COMPGI13/COMPM050 (Advanced Topics in ML): Assignment #2

Due on Tuesday, March 7, 2017

Thore Graepel, Koray Kavukcuoglu, Oriol Vinyals

Setup and Logistics

Submissions

- **Due date**: **7th March 2017, 11:55** (Start date: 08 Feb 2017, 11:55)
- Submit .zip via Moodle containing: **report, code, readme** (with instructions how to run the code), **trained weights** (final weights only).
- (Optional, but encouraged) If you want to use **github** to manage your code, please accept the following invitation. This will generate a private repo with your github id (make sure you are signed). Takes 3-5min to create it.
- For last task, the in-painting datasets are available here. For those using **git**, your repository will come with these automatically.
- Individual assignment.

MNIST as a sequence

In this assignment we will be using the MNIST digit dataset (http://yann.lecun.com/exdb/mnist/). – dataset containing images of hand-written digits $(0-9)$, 28×28 pixels and the corresponding labels. This is the same dataset as in Assignment 1, but we will be using this data a bit differently this time around. Since this assignment will be focusing on recurrent networks that model sequential data, we will be looking at each image as a sequence of pixels: the networks you train will be **"reading" the image one pixel at a time from top-left to bottom-right** – much-like we would process a page in a textbook, for instance.

You can use the TensorFlow build-in functionality to download and import the dataset into python.

```
from tensorflow.examples.tutorials.mnist import input_data

# import dataset with one-hot class encoding
mnist = input_data.read_data_sets(data_dir, one_hot=True)
```

Whether you choose the above, or downloading the datasets from the website, please make sure you have the following: **training set**: 55K samples, **testing set**: 10k samples.

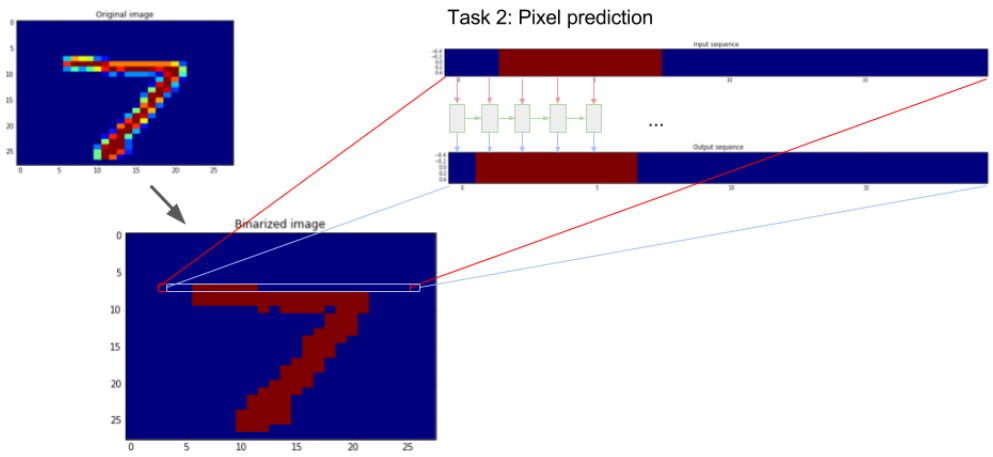

Figure 1: MNIST images as sequences. We take the MNIST images, binarise them, and interpret them as a sequence of pixels from top-left to bottom-right

For this assignment, we will work with a binarized version of MNIST – we constrain the values of the pixels to be either 0 or 1. You can do this by applying the following function to the raw images:

```
def binarize(images, threshold=0.1):
    return (threshold < images).astype('float32')
```

Recurrent Models for MNIST

As discussed in the lectures, there are various ways and tasks for which we can use recurrent models. A depiction of the most common scenarios is available in Figure 2 below. In this assignment we will look at two of these forms: **many-to-one** (sequence to label/decision) and the **many-to-many** scenario where the model receives an input and produces an output at every time step. You will use these to solve the following tasks: i) classification, ii) pixel prediction and iii) in-painting.

