Nima Karshenas, nk4718, 01500753

1. Tuning a Classification Model

Regularisation Techniques: The CIFAR10 dataset was trained with the same baseline CNN architecture(Figure 7 in Appendix) but with 3 Augmentation strategies. The first, no data augmentation is applied to the dataset(Vanilla Model). The second, random zoom with a height factor of 0.1, along with random rotation between -0.2π and 0.2π radians, and finally a random translation with a height and width factor of 0.1 (Light Model). The final model consists of, a random zoom of height factor 0.5, Random Rotation between $-\pi$ and π, random translation of height and width factor 0.3 and finally a random horizontal flip with probability 0.5(Aggressive Model). The loss curves for these models are shown in Figure 9 and the best validation accuracy out of the 40 epochs are shown in Table 1.

Model/Regularisation Approach	Best Validation Accuracy(%)
Vanilla	79.48
Light	76.25
Aggressive	59.89
Dropout probability	82.67
Batch Normalisation	80.73
BatchNorm + Dropout	83.55
Zeros Initialisation	0.100

Table 1: Best validation accuracies for the same base CNN model trained with different regularisation techniques.

With no augmentation, the model quickly overfits to the training data, seen when validation loss starts increasing and training loss continues decreasing after epoch 14(Figure 9). With light and aggressive augmentation both models seem to be still underfitting after 40 epochs, and thus still potentially have capacity to learn. With the increased variance of the training data, the complexity of the problem is also increased, with it increasing the time to learn the training data.
Although the 'Light' model shows slightly worse validation performance compared to 'Vanilla'(Table 1), it is likely that it will generalise better since it has the capacity to classify data with higher variance. By adding data augmentation, we prevent overfitting by increasing the variance of the data. As seen with the poor performance of the 'Aggressive' model, one must be careful such that this variance is not too large that it is no longer representative of 'real' data.

We now explore the effect of further regularisation techniques(also seen in Table 1) for the same baseline model as before. Batch Normalisation performs best in the first 2 layers, however, it shows a very minor improvement to the unregularised model. Batch Normalisation normalises the outputs from neurons to 0 mean and unit variance, this is particularly useful in tanh and sigmoid activation layers which are vulnerable to saturation and thus vanishing gradient by keeping values in 'linear' region. However, ReLU activation does not suffer from this problem, and so it has minimal effect on validation accuracy in this case.

A Dropout probability of 0.3 (in each layer) yielded the best validation accuracy(see Table 1). Dropout had a much more noticeable effect on validation accuracy compared to Batch Normalisation. Dropout stochastically changes the responsibility of each neuron, which prevents the co-adaptation of neurons. Dropout has an effect regardless of the activation. The best results of dropout and batch normalisation were combined (Dropout of probability 0.3 in each layer, Batch Normalisation in the first 2 layers), producing an improved validation accuracy, shown in Table 2. For zeros kernel initialisation, a validation accuracy of 0.1 is shown, this suggests that predictions are being made randomly (See Appendix 7.1.3). Biases and weights have been initialised to 0, the output of each neuron is thus initially 0, which is in the 'dead' region of ReLU, meaning the gradient is 0 for each neuron and the network cannot learn.

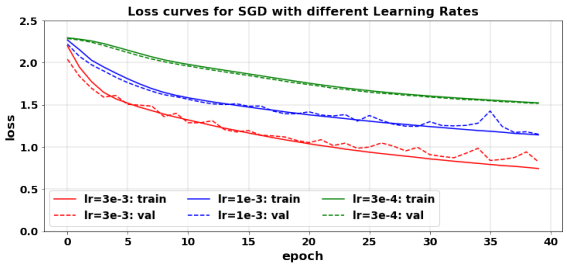

Figure 1: Training and Validation Loss curves for different SGD learning rates

Learning rate: The same base model was trained with Stochastic Gradient Descent(SGD) for various learning rates, the loss curves being shown in in figure 1. The learning rate in SGD corresponds to the step size when updating weights, a value too low results in long training times and a value too high can result in divergence from the global minimum. In figure 1 we see that, as expected the model with the highest learning rate is converging the quickest, how-

ever, it also shows no sign of divergence(has the smallest training loss) suggesting that the learning rate can potentially be further increased for even faster convergence.

