ECE657: Tools of Intelligent Systems Design
Instructor: Prof. Amir-Hossein Karimi
TA: Arezoo Alipanah & Priyank Avijeet

Assignment: #3
Total Marks: 107
Deadline: July 04, 2024, 11:59pm EST

[7 marks] Question 1: Universal Approximation Theorem

Universal Approximation Theorem states that a feed-forward neural network with a single hidden layer, a finite number of neurons, and a non-constant, bounded activation function can approximate any continuous function on a compact domain to arbitrary accuracy. Mathematically, this is represented as follows. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function on a compact set $K \subset \mathbb{R}^n$. Then, for any $\epsilon > 0$, there exists a neural network with weights W_1, W_2, biases b_1, b_2, and an overall network function $\phi(x)$ such that:

$$\sup_{x \in K} |f(x) - \phi(x)| < \epsilon$$

[2 marks] Q1.1: Imagine the following data distributions for a 2-class problem.

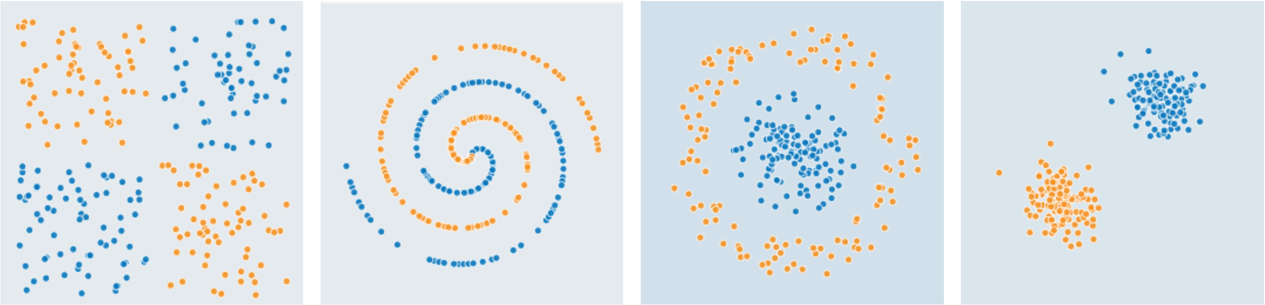


Figure 1: Left to right: dataset 1; dataset 2; dataset 3; dataset 4.

You have a neural net with one hidden layer and no activation function (i.e., linear). Your features are X_1 and X_2 (the Euclidean coordinates of each point); explain how augmenting these features with new features could potentially make each dataset separable. For example, consider the following features

$$X_1, X_2, X_1^2, X_2^2, X_1X_2, \sin(X_1), \sin(X_2)$$

does adding any subset of these features aid in linear separability of the datasets above?

[2 marks] Q1.2: Which one of the data distributions cannot be separable without an activation function, no matter what features are used? Add a nonlinear activation function and see how your results change.

[3 marks] Q1.3: Using the Neural Network Playground Demo, modify the training parameters in order to train for the dataset(s) from the previous part . Report the parameter settings that allow for a good training and test loss on that model and dataset (by good, we mean that your test loss with a 50-50 data split is below 0.1). Describe any patterns you observe in terms of how this is related to the universal approximation theorem. Show your results for different activation functions and parameters, like the number of neurons, number of hidden layers and the features used .

[15 marks] Question 2: Gradient Descent for Softmax Regression

Using only NumPy, implement batch gradient descent for softmax regression. Train this classifier on the Iris dataset, demonstrating the process without relying on Scikit-learn.

[1 mark] Q2.1: Load the Iris data as provided in the notebook of the assignment. Take only petal length and petal width as your features (follow the instructions in the notebook). Add the bias term for every instance ($x_0 = 1$).

[2 marks] Q2.2: The targets are class indices (0, 1 or 2). They have to turn to class probabilities to train the Softmax Regression model. Each instance must show a probability equal to 0.0 for all classes except for the target class (1.0) (so the class probability vectors should be a one-hot vector). Write a function to convert the vector of class indices to a matrix of one-hot vector for each instance.

[2 marks] Q2.3: Normalize the data using Z-Score Normalization and define the softmax function to be used later.

