Matilde Piccoli, mp2419, 01764158

1. 03 Network Training

Task 1: Reported in Table 1 is the best validation accuracy for each model. For the data augmentation two models were used: a lighter augmentation consisting of a *RandomRotation* of 0.2 and a *RandomZoom* of height 0.2; a more aggressive augmentation, using a *RandomRotation* of 0.30, a *RandomZoom* of height 0.4, vertical *RandomFlip*, and a *RandomTranslation* (height=0.2, width=0.2, fill='reflect', interpolation='bilinear'). As expected, the first strategy leads to similar performance as the one using no augmentation, as the data changed so little that the model did not find any more (or less) difficulty to classify (data still representative of the original input space). On the contrary, the more aggressive one changed the data so much that it became unrepresentative of the dataset (increasing the the input space) and the model's performance decreased.
When using Dropout with a rate=0.2 (details in Table 1), the performance slightly improved and the training and validation curves stayed closer throughout all epochs, indicating that using only a part of the the neurons at every batch can avoid overfitting, since it gives an opportunity to learn independent representations during training [2]. When Batch Normalisation was separately applied, this did not have a substantial difference on the best validation accuracy, but increased the speed of convergence during training, making the network more stable [3]. When both regularisation techniques where applied, the validation accuracy increased to almost 84% and, overall, the generalisation error was lower (thanks to Drop Out) than the model without regularisation and the speed of convergence higher (thanks to Batch Nomalisation), making this combination the highest-performing model.
Initialising the weights to zero lead to the model being stuck in a local minimum that kept the validation accuracy to 10% throughout all epochs, as the weight were unable to be updated (no gradient) and remained fixed at the same value.
Finally, the plots of the performance of models with SGD optimiser show how lower learning rates make the generalisation error decrease (validation and training curve closer to each other in Figure 1), as the model has more time to adapt to the problem; however, lower learning require also more time to reach the same level of training error as the one for higher learning rate, since the model is updated more slowly at each epoch and therefore the overall validation accuracy is lower (45.5% vs 71.7%) for the same number of training epochs used.

Model	Best val accuracy	Parameters used
No data aug.	79.8 %	/
Data aug. 1	74.4 %	rotation, zoom
Data aug. 2	59.5 %	rotation, zoom, flip, translation
Dropout	81.4 %	dropout after every max pool, rate=0.2
Batch norm.	79.3 %	batch norm. after every max pool
Batch+Dropout	83.5 %	as above, with batch before drop out
Zeros init.	10.0 %	zero initialiasation
SGD 1	71.7 %	learning rate $= 3e-3$
SGD 2	58.9 %	learning rate $= 1e-3$
SGD 3	45.5 %	learning rate $= 3e-4$

Table 1. Best validation accuracy for each model trained

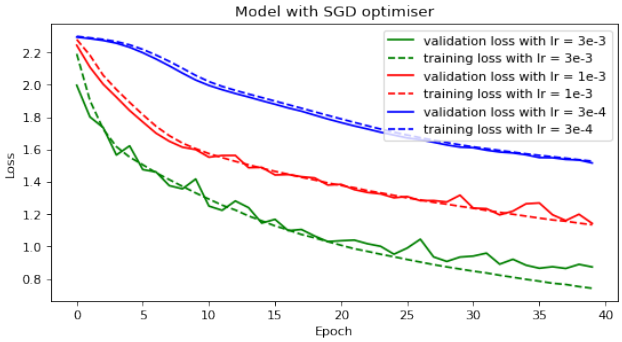

Figure 1. Losses with SGD optimiser at different learning rates: a lower learning rate makes the generalisation error decrease, but requires more time to reach the same level of accuracy

