IN5400 – Machine Learning for Image Analysis

Alex

Week 02/03: SGD for logistic regression in python

1 logistic regression in pytorch with applying the computed gradient by hand

The goal is to implement SGD for logistic regression. You work in `pytorch_logreg_gdesc_studentversion.py`.

What does one need to run logistic regression in pytorch? The computational flow is as follows:

1. define dataset class and dataloader class
2. define model
3. define loss
4. define an optimizer
5. initialize model parameters, usually when model gets instantiated
6. loop over epochs. every epoch has a train and a validation phase
7. train phase: loop over minibatches of the training data
   (a) set model to train mode
   (b) set model parameters gradients to zero
   (c) fetch input and ground truth tensors, move them to a device
   (d) compute model prediction
   (e) compute loss
   (f) run `loss.backward()` to compute gradients of the loss function with respect to parameters
   (g) run optimizer (here by hand) to apply gradients to update model parameters

8. validation phase: loop over minibatches of the validation data
   (a) set model to evaluation mode
   (b) fetch input and ground truth tensors, move them to a device
   (c) compute model prediction
   (d) compute loss in minibatch
   (e) compute loss averaged/accumulated over all minibatches
   (f) if averaged loss is lower than the best loss so far, save the state dictionary of the model containing the model parameters
9. return best model parameters

Things that need particular attention:

1. a subclass of `torch.utils.data.Dataset` class.

   Its functionality:

   `thisclass[i]` returns a tuple of pytorch tensors belonging to the i-th datasample.

   - We use `torch.utils.data.TensorDataset`.
   - Takes as argument n tensors (e.g. features, labels, filenames):
     `dataset=torch.utils.data.TensorDataset(t0,t1,...tn-1)` .
   - `thisclass[i]` will be then a tuple of n tensors:
     ```
     thisclass[i][0]=t0[i]
     thisclass[i][1]=t1[i]
     ...
     thisclass[i][n-1]=tn-1[i]
     ```
   - we create two datasets: one from the training data numpy array, a second from the validation data numpy array.
   - the numpy arrays must be converted into pytorch tensors before handing them over

2. a `torch.utils.data.Dataloader` class: takes the Dataset as input, and returns a python iterator which produces a minibatch of some batch size every time it is called. Common parameters are batchsize, whether the data should be shuffled (suggested during training), or a sampler class which allows to control how minibatches are created.

3. define a model which takes samples as input, and produces an input. In pytorch it is a class derived from `nn.Module`. A standard neural network module (no matter pytorch or mxnet) needs usually two functions:

   - `def __init__(self,parameter1,parameter2):`
     where you define all layers which have model parameters or you define your parameter variables

   - `def forward(self,x):`
     where the input x is processed until the output of your network.

   `class logreglayer(nn.Module):` is the model where I have predefined w and $bias$, and you have to define `def forward(self,x):`

4. a loss function, used at training phase to measure difference between prediction and ground truth. Possibly, also a loss function at validation phase to measure the quality of your prediction on your validation dataset

5. an optimizer to apply the gradients to model parameters. In this lecture we will not use a pytorch optimizer, but updates weights

What do you have to do? check the #TODO tags.

- define the forward in the model `class logreglayer(nn.Module):`
- use `torch.utils.data.TensorDataset`
- initialize an instance of your model
- define a suitable loss for a binary classification with only one output, check docs on pytorch loss functions
  - in the first step: run `optimizer.step()`
  - in the second step: apply gradient to model parameters. Note what was said in the lecture about fields of parameters.

The code should give you an accuracy of 0.78 to 0.79 most of the time - with `optimizer.step()`, and when doing gradient descent by coding

2 Task2

Let X,A be arbitrary matrices and A invertible. Solve for X:

$$XA + A^\top = I$$

As above, let $B \in \mathbb{R}^{n\times n}$ be positive definite, then solve for x

$$(Ax - y)^\top A + x^\top B = 0$$

3 n-dim Hyperplanes are a piece of bunny!

Here I want you to think a bit geometrically about these hyperplanes without the need to draw them.

