Competitive Online Generalised Linear Regression with Multidimensional Outputs

Raisa Dzhamtyrova
Department of Computer Science
Royal Holloway, University of London
Egham, United Kingdom
Raisa.Dzhamtyrova@rhul.ac.uk

Yuri Kalnishkan
Department of Computer Science
Royal Holloway, University of London
Laboratory of Advanced Combinatorics and Network Applications
Moscow Institute of Physics and Technology
Egham, United Kingdom
Yuri.Kalnishkan@rhul.ac.uk

***Abstract*—We apply online prediction with expert advice to construct a universal algorithm for multi-class classification problem. Our experts are generalised linear regression models with multidimensional outputs, i.e. neural networks with multiple output nodes but no hidden nodes. We allow the final layer transfer function to be a softmax function with linear activations to all output neurons. We build an online algorithm competitive with all the experts of relevant models of this type and derive an upper bound on the cumulative loss of the algorithm. We carry out experiments on three data sets and compare cumulative losses of our algorithm and a single neuron with multiple output nodes.**

I. Introduction

We consider the online setting in which predictions and outcomes are given step-by-step. Contrary to batch mode, where the algorithm is trained on a training set and gives predictions on a test set, we learn as soon as new observations become available. For example, suppose that our aim is to predict outcomes of football matches of a new season based on the data available from a previous season. A batch algorithm builds a model on a previous season's data and this model is used to make predictions for all matches of the current season. In the online setting we add data sequentially and adjust parameters of the model after each match. More formally, we consider an online protocol where at each trial $t = 1, 2, \ldots$ a learner observes x_t and attempts to predict an outcome y_t, which is shown to the learner later. The performance of the learner is measured by means of the cumulative loss.

The main goal of this paper is to develop a universal algorithm which will be competitive with all generalised linear regression models with multidimensional outputs. For this purpose we use the method of online prediction with expert advice. At each trial we have access to predictions of experts and need to make a prediction based on experts' past performance. In statistical learning, usually some assumptions are made about the mechanism that generates the data, and guarantees are given for the method working under these assumptions. For example, one may assume a linear dependence between electricity consumption and temperature and try to fit the best parameters for linear regression. We consider the adversarial setting where no assumptions are made about the data generating process. In this paper we consider competitive prediction when one provides guarantees compared to other predictive models that are called experts. Experts could be real-human experts, complex machine learning algorithms or even classes of functions. Our goal is to develop a merging strategy that will perform not much worse than the retrospectively best expert. As a result, we do not try to build a model that works under certain assumptions but try to combine predictions that are given to us by experts. One may wonder why we don't just use predictions of only one best expert from the beginning and ignore predictions of others. First, sometimes we cannot have enough data to identify the best expert from the start. Second, good performance in the past does not necessary lead to good performance in the future.

In this paper we consider our experts to be generalised linear regression models with multidimensional outputs, which can be seen as neural networks with multiple output nodes but no hidden nodes. We allow a final layer transfer function to be a softmax function with linear activations to all output neurons. Each expert follows a particular strategy, which means that it chooses some particular parameters of the softmax function. Our goal is to develop a merging strategy that will perform not much worse than the best expert.

In this paper we consider the method of online prediction with expert advice. Online convex optimization is a similar area where a decision-maker makes a sequence of decisions from a fixed feasible set. After each point is chosen, it encounters a convex cost function. In [4] a logarithmic regret bound is obtained for α-convex cost functions. However, a second derivative of our cost function is not lower bounded, therefore this analysis is not applicable here. A similar problem was considered in [6], where the authors proposed a general additive algorithm based on gradient descent and derived loss bounds that compared the loss of the resulting online algorithm to the best offline predictor from the relevant model class. They considered a softmax transfer function (Example 4 in [6]) and achieved a theoretical bound with a multiplicative coefficient of two in front of the loss of the best expert, whereas we achieved a multiplicative coefficient of one, which indicates that our theoretical bound is better for large losses.

We will consider an approach based on the Aggregating Algorithm (AA), which was first introduced in [8]. AA works

as follows: we assign initial weights to experts and at each step the weights of experts are updated according to their current performance. The approach is similar to the Bayesian method when the prediction is the expected prediction of all models based on the likelihood of available data. AA gives a guarantee on the learner's loss to be as small as the best expert's loss plus a constant in the case of a finite number of experts. The case of square-loss multi-class classification with a finite number of experts was first introduced in [10]; the authors apply the method of prediction with expert advice for the Brier loss function in forecasting of football outcomes and show that the proposed strategy that follows AA is as good as any bookmaker. In a recent paper [1], merging algorithms were proposed for prediction of packs with tight worst case loss upper bounds similar to those for AA; empirical experiments on sports and house price datasets are carried out to study the performance of the new algorithms and to compare them against an existing method.

Even if the decision pool is infinite, in a surprisingly wide class of problems it is possible to derive good bounds for competitive online procedures. The Aggregating Algorithm for Regression (AAR) was proposed in [9] for the case of linear experts under squared loss. The case of generalised linear regression experts under log-loss was introduced in [5]; however there were no generalisations of the method for the multidimensional problem setting. The case of squared loss for generalised linear models for one-dimesional predictions was considered in [12]. A similar approach for multidimensional prediction was proposed in [11], where the authors introduced an algorithm competitive with linear functions under squared loss. One of the drawbacks of introducing linear experts for multi-class classification is that predictions of experts could lie outside the probability simplex. In all the above cases the authors achieved the theoretical bounds which were logarithmic in the number of steps. In this paper we consider our experts to be a class of softmax functions as it seems to be the most natural choice of predictors for the probability games as they output a distribution that lies inside the probability simplex. In a recent paper [3] an algorithm was constructed for the case where outcomes and predictions are distributions on a finite set, the loss function is logarithmic, and competitors are linear functions with softmax applied on top of them. The paper contains an excellent survey of application domains. We propose an algorithm for a similar setting, but improve the regret term in the upper bound on the loss. Our regret does not contain the linear term in the number of steps; asymptotically in T the regret is still of order $C \ln T$, but our multiplicative constant C is lower.