2. Common CNN Architectures

The VGG-16 CNN architecture is loaded from the keras library and is trained on the Tiny ImageNet(TINet)in 3 different scenarios, referred to as M1, M2, and M3:

- Trained from scratch with no weight initialisation.(M1)
- Train the whole architecture with pre-loaded ImageNet weights.(M2)
- Train only the dense layers with pre-loaded ImageNet weights.(M3)

The accuracy curves for each of these approaches is shown in Figure 2,

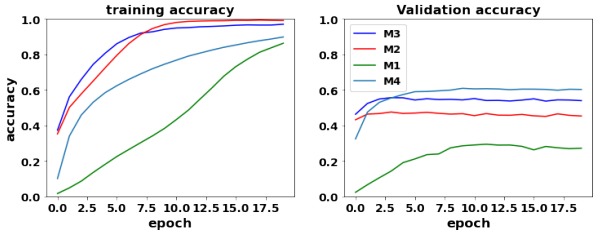

Figure 2: Accuracy curves for different approaches to learning the Tiny Image-net

M1 shows the worst validation performance out of the 3 methods, it also has a large discrepancy between the training and validation accuracy, meaning it is overfitting to the training data. The VGG16 architecture is designed for a much larger dataset(ImageNet(ImNet)) with higher resolution input images(224x224), and thus it is designed to be deep/complex as the data allows for this. When using it for a smaller dataset with smaller input dimensions, the features extracted in the final layers are too complex to have semantic meaning in the context of the problem and thus the model generalises very poorly. M2 and M3 are initialised with weights trained on ImageNet, and thus they begin training with weights being 'suggested' from a very similar problem, and thus closer to the global minimum. This allows for better validation accuracy in the first epoch than M1 ever reaches after 40 epochs, as well as faster convergence(see Total training time Table 2) compared to M1. M2 shows the fastest training time since not only does it have ImNet weights, backpropagation is only conducted in the dense layers, saving alot of calculation time. Weights in the convolution layers of M3 have been frozen at those trained only for ImNet, whereas M2 trains these weights along with Dense layer weights for TINet, hence why it shows better validation performance. All 3 VGG16 models show similar inference times as expected since forward passes are being made in the same architecture.

Approach	Best Val Acc(%)	Inference Time(ms)	Train Time(s)
M1	27.35	0.3081	907.5
M2	47.56	0.3079	192.8
M3	55.64	0.3394	528.1
M4	60.04	0.3621	745.2

Table 2: Performance of each CNN architecture: using Tesla P100-PCIE-16GB GPU

The EfficientNet B0(M4) was chosen as a reference model to compare VGG16 with. It shows better Top-1 accuracy in ImNet, with far fewer parameters than VGG16(See Table 7). Weights were initialised using pretrained ImNet weights as before. Since the network is more shallow, it overfits less to the smaller dataset and input dimensions of TINet, seen in its higher validation, and lower training accuracy in Figure 2. It has a slightly longer training time compared to M3 but this is a valid compromise for better validation accuracy(Table 2). Surprisingly though, although it has far fewer parameters, it has a longer inference time, this may be due to the complexity of the design of the architecture, and thus the complexity of calculations needed for a forward pass.

3. Recurrent Neural Networks(RNNs)

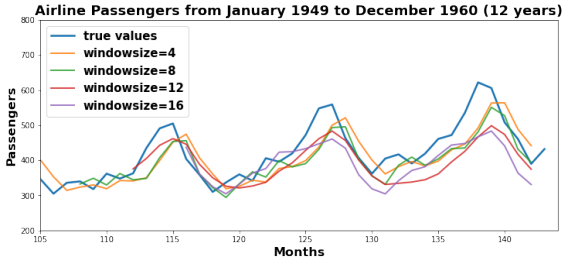

Figure 3: Predictions of number or Airline Passengers per month for different window sizes

RNN Regression: As window size is increased, the greater the temporal context the RNN has of previous data values. A window size of 8 shows the best performance. As window size is increased, there is 'smoothing' of the predictions. This notably happens when the window size is greater than the period of the data(12 months in this case). Moreover, The LSTM(all window sizes) is not able to predict turns in the data, it can only predict turns after it have observed them, i.e. the next month.

