Boon Liang Wong, blw219, 01700213

3. Network Training

Model	Best Validation Acc (%)
No Data Augmentation	79.63
Data Augmentation 1 (Less Aggressive)	79.05
Data Augmentation 2 (More Aggressive)	77.57
Baseline + Dropout	80.79
Baseline + BatchNorm	68.31
Baseline + Dropout + BatchNorm	81.21
Zeros Initialization	10.00

Table 1. Best Validation Accuracy for Different Models

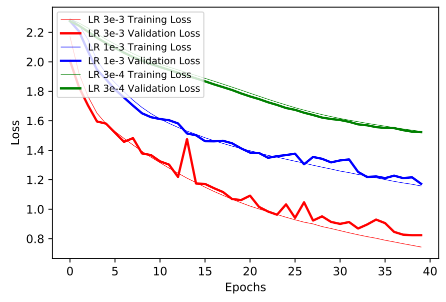


Figure 1. SGD with Different Learning Rate

Task 3.1: Data augmentation strategy used for Data Aug 1 (Less Aggressive) is RandomRotation with factor k=0.1. For Data Aug 2 (More Aggressive), RandomRotation(0.15), RandomTranslation(height_factor=0.3, width_factor=0.3) and RandomFlip("vertical") are used in combination to modify the data which yield the curves in figure 7. In table 1, the top 3 rows illustrate the best validation accuracy for each strategy. In figure 7, baseline model exhibits strong overfitting as training loss continues to decrease but validation loss increases after 10 epochs. In comparison, the gap between losses is smaller in model with Data Aug 1, signifying that soft augmentation can reduce overfitting and regularise the model to generalise better. But in model with Data Aug 2, aggressive data aug causes the data to differ from ground truth, decreasing the accuracy of the model. This model has strong overfitting to over-modified data as well as larger gap between losses.

Task 3.2: Last 4 rows in table 1 shows that Dropout+BatchNorm model has the highest accuracy followed by Dropout and BatchNorm model. In Dropout only model, Dropout layers of rate=0.15 are placed in every layer between ReLU and pooling. As shown in figure 8, these dropout layers increase accuracy and reduce the generalisation gap between losses when compared to batch normalisation. During training, the gap can be further reduced by higher dropout rate but this causes slower convergence. Whereas for BatchNorm only model, BatchNorm are placed in every layer between conv2D and ReLU. BatchNorm enables faster convergence of model but it leads to unstable curve with low accuracy. Too many layers of BatchNorm shows strong regularising effect that causes underfitting as shown in figure 8 where accuracy did not improve after 10 epochs. In Dropout+BatchNorm model, Dropout(0.2) layers are used after every ReLU layers but BatchNorm are only placed before ReLU for last 2 layers of Conv2D. Lesser layers of BatchNorm with correct placement can act as optimal regulariser that reduces gap between losses and allows faster convergence with higher accuracy. Lower dropout rate is also needed to complement with BatchNorm effect. **Task 3.3**: Zeros for kernel initialization causes vanishing gradient that deters the ability to learn during back-propagation, producing a mere 10% accuracy.

Task 3.4: Figure 1 shows losses for SGD with different learning rates. LR=3e-3 is the rate with smallest loss because higher rate leads to faster convergence. However, it also leads to higher fluctuations in validation loss due to its fast learning nature. Whereas in 1e-3 and 3e-4 rate, it is observed that the lower the rate, the less steep the losses. Therefore, the slower the decrease of loss, the smoother the curves. It is likely that all of them will converge to a global minimum with more epochs. To achieve a faster and optimal result, learning rate scheduler can be used.

4. Common CNN Architectures

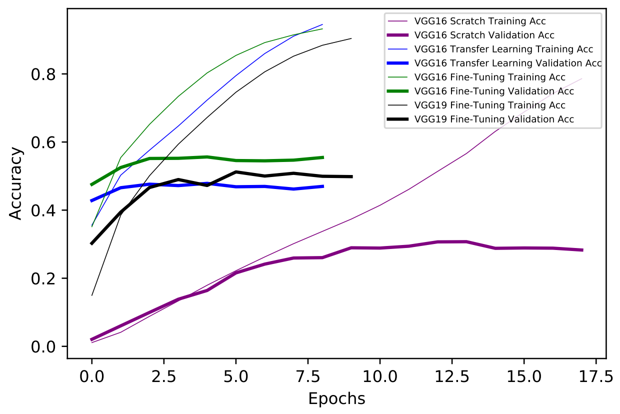


Figure 2. VGG16 Models + Self VGG19 with **Early Stopping**

Model	Test Acc	Train Time	Epochs	Inference
VGG16 Scratch	30.54%	62.88 s/epochs	18	0.4106ms
VGG16 TL	46.99%	26.55 s/epochs	9	0.5357ms
VGG16 FT	55.61%	62.44 s/epochs	9	0.4063ms
VGG19 FT	50.54%	72.91 s/epochs	10	0.4708ms