The main contribution of this paper is to provide an explicit universal algorithm for the multi-class classification problem which can compete with the best expert in terms of the cumulative logarithmic loss function. We conduct experiments to compare performance of our algorithm with neural networks with multiple output nodes, i.e. single neuron. In our experiments we check that the theoretical bound for our algorithm is not violated. Our prediction algorithm uses the Markov chain Monte Carlo (MCMC) method in a way which is similar to the algorithm introduced in [12], where AAR was applied to the generalised linear regression class of function for making a prediction in a fixed interval. MCMC is only a method for evaluating the integral and it can be replaced by a different numerical method. Theoretical convergence of the Metropolis-Hastings method in this case follows from (Theorems 1 and 3 in [7]). Estimating the convergence speed is more difficult. With the experiments provided we show that by tuning parameters online, our algorithm moves fast to the area of high values of the probability function and gives a good approximation of the predictions, and theoretical bounds are not violated.

We derive the theoretical bound on the loss of our algorithm and test its performance on three data sets. In our experiments we compare the performance of our algorithm with the model of a single neuron with multiple output nodes. Even though the main advantage of using neural networks is the possibility of using many layers with a large number of nodes to model non-linear dependencies of the target vector and input parameters, it might be possible to avoid multilayer networks by introducing many new inputs, each of which is a non-linear function of the original inputs. We compare the difference between the cumulative losses of our algorithm and the single neuron with the theoretical bound which was obtained for our algorithm.

II. Framework

A game of prediction contains three components: a space of outcomes Ω, a decision space Γ, and a loss function $\lambda : \Omega \times \Gamma \to \mathbb{R}$. We consider the problem of classification with d classes $\Sigma = \{1, \dots, d\}$. As the outcome space Ω we take d-dimensional unit vectors $e_1, \dots, e_d$, i.e. the distributions concentrated on the vertices of the simplex. The vectors $e_1, e_2, \dots, e_d$ form the standard basis in $\mathbb{R}^d$; the vector e_i has a one at the i-th position and zeros elsewhere. The decision space $\Gamma = \mathbb{P}(\Sigma) = \{(\gamma^{(1)}, \dots, \gamma^{(d)}) : \sum_{i=1}^{d} \gamma^{(i)} = 1,\ 0 \le \gamma^{(i)} \le 1\}$ is a simplex in d-dimensional space, and for any $y \in \Omega$ we define the loss

$$\lambda(y, \gamma) = -\sum_{\chi \in \Sigma} y^{(\chi)} \ln \gamma^{(\chi)},$$

where $y^{(\chi)}$ and $\gamma^{(\chi)}$ are the χ-th coordinates of the respective vectors.

Learner works according to the following protocol:

Protocol 1:

for $t = 1, 2, \dots$

nature announces $x_t \subseteq \mathbb{R}^n$

learner outputs $\gamma_t \in \Gamma \subseteq \mathbb{R}^d$

nature announces $y_t \in \Omega \subseteq \mathbb{R}^d$

learner suffers losses $\lambda(y_t, \gamma_t)$

end for

The cumulative loss of the learner is $L_T = \sum_{t=1}^{T} \lambda(y_t, \gamma_t)$.

We want to find an algorithm capable of competing in terms of cumulative loss with all experts θ that at step t output $\xi_t(\theta) = (\xi_t^1(\theta), \dots, \xi_t^d(\theta))$ such that

$$\xi_t^i(\theta) = \sigma_i(\theta, x_t),\ i = 1, \dots, d, \tag{1}$$

where $\sigma_i(\theta, x_t)$ is the softmax function

$$\sigma_i(\theta, x_t) = \frac{e^{\theta_i' x_t}}{\sum_{j=1}^{d} e^{\theta_j' x_t}},\ i = 1, \ldots, d, \tag{2}$$

and $\theta = (\theta_1', \ldots, \theta_d')' \in \mathbb{R}^{nd}$, $\theta_i = (\theta_{i,1}, \ldots, \theta_{i,n})' \in \mathbb{R}^n$.

Figure 1 shows the diagram of the single neuron with softmax transfer function. We define a single neuron as a softmax transfer function applied to a linear function of the input vector. The usual approach in statistics to obtain the best parameters of a single neuron is to use some optimization algorithm such as gradient descent. Our prediction algorithm differs from many algorithms commonly used to fit a generalized linear model. First, instead of fitting the data with the best parameter θ, it uses a regularization parameter $a > 0$ preventing θ to be too large, and thus preventing overfitting to a certain extent. Second, it does not look for the best regularized expert θ, but at each prediction step mixes all experts in a way which is similar to the Bayesian scheme, thus preventing overfitting even further.

