

### Homework 3 - Neural Network Models - Spring 2025

**Essay Question:** *The current generation of deep neural network (NN) models (2010's-present) addressed many of the issues faced by model sets used in earlier waves of neural network research.* Write a one paragraph essay that starts with the preceding sentence. The following sentences of the paragraph should support this assertion by describing the novelty of each wave's model sets, the issues that limited the utility of these earlier models, and how deep NN models address these issues. Your paragraph should close with your opinion concerning the future prospects of the current generation of deep NN models and should discuss why you feel this way.

**Problem 1:** Consider a linear regression problem whose inputs  $x \in \mathbb{R}^n$  and whose targets are  $y = \theta^T x + n \in \mathbb{R}$  where  $\theta \in \mathbb{R}^n$  is some unknown parameter vector and  $n$  is a zero mean normally distribution random variable with variance  $\sigma^2$ . Let the model set,  $\mathcal{H}$ , consist of all linear models of the form  $h_w(x) = w^T x$  where  $w \in \mathbb{R}^n$  is the model's *weight vector*. Let the problem's loss function be the squared prediction error  $L(y, h_w(s)) = (y - h_w(s))^2$ .

Let  $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^N$  be a randomly selected dataset of  $N$  samples and let  $w^* \in \mathbb{R}^n$  be the model weight vector that minimizes the model's empirical risk over the dataset. Define the matrices,

$$\mathbf{X} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \hat{\mathbf{Y}} = \begin{bmatrix} h_{w^*}(x_1) \\ h_{w^*}(x_2) \\ \vdots \\ h_{w^*}(x_N) \end{bmatrix}, \quad \mathbf{n} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_N \end{bmatrix}$$

where  $x_k$  are the data input samples,  $y_k = \theta^T x_k + n_k$  are the data targets,  $n_k$  is the noise in the data targets, and  $w^*$  are the model weights minimizing the model's empirical risk over the dataset.

1. Let  $w^*$  be the optimal model weights for a given dataset  $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$  with  $N$  samples. Determine a closed form expression for  $w^*$  as a function of the unknown system parameters,  $\theta$ , the input data matrix  $\mathbf{X}$  for the given dataset, and the target noise vector,  $\mathbf{n}$ .
2. For the optimal model,  $h_{w^*}$ , determined above, find an expression for the true risk,  $\mathbb{E}[h_{w^*}]$ , as a function of the target noise covariance,  $\sigma^2$ .
3. Let us define the following matrix

$$\mathbf{H} \stackrel{\text{def}}{=} \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$$

Determine an expression for the expected value of the risk difference,  $\mathbb{E}_{\mathcal{D}} \left\{ \hat{R}_{\mathcal{D}}[h_{w^*}] - R[h_{w^*}] \right\}$  for a optimal models  $h_{w^*}$  for the randomly selected datasets  $\mathcal{D}$ .

**Hint:** You will probably need to use the fact that  $\mathbf{H}$  is a symmetric matrix with  $\text{trace}(\mathbf{H}) = n$  and  $\mathbf{H}^2 = \mathbf{H}$ .

**Problem 2:** Consider the neural network model shown in Fig. 1 that uses the notational conventions from the lecture notes (chapter 3) in which  $\mathbf{x}^{(\ell)}$  and  $\mathbf{s}^{(\ell)}$  are the outputs and inputs, respectively, of the nodes in the  $\ell$ th layer (including the bias node). Let  $\mathcal{W} = \{\mathbf{W}^{(\ell)}\}_{\ell=1}^3$  be a

collection of weights where  $\mathbf{W}^{(\ell)}$  is the the weight matrix connecting the outputs from layer  $\ell - 1$  to nodes of layer  $\ell$ . Let  $\hat{\mathbf{y}}_k = h_{\mathcal{W}}(\mathbf{x}_k)$  denote the prediction made by the model for the  $k$ th input  $\mathbf{x}_k$ . Assume that all initial weights are 0.5 and that we are using the squared error loss function  $L(\mathbf{y}_k, \hat{\mathbf{y}}_k) = |\mathbf{y}_k - \hat{\mathbf{y}}_k|^2$ .

