Alessandro Serena - AS6316 - CID: 01188591

1. CNN Architectures

Task 1.1. VGG on Tiny-ImageNet. In this task we compare the training and test performance, on the Tiny-ImageNet dataset, of different weights initialization methods for the VGG-16 and VGG-19 architectures [8]. The initial weights values are (a) random, (b) pretrained on ImageNet [1], (c) pretrained on ImageNet and fine-tuned by updating only the parameters in the fully connected layers.

As shown in fig. 1, using the weights pretrained on ImageNet efficaciously jump-starts the learning process, making the model able to quickly learn the new dataset. However, it is necessary to note that if all layers are kept as trainable, the model quickly overfits, reaching almost 100% accuracy on the training set. On the other hand, fine-tuning only the fully connected layers effectively reduces the trainable capacity of the networks, causing a smoother training curve. I expect the fine-tuned network to perform better on the validation and test sets than the fully trainable one if trained for more than 20 epochs.
Given the large number of trainable parameters (∼140M), the random initialization method yields unsatisfactory results as it would need many more than 20 epochs to converge to a good accuracy.
These comments apply to both VGG-16 and VGG-19 architectures. The main difference between the two is that VGG-16 consistently outperforms VGG-19, as the former adapts more quickly to the new task given its smaller parameters space.

In fig. 1 are also shown the training curves for the fine-tuned ImageNet weights models when the input image size is upsampled from 64x64x3 to 224x224x3 (marked as *large input*). Using the larger input size lets us preserve all dense layers but the last one in the VGG architecture. We can then import the ImageNet weights also for the first two fully connected layers (4096 neurons each), bringing the number of trainable parameters to 819,400, while the previously used fine-tuned models had 25,993,416 trainable parameters. This yields a smaller trainable capacity of the network, which is reflected by a flatter learning curve.

In table 1 are reported the best-performing VGG-16 and VGG-19 models in terms of test set accuracy and inference time.
Check section 7.1, in the appendix, to see how the different models perform on the validation set (fig. 9), the complete test set results and the inference times (table 7), and to see separate plots for the architectures tested (figures 10, 11, and 12).

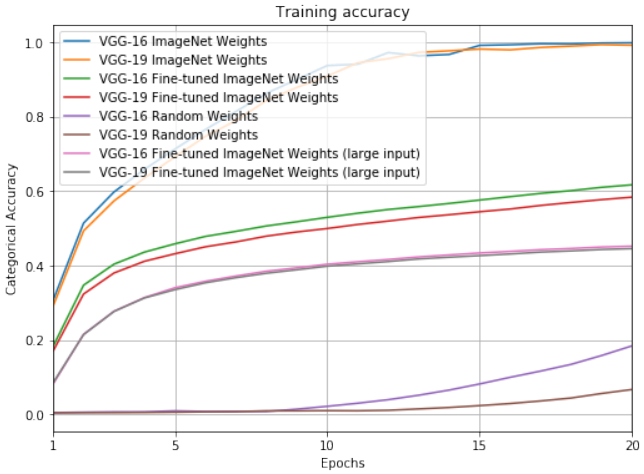

Figure 1. T1.1. Training and validation accuracy for different VGG architectures and weights initializations.

Architecture	Weights initialization	Input size	Test set accuracy (%)	Inference time (μs)
VGG-16	ImageNet	64x64x3	***58.42***	***283***
VGG-19	Random	64x64x3	3.14	***339***
VGG-19	ImageNet	64x64x3	***54.30***	343

Table 1. T1.1. Test set accuracy and inference time for the best model trained for both VGG architectures (the metric they are the best in is marked in bold).

2. Recurrent Neural Networks

Task 2.1: RNN Regression. The objective of this task is to create a LSTM [2] model to predict the number of passengers flying on a certain airline on a given month. We explored different window size values, and the predicted test set curves are presented in fig. 2. We cannot really see any specific pattern arising from a qualitative analysis of these curves, apart from the fact that larger window sizes force the prediction curve to start later in time. However, if we quantitatively inspect the test set MSE for different window sizes (reported in fig. 13, in the appendix), we can see that we have lower errors when we use windows of 12 months or in the range 18-22 months. This is because

the model is able to extrapolate the seasonality trend, that emerges from the data in summer each year, when it sees a year or 1.5 years of past data.