2. 04 Common CNN architectures

Task 1: Table 2 reports the test accuracy, training time and inference time of the models, while Figure 2 shows their accuracy behaviour per epoch. All three models show signs of strong overfitting after some epochs (training accuracy increasing 30% more while validation stays the same), due to the high complexity of VGG16 in comparison with the problem proposed. It was noticed that the VGG16 without pre-training was much slower in learning, as expected, as weights were initialised randomly, resulting in more training and therefore a longer total training time and lower test

accuracy for the same number of training epochs. Conversely, the one using transfer learning had much lower training and inference time since was based on pre-trained data and layers were frozen; its test accuracy was higher after a lower number of epochs, indicating that the features pre-learnt were representative of the test data. Finally, the model using fine tuning represents the best model as it achieved even higher test accuracy while maintaining a similar total training time of the transfer learning. This is because fine tuning uses transfer learning but adapts the model output to fit the specific problem, meaning that is a good compromise between using pre-acquired knowledge and adapting it to the specific data used for training. The inference time remains similar between the model as this is based only the forward propagation, which changes between different architectures and not different learning process, as in this case.

For the alternative model, multiple architectures were tested, however the ResNet50 with pre-training was the one that gave the best training and test accuracy, similar to the fine tuning models of VGG16, as its architecture is particularly suited for image classification problems. This model has faster training compared to the standard VGG16 (see slope at initial epochs in Figure 2) thanks to the transfer learning; however, since ResNet50 is a much deeper model, its total training time is still higher that VGG16 with similar pre-training (model 2). Also in this case, after a few epochs, the training accuracy keeps increasing and validation stays the same, indicating overfitting, due to the high complexity of the architecture.

Model	Test Accuracy	Tot. Train Time (ms)	Inference Time (ms)
VGG16 no pretraining	30.2%	1056	0.404
VGG16 transfer learning	46.5%	238	0.415
VGG16 fine tuning	52.2%	275	0.559
ResNet50 pretraining	51.7%	421	0.487

Table 2. Test accuracy and Times for VGG16 and ResNet50: highest performance achieved with fine tuning. The GPU used was the Tesla P100-PCIE-16GB.

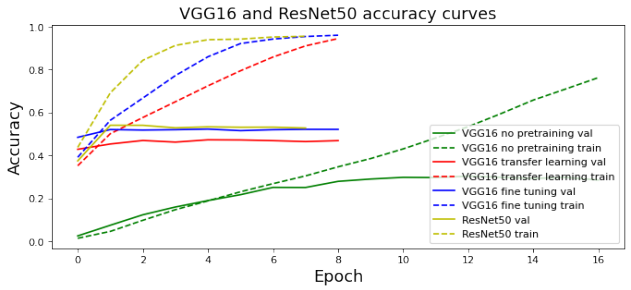

Figure 2. Accuracies of the VGG16 (no pretraining, transfer learning, fine tuning) and alternative model (ResNet50): overfitting after a few epochs for all the models, faster training for ResNet50

3. 05 RNN

Task 1: Figure 2 shows the test curves for different window sizes. Using a window size of 1, since the prediction is based only on the previous time step, the curve is very close to being just a copy of the true value delayed by one time-step, with little actual predictions on the next time-step value and giving a big error when the true value actually changes in between time-steps. As the window size increases, the shape of the curve is less similar to the true value but the prediction gets more similar to the true value in the same time-step, indicating more learning has been achieved. This is because the prediction is based on weighted average of multiple previous steps and therefore the models can better capture the time behaviour. However, if the widow size is increased beyond 14, corresponding to the approximated periodicity of the signal, the prediction is expected to decrease in accuracy back again as the information included are from sample too far in the past.

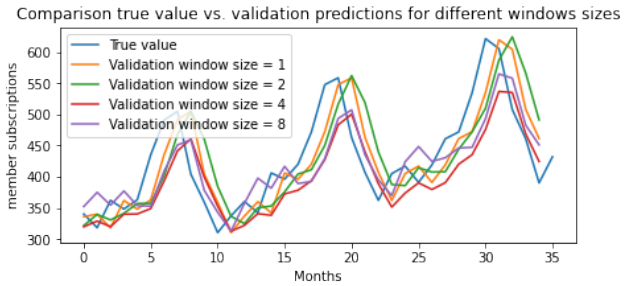

Figure 3. Test curves for different window sizes compared against the true value: better predictions of the time behaviour when the window size increases