[8 marks] Q2.4: Implement the gradient step using numpy. Make sure about the dimensions and the correctness of your calculations. The cost function is given by:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

and the gradients are:

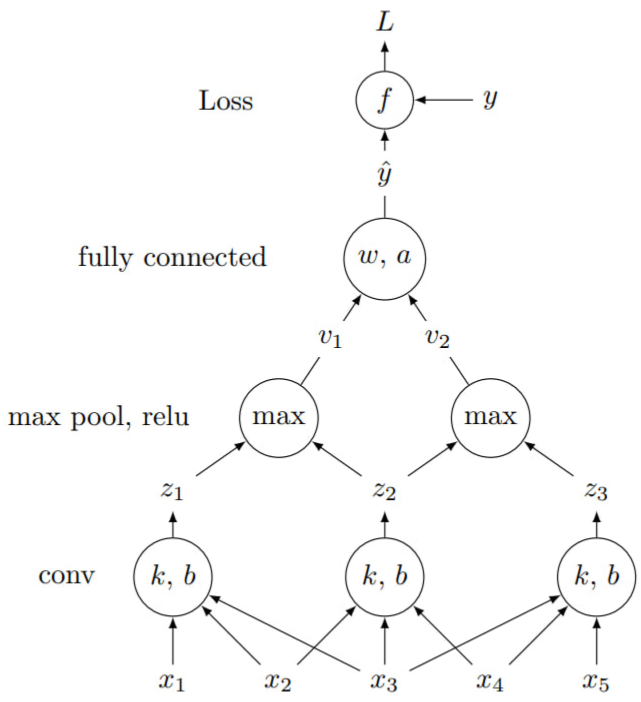
$$\nabla_{\theta^{(k)}} J(\mathbf{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

to avoid problems with getting `nan` values regarding $\hat{p}_k^{(i)} = 0$ you can add a tiny value ϵ to $\log\left(\hat{p}_k^{(i)}\right)$.

[2 marks] Q2.5: Using the plotting function provided in the notebook document and plot the results of your model, showing a scatter plot of the decision boundary and the data points with respect to the petal length and petal width.

[10 marks] Question 3: Backpropagation

Consider the following 1-dimensional convolutional neural network (ConvNet) where all variables are scalars:



$$L = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + a$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \max\{z_1, z_2, 0\} \\ \max\{z_2, z_3, 0\} \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

Figure 2: 1D convolutional net

[1 mark] Q3.1: Identify all the trainable parameters within this network.

[3 marks] Q3.2: Calculate the gradients of the loss function (L) with respect to the parameters w_1, w_2, and a. That is, find $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$, and $\frac{\partial L}{\partial a}$.

[3 marks] Q3.3: Given the gradients of the loss L with respect to the second layer activations (v), compute the gradients of the loss with respect to the first layer activations (z). Specifically, if $\frac{\partial L}{\partial v_1} = \delta_1$ and $\frac{\partial L}{\partial v_2} = \delta_2$, find $\frac{\partial L}{\partial z_1}$, $\frac{\partial L}{\partial z_2}$, and $\frac{\partial L}{\partial z_3}$.

[3 marks] Q3.4: Given the gradients of the loss L with respect to the first layer activations (z), calculate the gradients of the loss with respect to the convolution filter (k) and bias (b). Specifically, if $\frac{\partial L}{\partial z_1} = \delta_1$, $\frac{\partial L}{\partial z_2} = \delta_2$, and $\frac{\partial L}{\partial z_3} = \delta_3$, find $\frac{\partial L}{\partial k_1}$, $\frac{\partial L}{\partial k_2}$, $\frac{\partial L}{\partial k_3}$, and $\frac{\partial L}{\partial b}$.

[20 marks] Question 4: Convolutional Neural Networks

The dataset can be found here. Convolutional Neural Networks (CNNs) are a class of deep neural networks highly effective for analyzing visual imagery. In this exercise, you will work with an image dataset divided into two main folders: 'train' and 'test'. Within the 'train' dataset, there are two subfolders named 'polar' and 'not_polar', containing images of polar bears and images without polar bears, respectively. Your task will involve training a CNN to distinguish between these two categories. The skeleton code can be found here. Insert code wherever you see a 'TODO' comment in the section.

