ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Miniproject

Pattern Recognition and Machine Learning

Michaël Hobbs- Naïg Chenais

May 2013

TA : Vassilis Kalofolias

I. Introduction

The aim of this project is to solve two classification tasks thanks to multiple layer perceptrons. A dataset composed of two simultaneous camera recordings is available to us, and we want to build a generalizable model able to recognize the objects that were presented to the camera. In a first step, we will implement three-layer perceptron for binary classification task, and in a second step, extend it so that it can classify inputs in five different classes.

II. Methods

A. Preprocessing

Left and Right camera recordings are separately preprocessed thanks to preprocess_data function. Mean input to remove and standard deviation are calculated from training data, then all data - including validation and testing ones- are whitened with these parameters. With this process, validation error might be more representative of future testing error, but unlike it, validation error evaluation is included in update loops. Thus, classifier generalization ability should be improved. This use of validation dataset is also a way to simulate real life constraints: we don't always have test data available and we need to train on naïve subset of initial data set.

B. Mlp implementation

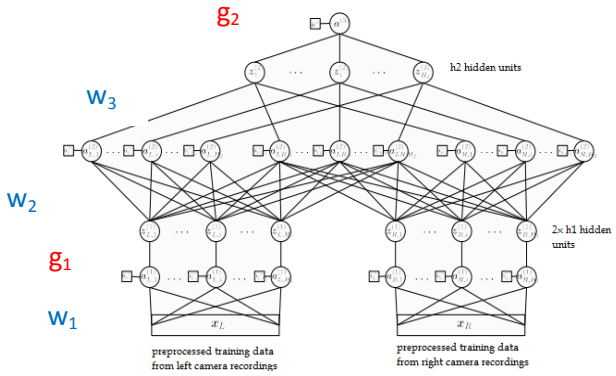


Fig 1 Perceptron architecture

Once activation values and weights are initialized, data are send in small batches through forward pass, that updates units values and returns the activation ones. Returned activations from forward pass are taken as input by backpropagation function, that computes residues and update weights-bias pairs from third layer to first one, minimizing logistic error with a stochastic gradient descent. (Calculations used to backpropagation algorithm implementations and variable notations can be found in the annex section. Gradient testing functions has been used to validate coherence between our calculations and finite difference results.)
Training data are sent to MLP in mini-batches, each data permutation is randomly chosen at the beginning of each epoch. Minimal number of epoch is set to 15 but criterion of stop is given by the validation error behavior : at the end of each epoch, validation logistic error is computed, and if it has increased respect to the previous epoch, we stop training.
To convert MLP architecture into multi-classes classifier, we extend the third layer outputs, residuals and weights vectors size, so that we obtain a final 5 by 1 t-output vector whose value 1 corresponds to class attribution achieved by the MLP. Multi-class MLP gradients and residual were computed from squared error, while binary MLP ones were computed from logistic error expression.

C. Parameter choices

1. Linear Regression

We used linear regression with Tikhonov regularized least squares. In this method, optimal weights are those of model that will provide the smallest risk value. To choose our model, we explored several Tikhonov parameters ϑ and computed risk value over 10-fold cross validation trials. Best ϑ was chosen so that it provides lower value for validation error taking into account test risk : we pick the largest ϑ such that :

$E_{val(\vartheta)} < E_{val(\vartheta)} + std\ (risk(\vartheta))$

This parameter determines in which extend will large weights be penalized in the error function and how important their influence on gradient direction has to be, since Tikhonov regularized squared error can be written as :

$E_{reg(w)} = E_{squared(w)} + \vartheta \sum_{i=1}^{N} ||w||_i^2$

We first explore wide range of values taken from 2^n series, and found that risk was minimal for ϑ in [16,64]. Fine-tuning the granularity of explored values, we ended to conclude that optimal value for ϑ was 40.

Note that we merged left and right data sets prior to computation, so that they were all treated together.

2. Logistic Regression

Using this second linear classifier with another error function , the main technical difference compared to the previous method is that we had to iterate gradient descent to minimize error, since there is no analytical solution for the gradient of the logistic error function.