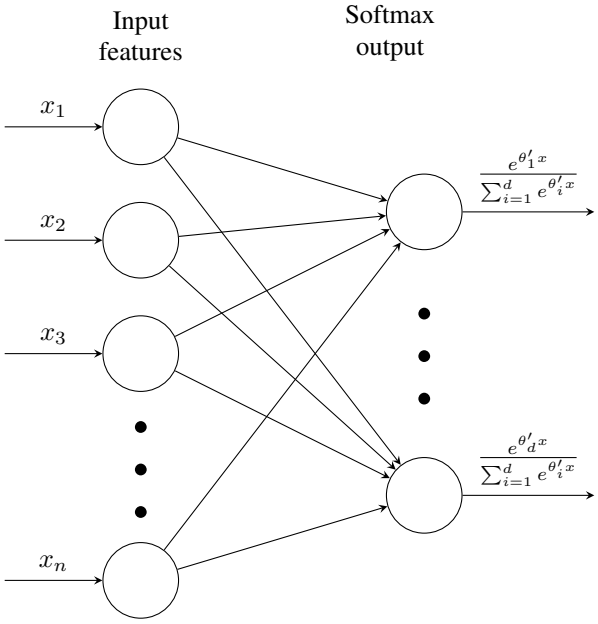

Fig. 1. Single neuron with softmax transfer function

III. Description of Aggregating Algorithm

We use the AA to derive our prediction algorithm which under logarithmic loss reduces to the Bayesian scheme. The AA works as follows. It is given a parameter η and an initial distribution on the experts. We choose the normal distribution

$$P_0(d\theta) = (a\eta/\pi)^{nd/2} \exp(-a\eta\|\theta\|^2) d\theta, \tag{3}$$

for some $a > 0$. After each step t the AA updates the experts' weights according to their losses:

$$P_t(d\theta) = e^{-\eta\lambda(y_t, \xi_t(\theta))} P_{t-1}(d\theta). \tag{4}$$

The weights of experts which suffer larger losses at some step will have smaller importances for making further predictions.

First, we introduce the Aggregating Pseudo–Algorithm (APA) which at step t outputs *generalised prediction*:

$$g_t(y) = -\frac{1}{\eta} \ln \int_\Theta e^{-\eta\lambda(y, \xi_t(\theta))} P_{t-1}^*(d\theta), \tag{5}$$

where $P_{t-1}^*(d\theta)$ are normalized weights:

$$P_{t-1}^*(d\theta) = \frac{P_{t-1}(d\theta)}{P_{t-1}(\Theta)},$$

where Θ is a *parameter space*, i.e. experts can output prediction $\theta \in \Theta$.

The generalised prediction can be seen as a weighted average of the experts' predictions in a way which is similar to the Bayesian method.

The AA is obtained from the APA by replacing each generalised prediction g_t by a *permitted prediction* $\Sigma(g_t)$, where the *substitution function* Σ maps every generalised prediction $g : \Omega \to [0, \infty]$ into a permitted prediction $\Sigma(g) \in \Gamma$ satisfying

$$\forall y : \lambda(y, \Sigma(g)) \le g(y). \tag{6}$$

Let us define $P(\Theta)$ as the set of all probability measures over Θ. If a substitution function satisfying (6) for any distribution $P_{t-1}^*(d\theta) \in P(\Theta)$ exists, we say that the loss function is *η-mixable*. The loss function is *mixable* if it is η-mixable for some $\eta > 0$. The game is called *mixable* if its loss function is mixable in the setting of the game.

The loss of the AA over the first T trials does not exceed the loss of the sum of the generalized predictions (5) over the first T trials:

$$L_T(AA) \le \log_\beta \int_\Theta e^{-\eta L_T(\theta)} P_0(d\theta). \tag{7}$$

IV. Aggregating Algorithm for the logarithmic loss function

Now we will show that the Aggregating Algorithm for the logarithmic loss function with $\eta = 1$ is the same as the Bayesian scheme.

The weights update (4) becomes

$$P_t(d\theta) = \xi_t^{\omega_t}(\theta) P_{t-1}(d\theta) = P_0(d\theta) \prod_{i=1}^{t} \xi_i^{\omega_i}(\theta). \tag{8}$$

Therefore, the normalized weights: $P_t^*(d\theta) = \frac{P_t(d\theta)}{\int_\Theta P_t(d\theta)}$ are identical to the posterior distribution of θ after observing $\omega_1, \omega_2, \ldots, \omega_t$.

The generalized prediction (5) becomes:

$$g_t(\omega) = -\frac{1}{\eta} \ln \int \xi_t^\omega(\theta) P_{t-1}^*(d\theta) \tag{9}$$

and thus represents the loss of the Bayesian mixture.

The log-loss game is mixable for $\eta \le 1$ and the substitution function $\Sigma : \mathbb{R}^\Omega \to \Gamma$ is simply $e^{-(\cdot)}$. The function x^η is concave for $\eta < 1$, and thus

$$\int_\Theta (\xi_t^\omega(\theta))^\eta Q(\theta) \le \Big(\int_\Theta \xi_t^\omega(\theta) Q(\theta) \Big)^\eta$$

for any ω and $Q \in P(\Theta)$.

After taking negative logarithms of both parts and multiplying by $\frac{1}{\eta}$, we obtain

$$-\frac{1}{\eta}\ln\Big(\int_\Theta \xi_t^\omega(\theta)Q(\theta)\Big)^\eta \le -\frac{1}{\eta}\ln\int_\Theta(\xi_t^\omega(\theta))^\eta Q(\theta).$$

In other words, the loss of the prediction corresponding to $\eta = 1$ is less than the generalized prediction calculated with any other $\eta < 1$.

