Machine Learning for Computer Vision

[MVA 2015/2016]

Programming Assignment 3

Maha ELBAYAD

1 Building classifiers as neural network layers

For each of the SVM, logistic regression and linear regression we implement a loss function to run a neural network version of the optimization algorithms. Each input x_i has a score $f(x_i)$ related to the probability of the sample being positive.

$$\begin{aligned}
\text{(Linear regression) } l_i &= \frac{1}{2}(f(x_i) - y_i)^2 \\
f(x_i) &= w^T x_i + b \\
\text{(Logistic regression) } l_i &= y_i f(x_i) + (1 - y_i)(1 - f(x_i)) \\
f(x_i) &= \text{sigmoid}(w^T x + b) \\
\text{(SVM) } l_i &= \max(1 - y_i f(x_i)) \\
f(x_i) &= w^T x_i + b
\end{aligned}$$

Results with different parameters α (leraning rate) and λ (weight decay)

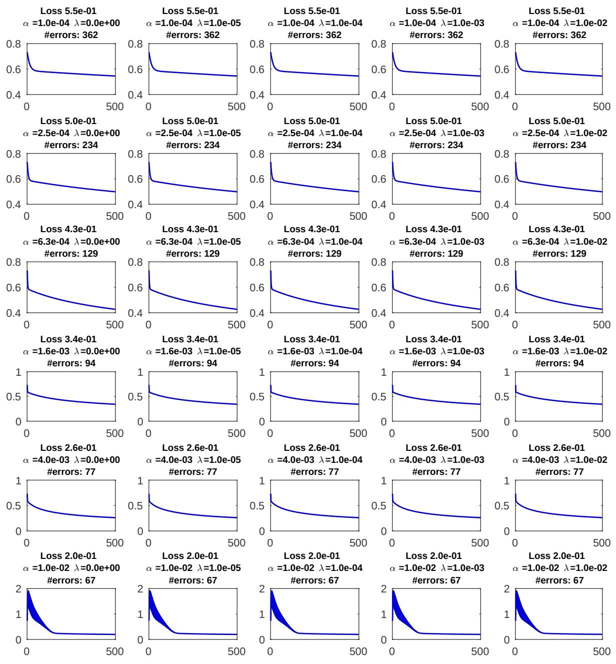


Figure 1: *Logistic regression*

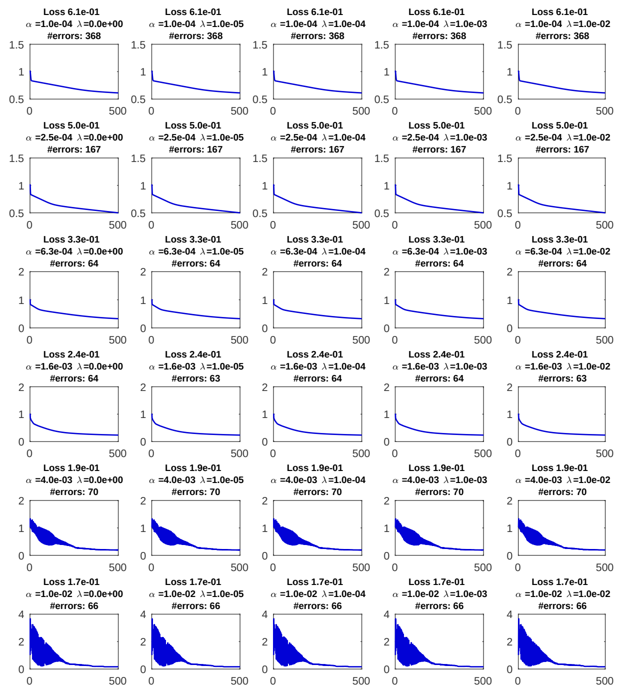


Figure 2: *SVM*

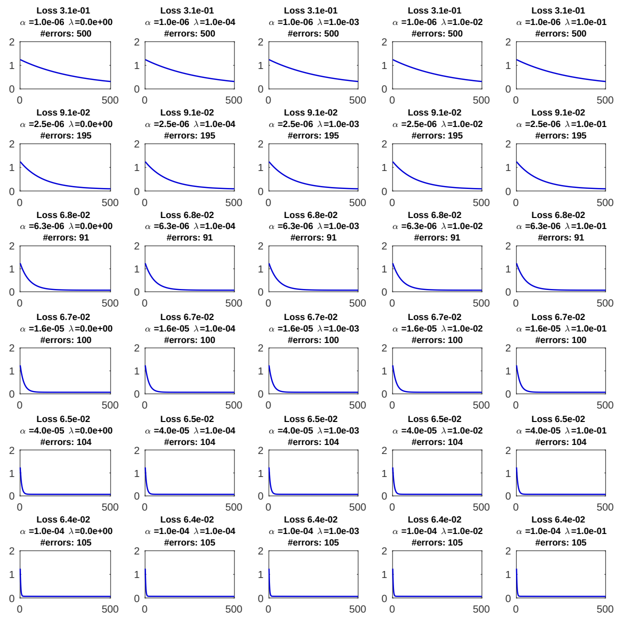


Figure 3: *Linear regression*

We note that for all three models, the evolution of the loss

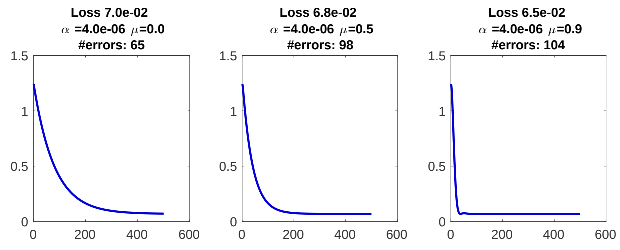
and the final number of misclassified samples is independent from the weight decay parameter. This term is equivalent to a regularization parameter that causes the weights w to exponentially decay to zero in order to avoid over-fitting the training set.
In the update:

$$w_{t+1}^{i} := w_i^t - \alpha \frac{\partial L(w)}{\partial w^i} - \alpha\lambda w_i^t$$

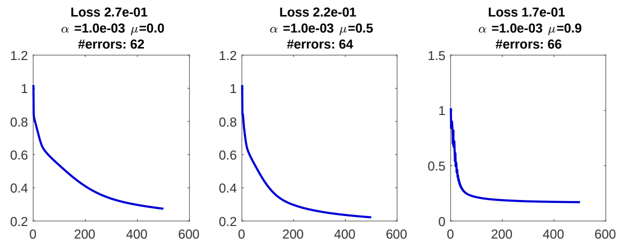
Since we're obliged to use small learning rate α for the optimization to converge, the decay term $\alpha\lambda$ is too small to be effective.
Best results with SVM and logistic regression were achieved with a relatively large learning rate α =1e-2 where the learning overshoots at first before stabilizing.

Momentum:

