

# Reverse Engineering AI: Basics

May 13, 2024

ⓘ This conversation may reflect the link creator's personalized data, which isn't shared and can meaningfully change how the model responds.

I'm writing a blog post about reverse engineering an MLP. Can teach me all I need to know about AI reverse engineering at a high level?

Certainly! Writing a blog post about reverse engineering an MLP (Multilayer Perceptron) can be quite fascinating. Here's a high-level overview of AI reverse engineering:

## What is Reverse Engineering in AI?

**Reverse engineering** in the context of AI involves dissecting and analyzing a trained machine learning model to understand its internal structure, behavior, and decision-making process. This can be useful for various reasons such as:

- **Model verification:** Ensuring the model behaves as expected.
- **Security analysis:** Identifying vulnerabilities, backdoors, or biases.
- **Intellectual property:** Understanding proprietary algorithms.
- **Debugging:** Diagnosing and fixing issues in model performance.

## Key Concepts in AI Reverse Engineering

### 1. Model Architecture:

- Understand the architecture of the MLP, which typically consists of an input layer, one or more hidden layers, and an output layer.
- Each layer contains neurons (nodes) that apply weights to inputs and pass them through an activation function.

### 2. Weights and Biases:

- Weights are the parameters that the model learns during training.
- Biases are additional parameters that adjust the output along with the weighted sum of inputs.
- These parameters are crucial for understanding how the model makes decisions.

### 3. Activation Functions:

- Activation functions introduce non-linearity into the model, enabling it to learn complex patterns.
- Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.

### 4. Backpropagation:

- The training algorithm used to adjust weights and biases.
- Understanding backpropagation helps in figuring out how the model learns and where it might go wrong.

## 5. Loss Function:

- The function used to evaluate the model's performance.
- Common loss functions for classification tasks include Cross-Entropy Loss and Mean Squared Error for regression tasks.

## Steps in Reverse Engineering an MLP

### 1. Model Extraction:

- Obtain the model's architecture, weights, and biases.
- This might involve accessing the model file (e.g., .h5, .pb, .pt) or intercepting the model during its execution.

### 2. Disassembling the Model:

- Break down the model into its constituent parts (layers, neurons, connections).
- Use tools like TensorFlow, PyTorch, or ONNX to visualize and manipulate the model structure.

### 3. Analyzing Parameters:

- Inspect the weights and biases to understand how the model processes inputs.
- Look for unusual patterns or anomalies that could indicate a backdoor or vulnerability.

### 4. Behavioral Analysis:

- Feed various inputs to the model and observe the outputs.
- Identify inputs that cause unexpected or suspicious behavior.

### 5. Interpretability Techniques:

- Use techniques like saliency maps, SHAP (SHapley Additive exPlanations), and LIME (Local Interpretable Model-agnostic Explanations) to understand the model's decision-making process.

### 6. Security and Robustness Testing:

- Test the model for adversarial examples, which are inputs designed to deceive the model.
- Perform penetration testing to identify potential security flaws.

## Tools and Techniques

- **Frameworks:** TensorFlow, PyTorch, Keras, ONNX.
- **Visualization Tools:** Netron, TensorBoard.
- **Adversarial Testing:** Foolbox, CleverHans.
- **Interpretability Tools:** SHAP, LIME, Integrated Gradients.

## Example Workflow

### 1. Extract Model:

```
python
```

```
from tensorflow.keras.models import load_model

model = load_model('model.h5')
```

## 2. Inspect Layers:

```
python

for layer in model.layers:
    print(layer.name, layer.input_shape, layer.output_shape)
```

## 3. Analyze Weights:

```
python

weights, biases = model.layers[0].get_weights()
print(weights, biases)
```

## 4. Visualize Model:

```
python

import netron

netron.start('model.h5')
```

## Conclusion

Reverse engineering an MLP involves understanding its architecture, analyzing its parameters, and testing its behavior to uncover hidden aspects or vulnerabilities. This high-level overview should provide a good foundation for your blog post. Feel free to dive deeper into specific areas or tools as needed!

