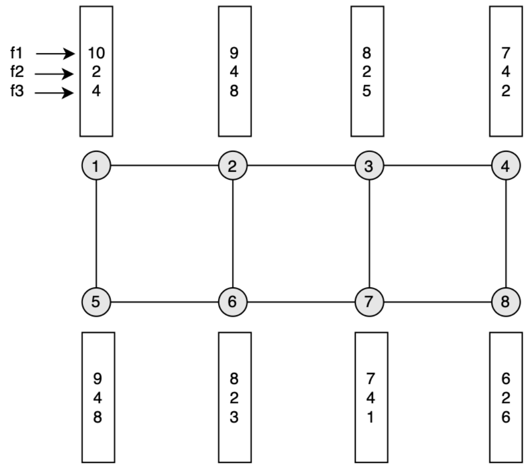
4. How does momentum modify SGD?
   Change "velocity" v instead of the "position" for each corresponding w variable
   Introduce a friction term $\mu \in (0,1)$ to control the damping of the system, which gradually reduces velocity

   Write an equation to illustrate the change.
   New update rule:

$$v \to v' = \mu v - \eta \nabla C$$
$$w \to w' = w + v'$$

   What is the advantage of using momentum over vanilla SGD?
   Speed up learning and faster converging

5. Compute the **unnormalized Laplacian** of the graph below.



Adjacency matrix $A = \begin{bmatrix} 0&1&0&0&1&0&0&0 \\ 1&0&1&0&0&1&0&0 \\ 0&1&0&1&0&0&1&0 \\ 0&0&1&0&0&0&0&1 \\ 1&0&0&0&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 0&0&0&1&0&0&1&0 \end{bmatrix}$

Degree matrix $D = \begin{bmatrix} 2&0&0&0&0&0&0&0 \\ 0&3&0&0&0&0&0&0 \\ 0&0&3&0&0&0&0&0 \\ 0&0&0&2&0&0&0&0 \\ 0&0&0&0&2&0&0&0 \\ 0&0&0&0&0&3&0&0 \\ 0&0&0&0&0&0&3&0 \\ 0&0&0&0&0&0&0&2 \end{bmatrix}$

Unnormalized Laplacian $L = D - A = \begin{bmatrix} 2&-1&0&0&-1&0&0&0 \\ -1&3&-1&0&0&-1&0&0 \\ 0&-1&3&-1&0&0&-1&0 \\ 0&0&-1&2&0&0&0&-1 \\ -1&0&0&0&2&-1&0&0 \\ 0&-1&0&0&-1&3&-1&0 \\ 0&0&-1&0&0&-1&3&-1 \\ 0&0&0&-1&0&0&-1&2 \end{bmatrix}$

Which of the features f_1, f_2, or f_3, are likely to load to the **low** eigenvalued eigenvectors of the Laplacian? f_1

Low eigenvalues correspond to low frequency signal on graph, which is smoother with lower s
Given smoothness $s = f^T L f,\ s_1 = 10, s_2 = 40,\ s_3 = 161$
Thus, feature f_1 is the smoothest, most likely to load to low eigenvalued eigenvectors of Laplacian.

6. Nodes in the graph above have three features, suppose you wanted to train a classifier for all graphs based on the **global** patterns of these features, What sort of neural network would you use?

I will use Graph convolutional network (GCN): first obtain node embeddings via several conv layers, then obtain graph embedding via a graph-level readout function of all the individual node embeddings, finally apply a linear layer for graph classification.

7. Write the **four equations of backpropagation** for a neural network with activations given by *sin(z)* where *z* is the weighted sum of activations from the previous layer.

Activation function $\sigma(z) = \sin(z), \sigma'(z) = \cos(z)$

1. Error at final layer $\delta^L = \nabla_a C \odot \sigma'(z^L) = \nabla_a C \odot \cos(z^L)$
2. Error at lth layer $\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l) = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \cos(z^L)$
3. Gradient of cost function w.r.t to bias at lth layer $\frac{\partial C}{\partial b_j^l} = \delta_j^l$
4. Gradient of cost function w.r.t to weight at lth layer $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

8. What is the Hessian matrix of a neural network? What do the eigenvalues of the Hessian indicate?

Hessian matrix of a neural network is the second derivative of loss function, i.e., Jacobian matrix of gradient of loss function

The eigenvalues of Hessian matrix indicate curvature in every direction. If all the eigenvalues are positive at a certain point, then curvature is positive everywhere, this point is a local minima of loss function. If eigenvalues are a mix of positive and negative at a certain point, then this point is a saddle point of loss function

9. What is mode collapse in a GAN? What is one solution for mode collapse?
Mode collapse: GAN only generates a few categories of data, lack of diversity
Solution: feature matching and minibatch discrimination

10. Suppose you have a set of *n* points in *d* dimensions, describe a polynomial time algorithm for training a linear autoencoder with 3 layers (input, output, and embedding layer) to preserve pairwise distances between these *n* points within an ε threshold, in the embedding described by the network.