[4 marks] Q4.1: The ImageDataGenerator class in Keras is a tool for real-time data augmentation, which can enhance the number and diversity of your training images available for training deep learning models. Data augmentation automatically introduces random transformations to the training data (such as rotation, scaling, and horizontal flipping), creating variations of the images. This helps prevent the model from overfitting and generalizes better, thus improving its performance on new, unseen data. Implement the `create_image_generators` function to create image data generators for training and validation:

- The function should take parameters: `train_dir` (e.g., "/path/to/data"), `target_size` ((150, 150)), `batch_size` (e.g., 32), and `val_split` (0.2).
- Initialize an `ImageDataGenerator` object with parameters for rescaling (1./255), rotation (40 degrees), width shift (0.2), height shift (0.2), shear (0.2), zoom (0.2), flipping (horizontal), fill mode ('nearest'), and validation split (0.2).
- Use `flow_from_directory` with `subset` set to 'training', `class_mode` set to 'binary', and `seed` for reproducibility (25).
- Similarly, create a validation generator with `subset` set to 'validation'. Use the same settings for `class_mode` and `seed`.
- The function should return both the training and validation generators.

Report the 'Training Generator Info' and 'Validation Generator Info' that you got when you executed `print_generator_info` function.

[4 marks] Q4.2: Build the following convolutional neural network architecture using the sequential model approach:

- Start with a **Conv2D layer** with 32 filters, a 3x3 kernel size, and 'relu' activation. Specify the input shape as (150, 150, 3) for images that are 150x150 pixels with 3 color channels.
- Add a **MaxPooling2D layer** with a pool size of 2x2 to reduce the spatial dimensions of the output from the previous layer.
- Include another **Conv2D layer** with 64 filters and a 3x3 kernel size, using 'relu' activation to introduce non-linearity.
- Follow this with another **MaxPooling2D layer** with a pool size of 2x2.
- Flatten the output to a single vector with a **Flatten layer**, preparing it for the dense layers that follow.
- Introduce a **Dropout layer** with a rate of 0.5 to reduce overfitting by randomly setting input units to 0 during training.
- Add a **Dense layer** with 512 units and 'relu' activation to process the vector of features coming from the flattened output.
- Conclude with a **Dense layer** with 1 unit and 'sigmoid' activation for binary classification, which will output probabilities indicating the presence or absence of a polar bear.

Report the model summary and total number of parameters in the model you just built.

[4 marks] Q4.3: To compile and train a Keras model, first, compile the model using the `compile` method, specifying the loss function as `binary_crossentropy`, the optimizer as `adam`, and tracking the `accuracy` metric. Then, train the model using the `fit` method, passing the `train_generator` for training data, setting the number of epochs to 20, and including the `validation_generator` for validation data. Plot the training and validation accuracies and losses. Does this model appear to be efficient at this stage?

[4 marks] Q4.4: There are just six images in the test folder. Load them one by one and infer by seeing what the model classifies. Report the predictions of the model. Which of the test images were classified wrong and why did that happen and how can we address that?

[4 marks] Q4.5: One of the intriguing aspects of deep learning is that the models we train are far more than just enigmatic boxes outputting predictions or classification results; instead, your network operates as an intricate, multi-layered system that transforms input data into a series of sophisticated encodings. Define the `get_layer_outputs` function, begin by specifying its signature, which includes two parameters: 'model', representing the Keras model, and 'img_tensor', the input tensor for the model. Within the function, extract the output tensors for all layers up to the last MaxPooling layer (index 3) in the model. Next, create a new model named 'activation_model' using the 'Model' class from Keras, which takes the original model's input and the tensors extracted in the previous step as output. Finally, utilize the 'predict' method of the 'activation_model' to predict the activations for the input 'img_tensor', and return these activations as a list of numpy arrays, where each array corresponds to the activations of one layer. These activations are then used by `display_layer_activations` function. Upload the visualizations of the activations for test_4.png and comment on increasing abstractness as we go deeper into the model.