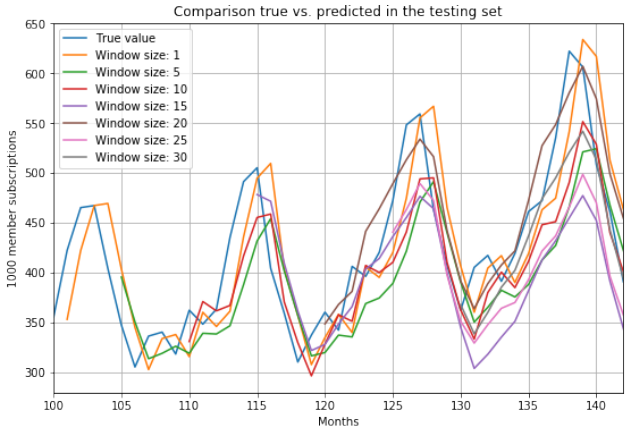

Figure 2. T2.1. Test set predictions for different window sizes.

Task 2.2: Text Embeddings Importance. In this task we start exploring the field of NLP by studying the effect of text embeddings in a sentiment analysis task based on a dataset of IMDB reviews [3]. We tested 3 different word embeddings initialization modalities: (a) random uniform, (b) GloVe [6] embeddings, and (c) non-trainable GloVe embeddings.

The model used is a single-layer bidirectional LSTM with 100 units, preceded by an embedding layer that has a fixed dimensionality of 300 for each embedding. The training and validation curves for the different embeddings initializations are presented in fig. 3.
All the models quickly overfit on the given dataset, reaching almost a perfect classification score on the training set in only 10 epochs, and peaking in terms of validation accuracy after 3-4 epochs. As expected, the model with non-trainable GloVe embeddings shows a smoother learning curve. This is because the model is unable to tune its embedding layer to overfit on specific input words, forcing the LSTM units to learn more complex patterns among words. This makes the model achieve the highest validation accuracy.

Table 2 shows the 10 closest words in the embeddings feature space to the word *action* for the three trained models. The randomly initialized embeddings fail to learn a human-comprehensible word representation, although the model performs well on the validation set. On the other hand, the fine-tuned GloVe embeddings learn some specific word-level similarities, that are specific to the domain of movie reviews.

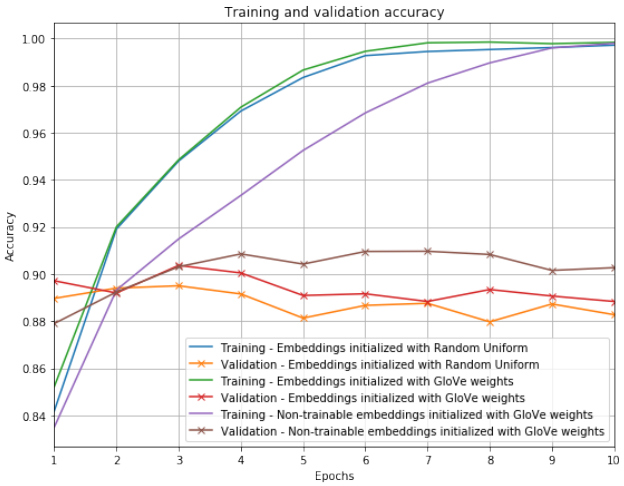

Figure 3. T2.2. Training and validation accuracy for different embeddings initialization methods.

Embeddings Initialization	**Most similar words to: `action`**									
Random	a	i	br	and						
GloVe	actions	taken	result	should	way	kind	similar	example	taking	take
Non-Trainable GloVe	actions	taken	take	result	taking	moves	instead	which	but	yet

Table 2. T2.2. Most similar words to `action` generated from different embeddings initializations.

Task 2.3: Text Generation. In this task we work on a text generation task at character-level based on scripts of *Game of Thrones*.