Text Embeddings: 3 models are trained on the IMDB sentiment dataset, one with embedding of dimension 1 and average pooling(Embeddings), an LSTM with trainable embeddings initialised with GloVe Embeddings(LSTM) and an LSTM unit with non-trainable GloVe embeddings(LSTM GloVe). The best validation accuracies are shown for each of the models in Table 3. Surprisingly the Embeddings model, with a much smaller embedding dimension and no LSTM unit, has similar validation performance to the other 2 models. This is largely due to

the simplicity of the task, the Embeddings model has no sequential context and thus if it, for example, sees largely 'negative' words, it will predict a negative sentiment. In the general case, the overall/average sentiment of individual words in a sentence will be a good indicator of the sentiment of the sentence, hence, the good performance. This is investigated on 2 phrases with opposite sentiment but similar words(See 7.3.1). A probability close to 0 equates to a negative sentiment, a probability close to 1 equates to a positive sentiment, results shown in Table 3.

Model	Best Val Acc(%)	Sentiment pred. (probability) phrase 1 ‖ phrase 2
Embeddings	85.45	5e-6 ‖ 5e-6
LSTM	85.65	N/A
LSTM GloVe	86.35	0.08 ‖ 0.50

Table 3: Table showing the performance of different models on the prediciton of the sentiment of an IMDB review

In both phrases the majority of the words are negative, the Embeddings model thus sees both phrases as totally negative as it does not have the capacity to capture the structure of the sentence. The LSTM GloVe model on the other hand detects at least that the second phrase **may** be positive(0.5 probability). The LSTM GloVe model performs better than the Embeddings, when the semantics of a phrase are in its structure and not in its words, due to its capability to capture the sequential context. In the Appendix, Table 8 shows the 10 most similar words to '8' for these two models. The embeddings model is trained only on the IMDB dataset and thus encodes the property of the word in the context of the sentiment of a review, hence why its closest words relate to a 'positive' review. The GloVe embeddings however have been trained on a very large, general dataset, and thus it encodes the meaning of the word in the general sense, hence why its most similar words are numbers.

Text Generation: In this task we predict the Game Of Thrones script for word and character level RNN models, with varying values of Temperature(See Section 7.3.3 for its effect). In the general sense, a temperature close to 0 encourages safe predictions to be made, and for larger temperatures, stochastic predictions are encouraged. The BLEU score for each of these models is shown in Figure 4.

The BLEU score was designed for text translation[2] and thus its focus is more on how much the text resembles training data. For temperatures close to 0, the word-level model is making repeated predictions in order to maximise 'safeness', this means that actually it does not resemble the structure in the original script, hence, it has the lowest BLEU score. As temperature increases, the BLEU score for the word-level model stays roughly constant, whereas the character-level begins to fall. Both models are making predictions more and more randomly, however, since the word-level model is making predictions on a word-to-word basis from a bank of words found in the script, it still has relevance to the training data, hence receiving a relatively good BLEU score. The character level model is making predictions randomly on a character-to-character basis and thus even words begin to not make sense, hence its poor BLEU score. It can thus be concluded that the BLEU score may be more suitable as an evaluation tool for character-level models.

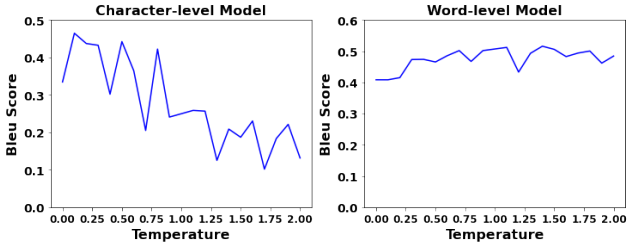

Figure 4: BLEU scores for varying levels of Temperature

4. Autoencoders

In this section we aim to reduce the representation of the MNIST dataset to 10 latent dimensions. This is done using 3 methods, Principal Component Analysis(PCA), Linear Autoencoder(AE) and a Convolutional Autoencoder(ConvAE). The AE is designed as following: **D(256), D(512), D(2048), D(1024), Rep(10), D(1024), D(2048), D(1024), Reshape(32,32,1))**, and the ConvAE: **(C(128), MP, C(256), MP, C(512), MP, C(1024), Rep(10), D(2048), D(2048), D(1024), Reshape(32,32,1))** (See Appendix for clarification on notation 7.4.1). Each dense layer in both models has ReLU activation following it, and for each Convolutional layer a (3x3) filter size is used with a stride of 1, and padding such that output dimensions are consistent with the layer prior.

