George Lydakis <lydakis@vision.rwth-aachen.de>
Stephanie Kaes <kaes@vision.rwth-aachen.de>

Exercise 4: Backprop, SoftMax

due **before** 2018-12-18

Important information regarding the exercises:

- The exercises are not mandatory. Still, we strongly encourage you to solve them! All submissions will be corrected. If you submit your solution, please read on:
- Use the Moodle system to submit your solution. You will also find your corrections there.
- Due to the large number of participants, we require you to submit your solution to Moodle **in groups of 3 to 4 students**. You can use the **Discussion Forum** on Moodle to organize groups.
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please do **not** include the data files in your submission!
- Please upload your pen & paper problems as PDF. Alternatively, you can also take pictures (.png or .jpeg) of your hand written solutions. Please make sure your handwriting is legible, the pictures are not blurred and taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.

Question 1: The softmax function ..... $(\Sigma = 4)$

Let $\mathbf{x} \in \mathbb{R}^{1\times N}$ and $\sigma(\mathbf{x}) = \text{softmax}(\mathbf{x}) = [\sigma_i(\mathbf{x})]_{i=1\dots N}$ where $\sigma_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$. In this exercise, you will compute the expressions needed for a naive implementation of the softmax for learning with backpropagation.

(a) Derive an expression for the softmax's Jacobian $D_{\mathbf{x}}\sigma(\mathbf{x}) \in \mathbb{R}^{N\times N}$ which is formulated in terms of only the softmax's output at $\mathbf{x}$. **(1 pt)**
Hint: Derive the expression for the Jacobian's diagonal entries separately from the off-diagonals.

(b) Since the `bprop` function computes the product of an incoming vector $\mathbf{v} = \begin{pmatrix} v_1 & \dots & v_N \end{pmatrix}$ (or matrix, in the batch-case) with the Jacobian: $\mathbf{z} = \mathbf{v} \cdot D_x\sigma(\mathbf{x})$, show that this product can be efficiently implemented as $z_i = \sigma_i(\mathbf{x})(v_i - \mathbf{v} \cdot \sigma(\mathbf{x})^T)$. **(1 pt)**

(c) The softmax output is usually coupled with a cross-entropy loss/criterion $\ell(\mathbf{z}, \mathbf{t}) = -\sum_{i=1}^{N} t_i \ln(z_i)$ where $\mathbf{t} \in [0, 1]^{1\times N}$ is the target/label of $\mathbf{z}$, with $\sum_{i=1}^{N} t_i = 1$. Derive an expression of the cross-entropy's Jacobian $D_{\mathbf{z}}\ell(\mathbf{z}, \mathbf{t}) \in \mathbb{R}^{1\times N}$. **(1 pt)**
Note: N is the number of classes here.

(d) What could go wrong with the cross-entropy's Jacobian? When does that happen? **(1 pt)**

Question 2: Softmax-regression with backpropagation ..... $(\Sigma = 5)$

In this exercise, you will implement a simple backpropagation framework with a `Linear`, a `SoftMax` and a `CrossEntropy` module and use it to do softmax-regression on popular benchmark-datasets such as `MNIST`, `CIFAR10` and `CIFAR100`.
Hint: Use numerical gradient-checking as explained in the exercise slides for all modules to verify and debug your implementation.
Hint: It is recommended to implement `fprop` and `bprop` in such a way that they support

batched computation, i.e. taking a matrix-valued instead of a vector-valued input. If this is too difficult, start with single-sample computation.

(a) Implement a `Linear` module with `fprop` and `bprop` functions as described in the exercise slides. **(1 pt)**
Hint: This module needs to cache its input during `fprop` and re-use it during `bprop`.

(b) Implement a `SoftMax` module with `fprop` and `bprop` functions using previously derived expressions. **(1 pt)**
Hint: Since the `SoftMax` has no parameters, there's no gradient to compute wrt. parameters.
Hint: This module needs to cache its output during `fprop` and re-use it during `bprop`.

(c) Implement a `CrossEntropy` module with `fprop` and `bprop` functions using previously derived expressions. **(1 pt)**
Hint: This module needs its target(s) to be set before `fprop` is called. It also needs to cache `fprop`'s input for re-use in `bprop`.