[25 marks] Question 5: word2vec

Create Python script for creating Word2vec from scratch by training a Continuous Bag of Words (CBOW) model using TensorFlow and Keras. Do not use the Gensim library. The dataset can be found here, and the skeleton code can be found here. You are required to complete the **TODO** sections in the skeleton code. The invocation of functions which are to completed will be made later in the code; do not change the function call and parameter values like window size and embedding size. Set seeds to ensure reproducibility in your code. Use the **same seed value (25)** for all random number generators in your environment, including libraries like NumPy, TensorFlow, and Python's built-in random module.

[2 marks] Q5.1: Complete the `preprocess()` function by converting the text in lowercase and splitting it into words. What is the total number of words?

[5 marks] Q5.2: Complete the `build_and_prepare_data()` function. This function preprocesses a list of words to create a vocabulary and generate training data. It builds a vocabulary by assigning a unique index to each word and generates context-target pairs by considering a specified window of words around each target word. The contexts and targets are then extracted and prepared for training by ensuring uniform length for contexts and encoding the targets in a suitable format. The processed vocabulary, context sequences, and target data are then returned, ready to be used in training tasks like word embeddings or other natural language processing models. The **window size should be 2**. What is the vocab size, number of contexts and number of targets?

[4 marks] Q5.3: Complete the `build_cbow_model` function that constructs a Continuous Bag of Words (CBOW) model using TensorFlow and Keras libraries. The CBOW model architecture includes an input layer for context words, an embedding layer to convert these words into dense vectors, an averaging layer to combine these vectors, and an output layer with a softmax function to predict the target word. The **embedding size should be 2**. Your function should return the constructed model. Print the model summary.

[4 marks] Q5.4: Extract the embeddings from the embedding layer. Since the size the embedding size is 2, you should plot the embeddings and visualize them. Provide the plot in your submission.

[6 marks] Q5.5: Complete the two functions, `cosine_similarity()` and `find_similar_words()`, to analyze word similarities using vector embeddings. The `cosine_similarity` function should measure how similar two vectors are based on their orientation. The `find_similar_words` function should identify and return the most similar words to a given `query_word` from a set of word embeddings, ranking them by their similarity. Use these functions to find and return the top 3 similar countries for each of the following: `Poland`, `Thailand`, and `Morocco`.

[2 marks] Q5.6: Consider a small window size, e.g., 2 or 3. In this scenario, is there a possibility that antonyms (opposite words) might end up with similar embeddings? Explain your views.

[30 marks] Question 6: Next Word Prediction

“Alice in Wonderland” is a whimsical tale by Lewis Carroll about a young girl named Alice who falls through a rabbit hole into a fantastical world full of peculiar creatures and surreal adventures. You can find the entire text here. Your task is to build a next-word prediction model, trained only on the Alice in Wonderland textual data. **Note:** We do not aim to achieve high accuracy in this exercise, the goal is to observe whether the model is learning or not and how to address overfitting. Skeleton code may be found here. Use the same seed value (25) for all random number generators in your environment, including libraries like NumPy, TensorFlow, and Python’s built-in random module.

[4 marks] Q6.1: For this task, you’ll be working with a text file containing data that needs to be preprocessed for further analysis. Start by accessing the file from its location on your drive. Once opened, read its contents into a string, ensuring that the text is handled in a case-*insensitive* manner by converting it to lowercase. To remove punctuations in the text, apply a regular expression that filters out all characters that are not letters, digits, underscores, or whitespace (not \w and \s in regex). For example:

Sample text: “Hello, World! Welcome to 2024. Let’s preprocess this text: #ECE-657 @UWaterloo"
Preprocessed sample text: “hello world welcome to 2024 lets preprocess this text ece657 uwaterloo"

This preprocessing step simplifies the text, making it uniform and easier to analyze in subsequent tasks. Print the length of the final processed text obtained.