Task 2: Table 3 shows the test accuracy for the model using three different embedding methods and the score of positive and negative reviews, corresponding to the sentiment prediction. The main results are: using embedding dimension of 300 and LSTM, instead of just embedding dimension of 1, makes the training faster (number of epochs in figure Appendix 7.1), but ultimately leads to lower test accuracy as the dimension is too high for complexity of the problem and the models overfits (training accuracy increases whilst validation stays the same); instead, using GloVe embedding helps both in terms of test accuracy and quality of the sentiment predictions, as this model keep tracks of the word-word co-occurrence globally [4]. This helps creating a more meaningful representation of the word space, as demonstrated when printing the closes words to the word *"good"* for the first and last model (simple embedding: *"maria", "hold", "soccer"*; Glove: *"better", "well", "always"*) and noticing that only the Glove ones share an actual link in the semantic/grammatical meaning.

The use of LSTM in the last two models helps keeping track of the order of the words and not only whether they are

present or not in the sentence. Ultimately, this leads to a better sentiment prediction for the two given examples review (*"boring and not good", "good and not boring"*), as the third model can distinguish when the only thing changing is the order of the words, leading to significant difference in the score of positive and negative reviews, conversely to the first model (Table 3).

Model Embedding	LSTM	Test Accuracy (%)	negative review score (%)	positive review score (%)
dim=1	no	85.2	49.9	49.9
dim=300	yes	84.9	/	/
GloVe	yes	86.7	4.8	58.4

Table 3. Test accuracy for different model and sentiment predictions in terms of score from 0 to 1: GloVe embedding with LSTM outperforms the other methods

Task 3: For the word level model, it has been noticed that if the temperature is low, the grammatical structure is good but the variety of word/sentences is not great; while when the temperature T is high, the vocabulary is wider but overall meaning of the sentence is poorer. This is expected as low temperature corresponds to taking always the safest choice (highest probability). Additionally, when the BLEU score is computed (Figure 4), since the score is based on matches on a word level, for the word level model, the scores remain very similar with different temperature, since the generated are always existing words, even if the semantics of the sentence might be corrupted.
On the contrary, the text generated by the character level model produces very different BLEU scores with different temperature. In particular, with high T, the words generated do not correspond to any real word and the sentence losses not only its grammatical structure, but also its semantic meaning. Therefore, as the temperature rises, the BLUE score decreases as there are less matches on a word level, whilst at low T the performance is very similar to the word level model, with poor variety both in words and characters used.

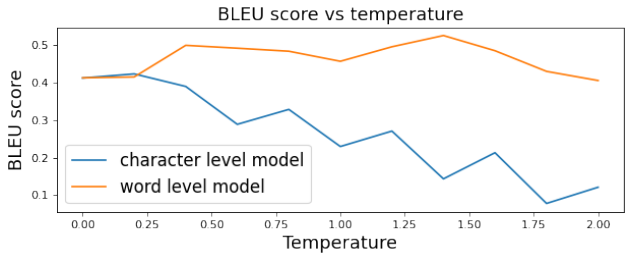

Figure 4. BLEU score for different temperature values (from 0 to 2) for both the character-level model and the word-level model

4. 06 Autoencoders

Task 1: From Table 4, the performance of linear autoencoder and PCA led to similar results in both accuracy and MSE of the training and validation sets, as both use linear representation of the feature space (using an orthogonal basis for PCA); in particular, the accuracy of non-convolutional autoencoder, when applied to the classifier, stayed around 80% for both training and validation no matter the number of dense layers, activation, dropout, batch etc (final architecture in Appendix 7.2). This indicated that the error is due to the non-linear relations in the feature space that cannot be represented with this type of autoencoder, rather than the architecture itself. In fact, when using a convolutional autoencoder and encoding also the non-linear features (asymmetric autoencoder used, architecture in Appendix 7.2), the performance increased substantially to almost 91%. The down side of both autoencoders, but especially the latter one, was the higher computation needed to train the model compared to PCA; however, in the second case, this can be largely justified given the improvement in performance.

