Adversarial Attacks on Structured Prediction

Aditya Shastry
University of Massachusetts
Amherst, MA, USA

Pratik Mehta
University of Massachusetts
Amherst, MA, USA

ABSTRACT

Deep Learning models can be led astray with adversarial examples, which are maliciously perturbed inputs. For a structured prediction task, specifically image-to-LaTeX, we demonstrate using targeted and non-targeted adversarial examples that the model can be fooled easily. We present the training procedure to generate the adversarial examples, and evaluation performed to demonstrate their effectiveness.

1 INTRODUCTION

The popularity and presence of Deep Learning models is on a steady rise. Due to their performance, Deep Learning based models form the state-of-the-art for many problems. So, they are used in various tasks like biometric authentication and autonomous vehicles. Despite their effectiveness, Deep Learning based models are vulnerable to Adversarial Examples. Adversarial Examples are inputs which are maliciously perturbed by minute additions, which are almost unnoticeable to humans, but can produce incorrect outputs. Adversarial examples can be generated using simple gradient and optimization based methods. They also generalize over multiple models which attempt to solve the same task. So, research into understanding adversarial examples is an important and urgent problem.

Structured Prediction is a class of machine learning, where the output is a structured object, instead of discrete classes. An example is Optical Character Recognition (OCR), which takes an image with natural language characters, and produces a sequence of characters which forms the plan text representation of the image. With the advancement in Deep Learning, the performance for OCR tasks have significantly increased. An example of OCR is to read details on a bank check, which is deposited by the bank's mobile application. Theoretically, adversarial examples can cause incorrect prediction of the amount field, which can be a disastrous outcome for the parties involved.

In this work, we attempt to generate and analyze adversarial examples for a Structured Prediction task, specifically image-to-LaTeX which is a subtask of OCR. This model takes as input as image which contains a mathematical formula and produces the corresponding LaTeX typeset as the output. Due to the constraints placed on the LaTeX typeset, this is a simpler task than performing OCR on images with natural language text. This task also presents us with a unique opportunity to render the output markup, given the LaTeX typeset. The pipeline of image-to-LaTeX-to-image offers better insights into the effects of adversarial examples. We use the model WYGIWYS [2] (What You Get Is What You See) as the target model for our adversarial attacks.

COMPSCI 660 Advanced Information Assurance, Spring 2018, Amherst, MA, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

[4] first described the lack of stability of neural networks against perturbed input images. Perhaps the most surprising property observed was the propensity of neural networks trained with disjoint datasets and different objective functions to mis-classify the same input images generated by adding minimal perturbations. This result has had widespread impact on the machine learning community by exposing the vulnerability of deep learning based systems to a highly generalizable and largely cheap method to compromise the effectiveness of black-box machine learning models.

Both the gradient-update based and fast-gradient approaches described in [4] have now been derived upon and widely used to generate adversarial perturbations to affect input images for a large number of image classification models - with good results (i.e. [3], [5] and [6]). Training models to generate adversarial examples using this method usually requires knowledge of the original model's objective function and gradients. While this can be overcome by using non-differentiable objectives or obfuscating gradients.

Another class of adversarial attacks depends on knowing the dataset used to train the original model. Such transfer-based attacks can be highly effective against ensembles of substitute models ([1]). Despite their strengths, it has been shown that simply training an augmented model containing adversarial examples is a relatively simple yet powerful measure against such attacks.

Decision-based adversarial attacks [1] are those that depend only on the final decision of the model such as top-1 class label or transcribed sentence.

There are limitations to studying adversarial attacks in the context of image recognition models. Firstly, the decision boundaries in any deep learning based models are difficult to conceptualize in the presence of non-linear activations in the final layer of the neural networks. Secondly, studying the effect of perturbations to better understand their effect on predictions is difficult. In light of this, [5] suggest that testing adversarial attacks on structured prediction models such as OCR systems gives us a wider net to throw, and a sequence of predictions can give a better indication of the effects of individual localized perturbations.