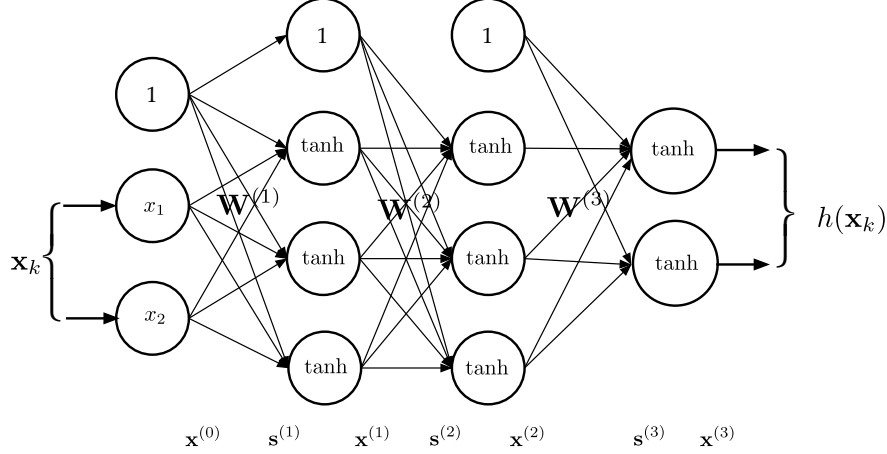


Figure 1: Problem 2

1. Compute the forward pass to obtain  $\mathbf{x}^{(\ell)}$ ,  $\mathbf{s}^{(\ell)}$ , and  $h(\mathbf{x}_k)$  for all layers assuming the input  $\mathbf{x}_k = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$  and that the target is  $\mathbf{y}_k = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Show the equations and scripts you used to compute these items.
2. Use the backward pass to obtain the error sensitivities,  $\delta^{(\ell)}$  for each layer with the same input/target and then compute the updated weights for each layer assuming a learning rate,  $\eta$ , of 0.3 and a dataset of size  $N = 5$ .

**Notebook Assignment (Deep Learning Classes):** Deep learning often requires extremely large datasets of labeled data. While it may not be hard to gather the input samples,  $x_k$ , of the dataset, labeling that data to generate targets,  $y_k$ , is expensive and time consuming. One way of addressing this issue is through *transductive inference* [1] (a.k.a. semi-supervised learning) that reasons from specific (training) cases to specific (test) cases. This approach may be possible when the data inputs are related to each other in a manner that can be described by a *graph*. In this case, the additional information provided by an input samples relationship to other samples may provide a way to infer the class membership of unlabeled samples.

This notebook assignment uses python class objects to instantiate a neural network model for transductive inference. Chapter 3 gave an example of such classes for a `SequentialModel` constructed from `DenseLayer` objects. This notebook develops class objects for a *Graph Convolution Network* (GCN) [2]. The GCN model is, essentially, a sequential model formed from `GCNLayers`. The input to the resulting model is the normalized *adjacency matrix*,  $\mathbf{A} \in \mathbb{R}^{N \times N}$ , for a graph with  $N$  nodes and a feature matrix  $\mathbf{X} \in \mathbb{R}^{N \times n}$  whose  $k$ th row is the *feature vector* for the  $k$ th node. The model's output is a matrix  $\hat{\mathbf{Y}} \in \mathbb{R}^{N \times m}$  whose  $k$ th row estimates the probability of node  $k$ 's class membership where there are  $m$  distinct classes.