V. Derivation of the Prediction Algorithm

Let $L_T^\theta = \sum_{t=1}^T \lambda(y_t, \xi_t(\theta))$ be the cumulative loss of predictor θ over T trials. To predict at step T our prediction algorithm works with the function

$$L_T^\theta + a\|\theta\|^2 = -\sum_{t=1}^{T-1}\sum_{i=1}^{d} y_t^i \ln \xi_t^i(\theta) + a\|\theta\|^2, \tag{10}$$

where $a > 0$ is the regularization parameter preventing θ to be too large, $\theta \in \Theta = \mathbb{R}^{nd}$, $y_t = (y_t^1, \dots, y_t^d)' \in \Omega$. At each step the algorithm does not look for the best regularized expert θ, but instead mixes all the experts in a way similar to the Bayesian scheme.

The weights update according to (4) is:

$$P_t(d\theta) = e^{-\eta\lambda(y_t,\xi_t(\theta))}P_{t-1}(d\theta) = e^{-\eta L_T^\theta}P_0(d\theta).$$

After putting the initial distribution of experts (3), the unnormalized weight of an expert θ at step t can be expressed as

$$P_t(d\theta) = (a\eta/\pi)^{nd/2}e^{-\eta(L_t^\theta + a\|\theta\|^2)}d\theta. \tag{11}$$

We obtain the generalised prediction (9) from unnormalised weights (11) and, omitting the factor $(a\eta/\pi)^{nd/2}$, we have:

$$G_t(\omega) = -\frac{1}{\eta}\ln\int_\Theta \xi_t^\omega(\theta)e^{-\eta(L_{t-1}^\theta + a\|\theta\|^2)}d\theta. \tag{12}$$

Normalization is avoided because calculating the normalizing constant is a computationally inefficient operation.

After putting the maximum value for $\eta = 1$ and applying the substitution function $e^{-(\cdot)}$, our predictions become:

$$\gamma_t(\omega) = \int_\Theta \xi_t^\omega(\theta)w_{t-1}^*(\theta)d\theta, \tag{13}$$

where

$$w_T^*(\theta) = Cw_T(\theta) = C\exp\Big(-\eta a\|\theta\|^2 + \eta\sum_{t=1}^{T}\sum_{i=1}^{d} y_t^i\ln\xi_t^i(\theta)\Big) \tag{14}$$

is the normalized posterior distribution of parameters θ, $w_T(\theta)$ denotes the unnormalized posterior distribution, and C is the normalising constant ensuring that $\int_\Theta w_T^*(\theta)d\theta = 1$.

We need to integrate $\xi_t^\omega(\theta)$ with respect to the posterior distribution $w_{t-1}^*(\theta)$. Our algorithm uses the MCMC technique to estimate the numerical integration of (13). The good introduction to MCMC for Machine Learning is in [2].

We will use Metropolis-Hastings algorithm for sampling parameters θ from the posterior distribution $\mathcal{P}$. As a proposal distribution we choose the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ with some chosen parameter σ. We start with some initial parameter θ_0 and at each step m we update:

$$\theta^m = \theta^{m-1} + \mathcal{N}(0, \sigma^2),\ m = 1, \dots, M,$$

where M is a maximum number of iterations in MCMC method.

The update parameter θ^m at step m is accepted with probability $\min\Big(1, \frac{f_\mathcal{P}(\theta^m)}{f_\mathcal{P}(\theta^{m-1})}\Big)$, where $f_\mathcal{P}(\theta)$ is the density function for the distribution $\mathcal{P}$ at point θ. At each step, by accepting and rejecting the updates of parameters θ, we move closer to the maximum of the density function. At the beginning it is common to use a 'burn-in' stage where the integral is not calculated till we reach the area of high values of the density function $f_\mathcal{P}$. Thus, we perform integration only from the area with high density of $\mathcal{P}$. Because we are interested only in the ratio of density functions of generated parameters, we can generate new parameters θ from the unnormalized posterior distribution $w_T(\theta)$ and avoid the weights normalization at each step, which is more computationally efficient.

At time $t = 0$ the algorithm starts with an initial estimate of the parameters $\theta_0 = 0$. At each iteration $t > 0$ we start with the parameter θ^M calculated at the previous step $t-1$. It allows the algorithm to converge faster to the correct location of the main mass of the distribution.

Algorithm

Parameters: number $M > 0$ of MCMC iterations,
standard deviation $\sigma > 0$,
regularization coefficient $a > 0$

$\eta := 1$
initialize $\theta_0^M := 0 \in \Theta$
define $w_0(\theta) := \exp(-a\eta\|\theta\|^2)$
for $t = 1, 2, \dots$ do
  $\gamma_t^i := 0,\ i = 1, \dots, d$
  read $x_t \in \mathbb{R}^n$
  initialize $\theta_t^0 = \theta_{t-1}^M$
  for $m = 1, 2, \dots, M$ do
    $\theta^* := \theta_t^{m-1} + \mathcal{N}(0, \sigma^2 I)$
    flip a coin with success probability
      $\min\Big(1, w_{t-1}(\theta^*)/w_{t-1}(\theta_t^{m-1})\Big)$
    if success then
      $\theta_t^m := \theta^*$
    else
      $\theta_t^m := \theta_{t-1}^m$
    end if
    $\gamma_t^i := \gamma_t^i + \eta\xi_t^i(\theta_t^m),\ i = 1, \dots, d$
  end for
  output predictions $\gamma_t^i = \gamma_t^i/M,\ i = 1, \dots, d$
end for