Table 2. Models performance on **GPU Tesla P100-PCIE**

Task 4.1: Figure 2 & Table 2 summarised the performance of VGG16 of different training methods with 64x64 input size. VGG16 Training from Scratch has the lowest test accuracy with longest total training time. This model is trained with random initialisation that requires more computation and epochs in order to converge. While VGG16 TL is trained with shortest time and converged fastest with moderate accuracy as it only requires training for dense layers with other layers being frozen. VGG16 FT has the best performance because FT is trained on top of pre-trained weights from a larger and similar dataset that provides good initialisation. VGG16 FT training time is between scratch and TL as all layers must be fine tuned/modified for specific dataset. Therefore, more features specific to this dataset are explored, producing highest accuracy. To conclude, with pre-trained weights like TL and FT, it reduces overall training time. This is further proven in figure 2 where it is observed that TL and FT have faster convergence and stopped earlier. Inference time for 3 models are close to each others due to similar architecture and same computation power.

Task 4.2: VGG19 with FT is selected to explore this dataset as it converges relatively fast with reasonable accuracy. Last row of Table 2 shows the results of this model where test accuracy is close to VGG16 FT. Even though VGG19 has 3 extra conv layers but the results are not improved by it. These extra layers did not explore any useful features in first 20 epochs, but it is highly likely that results can be improved with more epochs. Its training time and inference time are also slightly longer per epochs given slightly more complicated architecture. This concludes that inference time depends on network size and complexity of architecture. Side note, InceptionResNetV2 is also explored but the training time is too long for similar accuracy results.

5. RNN

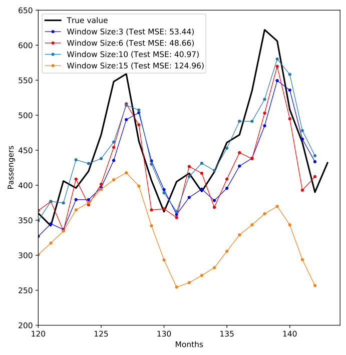


Figure 3. Regression Curve with Different Window Sizes

Model	Test Acc	Negative Review	Positive Review
Embeddings	84.93%	0.2894	0.2894
LSTM	82.91%	3.889e-05	0.004875
LSTM GloVe	86.08%	0.002382	0.7328

Table 3. Test Acc Sentiment Predictions for Different Embeddings

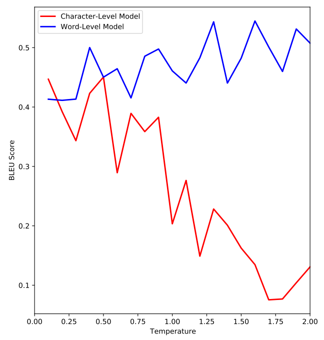


Figure 4. BLEU Score for Character and Word Level Models

Task 5.1: Figure 3 shows regression curve of different window sizes with MSE test results in the legend. Window Size of 10, has the lowest test MSE of 40.97 followed by window size of 3, 6 and 15 with test MSE of 48.66, 53.44 and 124.96 respectively. It is observed that the prediction becomes closer to the true peak when the window size increases until a certain size. These prediction curves also have slight delay in time compared to true value. As we know window size determines the length of past data into the network. This implies that by considering only the last 10 months values (window size=10), the network predicts more accurately relative to other window size. This is because window size that is too large leads to overfitting of past values during training and poorer generalization in testing, vice versa for window size that is too small.

Task 5.2: Figure 9 shows the training and validation curve for text embeddings. It can be seen that LSTM GloVe has the highest test accuracy followed by Embedding and LSTM. The embedding model converges the slowest with moderate test accuracy. While LSTM and LSTM GloVe models exhibit strong overfitting as compared to embedding model. Although LSTM model has lowest test accuracy but it converges the fastest. When GloVe is included in LSTM, the test accurracy is improved with longer convergence. Table 3 summarises the test accuracy and prediction score of each model. In the prediction assessment, all models are tested against the statement, ”the movie is good and not boring” and ”the movie is boring and not good” where low score indicates negative sentiment and high score indicates positive sentiment. In table 3, LSTM GloVe predicts positive statement and negative statement accurately with appropriate scores. Whereas LSTM model predicts both statements as negative sentiment with high confidence. Embedding model predicts both statements with slightly higher score but gives both the same score because it is lack of RNN and LSTM components to track the order of words, hence changing the order of words in statement does not impact the score output. Figure 10, 11 and 12 show the results of each model towards query word of 8. Since embedding model is trained with IMDB dataset, query word of 8 gives results of a list of compliments in movie comments. While GloVe embedding is trained with variable datasets, the query word of 8 will be paired with numbers closest to it like 7, 9.

Task 5.3: Figure 4 illustrates the BLEU scores for different temperature values. Temperature value around 0.3 is the best for both models. To summarise, the higher the temperature value, the lower the character-level model BLEU score, while world-level model score is relatively flat and stable. The higher the temperature value, the more variant the output. Therefore, character-level model gives out more spelling errors when temperature increases as it forms sentence character by character. Word-level model avoids this completely as it outputs word by word. When temperature value is close to 0, it is observed that both models generate repetitive grammatically correct sentences. These sentences vary as temperature value increases. Overall, word-level model is a better choice.

6. Autoencoders

Models	Train Acc	Val Acc	MSE Train	MSE Val
PCA	80.99%	81.53%	0.0258	0.0256
Non-CNN	82.25%	82.80%	0.0227	0.0225
CNN	92.12%	92.48%	0.0111	0.0112

Table 4. Results for Classifier with Different Representations

Loss Functions	MSE Training	MSE Test
1/PSNR	0.0070	0.0071
MSE	0.0059	0.0061
MAE	0.0057	0.0058
Correntropy	0.0057	0.0062

Table 5. Loss Functions MSE

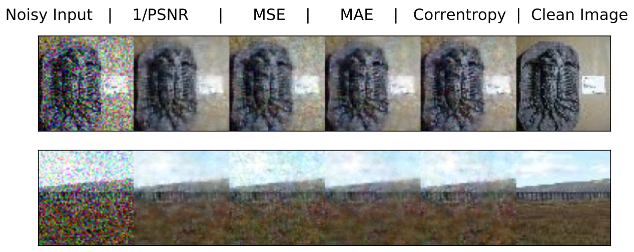

Noisy Input | 1/PSNR | MSE | MAE | Correntropy | Clean Image

Figure 5. Loss Functions Effect on Denoised Images

Task 6.1: The architecture for **Non-CNN Autoencoder** is $Input \rightarrow FC(128) \rightarrow FC(10) \rightarrow FC(128) \rightarrow [FC(32*32) \rightarrow ReLU] \rightarrow Output$. For **CNN Autoencoder**, the architecture is $Input \rightarrow [Conv2D(32) \rightarrow RELU] \rightarrow [Conv2D(64) \rightarrow RELU] \rightarrow [Conv2D(128) \rightarrow RELU] \rightarrow FC(128) \rightarrow RELU \rightarrow FC(64) \rightarrow RELU \rightarrow FC(10) \rightarrow RELU \rightarrow [Conv2D(64) \rightarrow RELU] \rightarrow [Conv2D(128) \rightarrow RELU] \rightarrow [FC(32*32) \rightarrow ReLU] \rightarrow Output$. The kernel size is 3x3 with strides of 2x2 and padding=same.

During training, it is observed that architecture with linear activation performs relatively poor no matter how deep the architecture is. Therefore, in both proposed non-CNN and CNN architectures, RELU activations are used to introduce non-linearity in the architecture to assist the training. As a result, both non-CNN and CNN perform better than PCA showed in table 4. In Non-CNN architecture, only 1 dense layer used RELU but the accuracy is improved. Whereas CNN architecture has RELU on every layers including Conv2D, therefore, the accuracy went up to 92%. Apart from RELU, use of convolution layers is a main factor to the results because Conv2D layers are more efficient in feature extraction in images/patterns so autoencoder gets to learn more useful features and produces meaningful representations.

Task 6.2: Table 5 shows the results for different loss functions and how it can affect the performance of autoencoder. For denoising autoencoder, a range of loss functions such as 1/PSNR, MAE, MSE and Correntropy were used to determine the best loss function for the task. In figure 5, we can visualise the effect of custom loss functions. Based on Table 5, MAE performs the best numerically where loss function improves results by changing squared terms in MSE to absolute difference. However, I believe that in figure 5, Correntropy loss function produces clearest denoised images among others since the difference in MSE error is almost negligible. As investigated in paper [1], Correntropy is insensitive to outliers and has better generalization performance, therefore, it focuses on denoising the noise closer to mean which is excellent for this task.

7. VAE GAN

Model	MSE Test	Inception Score
VAE with KL (dim 10)	0.0118	7.31
VAE without KL (dim 10)	0.0059	5.90
GAN (dim 10)	0.0057	8.27

Table 6. MSE & Inception Scores for VAEs and GAN

Task 7.1: In table 6, VAE wihout KL has better MSE results than VAE with KL but the inception score for VAE without KL is lower, implying that the quality of generated data is lower than VAE with KL. Therefore, we can conclude that KL divergence plays a huge role in improving the generated data. Essentially, KL divergence loss function encourages similar resentation for each input by minimising the divergence of probability distribution from desired distribution. It acts as a penalty/regularisation term that helps the latent space representation to be close to standard normal distribution, preventing it from producing incomplete and discontinuous data.

However, GAN outperforms VAE models with highest inception score. The generated images are much more sharper but it is harder to train in terms of training time. GAN consists of both generator and discriminator networks, where discriminator contributes enormously in improving generator through min/max game and generator is forced to create samples that can confuse powerful discriminator which explains its high inception score. However, the observed generated data are less diverse compared to VAE. This is because GAN experiences mode collapse which can be seen from consistent oscillations in the loss plot of generator model. To beat the discriminator, the generator model finds a type of data that is easily able to fool the discriminator and thus keeps generating that one type. The entire generator model will over-optimize on that distribution of data because there's no incentive to explore other distributions. Last but not least, since inception score is still high, we can say that the quality of generated data outweights the variant of data as the metric evaluates both aspects.

Task 7.2: Figure 13 shows MAE scores and generated images of both MAE and cGAN models. In the first row, MAE model achieves lower MAE errors than cGAN model. For the rest of the rows, MAE model is still very comparable with cGAN numerically. The generated visual images say otherwise; cGAN generated images are significantly better than MAE models visually. This indicates that MAE model might not be a better model even though the quantitative results are excellent. As aforementioned, cGAN/-GAN model has to fool a strong discriminator during training, this means that even though some output might have higher MAE error but it still managed to fool the discriminator, producing more reliable visual output despite being far from ground truth numerically. Since the task is recolourisation of image, MAE model approach of predicting RGB image using mean absolute error loss will only produce images that are near ground truth numerically in L1 sense. Therefore, MAE generated images are less coloured compared to cGAN. Hence, cGAN approach to predict the RGB pixel-wise values of a B&W image together with discriminator will produce a better qualitative results.

8. Reinforcement Learning

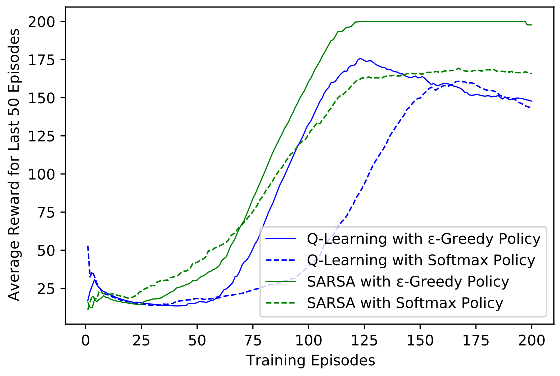


Figure 6. SARSA & Q-Learning with Different Policies

Task 8: Figure 6 illustrates the average reward for last 50 episodes of Q-Learning and SARSA with ϵ-greedy and Softmax policies. Appendix F shows key modifications to act and replay from the DQNAgent to implement different approaches. From figure 6, Q-Learning & SARSA with ϵ-greedy policy have better performance and converge faster to its peak value of reward when compared with softmax policy (temperature=0.025). To explore further about on/off policy, it is observed that SARSA with ϵ-greedy policy has the best results in terms of both convergence and stability in learning where reward remains at 200 when trained for further episodes. This is because both SARSA's (on-policy) acting and updating policy are ϵ-greedy, unlike Q-Learning (off-policy), its acting policy is ϵ-greedy but updating policy is greedy. On policy like SARSA is particularly useful when optimizing the value of an agent that is exploring. Whereas off-policy like Q-learning is generally more explorative, so it will suffer more in divergence from optimal policy. Q-learning (and off-policy learning in general) also has higher per-sample variance than SARSA, one of the reasons for slow convergence. To conclude, Q-learning learns the optimal policy directly when exploring, whilst SARSA learns a near-optimal policy by optimising it whilst exploring. So if conversative method is preferred, SARSA should be used instead of Q-Learning.

For softmax policy(0.025), we can see that it does not perform as good, however, in figure 14, it is observed that temperature value in softmax greatly affects the performance of both SARSA and Q-Learning. As learned in RNN task, higher temperature means more variant, making it more explorative, speeding up the convergence to find the optimal policy. However, high temperature value comes with a cost of being unstable in further episodes, there's large probability that the model might diverges from optimal policy. To explain further, in ϵ-greedy policy, agent goes into random action with probability ϵ, the available random actions are then chosen uniformly (equally good). In reality, certain random actions are worse than others. Softmax policy helps to select these random actions with probabilities proportional to their current values. The idea is good but Softmax policy might still fall away from optimal because it might focuses too much on the current best value (so setting good temperature value is instrumental in achieving good results with fast convergence).

References

[1] J. C. Principe and A. G. Singh. A loss function for classification based on a robust similarity metric. 2010.

Appendix

A. Data Augmentation Effect on Losses

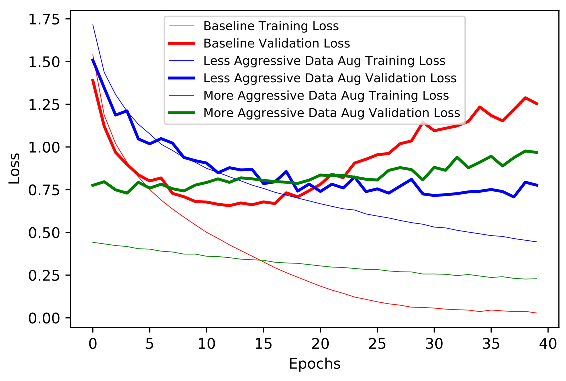


Figure 7. Losses for Different Data Augmentation Strategies

B. Effect of Dropout and Batch Normalization

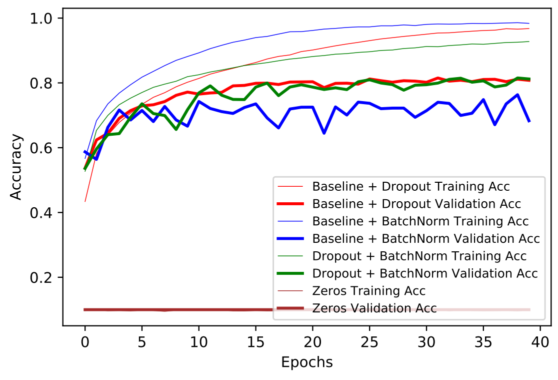


Figure 8. Accuracy Plot to investigate the effect of Dropout, Batch-Norm Zeros

C. Training and Validation Accuracy Curve for Text Embeddings

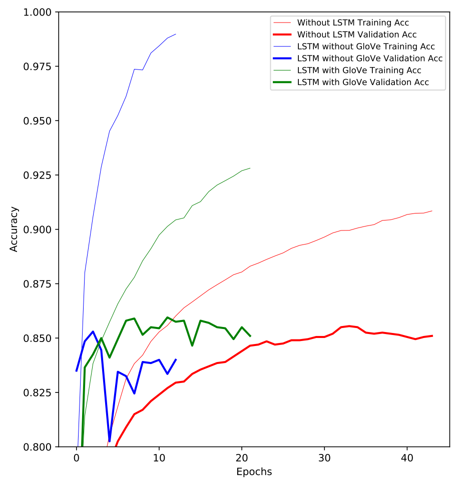
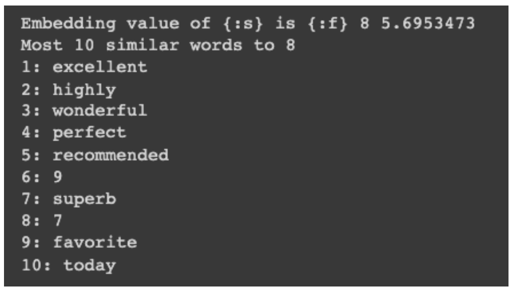


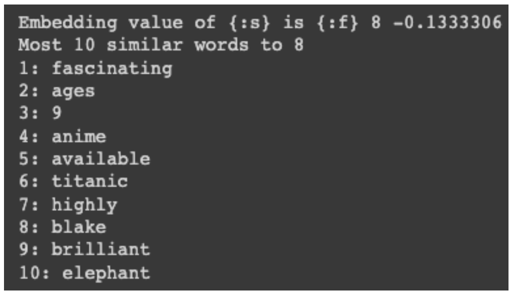
Figure 9. Training and Validation Accuracy Curve for Different Text Embeddings

D. Text Embeddings



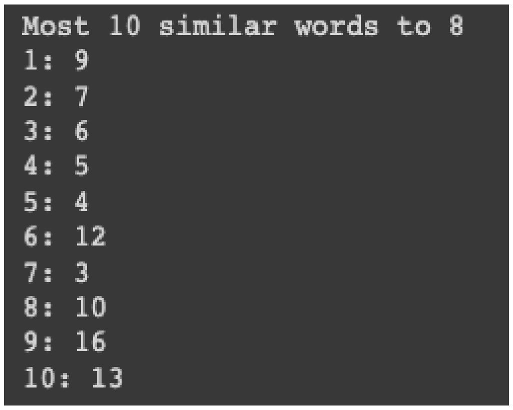
```
Embedding value of {:s} is {:f} 8 5.6953473
Most 10 similar words to 8
1: excellent
2: highly
3: wonderful
4: perfect
5: recommended
6: 9
7: superb
8: 7
9: favorite
10: today
```

Figure 10. Embedding Model results for Query Word=8



```
Embedding value of {:s} is {:f} 8 -0.1333306
Most 10 similar words to 8
1: fascinating
2: ages
3: 9
4: anime
5: available
6: titanic
7: highly
8: blake
9: brilliant
10: elephant
```

Figure 11. LSTM Model results for Query Word=8



```
Most 10 similar words to 8
1: 9
2: 7
3: 6
4: 5
5: 4
6: 12
7: 3
8: 10
9: 16
10: 13
```

Figure 12. GloVe LSTM Model results for Query Word=8

E. MAE vs cGAN

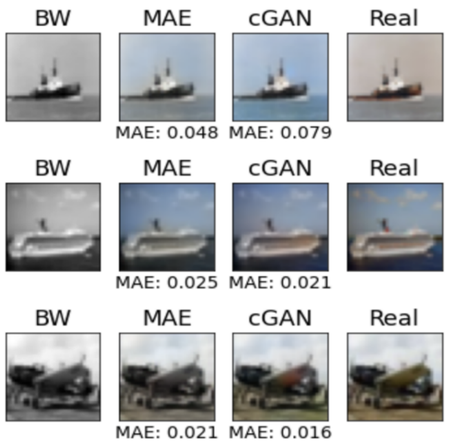


Figure 13. MAE vs cGAN with MAE Scores and Visual Images

F. Key Modifications to Act and Replay from DQNAgent

Q-Learning with ϵ-greedy Policy:
No modification because implementation is given.

Q-Learning with Softmax Policy:
$actfunction$: ϵ-greedy is removed and predicted act_value is passed into softmax, returning the highest probability of random action using argmax method

$replayfunction$: ϵ decay is removed

SARSA with ϵ-greedy Policy:
$actfunction$: use the same ϵ-greedy implementation

$replayfunction$: Target variable in implementation is calculated as prediction using the column of next action selected by ϵ-greedy. Example, $target = reward\_b + self.gamma * np.array(self.model.predict(next\_state\_b))[:, next\_action]$

SARSA with Softmax Policy:
$actfunction$: ϵ-greedy is removed and predicted act_value is passed into softmax, returning the highest probability of random action using argmax method

$replayfunction$: Like SARSA with ϵ-greedy Policy

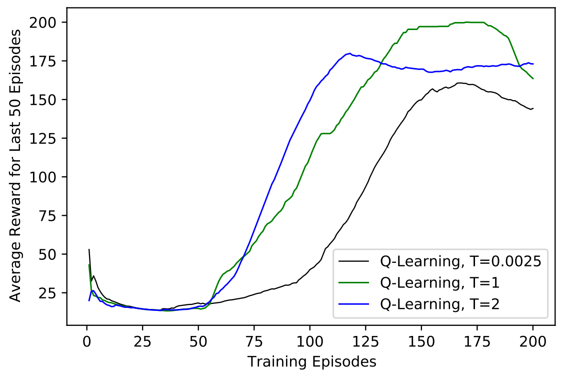


Figure 14. Q-Learning with Softmax Policy of Different Temperature Values