Machine Learning 2015, Hand In 2

October 1, 2015

Intro

In this handin, you will build more classifiers for optical character recognition (OCR). The main exercise is building a classifier for the AuDigits data set consisting of digits that you drew in class. The goal is simply to do as well as possible on the unknown images I have kept secret, and of course to tell the customer (me) how well your classifier will do. There are also a few theoretical exercises regarding gradients and neural nets which you find at the end.

The implementation part of this assignment is very open, and you yourself have to figure out where to put the focus. There are some things you must try which are listed below. Note that the experiments you are to run will **take quite some time**, so it is a good idea to get started early and use your time smart, that is, use the time on what you believe are the most relevant experiments.

Report

You can write your report in Danish if you prefer. The maximal report length is 5 pages. You are allowed to be up to 3 members in a group. You are encouraged to discuss the exercise between groups and help each other as much as possible without of course copying each others work. Particularly, discussing the quality of your classifiers is a good idea to get an indication if you are doing it correctly. Questions are always welcome on the BlackBoard discussion board. In your report, be sure to discuss all the choices you have made in your implementation, and explain why your think the quality of your classifier is as it is. Please upload your report in a PDF file and your code (for the great classifier you have made) in a separate ZIP file. The format for that code is described in the deliverables section.

There are associated data files for you on the website. The files are:

mnistTrain.npz MNIST data stored in NumPy format. We have changed it from hand-in 1 to be stored in the same order as auDigits.

mnistTest.npz A test set for MNIST digits.

auTrainMerged.npz The training set of your images.

nn.py Python implementation of neural nets.

handin.py Example code that shows how you must hand in your final classifier.

test_handin.py An example Python script that shows how we will run the classifier you upload. Your code must be able to work with this script.

to_students.py Python code showing how to write better and faster Python code and apply the minimize function.

Baseline

Before you start we need a baseline classifier to compare with. This will be the softmax model you did in hand in 1. To optimize that you must use a good optimizer and several are available in SciPy. See `http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize`. In `to_students.py` there is an example that shows how to apply the `minimize` function.

Support Vector Machines

The first thing you must try is Support Vector Machines. An implemetation is available in the `sklearn` package. There is excellent example code in the documentation: `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

SVM Code Write code such that you can apply an SVM to your input data and get a model back and use it to do prediction on new images. The model object contains (among other things) a list of the support vectors that you should look at.

Linear Kernel Try the linear kernel with different values for C, the cost of being on the wrong side of the margin.

Polynomial Kernel Try the polynomial Kernel with different polynomial degree d and C. You must at least try $d = 2$, but higher degrees are of course allowed.

RBF (Gaussian) Kernel Try RBF Kernels with different shape parameter γ (λ in the notes), and C.

You should use validation whenever you need to make a choice. Finding (few) hyperparameters is usually done by grid search. For instance for the RBF kernel you need a good C (cost of wrong side of margin) and γ. Usually this is done by defining a list of values for each parameter and then trying all combinations. This gives a grid. One can do that in multiple iterations to slowly zoom in on a good value pair.

Neural Nets

Try using standard neural nets with the code given to you. You must again experiment with the hyperparameters, in particular network size (hidden layer) and regularization, and use validation to find the best combination.

You are welcome to extend the code in any way you like for instance to implement softmax output or use it as an autoencoder and try making your own features as in the lectures. There are several examples in the code file that shows you how to use it.

Deliverables

In the report, explain what you have done and and provide tables that show your results. Include tables of your validation results for the baseline model, SVMs and Neural Nets on the Au data set as well as any other interesting experiments you have done.

After you have done all your experiments you must select what you think is the best classifier possible. You should create a Python code file that can evaluate that classifier on an input matrix of images (that are formatted as the training data). The file should be formed as in the example in *handin.py* and you must verify that it works with *test_ handin.py*. If you store your model in a separate `npz`-file, remember to include this in your ZIP alongside your code.

Describe the classifier you decide on in the report, and give an estimate of how well it will work on the unknown data and a reason for that estimate.

Finally there are some theoretical regarding neural nets and gradients you must answer. These can be found at the end.

Competition

We will evaluate all classifiers we receive, and the group that gets the most images correct will (probably) win a prize. (If a classifier takes excessively long time it will be killed and get a score of 0.) At the last lecture we will show the results of your labor and declare a winner. The TA and Teacher will enter the competition with a baseline classifier (based on SVM or Neural Net) and the prize is only given if the winner beats that classifier – so get creative. Last year's winner had just shy of 97% correct on last year's data set. There are more, and potentially more different, data to use this year.

Here are some tricks you can apply to have a better chance of winning the competition.

Small Data Set

You should use regularization (e.g. selecting any form of hyper parameters) for all the classifiers you train. But there is still the issue that the data set for AuDigits is small compared to the dimensionality of the images.

You can generate more data by considering the task we are given. A picture of a 5 is still a 5 even if add a little random noise to the image, or if you stretch, scale or rotate the image slightly. The `scipy.ndimage` module has a lot of functionality. Read more here: `https://scipy-lectures.github.io/advanced/image_processing/`

The MNIST digits is 60000 train images + 10000 test images you can use, but they may not behave the same way as the AU data set.

