Deep Learning 1, Homework 1

Mathieu Bartels
11329521
UvA
mathieubartels@gmail.com

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

a

$$\Big(\frac{\partial L}{\partial x^{(N)}}\Big)_i = \frac{-t_i}{x_i^{(N)}}$$

$$\begin{aligned}\Big(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\Big)_{ij} &= \frac{\partial exp(\tilde{x}_i^{(N)})}{\partial \tilde{x}_j^{(N)}} * \frac{1}{\sum_{k=1}^{dN} exp(\tilde{x}_k^{(N)})} + \frac{\partial}{\partial \tilde{x}_j^{(N)}} \frac{1}{\sum_{k=1}^{dN} exp(\tilde{x}_k^{(N)})} * exp(\tilde{x}_i^{(N)}) \\ &= \begin{cases} \frac{exp(\tilde{x}_j^{(N)})}{\sum_{k=1}^{dN} exp(\tilde{x}_k^{(N)})} - \frac{exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{dN} exp(\tilde{x}_k^{(N)}))^2} * exp(\tilde{x}_i^{(N)}) = x_j^{(N)} - x_j^{(N)} x_i^{(N)} = x_j^{(N)}(1 - x_i^{(N)}), & \text{if i==j} \\ -\frac{exp(\tilde{x}_j^{(N)})}{(\sum_{k=1}^{dN} exp(\tilde{x}_k^{(N)}))^2} * exp(\tilde{x}_i^{(N)}) = -x_j^{(N)} x_i^{(N)}, & \text{if i!=j} \end{cases}\end{aligned}$$

$$\Big(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\Big)_{ij} = \begin{cases} 1, & \text{if i=j and } x_j > 0 \\ a, & \text{if i=j and } x_j \leq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned}\Big(\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}\Big)_{ij} &= \frac{\partial w_i^{(l)} x^{(l-1)} + b_i^{(l)}}{\partial x_j^{(l-1)}} \\ &= w_{ij} \\ \Big(\frac{\partial \tilde{x}^{(l)}}{\partial w^{(l)}}\Big)_{ijk} &= \frac{\partial w_i^{(l)} x^{(l-1)} + b_i^{(l)}}{\partial w_{jk}} \\ &= \mathbb{1}_{i=j}\, x_k \\ \Big(\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}\Big)_{ij} &= 1\end{aligned}$$

b

$$\left(\frac{\partial L}{\partial \tilde{x}^{(N)}}\right)_i = \frac{\partial L}{\partial x^{(N)}} \frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}$$
$$= \sum^{j} \frac{-t_j}{x_j^{(n)}} x_j^{(N)} (\mathbb{1}_{ij} - x_i^{(N)})$$

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} \frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \sum^{i} \frac{\partial L}{\partial \tilde{x}_i} \frac{\partial x^{(l)}}{\partial \tilde{x}_i^{(l)}}$$
$$= \frac{\partial L}{\partial \tilde{x}_i} \begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & d_n \end{pmatrix}, \text{Where } d_i = 1 \text{ if } x_i > 0 \text{ else } d_i = a$$
$$= \begin{pmatrix} \frac{\partial L}{\partial \tilde{x}_1} d_1 \\ \vdots \\ \frac{\partial L}{\partial \tilde{x}_i} d_i \\ \vdots \\ \frac{\partial L}{\partial \tilde{x}_n} d_n \end{pmatrix}$$

$$\left(\frac{\partial L}{\partial \tilde{x}^{(l<n)}}\right)_i = \sum^{j} \frac{\partial L}{\partial \tilde{x}_j^{l+1}} \frac{\partial \tilde{x}_j^{l+1}}{\partial x_i^{(l)}}$$
$$= \sum^{j} \frac{\partial L}{\partial \tilde{x}_j^{l+1}} * w_{ji}$$
$$= \frac{\partial L}{\partial \tilde{x}^{l+1}} * w_j$$

$$\left(\frac{\partial L}{\partial W^{(l)}}\right)_{ik} = \sum^{j} \frac{\partial L}{\partial \tilde{x}_j^{(l)}} \frac{\partial \tilde{x_j}^{l}}{\partial W_{ik}^{(l)}}$$
$$= \sum^{j} \frac{\partial L}{\partial \tilde{x}_j^{(l)}} \mathbb{1}_{i=j}\, x_k$$
$$\left(\frac{\partial L}{\partial W^{(l)}}\right) = \frac{\partial L}{\partial \tilde{x}^{(l)}} x$$

$$\left(\frac{\partial L}{\partial B^{(l)}}\right) = \frac{\partial L}{\partial \tilde{x}^{(l)}}$$

c Argue how the backpropagation equations derived above change if a batchsize B >1 is used.