Our work involves understanding how adversarial attacks work, learning to produce adversarial images to fool a structured prediction models, and try to retrace the conclusions made by [5] albeit on a smaller data set. In the following sections, we see that our results are indicative of the same conclusions and that there is value in

studying adversarial attacks against OCR systems such as this one. The appendix shows the key parts of our algorithm in Lua Torch.

3 APPROACH

Suppose there exists a target model F, trained using parameters θ. For an input image X, the correct output is $Y = F_\theta(X)$. An adversarial example is an input $X^* = X + \delta$, which produces the output $F_\theta(X + \delta) \neq Y$. Here δ is the malicious perturbation, which is a small quantity that it doesn't change a human's interpretation of the inputs. Adversarial examples belong to the below two classes:

- Non-Targeted: The goal of this class is to produce any incorrect output, i.e. $F_\theta(X + \delta) \neq Y$
- Targeted: The goal of this class is to produce a specific incorrect output, i.e. $F_\theta(X + \delta) = Y^*$, where $Y^* \neq Y$

The subsequent subsections will describe a few methods to generate the adversarial examples. In the below descriptions, the symbol J denotes the loss function used to train the model F_θ.

3.1 Gradient based methods

This method is a single step method, where adversarial examples can be generated after a single step of computation. There are two methods which employ the gradient method for generating an adversarial example:

- **Fast Gradient Sign Method**
  For a non-targeted example, a δ is generated which maximizes the prediction loss, by following the gradient sign of the model loss function. This is demonstrated by the below equation:

$$\delta = \epsilon.\text{sign}(\nabla_x J(F_\theta(X, Y)))$$

  Similarly, for a targeted example, a δ is generated which minimizes the prediction loss for the adversarial label Y^*. This is demonstrated by the below equation:

$$\delta = -\epsilon.\text{sign}(\nabla_x J(F_\theta(X, Y^*)))$$

- **Fast Gradient Method**
  Unlike the Fast Gradient Sign Method, this method computes the perturbations in the direction of the gradient itself. So, a non-targeted example can be generated using the below equation:

$$\delta = \epsilon.\frac{\nabla_x J(F_\theta(X, Y))}{||\nabla_x J(F_\theta(X, Y))||}$$

  Similarly, a targeted example can be generated using the below equation:

$$\delta = -\epsilon.\frac{\nabla_x J(F_\theta(X, Y^*))}{||\nabla_x J(F_\theta(X, Y^*))||}$$

In this method, multiple examples can be generated by tuning the value ϵ. This acts as a control for the magnitude of the perturbations.

3.2 Optimization method

Optimization method finds the best perturbation by iteratively optimizing an objective function. For a non-targeted adversarial example, the below function is optimized:

$$\delta = \arg\min_\delta \lambda||\delta||_p - J(F_\theta(X + \delta, Y))$$

Similarly, for a targeted adversarial example, the below function is optimized:

$$\delta = \arg\min_\delta \lambda||\delta||_p + J(F_\theta(X + \delta, Y^*))$$

Here, λ is the regularization parameter, and p denotes the type of norm on which the regularization is to be applied. Both of them are tunable hyperparameters.

4 TARGET MODEL

The target model, which we attempt to mislead by using adversarial examples, is the model What You Get Is What You See [1].