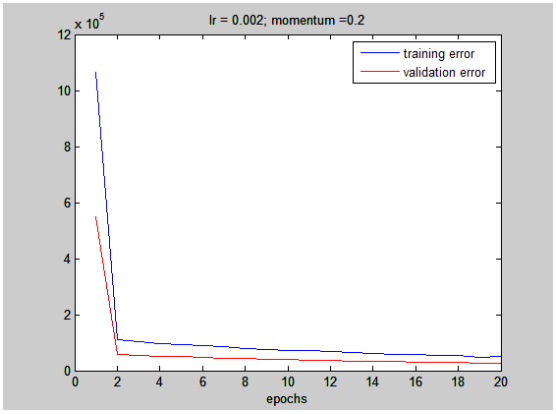
In order to optimize parameters involved in gradient descent minimizing the logistic error, i.e learning rate and momentum term, we evaluate training and validation errors depending on these two parameters.

Indeed, logistic error $E_{log(w)} = \sum_{i=1}^{N} (\log \sum_{i=1}^{N} \exp(w_i . x_i)) - t_i . w_i . x_i$ provides the following gradient and weights update :

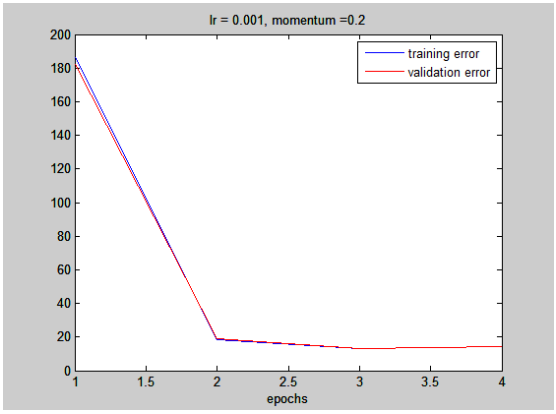
$$\Delta_w E = \frac{\exp(w_i . x_i)}{\sum_{i=1}^{N} \exp(w_i . x_i)} - t_i . x_i$$

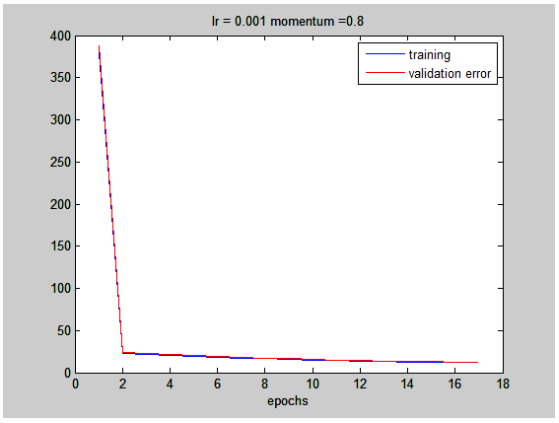
$\Delta w_{n-1} = w - \eta\,(1 - mu).\Delta_w E - mu.\Delta w_{n-1}$, where ŋ is the learning rate and mu the momentum term.

a.



b.



c. 

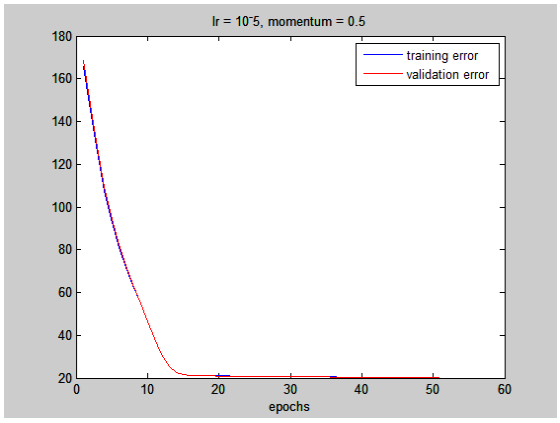
d. 

Fig 2 . Effects of learning rate and momentum term on logistic regression

High momentum, as presented in c. sticks gradient into a wrong direction, and final error remains extremely high. Indeed, the lower momentum, the faster method converges. However, looking at validation and misclassification errors, minima is not obtained for lowest momentum: lowest misclassification obtained is approximately 9% and is achieved with mu = 0.2.
Regarding to the learning rate, logistic regression gradient is highly sensitive to high learning rates, and classification cannot be achieved with high rate values. Graph d. on figure above was obtained with a very low learning rate, and convergence speed is significantly and excessively slowed down. From all these results, optimal learning rate – momentum pair was fixed to {0.001,0.2}.

3. MLP

MLP network parameters – learning rate, momentum term, hidden layers size - are first set according to theoretical values but we search for optimal parameters respect to the convergence speed and validation error, on a random subset of 50 images taken from training set .

