AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19AIE205
Python for Machine Learning

Sleep stage classification using ML Algorithms

PRESENTED BY :-

Aadharsh Aadhithya - CB.EN.U4AIE20001

Anirudh Edpuganti - CB.EN.U4AIE20005

Madhav Kishor - CB.EN.U4AIE20045

Onteddu Chaitanya Reddy - CB.EN.U4AIE20033

Pillalamarri Akshaya - CB.EN.U4AIE20049

Acknowledgment

We would like to express our special thanks of gratitude to our teacher (Dr. Neethu Mohan Mam) who gave us the golden opportunity to do this wonderful project on the topic (Sleep Stage Classification Using Machine Learning Algorithms), which also helped us in doing a lot of research and we came to know about so many new things. We are thankful for the opportunity given.

Abstract

This is a report regarding the efficiency of different Machine Learning algorithms to predict different stages of sleep. We build four Supervised Machine Learning Algorithms without explicitly using libraries and building each of them from scratch.
We have used 4 algorithms namely, Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbours (KNN), and Multi-Layer Neural Networks.
The dataset we have considered is highly non-linear and very big. The dataset contains approximately 8000 samples with 7 features. We have taken the hypnogram of 10 different people and formulated the data according to the needs of different Machine Learning algorithms. This report will show the methods and techniques used by different algorithms and also report the accuracy of each of the models after being trained.

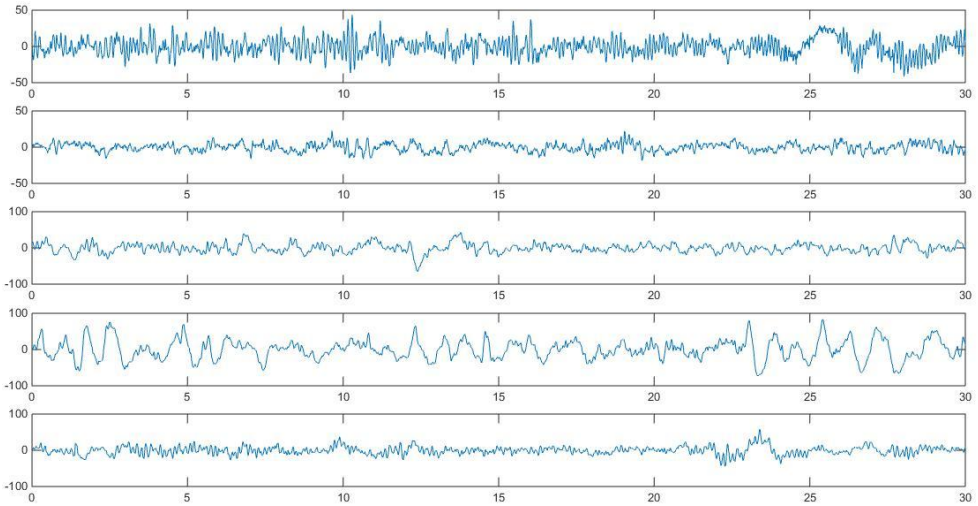
Introduction

Sleep is essential for maintaining and regulating a variety of biological activities. Non-rapid eye movement (NREM) and rapid eye movement (REM) sleep stages are the two most common types of sleep. The NREM and REM sleep stages are linked and alternate cyclically throughout the sleep cycle, with imbalanced cycling or the lack of sleep stages causing sleep disorders.
Sleep problems, which result in poor sleep quality, are unfortunately frequently overlooked.
Using EEG signals, we will identify sleep stages and find sleep problems using machine learning methods.
Using electrodes attached to a patient's scalp, an electroencephalogram (EEG) is a cost-effective and often non-invasive tool for monitoring and recording electrical impulses and voltage changes from neurons in the brain. These sensors enable the measurement of signals that can be stored for a long time, such as while sleeping. EEG can thus provide valuable information into a patient's neural activity with the goal of

analyzing and evaluating sleep quality in order to treat and address health problems related to sleep disorders.

Sleep Stage Classification

According to the AASM manual, sleep EEG consists of 5 stages.



From top to bottom, this diagram depicts the stages of Wake, N1, N2, N3, and REM.

-W: The awake state (stage W) is defined by alpha or rapid frequency bands

occupying more than half of the epoch, frequent eye movements, and a high EMG tone.

-N1: Stage N1 is defined as alpha occupying more than 50% of the epoch, with theta activity, slow rolling eye movements, and vertex waves visible.