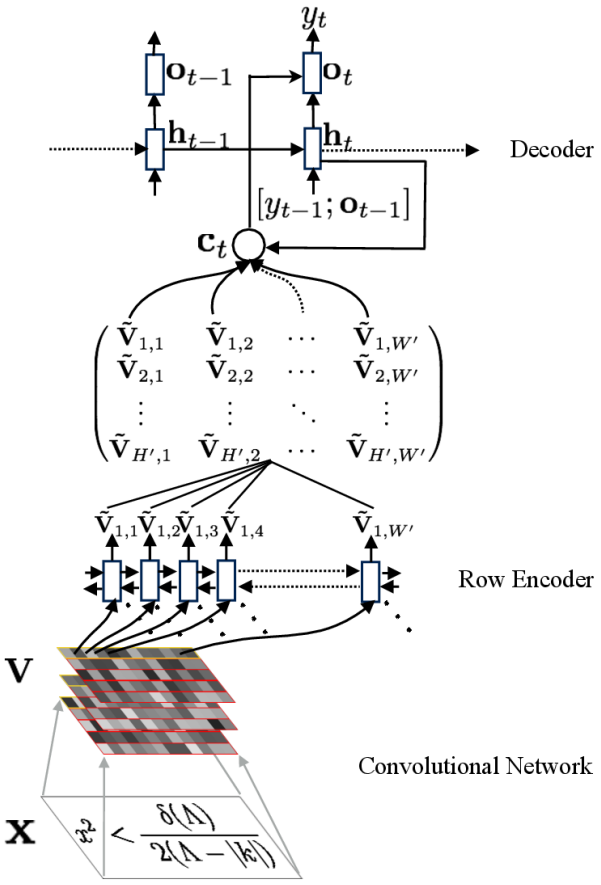

Figure 1: Architecture for WYGIWYS

Figure 1 describes the architecture of the target model. It comprises of a series of six convolution layers, which extract the necessary features from the input images. Each row of the image features extracted by the convolution layers is encoded using a Bi-Directional LSTM. This puts the complete encoding of the input images in a common feature space. An LSTM based decoder is used to decode the corresponding representation to produce the sequence of LaTeX typesets.

[1] https://github.com/harvardnlp/im2markup

For our experiments, we utilize the pre-trained model made available by the authors of WYGIWYS.

5 DATASET

The dataset used for our experiments is a subset [2] derived from the image-to-LaTeX dataset. It comprises of a set of images, their corresponding LaTeX typesets, and the train, validation, and test files. The images are grouped similar to sizes [3] to ease the data batching process.

For our experiments, we utilize the images, and their corresponding formulas, present in the test file, which is 93 images. The average length of the formulas in the dataset is 68, and the maximum length is 406. The lengths are based on counts of individual tokens in the formulas. Our final set of test images consists of 96 images.

6 EXPERIMENTAL SETUP

Our experiments explored on the Optimization Method to obtain the required perturbations for the input images. We used a standard Gradient Descent optimizer, with a constant learning rate of 0.3. To compare our results with [5], we follow their lead and report results without regularization. Surprisingly, the magnitude of perturbations did not blow up despite the fact and remained largely unclear to the human eye. We report results on four experiments - one in the context of non-targeted adversarial attacks, and the last three pertain to substitution, addition and deletion-based targeted attacks.

7 RESULTS

7.1 Non-targeted attacks

Non-targeted attacks aim to lead the target model astray from its predictions. We learned perturbations for the input images for 800 iterations. These perturbations learned for each image in the test set was then added to the images, and we performed inference in the target model once again. The resulting BLEU-1 scores and exact-match accuracies - i.e. the number of images for which the entire sequence of predicted labels is the same as the expected one - are both observed to have dropped when using the adversarially generated test examples.

Experiment	BLEU - 1
Original Model	80.5
Adversarial attack	77.13

Table 1: Drop in model performance due to non-targeted adversarial attack

Table 1 suggests that the added perturbations bring down model accuracy. A motivated attacker could continue training until convergence, making the method quite practical. The relatively smaller drop in BLEU-1 scores suggests that the sequence of labels for every image was not severely modified. Non-targeted attacks are perhaps best deployed in scenarios where a single change in the sequence of labels proves catastrophic. We also observed that the overall exact=

7.2 Targeted attacks

A successful targeted attack leads the target model to make predictions consistent with the attacker's goals. Therefore, generating adversarial examples in the context of a targeted attack requires specifying a sequence of target labels that the attacker would like the model to predict. The goal therefore is to generate perturbations that achieve this goal without adding very obvious visual cues to a suspecting human supervisor.

To generate adversarial examples for the targeted attacks, we focus on permuting exactly one label out of the gold sequence of labels. These permutations could involve either adding a new label, deleting an existing label, or substituting a label with another from the shared vocabulary (set of latex commands). Our observations were consistent with our expectation that deletion attacks are significantly harder than substitution or insertion attacks. We do not have a clear understanding of why this is the case. This observation could only have been made in the context of structured prediction models.