Transductive inference is performed using a GCN in the following manner. The dataset is a labelled graph,  $G = (V, E, x, y)$  where  $V$  is a set of *nodes*,  $E \subset V \times V$  is a set of *edges*,  $x : V \rightarrow$

$\mathbb{R}^n$  maps each node,  $v \in V$ , onto a real-valued *feature vector*,  $x(v) \in \mathbb{R}^n$ , and  $y : V \rightarrow \{0, 1\}^m$  is a target for each node  $v$  such that  $y(v) \in \{0, 1\}^m$  is a one-hot encoded vector of one of  $m$  class names. The model set maps graph  $G$ 's adjacency matrix,  $A$ , and a feature matrix,  $X$ , onto the prediction,  $\hat{Y}$ , of each node's class probabilities. What makes this different from the learning-by-example problem, is that rather than assuming the model is trained using all targets in  $Y$ , we assume that the model is trained on a subset of  $Y_{\text{train}}$  of training targets. The problem is whether the model can generalize to nodes in  $G$  whose labels were not in the training set  $Y_{\text{train}}$ . In other words, can knowledge of the topology around training nodes be used to infer the class of nodes that were not in the training set.

This assignment has you write GCN class objects that are trained on the CORA dataset [3] This dataset consists of 2708 scientific publications classified into one of seven classes. The dataset consists of 5419 links indicating which publications have been cited. Each publication in the dataset is described by a one-hot encoded word vector indicating the absence/presence of the corresponding word from a dictionary of 1433 unique words *and* a label indicating which class the publication belongs to. We have pre-processing the CORA dataset and stored it as a `pkl` (pickle) archive. Pre-processing forms the normalized adjacency matrix  $A$  used in the GCN and creates the input feature matrix,  $X$ , and targets  $Y$ . The pickle archive also includes a text string describing each node class and two Boolean mask vectors indicating whether the node is to be treated as a training dataset (we use its target in training) or as a testing dataset (we use its target in evaluating the GCN's transductive inference abilities). We have also created a set of utility (`HW3utils.py`) functions for HW3 that you will need to import into your notebook. The specific tasks to be completed for this exercise are enumerated below.

1. **TO DO:** Open the pickle file `cora_dataset.pkl` and load its dataset (list object). Please element of the list in a numpy array. These elements are
  - Element 0: adjacency graph,  $A$
  - Element 1: feature vector matrix,  $X$ , for all graph nodes
  - Element 2: hot-encoded labels,  $Y$ , for all graph nodes
  - Element 3: Text strings describing each of the node classes
  - Element 4: Boolean mask indicating if graph node is for training
  - Element 5: Boolean mask indicating if graph node is for testing

**TO DO:** Check the shape of  $A$ ,  $X$ , and  $Y$ . Print out the strings describing each node class.

2. **TO DO:** Create a graphical convolution network (`GcnModel`) model class derive from the `SequentialModel` class object discussed in lecture. Main difference is that the `GcnModel` object is a list of `GcnLayers`, rather than `Dense Layers`. Then create a GCN layer class (`GcnLayer`) derived from the `DenseLayer` Class discussed in lecture.
3. **TO DO:** Instantiate a GCN model consisting of 2 `GcnLayers`. The first `GcnLayer` takes the node feature vector as an input and has 8 outputs. The second layer takes the first layer's output and maps them to one of the classes.

4. **TO DO:** Write a function that evaluates the instantiated model using inputs `A`, `X`, and `Y` with a specified train/test mask. Evaluate the untrained model's training loss/accuracy and testing loss/accuracy.
5. **TO DO:** Write a `fit` function for your model that takes the `A`, `X`, `Y` matrices as inputs and uses the training/testing masks to determine which samples are used for training and which are used for validation. Your function should return a history array containing the training loss, test loss, training accuracy, and test accuracy after every training epoch. Your training will use the `GradientTape` object to compute the loss gradients. After every 10 epochs print something to show progress (similar to a progress bar) and after 100 epochs print the model's training/testing loss/accuracy. Then train your model for 1000 epochs with a learning rate of 0.2. Note this training function is not optimized, so it will take 15-20 minutes.
6. **TO DO:** Evaluate the trained model's training loss/accuracy and testing loss/accuracy. Compare your results to that of the untrained model. Use the `training_curve` function of `HW3utils` to plot the data in the history object.

## References

- [1] O. Chapelle, B. Scholkopf, and A. Zien, editors. *Semi-supervised Learning*. MIT Press, 2006.
- [2] T.N. Kipf and M. Welling. Semi-supervised classification with graph convolution networks. arXiv:16-02907, 2016.
- [3] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.