VI. Theoretical bounds

Theorem 1: Let $a > 0$. There exists a prediction strategy for Learner such that for every positive integer T, for every

sequence of outcomes of the length T, and for every $\theta \in \mathbb{R}^{nd}$, the cumulative loss L_T of Learner satisfies

$$L_T \leq L_T^\theta + a\|\theta\|^2 + \frac{d}{2}\ln\det\left(I + \frac{d}{8a}\sum_{t=1}^{T} x_t x_t'\right). \quad (15)$$

If, in addition, $\|x_t\|_\infty \leq B$ for all t, then

$$L_T \leq L_T^\theta + a\|\theta\|^2 + \frac{nd}{2}\ln\left(1 + \frac{d}{8a}B^2T\right). \quad (16)$$

VII. Experiments

In this section we apply our algorithm to three data sets and compare its performance with the of model of the single neuron. We obtained the best parameters of model of the single neuron by using gradient descent.

A. Synthetic Data Set

We generated the synthetic 'Smiley' data set that consists of two Gaussian eyes, a trapezoid nose and a parabola mouth. The function for generating this data set was taken from the R library 'mlbench'. Figure 2 shows the data set, which contains 1000 observations with two features and four classes. We divide our data in such a way that each class will have half of its observations in training and in test data sets. Figure 3 illustrates the generated training data set.

First, we will run our algorithm and train the model of a single neuron on training data set and compare their performance. We run our algorithm, with $M = 3000$ MCMC iterations and a 'burn-in' period of $M_0 = 1000$, for different values of the regularization parameter a and standard deviation σ.

Table I shows the total loss of our algorithm on the training data set. Low values of losses were achieved with small regularization parameters a and large standard deviations σ. Very small values of σ lead to big losses, as the algorithm is not able to reach the area of high values of the density function $f_{\mathcal{P}}$. The total loss of the single neuron on the training data set is 1.4, which is comparable with losses of our algorithm achieved with small values of a and large values of σ.

Table II illustrates the acceptance ratio of new sampling parameters of our algorithm. Large values of σ and large values of regularization parameter a result in low acceptance ratios. With large values of σ we move faster to the area of high values of the density function, while smaller values of σ can lead to more expensive computations as our algorithm would require more iterations to find the optimal parameters. Figure 4 illustrates the logarithm of parameters likelihood $w(\theta)$ defined in (14) for $\sigma = 0.1$ and 0.7. We can see from the graphs that for $\sigma = 0.7$ the algorithm reaches its maximum value of log-likelihood quite fast while for $\sigma = 0.1$ it still tries to find the maximum value after 3000 iterations. It is important to keep track of the acceptance ratio of the algorithm, as a high acceptance ratio means that we move too slowly and need more iterations and a larger 'burn-in' period to find the optimal parameters.

Now we want to demonstrate the 'power' of online learning compared to batch learning. We train the model of a single neuron on the training data set and compare its performance with our algorithm applied to test data set. We choose parameters of the algorithm to be $M = 3000$, 'burn-in' period $M_0 = 1000$, regularization parameter $a = 0.01$ and standard deviation $\sigma = 0.7$. Note, that even though we use prior knowledge about optimal parameters of our algorithm using results on training data set, we do not actually train our algorithm, and start with initial value $\theta_0 = 0$. Figure 5 shows the difference between cumulative losses of a single neuron and our algorithm $L_T^{\theta^*} - L_T$ on test data set, where θ^* was obtained by a single neuron model on the training data set. We can see from the graph that our algorithm needs a little time to train and after around 100 steps it becomes better than single neuron trained on the training data set. It is obvious from Figure 3 that there is an infinite number of linear classifiers that could classify the data correctly as the training data set contains linearly separable classes. The training data set does not describe the 'underlying nature' of the generated data. As a result, the retrespectively best model that was trained on training data does not perform well on test data.

Now we will train the single neuron on the test data set to find retrospectively the best model with parameters θ^*. Figure 6 shows the difference between cumulative losses of the retrospectively best expert θ^* on test data and the cumulative loss of our algorithm. We also plot the theoretical bound for our algorithm. The initial large gap corresponds to the value $-a\|\theta^*\|^2$, which gives the initial start to Learner on expert θ^*. As time increases, we add an additional value $-\frac{nd}{2}\ln(1+\frac{d}{8a}B^2T)$ to the bound. We can see from the graph that initially the loss difference is decreasing fast which means that the loss of our algorithm is becoming larger compared to the loss of the single neuron. The initial start $-a\|\theta^*\|^2$ gives us some time for training. After the initial training time passes, the difference between cumulative losses becomes smoother and behaves in a similar way to the theoretical bound of our algorithm which is decreasing logarithmically in the number of steps.

