Advanced Topics in ML - Assignment 2

Benoit Gaujac, SN: 16100324

March 2017

1 Understanding LSTM vs GRU

LSTM cells are defined by the following set of equations:

$$\begin{aligned}
&(1) \quad f_t = \sigma\big(w_f \cdot [h_{t-1}, x_t] + b_f\big) \\
&(2) \quad i_t = \sigma\big(w_i \cdot [h_{t-1}, x_t] + b_i\big) \\
&(3) \quad \tilde{c}_t = tanh\big(w_c \cdot [h_{t-1}, x_t] + b_c\big) \\
&(4) \quad o_t = \sigma\big(w_o \cdot [h_{t-1}, x_t] + b_o\big) \\
&(5) \quad c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \\
&(6) \quad h_t = o_t * tanh(c_t)
\end{aligned}$$

where $\cdot$ designs the dot product between matrix and $*$ the elements wise product. Equations (1)(2) correspond to the forget gate, equations (3)(4) to the input gate and equation (4) to the output gate. Equation (5) gives the internal state c_t at step time t and (6) the output h_t at time step t.
GRU cells are a variation of the LSTM cells. They merge their output h_t and state c_t into one unique vector and combine their forget and input gates. They are defined by the following equations:

$$\begin{aligned}
&(7) \quad z_t = \sigma\big(w_z \cdot [h_{t-1}, x_t] + b_z\big) \\
&(8) \quad r_t = \sigma\big(w_r \cdot [h_{t-1}, x_t] + b_r\big) \\
&(9) \quad \tilde{h}_t = tanh\big(w \cdot [r_t * h_{t-1}, x_t] + b\big) \\
&(10) \quad h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}$$

1.1 Question a)

In the case of the LSTM cells, if we want to have $c_t = x_t, \forall t$ we need, given (6):

$$\begin{aligned}
&f_t = 0 \quad \forall(h_{t-1}, x_t) \quad (a), \\
&\tilde{c}_t = x_t \quad \forall(h_{t-1}, x_t) \quad (b), \\
&i_t = 1 \quad \forall(h_{t-1}, x_t) \quad (c)
\end{aligned}$$

That is, activate the forget gate ($f_t = 0$) to forget everything from previous state, and set the input gate to 1 in order to store only the input. Conditions (a) and (c) are relatively straightforward given (1) and (2), taking $w_f = 0, w_i = 0$ and $b_f \to -\infty, b_i \to +\infty$.

Given (5) condition (b) is satisfied if:

$$\begin{aligned}&w_c \cdot [h_{t-1}, x_t] + b_c \to +\infty \quad \text{when } x_t = 1 \quad \forall h_{t-1}\\ &w_c \cdot [h_{t-1}, x_t] + b_c = 0 \quad \text{when } x_t = 0 \quad \forall h_{t-1}\end{aligned}$$

First, let's fix the notations. We call n_x respectively n_h the second dimension of x_t respectively h_{t-1}: $x_t \in \mathcal{R}^{batchsize,n_x}$ and $h_t \in \mathcal{R}^{batchsize,n_h}$. Here we have $n_x = n_h = 1$. Then we call *nunits* the size of the second dimension of w_c: $w_c \in \mathcal{R}^{n_h+n_x,nunits}$ and $b_c \in \mathcal{R}^{batchsize,nunits}$ To satisfy the condition, we can set $b_c = 0$ and take w_c as follow. We have $\forall(i,j) \in [1, batchsize] \times [1, nunits]$:

$$\begin{aligned}\big(w_c \cdot [h_{t-1}, x_t]\big)[i,j] &= \big([h_{t-1}, x_t]w_c^\top\big)[i,j]\\ &= \sum_{k=1}^{n_h} h_{t-1}[i,k]w_c[k,j] + \sum_{k=1}^{n_x} x_t[i,k]w_c[n_h+k,j]\\ &= h_{t-1}[i,1]w_c[1,j] + x_t[i,1]w_c[2,j]\end{aligned}$$

and thus:

$$\begin{aligned}\Big(w_c \cdot [h_{t-1}, x_t]\Big)[i] =& \Big(h_{t-1}[i,1]w_c[1,1] + x_t[i,1]w_c[2,1][1],\\ &\ldots, h_{t-1}[i,1]w_c[1, nunits] + x_t[i,1]w_c[2, nunits]\Big)\end{aligned}$$

Thus, if we choose $\forall k \in [1, nunits]$, $w_c[1,k] = 0$ and $w_c[2,k] \to +\infty$, we have:

$$\begin{aligned}\Big(w_c \cdot [h_{t-1}, x_t]\Big)[i] &= \Big(+\infty, \ldots, +\infty\Big) \quad \text{if } x_t = 1\\ \Big(w_c \cdot [h_{t-1}, x_t]\Big)[i] &= \Big(0, \ldots, 0\Big) \quad \text{if } x_t = 1\end{aligned}$$

and so we have the desired behaviour for the input gate. And thus, the LSTM is effectively able to *store* the current input for the next step

In the case of the GRU cells, given (10), to have $h_t = x_t, \forall t$, we need:

$$\begin{aligned}&z_t = 1 \quad \forall(h_{t-1}, x_t) \quad (d)\\ &\text{and}\\ &\tilde{h}_t = x_t \quad \forall(h_{t-1}, x_t) \quad (e)\end{aligned}$$

While condition (e) is relatively easy to achieve, either by fixing the reset gate to 0, $r_t = 0$, by setting $w_r = 0$, or by setting the columns of w corresponding to the matrix multiplication with $r_t * h_{t-1}$ to zero, with in both case $b = 0$. For condition (d) we can set $w_z = 0$ and $b_z \to +\infty$.

$$w_z \cdot [h_{t-1}, x_t] + b_z \to +\infty \quad \forall(h_{t-1}, x_t)$$

and thus,

$$z_t \to 1 \quad \forall(h_{t-1}, x_t)$$

1.2 Question b)

We now look at the state. We want to know if it is possible store a previous state, and at the limit, to have $c_t = c_{t-1}$ (respectively $h_t = h_{t-1}$) with LSTM cells (respectively GRU cells).