Are there any standard workflows or procedures for finding backdoors in MLPs through reverse engineering?

Yes, there are standard workflows and procedures for finding backdoors in MLPs through reverse engineering. These procedures typically involve a combination of model extraction, behavioral analysis, and various security testing techniques. Here's a step-by-step workflow that can guide you in identifying backdoors in MLPs:

## Standard Workflow for Finding Backdoors in MLPs

### 1. Model Extraction and Initialization:

- **Obtain the Model:** Secure the trained model's architecture and parameters (weights and biases). This could be from a file (e.g., `.h5`, `.pb`, `.pt`) or by intercepting the model during deployment.
- **Load the Model:** Use a deep learning framework to load and inspect the model.

```
python
```

```
from tensorflow.keras.models import load_model
model = load_model('model.h5')
```

## 2. Static Analysis:

- **Inspect Model Architecture:** Review the layers, input shapes, and output shapes.

```
python

for layer in model.layers:
    print(layer.name, layer.input_shape, layer.output_shape)
```

- **Analyze Weights and Biases:** Look for unusual patterns in the weights and biases. Significant deviations might indicate tampering.

```
python

for layer in model.layers:
    weights, biases = layer.get_weights()
    print(weights, biases)
```

## 3. Behavioral Analysis:

- **Test with Known Inputs:** Feed the model with a variety of known inputs and observe the outputs. Document any unexpected or suspicious outputs.
- **Compare Against Clean Model:** If possible, compare the suspicious model's behavior against a clean version of the model.

```
python

clean_output = clean_model.predict(input_data)
suspect_output = model.predict(input_data)
if clean_output != suspect_output:
    print("Potential anomaly detected.")
```

## 4. Adversarial Testing:

- **Generate Adversarial Examples:** Use tools like Foolbox or CleverHans to create adversarial inputs that could exploit potential backdoors.

```
python

from foolbox import PyTorchModel, accuracy, samples
from foolbox.attacks import FGSM

fmodel = PyTorchModel(model, bounds=(0, 1))
attack = FGSM()
adversarial_examples = attack(fmodel, inputs, labels)
```

## 5. Input Perturbation Analysis:

- **Gradient-Based Analysis:** Analyze the gradients of the output with respect to the input to identify sensitive regions. Use techniques like saliency maps.

```
python
```

```
import tensorflow as tf

with tf.GradientTape() as tape:
    inputs = tf.convert_to_tensor(inputs)
    tape.watch(inputs)
    predictions = model(inputs)
    loss = compute_loss(predictions, labels)

gradients = tape.gradient(loss, inputs)
```

## 6. Model Interpretability Techniques:

- **Local and Global Interpretability:** Use SHAP, LIME, or Integrated Gradients to understand how different features contribute to the model's decisions.

```
python
```

```
import shap

explainer = shap.DeepExplainer(model, background_data)
shap_values = explainer.shap_values(input_data)
shap.summary_plot(shap_values, input_data)
```

## 7. Data Poisoning Detection:

- **Training Data Analysis:** Analyze the training data for anomalies or poisoned samples that could have introduced a backdoor.
- **Re-train and Compare:** Retrain the model on clean data and compare its performance and behavior to the suspect model.

## 8. Runtime Monitoring:

- **Monitor Inference:** During inference, monitor the input-output pairs for signs of backdoor activation.
- **Set Thresholds:** Implement thresholds and anomaly detection mechanisms to flag suspicious behavior.

## Conclusion

By following this structured workflow, you can systematically investigate and identify potential backdoors in MLPs. Each step combines various analytical techniques to uncover hidden vulnerabilities or malicious modifications. This approach ensures a comprehensive examination of the model's integrity and security.

[| Terms of use](#) | [| Privacy policy](#) |