When a batchsize >1 is chosen the dimensions change. For every derivative we get an extra batch dimension. This means we need to average the weight and the bias in the batch dimension to get a correct value of the gradient.

1.2

In Figure 1 the error and accuracy is shown for the numpy implementation. The errors are very far apart. I think this is because the error was not normalised.

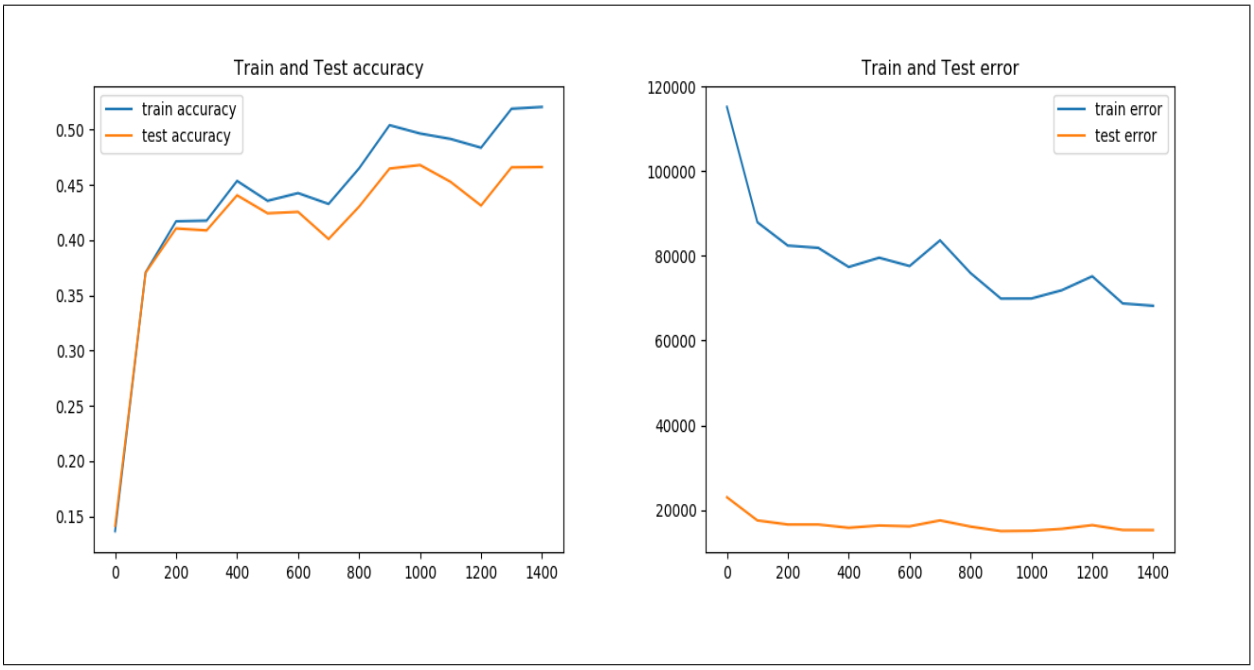

Figure 1: numpy implementation train and test, error and accuracy

2

For the implementation of the pytorch neural net I changed a few things. First lets look at the parameters.
–batch-size 500
–dnn-hidden-layers 1000,500,200,50
–learning-rate 1e-3
– max-steps = 1500
– eval-freq = 100
– neg-slope = 0.02

I lowered the learning rate, deepened the network, and increased the batch size. Because the the network is deep and the learning rate low, we need a new optimizer. The optimiser was changed to the Adam optimiser. Also the network structure was altered. Before every linear layer a dropout layer and a batch normalisation layer was added. Figure 2 shows the first signs of over fitting. It's hard to find big improvements because the parameter space is very big. And we only look at pixel level, not at regions.