-N2: Sleep spindles or K-complexes (less than 3 minutes apart) are assessed as stage N2.

-N3: Stage N3 is distinguished by the presence of delta activity in more than 20% of the period length.

-REM: When saw-tooth waves, fast eye movements, and the lowest EMG signals are present during sleep scoring, the epoch is labelled as REM are observed through each epoch.

Random Forest

"Random Forest is a classifier that combines a number of decision trees on different subsets of a dataset and averages the results to increase the dataset's prediction accuracy."

Bootstrapping

Bootstrapping: The process of creating random data is bootstrapping

Why Bootstrapping?

We need to bootstrap the data to make it less sensitive to the original data and train the model to attain the maximum accuracy for a diverse set of datasets.

Aggregation

Aggregation: The process of combining results from multiple models is called aggregation

- Bootstrapping and Aggregation together called Bagging

We will select data and bootstrap datasets

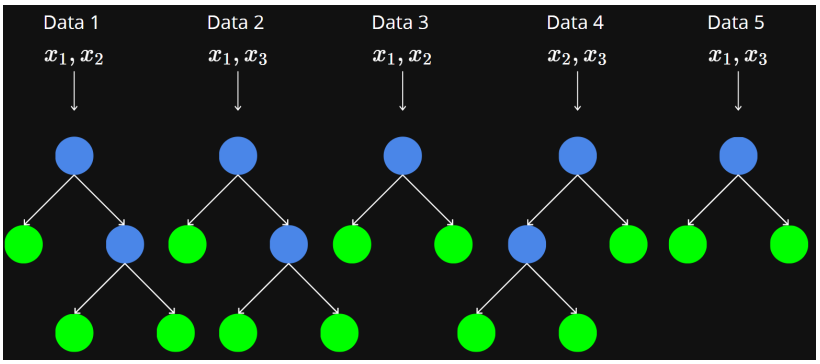
We will randomly select the subset of each features for each tree and use only those for training

Data 1	Data 2	Data 3	Data 4	Data 5
x_1, x_2	x_1, x_3	x_1, x_2	x_2, x_3	x_1, x_3

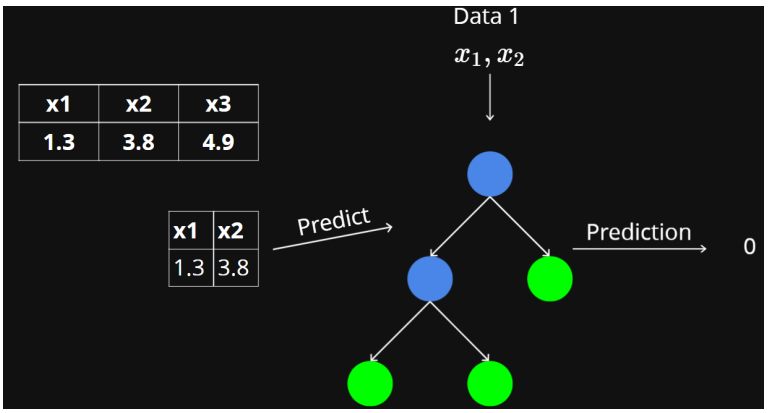
For the first case , we'll only use features x1,x2

In the second case we use x1 and x3 ,In 3rd case x1 and x2 ,In 4th case x2,x3 ,In 5th x1,x3

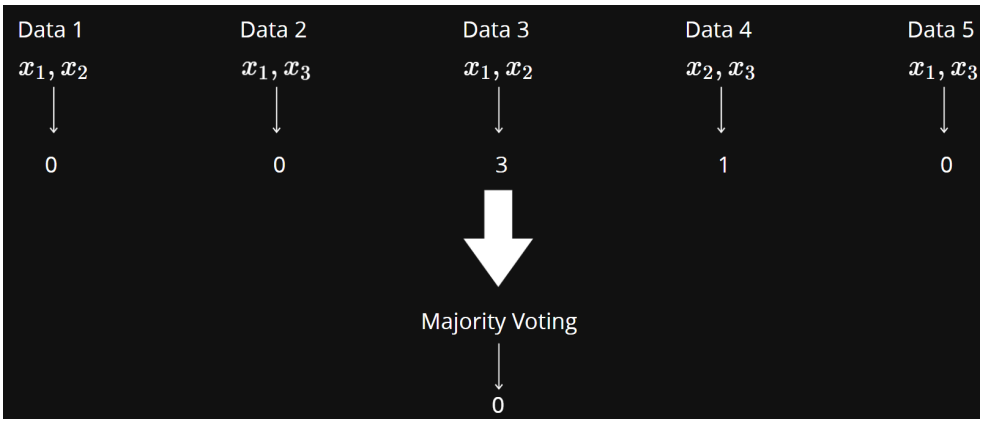
Now we will build trees



We will take a new data point and we'll pass the data point through each tree one by one and note down the predictions .



