From Complex Algorithms to Precise Detection: Optimizing Deep Learning & Neural Networks for Breast Cancer Classification

Debashis Chatterjee, Prithwish Ghosh*

April 2024

1 Detailed Methodology

1.1 Neural Networks Classifier

Let X be the input feature matrix with n samples and m features, where each feature corresponds to one of the parameters:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & \dots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,m} \end{bmatrix}$$

Let y be the target variable, representing the diagnosis (Malignant or Benign) for each sample in X. The target variable y is a vector of length n. Each neuron in the input layer corresponds to one of the features in X. The number of neurons in the input layer equals the number of features m. The output layer consists of two neurons corresponding to the two possible classes of the target variable (Malignant or Benign) [1].

Let $W^{(l)}$ and $b^{(l)}$ denote the weights and biases of the lth layer, respectively. The activation function $f^{(l)}$ is applied to the weighted sum of inputs in each neuron of the hidden layers.

The forward propagation process of the neural network can be described as follows:

$$\begin{aligned} &\text{For } l = 1 \text{ to } L-1: \\ &\quad Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)} \\ &\quad A^{(l)} = f^{(l)}(Z^{(l)}) \\ &\text{Output layer:} \\ &\quad Z^{(L)} = W^{(L)} A^{(L-1)} + b^{(L)} \\ &\quad A^{(L)} = \text{softmax}(Z^{(L)}) \end{aligned}$$

where L is the total number of layers in the neural network, $Z^{(l)}$ is the weighted sum of inputs for the lth layer, $A^{(l)}$ is the output of the lth layer after applying the activation function, and softmax is the softmax function applied to the output layer to obtain class probabilities [24].

The neural network is trained using backpropagation with an optimization algorithm such as stochastic gradient descent (SGD) or Adam.

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{2} y_{ij} \log(\hat{y}_{ij})$$

where n is the number of samples, y_{ij} is the true label (1 if the sample i belongs to class j, 0 otherwise), and $\hat{y}_{ij}$ is the predicted probability of class j for sample i obtained from the output layer.

The neural network classifier learns the optimal weights and biases $W^{(l)}$ and $b^{(l)}$ by iteratively updating them through the training process until convergence.

1.2 Adaptive Decision Learner (ADL)

The Adaptive Decision Learner (ADL) is a machine learning algorithm that classifies data based on input features. In the context of the given parameters and target variable, the ADL algorithm aims to learn a decision boundary that separates instances of different diagnoses (Malignant or Benign) based on the values of the provided features [2].

Let $X = [x_1, x_2, \ldots, x_n]$ denote the input features, where x_i represents the value of the i-th feature for a given instance. The target variable y represents the diagnosis, with $y = 1$ indicating Malignant and $y = 0$ indicating Benign.

The ADL algorithm learns a decision function $f(X)$ that maps input features to predicted diagnoses. This decision function can be represented as:

$$f(X) = \text{sign}(\sum_{i=1}^{n} w_i x_i + b)$$

where w_i are the weights associated with each feature x_i and b is the bias term. The sign$(\cdot)$ function returns $+1$ for positive inputs and -1 for negative inputs.

The ADL algorithm learns the weights w_i and bias b by minimizing a loss function, typically the hinge loss function for binary classification problems:

$$\min_{w,b} \sum_{i=1}^{m} \max(0, 1 - y_i(\sum_{j=1}^{n} w_j x_{ij} + b))$$

where m is the number of training instances, x_{ij} represents the i-th feature of the j-th instance, and y_i is the true label (Malignant or Benign) of the i-th instance [4].

During the learning process, the ADL algorithm adjusts the weights and bias to minimize the classification error on the training data. Once trained, the ADL algorithm can classify new instances by evaluating the decision function $f(X)$.

Overall, the ADL algorithm is a binary classification algorithm that learns to classify instances based on a set of input features, such as the ones provided in the given parameters, and predicts the diagnosis (Malignant or Benign) based on learned decision boundaries [9].

1.3 Logistic Regression Classifier

Let $X = [x_1, x_2, \ldots, x_n]$ denote the input features, where x_i represents the value of the ith feature. The parameters of the logistic regression model are denoted by $\beta = [\beta_0, \beta_1, \ldots, \beta_n]$, where β_0 is the intercept term [21].