In the case of the LSTM cells, we can do this by deactivate the input gate ($i_t = 0$) to block the input, and set the forget gate ($f_t = 1$) so as to let flow all the information from previous state. Using (2), $\tilde{i_t} = 0 \; \forall (h_{t-1}, x_t)$ if $w_i = 0$ and $b_i \rightarrow -\infty$. And given (1), if we take $w_f = 0$ and $b_f \rightarrow \infty$, we have $f_t = 1 \; \forall (h_{t-1}, x_t)$, and thus $c_{t-1} = c_t$.

For the GRU cells, using (10), $h_t = h_{t-1}$ if $z_t = 0$. We can achieve this by setting $w_z = 0$ and this time, $b_z \rightarrow -\infty$.

1.3 Question c)

GRU and LSTM are not equivalents, neither GRU are special case of LSTM. There are differences in their structures that set them appart.
First, in the LSTM, the output gate module the exposure and the transmission of the internal state to the next time-step. This is defined in equation (6) above where the resulting internal state transmitted from time step t to t+1, h_t is controlled by the output gate o_t. GRU cells on the contrary do not have any control on the internal state it will transmit from one time step to another as it is shown equation (10). Moreover, LSTM have an additional *tanh* non-linearity acting on the internal state which is not present in the GRU cells.
Secondly, LSTM and GRU present differences in their input and reset gates. Instead of controlling the amount of information flowing from previous time step in its input gate in equation (2), LSTM control the amount of new memory to be added, second term in the right side of (5), independently of the content it will forget from the previous step time via the forget gate, equation (2) and first term in the right side of (5). GRU on other hand, control the amount of internal seen from previous time step, element-wise product $r_t * h_{t-1}$ in (9), and tied to it the amount of new memory to be added via the update gate z_t, equation (10).

2 Task 1: Classification

We build the 8 different models using *tensorflow*. As the training process is relatively long and computationally demanding, we trained some of the models with AWS ressources (GPU or multiple CPUs).

For the optimizer, we used in all the trainings an Adam optimizer for the cross-entropy loss. We used also dropout with different probabilities. The different set up have been chosen after a roughly optimization over the hyperparameters (dropout probability, initial learning rate, learning rate decay, etc.). However, due to the very limited resources and the very high computational and time costs of the models trainings, this optimization is only approximate and coarse. Moreover, to gain some time, we perform evaluation on validation data only one every two epochs.

2.1 LSTM Cells

The first model is the 1 layer 32 units RNN with LSTM cell. The training process was the following: we trained the network over 100 epochs, with early stopping, using output dropouts for the LSTM cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $1.00e^{-3}$, a decay step of 0.87 every 10 epochs. The resulting learning curve is given figure 1.a) and table 1.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.8758	0.81	68.21	70.21

Table 1: Performances for the 1 layer 32 units LSTM

The second model is the 1 layer 64 units RNN with LSTM cell. We trained the network as follow: 100 epochs, with early stopping, using output dropouts for the LSTM cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $3.00e^{-3}$, a decay step of 0.9 every 10 epochs. The resulting learning curve is given figure 1.b) and table 2.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.4937	0.4796	82.53	82.65

Table 2: Performances for the 1 layer 64 units LSTM

The next model is the 1 layer 128 units RNN with LSTM cell. We trained the network as follow: 100 epochs, with early stopping, using output dropouts for the LSTM cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $0.75e^{-3}$, a decay step of 0.9 every 10 epochs. The resulting learning curve is given figure 1.c) and table 3.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.2325	0.2202	93.05	93.53

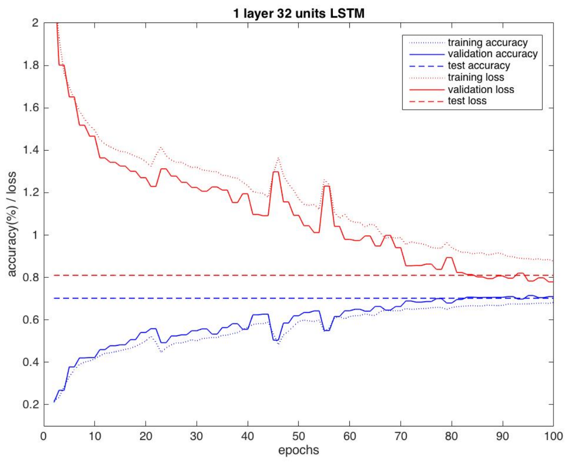
Table 3: Performances for the 1 layer 128 units LSTM

The last model with LSTM cell is the 3 layers 32 units RNN. We trained the network over 100 epochs, with early stopping, using output dropouts for the LSTM cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $2.00e^{-3}$, a decay step of 0.9 every 7 epochs. The resulting learning curve is given figure 1.d) and table 4.

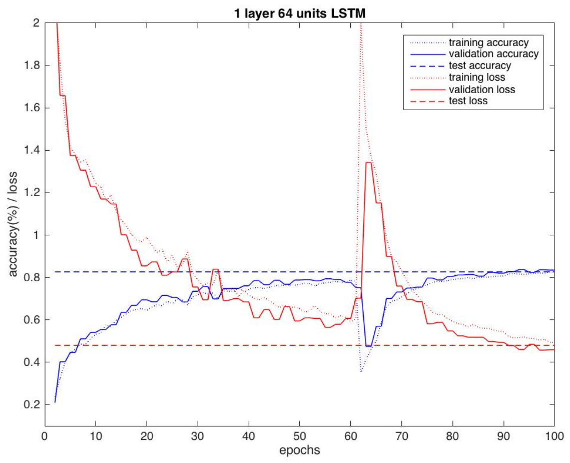
Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.5208	0.4738	81.23	82.21