Input: a set V of n points in d dimensions
Embedding layer: learn a map f: $\mathbb{R}^d \rightarrow \mathbb{R}^k$ such that for all $u, v \in V$,

$$(1-\epsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\epsilon)\|u-v\|^2.$$

Output: a set of n points in d dimensions
Loss: reconstruction loss (L2 norm)
Optimizer: SGD

What should the dimensionality of the embedding layer be? $O(log \frac{n}{\epsilon^2})$

Long answers (10 points each):

1. Design a neural network that transfers photos to impressionist paintings. What kind of regularizations can help you retain features of the photos in the paintings?

   use a CycleGAN to do the style transfer:
   Cycle consistency loss can help retain features of the photos in the paintings

   The architecture has 2 GANs, each GAN has a discriminator and a conditional generator.

   - GAN1: learn a mapping from domain A (photo) to domain B (impressionist painting)
     - generator
       - Input: photos
       - Output: generated impressionist paintings
     - Discriminator
       - Input: real and generated impressionist paintings from GAN1 generator
       - Output: generated impressionist paintings is fake or real
   - GAN2: learn a reverse mapping from domain B (impressionist painting) to domain A (photo)
     - generator
       - Input: impressionist paintings
       - Output: generated photos
     - Discriminator
       - Input: real and generated photos from GAN2 generator
       - Output: generated photos is fake or real

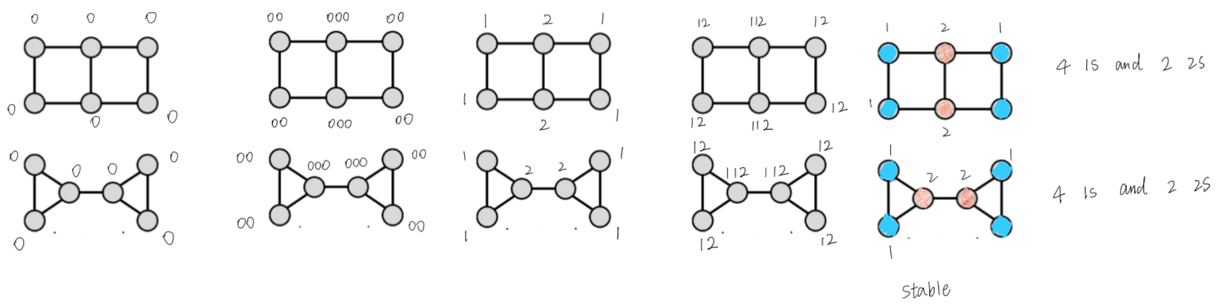
   Loss: adversarial loss + cycle consistency loss

   Standard adversarial loss are sufficient for generating plausible images in the target domain but are not guarantee to generate translations of the input image. So we introduce cycle consistency loss to the loss function.

   There is a cycle in the process: (1) forward cycle: GAN 1 takes a photo, generates an impressionist painting, which is provided as input to GAN 2, GAN2 in turn generates a photo. (2) backward cycle: GAN2 takes an impressionist painting, generates a photo, which is provided as input to GAN1, GAN1 in turn generates an impressionist painting.

   Cycle consistency loss will compare the difference between input photo and generated photo (or input impressionist painting and generated impressionist painting) using L1-norm or sum of pixel-level absolute difference.

2. Are these two graphs isomorphic? Run the Weisfeiler-Lehman test to show if it distinguishes these graphs. Recall the test involves 1) Labeling all nodes with the same label "0", 2) concatenating labels of adjacent nodes in the next iteration 3) relabeling (i.e. 00=1, 000=2 etc), 4) repeating step 2 until there is no change in labels. (See example here: https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/)



Though finally these 2 graphs colored the same pattern but they are not isomorphic because WL test is a necessary but insufficient condition for graph isomorphism.

3. Design an improved graph neural network for testing isomorphism of two graphs that works in many cases. Describe its architecture and if-applicable its aggregation function.

We can design a Graph Isomorphism Network (GIN)
Let G_1 and G_2 be any two non-isomorphic graphs, if a graph neural network $\mathcal{A}: \mathcal{G} \rightarrow \mathbb{R}^d$ maps G_1 and G_2 to different embeddings, then these two graphs are not isomorphic.

a) node embeddings $h_v^{(k)}$: aggregates and updates node features iteratively with a multi-layer perceptrons (MLPs)

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum\nolimits_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right).$$

Where $h_v^{(k)}$ is feature vector of node v at the k-th iteration, $\mathcal{N}(v)$ is a set of nodes adjacent to v, ϵ is a learnable parameter or a scalar

b) graph embedding h_G: concatenating graph representations across all the K iterations

$$h_G = \mathrm{CONCAT}\Big(\mathrm{READOUT}\Big(\Big\{h_v^{(k)} | v \in G\Big\}\Big) \mid k = 0, 1, \ldots, K\Big).$$

Where READOUT is a simple permutation invariant function, which is injective, operates on the multiset of node features $\left\{h_v^{(k)}\right\}$, such as summing all node features from the same iterations

Would it work on the test case above?
Because the GNNs are at most as powerful as WL test, the GIN also doesn't work on the test case above.