The logistic regression model predicts the probability that the target variable y belongs to a particular class (e.g., Malignant or Benign) using the logistic function:

$$P(y = 1|X, \beta) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}}$$

where e is the base of the natural logarithm.

To train the logistic regression model, we use a dataset with m samples, denoted by $(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(m)}, y^{(m)})$, where $X^{(i)}$ represents the features of the ith sample and $y^{(i)}$ represents the corresponding target variable [23] [21].

Logistic regression aims to maximize the likelihood of the observed data given the parameters β. This is equivalent to minimizing the negative log-likelihood, also known as the cross-entropy loss function:

$$\mathcal{L}(\beta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log P(y=1|X^{(i)},\beta) + (1-y^{(i)})\log(1-P(y=1|X^{(i)},\beta))\right]$$

To minimize the loss function and find the optimal parameters β, we typically use optimization algorithms such as gradient descent or Newton's method [12].

Once the logistic regression model is trained, we can use it to predict the probability that new samples belong to a particular class based on their features. The predicted class label $\hat{y}$ is usually determined by applying a threshold (e.g., 0.5) to the predicted probabilities [21].

1.4 Random Forest Classifier

The Random Forest classifier is an ensemble learning method that combines multiple decision trees to make predictions. Each decision tree in the forest is trained on a random subset of the training data and a random subset of the features. The final prediction is made by aggregating the predictions of all the individual trees in the forest [18].

The Random Forest algorithm works as follows:

1. Randomly select k features from the total m features.
2. Among the k features, find the best feature and split the data based on the feature value to minimize impurity. This process is repeated recursively for each child node until a stopping criterion is met.
3. Repeat steps 1 and 2 to create n decision trees.
4. During prediction, each tree in the forest predicts the class label. The final prediction is made by taking a majority vote (for classification) or averaging (for regression) over all the trees.

Mathematically, let $T_i(X)$ represent the i-th decision tree in the forest, and $\hat{y}_i$ represent the prediction of the i-th tree. Then, the prediction of the Random Forest classifier is given by:

$$\hat{y}_{\text{RF}}(X) = \text{mode}\left(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n\right)$$

where mode represents the mode of the predictions, i.e., the most frequent prediction among all the trees [3] [18].

The Random Forest algorithm is robust to overfitting and noisy data and often provides high accuracy in classification tasks. It also provides measures of feature importance, which can help in feature selection and understanding the underlying relationships in the data.

Support Vector Machine (SVM) Classifier

The Support Vector Machine (SVM) classifier aims to find the optimal hyperplane that separates the data points into different classes while maximizing the margin between the classes. Here's the mathematical theory of the SVM classifier concerning the given parameters and target variable:

Given a set of training data $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x_i represents the feature vector comprising the parameters (radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension) for the i-th sample, and y_i represents the corresponding target variable (Diagnosis: Malignant or Benign) [22].

The SVM classifier learns to classify the data points by solving the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

subject to:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

Where: - $\mathbf{w}$ is the weight vector representing the coefficients of the hyperplane, - b is the bias term, - ξ_i are slack variables that allow for misclassification of some data points, - C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error [11] [22].

The decision function of the SVM classifier is given by:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

The optimization problem can be solved using techniques such as the Sequential Minimal Optimization (SMO) algorithm or quadratic programming.

In summary, the SVM classifier learns to find the hyperplane that best separates the data points in the feature space, based on the given parameters while maximizing the margin between classes and minimizing the classification error.

XGBoost: Extreme Gradient Boosting

XGBoost works by sequentially adding decision trees to an ensemble, with each new tree correcting the errors made by the existing ensemble. The ensemble's output is the weighted sum of the predictions of all the trees [5, 15].

The objective function of XGBoost can be expressed as:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k).$$

Where n is the number of samples, K is the number of trees, y_i is the true label of the ith sample, $\hat{y}_i$ is the predicted label of the ith sample, f_k is the kth tree in the ensemble, l is the loss function measuring the difference between the actual and predicted labels, and Ω is the regularization term penalizing the complexity of the trees [17, 8].

The prediction of the ensemble for a new sample can be computed as:

$$\hat{y} = \sum_{k=1}^{K} f_k(x)$$

where $f_k(x)$ is the prediction of the kth tree for the input features x.