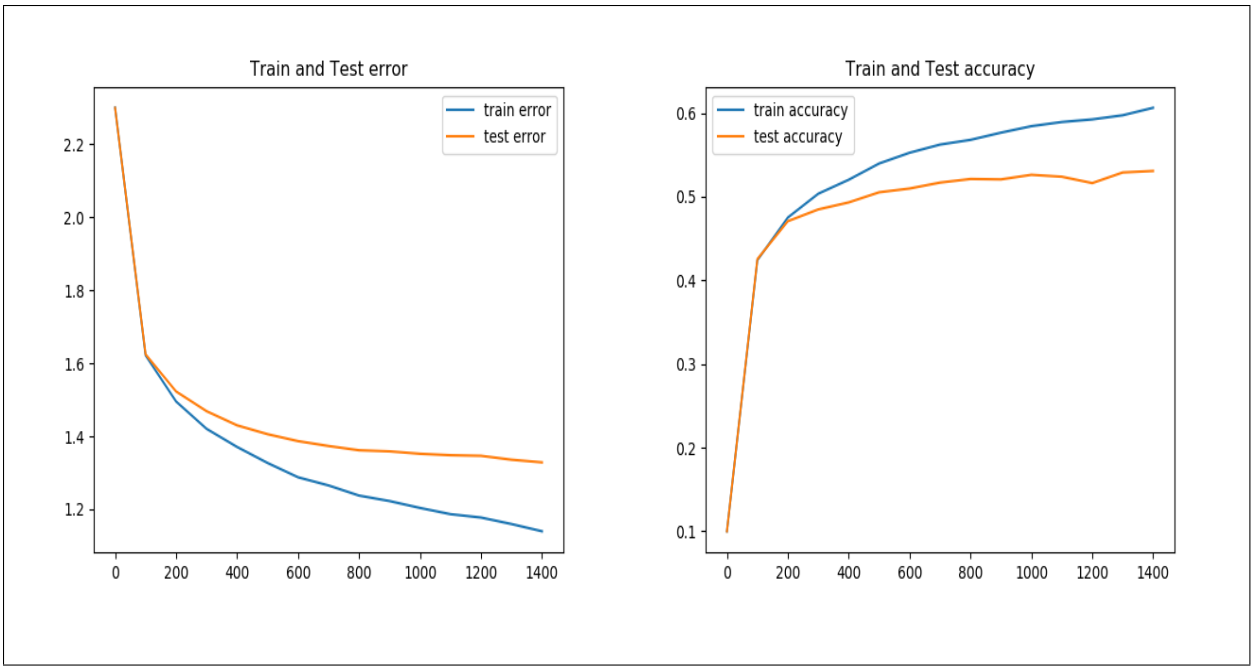

Figure 2: pytorch implementation train and test, error and accuracy

3

3.1

3.2

$$\Big(\frac{\partial L}{\partial \gamma}\Big)_j = \sum_S \sum_i \frac{\partial L}{\partial y_j^S}\frac{\partial y_j^S}{\partial \gamma_j}$$
$$= \sum^s \frac{\partial L}{\partial y_j^S}\hat{x}_j^S$$

$$\Big(\frac{\partial L}{\partial \beta}\Big)_j = \sum_S \sum_i \frac{\partial L}{\partial y_j^S}\frac{\partial y_j^S}{\partial \beta_j}$$
$$= \sum_S \frac{\partial L}{\partial y_j^S}$$

$$\frac{\partial x_i^S - \mu_i}{\partial x_j^R} = (\mathbb{I}_{\mathbb{RS}} - \frac{\mathbb{1}}{\mathbb{B}})\mathbb{I}_{ij}$$
$$\frac{\partial \sigma_i^2}{\partial x_j^R} = \frac{2}{B}(x_j^R - \mu_j)\mathbb{I}_{ij}$$
$$\frac{\partial \sqrt{\sigma_i^2 + \epsilon}}{\partial x_j^R} = \frac{(x_j^R - \mu_j)\mathbb{I}_{ij}}{\sqrt{\sigma_i^2 + \epsilon}} = \frac{1}{B}x_j^R\mathbb{I}_{ij}$$
$$\frac{\partial \hat{x}_i^S}{\partial x_j^R} = \frac{(\mathbb{I}_{\mathbb{RS}} - \frac{\mathbb{1}}{\mathbb{B}})\mathbb{I}_{ij}}{\sqrt{\sigma_i^2 + \epsilon}} * 1 - \frac{x_i^S - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} * \frac{x_j^R\mathbb{I}_{ij}}{B\sqrt{\sigma_i^2 + \epsilon}}$$
$$= \frac{(\mathbb{I}_{\mathbb{RS}}B - 1 - x_i^S x_j^R)\mathbb{I}_{ij}}{B\sqrt{\sigma_i^2 + \epsilon}}$$