Now we will combine all predictions for classification,as a classification problem we will take the majority voting .Here the prediction from our random forest is 0



KNN

The K-NN method assumes that the new case/data and existing cases are similar and places the new case in the category that is most similar to the existing categories.

The K-NN method stores all available data and classifies a new data point based on its similarity to the existing data.

Things that are similar are close to each other. This means that new data can be quickly sorted into a well-defined category using the K-NN method.

The K-NN algorithm is a non-parametric algorithm, which means it makes no assumptions about the underlying data.

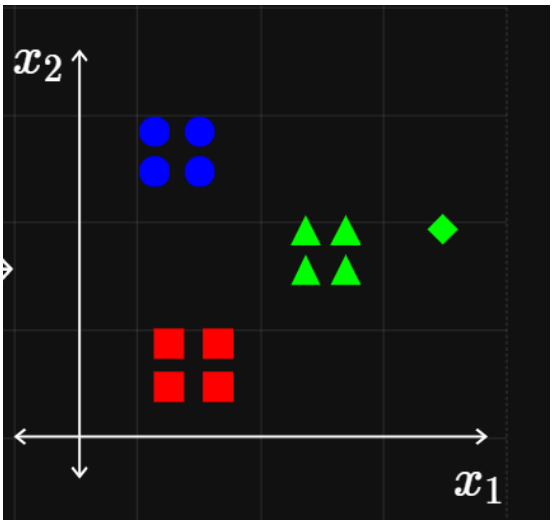
It's also known as a lazy learner algorithm since it doesn't learn from the training set right away; instead, it saves the dataset and performs an action on it when it comes time to classify it.

During the training phase, the KNN algorithm simply stores the dataset, and when it receives new data, it classifies it into a category that is quite similar to the new data.

Firstly, we will choose the number of neighbours, so we will choose the k=1

Next, we will calculate the Euclidean distance between the data points.We use euclidean distance and find distance between two points By calculating the Euclidean distance we got the nearest neighbours,

We will see the nearest neighbours for that data point here in the case the one yellow colored node represents the nearest node.



Then we will see which class that nearest neighbours belong to and classify the new data point as the same class.

SVM

Support Vector Machine is one of the supervised machine learning algorithms. In SVM we should find the hyperplane in an N-Dimensional space that classifies the data point. It is basically a binary classifier which classifies whether the given data point belongs to the particular class or not.

How can we use SVM to classify multiple classes??