Model	MSE train ‖ val.	Accuracy(%) train ‖ val.
PCA	.0258 ‖ .0256	81.04 ‖ 81.52
Autoencoder	.0064 ‖ .0078	93.73 ‖ 93.97
Conv. Autoencoder	.0069 ‖ .0081	94.56 ‖ 94.63

Table 4: Performance of each model on dimension reduction of cifar10

The ConvAE and the AE show significantly better performance compared to PCA. PCA is limited in the sense that it is a closed form linear projection onto the latent space, the AE and ConvAE allow for a learnt, non-linear projection onto the latent space, and thus allowing for a closer representation of the higher dimensional data. The AE shows very similar performance to the ConvAE, this indicates that for this task, little/nothing is gained by using spatial information as the basis for latent representation.

Custon Loss Functions: We now aim to denoise the TINet dataset with the same baseline UNet model using the Mean

Absolute Error(MAE), Structural Similarity Index(SSIM) and $1/$(Peak Signal-to-noise Ratio),referred to as PSNR. The test MSEs for each of the loss functions are shown in Table 5, along with the denoised images in Figure 5. We actually take 1-SSIM for the loss as this turns it from a maximisation to a minimisation problem.

Loss	Test MSE
MAE	.0056
SSIM	.0074
1/PSNR	.0055

Table 5: Table showing performance of UNet model trained with different loss functions

Figure 5: Denoised images using different loss functions

In terms of the MSE, PSNR and MAE losses perform most similarly, with SSIM performing slightly worse. Visually however it seems as though the SSIM and MAE look most similar, with the PSNR showing a slightly sharper image with less blurring. Overall however, the loss function has little effect on the performance of this task.

5. VAEs and GANs

A Variational Autoencoder(VAE) with and without KL divergence, and a Generative Adversarial Network(GAN) are trained on the MNIST dataset and are used to generate new examples. The Mean Squared Error(reconstruction error) and Inception Score(IS) for each of the models is shown in Table 6.

Model	MSE	Inception Score(IS)
VAE with KL	0.0117	7.23
VAE without KL	0.0109	5.52
GAN	N/A	8.29

Table 6: Table comparing performance of different models on MNIST data generation

The Inception Score in essence is a measure of two properties in the generated data, whether there is variety, and whether they can be distinctly classified[3]. The KL Divergence acts as a regularisation term that stops the model from simply minimising the reconstruction error. It instead encourages **continuity** between the latent and output space (similar latent value leads to similar output), as well as **completeness**(output can be distinctly classified)[4]. This regularisation will naturally increase MSE(reconstruction error) but instead will improve IS, which is observed in Table 6. With a better IS score than both VAE models, the GAN clearly has better capabilities of producing varied yet distinct data compared to a VAE.

Colouring B&W images: We next attempt to colour Black-and-white images using a cGAN and a UNet Autoencoder with MAE loss(MAE Model), the results of 5 predictions are shown in the appendix(Figure 10). The MAE model is trained to only minimise reconstruction error, thus it tends to make 'safe' predictions, it performs well in the mean sense (MAE of .0442), but qualitatively, the results are poor, the images seem to have very little colour. The cGAN however is playing a min-max game between a generator and discriminator, the discriminator plays the role of a human by attempting to differentiate between 'real' and 'fake' data, the cGAN hereby trains itself to generate data that resembles the real data, which is confirmed in the generated images. When the real truck is red, the GAN generates a blue truck, it will receive a poor reconstruction score, but qualitatively, it performs well, as images look more realistic. This is reflected in the fact that the cGAN has a MAE of .0456, higher than than the MAE model.

6. Reinforcement Learning

Figure 6 shows the plot for the average reward in the past 50 episodes for 4 different Reinforcement Learning strategies to tackling the CartPole problem; Q-Learning with ϵ greedy policy, Q-Learning with Softmax policy, SARSA with ϵ greedy policy and SARSA with Softmax policy.

