CSCE 636: Deep Learning (Spring 2024)

Assignment #1

---

1. **(a)** The function ‘train_valid_split’ is used to split a given dataset and its corresponding labels into two subsets - a training dataset and a validation dataset. This is process is used to evaluate the performance of a model during training and to prevent overfitting.

   Now, why do we need to split the data into training and validation sets?

   1. By having a separate validation set, we can **evaluate the model's performance** on data it has not seen during training. This gives us an unbiased estimate of how well the model performs on new, unseen data.

   2. The validation set helps in **identifying** if the model is **overfitting** the training data. If the model performs well on the training set but poorly on the validation set, it might be memorizing the training data rather than learning to generalize.

   3. During the training process, we may adjust parameters tos optimize the model's performance. The validation set is crucial for **assessing** these **parameters' effectiveness** without using the test set, which should only be used sparingly to avoid information leakage.

   The ‘train_valid_split’ function facilitates the separation of the dataset into training and validation sets, enabling us to do a proper model evaluation and tuning during the training process.

   (b) Yes, it is generally correct to re-train the model on the entire training set before testing, especially if we made any adjustments or fine-tuned the model based on validation set performance. The reasons are:

   1. We initially split our data into training and validation sets, the model learns from the training set. After we've fine-tuned or adjusted our model based on the validation set's feedback, we typically want the final model to **benefit from all available training data**. By re-training on the entire training set, we can ensure that the model captures the patterns present in the entire dataset.

   2. During the training process, we might have adjusted hyperparameters or made other modifications to the model to improve its performance on the validation set. To ensure that these adjustments are incorporated into the final model, we need to re-train the model on the entire training set.

3. Training on a larger dataset often leads to better learning. If our model was fine-tuned on a subset of the data, re-training on the full training set allows the model to learn more diverse patterns, potentially improving its ability to generalize to unseen data.

However, we need to be cautious about overfitting to the Training Set: If our model was extensively fine-tuned based on the validation set, there's a risk of overfitting to the validation set.

(d) The third feature, which is always set to 1, commonly referred to as the "bias term" or "intercept term" in the feature vector is essential for several reasons:

1. The bias term allows the model to have a non-zero output, i.e. make predictions, even when all input features are zero or close to zero. This is crucial for capturing the constant term in the underlying relationship between the features and the target variable. Without the bias term, the model would be forced to pass through the origin, which may not be appropriate for many real-world datasets.
2. In many cases, the input features might not be centred around zero. Including a bias term helps the model adjust for the offset, ensuring that the model is not biased by the inherent scale or shift of the input data.
3. The bias term provides the model with additional degrees of freedom, enabling it to better fit the training data and potentially improve the overall performance.

2. (a)

Given the cross-entropy error function:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \cdot w^T x_n}\right)$$

(b) To compute the gradient ∇E(w), we need to find the partial derivatives of the loss function with respect to the weight vector w. Let's compute the gradient step by step:

We will compute the derivative with respect to the weight vector w for one training sample (x, y).

Lets denote $z = y \cdot w^T x$, so $e^{-z} = e^{-y \cdot w^T x}$

Now, we can compute the derivative:

$$\frac{\partial E_{in}}{\partial w} = -\frac{1}{N} \sum_{n=1}^{N} \frac{e^{-y_n \cdot w^T x}}{1 + e^{-y_n \cdot w^T x}} \cdot (-y_n) \cdot x_n$$

For one training sample (x, y), this simplifies to:

$$\frac{\partial E_{in}}{\partial w} = -\frac{e^{-y_n \cdot w^T x}}{1 + e^{-y_n \cdot w^T x}} \cdot y \cdot x$$

$$= \frac{-y \cdot x}{1 + e^{y_n \cdot w^T x}}$$

(c) The decision boundary in logistic regression is linear, meaning it separates the feature space into two regions based on a linear combination of input features. The decision boundary is given by $(w^T x) = 0$. Using the threshold of 0.5, the predicted class of x is 1 if $\theta(w^T x) >= 0.5$ and -1 otherwise. However, this decision boundary is linear. The inefficiency comes from the fact that the sigmoid function $\theta(z)$ saturates for extreme values of z, leading to very small gradients during training. This can slow down the convergence of optimization algorithms.

The sigmoid function is needed in prediction to convert the linear combination $(w^T x)$ into a probability between 0 and 1. The thresholding at 0.5 is just a convention; we could choose a different threshold based on our specific needs.

(d) If the prediction rule is changed to use a threshold of 0.9, the decision boundary will shift to a region where the predicted class is 1 if $\theta(w^T x) >= 0.9$ and -1 otherwise. This will move the decision boundary towards more confident predictions. However, the decision boundary is still linear.

The linear decision boundary is determined by the linear combination $(w^T x)$. Changing the threshold only affects the confidence level required for a positive prediction, but it doesn't change the linearity of the decision boundary.

(e) The essential property of logistic regression that results in the linear decision boundary is its monotonic nature and being bounded on R. The sigmoid function maps the linear combination $(w^T x)$ to a range between 0 and 1, effectively creating a smooth transition between the two classes. The decision boundary is set at $(w^T x) = 0$, making it a linear equation.