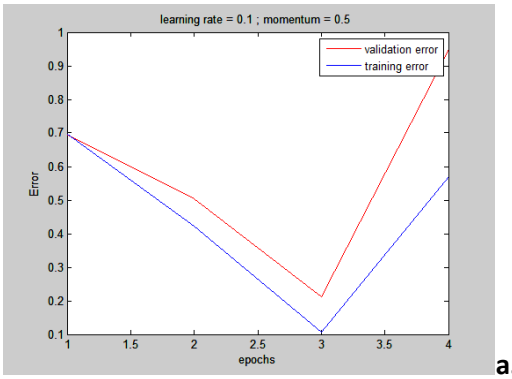
a) Weights initialization

At initialization we want the weights to be small enough around the origin so that the activation function operates in its linear regime, where gradients are the largest. We also want to conserve variance of the activation values and gradients from layer to layer so that convergence can be fast. One solution is to pick initial random weights in a variance range according to the size of layer they belong to : we pick weights in a Gaussian of width σ inversely proportional to layer size, so that the more inputs there are, the closer initial weights are to each other, and convergence speed and accuracy are maintained. Practically, all weights and bias pairs are stocked in a cell to facilitate their call.

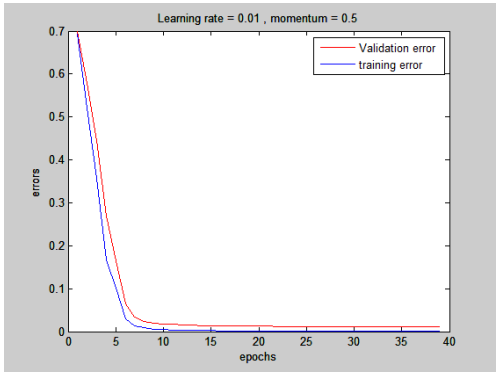
b) Learning rate & momentum term

We ran training on subset data with several combinations of momentum and learning rate : we observe that High values of these parameters allow fastened training convergence –especially high learning rates, as shown in fig a.- but there is less accuracy in the final result, since previous states are too much determining values update direction and training stops early. Too little momentum value makes system oscillate and convergence is slowed down. Too large learning rate makes system to converge very fast, but error consequently stays very large ; learning rate and momentum affect convergence speed, but also absolute error values at the end, since both are related together : the faster convergence, the higher remaining error.
To sum up, learning rate should not be too high for classification process and momentum term also does not have to be too large not to block gradient in a single direction but allow its readjustment when iterating. Decreasing the learning rate over time could also be relevant to ensure classification time-accuracy balance (high learning rate at the beginning ensure fast processing, while small rate at the end fine tunes result).
Minimizing the validation error over the several tested combinations, and taking into account that convergence has to be fast enough, we decided to choose a 0.01 value for learning rate (as presented in b.) during first 50 epochs to ensure relatively fast convergence, and then to decrease it as an hyperbolic function. A lower learning rate like 0.001 gives a lower error, but the difference is not significant respect to the additional time its take to converge.

Regarding to momentum, as it only affect total time of convergence and not the initial error slope neither the final error value, we chose the lowest value (not to block gradient in a unique direction) that ensures faster convergence as possible, i.e mu= 0.2.



a.



b.

Fig 3 Effects of learning rate on MLP convergence speed and accuracy

c) Hidden units

The number of hidden units needed is highly dataset-dependent. Theoretically optimal number of hidden unit can be analytically determined[1] :

THEOREM 2. *Let $n \in \mathbb{N}$ with $n \geq 2$, $\sigma:\mathbb{R} \to I$ be a sigmoidal function, $f \in C(I^n)$ and ϵ a positive real number. Then for every $m \in \mathbb{N}$ such that $m \geq 2n + 1$ and $n/(m - n) + v < \epsilon/\|f\|$ and $\omega_f(1/m) < v(m - n)/(2m - 3n)$ for some positive real v, f can be approximated with an accuracy ϵ by a perceptron type network with two hidden layers, containing $nm(m + 1)$ units in the first hidden layer and $m^2(m + 1)^n$ units in the second one, with an activation function σ in such a way that all weights and biases, with the exception of weights corresponding to the transfer from the second hidden layer to the output unit, are universal for all functions g with $\|g\| \leq \|f\|$ and $\omega_g \leq \omega_f$.*