Model	Training Accuracy	Validation Accuracy	Training MSE	Validation MSE
PCA	81.0%	81.4%	0.0258	0.0256
non-conv	80.9%	81.3%	0.0156	0.0124
conv	91.0%	91.5%	0.0214	0.0160

Table 4. Performance evaluation: highest performance achieved by convolutional autoencoder, despite the higher computation; PCA and linear autoencoders have similar accuracy. Note: a slightly higher value in the validation set is due to the test data used, in this case probably similar to the one used for training

Task 2: Using different loss function led to both different quantitative performances, in terms of MSE on the test set (Table 5), and qualitative ones, as observed form the reconstructed images(Figure 5). In general, the MSE error seems to match quite accurately the actual quality of the image retrieved, and the best performance obtain, from both perspectives, was with MS-SSIM. In fact, this one inherits the features of the structural index similarity of SSIM, but is more robust to variations as it combines the SSIM index of several versions of the image at various scales (high accuracy in colour, contrasts, edges and lighting). On the contrary, SSIM measures the similarity between pictures globally but does not take into account differences regionally, meaning that the images are distorted in the edges, colour and have patches. The other two loss functions tested were MAE, which finds the average absolute distance between the predicted and target values, and resulted in pictures robust to outliers but blurry; and PSNR, which minimises the reconstruction error computed from ratio of power of real and generated image globally (not particularly suited for images) and therefore resulted in very poor quality results.

Loss function	SSIM	MS-SSIM	1/PSNR	MAE
MSE (test set)	0.0081	0.0053	0.0318	0.0056

Table 5. Performance evaluation of different loss functions: lowest error using MS-SSIM loss function

Figure 5. comparison between different loss function's results and the noisy and clean image: best picture using MS-SSIM

5. 07 VAE GAN

Task 1: Comparing the two VAE with and without KL divergence loss led to the following results (Table 6): the first model outperforms in terms of the Inception Score, as this takes into consideration the probability distribution of y compared to real values, and the KL has the effect of optimising this distribution (making it close to uniform); the MSE is, however, slightly lower for the model without KL divergence, since this model's optimisation process is fully relaying on this type of loss. The cGAN had the best IS out of all the three models, indicating that even if cGAN don't actually minimise the difference in distribution between produced and target ones, it still recreates it by trying to generate data that can fool the discriminator [5].

Model	MSE	IS on test set
VAE with latent dim = 10	0.0116	7.3363
VAE without KL divergence	0.0107	6.0321
GAN with random dim = 10	/	8.1955

Table 6. MSE and Inception score on the test set for different VAE and GAN models: best performance given by GAN

Task 2: The quantitative scores (MAE) of the MAE and cGAN trained model are very similar (score of 0.0449 and 0.0458 respectively), with the first model characterised by a slightly better performance (probably due to the loss function itself being MAE). However, when inspecting the qualitative results (Appendix 7.3), it is clear that the two strategies used lead to different result: the MAE-trained model leads to a range of colours quite restricted, whilst the cGAN-trained model uses a range of colours more similar to real pictures one and, despite the error compared to the actual picture, and maintain a quite realistic relation between coloured areas within the pictures. The MAE-model have a lower quantitative error, but its qualitative outcome is closer to the BW picture than to the original one, opposite to the cGAN results. This is because cGAN has a more flexible and accurate way of calculating the loss during training, as the discriminator works as loss function that is learned directly from the data and updated throughout the training, instead of using a fixed loss function (such as MAE) a priori [1]. Therefore, we could say that, overall, the performance of the second model is higher, since for similar quantitative result, the qualitative ones are better.

6. 08 RL

Task 1: Q-learning and SARSA mainly differ in the way of updating Q, since the first bases it on the maximum Q' over all possible actions A for the next state S' (off-policy learning method) while the second uses the same policy used for updating of A' (on-policy). Therefore, since Q learning does not use the policy to update Q(s,a)[6], when changing between ϵ-greedy and Softmax, its behaviour (Figure 6) does not differ much in terms of speed of convergence to a high score. Conversely, SARSA behaviour varies a lot between the two policies: using ϵ-greedy, SARSA has a similar performance to Q-learning, reaching the maximum score similarly fast and remaining stable; however, when SARSA is based on Softmax policy, the performance decreases, mainly because of the temperature used (T = 0.025); this temperature influences the speed of convergence as the algorithms chooses almost always the value in the same column of the array of possible actions (left/right). In fact, when changing the temperature of Softmax, the performance changed significantly, outperforming ϵ-greedy policy if T = 0.04, and performing slower training again when T was further increase (very low confidence in the action taken). This is a further indication of how SARSA performance is strictly linked to the policy (updating the next state) and how Softmax is itself highly dependent on the parametrisation of its probability distribution (high temperatures correspond to low penalisation of bigger logits and lower confidence, but also possibility of successful exploration [7]). Modifications for SARSA and Softmax can be found in the Appendix 7.4.