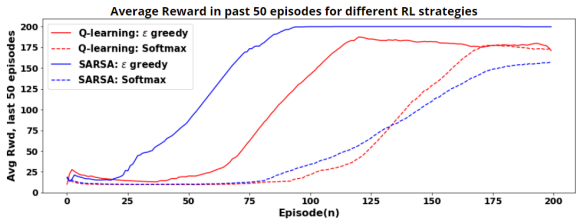

Figure 6: Average episode reward for each Reinforcement Learning Strategy

The adjustments made to implement each strategy is explained in Section 7.6. SARSA is an **on-policy** method, exploration is done using the same policy we are improving, hence it directly improves the ϵ-greedy policy at each iteration, why it converges the fastest. Q-learning uses an **off-policy** which learns the optimal policy whilst exploring a new one, this causes slightly convergence with ϵ greedy policy, but offers a more robust approach to learning, thus performing better with softmax policy. The softmax policy aims to learn the degree of exploration by sampling actions from a probability distribution according to the performance of actions in the state-action space, the higher its Q-function, the more likely it is to be sampled. The temperature in softmax(See equation 6), can be used to tune the degree of exploration, it has the same effect as explored in Section 3 for text generation, and is synonymous to ϵ, they both are used to control exploration in their respective policies. In this particular environment, with the selected values for these 2 parameters, softmax policy converges slower than ϵ greedy policy. It must be noted that, results for each strategy vary from experiment to experiment due to a high dependence on the initial state of algorithms.

7. Appendix

7.1. Appendix for Section 1

7.1.1 Baseline CNN Model

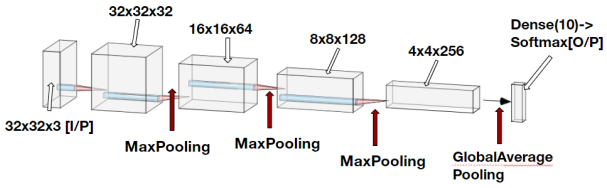

Figure 7: Baseline CNN model[1]

Note that ReLU activation and a (3x3) filter size are used in each of the convolutional layers.

7.1.2 Loss curves for different augmentation techniques

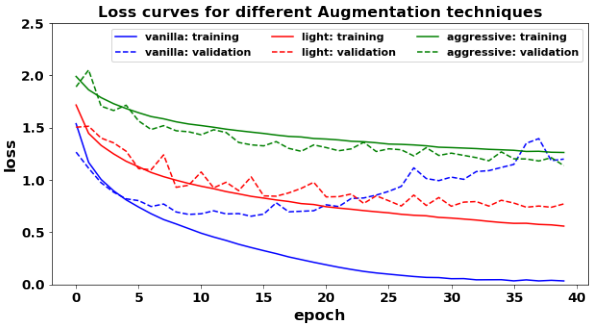

Figure 8: Training and Validation Loss curves for different data Augmentation techniques

7.1.3 Dying ReLU for zeros kernel initialisation

The model label ground truth for the kth label is denoted by y_k where in $k \in 1, 2, .. , 9, 10$ in this example. The prediction for that label is denoted by $\hat{y_k} \in [0, 1]$, which acts as a certainty of it being a certain label. $\mathbf{W_k}$ denotes the weight matrix of the final layer and $\mathbf{x_k}$ denotes the vector of outputs from the ReLU activation function of the previous layer. The softmax activation function is used for the final layer neurons of classification tasks, and in this task. Such that:

$$\hat{y_k} = \frac{e^{\mathbf{W}_k \mathbf{x}_k + \mathbf{b}}}{\sum_i e^{\mathbf{W}_i \mathbf{x}_i + \mathbf{b}}}$$

For zeros kernel initialisation, $\mathbf{W}_k = 0, \mathbf{b} = 0$, thus,

$$\hat{y_k} \approx \frac{e^0}{\sum_i e^0} = \frac{1}{10} \quad \text{for all k.}$$

Since each label has the same probability of being true, guesses are essentially being made randomly, hence why there is an accuracy of approximately $\frac{1}{10}$. When the output of the ReLU is 0, its gradient is also 0, this is the case for every neuron in the network, hence, the network cannot learn.

7.2. Appendix for Section 2

Model	Top-1 Accuracy(%)	No. Parameters
VGG16	70.5	138 million
EfficientNet B0	77.3	5.3 million

Table 7: Table comparing ImageNet performance of VGG16 and EfficientNetB0

7.3. Appendix for Section 3

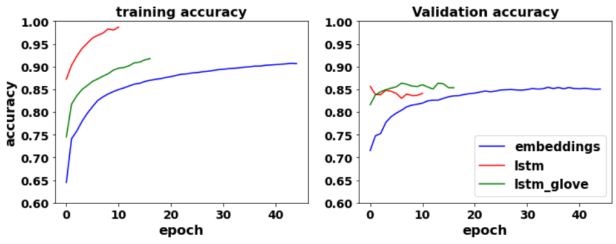

Figure 9: Training and Validation Accuracy curves for ImDB review sentiment prediction for 3 different models

