Assignment 2 report

Name: Oswaldo Russian
CougarNetID: 1299650
Name on Kaggle leaderboard: Oswaldo Russian

Theory

Type out the answers to the questions in the table below, taking as many lines as you need. We recommend using the word equation editor for equations. If you would like to use Latex or handwrite the equations, you can paste a screenshot or picture of your work in the corresponding row instead. Handwritten equations which are not legible will receive no points.

A1	A python dictionary is a data structure, which is unordered, changeable, and indexed. The indexing works by having each value assigned to a corresponding 'key'. In our implementation, dictionaries are useful to store the output of each layer of our neural networks, since these outputs are also used in the parameter updates required for training.
A2	(i) The next raw response, is given by $$\boldsymbol{z}^{i+1} = \boldsymbol{h}^{i} \cdot \boldsymbol{W}^{i+1} + \boldsymbol{b}^{i+1}$$ (ii) The next layer output is given by $$\boldsymbol{h}^{i+1} = g(\boldsymbol{z}^{i+1})$$
A3	For a minibatch of size N, and C classes, the last layer's raw response, $\boldsymbol{z}^{l}$, is (N x C). The i^{th} row of $\boldsymbol{z}^{l}$ contains the scores for each class corresponding to the i^{th} row of the input, $\boldsymbol{X}$. (In this case we are considering that the last layer is not activated).
A4	The k^{th} component of the variable P contains the normalized probability associated with the k^{th} component of the output layer, $\boldsymbol{z}^{l}$ as shown below. When $k = y_i$, the data loss associated with the i^{th} row of $\boldsymbol{X}$ is given by $L_i = -log(P_{yi})$, which is the equation meant to be provided in this question. $$P_k = \frac{e^{z_k^l}}{\sum_j e^{z_j^l}}$$
A5	$$\boldsymbol{\delta}^{l} = \frac{\partial e^{i}}{\partial \boldsymbol{z}_k^l} = P_k - 1\ (for\ k\ =\ y_i),\ P_k\ (otherwise)$$
A6	i) Weight gradient: $$\frac{\partial e}{\partial \boldsymbol{W}^{i}} = \frac{\partial e}{\partial \boldsymbol{h}^{i}} \cdot \frac{\partial \boldsymbol{h}^{i}}{\partial \boldsymbol{W}^{i}} + 2\boldsymbol{\lambda}\boldsymbol{W}^{i}$$ *In my code, I pre-multiplied λ (regularization strength) by a factor of 0.5 in the regularization loss calculations, so the factor of '2' shown above does not appear in my gradient expression. ii) Bias gradient: it is the sum of all the entries in the upstream gradient, $\boldsymbol{\delta}_i$

	$$\frac{\partial e}{\partial \boldsymbol{b}^i} = \sum_j \boldsymbol{\delta}^i{}_j$$ For completeness, the upstream gradient is given by: $$\boldsymbol{\delta}^i = \frac{\partial e}{\partial \boldsymbol{h}^i}$$
A7	As shown in the class, such expression would be: $$\frac{\partial e}{\partial \boldsymbol{h}^{i-1}} = \frac{\partial e}{\partial \boldsymbol{h}^i} \cdot \frac{\partial \boldsymbol{h}^i}{\partial \boldsymbol{h}^{i-1}}$$ To account for a nonlinearity, g, we write: $$\frac{\partial e}{\partial \boldsymbol{h}^{i-1}} = \frac{\partial e}{\partial \boldsymbol{h}^i} \cdot \boldsymbol{W}^i \cdot g(\boldsymbol{h}^i)$$
A8	$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$
A9	$$\frac{\partial relu(x)}{\partial x} = 1\ (for\ x > 0), 0\ (otherwise)$$

Hyperparameter tuning strategy

(For each of the sections below, your reported test accuracy should approximately match the accuracy reported on Kaggle.)

Briefly describe the hyperparameter tuning strategies you used in this assignment.

My strategy for hyperparameter tuning:	While I understand the effect of regularization to help overfitting issues, the first thing I identified using the provided training module, was that I consistently achieved the highest accuracies (including validation and testing accuracies) for the lowest values of the regularization strength. As such, in my hyper-parameter optimization matrix, I set the regularization strength to zero across the board. *However,* my code **can** handle any reasonable value of the regularization strength. That is, I did consider it in my loss and gradient calculations.

	My hyper-parameter matrix included these values: Learning rate (0.1, 0.05, 0.03), Hidden layer size (20, 50, 80, 120), and Batch size (50, 100, 200) Forty-five combinations of these values for each of the models. This was my general parameter tuning strategy as of the time of writing this report. Additional combinations may be tried. If one of these yields better results, it will be reported below

---

Record your optimal hyperparameters and validation/test performance for the four different network types using the following forms:

Two-layer network with ReLU activation

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	200
Learning rate:	0.1
Hidden layer size:	120
Regularization coefficient:	0.0
Note:	Used default values for all other parameters.

Record the results for your best hyperparameter setting below:

Validation accuracy:	0.524
Test accuracy:	0.504

Two-layer network with Sigmoid activation

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	200
Learning rate:	0.1
Hidden layer size:	80
Regularization coefficient:	0.0
Note:	Used default values for all other parameters.

Record the results for your best hyperparameter setting below:

Validation accuracy:	0.436
Test accuracy:	0.435

Three-layer network with ReLU activation

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	200
Learning rate:	0.1
Hidden layer size:	100
Regularization coefficient:	0.0
Note:	Used default values for all other parameters.

Record the results for your best hyperparameter setting below:

Validation accuracy:	0.509
Test accuracy:	0.505

Three-layer network with Sigmoid activation

Best hyperparameters (if you changed any of the other default hyperparameters like initialization method, etc. please note that as well):

Batch size:	200
Learning rate:	0.1
Hidden layer size:	80
Regularization coefficient:	0.0
Note:	Used default values for all other parameters.

Record the results for your best hyperparameter setting below:

Validation accuracy:	0.414
Test accuracy:	0.396