Large Language Models Final Project Proposal: SAE-Informed Adversarial Perturbations for LLMs

Submitted by: Brendan McKinley

April 14, 2025

Context

Adversarial perturbations are deliberately chosen small changes to a model's input data which are designed to produce large changes in the model's output. These perturbations can be used to trick a model into producing erroneous results, or to highlight the areas in which the model is vulnerable to unwanted manipulation [2] [4].

Mathematically, designing adversarial perturbations requires solving the following optimization problem:

Let $\{\mathcal{V}, \|\cdot\|\}$ denote the normed vector space in which some given input data x resides. Let $f(x, \theta)$ denote the model under test, where x represents the model's inputs and θ represents the model's parameters. Then the ideal adversarial perturbation for x will be given by

$$\min_{\|\delta\|}\{x + \delta \text{ such that } f(x + \delta, \theta) \neq f(x, \theta) \text{ and } x + \delta \in \mathcal{V}\} \qquad [1]$$

Three approaches exist for designing adversarial perturbations for LLMs (and, more generally, for adversarial attacks on machine learning models in general): white box, in which the attacker is aware of the model's architecture, parameters, and the methods and data used to train it; black box, in which the attacker is entirely unaware of this information; and gray box, in which the attacker has some awareness of this information (e.g., access to the model's API). For this project, I propose a white-box approach, with a sparse auto-encoder that has already been trained on the model under test.

The aforementioned minimization problem becomes significantly more complex when we constrain the possible perturbations using the rules of human language. English, for example, induces a notion of semantic "nearness" with synonyms – words which can be thought of as small perturbations of a given input word. Changing the encoded vectors is mathematically straightforward, but what if we want to perturb at the input prompt level, with an adversarial prompt composed entirely of English words?

One example of word-based adversarial perturbations for neural networks is provided by Meng and Wattenhofer in their paper "A Geometry-Inspired Attack for Generating Natural Language Adversarial Examples". The authors detail a method for developing adversarial attacks on language classification models. This method uses the `DeepFool` algorithm to find a synonyms for a given input prompt which are close to the model's decision boundary, or the threshold which separates one semantic concept from another [3]. Meng and Wattenhofer are able to successfully trick the model under test into incorrectly classifying various input prompts without replacing more than a few words in the input prompt with adversarial synonyms.

For this attack, deep neural networks are tasked with classifying a prompt string. First, a word saliency score is computed for each individual word in the prompt by evaluating how the classification changes when a given word is replaced with another word from outside of the vocabulary. The word which yields the highest changes in the model's output is assigned the highest saliency score. Next, a group of synonyms for the word with the highest saliency score is compiled using Princeton's `WordNet` tool. Finally, the synonym

with the largest projection onto r_i, the vector between the original text vector and the decision boundary (which is determined using the `DeepFool`, is selected, and this synonym replaces the word with the highest saliency score in the prompt. This process is repeated until the model is fooled into wrongly classifying the text input. The authors provide the code they wrote for this paper as a GitHub repository, making it accessible for the purposes of this project.

Observations

Meng and Wattenhofer evaluate their method's performance against a convolutional neural network and a recurrent neural network. Approximating these models' respective decision boundaries can be done using an algorithm (in this case, DeepFool [**?**]) which leverages the model's loss function to iteratively move perturbations close to the decision boundaries of the model. However, this approach makes less sense in the context of a transformer model, particularly a large language model like OpenAI's `gpt2-small`, whose decision "geometry" is impossible to decipher as it's compressed in parameter-space.

Transformer architectures present unique complexities for adversarial defense that differ from traditional neural network models. Unlike convolutional neural networks used in image classification, transformers rely on attention mechanisms that create intricate, context-dependent representations of input data. The self-attention mechanism means that each token's representation is dynamically computed based on its relationships with all other tokens in the sequence, creating a highly context-sensitive mapping.