In each iteration, XGBoost fits a new tree to the negative gradient of the loss function concerning the predictions of the current ensemble. This process is repeated until a stopping criterion is met, such as reaching a maximum number of trees or no further improvement in the objective function.

1.5 Convolutional Neural Network (CNN) Model

Let $X = [x_1, x_2, \ldots, x_N]$ denote the input data, where each x_i represents a cell nucleus with the following features mentioned above.

The target variable, y, represents the diagnosis of the cell nucleus, which can be Malignant or Benign [20].

The CNN architecture comprises several key components:

1. Convolutional Layers: These layers consist of filters that slide across the input data, extracting features relevant to the task. The output of each filter is computed using convolution operation [16] [20].

2. Pooling Layers: These layers reduce the dimensionality of the feature maps generated by the convolutional layers while retaining the most essential information. Common pooling operations include max pooling and average pooling.

3. Fully Connected Layers: These layers take the flattened output of the previous layers and perform classification based on the extracted features. Each neuron in the fully connected layers is connected to every neuron in the previous layer.

Mathematically, the output of a CNN can be represented as follows:

$$\hat{y} = \text{softmax}(W_{\text{fc}} \cdot \text{ReLU}(W_{\text{fc}-1} \cdot \text{maxpool}(W_{\text{conv}-L} * X + b_{\text{conv}-L}) + b_{\text{fc}-1}) + b_{\text{fc}})$$

, mentioned above

The parameters W and b are learned during the training process using optimization algorithms such as stochastic gradient descent (SGD) or Adam.

The objective of the CNN is to minimize the cross-entropy loss function:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Where:

- N is the number of samples,
- C is the number of classes,
- $y_{i,c}$ is the true probability that sample i belongs to class c, and
- $\hat{y}_{i,c}$ is the predicted probability that sample i belongs to class c.

By optimizing the parameters W and b of the CNN using gradient descent, the network learns to classify cell nuclei into different classes (Malignant or Benign) based on their features.

1.6 AdaBoost Classifier

The AdaBoost classifier is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. It is particularly effective for classification tasks with binary outcomes, such as diagnosing medical conditions like Malignant or Benign tumors based on cell nucleus features. Let's denote the input features as $X = [x_1, x_2, \ldots, x_n]$, where x_i represents a vector of features for sample i, and the corresponding target variable as y [10, 7].

Given a set of weak classifiers $h_1(x), h_2(x), \ldots, h_T(x)$, the AdaBoost algorithm works as follows [19]:

1. Initialize the sample weights $w_i = \frac{1}{n}$, where n is the number of samples.

2. For $t = 1$ to T: a. Train a weak classifier $h_t(x)$ using the weighted training data, where the weight of each sample is determined by w_i. b. Compute the classification error ϵ_t of $h_t(x)$ on the training data:

$$\epsilon_t = \sum_{i=1}^{n} w_i \cdot 1(h_t(x_i) \neq y_i)$$

where $1(\cdot)$ is the indicator function. c. Compute the weight α_t of $h_t(x)$ as:

$$\alpha_t = \frac{1}{2} \log\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

d. Update the sample weights:

$$w_i \leftarrow w_i \cdot \exp\left(-\alpha_t \cdot y_i \cdot h_t(x_i)\right)$$

e. Normalize the sample weights:

$$w_i \leftarrow \frac{w_i}{\sum_{i=1}^{n} w_i}$$

3. Compute the final strong classifier $H(x)$ as a weighted sum of weak classifiers:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)$$

4. Output the final strong classifier $H(x)$.

In the context of diagnosing medical conditions based on cell nucleus features, the weak classifiers $h_t(x)$ could be decision stumps or decision trees trained on individual features such as radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension.

1.7 The Long Short-Term Memory (LSTM) Classifier

To formulate the mathematical theory of the Long Short-Term Memory (LSTM) classifier concerning the given parameters and target variable, we need to define the architecture of the LSTM network and how it processes the input features to make predictions [13, 14].

Let's denote the input features as $X = [x_1, x_2, \ldots, x_T]$, where each x_t represents a vector of features at time step t. In this case, the input features consist of the parameters provided: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension[6] [13, 14].