Table 4: Performances for the 3 layer 32 units LSTM

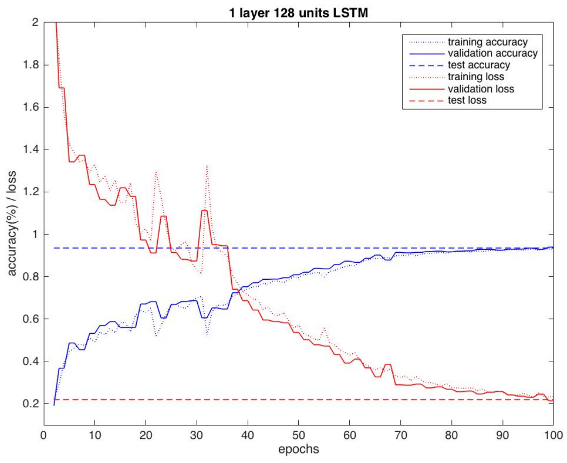
Figure 2, we plotted the learning curves of the 4 previous models using LSTM. On this figure, it looks like the best performing model is the 1 layer with 128 units model. However, it is difficult to compare all these models as different training processes have been used (and especially different combination of initial learning rate and rate decay). Thus performances described here are not the most relevant for the comparison of the performances of the



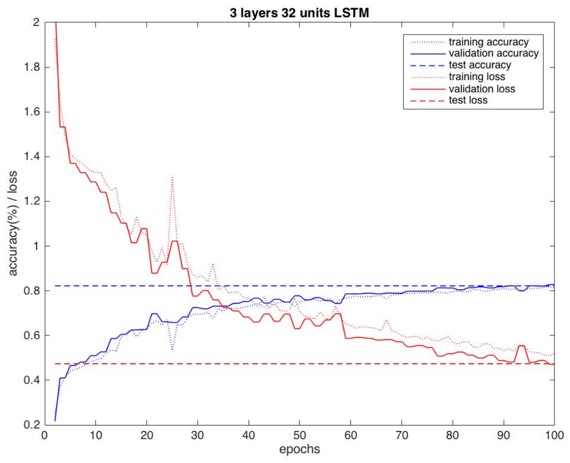
(a) Learning curve for LSTM 1 layer 32 units



(b) Learning curve for LSTM 1 layer 64 units



(c) Learning curve for LSTM 1 layer 128 units



(d) Learning curves for LSTM 3 layers 32 units

Figure 1: Learning curves for LSTM cells

models. However, they give us a very approximate picture of the learning process of the different models.

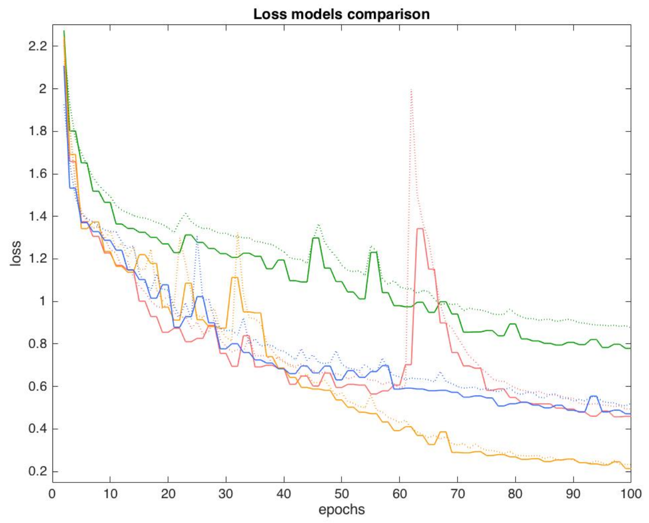


Figure 2: Training loss curve for the 4 models with LSTM. the dot line represents the training loss while the plain line the validation loss. Green: 1layer 32units, Red: 1layer 64units, Orange: 1layer 128units, Blue: 3layers 32units

2.2 GRU Cells

The first model is the 1 layer 32 units RNN with GRU cell. The training process was the following: we trained the network over 100 epochs, with early stopping, using output dropouts for the GRU cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $5.00e^{-3}$, a decay step of 0.87 every 10 epochs. The resulting learning curve is given figure 3.a) and table 5.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.0607	0.0872	98.24	97.47

Table 5: Performances for the 1 layer 32 units GRU

The second model is the 1 layer 64 units RNN with GRU cell. We trained the network as follow: 100 epochs, with early stopping, using output dropouts for the GRU cell with a probability to keep the output being set to 0.5. We use an exponential decay learning rate, with initial learning rate set to $5.00e^{-3}$, a decay step of 0.9 every 10 epochs. The resulting learning curve is given figure 3.b) and table 6.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.1121	0.1235	96.89	96.47

Table 6: Performances for the 1 layer 64 units GRU

The next model is the 1 layer 128 units RNN with GRU cell. We trained the network as follow: 100 epochs, with early stopping, using output dropouts for the GRU cell with a probability to keep the output being set to 0.75. We use an exponential decay learning rate, with initial learning rate set to $1.00e^{-3}$, a decay step of 0.9 every 10 epochs. The resulting learning curve is given figure 3.c) and table 7.

Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.0459	0.0782	98.66	98.00

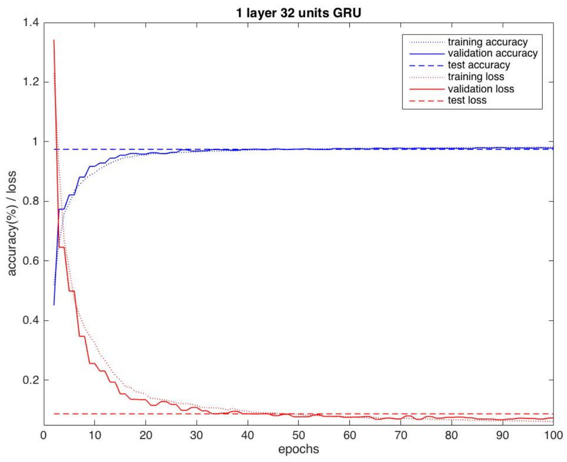
Table 7: Performances for the 1 layer 128 units GRU

The last model is the 3 layers 32 units RNN with GRU cells. We trained the network over 80 epochs (trained on the several CPU with AWS), with early stopping, using output dropouts for the GRU cell with a probability to keep the output being set to 0.5. We use an exponential decay learning rate, with initial learning rate set to $5.00e^{-3}$, a decay step of 0.9 every 7 epochs. The resulting learning curve is given figure 3.d) and table 8.