Sparse auto-encoders provide a potential solution. SAEs embed the encoded vectors used by the model in a much higher-dimensional space, providing a window into where feature activations actually occur. A language model's "geometry" is more easily understood in this higher-dimensional space, and a given vector's alignment with or distance from a particular concept vector may be useful for designing adversarial perturbations.

To be clear, concept vectors as learned by an SAE are not analogous to decision boundaries for a large language model. However, if we iteratively select synonyms which move the prompt as far away as possible from its most salient word's dominant feature, we may be able to more efficiently design adversarial perturbations to an input prompt which fool or confuse the model than by selecting those perturbations at random.

Description/Proposal

For my final project, I plan to adapt the geometry-informed adversarial perturbation approach proposed by Meng and Wattenhofer for use against a large language model. I will test using OpenAI's `gpt2-small`. Word saliency scores for a given prompt will be computed similarly to the method described above. Instead of using the `DeepFool` algorithm to iteratively move perturbations towards a model's decision boundary, I will determine the strongest feature vector of the word with the highest saliency score, using a sparse auto-encoder trained on `gpt2-small` and provided by Neuronpedia's API. I will then select the synonym with the weakest activation for this feature vector, and will replace the highest-scoring word with this synonym. By iteratively moving

Hypothesis

I propose that this method for designing adversarial perturbations will provide significant advantages over randomly-selected synonyms for generating large changes in the model's output for a given prompt string.

Test method

I will test the aforementioned adversarial attack against `gpt2-small`. For a given list of input strings and a fixed number of iterations, I will compare the output of the SAE-informed attack with the output for a random attack, where a random synonym is selected for the word with the highest saliency score. I will compare the effectiveness of the SAE-informed attack against the random attack, evaluating the effectiveness mathematically by comparing the suggested next word for both perturbed input prompts with the suggested next word with the original input. I will represent this effectiveness as a percentage called "adversarial success" which will be the ratio between the number of times that the perturbed inputs resulted in changes to the model's output and the total number of attempted perturbed inputs.

Expected outcome

I expect that using an SAE-informed approach to designing adversarial perturbations will have a greater adversarial success rate than the randomly-selected perturbations.

Minimal viable example description

For the minimal viable example of this project, I will write two methods and a simple test file which demonstrates that they work. Both methods will use Neuropedia's API and their `gpt2-small` sparse auto-encoder. The first method will determine the strongest feature activations for a given text input. The second method will choose the word with the weakest activation for a particular feature from a given list of words. These methods will be fundamental to designing the adversarial

Update

Minimum viable example completed: `https://github.com/brendan-e-mckinley/sae-informed-perturbations/tree/main`

Code:

Listing 1: Test

```
from utils import getStrongestFeatureForWord, getWeakestWordForFeature

model_id = "gpt2-small"
target_word = "upset"
syonyms = ["trouble",
"perturb",
"discompose",
"unsettle",
"disconcert",
"distress",
"discountenance",
"dismay",
"disquiet",
"worry",
"bother",
"plague"]

layer, index = getStrongestFeatureForWord(model_id, target_word)
weakest_synonym = getWeakestWordForFeature(model_id, layer, index, syonyms)

print("Weakest␣synonym:␣", weakest_synonym)
```