The target variable is the Diagnosis, which can be Malignant or Benign.

The LSTM architecture consists of several key components:

1. Input Gate (i_t): This gate controls the information stored in the cell state. It takes input features x_t at each time step t and computes the input gate activation i_t using the sigmoid function.

2. Forget Gate (f_t): This gate controls the information discarded from the cell state. It takes input features x_t at each time step t and computes the forget gate activation f_t using the sigmoid function.

3. Output Gate (o_t): This gate controls the information output from the cell state. It takes input features x_t at each time step t and computes the output gate activation o_t using the sigmoid function.

4. Candidate Cell State ($\tilde{c}_t$): This represents the new candidate values for the cell state. It takes input features x_t at each time step t and computes the candidate cell state $\tilde{c}_t$ using the hyperbolic tangent (tanh) function.

5. Cell State (c_t): This represents the current cell state. It is updated based on the input gate, forget gate, and candidate cell state computed at each time step.

6. Hidden State (h_t): This represents the output of the LSTM unit at each time step. It is computed by applying the output gate to the cell state after passing it through the tanh function.

By optimizing the parameters of the LSTM network using gradient descent, the network learns to classify input data sequences into different classes (Malignant or Benign) based on the temporal dependencies of the input features.

References

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), 2018.

[2] Pasquale Ardimento, Nicola Boffoli, VN Convertini, Giuseppe Visaggio, et al. Decision table for adaptive e-learning systems. *Education in a Technological World: Communicating Current and Emerging Research and Technological Efforts, Publisher: Formatex Research Center, Editors: A. Mendez-Vilas*, 2011.

[3] Mariana Belgiu and Lucian Drăguţ. Random forest in remote sensing: A review of applications and future directions. *ISPRS journal of photogrammetry and remote sensing*, 114:24–31, 2016.

[4] Andriy Burkov and Brahim Chaib-draa. Effective learning in adaptive dynamic systems. In *AAAI Spring Symposium: Game Theoretic and Decision Theoretic Agents*, pages 1–7, 2007.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] Alessandro Di Nuovo. Long-short term memory networks for modelling embodied mathematical cognition in robots. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[7] Carlos Domingo, Osamu Watanabe, et al. Madaboost: A modification of adaboost. In *colt*, pages 180–189, 2000.

[8] Prithwish Ghosh. Breast cancer wisconsin (diagnostic) prediction.

[9] Juan Miguel Gómez-Berbís and Ángel Lagares-Lemos. Adl-mooc: Adaptive learning through big data analytics and data mining algorithms for moocs. In *Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings 2*, pages 269–280. Springer, 2016.

[10] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[11] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[12] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[13] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *2016 international conference on platform technology and service (PlatCon)*, pages 1–5. IEEE, 2016.

[14] Mengshi Li, Zhiyuan Zhang, Tianyao Ji, and QH Wu. Ultra-short term wind speed prediction using mathematical morphology decomposition and long short-term memory. *CSEE Journal of Power and Energy Systems*, 6(4):890–900, 2020.

[15] Rory Mitchell and Eibe Frank. Accelerating the xgboost algorithm using gpu computing. *PeerJ Computer Science*, 3:e127, 2017.

[16] Xiao-Xiao Niu and Ching Y Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318–1325, 2012.

[17] Adeola Ogunleye and Qing-Guo Wang. Xgboost model for chronic kidney disease diagnosis. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(6):2131–2140, 2019.

[18] Mahesh Pal. Random forest classifier for remote sensing classification. *International journal of remote sensing*, 26(1):217–222, 2005.

[19] Robert E Schapire. Explaining adaboost. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer, 2013.

[20] Alok Sharma and Kuldip K Paliwal. Linear discriminant analysis for the small sample size problem: an overview. *International Journal of Machine Learning and Cybernetics*, 6:443–454, 2015.

[21] Sandro Sperandei. Understanding logistic regression analysis. *Biochemia medica*, 24(1):12–18, 2014.

[22] Shan Suthaharan and Shan Suthaharan. Support vector machine. *Machine learning models and algorithms for big data classification: thinking with examples for effective learning*, pages 207–235, 2016.

[23] Raymond E Wright. Logistic regression., 1995.

[24] Yu-chen Wu and Jun-wen Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102:1645–1656, 2018.