Dimensionality Reduction/Pretraining

As discussed in the lecture, we can reduce the input dimensionality while still having most of the information intact. This should at least speed up the computation, but may also lead to better generalization. You can use SciPy's built in packages for this purpose, see `http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`.

Theory Exercise: Gradient for Neural Nets

In this exercise we consider the gradient computation for feed forward neural nets for supervised learning. We are given some neural net architecture that defines our hypotheses space. The neuron activation functions (except for the output neurons) is the logistic function $\sigma(z) = 1/(1 + e^{-z})$. Let D denote the input training data and $nn_{w,b}(x)$ the value of the neural net using weights w, b on input x. We are interested in the first step in backpropagation. Let s_1^L be the weighed input to the (first) output neuron, and let ϕ_o denote the transform in the output neuron(s). This means that $nn_{w,b}(x) = \phi_o(s_1^L)$ for a network with one output. Let the (pointwise) error function we are minimizing be denoted e. In backpropagation, after the forward pass the first we do is to compute $\delta_1^L = \frac{\partial e}{\partial s_1^L}$.

Regression In this first part, we want to do solve a regression supervised learning problem and use the least squares error function. This means that $e(x, y) = \frac{1}{2}(x - y)^2$. There is only one output neuron and it computes the identity function, i.e. $\phi_o(z) = z$. Write down the cost function (in sample error) we want to minimize using D and $nn_{w,b}(x)$.
Consider the in sample error on one point x, y, that is $e(nn_{w,b}(x), y)$. By using the chain rule, show that $\delta_1^L = \frac{\partial e}{\partial s_1^L} = (nn_{w,b}(x) - y) = (s_1^L - y)$.

Binary Classification This second part is the same as the first part, but with two class classification where we are estimating probabilities as with logistic regression. The output neuron ϕ_o is the logistic function, σ, and the error function is cross entropy, $e(p, z) = -(z \ln p + (1 - z)\ln(1 - p))$. The neural net is outputing the probability of being in class 0, i.e. $nn_{w,b}(x) = \phi_o(s_1^L) = \sigma(s_1^L)$ which is a number between zero and one.

Write down the cost function as a function of D and $nn_{w,b}(x)$.

Show that $\delta_1^L = -y(1 - \sigma(s_1^L)) + (1 - y)\sigma(s_1^L)$. Again you must use the chain rule.

Hint: It is probably easier if you write the cost function e (still on one point) as a function that uses σ and s_1^L; then the work should look familiar. Look at hand-in 1 if you need help with the derivative of σ.

Softmax: Bonus Exercise – For the adventurous This exercise is for people who feel that they not used all the time allocated for the the hand in solving the other exercises.

Softmax is as usual more technical. But let us try. We will change the notation a bit from previous exercises.

Let $h(x) = a = [a_1^L, \dots, a_k^L]$ be the vector of K outputs of the neural net after the softmax transform. The error function is $e(p, z) = -\sum_{i=1}^K [z = i] \ln p$, where p is a vector of size K (of non-negative numbers that sum to 1) and z is a label in $\{1, \dots, K\}$. Again this is just as cross entropy above, generalized to K classes. Write down the cost function as a function of D and $h(x)$.

In Figure 1, we see a simple example of the output part of a neural net with three outputs. The problem is that in order to compute softmax in the output nodes, the neurons must share information (see information on softmax). One can encode the softmax function as a feed forward net, which leads us to the net in Figure 2. We have split the net in two, one for the softmax and one for the *standard* neural net. As you can see, some of the weights in the softmax net are hardcoded so that it computes the softmax function. This is an illustration of softmax as a neural net.

This indicates to us the following idea. We think out hypothesis as a function composed of the output neurons and the remaining network, that is $\phi_0(nn_{w,b}(x))$, where in this case ϕ_o is actually the softmax neural net, while in regression above it was the identity function neural net.

We write $h_{\theta,w,b}(x) = \text{softmax}_\theta(nn_{w,b}(x))$, and we need $\frac{\partial e}{\partial \theta}, \frac{\partial e}{\partial w}, \frac{\partial e}{\partial b}$.

In this case, θ is a $d \times K$ matrix where d is size of the last (output) neural net layer before the softmax net. This is the same as in hand in 1, and you can refer to the results from there without proof. Let A be the one row matrix that stores the output of the neural net before input to the softmax layer.

What is $\frac{\partial e}{\partial \theta}$? (Hint: See Hand In 1)

Show that $\frac{\partial e}{\partial w} = A^\intercal(Y - \text{softmax}(A\theta))\frac{\partial nn_{w,b}(x)}{\partial w}$. The last part we can compute with backpropagation so we will ignore it here (similar for bias).