But since typical number of hidden units vs. generalization performance graph is U-shaped [2], we chose to find the minimizing values thanks to a manual dichotomy method looking at validation error got – testing has been done with optimal learning and momentum parameters founds above-. Since we are using early stopping on validation train, it is essential to use a rather high number of hidden units to avoid bad local optima.[3]

The size of the hidden layers, if too small (h_1, h_2 under 5) or too big (over 20) leads to high error rates what can be attributed to underfitting and overfitting phenomenons respectively.
Comparing the validation and training errors obtained running MLP with several configuration, we observed that convergence was very rapid in case of restricted number of hidden units, with a particular sensitivity to the second layer size. Extreme cases with single hidden unit per layer are presented below: if it converges in less than two epochs, it is clear that data is overfitted. Increasing the first layer size has no effect on the convergence speed but can decrease final error. It could even be a source of overfitting if it is too large and first layer weights

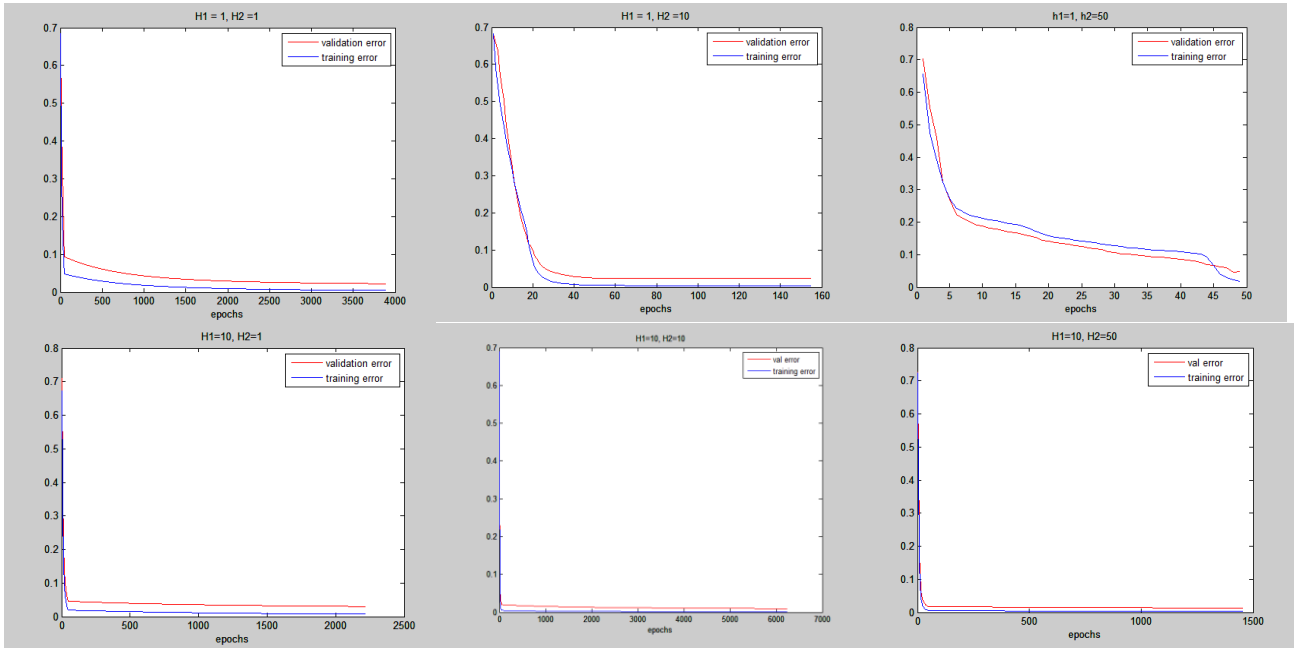
[1] Kolmogorov's Theorem and Multilayer Neural Networks
VERA KORKOVA, Neural Networks Volume 5, Issue 3, 1992, Pages 501–506

[2] Generalization Error and the Number of Hidden Units in a Multilayer Perceptron, David J.C. MacKay , Martin J. Oldfield (1995)

[3] Sarle, W. S. (1995). Stopped training and other remedies for overfitting. In*Proceedings of the 27th Symposium on the Interface of Computing Science and Statisfi (.'. _\'. pp. 352-360. Interface Foundation of North America, Fairfax Station. VA, USA*.

too close of inputs. Until h1 = 40 hidden units, validation error is slightly higher but do not explode, meaning that fitting is correct. For both speed convergence and model generalization capacity, the critical variable is second layer size – or more precisely h1/h2 ratio- : if h2 >>h1, the model is underfitted, and input arriving to second layer unsufficient to match relevant data (cf graph h1=1, h2=50). If now we increase h2 keeping a h2/h1 ratio into [1.5;3], convergence speed is increased and there is no aberrant step – errors curves are smooth-. The smaller second layer is, the faster it converges, and there is no significant change when h2/h1 ratio is inverted, so we decided to fix optimal network size to H1=10, H2=8 for binary classification. Similar adjustment make us choose H1=80, H2=60 for multi classes MLP, that indeed needs more units in order to deal with task complexity.

Fig 4 Effects of hidden layers size on MLP convergence speed and accuracy



Regarding to the overfitting , it has been observed that a neural network can support seven times as many parameters as data points without significantly overfit data[4]. A net initialized with small weights can develop large weights only after a large number of iterations: using early stopping, there is few overfitting phenomenon, whatever the excess network capacities are.

III. Results and discussion

In this section we will present validation and traing errors time-evolution during traing phase, then test and discuss the classifiers accuracy by computing the misclassification rate, confusion matrices and testing error from trained algorithms.

A. Linear Regression

Validation & training error in function of Tikhonov regularization parameter are linearly increasing, with a growing deviation between validation and training curve. Indeed, training weights are directly penalized by regularization and thus it have immediate balancing impact on the resulting error , but less on the validation one. We thus have to pick a regularization model that do not increase too much this deviation, as well as the validation error neither.

[4] Caruana, R., Lawrence, S., & Giles, L. (2001). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 402-408.

When choosing so ($\vartheta = 40$), regularized linear classifier is very robust and presents high reproducibility over trials, providing satisfying low error on testing set, with 22 % of misclassified images.

Errors values: (note that as solution is analytically provided, it is unique for one regularization parameter, and errors values remains constant over trials. Values were averaged over CV trials.)
Training Error = 0.035509
Validation Error = 0.072919
Testing Error = **0.84162**
Misprediction on test set = 22.9815 %

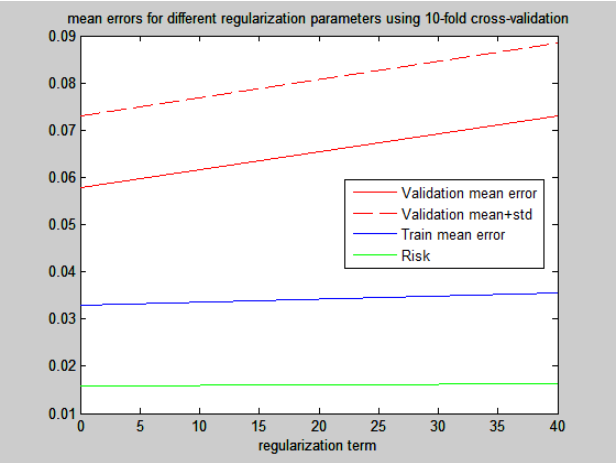


Fig 5 . Validation, risk and training errors of linear regularized classifier

Confusion matrix

predicted class \ Real class	1	2	3	4	5
1	589	713	361	165	178
2	176	232	99	185	114
3	74	28	218	151	229
4	98	33	226	372	139
5	143	74	176	207	420

The main advantage of this model is its mathematical simplicity in the mathematics as well as its short computation time, and it ability to generalized to extreme data sets because it takes into account marginal weights. Despite these considerable advantages, it remains the issue of knowing whether the data set is linearly separable or not.

B. Logistic Regression and MLP

1. Logistic regression

Evaluation of validation and training error over epochs is following a linear by interval curve, with initial elevated slope and slower convergence afterwards. Logistic regression behavior seems have lower robustness than the other methods, since time to converge can vary a lot depending on weights initialization. With optimal parameters chosen in method section (learning rate = 0.001, momentum = 0.2), validation error provides very encouraging results, since it is still lower than training error, meaning the ability to adapt new data is high and training set has not be overfitted.
However, error on testing set remains high : with the optimal parameters, we obtain the following average errors:

Average errors values
Training Error = -10.8
Validation Error = -11.09
Testing Error = 9.7
Misprediction on test set = 40% +/- 2.1%

Example of confusion matrix

predicted class \ Real class	1	2	3	4	5
1	589	713	361	165	178
2	176	232	99	185	114
3	74	28	218	151	229
4	98	33	226	372	139
5	143	74	176	207	420

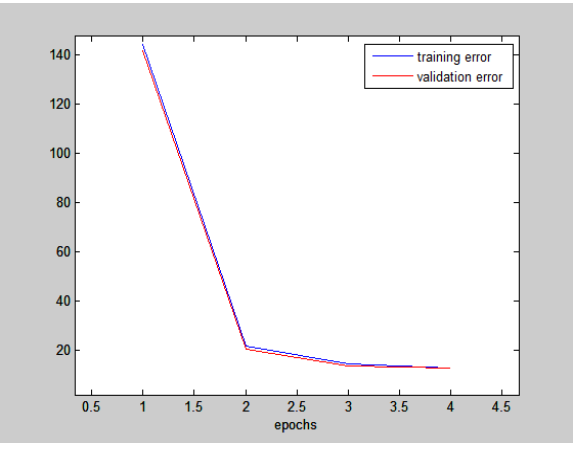


Fig 6. Validation and training error obtained with logistic regression

test_error = 10.2597

2. Multilayer Perceptron

A- Binary MLP : A reasonable fitting to data and convergence speed can be achieved fine-tuning the learning rate, momentum and layers sizes. Gradient descent on error is smoother when we use a medium momentum term, otherwise direction is oscillating at each step.

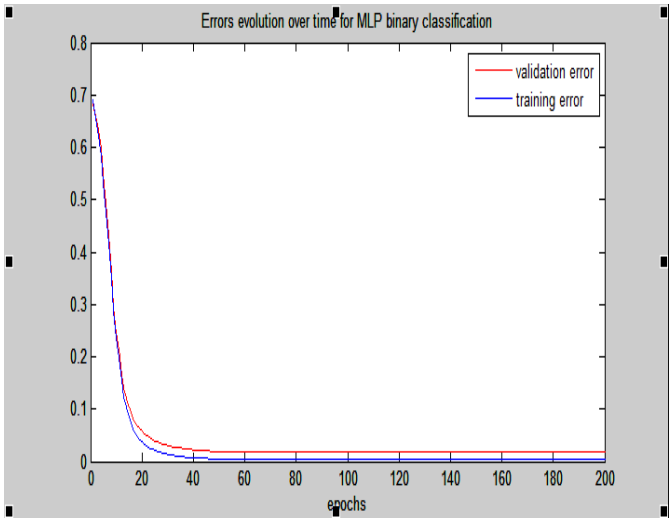


Fig 7 . binary MLP validation and training errors over epochs(settings: H1 = 10;H2 = 8; nu_start = 0.001; pi = 0.01) . Test error: 0.0089

Average errors values
Training error = 0.029
Validation Error = 0.021
Testing Error = 0.0189
Misprediction on test set = 3.5% +/- 0.5%

Example of confusion matrix :

(misclassification rate = 3.7 %)

	1	-1
1	870	7
-1	61	862

B- Multi-way MLP : Note that initial misclassification error is oscillating between 70-80% depending on weights initialization, what is coherent with the prior probability to misclassify data of 4/5. Once trained weigth optinal parameters, we obtain :

Average errors values
Training error = 0.042
Validation Error = 0.089
Testing Error = 0.565
Misprediction on test set = 14.5% +/- 1 %

Example of confusion matrix

		real class				
		1	2	3	4	5
predicted class	1	923	142	75	4	26
	2	108	899	8	0	0
	3	42	13	922	0	24
	4	0	0	4	1067	175
	5	7	26	71	9	855

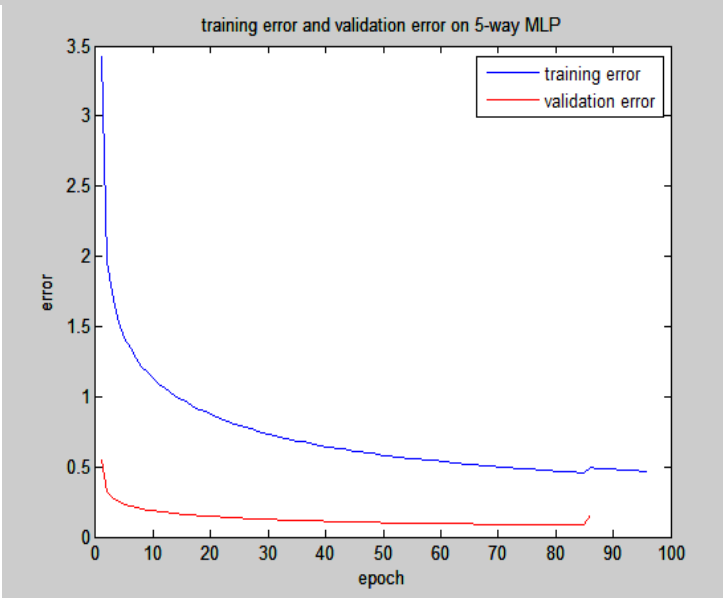


Fig 8. MLP validation and training errors over epochs(settings: H1 = 100;H2 = 80; nu_start = 0.001; pi = 0.01) . Test error: 13.2037

In both cases, error constantly goes down, indicating that the model is not prone to overfitting.
Compared to binary MLP, computing time is longer: gradient descent has to be slower since it will have to proceed into larger calculations to discriminate the different classes, and also because the number of hidden units had to be increased to match the complex output landscape.

IV. Conclusion

We managed to achieve satisfying error rates for both the binary and the five-way classification using MLP. As expected, the network has to be adjusted and extended to be able to treat more data and divide them into more classes. For the binary set, error rate we obtain is remarkably close to zero. For five-classes problem, comparison between the three methods enables us to conclude that linear classifiers, trained with logistic or regularized squared error, are providing useful indications on dataset and are faster to run, but their error rate doesn't go below 20%. Regularized Tikhonov classifier is especially interesting because of the unicity of its solution, combined with an easy way to optimize and change parameter according to the classification needs. is more subtle and a few observations can be pointed out. But if linear classifiers can be useful as first baseline, there is no doubt that perceptron achieves better classification, regarding to the generalization and flexibility toward the input dataset size, thanks to its non-linearity- despite it is costly and delicate-.

V. Appendix

A. MLP Backpropagation calculations

$$r^{(3)} = \frac{dElog\ (w)}{d\ a^{(3)}} = \frac{d}{d\ a^{(3)}}\ \frac{1}{n}\ \sum_{i=1}^{N} \log\left(1 + e^{-t.a^{(3)}(x_i)}\right) = \frac{1}{n}\ \sum_{i=1}^{N} -\mathrm{t}.\, e^{-t.a^{(3)}(x_i)} / \left(1 + e^{-t.a^{(3)}(x_i)}\right)$$

Multi-classes MLP with squared error: $r^{(3)} = \frac{dEsquare\ (w)}{d\ a^{(3)}} = \frac{1}{2}\ \sum_{i=1}^{N} ||a^{(3)} - t_i||^2 =$

$$\begin{cases} \Delta_{w(3)}E = r^{(3)}.\, z^{(2)} \\ \Delta_{b(3)}E = r^{(3)} \end{cases}$$

$$\begin{cases} w^{(3)} = w^{(3)} + \eta\Delta_{w(3)}E \\ b^{(3)} = b^{(3)} + \eta\Delta_{b(3)}E \end{cases}$$

$$r_{L,R,LR}^{(2)} = \frac{dE}{da^{(2)}} = \frac{dE}{d\ a^{(3)}}\frac{da^{(3)}}{dz^{(2)}}\frac{dz^{(2)}}{da^{(2)}} = r^{(3)}.\, w^{(3)}.\frac{dg\ _2(a_{L,R,LR}^{(2)})}{da_{L,R,LR}^{(2)}} \quad \text{for L,R, LR units}$$

$$\begin{cases} \Delta_{w(2)L,R}E = r_{L,R}^{(2)} z_{L,R}^{(1)} \\ \Delta_{b(2)L,R}E = r_{L,R}^{(2)} \end{cases} \quad \text{for L and R units} \quad \& \quad \begin{cases} \Delta_{w(2)LR}E = \{ r_{LR}^{(2)} z^{(1)}{}_{L};\ r_{LR}^{(2)} z^{(1)}{}_{R} \} \\ \Delta_{b(2)LR}E = r_{LR}^{(2)} \end{cases} \quad \text{for LR units}$$

$$\begin{cases} w^{(2)} = w^{(2)} + \eta\Delta_{w(2)}E \\ b^{(2)} = b^{(2)} + \eta\Delta_{b(2)}E \end{cases} \quad \text{for L, R, LR units}$$