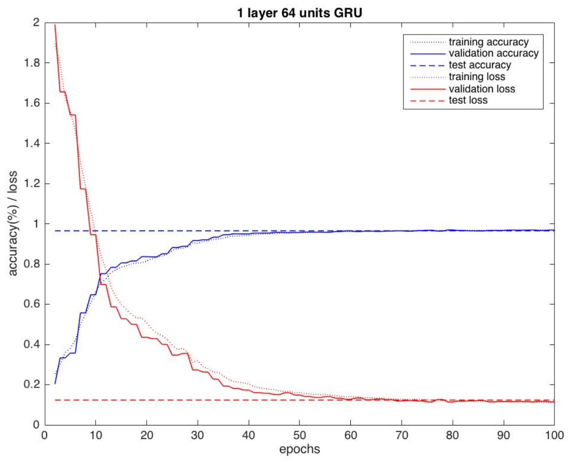
Training loss	Testing loss	Training accuracy (%)	Testing accuracy (%)
0.0231	0.0709	99.39	98.52

Table 8: Performances for the 3 layer 32 units GRU

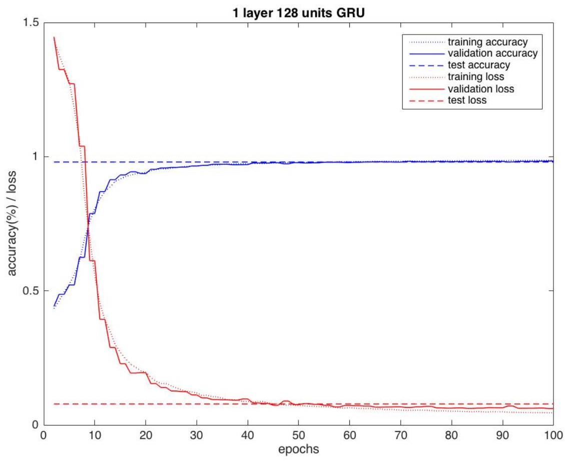
Once again, figure 10, we can plot the learning curves of the 4 previous models for comparison purpose. One more time, while this figure is good to give us an approximate view on the training process, it is difficult to draw a real comparison of the performances of the different models. However, we see clearly that the training of the GRU models is much faster and much stable that the one of their LSTM counterparts.



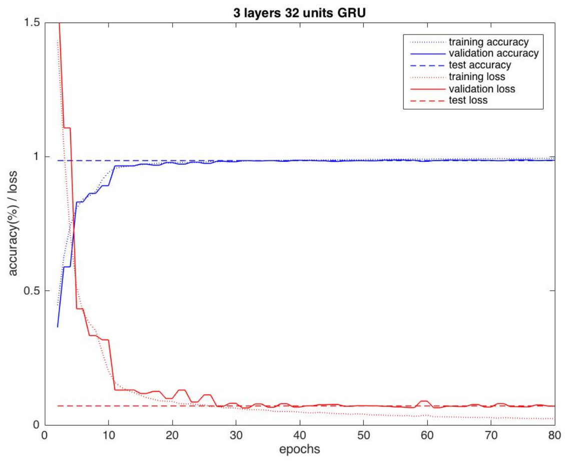
(a) Learning curve for GRU 32 1 layer 32 units



(b) Learning curve for GRU 1 layer 64 units



(c) Learning curve for GRU 1 layer 128 units



(d) Learning curves for GRU 3 layers 32 units

Figure 3: Learning curves for GRU cells

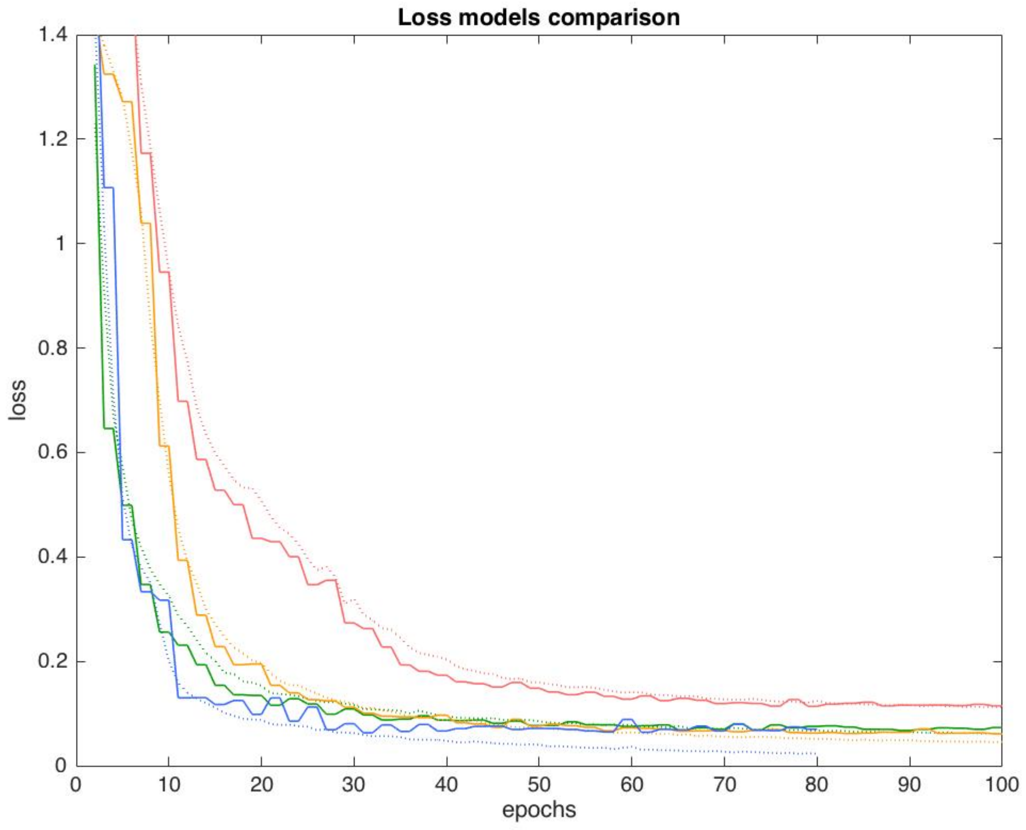


Figure 4: Training loss curve for the 4 models with GRU. the dot line represents the training loss while the plain line the validation loss. Green: 1layer 32units, Red: 1layer 64units, Orange: 1layer 128units, Blue: 3layers 32units

2.3 Performances summary

Table 9 & 10 summarizes the performances of the different models.