(d) Download the dataset for this exercise from the following URL: **(1 pt)**
`https://omnomnom.vision.rwth-aachen.de/data/mnist.tgz`.
Write a training loop as explained in the exercise slides using the following network architecture: `Linear`$(28 * 28, 10)$, `SoftMax` and the `CrossEntropy` criterion. Run the training for 100 epochs using a learning-rate $\lambda = 0.1$ and a batch-size of 600 and you should reach 750 errors on the validation-set.
Hint: Without mini-batching, you will need to reduce the learning-rate to $\lambda = 0.01$ to avoid numerical problems.
Hint: Monitor the cost.

(e) You may play with more difficult datasets by downloading `CIFAR10` and `CIFAR100` from the following URLs: **(1 pt)**
`https://omnomnom.vision.rwth-aachen.de/data/cifar10.tgz`
`https://omnomnom.vision.rwth-aachen.de/data/cifar100.tgz`

Question 3: A deeper network ..... $(\mathbf{\Sigma = 3})$

So far, you've only trained a shallow softmax-regressor. It can be turned into a deep network by stacking more modules and introducing nonlinearities, such as the `tanh` or `ReLU`.

(a) Show that the derivative of $\tanh(x)$ can be computed as $\partial_x \tanh(x) = 1 - \tanh^2(x)$. **(1 pt)**
Hint: Use the alternate definition $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

(b) Implement a `Tanh` module with `fprop` and `bprop` functions using the above expression. **(1 pt)**
Hint: This module needs to cache its output during `fprop` and re-use it during `bprop`.

(c) Train a deeper network on the `MNIST` dataset. For example, the following network architecture: `Linear`$(28 * 28, 200)$, `Tanh`, `Linear`$(200, 10)$, `SoftMax` should get you down to about 290 errors on the validation-set. **(1 pt)**
Hint: Don't forget to initialize your matrices as described in the exercise slides. What happens if you initialize them to zero?

Question 4: A more stable softmax ..... $(\mathbf{\Sigma = 5})$

You may have encountered numerical problems with your `SoftMax` implementation. In this exercise, you will learn how to implement a numerically stable version by merging the log operation from the cross-entropy computation into the `SoftMax` module.

(a) Prove the lecture's statement that $\text{softmax}(x) = \text{softmax}(x+c)$ for $c \in \mathbb{R}$. This fact can be used in the softmax implementation by subtracting $\max_{i=1\ldots N} x_i$ from the input, such that all values passed to the exp function are negative. **(1 pt)**

(b) Derive an expression for the log-softmax's Jacobian $D_{\mathbf{x}} \log(\sigma(\mathbf{x})) \in \mathbb{R}^{N \times N}$ which is formulated in terms of only the softmax's output at $\mathbf{x}$.
Hint: Derive the expression for the Jacobian's diagonal entries separately from the off-diagonals. **(1 pt)**

(c) Since the `bprop` function computes the product of an incoming vector $\mathbf{v} = \begin{pmatrix} v_1 & \ldots & v_N \end{pmatrix}$ (or matrix in the batch-case) with the Jacobian: $\mathbf{z} = \mathbf{v} \cdot D_{\mathbf{x}} \log(\sigma(\mathbf{x}))$, show that this product can be efficiently implemented as $z_i = v_i - \sigma_i(\mathbf{x}) \sum_{j=1}^{N} v_j$. **(1 pt)**

(d) The log-softmax output must now be optimized using a cross-entropy criterion adapted for log-probabilities, i.e. $\ell(\mathbf{z}, \mathbf{t}) = -\sum_{i=1}^{N} t_i z_i = -\mathbf{t} \cdot \mathbf{z}^T$. Derive an expression of this modified cross-entropy's Jacobian $D_{\mathbf{Z}} \ell(\mathbf{z}, \mathbf{t}) \in \mathbb{R}^{1 \times N}$. **(1 pt)**

(e) Implement both a `LogSoftMax` and a `CrossEntropyLog` module using the above expressions. With these modules, it is now possible to train with a mini-batch size of 1 using a larger learning-rate such as $\lambda = 0.1$. **(1 pt)**