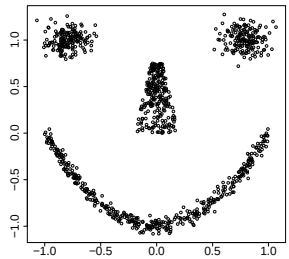

Fig. 2. Smiley data set

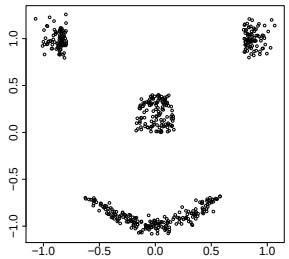

Fig. 3. Training data set

B. Glass Identification Data Set

We conduct similar experiments on the Glass Identification data set which is the part of the library 'mlbench' in R or which can be downloaded from UCI Machine Learning

TABLE I
TOTAL LOSSES OF OUR ALGORITHM ON TRAINING SET

$a \setminus \sigma$	0.1	0.3	0.5	0.7	0.9	1
0.01	175.3	5.2	1.9	2.5	1.8	1.8
0.05	160.1	6.6	5.9	6.6	6.3	7.0
0.1	177.3	11.3	10.1	11.1	10.8	11.8
0.5	160.0	35.4	36.1	32.7	37.2	33.7
0.7	168.0	43.4	45.8	42.1	48.3	36.0
1	166.3	59.2	61.8	58.6	48.1	59.8

TABLE II
ACCEPTANCE RATIO OF OUR ALGORITHM ON TRAINING SET

$a \setminus \sigma$	0.1	0.3	0.5	0.7	0.9	1
0.01	0.79	0.81	0.74	0.56	0.40	0.32
0.05	0.82	0.78	0.61	0.32	0.16	0.08
0.1	0.80	0.77	0.48	0.22	0.09	0.04
0.5	0.81	0.63	0.23	0.06	0.02	0.01
0.7	0.82	0.61	0.18	0.03	0.01	0.01
1	0.81	0.57	0.14	0.03	0.01	0.01

Repository. The goal is to classify the six types of glasses. The study of classification of types of glass was motivated by criminological investigation. At the scene of a crime, the glass left can be used as evidence. The data set contains nine features and a total of 214 observations. As there were no timestamps in the data sets, observations were randomly shuffled, and this order was used as the time. We normalise all features between -1 and 1 and add an addition bias of 1 to all observations.

Similar to the previous experiment, we find retrospectively the best single neuron with parameters θ^* using the whole data set. We want to compare the performance of the retrospectively best expert θ^* with the performance of our algorithm. We run our algorithm several times on the data set to choose parameters for our algorithm. Table III shows the acceptance ratio of our algorithm for different regularization parameters a and standard deviation σ. We can see from the table that for $\sigma = 0.3$ we have a reasonable acceptance ratio of our algorithm.

Now we will show how the performance of our algorithm and the behavior of the loss bound depend on the different parameters of regularization a. We choose number of steps $M = 3000$, 'burn-in' period $M_0 = 1000$ and $\sigma = 0.3$. First, we run our algorithm for small regularization $a = 0.01$. Figure 7 shows the difference between cumulative losses of the single neuron and our algorithm. Small values of regularization give a small start $-a\|\theta^*\|^2$ on the initial parameters at time $t = 0$. However, the theoretical bound will grow faster with time $-\frac{nd}{2}\ln(1+\frac{d}{8a}B^2T)$ as it is inversely proportional to the logarithm of the regularization parameter a.

We will conduct the second experiment for a large regularization $a = 1$. Figure 8 shows the difference between cumulative losses of the single neuron and our algorithm. We can see from the graph that for larger regularization values our algorithm performs better as the difference in cumulative losses is smaller. For this larger regularization we allow a larger initial start $-a\|\theta^*\|^2$ on the parameters. Hovewer, the theoretical bound decreases slower with time compared to the previous experiment.

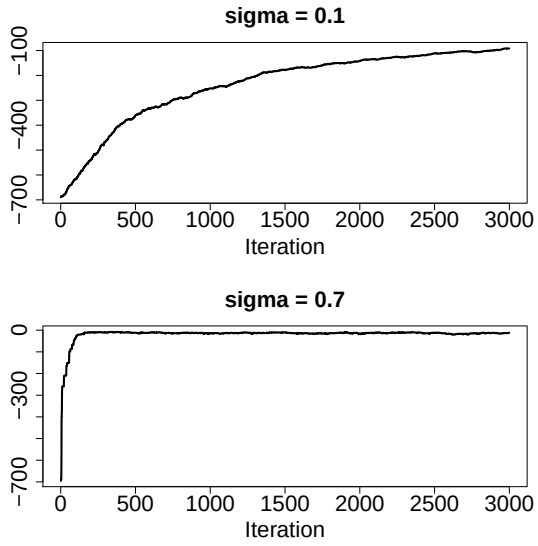

Fig. 4. Log-likelihood of parameters depending on iteration step

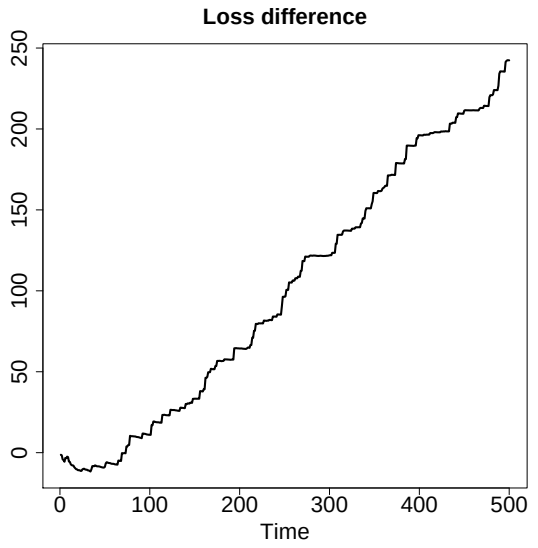

Fig. 5. Comparison with single neuron trained on training set

The choice of the regularization parameter a is important as it affects the behaviour of the theoretical bound of our algorithm. Larger parameters of regularization gives larger start on the parameters of the best model, however the theoretical bound will have a smaller growth rate as time increases.