Model: LSTM	1layer32units	1layer64units	1layer128units	3layers32units
Testing loss	0.81	0.4796	0.2202	0.4738
Training loss	0.8758	0.4937	0.2325	0.5208
Testing accuracy	70.21	82.65	93.53	82.21
Training accuracy	68.21	82.53	93.05	81.23

Table 9: Performances for networks with LSTM cells

Model: GRU	1layer32units	1layer64units	1layer128units	3layers32units
Testing loss	0.0872	0.1235	0.0782	0.0709
Training loss	0.0607	0.1121	0.0459	0.0231
Testing accuracy	97.47	96.47	98.00	98.52
Training accuracy	98.24	96.89	98.66	99.39

Table 10: Performances for networks with GRU cells

Comparing with the results obtained in assignment 1, we see that these RNN are slightly below the performances obtained with the deep fully connected layers and CNN networks. More precisely, while the GRU cells offer performances relatively close and similar than the CNN in assignment 1, LSTM cells are much less performing, with a training less stable and longer. Another big difference is computational and time cost. The RNN networks are much longer and require much more computational capacities than their CNN counterpart.

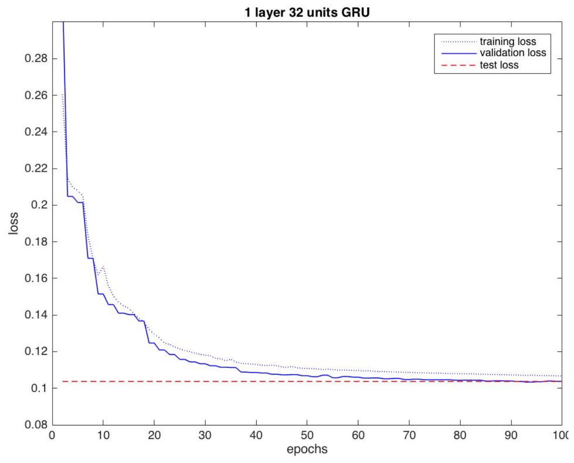
3 Task 2: Pixel prediction

3.1 a) models training

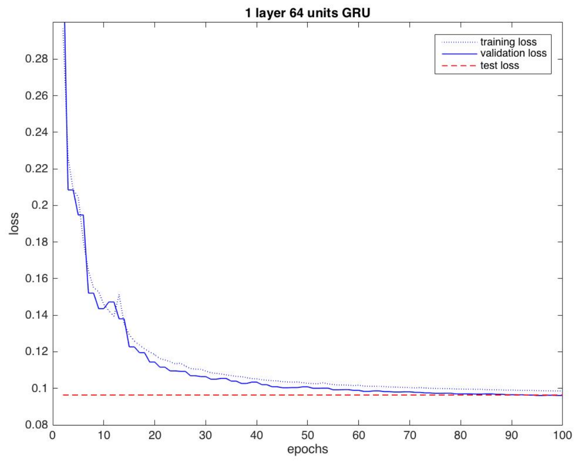
We trained the 4 different models using GRU cells as they offer much better performances over their LSTM counterpart (see previous section). As in task 1, we used different training setting with different hyperparameters for the models chosen after a very quick optimization. However, for all the four model, we used the same Adam optimizer on the cross-entropy between our predicted probability $\hat{p}(x_{t+1}|x_{1:t})$ and the observed value (pixel in the image) for all the sequence trained over 100 epochs. We also use dropouts with probability 0.75, exponential decay learning rate with different initial learning but with fixed rate and step decay (0.90 decay every 5 epochs) and early termination (evaluation on the dev set every two other epochs).
For the 1 layer 32 units and 3 layers 32 units models with GRU cells, we start the training wth an intial learning rate of $10.00e^{-3}$. For the 1 layer 64 units and 128 units, we took an intial learning rate of $10.00e^{-3}$. The results after training are given table 11. We plot the different learning curves for the four models figure 5, and figure 6, we plot the same curves but in the same graph for easier comparison.

Model: GRU	1layer32units	1layer64units	1layer128units	3layers32units
Testing loss	0.1037	0.0963	0.0819	0.0906
Validation loss	0.1038	0.0961	0.0816	0.0932
Training loss	0.1067	0.0985	0.0829	0.0983

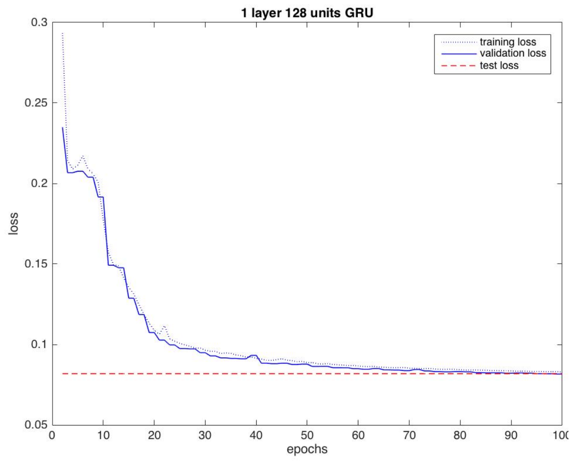
Table 11: Performances with GRU cells



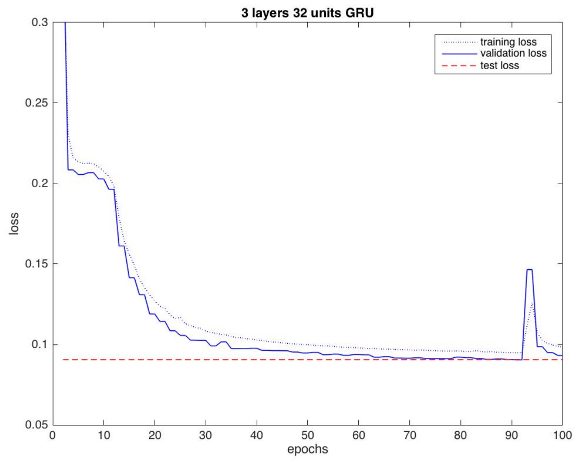
(a) Learning curve for GRU 1 layer 32 units