In table 3 we show the text generated for three different temperatures. Using a lower temperature, the text is more predictable with shorter, more common, words. When we increase the temperature, the text becomes more surprising and uses more complex vocabulary and semantic structures, in this case at the expense of the meaningfulness of the exchange of lines between characters.
A solution would be to create a word-level predictive model, which will have more control over the general meaning of the sentence, with the downside of heavily increasing the vocabulary space.

Input text seed	JAIME: Do you know how long it's going to take us to get to King's Landing walking through fields an
Temperature 0.0	[an]d how many missing me to the throne room. TYRION: I don't know what you want to stay to the great ma
Temperature 0.5	[an]d men who didn't know he was a drink. ROOSE: Where is not the two– DAENERYS: Oh, I wanted me the tr
Temperature 1.0	[an]d few is a rule Winterfell. DOREAH: I am not takes yearst's a chin. JAIME: There's nothing to deavy

Table 3. T2.3. Text generated using different temperature values.

In fig. 4 we quantitatively compare different temperature values by reporting the BLEU score [5] of the text generated. From the plot we can see that there is no correlation between the values. This is because BLEU is not an effective metric for scoring text generation tasks, as it does not

consider meaning or the sentence structure. Also, BLEU does not take into account the morphological richness of the text, which would ideally increase the score for higher temperatures.

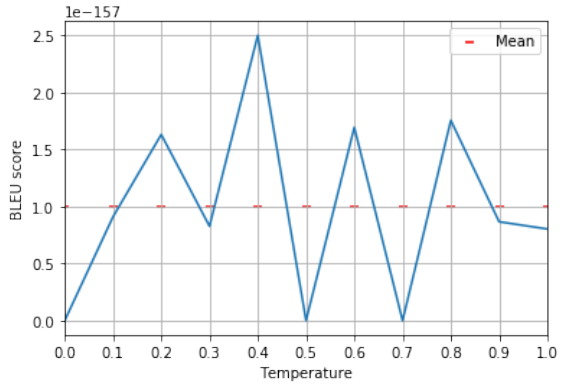

Figure 4. T2.3. BLEU scores for different temperatures.

3. Autoencoders

Task 3.1: Non-linear Transformations for Representation Learning For this task we analyse the impact of using non-linear activation functions and more layers in the context of representation learning with autoencoders. We used the MNIST dataset.

We explored different depths (the results are shown in fig. 14) and the best architecture was found to be the one described in table 4. By using more layers, non-linearity, and a representation vector size of 50, this model has a much lower MSE on the test set than the simple linear model defined in the tutorial (MSE 0.056). Examples of reconstructed images by the two models are shown in fig. 5.

In fig. 6, we show how digit classes are distributed in the representation space (see fig. 15 for a clearer distribution obtained with t-SNE). By clustering the encoded test set images using K-Means and assigning labels to the clusters based on a majority-voting method, the proposed autoencoder yields a categorical accuracy of 61.0%. This is much better than both random guessing (around 10%) and the linear model from the tutorial (39.9%).

Model category	3-layer encoder, 3-layer decoder
Representation vector size	50 (Compression rate 93.62%)
Number of parameters	1,197,324
Feature vector sizes	[784, 540, 295, 50, 295, 540, 784]
Epochs of training	49
Activation functions	ReLu, Sigmoid (output)
Reconstruction error (test MSE)	0.0031

Table 4. T3.1. Best performing autoencoder found.

Task 3.2: Custom Loss Functions In this exercise we compare the performance of a UNet [7] model on CIFAR-100 image denoising, when the model is trained using different loss metrics.

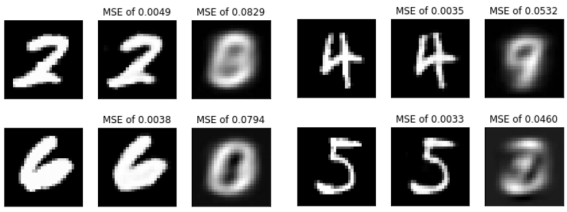

Figure 5. T3.1. Original image VS. best autoencoder reconstruction VS. linear autoencoder reconstruction.