Experiment	BLEU - 1
Substitution	77.63
Addition	78.08
Deletion	79.21

Table 2: Drop in model performance due to targeted adversarial attacks

7.3 Visualizing perturbations

We visualize the perturbations from our experiments in the non-targeted examples case to check if locality of perturbations is correlated with the labels that end up permuted. While we did not find a clear link between spatial locality of perturbations and the corresponding predictions, we believe that a more rigorous set of experiments controlling for masked perturbations in sections of the image should make this clearer. We present two examples from the non-targeted experiments below.

[2] https://github.com/harvardnlp/im2markup/tree/master/data/sample

[3] Width-Height groups used are (120,50), (160,40), (200,40), (200,50), (240,40), (240,50), (280,40), (280,50), (320,40), (320,50), (360,40), (360,50), (360,60), (360, 100), (400,50), (400,160), (500,100)

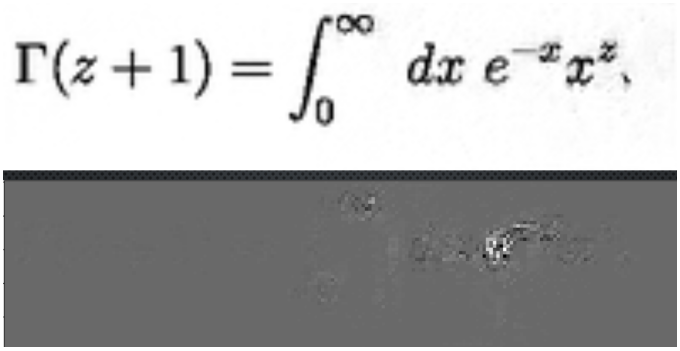

Figure 2: Perturbations for 'insertion' attack. The perturbations result in the target model predicting an extra "6" after the integral sign.

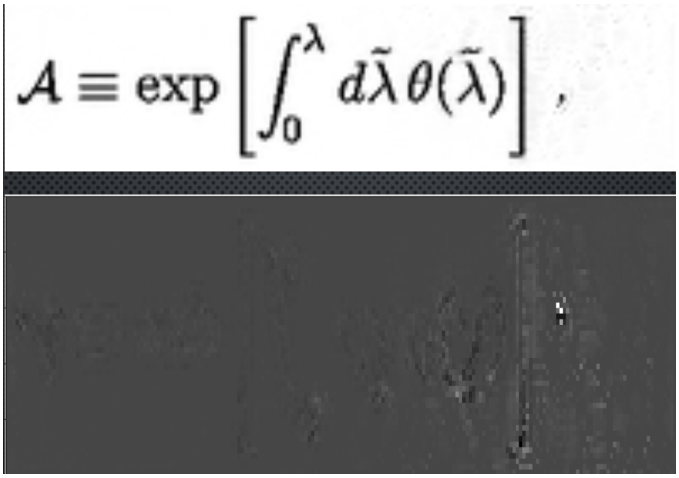

Figure 3: Perturbations for 'substitution' attack. The perturbations result in the target model predicting a "bar" in place of the "tilde" in the formula.

From the above examples, a clear correlation between where the perturbations appeared and the corresponding position of change in the prediction sequence is not forthcoming. We recommend further experiments to study this phenomenon in detail as future work.

8 CONCLUSION

In conclusion, our results largely agree with [5]. More specifically, we observed the comparative difficulty of training adversarial perturbations to perform deletion, addition and insertion attacks and the sharp decline in results as compared to the original model. Further lines of inquiry include selectively 'muting' perturbations on one side of the image to check their effect on the sequence of predictions to understand how locality affects predictions. It should also be possible to test out ways to detect and deter the effect of adversarial perturbations in the target model. Such an effort would likely require more analysis of the magnitude and position of perturbations and in the case of an image-to-latex task, we predict that a relatively simple method to detect noise in the 'white-space' part of the images should prove helpful.

REFERENCES

[1] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2017. Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models. *arXiv preprint arXiv:1712.04248* (2017).

[2] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. 2016. Image-to-Markup Generation with Coarse-to-Fine Attention. *arXiv preprint arXiv:1609.04938* (2016).

[3] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

[4] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*. http://arxiv.org/abs/1312.6199

[5] James Wei. 2017. Adversarial Examples for Visual Decompilers. UCB/EECS-2017-81 (May 2017). http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-81.html

[6] Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. 2017. Adversarial Examples: Attacks and Defenses for Deep Learning. *CoRR* abs/1712.07107 (2017). arXiv:1712.07107 http://arxiv.org/abs/1712.07107

Appendices

A Non-Targeted Adversarial attacks

```
function model:adversarial_step(batch, adversarial_phase, beam_size
    , trie, data_path)

    local input_batch = localize(batch[1])
    local target_batch = localize(batch[2])
    local target_eval_batch = localize(batch[3])
    local num_nonzeros = batch[4]
    local img_paths

    -- load adversarial perturbations for the input images
    local adversarial_perturbations_file_names = batch[5]
    local adversarial_perturbations = torch.Tensor(input_batch:size
        ()):zero()
    adversarial_perturbations = localize(
        load_adversarial_perturbations(
        adversarial_perturbations_file_names,
        adversarial_perturbations, data_path, 0))
    local input_batch_backup = localize(batch[1])

    -- augment input batch by adding perturbations
    input_batch:add(adversarial_perturbations)

    -- Code for forward pass till the decoder LSTM

    -- perform prediction for each LSTM Cell
    for t = target_l, 1, -1 do
        local pred = self.output_projector:forward(preds[t])
        max_pred, max_pred_indices = torch.max(pred, 2)
        pred_labels = max_pred_indices:select(2,1)

    -- Compute loss for the batch
    loss = loss + self.criterion:forward(pred, pred_labels)/
        batch_size

    -- Code for backward pass till the CNN layers

    -- Compute gradients with respect to input
    local update_grads = self.cnn_model:backward(input_batch,
        cnn_final_grad)
```

```
    -- Perform the gradient <> step
    adversarial_perturbations:add(self.optim_state.learningRate,
       update_grads)

    -- Correct perturbations by clamping the produced image
    local corrected_adversarial_perturbations = torch.clamp(
       torch.add(input_batch_backup, adversarial_perturbations), 0,
        255) - input_batch_backup

    -- Save the computed perturbations
    save_adversarial_perturbations(
       adversarial_perturbations_file_names,
       corrected_adversarial_perturbations, data_path)

    return loss, {num_nonzeros, accuracy}

end
```

B Targeted Adversarial attacks

```
function model:adversarial_step(batch, adversarial_phase, beam_size
   , trie, data_path)

    local input_batch = localize(batch[1])
    local target_batch = localize(batch[2])
    local target_eval_batch = localize(batch[3])
    local num_nonzeros = batch[4]
    local img_paths

    -- load adversarial perturbations for the input images
    local adversarial_perturbations_file_names = batch[5]
    local adversarial_perturbations = torch.Tensor(input_batch:size
       ()):zero()
    adversarial_perturbations = localize(
       load_adversarial_perturbations(
       adversarial_perturbations_file_names,
       adversarial_perturbations, data_path, 0))
    local input_batch_backup = localize(batch[1])

    -- augment input batch by adding perturbations
    input_batch:add(adversarial_perturbations)
```

```
    -- Code for forward pass till the decoder LSTM

    -- perform prediction for each LSTM Cell
    for t = target_l, 1, -1 do
        local pred = self.output_projector:forward(preds[t])
            local pred_labels = target_eval[t]

    -- Compute loss for the batch
    loss = loss + self.criterion:forward(pred, pred_labels)/
       batch_size

    -- Code for backward pass till the CNN layers

    -- Compute gradients with respect to input
    local update_grads = self.cnn_model:backward(input_batch,
       cnn_final_grad)

    -- Perform the gradient <> step
    adversarial_perturbations:add(-1*self.optim_state.learningRate,
        update_grads)

    -- Correct perturbations by clamping the produced image
    local corrected_adversarial_perturbations = torch.clamp(
       torch.add(input_batch_backup, adversarial_perturbations), 0,
        255) - input_batch_backup

    -- Save the computed perturbations
    save_adversarial_perturbations(
       adversarial_perturbations_file_names,
       corrected_adversarial_perturbations, data_path)

    return loss, {num_nonzeros, accuracy}

end
```