(b) Learning curve for GRU 1 layer 64 units



(c) Learning curve for GRU 1 layer 128 units



(d) Learning curve for GRU 3 layers 32 units

Figure 5: Loss during training

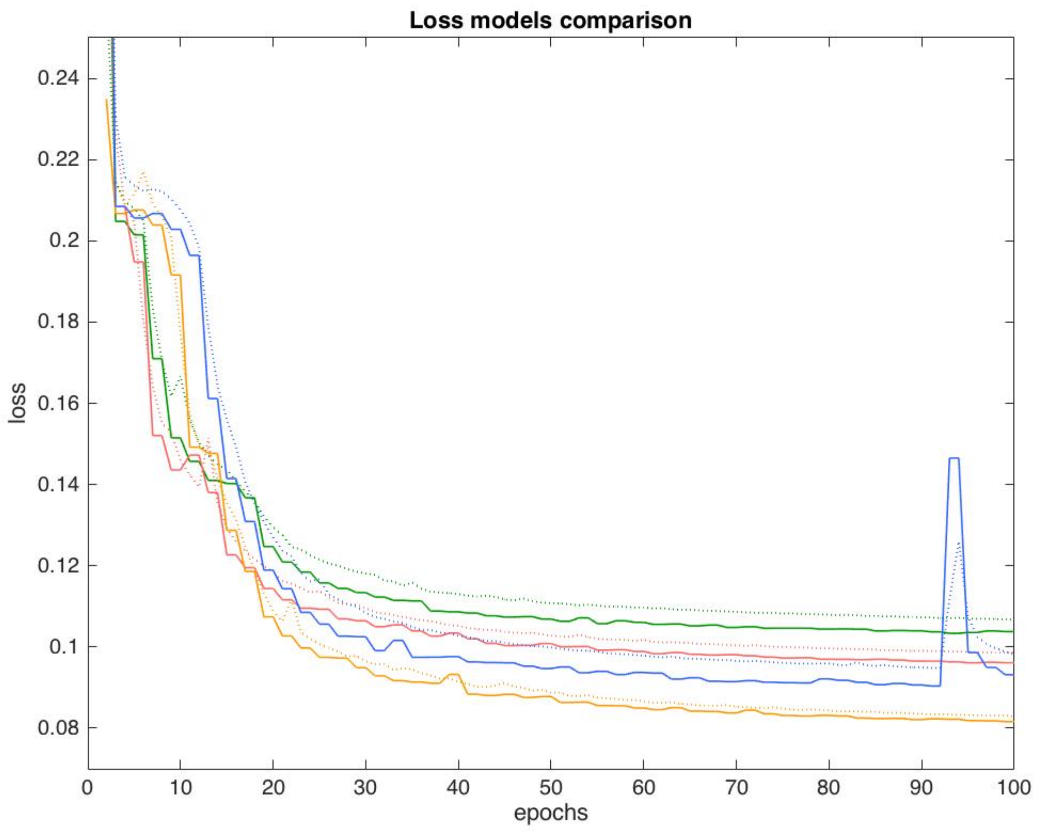


Figure 6: Loss vs epochs during training for the 4 models. the dot line represents the training loss while the plain line the validation loss. Green: 1layer 32units, Red: 1layer 64units, Orange: 1layer 128units, Blue: 3layers 32units

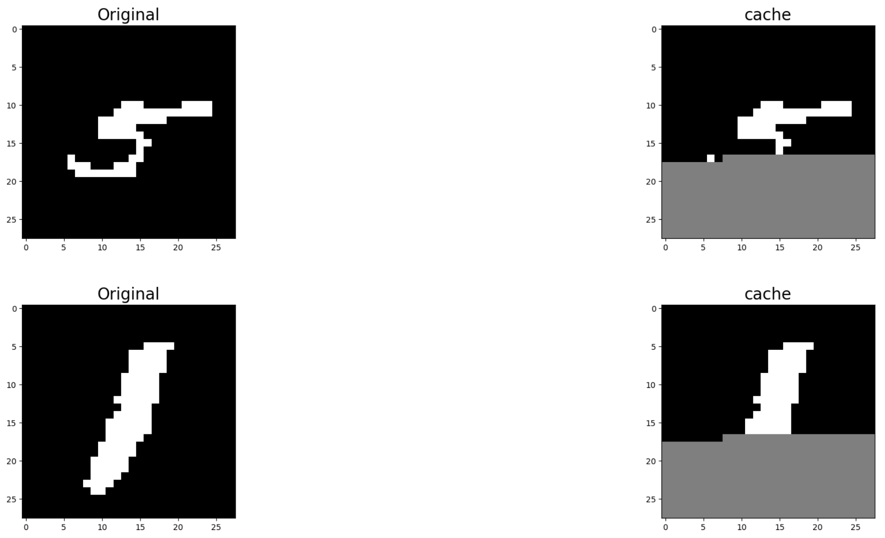


Figure 7: Examples of cache data. Right: Original (binary) picture. Left: Cache data, grey pixels correspond to hidden pixels

3.2 b) In-painting

We first sampled randomly 100 images and created our in painting set composed of the cache images. Figure 7 shows examples of such cache images.

Given this data set of partially hidden sequences, we in-painted the cache images for the next n pixels, with $n \in [1, 20, 28, 300]$. We report table 12 the average cross-entropy over the generated sequences as well as the ground truth entropy for each of the four models. More especially, for the each of the 100 cache images, we generated 10 continuation sequences for the different values of n. Figures 1 to 4 Annex 1.1 show the average cross-entropy over the 10 generated sequences for each of the 100 cache images. Table 12 reports the averaged cross-entropy over the 100 images for each model. (plain line in the figures 1 to 4 of Annex 1.1)

Model	n=1	n=10	n=28	n=300
Predicted 1layer32units	0.1557	0.2178	0.1281	0.1047
GT 1layer32units	0.1228	0.3628	0.2040	0.5615
Predicted 1layer64units	0.1693	0.2243	0.1252	0.0860
GT 1layer64units	0.1590	0.4034	0.2280	0.5426
Predicted 1layer128units	0.1517	0.1956	0.1040	0.0627
GT 1layer128units	0.2354	0.2796	0.1569	0.3534
Predicted 3layers32units	0.1354	0.2004	0.1182	0.0821
GT 3layers32units	0.0964	0.3011	0.1803	0.5048

Table 12: Average cross-entropy for the continuation sequences

We visualize the in-painting images in the figures given Annex 1.2. For each model, for each value of $n \in [1, 10, 28, 300]$, we show the in-painting samples of 3 images (common to all the models). For each of these images, we generate 5 in-painting samples as well as the most probable in-painting sampled as follow, $\forall t \in [1, 300]$:

$$\begin{aligned} x_t &= 1 \quad \text{if } p_t > 0.5 \\ x_t &= 0 \quad \text{otherwise} \end{aligned}$$

For each image example in Annex 1.2, the first image of the set is the original one, the second one is the cache image and the third one is the most probable in-painting.

4 Task 3: In-painting

4.1 a) missing pixel probability

In the case of the one pixel missing, x_i, for $i \in [1, \dots, 28 \times 28]$ (we note $T = 28 \times 28$), we have:

$$\begin{aligned} p(x_i | x_{1:i-1}, x_{i+1:T}) &= \frac{p(x_i, x_{1:i-1}, x_{i+1:T})}{p(x_{1:i-1}, x_{i+1:T})} \\ &= \frac{p(x_1) \prod_{k=1}^{T} p(x_{k+1} | x_{1:k})}{p(x_1) p(x_{i+1:T\,1:i-1}) \prod_{k=1}^{i-1} p(x_{k+1} | x_{1:k})} \\ &= \frac{\prod_{k=i}^{T} p(x_{k+1} | x_{1:k})}{p(x_{i+1:T\,1:i-1})} \\ &\propto \prod_{k=i}^{T} p(x_{k+1} | x_{1:k}) \end{aligned}$$

where we omitted all the terms not depending on x_i in the last line.

For the 2×2 pixels blocks missing data set, we have: $(i, k) \in [1, \dots, 28 \times 28]^2, k \geq i + 2$, we have:

$$\begin{aligned} p(x_i, x_{i+1}, x_k, x_{k+1} | x_{1:T}^{\neq(i,i+1,k,k+1)}) &= \frac{p(x_1^{i-1}) p(x_i, x_{i+1} | x_1^{i-1}) p(x_{i+2}^{k-1} | x_1^{i+1}) p(x_k, x_{k+1} | x_1^{k-1}) p(x_{k+2}^{T} | x_1^{k+1})}{p(x_1^{i-1}) p(x_{i+2}^{k-1} | x_1^{i-1}) p(x_{k+2}^{T} | x_1^{i-1}, x_{i+2}^{k-1})} \\ &\propto p(x_i, x_{i+1} | x_1^{i-1}) p(x_{i+2}^{k-1} | x_1^{i+1}) p(x_k, x_{k+1} | x_1^{k-1}) p(x_{k+2}^{T} | x_1^{k+1}) \\ &= \prod_{k=i}^{T} p(x_{k+1} | x_{1:k}) \end{aligned}$$

where we omitted all the terms not depending on x_i, x_{i+1}, x_k and x_{k+1} in line 2.

As we saw in the previous expressions, the probability is proportional to the truncated log-likelihood of the whole sequence. We can show that in the case of binary data (*i.e* with two classes 0 and 1), the log-likelihood is equal to the opposite of the cross-entropy.
First lets set up the notations. We will call p the *true* distribution of the pixel value. Under p, e have, $\forall t \in [1, T]$ where $T = imagesize \times imagesize$:

$$\begin{aligned} P_p(x_t = 1) &= x_t \\ P_p(x_t = 0) &= 1 - P(x_t = 1) \\ &= 1 - x_t \end{aligned}$$

Now we call q the learned distribution of the pixel value. Under q, with the notation of the assignment, we have, $\forall t \in [2, T]$:

$$\begin{aligned}
P_q(x_t = 1) &= P_{\hat{p}}(x_t = 1|x_{1:t-1}) \\
&= \sigma\big(f(x_{1:t-1})\big) \\
P_q(x_t = 0) &= 1 - \sigma\big(f(x_{1:t-1})\big)
\end{aligned}$$

that we can express in term of the distribution q:

$$\begin{aligned}
q(x_t) &= \hat{p}(x_t|x_{1:t-1}) \\
&= \Big(\sigma\big(f(x_{1:t-1})\big)\Big)^{x_t}\Big(1 - \sigma\big(f(x_{1:t-1})\big)\Big)^{1-x_t}
\end{aligned}$$

where σ denotes the sigmoid function and f is the output of the RNN (after linear operation to get the logits from the n_{units} ouput vectors of the RNN).
The cross entropy of the sequence for our model is:

$$\begin{aligned}
H(p, q) &= -\mathbb{E}_p[\log(q)] \\
&= -\sum_t P_p(x_t = 1)\log(P_q(x_t = 1)) + P_p(x_t = 0)\log(P_q(x_t = 0)) \\
&= -\sum_t P_p(x_t = 1)\log(P_q(x_t = 1)) + (1 - P_p(x_t = 1))\log((1 - P_q(x_t = 0))) \\
&= -\sum_t x_t \log(\sigma\big(f(x_{1:t-1})\big)) + (1 - x_t)\log(1 - \sigma\big(f(x_{1:t-1})\big)) \\
&= -\sum_t \log(\Big(\sigma\big(f(x_{1:t-1})\big)\Big)^{x_t}) + \log(\Big(1 - \sigma\big(f(x_{1:t-1})\big)\Big)^{1-x_t}) \\
&= -\sum_t \log(\Big(\sigma\big(f(x_{1:t-1})\big)\Big)^{x_t}\Big(1 - \sigma\big(f(x_{1:t-1})\big)\Big)^{1-x_t}) \\
&= -log\Big(\prod_t \left(\sigma(f(x_{1:t-1}))\right)^{x_t}\left(1 - \sigma(f(x_{1:t-1}))\right)^{1-x_t}\Big) \\
&= -\log\big(\prod_t q(x_t)\big) \\
&= -\log\big(q(x_{1:t})\big)
\end{aligned}$$

and we recognize the log-likelihood of the data under q.