$$r_{L,R}^{(1)} = \frac{dE}{da_{L,R}^{(1))}} + \frac{dE}{da_{LR}^{(1))}} = r_{L,R}^{(1)}.\, w_{L,R}^{(2)}.\, g'\ _1(a^{(1)}{}_{L}) + r_{LR}^{(1)}.\, w_{LR}^{(2)}.\, g'\ _1(a_{LR}^{(1)}) \quad \text{for L and R units}$$

$$\begin{cases} \Delta_{w(1)L,R}E = r_{L,R}^{(1)} x_{L,R} \\ \Delta_{b(1)L,R}E = r_{L,R}^{(1)} \end{cases} \quad \text{for L and R units}$$

$$\begin{cases} w^{(1)} = w^{(1)} + \eta\Delta_{w(1)}E \\ b^{(1)} = b^{(1)} + \eta\Delta_{b(1)}E \end{cases} \quad \text{for L and R units}$$

B. MLP Notations & dimensions

Input	Notation	Dimension
number of images	N (=3600)	scalar
preprocessed data from left camera, resp. training , validation and testing sets	dlt1,dlv1,tdl1	1/3 N x input size, 1/6 N x input size , 1/2 N x input size
preprocessed data from right camera, resp. training , validation and testing sets	drt1,drv1,tdr1	1200x576, 600x576 , 1800X 576
Activation values and weights variables		
layer 1 activation values for resp. Left and Right groups	a1_L, a1_R	1xh1, 1xh1
layer 1 transformed activation values for resp. Left and Right groups	z1_L, z1_R	1xh1, 1xh1
layer 2 activation values for resp. Left, Right and Middle groups	a2_L, a2_R, a2_LR	h2x1, h2x1, h2x1
layer 2 transformed activation values	z2	1xh2
layer 3 activation value	a3	scalar / 5x1
weights of connections between resp. Left or Right inputs and first layer units	network{1}.W , network{2}.W	h1xh2, h1xh2
bias of connections between resp. Left or Right inputs and first layer units	network{1}.B, network {2}.B	h1x1,h1x1
weights of connections between first and second layer units for resp. Left, middle and Right units groups	network{3}.W , network{4}.W , network{5}.W	h2xh1, h2x2h1, h2xh1
bias of connections between first and second layer units for resp. Left middle and Right units groups	network{3}.B, network{4}.B , network{5}.B	h2x1, h2x1, h2x1
weights of connections between second and third layer units	network{6}.W	1xh2 / 5xh2
bias of connections between second and third layer units	network{6}.B	scalar / 5x1
weight update between resp. Left or Right inputs and first layer units	network_delta{1}.W , network_delta{2}.W	h1xh2, h1xh2
bias update between resp. Left or Right inputs and first layer units	network_delta{1}.B, network_delta {2}.B	h1x1,h1x1
weights update between first and second layers units for resp. Left , middle and Right units groups	network_delta{3}.W , network_delta{4}.W , network_delta{5}.W	h2xh1, h2x2h1, h2xh1
bias updates between first and second layers units for resp. Left and Right units groups	network_delta{3}.B, network_delta{4}.b , network_delta{5}.B	h2x1, h2x1, h2x1
weights update between second and third layers units	network_delta{6}.W	1xh2 / 5xh2
bias update between second and third layers units	network_delta{6}.B	scalar / 5x1
Backpropagation variables		
Layer 3 residual error	r3	scalar / 5x1
Layer 2 residual error	r2_L, r2_R,r2_LR	1xh2, 1xh2, 1xh2
Layer 1 residual error	r1_L, r1_R	1xh1, 1xh1
Layer 3 gradient along weights	gradw3	1xh2 / 5xh2
Layer 3 gradient along bias	gradb3	scalar / 5x1
Layer 2 gradient along weights for resp. Left, Right and Middle units groups	gradw2_L, gradw2_R, gradw2_LR	h1xh2, h1xh2, h1x2h2
Layer 2 gradient along bias for resp. Left, Right and Middle units groups	gradb2_L, gradb2_R, gradb2_LR	1xh2, 1xh2, 1xh2
Layer 1 gradient along weights for resp. Left or Right units groups	gradw1_L, gradw1_R	1xinput size, 1xinput size
Layer 1 gradient along bias for resp. Left or Right units groups	gradb1_L, gradb1_R	1xh1, 1xh1

Red : multi-class MLP