[2 marks] Q6.2: Initiate the process of text tokenization which is vital for preparing data for natural language processing models. Utilize the Tokenizer from the TensorFlow Keras library used for preparing text data for deep learning models to analyze the text and identify unique words. By fitting the tokenizer to the text, it constructs a comprehensive dictionary of these unique words. Subsequently, calculate the total number of unique words, which is essential for configuring various model parameters, such as input dimensions in neural networks. This total also includes an additional count to accommodate the tokenizer’s indexing method. Print the total number of words.

[4 marks] Q6.3: In this task, you’ll prepare input sequences for training by first splitting the preprocessed text on newline character and converting each line into a list of tokenized words. For each line, generate n-gram sequences of increasing length to create a comprehensive set of training samples. These n-grams, which consist of consecutive tokens, help the model learn contextual relationships within the data. After constructing these sequences, identify the maximum sequence length and standardize all sequences to this length using padding. This padding, typically added to the beginning of sequences, ensures that all input data fed into the model maintains a consistent format, crucial for effective training of sequence-based neural networks like LSTMs or RNNs. For example for the given text,

```
"Hello world
How are you"
```

Following steps will occur:
Tokenizer mapping: {`"hello"`: 1, `"world"`: 2, `"how"`: 3, `"are"`: 4, `"you"`: 5}
After tokenization and creating n-gram sequences:
For `"Hello world"`: `[1, 2]`
For `"How are you"`: `[3, 4], [3, 4, 5]`
Combining all n-gram sequences:[[1, 2], [3, 4], [3, 4, 5]]
Maximum sequence length: `3`
After padding:[[0, 1, 2], [0, 3, 4], [3, 4, 5]]

For the following steps print the number of input sequences finally created for the actual given text.

[3 marks] Q6.4: In this phase of preparing your data for machine learning models, you'll separate the previously formatted input sequences into predictors (features) and labels (targets). By slicing the sequences, the last token of each sequence becomes the label, while the preceding tokens form the predictors. Convert the label tokens into one-hot encoded vectors using TensorFlow's utility function, facilitating effective categorical output prediction. Subsequently, divide your dataset into training and validation subsets using a 20% split for validation. Print the size of the train and validation subsets for the features and targets.

[8 marks] Q6.5: Create a simple LSTM-based model by defining a sequential architecture. Begin with an embedding layer that uses the total number of words as the input dimension, and an output dimension of 100. Follow this with an LSTM layer containing 150 units. Then, add a dense layer with the total number of words as the output dimension, using the softmax activation function. Compile the model with categorical cross-entropy as the loss function, the Adam optimizer, and accuracy as the metric. After defining the model, print its summary. Build and train the model for 20 epochs. After training, visualize the performance by plotting the training and validation accuracy and loss over the epochs. Is the model overfitting? Explain your observation. Create one more model of your choice (you may explore Bidirectional, LayerNormalization, Dropout, Attention and GRU etc) that improves upon the previous model in terms of overfitting. Print this new model summary, train for 20 epochs, and then plot the training and validation accuracy and loss.

[6 marks] Q6.6: Define the `generate_text()` function that takes a starting text, the desired number of additional words, a predictive model, the maximum sequence length, and a temperature parameter as inputs. Within the function, predict the subsequent word iteratively based on the evolving text. In each iteration, convert the current text into tokens, pad these tokens to the required sequence length, and use the model to predict the logits (unnormalized predictions generated by the last layer of a neural network before applying an activation function) for the next word. Adjust the logits by the temperature parameter, apply the softmax function to get probabilities, and sample the next word's index based on these probabilities. Map this index back to the corresponding word using the tokenizer, and append the word to the current text. Ultimately, return the expanded text that now includes the newly generated words, effectively extending the original text input. The temperature parameter in NLP controls the randomness of the predictions by adjusting the probability distribution of the next word. Lower temperatures make the model more confident and deterministic, often resulting in more repetitive and conservative text, while higher temperatures increase randomness, producing more diverse and creative outputs but also raising the risk of generating incoherent text. Demonstrate the function by generating text with temperature values 0.05 and 1.5 using the previously created model with less overfitting.

[3 marks] Q6.7: In the preprocessing step for NLP, removing stop words is often considered important. We did not perform stop word removal in our text generation task. Should we have done that? Explain reasons to support your answer.