4.2 b) missing pixel visualization

Using the work of previous sections (trained models and result on the cross-entropy), we can now in-paint the missing pixels using all the information available (future pixels). Figure 16 and 17 visualizes the in-painting for the *one pixels* missing and figure 18 and 19 for the *2x2 pixels* missing data set. Table 13 compares the average cross entropy between the in-paintings and the ground truth in both cases.

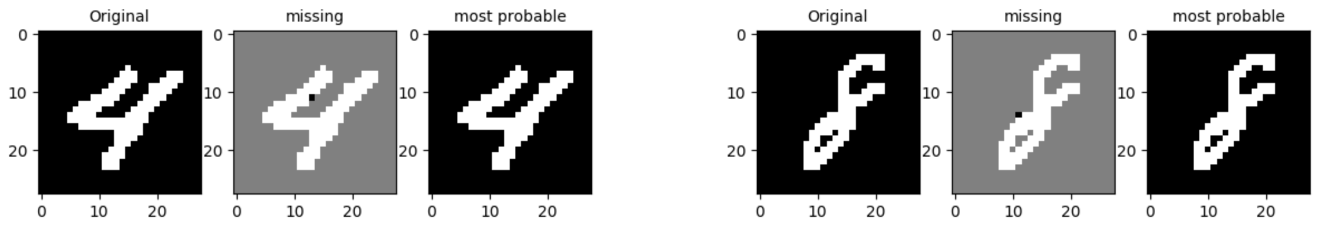


Figure 8: Succesful In-painting for the one missing pixels data set. Left image: original, middle image: missing pixels, right: in-painting

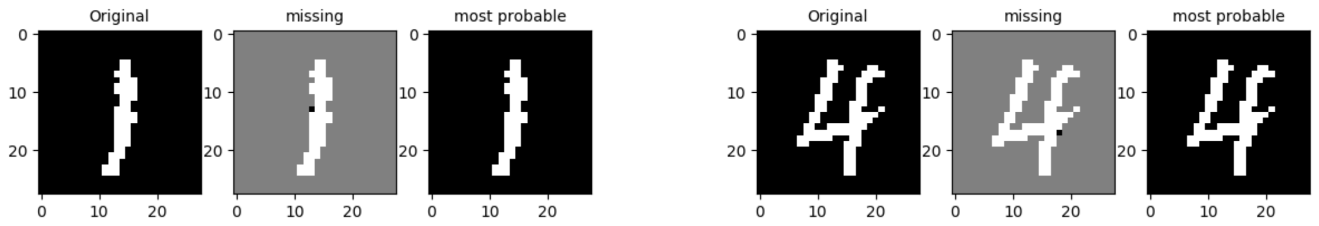


Figure 9: Different from original In-painting for the one missing pixels data set. Left image: original, middle image: missing pixels, right: in-painting

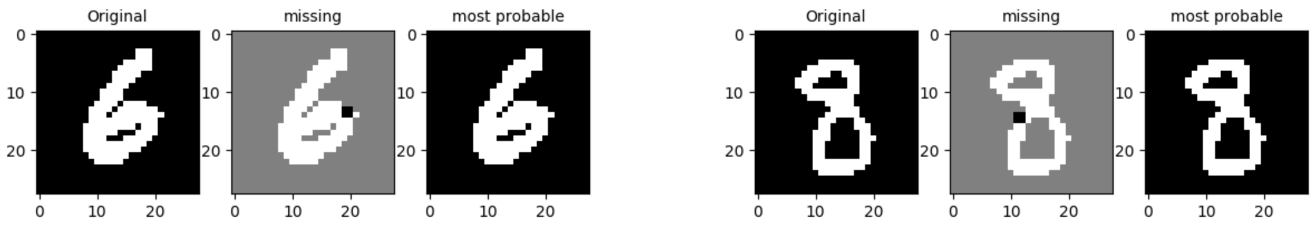


Figure 10: Succesful In-painting for the 2 × 2 missing pixels data set. Left image: original, middle image: missing pixels, right: in-painting

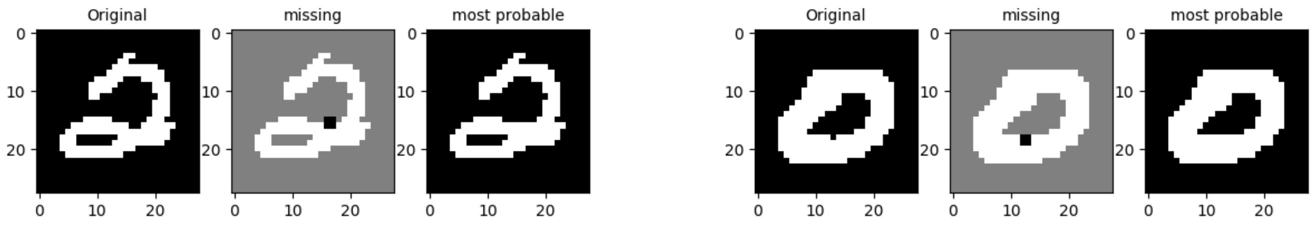


Figure 11: Different from original In-painting for the 2 × 2 missing pixels data set. Left image: original, middle image: missing pixels, right: in-painting

Model: GRU 1layer128units	1x1 block	2x2 block
GT cross-entropy	0.0831	0.0818
In-painting cross-entropy	0.0833	0.0821

Table 13: Cross-entropy of the ground truth images and in-painting images. The cross-entropy reported is averaged over the 1000 images in the data sets.