There are two methods for using SVM for classifying data into multiple classes

1. One vs One SVM
2. One vs other SVM

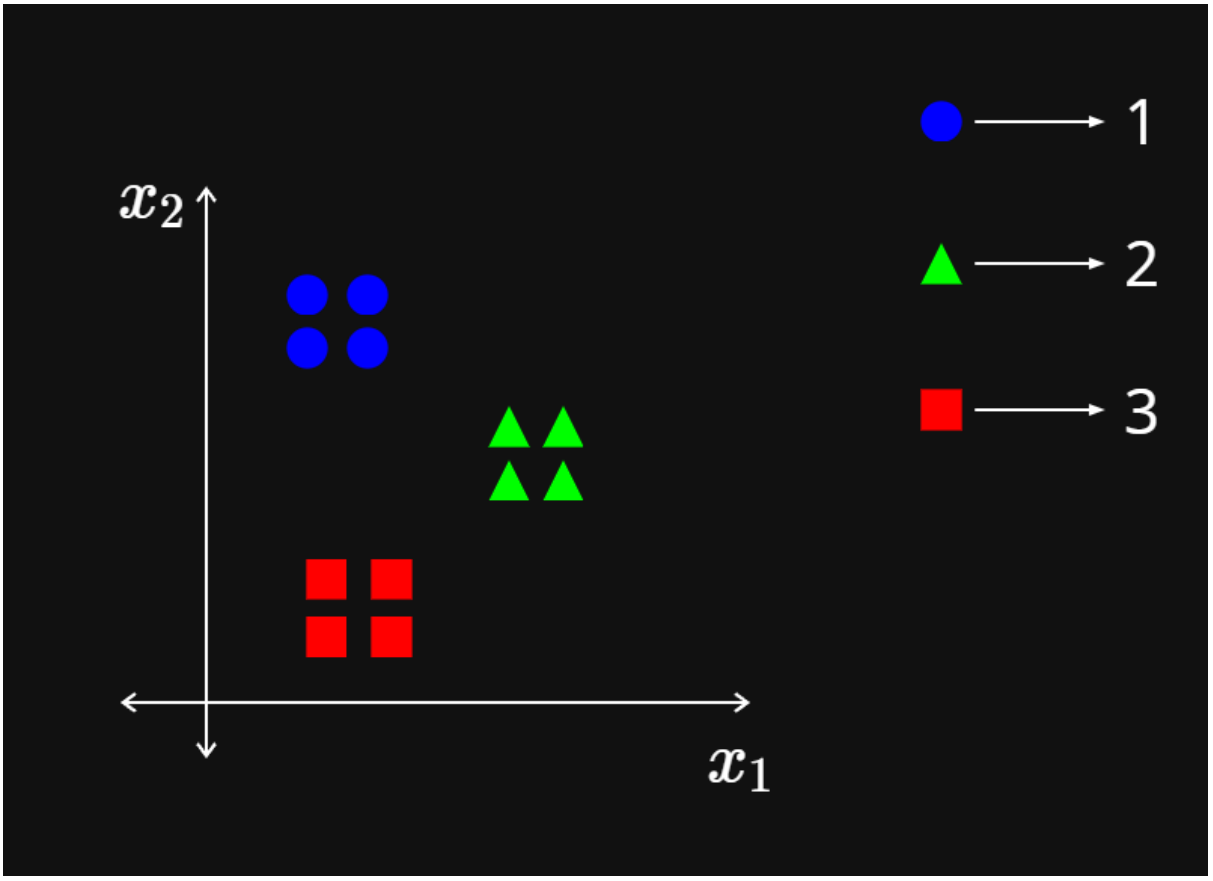
One vs One Multi class SVM:

Suppose lets say we have 3 classes labelled as 1,2,3. Now using SVM we need to classify a data point. Here the number of classes are 3 , so we will be training three different SVMs. The three different SVMs are 1 vs 2, 1 vs 3 and 2 vs 3. To understand that in more detail we will take one example 1 vs 2. In this particular data point is classified to either of the points 1 or 2 and at the end we will performing a majority voting to take more accurate class for that data point

One vs Other Multi class SVM:

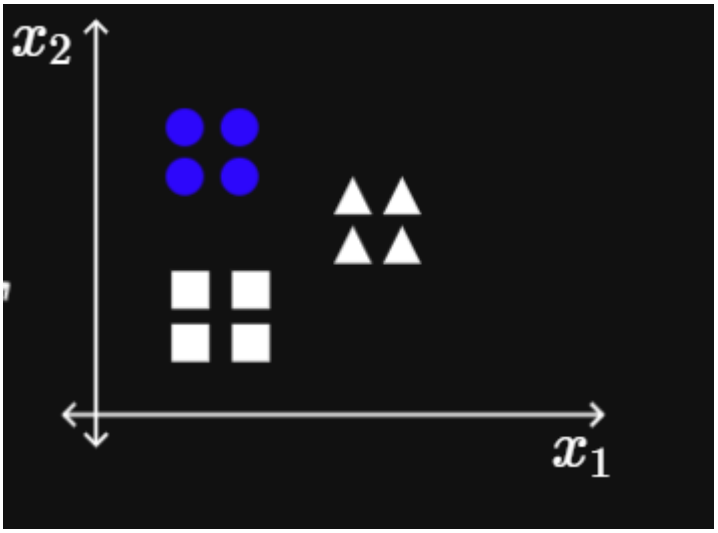
In One vs other multi class SVM the number of SVMs required would be equal to the number of classes. Suppose lets say we have 3 classes labelled as 1,2,3. Here we will train 3 SVMs such those SVMs tell whether that particular data point belong to that particular class or does it belong to other class.