$$\frac{\partial L}{\partial x}^r{}_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^S}{\partial x_j^R}$$
$$= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^S}{\partial \hat{x}_i^s}\frac{\partial \hat{x}_i^s}{\partial x_j^R}$$
$$= \sum_s \frac{\partial L}{\partial y_j^s}\frac{\partial y_j^S}{\partial \hat{x}_j^s}\frac{\partial \hat{x}_j^s}{\partial x_j^R}$$
$$= \sum_s \frac{\partial L}{\partial y_j^s}\gamma_j\frac{(\mathbb{I}_{\mathbb{RS}}B - 1 - x_i^S x_j^R)}{B\sqrt{\sigma_i^2 + \epsilon}}$$
$$= \frac{\gamma_j}{B\sqrt{\sigma_i^2 + \epsilon}}\Big(\sum_s \frac{\partial L}{\partial y_j^s}B - \sum_s \frac{\partial L}{\partial y_j^s} - x_j^R * \sum_s \frac{\partial L}{\partial y_j^s}x_i^S\Big)$$

4

In figure 3 the accuracy and error are shown of the cnn model. There's a great improvement with the previous models. We also see that the train line is more unstable, this is because I couldn't fit the whole training set on the gpu and took a 5000 images from the train set to validate. There are no signs of overfitting yet. The training took more time, and the parameter space was greatly increased, but the results show a significant improvement with the previous methods

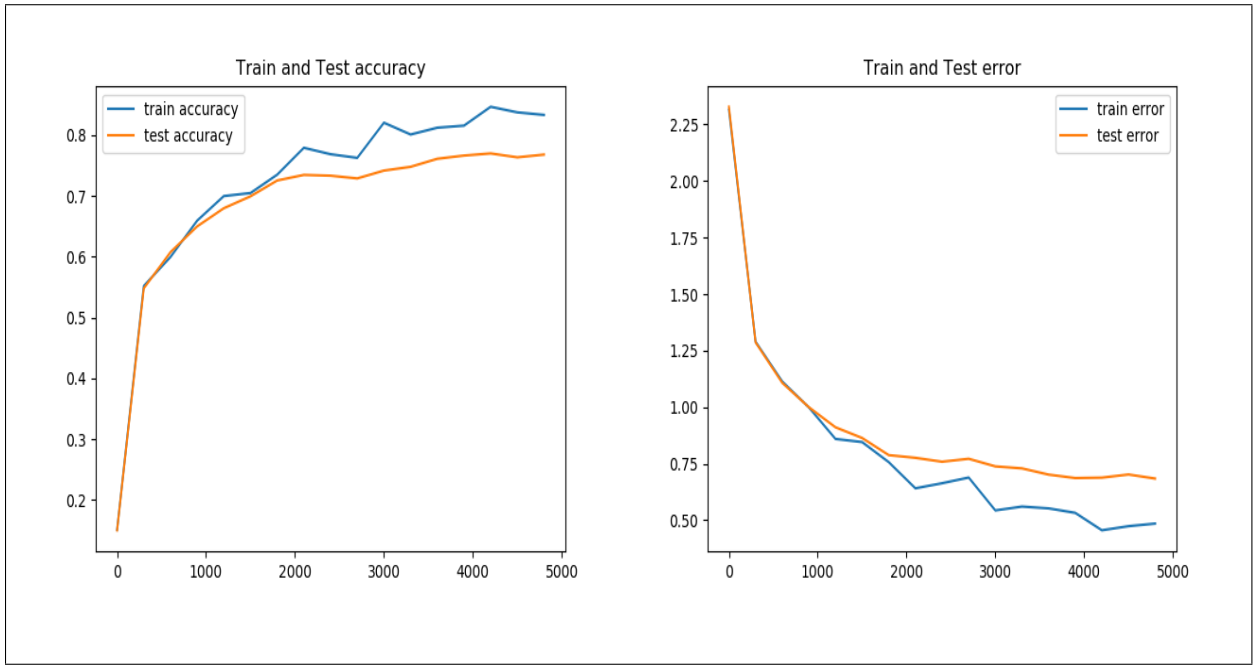

Figure 3: Cnn implementation train and test, error and accuracy