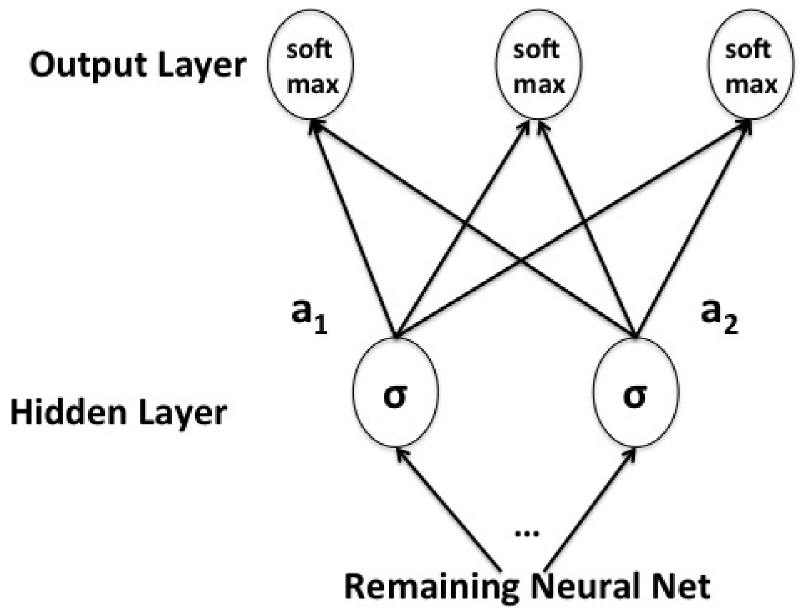
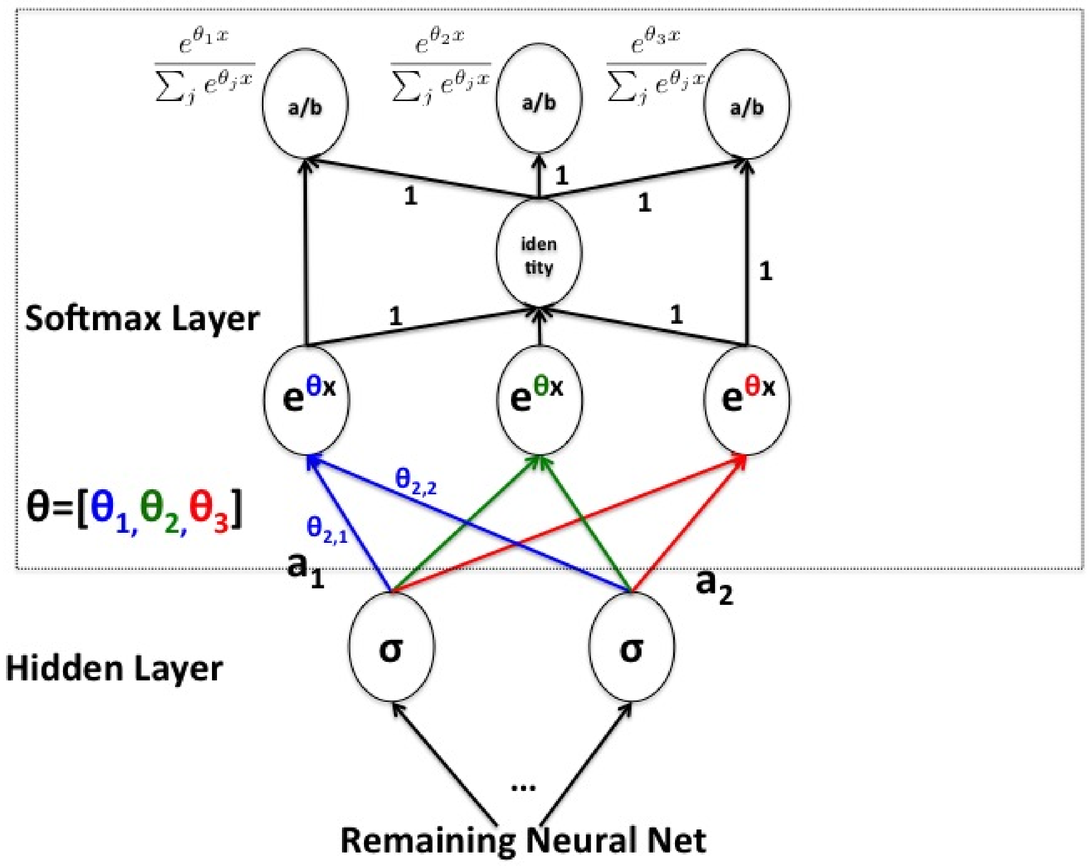


Figure 1: Abstract Representation of Softmax Output Layer



$\frac{e^{\theta_1 x}}{\sum_j e^{\theta_j x}}$
$\frac{e^{\theta_2 x}}{\sum_j e^{\theta_j x}}$
$\frac{e^{\theta_3 x}}{\sum_j e^{\theta_j x}}$
a/b
a/b
a/b
1
1
1
1
1
1
1
iden tity
Softmax Layer
$e^{\theta x}$
$e^{\theta x}$
$e^{\theta x}$
$\theta_{2,2}$
$\theta=[\theta_1,\theta_2,\theta_3]$
$\theta_{2,1}$
a_1
a_2
Hidden Layer
σ
σ
...
Remaining Neural Net

Figure 2: Softmax Computation as Neural Net