7.3.1 'phrase 1' and 'phrase 2'

- phrase 1: the movie is boring and not good - negative sentiment
- phrase 2: the movie is good and not boring - positive sentiment

7.3.2 10 most similar words to '8'

Embeddings	GloVE LSTM
excellent	9
highly	7
wonderful	6
perfect	5
recommended	4
9	12
7	3
superb	10
favorite	16
today	13

Table 8: 10 most similar words to '8' for trained embeddings and preloaded GLoVe embeddings(from top to bottom)

7.3.3 Relation of prediction and Temperature

The probability p_i represents the probability of the element i being output by the RNN model prior to temperature 'smoothing', with temperature T. The probability of this element being output by the RNN becomes $\hat{p}_i$ such that:

$$\hat{p}_i = \frac{exp(log(p_i)/T)}{\sum_j exp(log(p_i)/T)} \quad (1)$$

We examine the case of $p_1 = 0.9$, $p_2 = 0.05$, $p_3 = 0.05$ and $T = 0.01, 2$, to see its effects. For $T = 0.01$:

$$\hat{p}_1 = \frac{e^{log(0.9)/0.01}}{e^{log(0.9)/0.01} + e^{log(0.05)/0.01} + e^{log(0.05)/0.01}}$$
$$\approx 1.00$$

Now for $T = 2$:

$$\hat{p}_1 = \frac{e^{log(0.9)/2}}{e^{log(0.9)/2} + e^{log(0.05)/2} + e^{log(0.05)/2}}$$
$$\approx 0.8$$

It can be seen for Temperature close to 0, the most likely prediction becomes even more probable, this means that the 'safest' option will almost always be taken. For a larger temperature we can see that the most probable prediction is given less probability to be taken, this means that the model is encouraging for more stochastic predictions to be made.

7.4. Appendix for Section 4

7.4.1 Network Notation

- D(N): refers to a dense layer of N fully connected neurons.
- Rep(N): refers to the representation layer of latent dimension N, which is simply a dense layer with 10 neurons.
- Reshape(X,Y,Z): refers to a tranformation of the vector output from the previous layer into a matrix of dimension $X \times Y \times Z$.
- C(N) refers to a 2D convolution layer of depth N.
- MP: refers to a MaxPooling layer of pool size (2×2).

7.5. Appendix for Section 5

Figure 10: Colouring B&W images using an Autoencoder, and cGAN approach

7.6. Appendix for Section 6

7.6.1 Implementing SARSA

To make the changes we first observe how updates to the Q function are made for each method, this is where they differ.
SARSA:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (2)$$

Where a_t is sampled from $\pi_{Q,\epsilon}(a|s_{t+1})$.
Q-Learning:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \text{max}_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t)) \quad (3)$$

In the code given, $\alpha = 1$, therefore: **SARSA**:

$$Q(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \quad (4)$$

Q-Learning:

$$Q(s_t, a_t) = r_{t+1} + \gamma \text{max}_{a' \in \mathcal{A}} Q(s_{t+1}, a') \quad (5)$$

It is clear now that the only change that needs to be made is to the correction term (term multiplied by γ). In Q-learning, this value is taken as the maximum Q-function value in the action space at state s_{t+1}, whereas in SARSA, instead of taking the maximum, we sample the new action at s_{t+1}

from the current policy, $\pi_{Q,\epsilon}(a|s_{t+1})$, and obtain the estimate of the Q function at s_{t+1}, a_{t+1}. This is done by computing a forward pass in the network to obtain the Q functions at s_{t+1} for the whole action space, and then selecting the Q function of the sampled action, a_{t+1}, that was sampled using the policy already in place.

7.6.2 Implementing Softmax Policy

SoftMax function is defined in Equation 6, where T is the temperature, and in Section 6 has the value 0.025.

$$p_{a_k} = \frac{exp(Q(s, a_k)/T)}{\sum_i exp(Q(s, a_i)/T)} \tag{6}$$

To implement the SoftMax policy we compute a forward pass in the network at the current state to obtain all the Q-functions in the action space at the current state. The vector holding these Q-functions is then input as the argument of the SoftMax function. This returns a probability distribution where the actions corresponding to the highest Q-function values, have the highest probability of being sampled. We then sample from this distribution to obtain the action for the current state.

References

1. https://alexlenail.me/NN-SVG/AlexNet.html

2. https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

3. https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a

4. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf