ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE SCHOOL OF LIFE SCIENCES

Master project in Life Sciences and Technology

A Primer on the Delayed Adversarial Attack in Using Recurrent Neural Networks for Reinforcement Learning

Done by

Hang Yuan

Under the direction of

Dr. Mathieu Salzmann
Computer Vision lab

Dr. François Fleuret
Idiap

EPFL

Lausanne, June 20, 2019

Abstract

In this master project, we proposed a new class of adversarial attack strategies called delayed attack for deep recurrent neural networks. The delayed attack focuses more on the temporal dimension of the attack effect which is often overlooked in the existing literature. Its main goal is to modify some of the input by an innocuous amount in the earlier time step whose adversarial effects do not appear until some time later. We extended the general delayed attack formulation into reinforcement learning, natural language processing and a synthetic toy experiment and evaluated its effectiveness in natural language processing and the toy experiment. We were able to perturb the output in the distanced future by changing the input at an earlier time but we did not manage to delay the adversarial attacks in our experiments.

Contents

Introduction

Reinforcement learning (RL) has become one of the subfields where deep learning (DL) helps advance the most. AlphaGo (Mnih et al., 2015), OpenAI Five (OpenAI, 2018) are examples of complex tasks where machines already outperform humans. RL also has lots of potential for real-world application, especially in robotics. However, before we deploy these powerful machinery in reality, we need to better understand the potential risks involved. The reason being many of these breakthroughs depend on deep neural networks (DNNs). DNNs have been shown to exhibit various weaknesses when facing adversarial attacks (AAs) such as fast gradient sign method (FGSM) (Goodfellow et al., 2015) and forward gradient attack (Papernot et al., 2016a) where even some seemly innocuous perturbation in the input can lead to a radical difference for the network output. Therefore, one can also exploit deep RL applications by utilizing DNNs' limitations.

AA has gained upward popularity from researchers in recent years. In fact, AAs have been an active topic for investigation since a long time ago and the most recent studies have been focused on their usage in deep learning applications which are becoming increasingly more prevalent in real-world and large-scale deployment, and thus more efforts have been given in this specific topic.

Most existing AA techniques concentrate on classification and sequence to sequence tasks, and thus the temporal dimension of the attack is overlooked. In this project, we are primarily interested in the feasibility of doing the attack in this dimension. More specifically, we would like to investigate a class of AAs which we call "delayed attack" (DA) . DA aims to insert the attack vector within a specific attack window, but the attack effects do not take place until some time later. In this project, we first formalize DA in the RL setting. This part of the analysis will have a concentration on the recurrent neural networks (RNNs) as RNNs allow the agent to have better long-term memory. Then we will demonstrate the feasibility of RNNs attack under the DA framework. Within the scope of the project, we did not directly apply DA in RL due to the need of a generative model to mimic the environment change for us to compute attack vector, instead, we tested out DA on a toy example and an NLP language model to understand whether DA is possible on RNNs. The exact rationale behind this will be given in the methodology section.

Background

We will first dissect the backbone of this project, RNNs. This is important because RNNs are used to train the agents in RL and are also our attack targets. Then, we will elaborate on how RL works and how RNNs are used for RL applications, after which, we will take a look at the attack strategies for both RNNs and RL. Lastly, we will briefly touch upon how our NLP modeling works because we will attack NLP as a validation for our attack strategy before we deploy DA in RL.

2.1 Recurrent neural networks

Even though feed-forward neural networks have been widely used in RL (Mnih et al., 2015, 2016; Schaul et al., 2016), when it comes to the tasks that require long-term temporal dependency in a partially observable environment, RNNs posse obvious advantages as the networks can store historical information in their states. It is worth noting that RNNs do not present systematic advantages over normal networks whose input at each time step is a stack of frames if the same length of history is given (Hausknecht and Stone, 2015). A vanilla RNN usually has the simple form, $y_t = f(h_{t-1}, x_t), h_t = g(h_{t-1}, x_t)$. In practice, vanilla RNN is not easy to use because it suffers from gradient explosion and vanishing gradient problems (Hochreiter, 1991). This is where Long Short Term Memory (LSTM) proposed in Hochreiter and Schmidhuber (1997) comes to rescue. A different but similar architecture, gated recurrent unit (GRU) has also been proposed in Chung et al. (2014). Often, we see LSTM is used because of its reliability as compared to other architectures. Even though different networks have a similar per-unit capacity, it is more due to the difficulty of training that makes the empirical performance difference (Collins et al., 2016). In this project, we will stay with LSTM for the sake of convenience. Here, we will formulate the LSTM version that we are using in this project.

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t_1}, x] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t_1}, x] + b_f) \\
g_t &= tanh(W_g \cdot [h_{t_1}, x] + b_g) \\
o_t &= \sigma(W_o \cdot [h_{t_1}, x] + b_o) \\
c_t &= f_t * c_{t-1} + i_t * g_t \\
h_t &= o_t * tanh(c_t)
\end{aligned}
$$

h_t is the hidden state at time t. c_t is the cell state at time t. i_t, f_t, g_t, c_t, o_t are the input, forget, cell and output gates. $*$ is element-wise product. The input and forget gates decide what new information to keep and forget. The cell state carries over historical information and the output state simply generates the desired signal at the current time step based on the cell state and output gate information.

It is exactly because that RNNs have the hidden cell state that makes it possible to induce DA under specific constraints. LSTM is trained using back-propagation through time (BPTT).

Note that in the training phase, the backwards unfolding does not reach the beginning of the whole input sequence due to computational complexity, but it should not impede our ability to attack. We are mainly interested in the attack during inference time, in which the forward pass can carry historical information from a long time ago as the forward pass does not reset cell state in the middle of one sequence.

2.2 Reinforcement learning

RL describes a system that tries to conduct the optimal actions in sequential decision making process (Sutton et al., 1998). Thanks to recent advances in deep learning (LeCun et al., 2015), DNN-based RL algorithms such as deep-Q-network (DQN) (Mnih et al., 2015) and asynchronous advantage actor critic (A3C) (Mnih et al., 2016) have surfaced. In this section, we will talk about the three major categories of RL methods and then elaborate on the different features that are commonly used in the state-of-art frameworks. The goal is to understand which kind of RL setup is most prone to DA, and set up the foundation why we are falling back to NLP attack in this project even though in the end we would like to do the attack for RL frameworks.

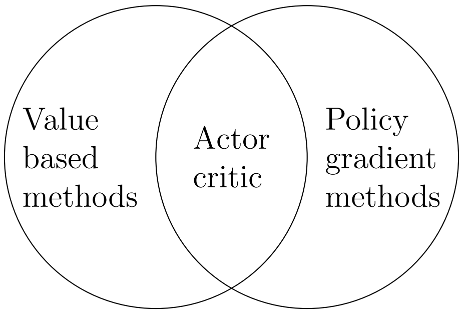


Figure 2.1: Reinforcement learning category relationship

As described in 2.1, deep RL has three types of algorithms. Actor-critic (AC) methods lay between value-based and policy gradient methods. AC estimates both the policy and value functions. Value-based methods only estimate value functions and have an implicit ϵ-greedy policy, whereas policy gradient methods do not have value function and only estimate the policy.

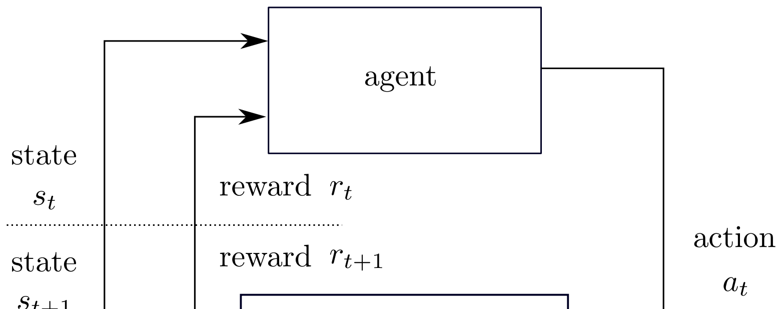


Figure 2.2: Markov decision process setup

In a RL problem, we typically have a Markov decision process (MDP) shown in figure 2.2. A game agent interacts with an environment to achieve a certain goal. The interaction happens at every discrete time $t = 1, 2, 3, ..., T$. The agent observes certain state of the environment $S_t \in \mathcal{S}$, selects some action $A_t \in \mathcal{A}$ and then receives certain reward $R_{t+1} \in \mathcal{R}$. In a finite MDP, $p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$. The expected reward can be computed by $r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$. The goal of a game is typically to maximize the return. The discounted reward can be framed by $G_t \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k}$. $\gamma \in [0, 1]$ is the discount factor. Having this in mind, let us see how exactly the major RL algorithm categories differ.

Value based methods This class of methods either tries to estimate the state-action function $Q^*(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ or the state value $V(s) = \mathbb{E}_\pi[G_t|S_t = s]$. Here, we will use Bellman equation to describe the update for the state-action function $Q^*(s, a) = \mathbb{E}_\pi[R_t + \gamma \max_{a'} Q^*(s', a'|s, a)]$. Given a neural network with parameter θ, and the target Q-value $R_t + \gamma \max_{a'} Q^*(s', a'|s, a)$ the update gradient on the loss function can be defined as:

$$\nabla_{\theta^-} L(\theta) = \mathbb{E}[(R_t + \gamma \max_{a'} Q(s', a'|\theta^-) - Q(s, a|\theta))\nabla Q(s, a|\theta)]$$

where θ^- is the old network parameter used to estimate the target value. This is what DQN is primarily based on. Furthermore, it also incorporates two other techniques to improve the performance, namely experience replay and the use of a second network to generate the target Q value.

Policy gradient methods Policy gradient tries to find out the best policy to follow. The policy function $\pi(a|s) \doteq Pr\{A_t = a|S_t = s, \theta_t = \theta\}$ where θ is the model parameter. The optimization of the function parameter can be described as $\theta_{t+1} = \theta_t + \alpha\nabla_\theta J(\theta)$. The update of policy gradient is based on the policy gradient theorem which states that

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a)\nabla\pi(a|s, \theta) \tag{2.1}$$

where μ is the on policy distribution of states. Essentially, the policy gradient is an weighted average of the Q value if π is followed. What we will end up having is called REINFORCE update (Sutton et al., 1998): $\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$. α is the coefficient for the update step. Moreover, if we add a baseline comparison to the update step, it will allow us to explore more states, which is called REINFORCE with baseline: $\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t))\frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$. The optimal choice for $b(S_t)$ is certainly the state value $v(S_t, W)$ where W is the model parameter.

Actor-critic : AC methods can best be understood as an extension for policy gradient methods. REINFORCE with baseline has both the state-action value as well as the policy gradient but it is still not yet an AC method because it is not using bootstrapping which needs one to approximate the state using the estimate for the subsequent state. Let us use an one-step horizon method to do this which can turn $G_t - v(S_t)$ into $R_{t+1} + \gamma v(S_{t+1,W}) - v(S_t, W)$.

Having introduced the major types of RL frameworks, we would like to also cover some important features that the state-of-art methods tend to use, namely experience replay and prioritized replay . Experience replay means that the agent will store a fixed repertoire of episodes in the first instance, and then relearn from the stored episodes.This has numerous advantages such increase of data efficiency and decrease of variance between each update (Mnih et al., 2015). Prioritized replay is an improvement on the experience replay, which turns the uniform sampling from the stored episodes into an ordered list of episodes based on their importance for learning (Schaul et al., 2016). This gives a chance to let the more informative episodes appear more often in the learning phase.

Source	Experience replay	Priority	Recurrent network
R2C2 (Kapturowski et al., 2019)	Yes	Yes	Yes
Ape-X (Horgan et al., 2018)	Yes	Yes	No
IMPALA (Espeholt et al., 2018)	Yes	No	Yes
UNREAL (Jaderberg et al., 2016)	Yes	Yes	Yes
A3C (Mnih et al., 2016)	No	Yes	Possible

Table 2.1: Comparison of different reinforcement learning frameworks.

In table 2.1, we summarize some of the popular RL frameworks and their corresponding feature situations. We also consider whether or not they utilize RNNs because only the ones that make use of RNNs are prone for the DA setup. In fact, both experience replay and prioritized replay are only used during training time, in inference time, regardless there training methods, the networks will be used in the same fashion.

2.3 NLP language modeling

Our coverage about NLP modeling will be relatively brief because our main interests lie in using a standard model to test out our DA framework but eventually DA is meant to be used in RL with RNNs. NLP language modeling is usually framed as an unsupervised distribution learning task with a set of examples and each sample consists of a list of symbols $x_1, x_2, \cdots, x_n$. Bengio et al. (2003) first tried to model a language by finding out its joint conditional distribution:

$$p(x_1, \cdots, x_n) = \prod_{i=1}^{n} p(x_n | x_1, \cdots, x_{n-1})$$

The trained unsupervised model can be used to execute various tasks such as language inference, sentiment analysis and summarization without direct supervised objective because the minimums for both the supervised and unsupervised objectives are identical. LSTM was initially shown to be able to generate realistic text in Graves (2013). More recently, many other methods have been developed to enhance the modeling outcome (Vaswani et al., 2017; Hassan et al., 2018; Radford et al., 2019)

.

Following Graves (2013), we let the predictive network take a list of symbols $\mathbf{x}$ and output a vector sequence $Y = (y_1, \cdots, y_n)$. Each y is a probability vector for the predictive distribution $Pr(x_{t+1}|y_t)$. The first element x_1 will be a randomly selected symbol, then for every time step afterwards, the input will be determined by a multinomial distribution based on the output from the previous time step. Therefore, the probability output for a given sequence will be

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_{i+1} | y_i)$$

The loss function will be the negative log:

$$\mathcal{L}(\mathbf{x}) = -\sum_{i=1}^{n} log\ p(x_{x+1} | y_i)$$

Our network input symbol can be modeled by a one-hot vector but also an embedding layer which has been reported to have good general properties across different language modeling tasks (Collobert et al., 2011). Another added advantage is that it will allow us to take derivative directly w.r.t. input when we are doing the attack. We will elaborate more on this in the methodology section. Assume we have K word classes. The network output will be a multinomial distribution. When being represented by a softmax function, it will become:

$$p(x_{t+1} = k|y_t) = \frac{exp(y_k^t)}{\sum_{i=1}^{K} exp(y_i^t)}$$

.

2.4 Adversarial attack strategies

The core of this project is AA, so in this section, we will talk about three different classes of attacks. The attack for DNNs set up an overarching foundation for the discussion. Then, we will dive deeper to review what AAs are like in both RL and NLP.