3.1 3 dims

Suppose you have in 3 dimensional space the following dataset: $D_n = \{(x_1 = (-5, 1, 2), y_1 = +1), (x_2 = (1, 2, -3), y_2 = -1)\}$.
Find a linear classifier $f(x) = wx + b$ (means its parameters) which predicts all points correctly.

Hint:
It can help to think of a hyperplane defined in terms of an orthogonal vector and a bias term. Get the orthogonal vector to the hyperplane right first, bias you can determine then afterwards by plugging in points, and solving for the bias.
For two points, what would be a good orthogonal vector for a hyperplane if you have two points which need to be separated??

3.2 5 dims I

Suppose you have in 5 dimensional space the following dataset: $D_n = \{(x_1 = (1, 0, 1, -2, 6), y_1 = +1), (x_2 = (3, 1, -3, 5, 1), y_2 = -1)\}$.
Its the same thing as above, now in 5 dims. Find a linear classifier $f(x) = wx+b$ (means its parameters) which predicts all points correctly.

3.3 5 dims II

Suppose you have in 5 dimensional space the following dataset: $D_n = \{(x_1 = (1, 0, 1, 6, 3), y_1 = +1), (x_2 = (3, 1, -3, 5, 1), y_2 = -1), (x_3 = (2, -1, 0, 4, 2), y_2 = -1)\}$.
Good news: you cant draw it but still solve it. An alternative way is to project these three points into a 2-dim subspace, by the way, and solve it in the subspace.

How to solve it:
You can draw a line between the two points x_2, x_3 of the same label. This line can be part of many hyperplanes (its a 4d space of all possible hyperplanes). The vector $x_2 - x_3$ is parallel to this line .
These are the steps:

I **Now choose a vector w:**

- which is **not** parallel to $x_2 - x_3$,

and which will serve as candidate for the orthogonal of the hyperplane to be found.

II **Answer for yourself:**

  - what is a possible mathematical criterion to test that w is not parallel to the line $x_2 - x_3$?
  - what is a possible mathematical criterion to test that w is not orthogonal to the line $x_1 - x_3$?

III Next, **make this vector w into a vector w_2 which is orthogonal** to the line $z = x_2 - x_3$ by using the following:

$$w_2 = w - (w \cdot z)\frac{z}{\|z\|^2}$$

Why do I want you to do it?

$$w_2 \cdot (x_2 - x_3) = 0$$

implies that:

$$w_2 \cdot x_2 + b = w_2 \cdot x_3 + b$$

means $f(x_2) = f(x_3)$ no matter what choice of bias. Once you choose a bias, both points will lie on the same side of the hyperplane! Also note that the projection formula has a more symmetrical writing form which makes clearer what it does:

$$w_2 = w - (w \cdot \frac{z}{\|z\|})\frac{z}{\|z\|}$$

We remove from w the component in the direction of $\frac{z}{\|z\|}$. The amount of the component is $w \cdot \frac{z}{\|z\|}$

IV **Use w_2 to for your classifier as weight/direction, and determine a bias b which separates points x_1 from the other two.**

V If w would be parallel to $x_2 - x_3$. Then applying step III would result in the zero vector which is useless. **Answer for yourself:** If you would choose a w which is parallel to $x_2 - x_3$ and **use it without step III** to try to find a separating hyperplane, can you think of a case where x_1 is positioned in such a way, that w is unsuitable as a separating hyperplane ? Think of it in 2 or 3 dimensions to produce an answer.

Off the tasks: This gives you a hint, why I asked you in step III to remove the component in w parallel to $x_2 - x_3$ before solving for a bias ;-).

Offtopic: this works for any K points z_k in D dims for $K <= D$ which are linearly separable from one single point x, such that the z_k are to be classified against x.

- find a hyperplane spanning these K points z_k – for this you can consider the span of $z_2 - z_1, z_3 - z_1, \dots, z_k - z_1$.

  If $x - z_1$ lies in the hyperplane, then you cannot separate it, otherwise:

- find a vector w_2 which is orthogonal to the hyperplane ($D - K + 1$ dim space of those orthogonals, it must be orthogonal to all the $z_i - z_1$).
- use that w_2 to solve for the bias.