Lets see One vs Other Multiclass SVM more intuitively:



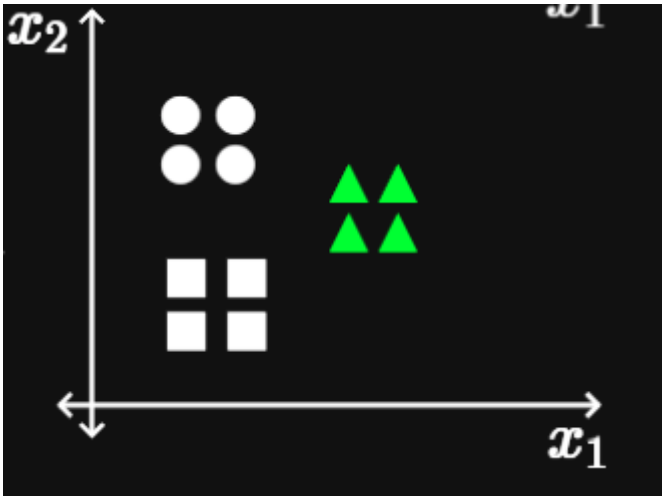
Here in the above picture we have three classes 1,2,3 which are respectively circle, triangle, square.

1st SVM Model:



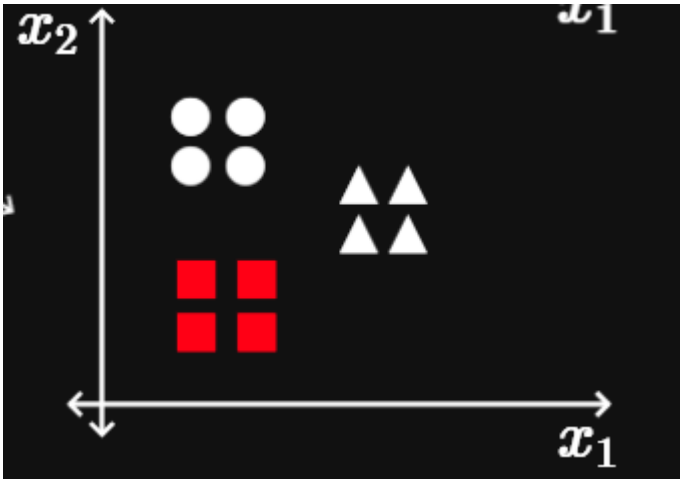
In this model we will train SVM such that it classifies whether the given data point belongs to either class 1 or it tells that data point doesnt belong to class 1

2nd SVM Model:



In this model we will train SVM such that it classifies whether the given data point belongs to either class 2 or it tells that data point doesnt belong to class 2

3rd SVM Model:



In this model we will train SVM such that it classifies whether the given data point belongs to either class 3 or it tells that data point doesnt belong to class 3

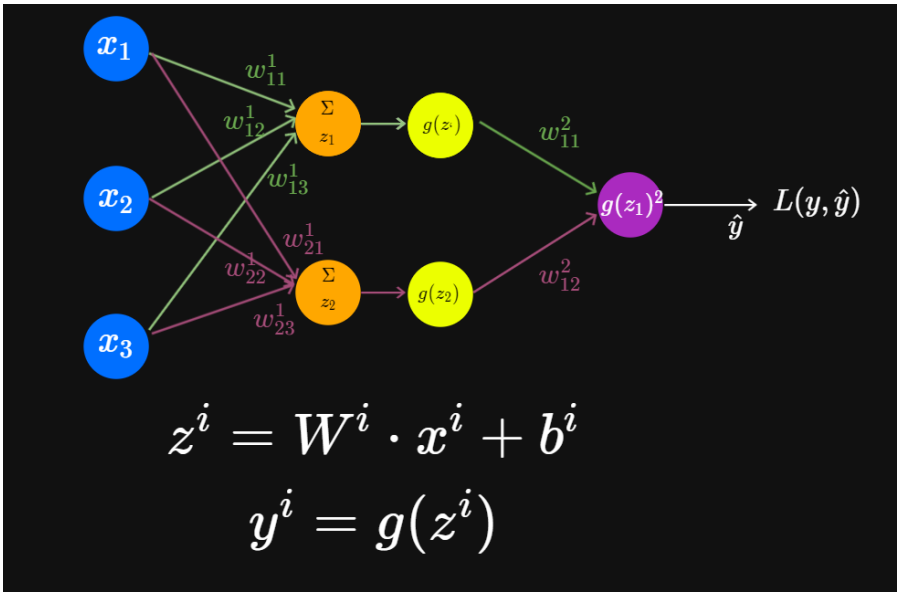
Whenever there is a new data point we will pass that data point into three SVMs and classify which label that the given data point belong too.