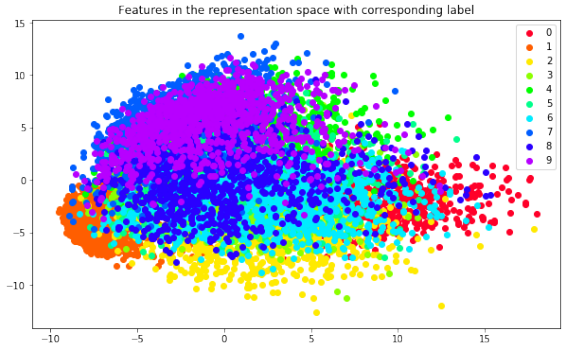

Figure 6. T3.1. Classes distribution in the representation space. Two-dimensional representation obtained with PCA.

We used as loss functions MSE, MAE, and correntropy with different σ values. To compare the models, we used MSE as a common metric for the test set results.
As shown in table 5, MAE yields the best score, followed closely by correntropy ($\sigma = 10^2$) and MSE. We can also note that increasing the value of σ in correntropy has a negative effect on the network training, with *early stopping* activating before the end of the target 15 epochs. This is because a larger σ scales up the error terms, leading to larger gradients being backpropagated and to a faster training network that overfits more quickly.

Loss Metric	**Epochs Trained**	**Test MSE**
Correntropy ($\sigma = 10^2$)	15	0.0018
Correntropy ($\sigma = 10^4$)	9	0.0023
Correntropy ($\sigma = 10^6$)	5	0.0063
MSE	15	0.0019
MAE	15	0.0015

Table 5. T3.2. Test set MSE for UNet trained with different loss functions.

4. Variational Autoencoders & GANs

Task 4.1: MNIST Generation Using VAE and GAN In this task we explore the field of generative models by comparing different VAEs and GANs on a MNIST generation task. All models are compared on their respective Inception scores, and VAEs also on the test set reconstruction MAE. The results are presented in table 6.

For VAEs, we varied the latent vector dimensionality and the number of layers in both the encoder and decoder subnetworks. We note from the results that the reconstruction MAE is impacted more by the dimensionality of the latent vector, rather than by increasing the number of layers. This result also confirms the trend seen in the autoencoders section, as reported in fig. 14. On the other hand, the Inception score is more correlated with the increase of layers, compared to augmenting the size of the latent representation vector. This is because, with an extra hidden layer, the decoder network can use more non-linearity to generate more realistic data.

For what concerns the two GANs tested, we see that the network with a larger random input vector reports a significantly higher inception score, also when compared to the VAEs. This is because the generated data is more diverse, since the model can rely on more randomness in the input. Therefore, the better score is due to the $\mathbf{P}(y)$ term, in the IS formulation, having a more uniform distribution.

Model	Latent (Random input) dimension	Extra layer	Inception Score	MAE
VAE	2	No	4.42	0.101
VAE	20	No	5.88	0.041
VAE	2	Yes	6.23	0.091
VAE	20	Yes	6.84	0.043
GAN	1	n/a	2.43	n/a
GAN	10	n/a	8.33	n/a

Table 6. T4.1. Inception and MAE scores for different generators.

Task 4.2: Quantitative VS Qualitative Results In this task we compare a UNet (trained with MAE as loss function) and a cGAN [4] on the task of colouring B&W images.

From a purely quantitative point of view, the UNet seems to be a slighlty better model, as it yields a lower test set error. The UNet has a MAE of 0.045, compared to 0.046 for the cGAN. However, if we inspect qualitatively the images coloured by the two networks, we notice that the difference between the quality of the output is not captured by the metrics used. Referring to fig. 7, we see that the cGAN generates coloured images that are much more vivid and realistic, compared to the UNet, even at the cost of using wrong colours for objects.

This key difference comes from the way the two models are trained. Indeed, as the UNet is trained by backpropagating the MAE given a reference image, the model learns to minimize the average distance between pixel values, hence learning a more conservative policy that qualitatively translates to a smooth, more uniform colouring scheme. On the other hand, the cGAN is trained to deceive a discriminator by creating images that are similarly coloured to a given distribution, and not by generating images that look exactly the same as the reference.