2.4.1 Attack for deep neural networks

AAs encapsulate the methods that can construct input perturbation which can guide the network to exhibit undesired behavior. Szegedy et al. (2014); Goodfellow et al. (2015) first led the discovery that hardly noticeable input changes can cause large differences in the deep network output. This can be quite troublesome in reality as more and more real-world applications are based on DNNs. Hence, it is important to understand the robustness of such models, before the actual deployments. Depending on the context of the attack, there exist many different kinds. To fully identify the type of the attack, one needs to answer the following questions (Gilmer et al., 2018):

- What is the goal of the attack? This comes in different forms. For instance, when dealing with a classifier, an attacker can try to guide the network to output a class label that is different from the true class or a specific class that the attacker wants. This former is called untargeted attack and the latter is called targeted attack (Papernot et al., 2016a).
- What information does the attacker know? We have mostly two types of attack modes here, white-box and black-box. For the white-box attack, we shall know the training, testing sets as well as information such as network architecture and training method. However, for the black-box attack, the available is much more limited, one is allowed to glean information by trying out different inputs and observing responses in this context.
- What constraint does the attack have? There are many ways that the attacks could be constrained. For example, one might require the attack to be indiscernible to humans. One could also limit the attack time allowed because the more often one attacks, the more likely it is for the attacker to get caught. Even for the extreme case, where no constraints are imposed at all, this situation is still meaningful because it can help us better understand the behavior of the system. Different constraints will pose different challenges for the problem formulation.

FGSM is the backbone for many AA applications (Goodfellow et al., 2015), as it is fast to compute, easy to implement and integrate with other attack methods. FGSM mainly makes use of the fact that the output activation will increase quickly with the dimensionality of the weight matrix even when the attack is subject to a max norm constraint. We will showcase this principle using a linear example: For an input example x, we will perturb the input by μ, $\bar{x} = x + \mu$. μ is subject to a max norm constraint ϵ. The dot product between the weight vector w of size n and $\bar{x}$ will become $w^T\bar{x} = w^Tx + w^T\mu$. If we let $\mu = \epsilon \text{ sign}(w)$, the increase is maximized under the constraint. This means that the output will increase by ϵmn if the mean of w is m. $||\mu||_\infty$ does not change with the size of the problem, nonetheless, the increase in output does change with n. This implies that for a high-dimensional problem, a small change in input can have a large impact on the output.

Now, let us put this view in a network. If θ is the network parameter, x is the input, y is the output, and $J(\theta, x, y)$ is the cost function during training. The max norm perturbation is simply:

$$\mu = \epsilon \text{ sign}(\nabla_x J(\theta, x, y)) \tag{2.2}$$

2.4.2 Attack for reinforcement learning

Many RL agents are based on DNNs, and thus they are also vulnerable to AAs. In order to attack an RL agent, we will need to modify our attack objectives based on the different RL frameworks that are being used. Huang et al. (2017) showed that by the mere use of FGSM, RL agents will likely to suffer from severe performance decline in standard RL frameworks like A3C and DQN. FGSM was used in this work to attack. To achieve the attack objective, one will need to find out the cost function $\nabla_x J_(\theta, x, y)$, the gradient w.r.t. the input. In RL, we have the policy $\pi_\theta : S \rightarrow A$ that outputs the relative importance of different actions during for next time step with model parameter θ. We assume that the max value in y represents the best action to take, and thus $J(\theta, x, y)$ is the cross-entropy between y and the distribution that maximizes the most-weighted action in y. In plain words, the cost function is the cross-entropy between the real output and an output distribution that has weight only in the best action. The attack occurs only in the test time, and the model training remains unchanged.

Lin et al. (2017) reported that the attack does not need to occur at every single time step and also it is also possible the direct an RL agent to a specific location in the environment using a series of action planning in the black-box setting. The former is referred as the time-limiting attack and the latter is referred as the enchanting attack. The time-limiting attack is able to reduce its attack window by looking for the best moments to attack. Specifically, Lin et al. (2017) defines the best attack moments to be the time when the ratio between the weight of the most favored action and the least favored action is the greatest. As for the enchanting attack, in a black-box setting, a state prediction model is trained and then via sampling-based action planning, an RL agent can be guided to a pre-designated state in the future.

How our attack strategy differs is that we are looking at the class of attacks whose effects do not appear until some time later. Most of the existing techniques attack all the time or attack at a critical time where a human observer can notice the change of behavior of the agent. For instance, in the time-limiting attack for Pong, the attack in imposed when the brick is about to hit the ball such that the brick will miss the ball. The successful attack for a well-behaving agent will almost always make it easy to someone to catch, however, for DA, at the time of the attack, a human observer should not notice any obvious change of the agent because it is only until some time later that the attack effects show up. We will explain this more in our methodology section.

2.4.3 Attack for NLP

One major difficulty in NLP attack is that it is hard to do an attack that is indiscernible for humans. For reading and comprehension (questions and answering) task, there are attempts to do this by paraphrasing, however, it remains a challenge to produce high-quality paraphrased adversarial text. Jia and Liang (2017) proposed an alternative which is to add adversarial text to the end of the original text. The additional text could be either similar to the original text or drastically different. Both attack modes, when being tested with a verity of models, successfully decrease the F1 score by a large margin, from 75% to 36% and 7% respectively.

As mentioned before, our objective mainly centers on the temporal aspect of such attacks. We need to look at the NLP attacks that are geared towards RNNs. To our best knowledge, there are not many works that are in the area of temporal attack in RNNs, but Papernot et al. (2016b) seems to be relevant, which focuses on the AAs in discrete input space using RNNs. Instead of FGSM, it uses forward gradient attack introduced in Papernot et al. (2016a) to construct the input perturbation. It aims to change the classification for a given sequence by modifying specific words in the input sequence. To do this, one will have to resolve the issue of processing discrete-input space because AAs typically require taking the gradient from the output w.r.t. the input. We will also use this technique in our attack setup. The key idea is instead of directly taking the gradient w.r.t. the input, what we can do is to take the gradient w.r.t. the embedding layer. When we are looking for the word to perturb, we basically find a new word whose difference to the current word will be the closest to that of the gradient. Let us say the net work approximate a function f, we are provided with input sequence $\mathbf{x}$ of length n, trying to obtain the adversarial sequence $\bar{\mathbf{x}}$. The embedding dictionary is D. The true class label is y. Following (Papernot et al., 2016b), the perturbation search algorithm can be formulated as the following:

```
Data: $\mathbf{x}, D, f$
Result: $\bar{\mathbf{x}}$
1 $y = f(\mathbf{x})$;
2 $\bar{\mathbf{x}} = \mathbf{x}$;
3 while $f(\bar{\mathbf{x}}) == y$ do
4     $i, \bar{w} = argmin_{i \in [1,n], z \in D} || \text{ sign}(\bar{x}[i] - z) - \text{ sign}(\nabla_{\bar{x}[i]} J(f, \bar{x}, y))||$ ;
5     $\bar{\mathbf{x}}[\mathbf{i}] = \bar{w}$ ;
6 end
7 return $\bar{\mathbf{x}}$;
```

Algorithm 1: Perturbation search on word embedding.

In line 4, the algorithm is simply looking for in which direction to change such that the probability for the given class will be the smallest. Papernot et al. (2016b) also conducted a sequence-to-sequence experiment that is more similar to what we are doing. The experiment had a multi-channel input and output model for 10 time steps. The output at one time step had a strong correlation with the input of the previous time step. What they managed to accomplished was that the output at a later time step could be changed by only a few perturbations in the earlier input. This experiment differs from our eventual goal in two major aspects:

- We have stronger input-output coupling. Not only does the output depend on the previous input but also the input of the current time step only depends on previous output.
- The tasks we are investigating last much longer than 10 time steps, so the difficulty of attack success is also greater.

Methodology

In this section, we will formalize DA and entail two different applications on which we are going to evaluate the effectiveness of DA.

3.1 Delayed attack formulation

Intuitively, DA encapsulates the kind of the attacks whose effects do not appear at the time of the attack but some time later, which means that DA can be rather broad, and therefore, one would need to tailor make the exact optimization objective to achieve DA for a specific application. Despite DA's broad coverage, DA problems should generally have the following setup. Provided some input signal time series $x_1, x_2, x_3, \cdots, x_T$ and the respectively output y_t at each time step. We want to have three sets of time periods, $\mathcal{A}, \mathcal{P}, \mathcal{E}$: the attack period, peace period, and error period, which are defined by:

$$\begin{aligned}
\mathcal{A} &= \{t | t_0 \leq t \leq t_0 + \Delta_A\} \\
\mathcal{P} &= \{t | t_0 + \Delta_A < t < t_0 + \Delta_E\} \\
\mathcal{E} &= \{t | t_0 + \Delta_E \leq t\}
\end{aligned}$$

In DA, we are free to perturb the input during $\mathcal{A}$. However, during $\mathcal{P}$, we will only feed the network with the original input and the network should output normally as if nothing has happened. The definition of normal behavior during $\mathcal{P}$ depends on the context. For instance, in an RL application, the network shall still direct the network to the state with high value, or in a text generation application, the model still generates some reasonable phrases. It is only when period $\mathcal{E}$ kicks in, that we allow the attack effects to appear, which could mean that for an RL agent, the agent moves to a state with low value or the text-generating model produces some pre-defined phrases. The attack effects vary for different applications, nonetheless, the input perturbation can be described by $\bar{x}_t = x_t + \mu_t$ and $\mu_t = 0, \forall t \in \{\mathcal{E}, \mathcal{P}\}$.

We now use a one-to-one series prediction task to illustrate our problem setup. As shown in figure 3.1, we let the input signal x be a sinusoidal wave and the output y also a sinusoidal wave. The solid lines plot the input-output pair before the attack and the dotted line plot the input-output pair after the attack. When it comes to input, we only see $\bar{x}_{\mathcal{A}}$ deviates from $x_{\mathcal{A}}$, whereas for the output, only in $\bar{y}_{\mathcal{P}}$ remains close to $y_{\mathcal{P}}$, which reflects our delay design that the effect of the adversarial input influences not only the attack period but also another period that is in the future but not the one that immediately follows. The author recognizes that it is entirely possible to add an additional constraint on the network output during the attack to make $\bar{y}_{\mathcal{A}}$ close to $y_{\mathcal{A}}$. This constraint further limits the search space of the possible attack vectors, so it will make it more challenging for the attack to succeed, and thus we leave this constraint for future work.

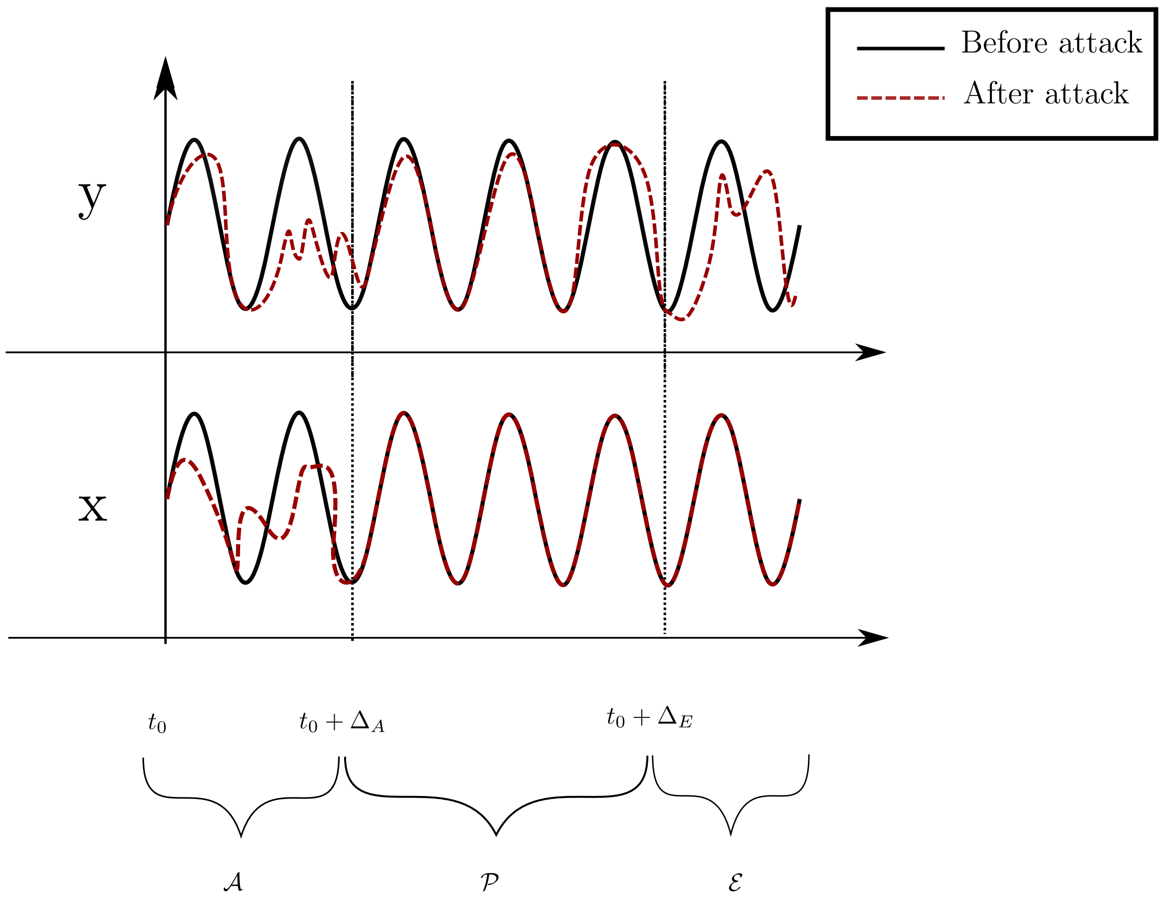


Figure 3.1: Visualization of when to attack and when not to. $\mathcal{A}$ is when the attack is allowed and thus, you can see a large deviation of $\bar{x}_{\mathcal{A}}$ from $\bar{x}_{t_{\mathcal{A}}}$. $\mathcal{P}$ is the peace domain when the attack has stopped and we are trying to make sure that $\bar{y}_{\mathcal{P}}$ does not change much from $y_{\mathcal{P}}$. $\mathcal{E}$ is when the delayed attack's effect kicks in.

Having elucidated DA setup, we can now formulate the attack objective:

$$\arg\max_{\bar{x}_{\mathcal{A}}}(\min_{\bar{x}_{\mathcal{P}},\bar{x}_{\mathcal{E}}} \|f(\bar{x}_{\mathcal{E}}|\bar{x}_{\mathcal{A}}, \bar{x}_{\mathcal{P}}) - f(x_{\mathcal{E}}|x_{\mathcal{A}}, x_{\mathcal{P}})\|) \tag{3.1}$$

$$\text{s. t. } \max_{x_{\mathcal{P}}} \left\|f(x_{\mathcal{P}}|x_{\mathcal{A}}) - f(x_{\mathcal{P}}|\bar{x}_{\mathcal{A})}\right\| \leq \epsilon \tag{3.2}$$

$f(x_{\mathcal{B}}|x_{\mathcal{A}})$ denotes the output of function f with $x_{\mathcal{B}}$ as input after seeing $x_{\mathcal{A}}$. In equation 3.1, the min operator tries to find the new $\bar{x}_{\mathcal{E}}$ that makes the network stay with its the original network output, whilst the max operator wants to search for the attack vector that can make network generate different things. Intuitively, the max-min operation aims to find the attack vector $\bar{x}_{\mathcal{A}}$ such that even under the worst situation, the difference between $\bar{y}_{\mathcal{E}}$ and its original value can be large. The constraint in equation 3.2 is trying to make the peaceful time perturbation small for the delay effect.

To simplify notations, we let

$$\mathcal{L}_{\mathcal{E}} = \|f(\bar{x}_{\mathcal{E}}|\bar{x}_{\mathcal{A}}, \bar{x}_{\mathcal{P}}) - f(x_{\mathcal{E}}|x_{\mathcal{A}}, x_{\mathcal{P}})\|$$
$$\mathcal{L}_{\mathcal{P}} = \left\|f(x_{\mathcal{P}}|x_{\mathcal{A}}) - f(x_{\mathcal{P}}|\bar{x}_{\mathcal{A})}\right\|$$

The original objective can be turned into:

$$\arg\max_{\bar{x}_{\mathcal{A}}}(\min_{\bar{x}_{\mathcal{P}},\bar{x}_{\mathcal{E}}} \mathcal{L}_{\mathcal{E}}) \text{ s. t. } \max_{x_{\mathcal{P}}} \mathcal{L}_{\mathcal{P}} \leq \epsilon$$

Hence, our objective now becomes:

$$\max_{\bar{x}_{\mathcal{A}}, x_{\mathcal{P}}} (\min_{\bar{x}_{\mathcal{P}}, \bar{x}_{\mathcal{E}}} \mathcal{L}_{\mathcal{E}}) - \lambda \mathcal{L}_{\mathcal{P}} \tag{3.3}$$

3.2 Reinforcement learning attack

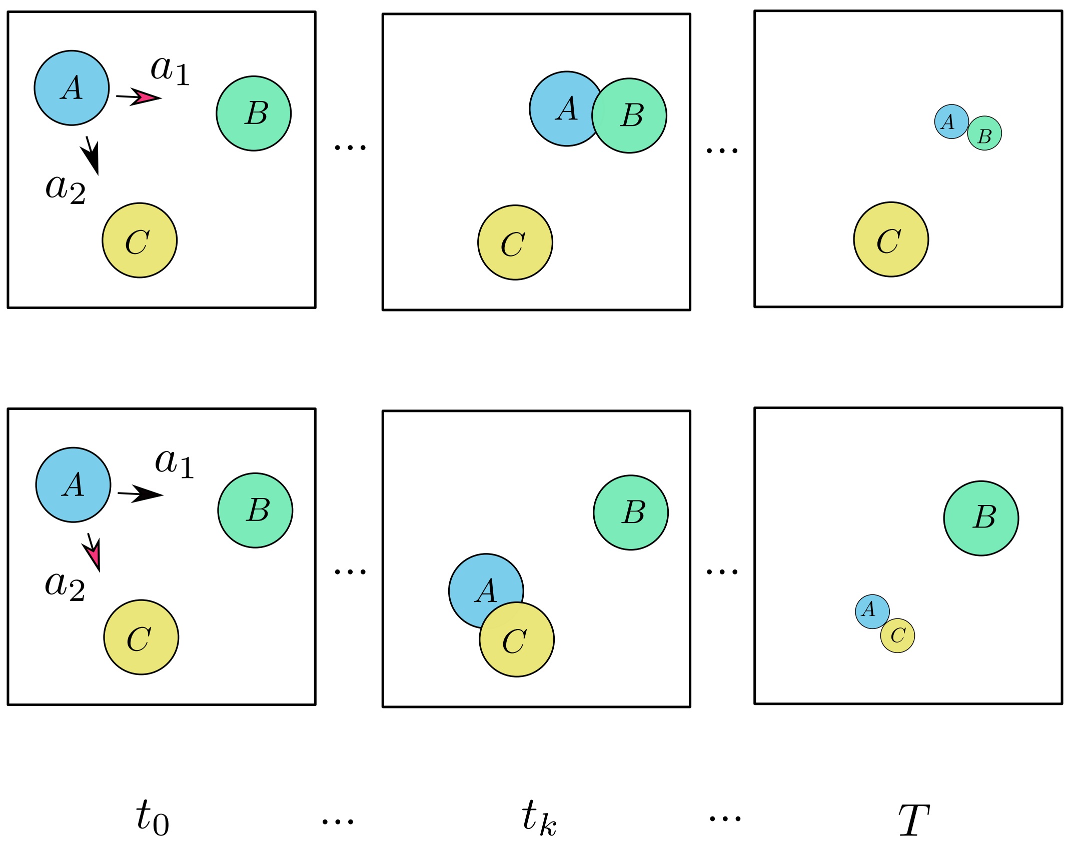


Figure 3.2: A simulation of DA in RL setting. A, B, C are three different agents in the environment. The top row simulates the scenario before the attack, when agent A executes action a_1 at t_0 and C wins the game in the end. The bottom row simulates the scenario after the attack that agent A chooses to execute a_2 instead at t_0 which makes agent B win in the end.

Having explained the general DA formulation on a simple one-to-one series prediction task. Let us dive into its meaning in RL setting. The usage of DA in RL is best illustrated using an example shown in figure 3.2. This is a sample multi-agent game where we have three agents $A, B,$ and C. The goal of the game is to be the largest at the end game. If two agents get attached to each other, both of their sizes will decrease at the same rate until one disappears. Assume agent A is a naive player who will choose to march at a_1 or a_2 direction at time t_0. Prior to the adversarial attack, agent A will choose a_1 such that A and B will meet each other at t_k. Both A and B sizes will decrease and when the game ends at time T, agent C wins. This is shown on the top row of figure 3.2. The bottom row showcases the scenario where DA takes place. If we are trying to attack agent C, at time t_0, all we need to do is to let agent A execute a_2 instead. At that point, the expected return of all players should not change much, but it is at time t_k when A and C encounter each other that the attack effect is obvious, which makes B win the game in the end.

In the multi-agent game that we have just shown, peace could mean that the agents do not touch each other so that no one's size is decreased. Generally, to accomplish DA in RL, we need to define what peace means in different tasks. As long as the agent still are directed to high-value states after the attack, we still consider the attack allows the agent to enter the peace period successfully, even if those states might differ from the original states that the agent might go to. One could also argue that if these alternative high-value states after the attack eventually lead to poor rewards, those states should also have low values too for a perfectly learned state value function. This is indeed true, but we do not know how well our network can foresee those rewards in the distanced future. That is exactly what we are trying to figure out.

Now, we will explain what mechanics we need to do DA in RL. Assume that we have a deterministic environment $Env(s_t|a_t) = s_{t+1}$, and an agent that learns the policy function $\pi(a|s)$. The optimal action to take is the maximum of the network output which also makes the agent deterministic. DA's objective can come in various forms, letting the agent reach a state with low state value at time T and afterwards or guiding the agent to a specific state at time T. If we use latter as an example. Our two loss terms will need to be:

$$\mathcal{L}_{\mathcal{E}} = \|s_T - \bar{s}_T\| \tag{3.4}$$

$$\mathcal{L}_{\mathcal{P}} = \|v(s_{\mathcal{P}}) - v(\bar{s}_{\mathcal{P}})\| \tag{3.5}$$

In order to go from the network output's probability vector over actions y_t to state s_t, we need to feed the best action, $argmax(y_t)$, to the environment function. However, even in a white-box model where the environment transition function is provided, we will not be able to take the derivative for 3.5 because the transition function is not differentiable. We will need to do something similar to Lin et al. (2017) for the enchanting task to train a generative transition function for the environment. This task might be too ambitious for the scope of this project, and therefore, we would like to test out our framework on easier tasks such as the simple copy task or NLP that do not require this transition function first before coming back to the RL setting.

3.3 Simple copy task

In order for DA to work, the model that we are attacking needs to rely on its temporal dependency to generate its output. Secondly, it will be easier if our input space is continuous when we are taking the derivative. Having these two design goals in mind, here we present out one-to-one copy task which is defined by a shift function, $y_t = x_{t-b}$ where $b \in \mathbb{R}$. x_t is constructed using an alternation between Poisson arrival of a random variable $k \sim Uniform(-1, 1)$ and rest period. The pseudocode for the copy task construction is shown in algorithm 4 in the appendix.

There are a few key parameters for the data generation. In top row of figure 3.3, we see an example of input and output signal pair. For the input signal, each pattern lasts for 20 time steps and then alternates with a 20 time steps rest period. We continue this alternation until the end of the sequence. The output signal is basically the input signal shifted to the right by 20 time steps. We refer this task as the simple copy task. We also have a variant of this called accumulative copy task as shown in the bottom row of figure 3.3.

Iterative FGSM Since the simple copy task is deterministic, we can remove the *min* from equation 3.1, and obtain $\arg\max_{\bar{x}_{\mathcal{A}}} \|f(\bar{x}_{\mathcal{E}}|\bar{x}_{\mathcal{A}}, \bar{x}_{\mathcal{P}}) - f(x_{\mathcal{E}}|x_{\mathcal{A}}, x_{\mathcal{P}})\|$ whilst still subject to the constraint in equation 3.2. We will use a similar attack strategy as FGSM except that FGSM will do the perturbation in one update, we will have to do the attack in an iterative fashion because for a time series prediction, the later time steps depend on the previous time steps. In

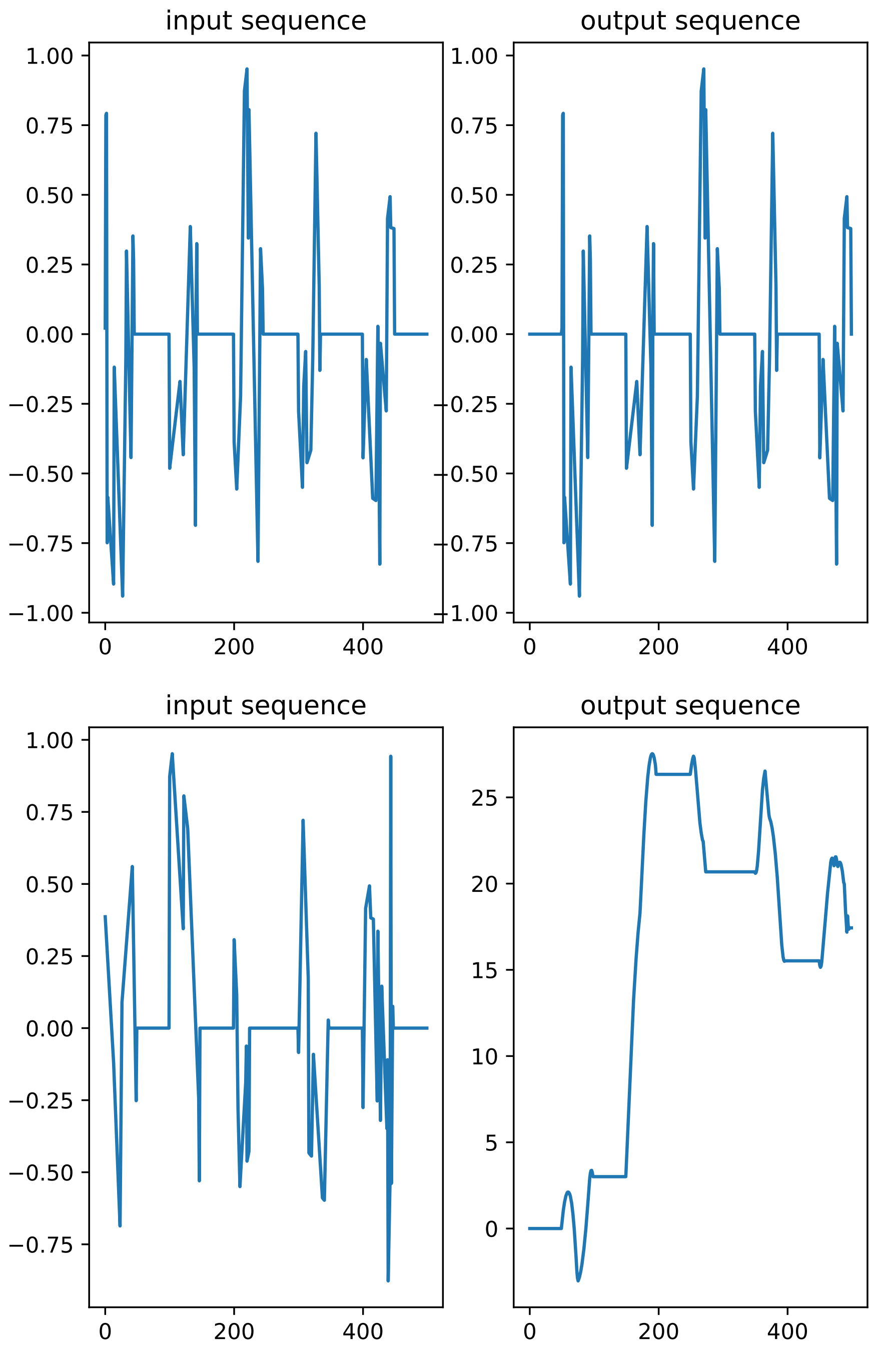


Figure 3.3: On the left hand side are sample inputs of patterns length 50, delay length 50 and rest period length 50. On the right hand side is the corresponding output signals. The top row is for simple copy task and the bottom row is for accumulate copy task. The x-axis is the time step. The y-axis stands for the input and output value.

algorithm 2, we demonstrate how we are doing the attack for the copy task. Initially, we will add some small Gaussian noise to y to make the gradient nonzero and during each update, we first compute the minimal squared error (MSE) and then find out the x_t that has the biggest gradient w.r.t. the loss. In the end, we use a FGSM to do the update.

```
Data: x, y
Result: $\bar{x}$
1  initialize a, p, e, $\alpha, \beta$ // window lengths and their importance coefficients ;
2  initialize $\epsilon, i, \eta, itr_{max}$;
3  $y = model(x)$ ;
4  $y = y + Gaussian(\mu, \sigma)$ ;
5  $\bar{x} = x$ ;
6  while $loss < \epsilon$ and $i < itr_{max}$ do
7      $\bar{y} = model(\bar{x})$ ;
8      $L_{\mathcal{A}}, L_{\mathcal{P}}, L_{\mathcal{E}} = computeMSE(\bar{y}, y, a, p, e)$ ;
9      Loss = $\alpha L_{\mathcal{P}} - \beta L_{\mathcal{E}}$ ;
10     compute $\nabla_{x_{\mathcal{A}}} Loss$;
11     $k = \arg\max_k \nabla_{x_k} Loss$;
12     $\bar{x}_k = \bar{x}_x - \eta \cdot \text{sign}(\nabla_{x_k} Loss)$ ;
13     $i = i + 1$;
14 end
15 return $\bar{x}$;
```

Algorithm 2: Attack vector generation on copy task.

3.4 NLP task

The attack method for NLP remains largely similar to that of the copy task, however, there exist two major differences between the NLP and the copy task attack strategies:

- The input space for NLP is discrete whereas, for the copy task, it is continuous.
- We slightly revise our objective in NLP to make things easier for the first attempt. Instead of trying to make the model generate wildly different results from its original output during the period $\mathcal{E}$, in our setup, we want the model to generate a specific word immediately after period $\mathcal{P}$.

Now, let us try to resolve these two issues one by one. The difficulty of having discrete input space lies in the computation of the gradient w.r.t. the input. In order to make this work, we take inspirations from Papernot et al. (2016b), where they are also the adversarial attack for an NLP classification task. The workaround is to take the gradient w.r.t. the embedding layer instead. When we are searching for the word to perturb, we try to look for the new word whose embedding difference from that of the original word is closest to the gradient. Concretely, let us say that our text sequence S is a concatenation a list of word embeddings $x_1, x_2, \cdots, x_T$. We are also provided with the embedding dictionary $D \in \mathbb{R}^{n \times m}$ where n is the number of words and m is the size of each embedding. We will show how to conduct the attack such that the model will output a designated word after period $\mathcal{P}$. The exact procedure is shown in algorithm 3.

```
Data: S: $\{x_1 x_2 \cdots x_{T-1} x_T\}$; D; k
Result: $\bar{S}$
1  initialize a, p, e, $\alpha, \beta$ // window lengths and their importance coefficients ;
2  initialize $\epsilon, i, itr_{max}, \eta$;
3  $Y = model(S)$ ;
4  $\bar{S} = S$ ;
5  while $loss < \epsilon$ and $i < itr_{max}$ do
6      $\bar{Y} = model(\bar{S})$ ;
7      $H_{\mathcal{A}}, H_{\mathcal{P}}, H_{\mathcal{E}} = computeCrossEntropy(\bar{Y}, Y, a, p, e)$ ;
8      Loss = $\alpha H_{\mathcal{P}} - \beta \bar{Y}_K^{a+p+1}$ ;
9      compute $\nabla_{S_{\mathcal{A}}} Loss$;
10     Select $k \le a$ such that
11         $\bar{W}_{new} = argmax_{z \in \mathcal{D}} < \nabla, x_k - z > -\frac{1}{\eta}|x_k - z|^2$ ;
12     $\bar{W}_i^* = \bar{W}_{new}$ ;
13     $i = i + 1$;
14 end
15 return $\bar{S}$;
```

Algorithm 3: Attack vector generation on language modeling task.

Here, η represents the weight of the Euclidean distance between the current word embedding and the new word embedding. The reason why this is important is that, perhaps during optimization even if the dot product of the gradient and the embedding difference is 1, this new word embedding might be vary far away from the current word and hence a huge update, it might be better to change this word to a closer word but has a slightly smaller dot product value.

To obtain a specific word after period $\mathcal{P}$, we would like to maximize the probability of word k at time step $a + p + 1$. Furthermore, since the network outputs probability vectors, we simply want to keep the cross-entropy similar before and after attack during period $\mathcal{P}$.

Beam search NLP attack What we observed empirically was that, if we always tried to use the maximum value as the new word to change, we tended to run into a loop that the algorithm tried to change the same words in the same fashion repeatedly. What worked nicer in practice was to add beam search in choosing the word to perturb. This implies for line 11 in algorithm 3, we will select the 3-5 best candidates and then do a random selection among them to be our actual change.

Absolute peace NLP attack with beam search In addition to algorithm 2 with beam search, we also want to introduce a variant which strictly enforces the peace time constraint. Concretely, when we are finding the best word to perturb, if this perturbation changes any word in the peace period, then we will discard this perturbation and try out the next optimal candidate.

Experiments

4.1 Copy task

4.1.1 Experiment setup

For the simple copy task, we let the pattern length be 15/20/25 time steps and the delay length also be 15/20/25 time steps respectively. We will refer them as “15-15”, “20-20” and “25-25” in the later section. Each sequence contains 30 patterns and the same number of rest periods. We have 10K sequences in in our dataset for each setup. Half of the data is used in training and the other half is used in testing.

As for the network architecture, we used a single layer LSTM with 128 hidden units. The input and output layer both only had one unit. We also used Adam as the optimizer with a learning rate of 0.0007. The minibatch size was 500. To counteract the gradient explosion, we also imposed a gradient clip of 1.0 in the training process.

h-detach We found it rather challenging to converge on delays greater than 30 time steps. That was why we tried out h-detach (Arpit et al., 2018) to hopefully make it easier to train the network with greater temporal dependency, albeit, the improvement was not obvious, so we discarded h-detach in our training procedure and stayed with the synthetic data that has less temporal dependency.

Hidden-state initial value There exist two types of initialization methods. One is to initialize the hidden state values from zero for every single new sequence. The other one is to keep the last hidden state value from the previous sequence as the initial value for the new sequence. In our experiment, we did not find any major difference in performance, so we just opted for the zero initialization approach as it is easier to implement.

Attack strategy We implemented algorithm 2, and set the iteration cap to be 100. We also have three different attack modes with the lengths of all three periods of the same. We tried all three different different windows lengths 15, 50 and 100. Each of these setup up is being run five times. To differentiate the attack modes from the copy task type, we will refer the attack strategies as “a15”, “a50” and “a100” in the following discussion.

Accumulative copy task For this variant of the simple copy task, we keep all the network parameters the same except that we only train the model for 15-15 version and do the attack only with window length of 15.

Setup	Training loss	Testing Loss
15-15	0.00206	0.00207
20-20	0.00297	0.00281
25-25	0.00549	0.00514

Table 4.1: Training and testing loss for each task.

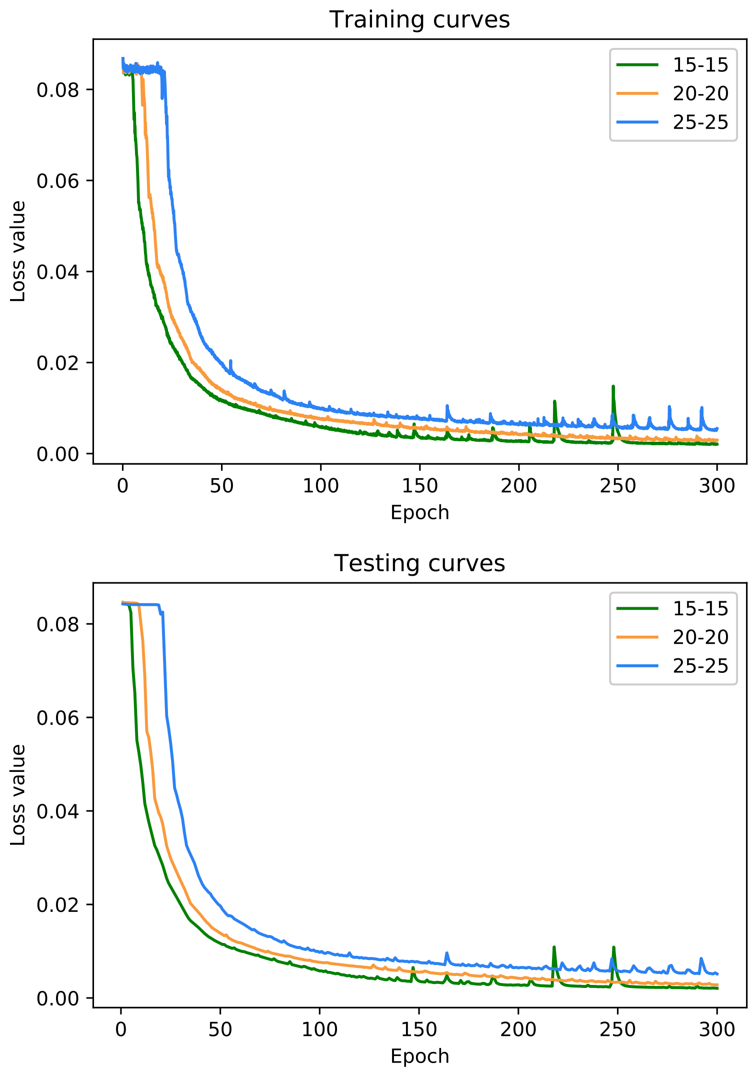


Figure 4.1: Training and testing curves for the simple copy task model.

4.1.2 Experiment results

Having a small learning rate and enforcing gradient clip, we could achieve reasonable good testing and training errors after 300 epochs as shown in figure 4.1. All three setups, have similar training and testing loss with $15-15$ having the overall smallest loss, $20-20$ having the second smallest and $25-25$ having the greatest. There exist occasional fluctuations in the learning curves after 200 epochs, but those did not cause major issue for the eventual performance. Additionally, we can find out the exact training and testing loss in table 4.1 for each task. 15-15 has both the lowest training and testing loss, 0.00206 and 0.00207, whereas 25-25 has both the greatest training and testing loss, 0.00549 and 0.00514.

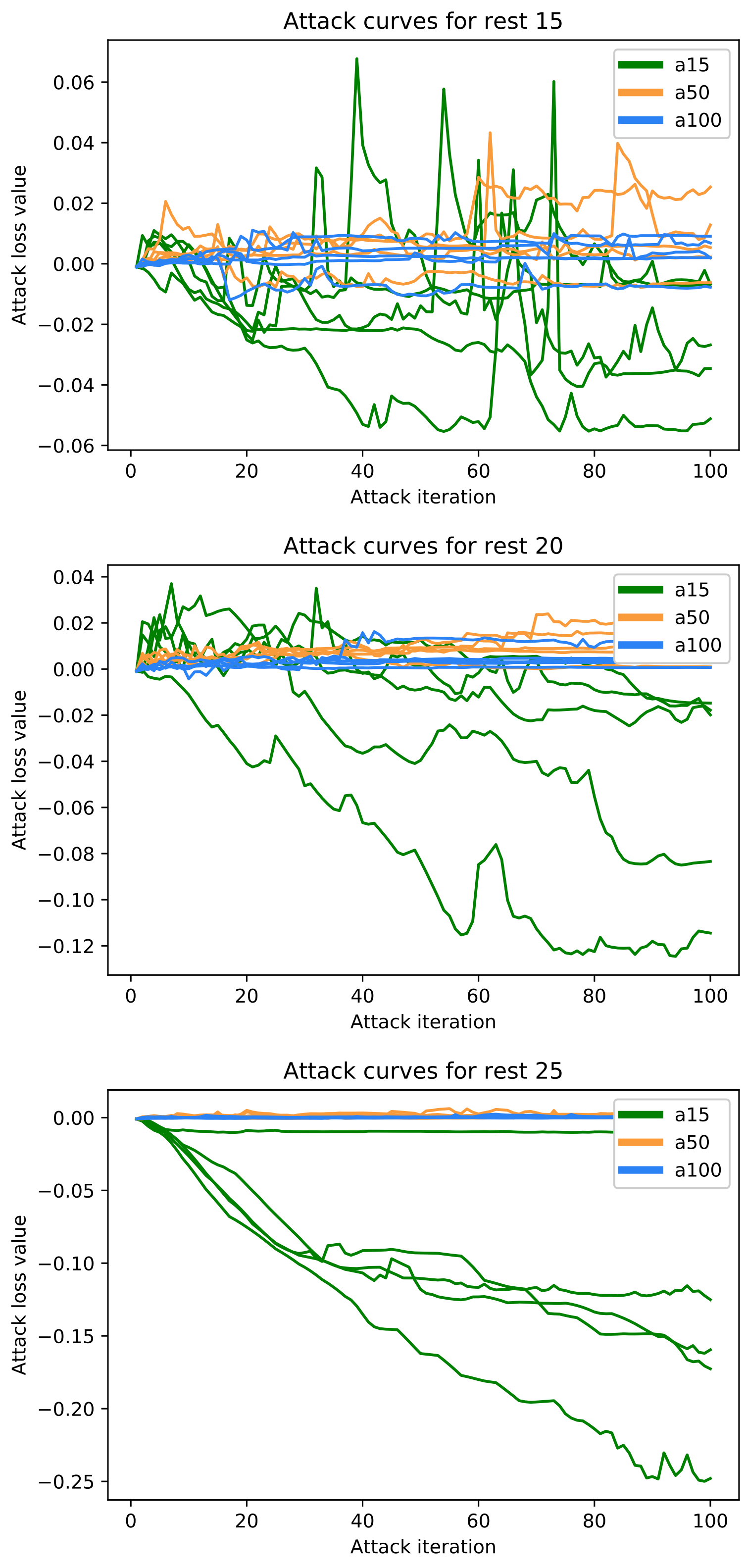


Figure 4.2: These are the attack curves for the simple tasks. From top to bottom are the attack curves for simple copy task of 15-15, 20-20 and 25-25. Each colored line in the plot represents different attack setups. For instance, 15 represents that the period lengths of different periods are all 15, $a = p = e = 15$.

In figure 4.2, we are showing the attack loss function evolutions in different attack setups for each data distribution. The attack loss value here is the weighted MSE between the original network and the new network output for the peace and error periods as described in algorithm

2, $\alpha L_{\mathcal{P}} - \beta L_{\mathcal{E}}$, where $\alpha = \beta = 1$. Each plot represents the attack outcome for one data distribution. Each line describes the attack loss evolution for a particular attack setup. Overall, a15 outperforms a50 and a100. a50 and a100's performances are very similar to each other.

In the top panel, the plot for the distribution 15-15, we see that curves of period a50 and a100 mostly fluctuate around zero. There are some ups and downs after iteration 60. However, for a15, it can go as low as -0.06 and most of a15 curves have an average below the zero mark. In the middle panel, the plot for the distribution 20-20, similar patterns are observed except that the curves of a15 have a stronger downward trend w.r.t. the iteration count. The a50 and a100 curves stay close zero, some of which have positive loss value around 0.02 in the end. Lastly, for the bottom panel, the plot for the distribution 25-25, again the general pattern has not changed but that the a15 curves can reach to -0.25, the smallest loss value observed across all data distributions and attack modes.

Now, let us examine some of the attack examples in detail. In figure 4.3, we are plotting the sample effect of our proposed attack on the simple copy task. The green dotted line is the network output prior to the attack. The colored lines are what the network output after the attack is in different periods: red for $\mathcal{A}$, blue for $\mathcal{P}$, purple for $\mathcal{E}$ and yellow for post-$\mathcal{E}$. Each plot is showing the attack effect of a different sequence of type 15-15. Generally speaking, the model output only deviates from its original value in the second half of $\mathcal{A}$, and the beginning of $\mathcal{P}$. The deviation from the original output does not seem to propagate into the later periods. Across all three examples, the modified output in $\mathcal{A}$ tend to have lots of conciliations and the distinction between rests and patterns also cease to exist. In most of the other periods, we can still see a clear separation between the patterns and the pauses in between. The modified output also generally follows the original trajectory.

Here, we want to point out the phenomenon that the adversarial effects will only propagate to the immediate future. It is patent across all the sample attack trials and the ones that have not been shown. Even though the early steps of period $\mathcal{P}$ also seem to be influenced by the attack vectors, period $\mathcal{E}$ almost seems to be completely undisturbed.

Lastly, let us conclude this section with the attack performance for the accumulative copy task. We were able to obtain a loss value of 1.324 on the training set and 1.724 on the test set after 300 epochs. We did not conduct a systematic evaluation of the accumulative copy task, however, in figure 4.4, we can see several attack curves for DA. These experiments have the same setup as the simple copy task having the 15-15 data distribution and iterative DA for 100 iterations using the a100 mode. The adversarial effects of DA on the accumulative copy task, however, differs greatly from that of the simple copy task. In period $\mathcal{A}$, the modified output does not have as much oscillations. The changes occur more gradually. It is worth noting that the deviation almost always starts out small and starts to grow to a certain point at the end of the attack period and then stabilizes after the attack period. The dynamics of the original network is mostly persevered but not the offset. After period $\mathcal{A}$, the attacked network maintains its output deviation from the its original value into the other periods.

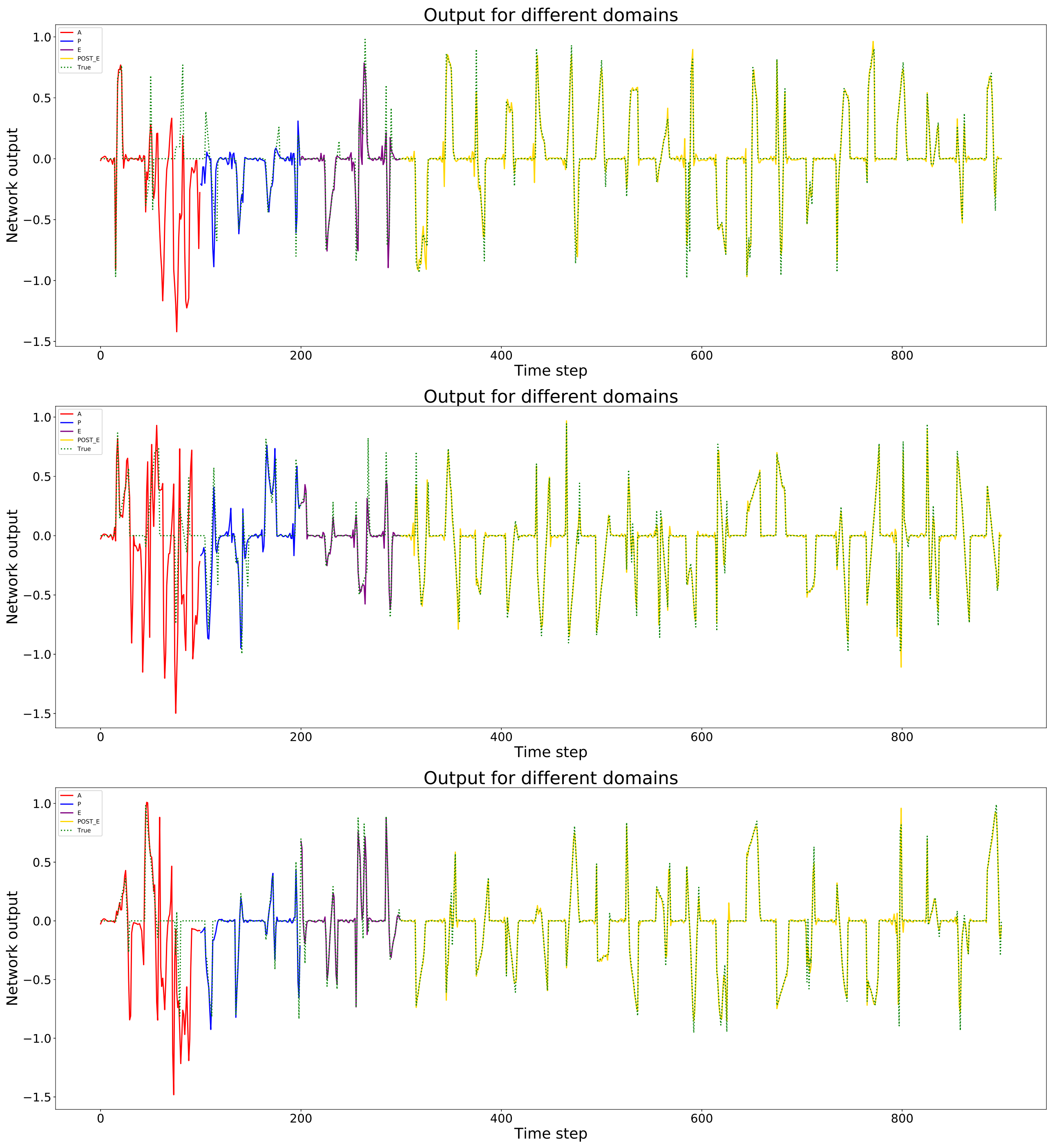


Figure 4.3: This plot shows the network output pre- and post-attack for simple copy task model $15 - 15$ after 100 iterations using a100.

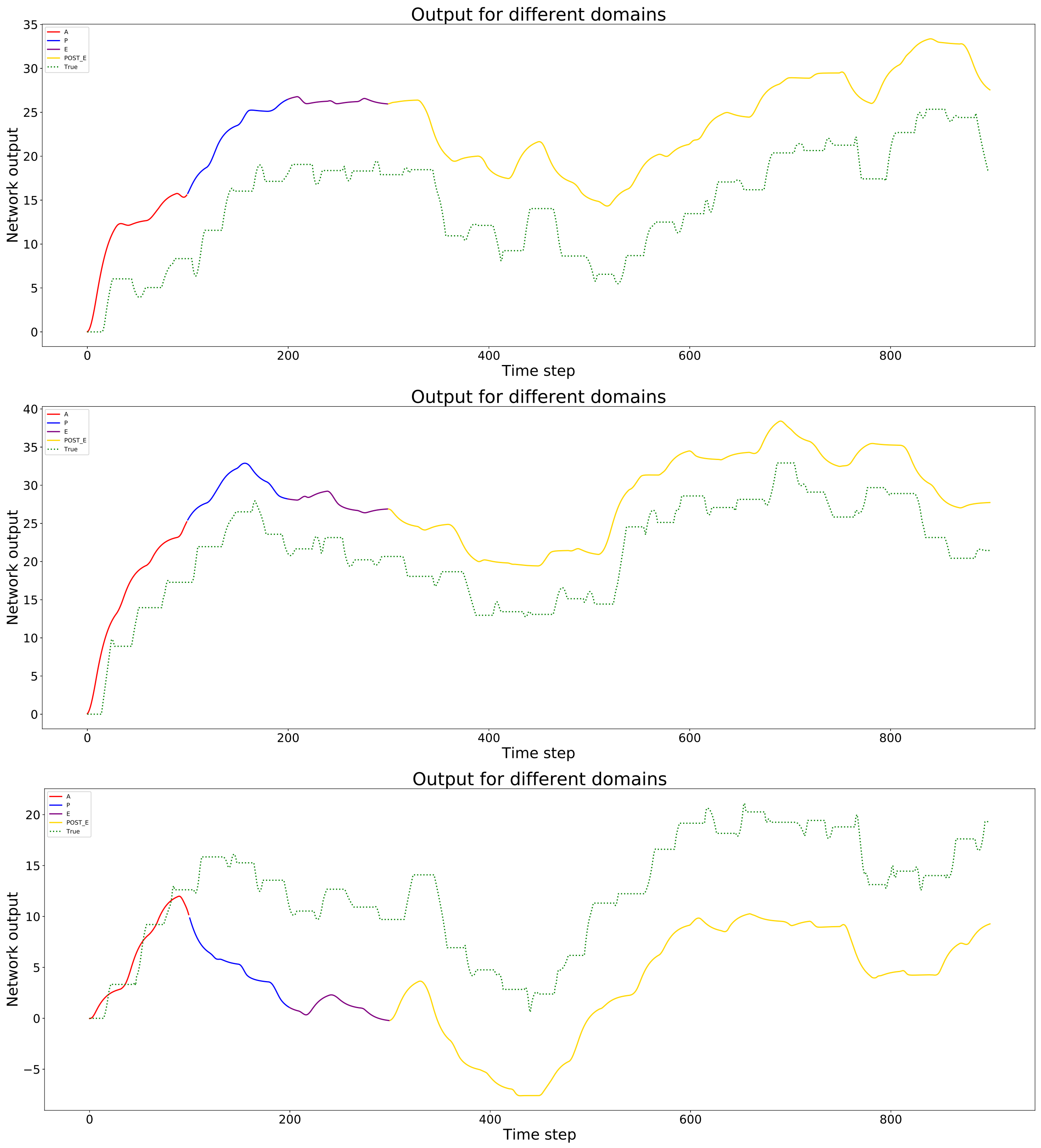


Figure 4.4: These are the attack curves for the accumulative copy tasks at different domains after 100 iterations.

4.2 Text generation task

4.2.1 Experiment setup

We mainly used the dataset, Wikitext-2 (Merity et al., 2016) which has 33279 vocabularies and about 2M tokens. Each data entry is an article on Wikipedia that has gone through human review and is considered well-written and accurate. Punctuation and special characters are also included in the corpus. You can view some sample entries in the appendix listing 8.4.

We train an LSTM with two layers coupled with an embedding layer of size 200 and a linear readout layer. Each LSTM layer also has 200 units. We use 0.2 dropout during the training. The batch size is 20 and the gradient clipping is 0.25. The initial learning rate is 4 and we will do annealing by a factor of 4 if the performance on the validation set has not improved after one epoch.

For the purpose of comparison, we also trained another model using Wikitext-103 (Merity et al., 2016) whose vocabulary size is almost 10 times as many as that of Wikitext-2, around 267735. It shares the same test and validation sets with Wikitext-2 but its train test is about 50 times as large having around 103M tokens in total. The LSTM also has two layers but the embedding layer and the recurrent layers have 650 units instead. The primary of this model to see how DA performs on a better-performing model.

Forward and backward pass of BPTT In training and testing, we split the original stream of text into mini-batches and set the BPTT unfolding to be 35. What this means is that for the forward pass, we might be able to remember the information from 35 time steps ago, however, for the backwards pass, we will not. Note that we only initialize our hidden state once such that the information gets carried through different batches. In inference time this does not make any difference.

Text generation After the network has been trained, we shall use it to generate a text sequence. The common practice is to initialize the network with a random input, and then the network will output a probabilistic vector for the multinomial distribution. In order to find out what the next word is, we can either use $argmax$ for selection or do a beam search which is to choose a few top words and then do a random select among these top words. $argmax$ operation will be likely to induce cycle in the network output, whereas the beam search strategy can help us create much better text. In our experiment, however, we are sticking with $argmax$ because we will need to have a deterministic output sequence to do the attack. The point of this project is to exploit the vulnerabilities of RNNs, whether or not the RNNs, in the end, produce some reasonable good-quality text does not matter as long as it can accomplish what its objective is.

Attack setup To evaluate DA, we aim to change the word that comes immediately after period $\mathcal{P}$ to a word that we predefine. Here, we will try to make the target word the top 50 most common words in our training set. For each target word, we will try out 5 different random seeds. You can check out the top 50 words are in the appendix listing 8.1 . For each target word, we will also have four attack modes a5-5, a15-5, a15-10, a15-20. The first digit stands for the attack period length and the second digit stands for the rest period length. Lastly, the maximums number of iteration is capped at 70. Note that due to computational constraints, for a15-20, we only tested it for the top 25 words but increased the max iteration cap to 100.

We are primarily interested in tracking the following metrics:

- The similarity between the new and original peace periods.
- The distribution of the perturbation position in period $\mathcal{A}$.
- The percentage of attack success.
- The number of iterations that we have to do to make an attack succeed.

4.2.2 Experiment results

The baseline model that we used to test out DA was trained for 40 epochs using n Wikitext-2. It could achieve a validation perplexity of 115.7. To show the capacity of this model, you can refer to the appendix listing 8.2 to see the text that it can generate using both *argmax* and beam search. The other model was trained on Wikitext-103 for only two epochs but could already achieve a validation perplexity of 79.91. This model can obviously produce more sensible sequences as you can see in the appendix listing 8.3. The main reason why we only trained this model for just two epochs was that it took on average one day to train one epoch for given our current setup. Our experiment was conducted on a single GPU, TITAN X Pascal, with 12 GB of memory. Most of our attack experiments were conducted on the baseline model due to computational constraints.

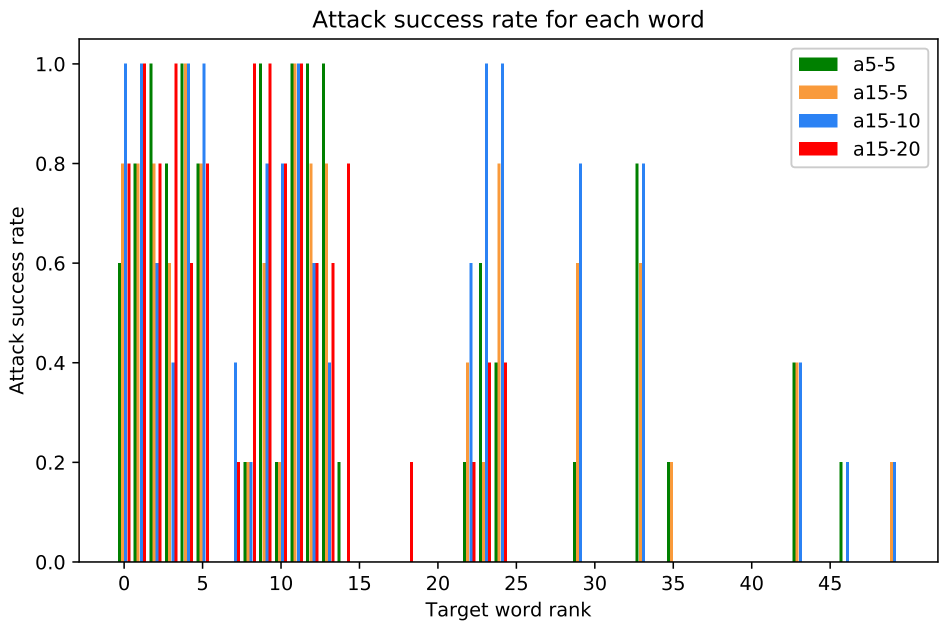


Figure 4.5: The success rate distributions for each word. The x-axis is the rank of the word probability in text corpus. The smaller the value, the more likely a word is going to appear.

For the baseline model, the overall success rates of the four different attack setups are 25.2%, 23.6%, 28.4% and 48.8%. If during a trial, an attack vector is able to make the first word after the peace period the target word, we will consider this attack trial to be successful. Let us take a closer look at the success rate for each word in figure 4.5. We see that the attack outcomes for different attack modes very similar. The words that are higher ranked tend to have better success rates. The lower ranked words have lower success rates. It is also common to see that if a word can be attacked successfully using one attack mode, this word is also susceptible to the other attack modes. Only a few words can be successfully attacked by just one mode such as word 19.

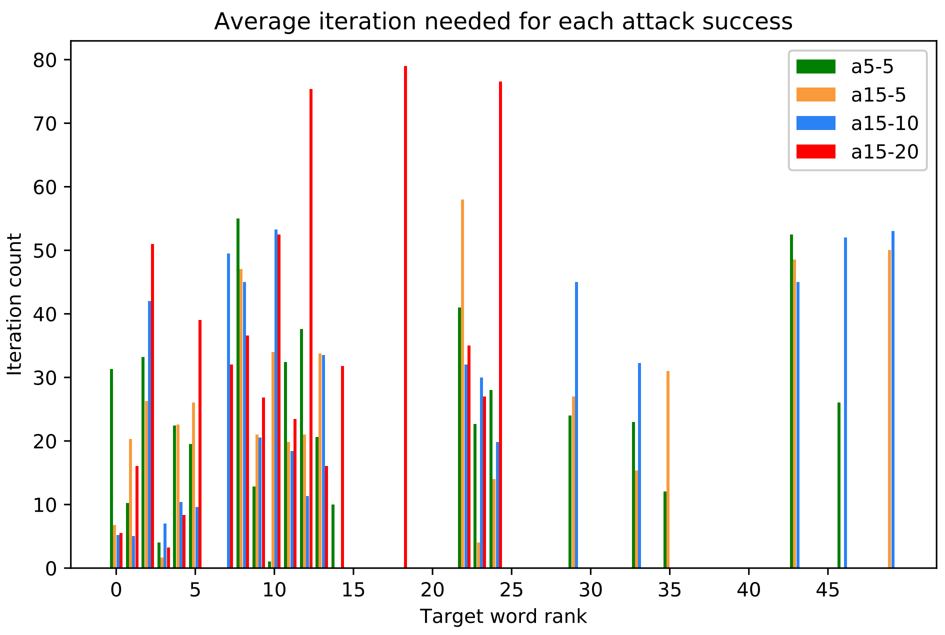


Figure 4.6: The average iterations needed across the successful trials.

In figure 4.6, we can see the average number of iterations needed for an attack success. The iteration count will be zero if we are not able to have a successful attack within 70 iterations for a5-5, a15-5 and a15-10 and 100 iterations for a15-20. There is a week positive correlation between the iteration count and the target word rank. The lower ranked a word is, the more iterations we will need for an attack success. Meanwhile, we cannot find obvious similarities between different attack mode. For the target words that might require one mode many iterations to succeed, it might take another attack mode far fewer iterations to succeed. This means that the iteration count of one mode might not be indicative of the iteration count needed for another mode.

To fully understand the correlation between the attack success and the rank of a word, we also conducted another experiment that used 200 as the max iteration count in the attack mode 15-20 for 5 much lower ranked words, [England, career, position, children, construction]. It was not possible to have even one attack success with the 200 iteration cap.

Now, let us turn to the peace period changes in figure 4.7. We are plotting the average number of words changed in the peace period across the attack successes. The mean number of words being changed in peace period are $4.20, 4.56$, 8.92 and 19.04 for modes a5-5, a15-5, a15-10 and a15-20, which means that for DA to accomplish its objectives, it will need to change almost every single word in the peace period.

It is also worthwhile looking at how exactly the attacks are being done, we will illustrate how the attack vectors are distributed in the attack period. In figure 4.8, we are showing the exact locations of words being perturbed for the attack successes. In the left panel, we see that words on all positions in the attack period for the a5-5 mode, have a probability of approximately 0.8 being changed. In the right panel, we see a more disparate distribution of the attack vectors. Overall, a15-5, a15-10 and a15-20 follow a similar general distribution. The first word for these modes have on average have a greater probability (0.8) being changed as compared to an average probability around 0.6. a15-20 also appears to change most of the positions more often than the other two modes.

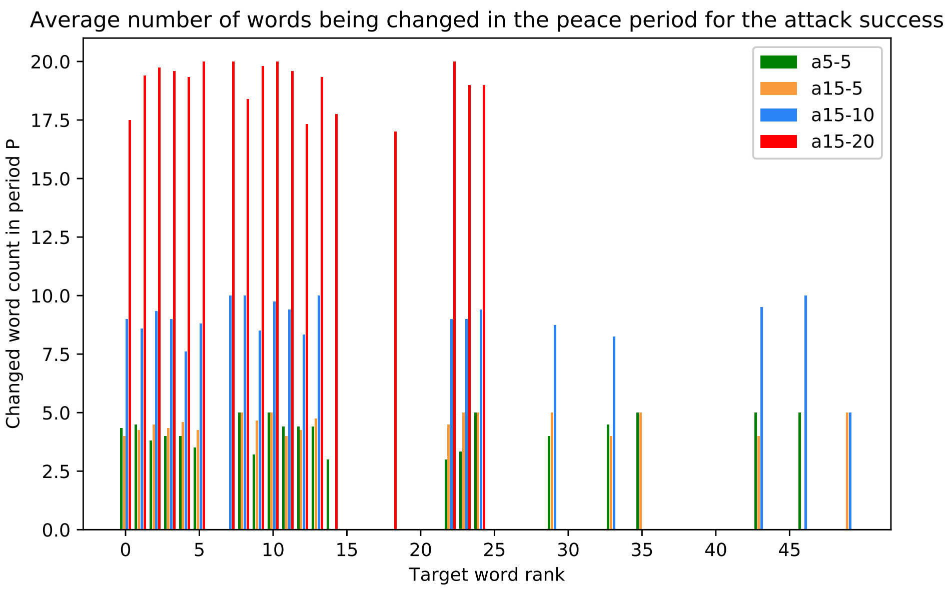


Figure 4.7: The average number of words being changed in the peace period after the attack. The x-axis is the rank of the word probability in text corpus. The smaller the value, the more likely a word is going to appear. The y-axis is the average word changes in the peace period.

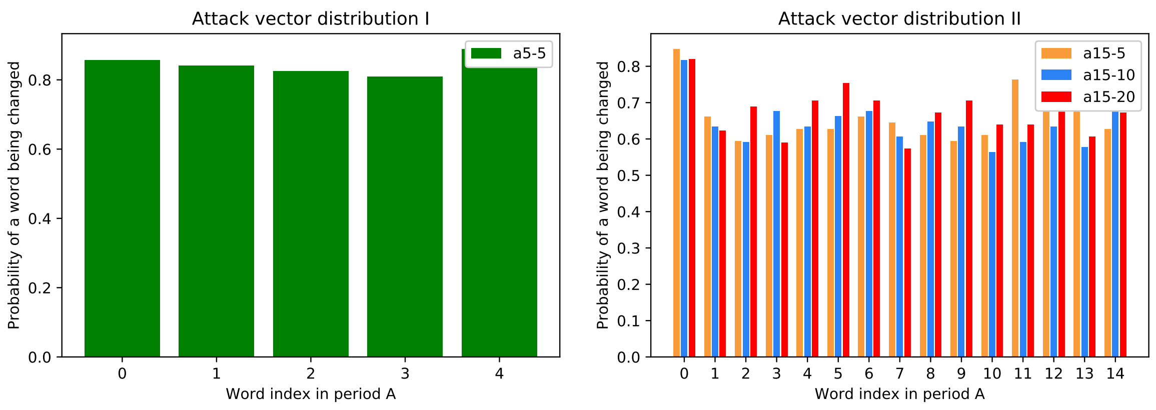
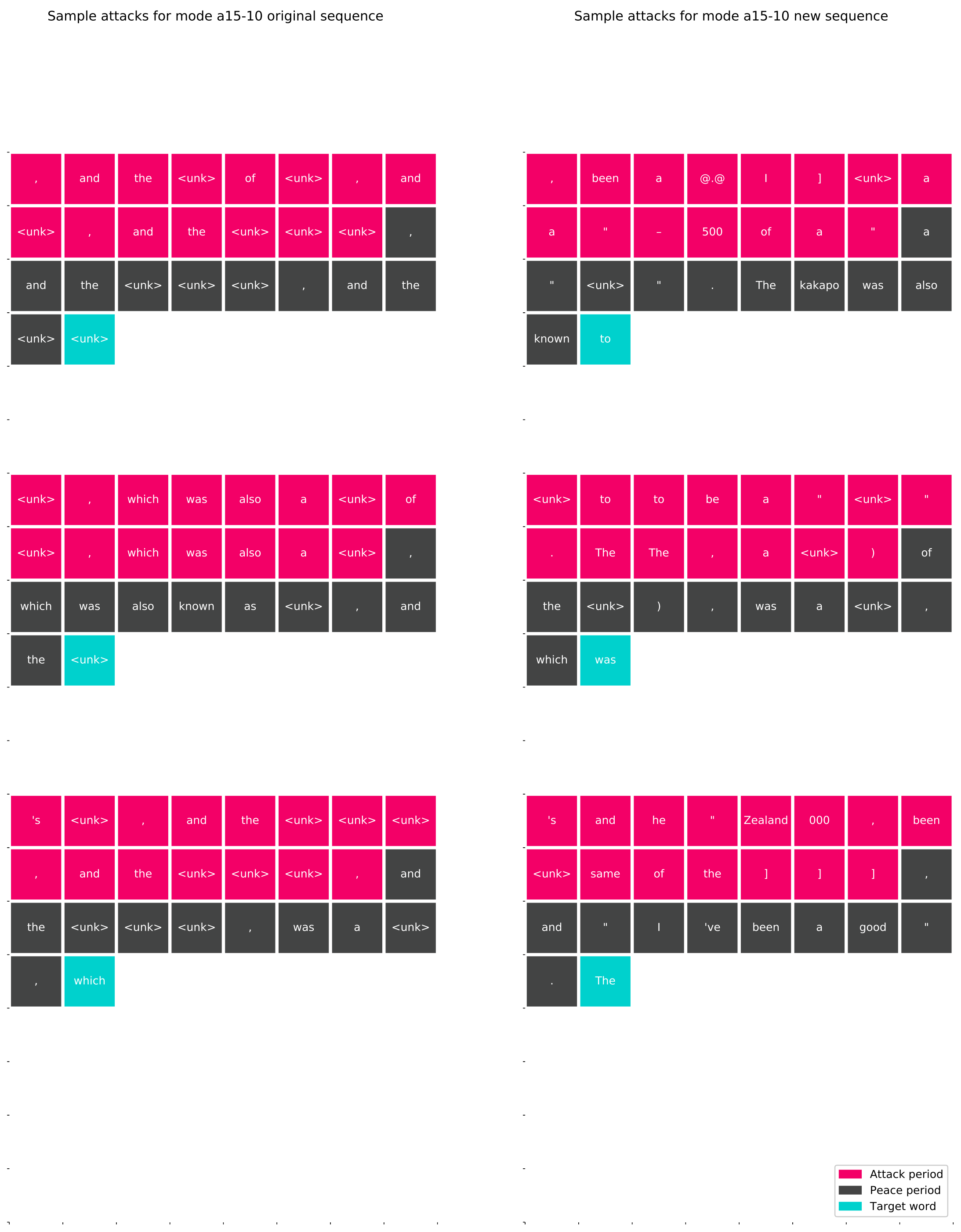


Figure 4.8: The attack word distributions. The x-axis is the position of the word being changed in the original sequence. On the y axis is the probability of the word being changed in an attack success.

Finally, we will dive into what the network outputs are before and after the attack. In figure 4.9, we see three sample success trials for a15-10. The three different periods are marked using red, black and cyan respectively. We see that for the original text sequences, the sentences are really not very coherent and have some repeated sentences. The variety of words being used is also not great. The attack positions are occupied by words "<unk>", "<unk>" and "which" prior to the attack. After the attack, the attack periods have been changed to phrases that do not make much sense. The peace periods are also changed something very different. The interesting parts lie in the few words right before the attack target, "The kakpo was also known to", "was a <unk>, which was", and "been a good″. The". The phrases immediately before the target words all have become something much more reasonable and have semantics in them.

a15-20 also demonstrates similar characteristics. In figure 4.10, we can see that the longer examples allow us to understanding the sentences better. When the peace period is too should,



Sample attacks for mode a15-10 original sequence
Sample attacks for mode a15-10 new sequence
, and the <unk> of <unk> , and
<unk> , and the <unk> <unk> <unk> ,
and the <unk> <unk> <unk> , and the
<unk> <unk>
, been a @.@ I] <unk> a
a " – 500 of a " a
" <unk> " . The kakapo was also
known to
<unk> , which was also a <unk> of
<unk> , which was also a <unk> ,
which was also known as <unk> , and
the <unk>
<unk> to to be a " <unk> "
. The The , a <unk>) of
the <unk>) , was a <unk> ,
which was
's <unk> , and the <unk> <unk> <unk>
, and the <unk> <unk> <unk> , and
the <unk> <unk> <unk> , was a <unk>
, which
's and he " Zealand 000 , been
<unk> same of the]]] ,
and " I 've been a good "
. The
Attack period
Peace period
Target word

Figure 4.9: Attack samples for a15-10.

it might be hard to evaluate what the attack is really trying to do. In this attack mode, we also obverse that DA has changed the words in the attack period something very different from its original sequence but the peace periods seem to have kept the sensible phrases. For the words that are right in front of the target word, we again see phrases that try to keep the impose the flow the sentence that the target word will naturally appearance such as “The series was” and “old series off”. It is interesting to see that how the network also keeps the semantics of the symbol “, and”, a comma right before the word “and”.

Before we end this chapter, we still have some more results on the model trained using Wikitext-103 dataset to show. We will call the attack with 15 attack words and five peace words using this model aa-15-5. The number of experiments run on this model is significantly less because

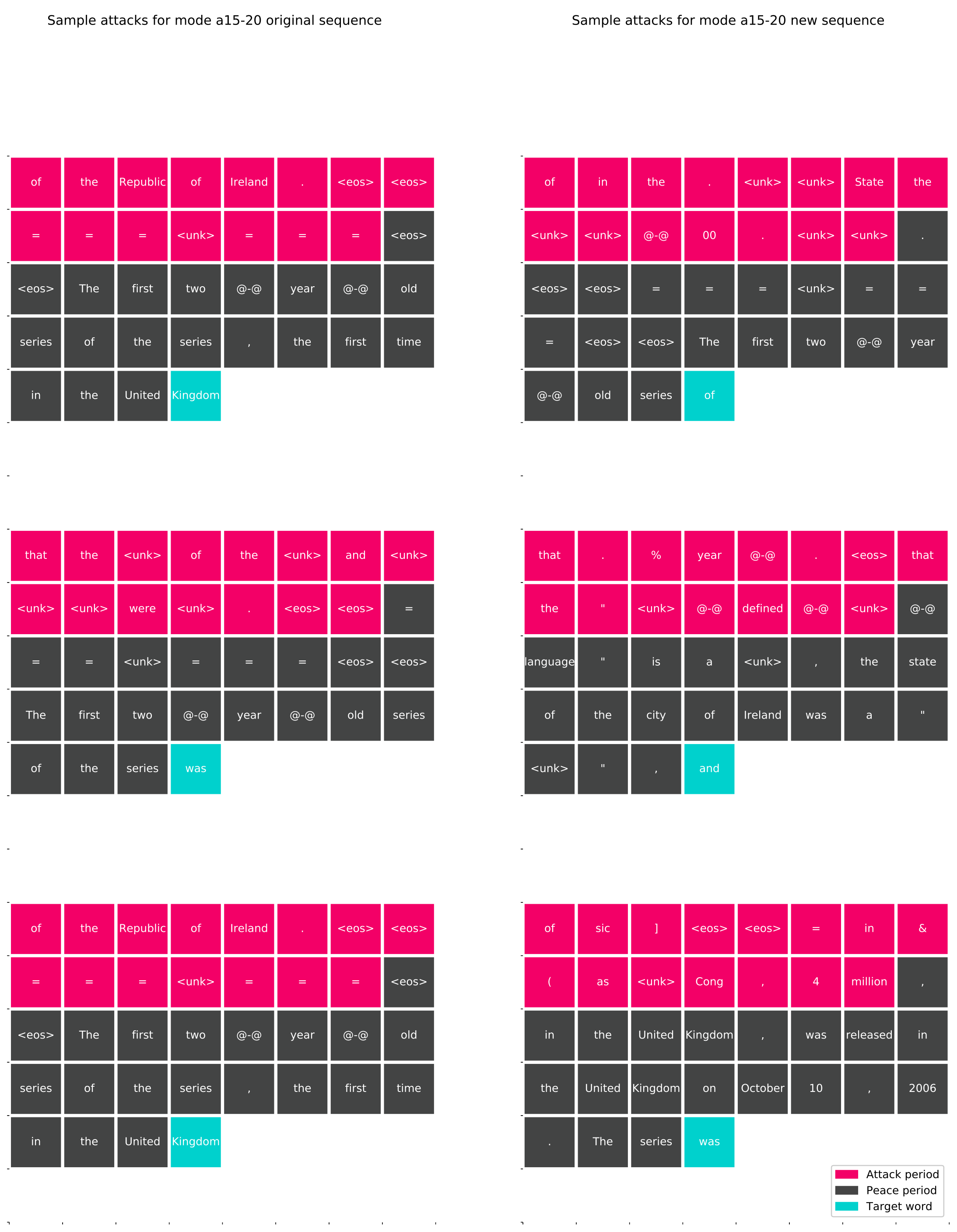

Sample attacks for mode a15-20 original sequence
Sample attacks for mode a15-20 new sequence
of the Republic of Ireland . <eos> <eos>
= = = <unk> = = = <eos>
<eos> The first two @-@ year @-@ old
series of the series , the first time
in the United Kingdom
of in the . <unk> <unk> State the
<unk> <unk> @-@ 00 . <unk> <unk> .
<eos> <eos> = = = <unk> = =
= <eos> <eos> The first two @-@ year
@-@ old series of
that the <unk> of the <unk> and <unk>
<unk> <unk> were <unk> . <eos> <eos> =
= = <unk> = = = <eos> <eos>
The first two @-@ year @-@ old series
of the series was
that . % year @-@ . <eos> that
the " <unk> @-@ defined @-@ <unk> @-@
language " is a <unk> , the state
of the city of Ireland was a "
<unk> " , and
of the Republic of Ireland . <eos> <eos>
= = = <unk> = = = <eos>
<eos> The first two @-@ year @-@ old
series of the series , the first time
in the United Kingdom
of sic] <eos> <eos> = in &
(as <unk> Cong , 4 million ,
in the United Kingdom , was released in
the United Kingdom on October 10 , 2006
. The series was
Attack period
Peace period
Target word

Figure 4.10: Attack samples for a15-20.

this model has more network weights and requires a lot more time to finish the experiments. In figure 4.11, we are plotting the attack success rate for each target word. We only managed to run the test on top 10 targets words in this instance. If we also compare the general attack success across all modes for the top 10 words, we see that their attack rates are 62%, 56%, 64%, 72% and 52% for a5-5, a15-5, a15-10, a15-20 and aa15-5. So aa15-5 is not doing any better than a15-5 but they are only 4% apart. aa15-5 could not get an attack success for word 4 and 6, but for the rest it was doable. It seems to be difficult to always have an attack success on a particular target word.

Following our discussion, we shall end this chapter with the sample attacks for aa15-10. aa15-10 seem to produce more reasonable sentences with a more capable base model. Among the

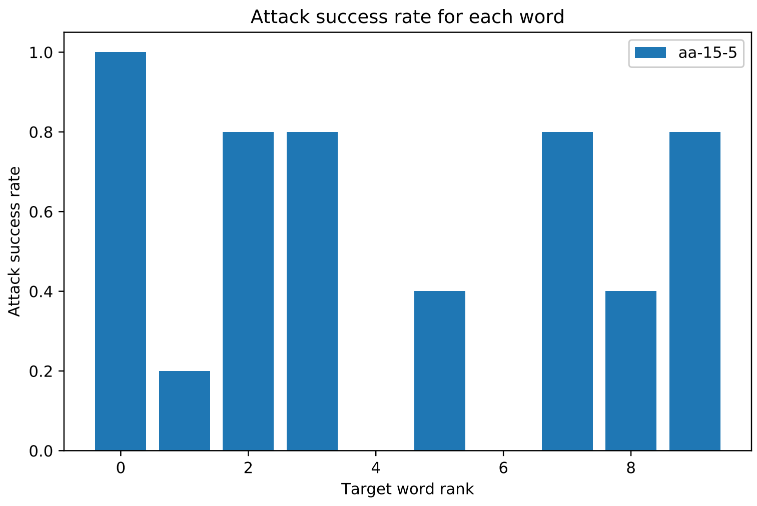


Figure 4.11: The success rate distributions for each word. The x-axis is the rank of the word probability in text corpus. The smaller the value, the more likely a word is going to appear.

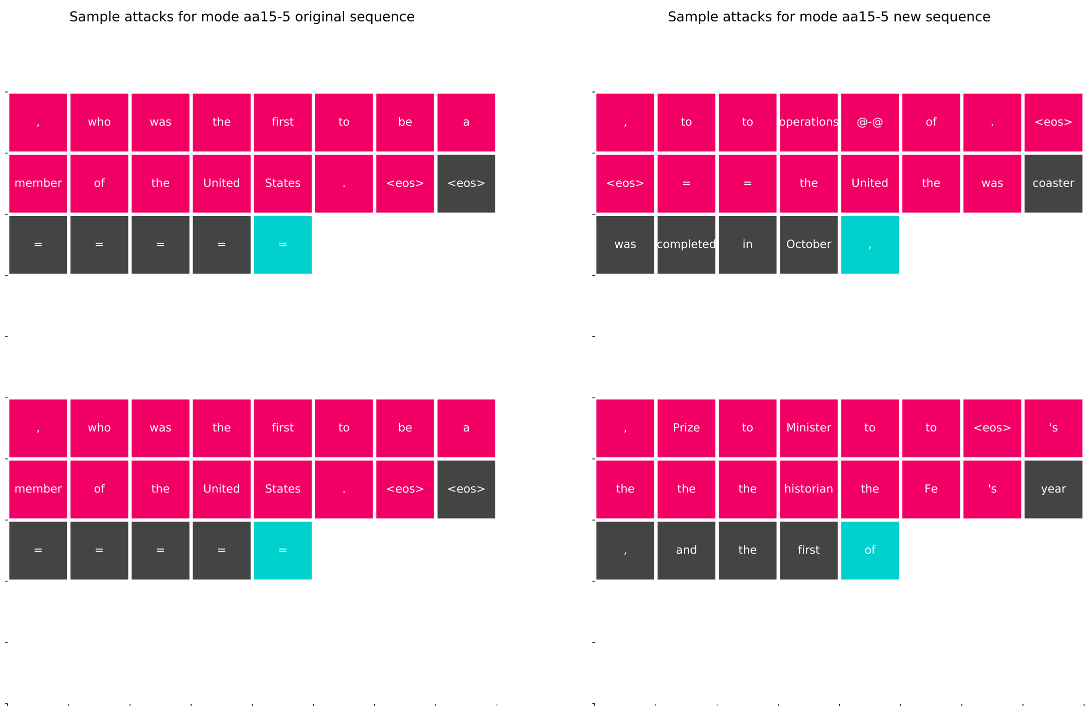

Sample attacks for mode aa15-5 original sequence
Sample attacks for mode aa15-5 new sequence
, who was the first to be a
member of the United States . <eos> <eos>
= = = = =
, to to operations @-@ of . <eos>
<eos> = = the United the was coaster
was completed in October ,
, who was the first to be a
member of the United States . <eos> <eos>
= = = = =
, Prize to Minister to to <eos> 's
the the the historian the Fe 's year
, and the first of

Figure 4.12: Attack samples for a15-20 using the model trained on Wikitext-103.

successful cases, a similar pattern still persists as compared to the attack samples using a15-5: it is difficult to see what the attacked period is trying to say and the peace period is composed of the rephrases that will naturally let the target word flow through.

Discussion

5.1 Copy task

We were able to train the models to fit with data from 15-15 to 25-25 for the simple copy task. It was difficult to increase the length of the pattern in the training phase. When it comes to the attack effect, the biggest lesson that we can learn is that the deviations induced can only influence their immediate future. Trying to propagate this error forward even without the peace period seems challenging. Nonetheless, we could replicate what other people have done, drastically change the network output by just modifying the input signal by a small and innocuous amount.

We also draw our attention to the ratio between the underlying temporal dependency and the attack setup. This should be a key parameter in determining the effectiveness of DA. Let us use a15, it is obviously much easier to do the attack when the underlying data distribution has a greater temporal dependency. This pheromone is reflected in figure 4.2 where the attack loss is getting smaller and smaller as we increase the temporal dependency of the copy task from 15 to 25. During this transition, the smallest loss value for attack mode 15 decreases from -0.06 to -0.25 whilst the smallest loss values for the other attack modes remain similar. This also indirectly indicates how the ratio between the data's own temporal dependency and the attack setup has significant importance for the attack success. Because for the other two modes of attack 50 and 100, every period of their setups exceed 25, the greatest temporal delay in our synthetic data. If the network were to have a cycle in producing the delayed output, the attack modes that have longer period lengths would have to overcome several cycles of dynamics to have a lasting effect. Whereas when we only have a window length of 15, the network dynamics is probably not completely reset yet when we want to have the attack effects to occur.

We also hypothesize that the reason why the attack effects cannot propagate into the future is that our copy task did not incorporate enough temporal dependency in its design. Concretely, it is because that no matter what the network generates at the current time step, the network does not need to know about this output to still have a perfect output at the next time step if the input is correct. This is easily seen from the definition of the simple copy task $y_t = x_{t-b}$.

Now, Let us we fall back to the RL setup, how RL differs from our copy task is that in RL, the output of the network will have an influence on the input of the next time step, however, in the copy task, input and output stay independent of each other. In the view of RNNs, the cell states carry information about its input and determine the history but it is the network weights that determine the model behavior. If after a while, the history is forgotten, without changing the network weights, DA will really not be applicable. It was exactly because of this insufficient temporal dependency that we decided to include the accumulative copy task, whose the output at a later time step also has a correlation with the output of the previous time step.

Nonetheless, even for the accumulative copy task, what we discovered was that it did not fulfill the conditions for DA to succeed. Our single layer model was able to learn the new pattern. However, during DA, the error produced in the attack period was accumulated throughout the peace and error periods. So it was impossible to have a real peace period. The real reason why the accumulative copy task differs from what we are looking for in figure 3.2 is that in RL we are looking for two different compositions that are drastically different from each other but still have similar state values, in a naive task like the accumulative copy task, it does not have the same equivalence where the state in peace period should be different but occupies a similar state value. Therefore, even though both copy tasks might seem to be great candidates to test out our DA framework in the first glance, they fail to meet the additional requirements, having strong input-output coupling and the existence of peace period where there are some other states that are highly different but have similar state values.

5.2 Text generation

To start with, we were able to train the two-layer LSTM to reasonably well model the Wikitext datasets. When it comes to the attack effect, it seems that the attack success is more dependent on the choice of the target word and less on the attack setup as most of the attack success rates are very close to each other except for mode a15-20. This is probably due to that it only needed to attack the top 25 target words but the other modes had 50 target words instead. The lower ranked words generally have lower success rates, so it makes sense that a15-20 has a greater success rate. We also observed that if a target can be produced by one attack mode, it is more susceptible to other attack modes, even though the iterations needed for different attack modes to produce this target word can be drastically different. Initially, we thought because the average iterations needed to an attack success increases for the lower ranked words, it was entirely possible that if we increased the iteration count that we could also achieve attack success for the lower ranked words, however, our experiments so far told us the contrary. Among the five lower ranked words that we selected, we could not get any attack success even when we almost tripled the iteration cap to 200 from 70. Perhaps, to really see the correlation between the target word rank and attack success, one probably needs to do more systematic experiment but at least what we do know is that for the lower ranked words, it will take more iterations to attack.

When it comes to the peace period, we did not have any satisfactory performance. For most of the attacks to work, they seem to require changing all the words in the peace period. More interestingly, they also want to change the phrases right in front of the attack word to be a reasonable sentence for the target word to follow. Even when the original sequence was not sensible prior to the attack, the attacks tend to force the network in producing a peace period where the target word can naturally follow afterwards such as putting a period at the end of the peace period to let “The” comes out, and putting a comma at the end to let symbol “,” come out. It is worth mentioning that in this example “The” has its first letter in upper case, there is another word where the letter “t” is lower case. It is also surprising to see that the attack success rates have nothing to do with the attack mode. No matter if your peace period is 5, 10 or 20, they all have achieved a similar success rate. One more note on the difference in the attack period between the original and the new sequences. The attack vectors that DA tries inject seem to be something completely random but they are able to enforce more grammatically correct phrases in the peace period. It is hard to interpret the phenomenon.

As for the attack vector distribution, we found out that at least for mode a5, the probability of a word being changed at every single position is roughly the same but the situation is different for the other two modes. Under attack schemes a15-5, a15-10 and a15-20, they all tend to change the first word much more likely than to change other words among the success cases. We do not have a very clear idea of why this is the case. It could be that since the initial hidden state values are zero, and thus the first input can have a major influence in guiding the network dynamics.