3. Found the best-performing model, after experimenting with different hyperparameters and training multiple logistic regression models using the optimization algorithms - Batch Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent, with different learning rates and maximum iterations.
The best logistic regression model achieved a test accuracy of 93.51%, using 'SGD' with the parameters - Learning Rate: 1.0, Max Iterations: 50

4. Found the best-performing model, after experimenting with different hyperparameters and training the logistic regression models using the optimization algorithm - Mini-Batch Gradient Descent, with different learning rates and maximum iterations.
The best logistic regression model achieved a test accuracy of 87.89%, with the parameters - Learning Rate: 1.0, Max Iterations: 200

5. **(a)**

1. Similarities between Sigmoid and Softmax –

a. Logistic Function: Both classifiers utilize logistic functions to model the relationship between the input features and the output class probabilities. The logistic function transforms the linear combination of input features and model parameters into a probability value.
b. Optimization: Both classifiers are trained using optimization algorithms to minimize a loss function. Gradient-based optimization algorithms, such as gradient descent or its variants, are commonly used to update the model parameters iteratively.
c. Binary Classification: Both classifiers can be used for binary classification tasks. While the softmax classifier is primarily designed for multi-class classification, it can be adapted for binary classification by considering two classes and using binary cross-entropy loss.
d. Parameter Interpretation: The parameters (weights) learned by both classifiers represent the strength of the relationship between input features and the output class probabilities. Positive weights indicate a positive correlation with the target class, while negative weights indicate a negative correlation.
e. Probabilistic Interpretation: Both classifiers provide probabilistic interpretations of their predictions. They output class probabilities that indicate the likelihood of each class given the input features. This probabilistic output can be useful for uncertainty estimation and decision-making.
f. Bias Term: Both classifiers typically include a bias term (intercept) in addition to the weights associated with input features. The bias term allows the classifiers to capture the baseline probability of the target class independent of the input features.

2. Differences between Sigmoid and Softmax –

a. Decision Boundary: One significant difference between the two classifiers is the decision boundary they learn. The softmax classifier learns a multi-class decision boundary that can separate multiple classes simultaneously, while the sigmoid classifier learns binary decision boundaries for each class separately.

b. Output: The output of the softmax classifier represents the probability distribution over all classes, allowing it to handle multi-class classification naturally. In contrast, the output of the sigmoid classifier represents the probability of the positive class in binary classification tasks.

c. Loss Function: The loss function used in the softmax classifier is the categorical cross-entropy, which measures the difference between the predicted probability distribution and the true distribution of classes. In contrast, the sigmoid classifier

typically uses binary cross-entropy loss, which measures the difference between the predicted probability of the positive class and the true label.

d. Training Dynamics: The softmax classifier appears to converge slowly when compared to the sigmoid classifier.

e. Scalability: In terms of scalability, the softmax classifier might be more suitable for problems with a large number of classes, as it can handle multi-class classification tasks directly. In contrast, the sigmoid classifier is typically used for binary classification tasks but can be extended to multi-label classification by training multiple binary classifiers.

(b)

$$h_w(x) = \frac{1}{e^{w_1^T x} + e^{w_2^T x}} \begin{bmatrix} e^{w_1^T x} \\ e^{w_2^T x} \end{bmatrix}.$$

$$= \begin{bmatrix} \frac{1}{1+e^{-\hat{w}^T x}} \\ 1 - \frac{1}{1+e^{-\hat{w}^T x}} \end{bmatrix} = \begin{bmatrix} \theta_{\hat{w}}(x) \\ 1 - \theta_{\hat{w}}(x) \end{bmatrix}$$

where $\hat{w} = w_1 - w_2$. Then,

$$\frac{\partial E(w)}{\partial w_1} = \frac{1}{1+e^{y \hat{w}^T x}} (-y) x$$

$$\frac{\partial E(w)}{\partial w_2} = \frac{1}{1+e^{y \hat{w}^T x}} (-y)(-x)$$

$$= -\frac{\partial E(w)}{\partial w_1}.$$

$$\frac{\partial E(\hat{w})}{\partial \hat{w}} = \frac{1}{1+e^{y \hat{w}^T x}} (-y) x$$

Updating is,

$$\begin{bmatrix} w_{1,new} \\ w_{2,new} \end{bmatrix} = \begin{bmatrix} w_1 - \eta \frac{\partial E(w)}{\partial w_1} \\ w_2 - \eta \frac{\partial E(w)}{\partial w_2} \end{bmatrix}.$$

and $\hat{w}_{new} = \hat{w} - \hat{\eta} \frac{\partial E(\hat{w})}{\partial \hat{w}}$.

Then,

$$w_{1,new} - w_{2,new}$$
$$= w_1 - w_2 - \eta \times 2 \times \frac{\partial E(\hat{w})}{\partial \hat{w}}$$
$$= \hat{w} - \hat{\eta} \frac{\partial E(\hat{w})}{\partial \hat{w}} = \hat{w}_{new}$$

So, this means that, when learning rate of softmax is set to half of that of sigmoid, w1 − w2 = w holds for all training steps.