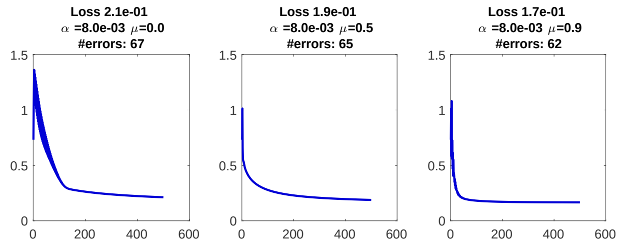
$$w_{t+1}^{i} := w_i^t + \mu\Delta w_t^i - \alpha \frac{\partial L(w)}{\partial w^i} - \alpha\lambda w_i^t \tag{1}$$



(a) *Linear regression*



(b) *SVM*



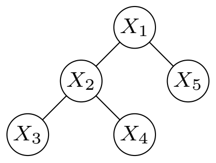
(c) *Logistic regression*

Figure 4: *Neural networks with momentum*

As can be seen in figure 4 and equation (1), the additional momentum term allows an accelerated convergence to the minimum as it keeps track of the previous update and take larger steps in directions of persistent reduction as well as preventing chaotic jumps (e.g. in the logistic regression we eliminated the overshooting occurring with $\mu = 0$)

2 Sum/Max-Product algorithms

We implement the sum-prodcut algorithm on the following undirected tree:



We first pass the messages from the leaves $\{3, 4, 5\}$ to their parents $\{2, 2, 1\}$ and then 2 passes its messages to the root 1, which will allow us to compute the marginal probability $\widetilde{P}(X_1)$. Afterwards, we pass the messages in the opposite direction and we stop at 2 since we only need to compute $\widetilde{P}(X_2)$.
A message from node X_i to X_j is evaluated as:

$$\mu_{i\to j}(X_j) = \sum_{X_i} \psi_i(X_i)\psi_{i,j}(X_i, X_j) \prod_{k\in\mathcal{N}(i)\setminus j} \mu_{k\to i}(X_i)$$

In a similar fashion, we implement the max product algorithm (takes the maximum instead of the sum in the definition of the passed message).

Sum-product results: Marginal probabilities:

$$\widetilde{P}(X_1) = [0.5580, 1.0170, 0.3400]$$
$$\widetilde{P}(X_2) = [0.8700, 1.0450]$$

Both summing to $Z = 1.9150$.

$$P(X_1) = [0.2914, 0.5311, 0.1775]$$
$$P(X_2) = [0.4543, 0.5457]$$

We evaluate $P(X)$ on some events:

$$P(X = \{1, 2, 4, 2, 1\}) = 0.0056$$
$$P(X = \{3, 1, 2, 1, 2\}) = 0.0033$$
$$P(X = \{2, 2, 1, 2, 1\}) = 0.0134$$