Listing 2: Neuronpedia API Interaction Methods

```
import requests

def getWeakestWordForFeature(model_id, layer, index, synonyms):
    # Initialize variables
    weakest_word = None
    smallest_max_value = float('inf')

    # Request params for all requests
    url = "https://www.neuronpedia.org/api/activation/new"
    headers = {
        "Content-Type": "application/json"
    }

    for synonym in synonyms:
      # Build the request
      payload = {
          "feature": {
          "modelId": model_id,
          "source": layer,
          "index": index
        },
        "customText": synonym
      }

      response = requests.post(url, headers=headers, json=payload)

      # Handle the response
      if response.status_code == 200:
          data = response.json()
          max_value = data.get("maxValue", [])

          print(synonym + " value: ", max_value)

          # Check if this is the smallest max_value so far
          if max_value is not None and max_value < smallest_max_value:
              smallest_max_value = max_value
              weakest_word = synonym
      else:
          print("Request failed:", response.status_code)
          print("Response content:", response.content.decode())

    return weakest_word

def getStrongestFeatureForWord(model_id, target_word):
    source_set = "res-jb"  # Pretrained SAE set
    layer = 6              # A mid-level layer where concept features tend to emerge

    # Compose the layer ID string expected by the API
    selected_layer = f"{layer}-res-jb"

    # Build the request
    url = "https://www.neuronpedia.org/api/search-all"
    headers = {
        "Content-Type": "application/json"
    }
    payload = {
        "modelId": model_id,
        "sourceSet": source_set,
```

```
    "text": target_word,
    "selectedLayers": [selected_layer],
    "sortIndexes": [1],          # Sort by importance
    "ignoreBos": False,
    "densityThreshold": -1,      # No threshold
    "numResults": 50             # Top 50 features
}

# Send the request
response = requests.post(url, headers=headers, json=payload)

# Handle the response
if response.status_code == 200:
    data = response.json()
    results = data.get("result", [])
    top_result = results[0]
    top_layer = top_result['layer']
    top_index = top_result['index']
    # print(f"Top concept features for the input: '{target_word}'\n")
    # for feature in results:
    #     index = feature['index']
    #     print(f"Index: {feature['index']}, Max Value: {feature['maxValue']}")

    return top_layer, top_index
else:
    print("Request failed:", response.status_code)
    print("Response content:", response.content.decode())

# Build a new request to understand concept feature
# url =  "https://www.neuronpedia.org/api/vector/get"
# headers = {
#     "Content-Type": "application/json"
# }
# payload = {
#     "modelId": model_id,
#     "source": top_layer,
#     "index": top_index
# }

# response = requests.post(url, headers=headers, json=payload)
# if response.status_code == 200:
#     data = response.json()
#     result = data.get("hookName")
#     hook_name = result.get("hookName")
#     print(f"Top Concept Feature Hook Name: {hook_name}")
# else:
#     print("Request failed:", response.status_code)
#     print("Response content:", response.content.decode())
```

What remains: I still need to compute saliency scores for a given input prompt and evaluate the perturbation methods' success against the model.

Reflection: Applying the dent defense to a transformer model is still intriguing to me, but I realized that it wasn't feasible for this project. I think choosing to look into a geometry-based adversarial perturbations was the right move, because it's helping me to think about connections between the model's geometry and representations of model features in higher-dimensional spaces, and what sparse auto-encoders can be used for.

LLM Logs for Update

https://claude.ai/share/77d8c833-88ed-407a-ad30-640b75276cb8

https://claude.ai/share/08658e6c-5dbe-4163-9161-633726269da0

https://claude.ai/share/53cf0e73-ccaa-4465-9c5b-65bc680f8f03

References

[1] Hervé Chabanne, Vincent Despiegela, Stéphane Gentrica, and Linda Guiga. Dynamic autoencoders against adversarial attacks. Procedia Computer Science, https://pdf.sciencedirectassets.com/280203/1-s2.0-S1877050923X00040/1-s2.0-S1877050923006397/main.pdf, 2023.

[2] Zico Kolter. Ten years hence lecture: Adversarial attacks on large language models. YouTube, https://www.youtube.com/watch?v=Ih34Ow9mdSY, 2024.

[3] Zhao Meng and Roger Wattenhofer. A geometry-inspired attack for generating natural language adversarial examples. In *Proceedings of the 28th International Conference on Computational Linguistics*, 2020.

[4] Zeyu Yang, Zhao Meng, Xiaochen Zheng, and Roger Wattenhofer. Assessing adversarial robustness of large language models: An empirical study. https://arxiv.org/html/2405.02764v1, 2024.