Bonus

Can a general neural network be trained without backpropagation, only using forward propagation? If yes, explain how.

Yes
Atılım G ̈une s ̧Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. 2022. Gradients without backpropagation. https://arxiv.org/abs/2202.08587

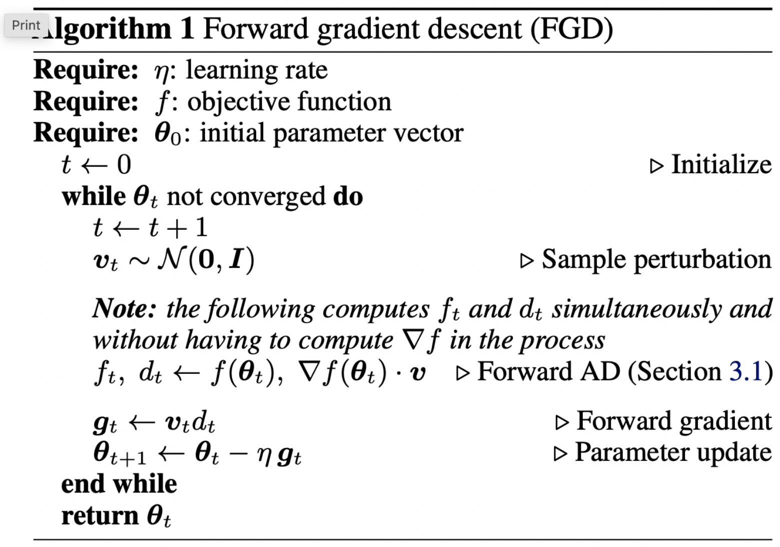
Using backpropagation to compute gradients of objective functions for optimization is a special case (i.e., reverse mode) within general family of automatic differentiation algorithms. This family also includes forward mode, we can evaluate gradient in a single forward run of the objective function.
Given an objective function $f\colon \mathbb{R}^n \to \mathbb{R}$ with parameter $\theta \in \mathbb{R}^n$

Forward mode: evaluates the vector-Jacobian product, which is directional derivative $\nabla f(\theta) \cdot v \in \mathbb{R}$, i.e, the projection of the gradient ∇f at θ onto the direction vector v, representing the rate of change along that direction.

$$\begin{tikzcd} \boldsymbol{\theta} \quad \boldsymbol{v} \arrow[r, "\text{Forward}"] & \boldsymbol{f}(\boldsymbol{\theta}) \quad \boldsymbol{J_f}(\boldsymbol{\theta})\,\boldsymbol{v} \end{tikzcd}$$

Reverse mode: evaluates the Jacobian-vector product, which is full gradient $v^T J_f(\theta) = 1 \cdot \nabla f \in \mathbb{R}^n$ $(v = 1)$, i.e., the partial derivatives of f w.r.t. all n inputs $\nabla f(\theta) = \left[\frac{\partial f}{\partial \theta_1}, \dots, \frac{\partial f}{\partial \theta_n}\right]^T$

$$\begin{tikzcd} \boldsymbol{\theta} \arrow[r, "\text{Forward}"] & \boldsymbol{f}(\boldsymbol{\theta}) \\ \boldsymbol{v}^\top \boldsymbol{J_f}(\boldsymbol{\theta}) & \boldsymbol{v} \arrow[l, "\text{Backward}"'] \end{tikzcd}$$



lgorithm 1 Forward gradient descent (FGD)

```
Require: $\eta$: learning rate
Require: $f$: objective function
Require: $\boldsymbol{\theta}_0$: initial parameter vector
  $t \leftarrow 0$                                         ▷ Initialize
  while $\boldsymbol{\theta}_t$ not converged do
    $t \leftarrow t + 1$
    $\boldsymbol{v}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$                      ▷ Sample perturbation
    Note: the following computes $f_t$ and $d_t$ simultaneously and
    without having to compute $\nabla f$ in the process
    $f_t,\ d_t \leftarrow f(\boldsymbol{\theta}_t),\ \nabla f(\boldsymbol{\theta}_t) \cdot \boldsymbol{v}$   ▷ Forward AD (Section 3.1)
    $\boldsymbol{g}_t \leftarrow \boldsymbol{v}_t d_t$                           ▷ Forward gradient
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \boldsymbol{g}_t$                  ▷ Parameter update
  end while
  return $\boldsymbol{\theta}_t$
```