Neural Networks

Neural Networks are built from building blocks known as sigmoid neurons. The sigmoid neuron is “activated” according to the inputs. The Input is aggregated with weights and passed onto a activation function. Popular activation function is the sigmoid function and the corresponding neuron is called a sigmoid neuron.

Irrespective of the activation function, what is required is some sort of nonlinearity that has to be applied to aggregated inputs so as to the entire structure can act as an universal approximator.

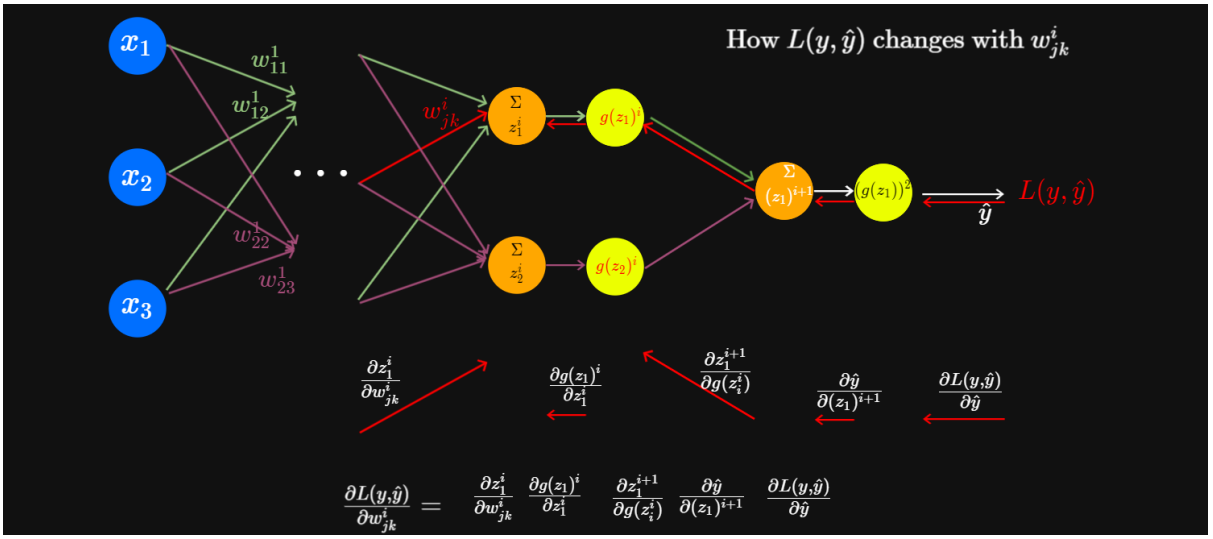
Forward Pass



The Forward Pass includes Aggregation and application of nonlinearity for each node. Final output is obtained from the output layer. The predicted output can be passed onto a loss function to update the weights in the backward pass

Backward Pass/ BackPropogation:

The Backpropogation algorithm uses chain rule to update the weights. We find how the loss function changes with respect to the weights. This can be done by tracing a path from loss function to the appropriate weights.



Once this quantity is obtained, We can try optimisation using optimisation methods such as gradient descent.

For L layers of n_i , neurons each, k neurons in Output layer ,

Algorithm 1: Forward Propogation

initialize weights and biases ;
for $k \leftarrow 1$ **to** $L-1$ **do**
  $a_k \leftarrow b_k + W_k \cdot h_{k-1}$;
  $h_k \leftarrow g(a_k)$
end
$a_L \leftarrow b_L + W_L \cdot h_{L-1}$;
$\hat{y} = O(a_L)$;

Algorithm 2: Back Propagation

for $k \leftarrow L$ **to** 1 **do**
  $\nabla_{w_k} L(\theta) \leftarrow \nabla_{a_k} L(\theta)(h_{k-1})^T$;
  $\nabla_{b_k} L(\theta) \leftarrow \nabla_{a_k} L(\theta)$;
  $W_k \leftarrow W_k - \eta \cdot \nabla_{w_k} L(\theta)$;
  // Gradient w.r.t layer below
  $\nabla_{h_{k-1}} L(\theta) \leftarrow (W_k)^T \cdot \nabla_{a_k} L(\theta)$;
  $\nabla_{a_{k-1}} L(\theta) \leftarrow \nabla_{h_{k-1}} L(\theta) \odot [\cdots g'(a_{ij}) \cdots]$;
end