Now we want to address the influence of *argmax* in our attack outcome. As we have seen, both the baseline models trained on Wikitext-2 and the more complex model trained on Wikitext1-103 can generate sensible text using the beam search approach. *argmax* severely sabotages the text generation quality. From a theoretical perspective this makes sense because the greedy optimal of a word does not guarantee the global optimal for the whole sequence, however, it does offer practical advantages allowing us to test DA without worrying about the stochasticity that beam search has. This can also partially explain why in the peace time so many words have been changed. Prior to the attack, many words in the peace time are simply a repetition of some cycle that might not make sense, however, in order to make the target word the one we want, DA has to push phrases in the peace to a point where the target word can naturally follow.

On the positive note, the model trained on Wikitext-103 can generate more reasonable sentences using *argmax*, but it is challenging to run many tests on this model because of the run time. The same attack mode on different base models a15-5 and aa15-5 have similar attack success rates. Originally we thought it would be easier for aa15-5 to attack since in our DA algorithm, we are approximating the gradient difference using the difference between word embeddings, so with a denser embedding, aa15-5 should be able to achieve better performance, but we did not observe any obvious attack improvement on aa15-5. Nonetheless, we shall keep in mind that we only have 10 target words and one attack mode using the Wikitext-103 model, so it is difficult to say anything conclusive.

Conclusion

In this project, we have presented a new class of AA strategy, DA and formulated DA in RL, a synthetic copy task and NLP text generation. Our DA formulation differs from AA because we are mainly interested in the temporal dimension which is often overlooked in the existing literature. We have identified the ingredients that we need to construct DA and its applicability in various scenarios. Here, we will summarize again the criterion DA needs to make it possible:

- The data itself needs to have a strong temporal dependency such that an input long in the past can also have an impact on its future.
- There needs to be an input-output coupling. Coming back to the RL setting, the input of one time step will not only influence the output of the future time steps but also the inputs of future time steps. Without this property, we might run into the situation like the simple copy task where the errors cannot propagate into the future.
- There have to be states which are very different from each other but have similar state values. This is to make sure that during the peace period, even if we change the state to something else, people might still consider these new states that what a well-behaving agent will choose to go to.

This project also opens up the door to lots of future research directions:

- Learning theory of RNNs: when we were doing the attack on the simple copy task, We noticed that the adversarial effects would not last long into the future and the network always seemed to reset its behavior after certain time steps. It would be interesting to look at the properties of RNNs' weight matrices to see what exactly can engender such a stable system and whether or not there exists any input that can make the system unstable.
- DA in RL: our DA is mainly motivated by its applications in RL after all as in many game environments that DA can pose new challenges to defend. One can certainly follow what we have developed with the addition of a generative model to test out DA's applicability in RL.
- AA along the temporal dimension: even though we could not get good performance for the peace period, we were still able to perturb words that are very far from the attacked words. Aspects such as how inputs can have an impact on their future output and the semantics of the attack vectors and peace periods are all worth exploring further.

Lastly, we want to recognize the difficulty of doing DA on discrete input domain. This additional layer of complexity hinders our ability to interpret the experiment results. Perhaps it would have been easier to test out DA on the auto-captioning task where we will be able to have continuous input and use RNNs for prediction, but this is left for future work as well.

Acknowledgment

In the end, I would like to thank my two beloved supervisors Dr. Mathieu Salzmann and Dr. François Fleuret for their much-needed advise and great sense of humor, without which this project would not have been be possible. I would also like to thank for my roommate Liu Chen for the many helpful discussions we had in pushing this project through.

Appendices

Listing 8.1: The top words in Wikitext-2

```
the, ,, ., <unk>, of, and, in, to, <eos>, a, =, ", was, The, @-@,
    that, as, 's, on, for, with, by, ), (, is, from, at, his,
   were, it, he, an, had, which, In, be, are, ;, their, but, not,
    first, -, also, its, :, or, have, one, been
```

Listing 8.2: Sample generated text for the model trained on Wikitext-2

```
=======================Beam search result========================
in the campaign . Only the `` remains in associated '' , it is
   the first type to stepped river in Lake Maratha , one nuclei
   of <unk> , John trance Resor , called Algiers <unk> wisdom is
   unknown but Household on the brooches . In spring office in
   the east is currently 200 <unk> et Boom 2 , as they larger
   attempts to have Yvonne any Peninsula . It <unk> for a member
   of related Ishmaelites who introduced off increased elements
   of greater Norway member of <unk> <unk> <unk> . Around 40
   hours lauded capped Art hospitals they tells


 52nd Love branding . However , it tied of 4 @.@ 4 % and then
    motion on Destiny 's <unk> hero . In his final mix of being
    beaten for African a special measurements reported , Three
    minutes translation , wet , 1947 ( 22 @.@ 08 , merge Gofraid
    ) and dramatic . Together in the floor or desk . Nevertheless
     , in Square Akrad , in with the aground to graphs after the
    Adventure chaired , rate of reach PR legislation were rock
    out of a 5 @-@ 1 guns . In South Angeles in 1969 ,

=======================Greedy search result=====================
that they were " <unk> " . The <unk> of the <unk> <unk> ( <unk> )
    , was a <unk> of <unk> , which was a <unk> of <unk> , which
   was a <unk> of <unk> , which was a <unk> of <unk> , which was
   a <unk> of <unk> , which was a <unk> of <unk> , which was a <
   unk> , which was a <unk> , which was a <unk> , which was a <
   unk> , which was a <unk> , which was a <unk> , which is a <unk
   > , which is a <unk> , which is
```

```
, and the <unk> of <unk> , and <unk> , and the <unk> <unk> <unk>
   , was a <unk> , which was a <unk> of <unk> , which was a <unk>
    , which was a <unk> , which was a <unk> , which was a <unk> ,
    which was a <unk> , which was a <unk> , which was a <unk> ,
   which was a <unk> , which was a <unk> , which was also known
   to be a planet . The kakapo 's <unk> , <unk> , and <unk> , and
    the <unk> <unk> , <unk> , and <unk>'
```

Listing 8.3: Sample generated text for the model trained on Wikitext-103

```
======================Beam search result======================
``indicates guidance of deliberately patient bodies . Stan
   bracteates forebearers in a collection of illnesses that do
   not feel against the repelled mistake , allowing blood
   products to fool him . <eos> Faulkner and his best friend ,
   Will shippers , are elected -- with procuring intelligence and
    art company to recorder . Today , instead of building the
   Indus Valley square mash shop wetland , it is passing by
   Infrared Park along their large street under San Diego Park
   and New York Airport , North Dakota . <eos> WDL , Lassiter 's
   firm , used ... weapons to create''


TID . This is an influence on the other more consumer categories
   of bringing the American diplomatic organization for two
   decades to find put on side steadily . <eos> Hawthorne 's own
   choice for the two years was heavily compromised , including
   exploring singles to the northeast Kaiser Circling Kapoor in
   England , and Orton 's fleet initiative before Elliott was
   subsequently landing on Grand Isle in summer . Britain had
   established their determination to direct Venice outside China
    but was once again renamed to take on close relations with
   Zanzibar . <eos> Once again , many British businessmen

======================Greedy search result====================
. <eos> <eos> = = = = = = = = = = = = = = = <eos> <eos> The first
    two years of the 20th century were the first to be released
   in the United States . The first two @-@ year @-@ old , the
   first @-@ person , was the first to be the first to be the
   first to be the first to be the first to be the first to be
   the first to be selected . The first two @-@ year @-@ old ,
   the former president of the United States , was

, and the first to be the first to be a member of the United
   States . <eos> <eos> = = = = = = = = = = = = = = = <eos> <eos>
    The first two years of the war were the first time in the
   United States . The first two @-@ year @-@ old , the first of
   the two men , was the first to be the first to be the first to
    be the first to be the first to be selected . The first two @
   -@ year @-@ old son , John'
```

Listing 8.4: Sample entries from Wikitext

```
= = Works = =

Criticism of Du Fu 's works has focused on his strong sense of
   history , his moral engagement , and his technical excellence
   .

= = = History = = =

Since the Song dynasty , critics have called Du Fu the ``poet
   historian'' ( <unk> <unk> <unk> ) . The most directly
   historical of his poems are those commenting on military
   tactics or the successes and failures of the government , or
   the poems of advice which he wrote to the emperor . <unk> , he
    wrote about the effect of the times in which he lived on
   himself , and on the ordinary people of China . As Watson
   notes , this is information `` of a kind seldom found in the
   officially compiled histories of the era '' .
Du Fu 's political comments are based on emotion rather than
   calculation : his <unk> have been <unk> as , `` Let us all be
   less selfish , let us all do what we are supposed to do'' .
   Since his views were impossible to disagree with , his <unk>
   expressed <unk> enabled his installation as the central figure
    of Chinese poetic history .

= = = <unk> engagement = = =

A second favourite epithet of Chinese critics is that of `` poet
   sage '' ( <unk> <unk> <unk> ) , a counterpart to the
   philosophical sage , <unk> . One of the earliest surviving
   works , The Song of the <unk> ( from around 750 ) , gives
   voice to the <unk> of a <unk> soldier in the imperial army and
    a clear @-@ sighted consciousness of suffering . These
   concerns are continuously articulated in poems on the lives of
    both soldiers and civilians produced by Du Fu throughout his
   life .
Although Du Fu 's frequent references to his own difficulties can
    give the impression of an all @-@ consuming <unk> , Hawkes
   argues that his `` famous compassion in fact includes himself
   , viewed quite <unk> and almost as an <unk> ''. He therefore
   ``lends grandeur '' to the wider picture by comparing it to "
   his own slightly comical triviality " .
Du Fu 's compassion , for himself and for others , was part of
   his general broadening of the scope of poetry : he devoted
   many works to topics which had previously been considered
   unsuitable for poetic treatment . Zhang Jie wrote that for Du
   Fu , ‘‘ everything in this world is poetry ’’ , Du wrote
   extensively on subjects such as domestic life , calligraphy ,
   paintings , animals , and other poems .

= = Life = =

Traditional Chinese literary criticism emphasized the life of the
    author when interpreting a work , a practice which Burton
   Watson attributes to ‘‘ the close links that traditional
   Chinese thought posits between art and morality ’’ . Since
   many of Du Fu ’s poems feature morality and history , this
   practice is particularly important . Another reason ,
   identified by the Chinese historian William <unk> , is that
   Chinese poems are typically <unk> , <unk> context that might
   be relevant , but which an informed contemporary could be
   assumed to know . For modern Western readers , ‘‘ The less
   accurately we know the time , the place and the circumstances
   in the background , the more liable we are to imagine it
   incorrectly , and the result will be that we either <unk> the
   poem or fail to understand it altogether ’’ . Stephen Owen
   suggests a third factor particular to Du Fu , arguing that the
    variety of the poet ’s work required consideration of his
   whole life , rather than the ‘‘ <unk> ’’ <unk> used for more
   limited poets .
```

```
Input: arrivial rate: $\lambda$, delay steps: $d$, pattern length: $l$, buffer length: $b$
Result: $x, y$
1  $i = 0, j = 0, isPattern = True$ ;
2  $t = x = y = [\ \ ]$ ;
3  $nextArrival = Possion(\lambda)$;
4  while $i < l$ do
5      if $isPattern$ and $i == nextArrival$ then
6          $nextArrival = i + Possion(\lambda)$;
7          $t.append(i)$ ;
8          $x.append(Uniform(-1, 1))$ ;
9      end
10     $i = i + 1$;
11 end
12 $x = interpolate(t, x)$ ;
13 $x = x + zeros(n)$ ;
14 Make $y_t = x_{t-d}$ ;
15 return $x, y$;
```

Algorithm 4: Copy task data generation

In line 13, we add n zeros to the end of vector x. In line 14, we fill also in zeros the first d elements of y to make the dimensions of x and y match.

List of Abbreviations

AA	Adversarial attack
BPTT	Back-propagation through time
DA	Delayed attack
DL	Deep learning
DNN	Deep neural networks
DQN	Deep Q network
FGSM	Fast gradient sign method
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
MDP	Markov decison process
RL	Reinforcement learning
RNN	Recurren neural network

Bibliography

Arpit, D., Kanuparthi, B., Kerg, G., Ke, N. R., Mitliagkas, I., and Bengio, Y. (2018). h-detach: Modifying the LSTM Gradient Towards Better Optimization. pages 1–19.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence modeling. *arXiv preprint arXiv:1412.3555*.

Collins, J., Sohl-Dickstein, J., and Sussillo, D. (2016). Capacity and Trainability in Recurrent Neural Networks. pages 1–17.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., and Jun, L. G. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.

Gilmer, J., Adams, R. P., Goodfellow, I., Andersen, D., and Dahl, G. E. (2018). Motivating the Rules of the Game for Adversarial.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and Harnessing adversarial examples. pages 1–11.

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. pages 1–43.

Hassan, H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., Huang, X., Junczys-Dowmunt, M., Lewis, W., Li, M., et al. (2018). Achieving Human Parity on Automatic Chinese to English News Translation. *arXiv preprint arXiv:1803.05567*.

Hausknecht, M. and Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. pages 29–37.

Hochreiter, S. (1991). Untersuchungen zu Dynamischen Neuronalen Netzen. *Diploma, Technische Universität München*, 91(1).

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Horgan, D., Barth-maron, G., Quan, J., Hessel, M., Budden, D., and Silver, D. (2018). Distributed Prioritized Experience Replay. pages 1–19.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial Attacks on Neural Network Policies.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement Learning With Unsupervised Auxiliary Tasks. *arXiv preprint arXiv:1611.05397*.

Jia, R. and Liang, P. (2017). Adversarial Examples for Evaluating Reading Comprehension Systems.

Kapturowski, S., Ostrovski, G., Quan, J., and Dabney, W. (2019). Recurrent Experience Replay in Distributed Reinforcement learning. *International Conference on Learning Representations (ICLR)*, pages 1–19.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

Lin, Y.-c., Hong, Z.-w., Liao, Y.-h., Shih, M.-l., Liu, M.-y., and Sun, M. (2017). Tactics of Adversarial Attack on Deep Reinforcement Learning Agents.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level Control Through deep Reinforcement earning. *Nature*, 518(7540):529–533.

Mnih, V., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., and Silver, D. (2016). Asynchronous Methods for Deep Reinforcement Learning. 48.

OpenAI (2018). Openai five.

Papernot, N., Mcdaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016a). The Limitations of Deep Learning in Adversarial Settings. *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 372–387.

Papernot, N., McDaniel, P., Swami, A., and Harang, R. (2016b). Crafting Adversarial Input Sequences for Recurrent Neural Networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pages 49–54. IEEE.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models Are Unsupervised Multitask Learners. *OpenAI Blog*, 1(8).

Schaul, T., Quan, J., Antonoglou, I., Silver, D., and Deepmind, G. (2016). Prioritized Experience Replay. pages 1–21.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge.

Szegedy, C., Bruna, J., Erhan, D., and Goodfellow, I. (2014). Intriguing Properties of Neural Networks. pages 1–10.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.