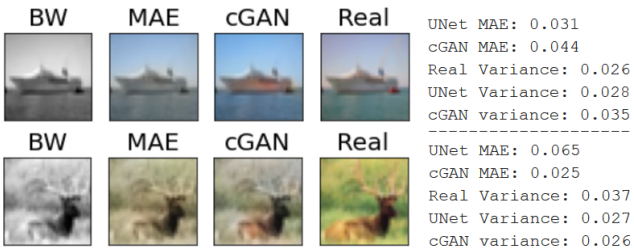

Figure 7. T4.2. Comparison of images coloured by a UNet (MAE, in the pictures) and a cGAN. On the right, we show that variance and MAE cannot fully express how realistic an image is.

5. Reinforcement Learning

Task 5.1: On-policy vs. Off-policy For this task we have trained four RL agents to move a cart in a mono-dimensional space to balance a pole. By looking at fig. 8, we note that Q-learning is a more stable learning procedure, when using both ϵ-greedy and softmax action-choosing policies, compared to SARSA. This is because, in the former, the policy used to learn the Q-function is greedy, as we are taking the maximum Q-value. Oppositely, in SARSA, being it an on-policy method, we update the Q-function by using the Q-value for the next chosen action based on the same policy that we are trying to learn. In other words, we learn the Q-function using an estimate of an estimate, hence leading to a greater instability. See fig. 16 for results obtained with different initial random seeds.

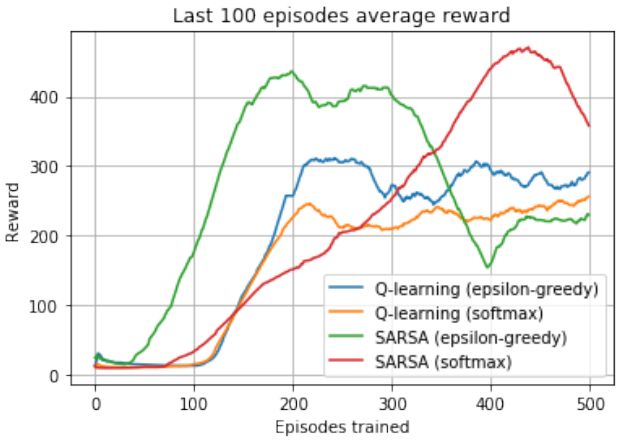

Figure 8. T5.1. Comparison of different learning approaches, in terms of average of last 100 episode rewards VS. episodes trained.

6. Conclusion

In this report, we have given an overview of many fields within Deep Learning. We have highlighted multiple common issues and discussion points, and tried to address them from both a quantitative and qualitative points of view.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[2] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[3] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[4] M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[5] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[6] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[7] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

7. Appendix

7.1. CNN Architectures

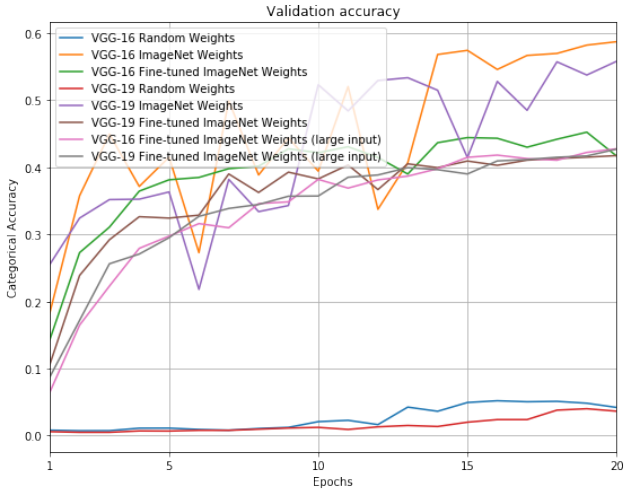

Figure 9. T1.1. Training and validation accuracy for architectures with large input data.