C. Football Data Set

The third data set was compiled from historical information on football matches and bookmakers' odds[1]. The data set covers three seasons, 2014/2015, 2015/2016 and 2016/2017 of the English Premier League and a total of 1140 matches.

[1]Available at http://Football-Data.co.uk

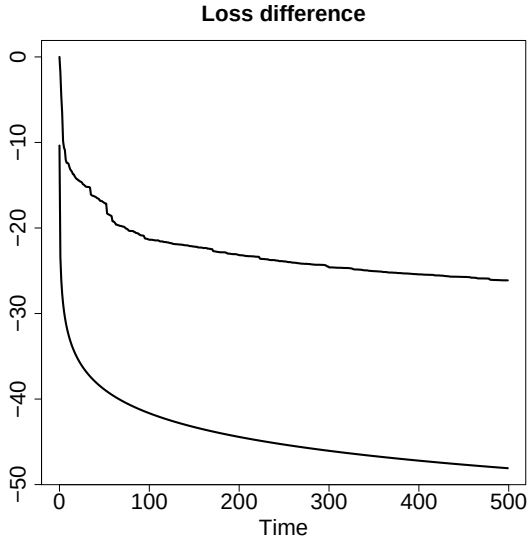

Fig. 6. Comparison with retrospectively best single neuron

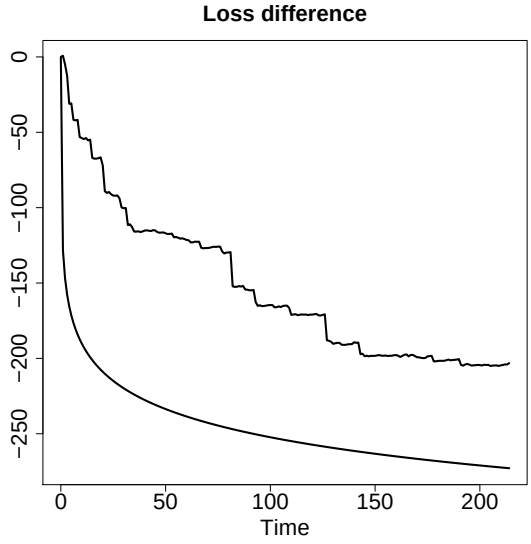

Fig. 7. Glass data set, $a = 0.01$

TABLE III
ACCEPTANCE RATIO OF OUR ALGORITHM

$a \setminus \sigma$	0.05	0.1	0.2	0.3	0.4	0.5
0.01	0.92	0.86	0.67	0.39	0.16	0.06
0.05	0.92	0.85	0.65	0.36	0.16	0.06
0.1	0.92	0.86	0.64	0.34	0.14	0.04
0.5	0.92	0.84	0.64	0.32	0.11	0.02
0.7	0.92	0.86	0.62	0.29	0.09	0.02
1	0.91	0.85	0.62	0.26	0.07	0.02

Each match can have three outcomes: 'home win', 'draw', or 'away win'. The data contains historical information such as total number of goals, shots, corners, yellow and red cards after half-time and full-time and bookmakers' odds from different providers. For each team we generated features such as average number of games won / lost, average number of goals scored / conceded, average number of shots during the first-half, etc. In addition, we combined the odds of different bookmakers provided for the current match. There were a total of 46 generated features. The first two seasons were used for the training of single neuron and the last season was left for test. We wanted to check if our algorithm could perform close to the model of the single neuron by starting to learn online only on the current season. We chose the parameters of our algorithm $M = 2000$, 'burn-in' period $M_0 = 500$, regularization parameter $a = 0.05$ and standard deviation $\sigma = 0.2$. Figure 9 shows the difference between cumulative losses of the single neuron trained on the first two seasons and our algorithm that starts to train only on the current season. Initially the loss difference is decreasing as our algorithm is learning, and after around 100 steps the loss difference starts to increase which means that our algorithm starts to predict better than the single neuron. After around 200 steps the loss difference becomes positive which indicates that we compensate the loss suffered during the initial training.

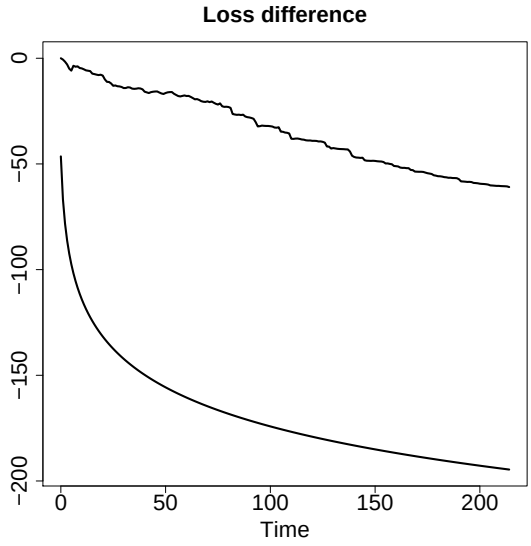

Fig. 8. Glass data set, $a = 1$

Now we want to compare the performance of our algorithm with the model of the single neuron that will be trained in online mode. At the initial step single neuron uses the parameters of the model that was trained on the first two seasons. After that, we add data sequentially and re-train the model after each match. Figure 10 illustrates the difference between cumulative losses of the single neuron trained online and our algorithm. Initially our algorithm performes much worse than the single neuron in online mode as the difference of cumulative losses decreases fast. However, after around 200 steps the difference of cumulative losses stabilizes and becomes more 'flattened' which indicates that the performance of our algorithm becomes close to the performance of the single neuron.