Understanding LSTM vs GRU (7 pts)

Before going deeper into your practical tasks, take some time to revise and make sure you understand the two major types of recurrent cells you will be using in this assignment: Long-Short Term Memory Units (LSTM) first introduced in [Hochreiter and Schmidhuber, 1997] and the more recent Gated Recurrent Units (GRU) Cho et al. [2014]. Once you have done this, answer the following questions:

(a) Can LSTMs (and respectively GRUs) just store the current input for the next step? If so, give the gates activation that would enable this behaviour? If not, explain why that cannot be done using this model. [2 pts]

(b) Can LSTMs (and respectively GRUs) store a previous state (say currently in present in c_{t-1}, respectively h_t) and ignore the current input? If so, give the gates' activation that would enable this? If not, explain why that can't be done using this model. [2 pts]

(c) Are GRUs as special case of LSTMs? If so, give the expression of the GRU gates in term of LSTM's gates(o_t, i_t, f_t). If not, give a counter-example. Assume here the same input/output. [3 pts]

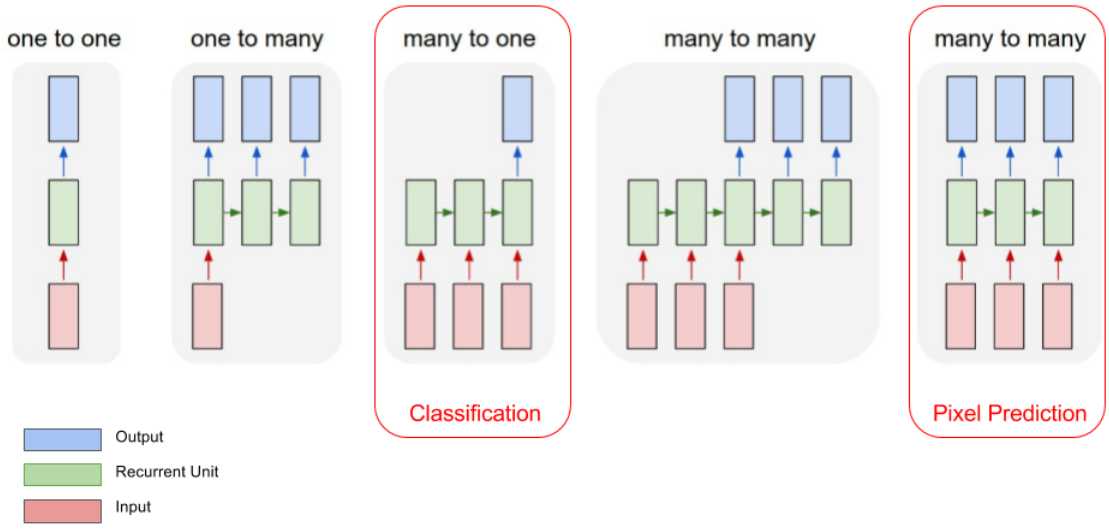

Figure 2: Major types of Recurrent Models. (Adapted from Karpathy's The Unreasonable Effectiveness of Recurrent Neural Networks – http://karpathy.github.io/2015/05/21/rnn-effectiveness/). You will be implementing variants of "many to one" for classification: sequence of pixels to classification label and "many to man" for pixel to (next) pixel predictions.

Task 1: Classification (43 pts)

In this part you will train a number of many-to-one recurrent models that takes as input: an image (or part of an image) as a sequence (pixel by pixel) and after the last input pixel produces, as output, a probability distribution over the 10 possible labels $(0-9)$. The models will be trained using a cross-entropy loss function over these output probabilities.

Optimization

You can use SGD for training, but might be worth looking into ADAM and MOMENTUM as alternatives – check TF documentation, or wait till the optimization lecture to learn more about these. Treat learning rate as a hyperparameter and optimize.
[Optional] Some time droupout has been show to be beneficial in training recurrent model, so feel free to use it or any other form of regularization that seems to improve performance.

Models

Your models will have the following structure:

(Red Block) The **input** (current binarised pixel) can be fed directly into the recurrent connection without much further pre-processing. The only thing you need to do is have an affine transformation to match the dimensionality of the recurrent unit (32, 64, 128).

(Blue Block) The **output** (probabilities over the 10 classes) is produced by looking at the last output of the recurrent units, transforming them via an affine transformation (linear layer) + non-linearity (ReLu) + affine transformation + softmax layer.