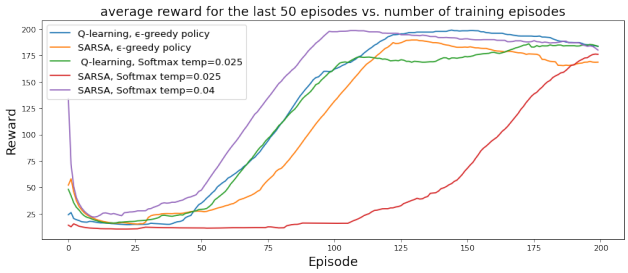

Figure 6. average reward for the last 50 episodes vs. number of training episodes: Q-learning depends much less on the type of policy then SARSA

7. Appendix

7.1. 05 RNN

Figure 7 shows the training and validation accuracy curves for the different text embedding models. the highest accuracy was obtained when using LSTM (model 2 and 3), faster training (y axis) using GLoVe embedding (model 3, pre-learning), and smallest generalisation error with the simpler model (model 1, no overfitting).

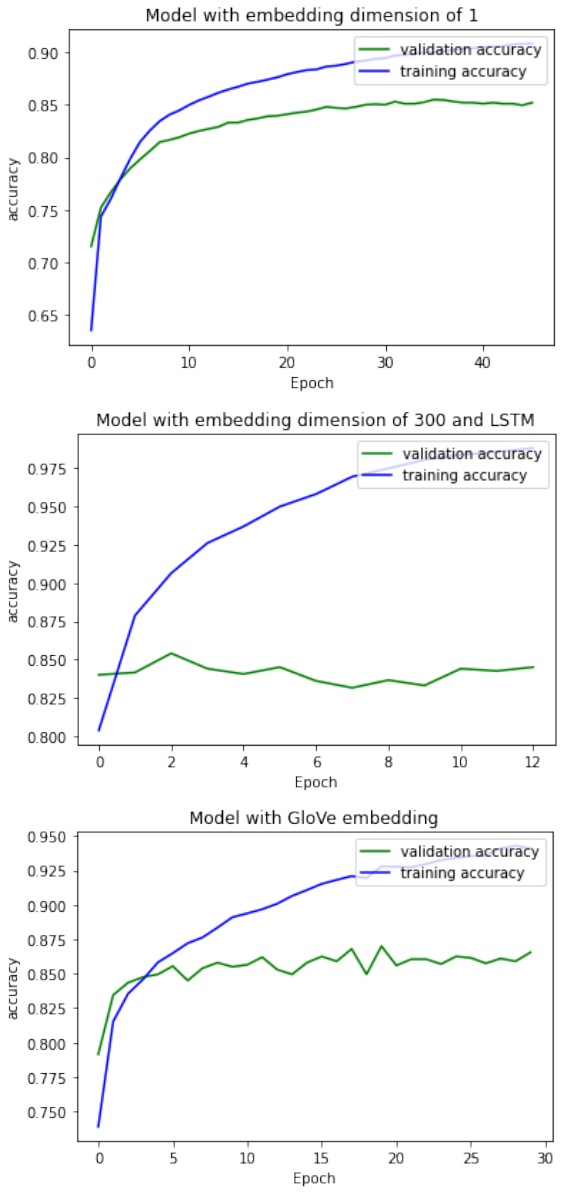

Figure 7. performance of different embedding models: highest accuracy obtained when using LSTM; faster training (y axis) using GLoVe embedding