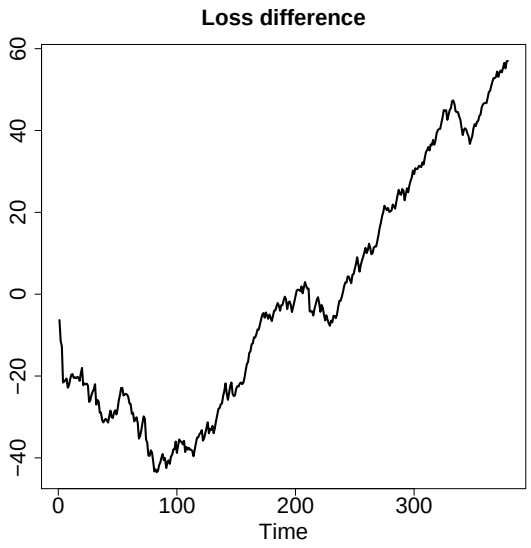

Fig. 9. Comparison with single neuron trained in batch mode

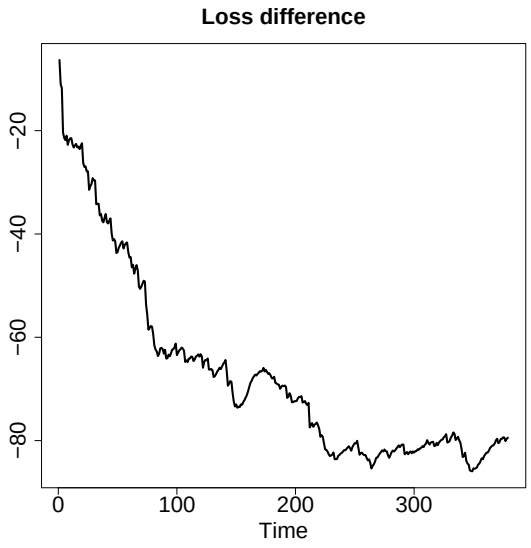

Fig. 10. Comparison with single neuron trained in online mode

VIII. Conclusions

We proposed an online algorithm for the multi-class classification problem which is capable of competing with all neural networks with multiple output nodes but no hidden nodes with softmax transfer function. The main advantage of our algorithm is the upper bound on the cumulative loss.

We carry out the experiments on three data sets to evaluate the performance of our algorithm. Results show that our algorithm can outperform retrospectively the best single neuron trained on the training data set. When comparing with the single neuron trained in online mode, our algorithm requires some time for training but then its performance becomes close to the performance of the single neuron. We also compare the difference between the cumulative losses of retrospectively the best single neuron trained using the whole data set and our algorithm, and we check that the theoretical bound of our algorithm is not violated.

One of the disadvantages of our algorithm is that it might perform much worse with non-optimal input parameters of regularization a and standard deviation σ. If no prior knowledge is available, one can start with some reasonable values of input parameters and keep track of the acceptance ratio of newly generated θ. If the acceptance ratio is too high it might indicate that the algorithm moves too slowly to the area of high values of the probability function of θ, and thus that the standard deviation σ should be increased. Another option is to take a very large number of steps and larger 'burn-in' period. The choice of the regularization parameter a is important as it affects the behaviour of the theoretical bound of our algorithm. Larger parameters of regularization give a larger start on the parameters θ^* of the best model, however the theoretical bound will have a smaller growth rate as time increases. The choice of the regularization parameter depends on the particular task and goals that are desired to be achieved. Another disadvantage of our algorithm against the competitors is in its training speed. Increasing the training speed of our algorithm is an interesting area of future research.

References

[1] D. Adamskiy, A. Bellotti, R. Dzhamtyrova, and Y. Kalnishkan. Aggregating algorithm for prediction of packs. *To appear in* Machine Learning, 2019. https://doi.org/10.1007/s10994-018-5769-2.

[2] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning Journal, 50*.

[3] D. G. Foster, S. Kale, H. Luo, M. Mohri, and K. Sridharan. Logistic regression: The importance of being improper. *Proceedings of Machine Learning Research vol, 75*, pages 75: 1–42, 2018.

[4] E. Hazan. *Introduction to Online Convex Optimization*. nowpublishers.com, 2016.

[5] S. M. Kakade and A. Y. Ng. Online bounds for bayesian algorithms. *Advances in Neural Information Processing Systems 17*, pages 641–648, 2005.

[6] J. Kivinen and M. Warmuth. Relative loss bounds for multidimensional regression problems. *Advances in Neural Information Processing Systems 17*, page 301–329, 2001.

[7] G. O. Roberts and A. F. M. Smith. *Simple conditions for the convergence of the Gibbs sample and Metropolis-Hastings algorithms*. Stoch. Processes Appl., 49, 1993.

[8] V. Vovk. Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 371–383, San Mateo, CA, 1990. Morgan Kaufmann.

[9] V. Vovk. Competitive on-line statistics. *International Statistical Review*, 69(2):213–248, 2001.

[10] V. Vovk and F. Zhdanov. Prediction with expert advice for the Brier game. *Journal of Machine Learning Research*, 10:2445–2471, 2009.

[11] F. Zhdanov and Y. Kalnishkan. Linear probability forecasting. In *Artificial Intelligence Applications and Innovations, AIAI-2010, Proceedings*, volume 339 of *IFIP Advances in ICT*, pages 4–11. Springer, 2010.

[12] F. Zhdanov and V. Vovk. Competitive online generalized linear regression under square loss. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 531–546, 2010.