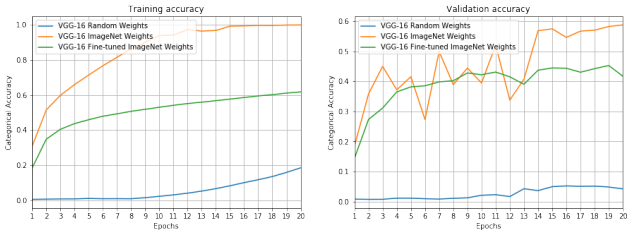

Figure 10. T1.1. Training and validation accuracy for VGG-16.

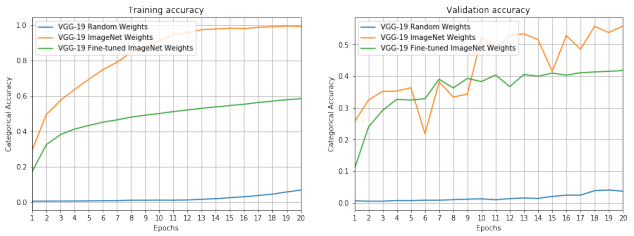

Figure 11. T1.1. Training and validation accuracy for VGG-19.

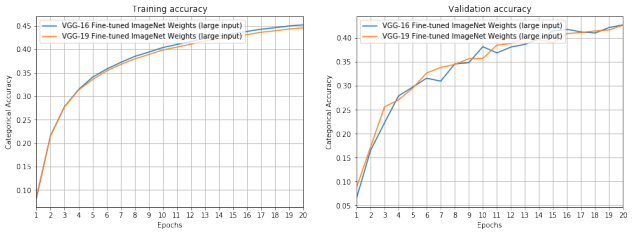

Figure 12. T1.1. Training and validation accuracy for architectures with large input data.

Architecture	Weights Initialization	Input Size	Test set accuracy (%)	Inference time (μs)
VGG-16	Random	64x64x3	3.82	298
VGG-16	ImageNet	64x64x3	58.42	283
VGG-16	Fine-tuned ImageNet	64x64x3	40.95	284
VGG-16	Fine-tuned ImageNet	224x224x3	41.76	1879
VGG-19	Random	64x64x3	3.14	339
VGG-19	ImageNet	64x64x3	54.30	343
VGG-19	Fine-tuned ImageNet	64x64x3	41.63	340
VGG-19	Fine-tuned ImageNet	224x224x3	42.49	2104

Table 7. T1.1. Test set accuracy and inference time for the different models trained.

7.2. Recurrent Neural Networks

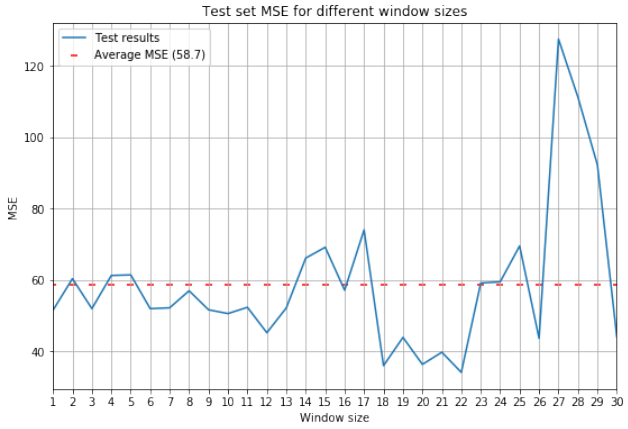

Figure 13. T2.1. Test set MSE for different window sizes.

7.3. Autoencoders

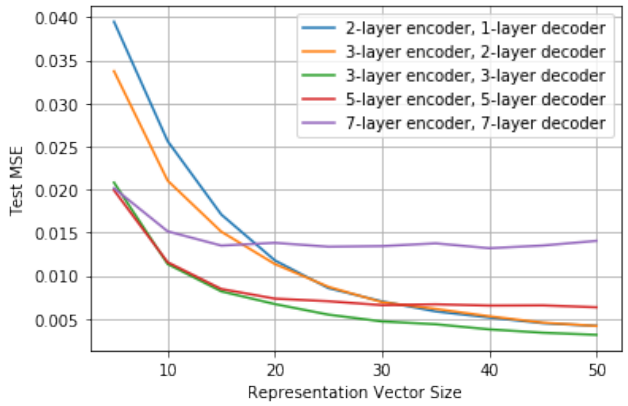

Figure 14. T3.1. Test set MSE for different autoencoder architectures.

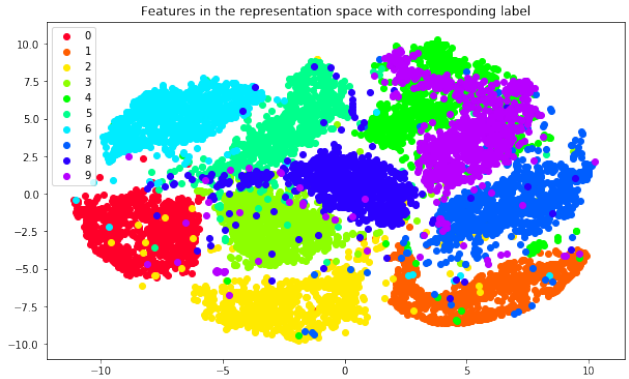

Figure 15. T3.1. Classes distribution in the representation space. Two-dimensional representation obtained with t-SNE.

7.4. Reinforcement Learning

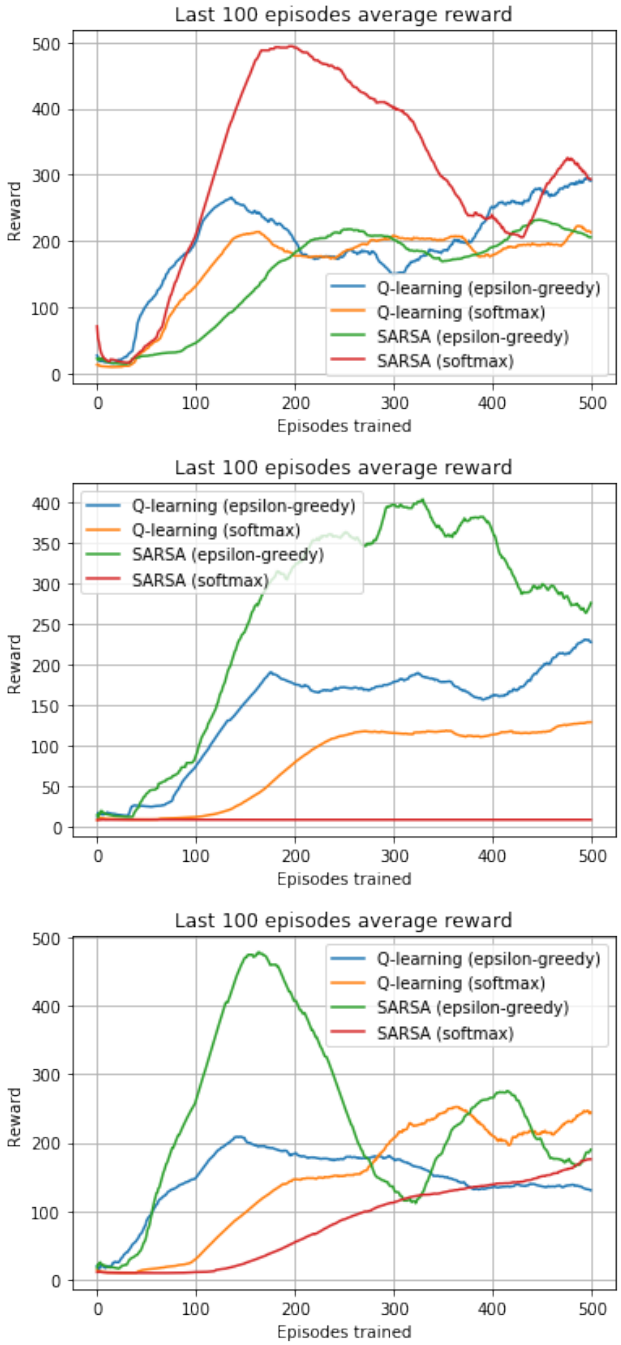

Figure 16. T5.1. Comparison of different learning approaches, in terms of average of last 100 episode rewards VS. episodes trained. Different figures represent different initial random seeds