7.2. 06 Autoencoders

Figure 8 shows the architectures used for the non-convolutional and convolutional autoencoders: the first one consists of a sequences of Dense, Batch Normalisation and Drop Out layers with Softmax activation; the second consists of four series of Convolutional and Max Pooling (or Global Pooling) layers with increasing number of convolutional filters and ReLu activation, and then Dense, Batch Normalisation and Drop Out layers. The autoencoder is asymmetric as this showed to improve the timings whilst maintaining the performance (encoder and decoder are not symmetric).

```
autoencoder = Sequential()
autoencoder.add(Flatten(input_shape=(32,32,)))
# The representation has dimensionality 10, do not change the dimensionality
autoencoder.add(Dense(10, name='representation'))
autoencoder.add(Dense(128*128))
autoencoder.add(Activation('softmax'))
autoencoder.add(Dropout(0.3))
autoencoder.add(BatchNormalization())
autoencoder.add(Dense(128*128))
autoencoder.add(Activation('softmax'))
autoencoder.add(Dropout(0.3))
autoencoder.add(BatchNormalization())
autoencoder.add(Dense(32*32))
autoencoder.add(Reshape((32,32,1)))

autoencoder.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['mse'])
```

```
#ENCODER
conv_autoencoder = Sequential()
conv_autoencoder.add(Conv2D(32, (3, 3), padding='same'))
conv_autoencoder.add(Activation('relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))

conv_autoencoder.add(Conv2D(64, (3, 3), padding='same'))
conv_autoencoder.add(Activation('relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))

conv_autoencoder.add(Conv2D(128, (3, 3), padding='same'))
conv_autoencoder.add(Activation('relu'))
conv_autoencoder.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))

conv_autoencoder.add(Conv2D(256, (3, 3), padding='same'))
conv_autoencoder.add(Activation('relu'))
conv_autoencoder.add(GlobalAveragePooling2D())

# The representation has dimensionality 10, do not change the dimensionality
conv_autoencoder.add(Dense(10,  name='representation'))

# NB 'Dense(10, name='representation')'
# layer have to be the one with the smallest output vector for it to qualify as the final encoding layer
# (and therefore for  the representation vector to have dimensionality 10)

# DECODER
conv_autoencoder.add(Dropout(0.3))
conv_autoencoder.add(BatchNormalization())
conv_autoencoder.add(Dense(128))
conv_autoencoder.add(Dropout(0.3))
conv_autoencoder.add(BatchNormalization())
conv_autoencoder.add(Dense(1*16*16))
conv_autoencoder.add(Reshape((16,16,1)))
conv_autoencoder.add(UpSampling2D((2, 2)))

conv_autoencoder.build((None,32,32,1))
conv_autoencoder.summary()
conv_autoencoder.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['mse'])
```

Figure 8. architectures used for the non-convolutional and convolutional autoencoders

7.3. 07 VAE GAN

Figure 9 shows the comparison in terms of qualitative performance between the MAE and cGAN-based model. The colours are more realistic in the pictured produced by cGAN, despite the fact that they might be wrong compared to the original picture and the quantitative error is very similar to the MAE model.

Figure 9. qualitative examples of the performance of MAE and cGAN models, more realistic colouring obtained with cGAN

7.4. 08 RL

The modification to use SARSA instead of Q-learning consisted in changing the function that updated S' in Q'(S',A) by introducing exploration on top of the possibility to exploit the policy.
Additionally, to implement the Softmax policy, the e-greedy was modified so that exploration happens instead of depending on a threshold set by epsilon, by a number generated by a uniform distribution and compared to the output of softmax applied to the score produced by the function model.predict().

References

[1] A blog on a.i. https://nchlis.github.io/2019_11_22/page.html.

[2] J. Brownlee. Understand the impact of learning rate on neural network performance. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras.

[3] J. Huber. Batch normalisation. https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding.

[4] C. D. M. Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/.

[5] S. Li. Gan or vae? https://medium.com/@lishuo1/which-one-should-you-choose.

[6] A. R.Sutton. difference between q-learning and sarsa. https://stackoverflow.com/questions/6848828.

[7] K. Te. How does temperature affect softmax in machine learning. http://www.kasimte.com/2020/02/14/how-does-temperature-affect-softmax-in-machine-learning.