(Green Block) For the **recurrent** part of the network, please implement and compare the following architectures:

(a) LSTM with 32, 64, 128 units [15 pts]

(b) GRU with 32, 64, 128 units [15 pts]

(c) stacked LSTM: 3 recurrent layers with 32 units each [5 pts]

(d) stacked GRU: 3 recurrent layers with 32 units each [5 pts]

Results

Please report the **cross-entropy** and the **classification accuracy** of the models trained:

Model: LSTM	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layer, 32 units)
Testing Loss Training Loss				
Testing Accuracy Training Accuracy				

Model: GRU	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layer, 32 units)
Testing Loss Training Loss				
Testing Accuracy Training Accuracy				

How does this compare with the results you obtained in the first assignment, when training a model that "sees" the entire image at once? [3 pts]

Task 2: Pixel prediction (30 pts)

In this part, you will train a many-to-many recurrent model: at each time t, the model receives as input a pixel value x_t and tries to predict the next pixel in the images x_{t+1} based on the current input and the recurrent state. Thus, your output function is now a probability over the value of pixel x_{t+1} – which can be either 0 or 1 (black or white).

$$\hat{p}(x_{t+1}|x_{1:t}) = g(x_t, h_t, c_t) \tag{1}$$

Once we get to observe the actual value of x_{t+1} at the next time-step, we can compute the cross-entropy between our predicted probability $\hat{p}(x_{t+1}|x_{1:t})$ and the observed value (pixel in the image). We can (and will) do that for every time-step prediction within a sequence. This will provide us with the training signal for optimizing the parameters of the mapping g and the recurrent connections – remember these are shared!, they do not change with t.

Optimization

You can use SGD for training, but again might be worth looking into ADAM/MOMENTUM as alternatives. *[Optional] Some time droupout has been show to be beneficial in training recurrent model, so feel free to use it or any other form of regularization that seems to improve performance.*

Models

Your models will have the following structure:

(Red Block) The **input** (current pixel) can be fed directly into the recurrent connection without much further pre-processing. The only thing you need to do is have an affine transformation to match the dimensionality of the recurrent units(32, 64, 128).

(Blue Block) The **output** (probability over the colour of the next pixel black/white) is produced at every time step by looking at the current outputs of the recurrent unit, transforming them via an affine transformation (linear layer) + sigmoid layer (this produced values between 0 and 1, which are to be interpreted as the probability of the pixel being active – revise logistic regression and Thore's 2nd DL lecture, if unclear).

(Green Block) For the **recurrent** part of the network, you *can choose* if you want to use GRUs or LSTMs – but be consistent for the purpose of comparison. Please implement and compare the following architectures:

(a) GRU(or LSTM) with 32, 64, 128 units [15 pts]

(b) stacked GRU(or LSTM): 3 recurrent layers with 32 units each [5 pts]

Results

(a) Please report the **cross-entropy** of the models trained:

Model	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layer, 32 units)
Testing Loss Training Loss				

(b) Using the previously trained model, visualize the 1-step predictions, 10-step predictions, one row prediction (28 steps) and filling out the image (fill out all the pixels using the recurrent model).

Generate a small in-painting dataset. Sample 100 images from your test set. Mask/Remove the last 300 pixels (roughly 10 rows and a half).

Predict missing parts and compare with GT. Given the above generated partial sequences as input to your train models, generate the continuation of these masked images (for the next 1, 10, 28, 300 pixels). Report the cross-entropy of your in-paintings and compare this with the cross-entropy of the ground truth images. For multiple steps in-paintings, average the loss over 10 samples. [5 pts]

Visualize completing the image. Pick out 3 examples from your in-painting dataset to visualize the resulting images – this can be done at random, but should include *a successful example, failure example and one that displays high variance between samples.* Please provide 5 samples for the last three scenarios (10, 28, 300 pixels). The samples should be generated recursively by sampling the generative process provided by the trained recurrent connections. [5 pts]

Task 3: In-painting (20 pts)

Using the models trained in the previous section, please in-paint the missing pixels in the following datasets:

- One-pixel missing
- Window of 2x2 pixels missing

This is similar to Task 2b, but now you have information not only about the past(previous pixels in the image) but also future (pixels that come after your predictive target)

Results

(a) Provide the formula used to compute the probability over the missing pixel (missing patch) [2x5 pts]

(b) Visualize the most probable in-painting, according to your model. How does this compare to the ground truth? (Compare cross-entropy between your most probable sample and the ground truth) [2x5 pts]