Pseudo Code for Symbolic Interval Propagation:

1. Inputs:
    a. network: tested a neural network
    b. input: input interval
2. Variables:
    a. equation = (equation_upper, equation_lower)
    b. R[numLayer][layerSize]: For storing cache mask matrix that will be used in backward Propagation.
    c. Concrete value(equation_upper,equation_lower): The result of the symbolic equation through ReLU Nodes.
3. Procedure SymbolicIntervalAnalysis(network, input):
    a. Initialize eq = (equation_upper, equation_lower)
    b. Initialize R[numLayer][layerSize]
4. Logic:

```
for each layer = 1 to number_of_layers {
    Update equation = matrix multiplication equation of (weight * equation)
    //Update output ranges for each node in the layers
    Condition on if layer is not the last_layer {
            for i = 1 to layer_size[layer] {
              Condition on if Concrete value of (equation_upper[i]) ≤ 0 {
             //Goes to next layer
                R[layer][i] = [0, 0] → Scenario: d(relu(x))/dx = [0, 0]
                equation_upper[i] = equation_lower[i] = 0
             Condition on else if Concrete value of (equation_lower[i]) ≥ 0 {
             //Goes to next layer
                R[layer][i] = [1, 1] → Scenario: d(relu(x))/dx = [1, 1]
              Else
                R[layer][i] = [0, 1] → Scenario: d(relu(x))/dx = [0, 1]
                equation_lower[i] = 0
                // Handling ReLU activation by, if equation_upper[i]> 0, we pass the
                //upper equation on to the next layer. Otherwise, we concretize it as
                //equation_upper(X)
               if equation_upper[i] ≤ 0
                   equation_upper[i] = concrete value of (equation_upper[i])
  else
      return R, output → Return R matrix and output
```

5. Understanding the logic:

The symbolic interval analysis algorithm for neural networks begins by initializing crucial variables and data structures such as the network itself, input parameters, as well as matrices like 'equation' and 'R'. Through a loop iterating over each layer of the neural network, the algorithm updates matrices using equations that involve weights and intervals, facilitating computations within the network. Within each layer, conditions are checked for every node to update the output ranges ('R') based on specific criteria. A significant aspect involves handling Rectified Linear Unit (ReLU) activation, wherein conditions related to ReLU are assessed, and subsequent output range adjustments are made according to various scenarios encountered,

denoted as d(relu(x))/dx =[0,0], d(relu(x))/dx =[1,1], and d(relu(x))/dx =[0,1]. Ultimately, the algorithm produces output, typically 'R', and calculated output bounds, determined by certain conditions reached during the process.

Pseudo Code of Backward propagation for gradient interval Inputs:

1. Inputs:
    a. network: tested a neural network
    b. R: gradient mask
2. Initialize upper and lower gradient bounds
    a. gradient_upper = gradient_lower = weights[last_layer]
3. Logic:

```
for each layer = number_of_layers-1 to 1 { → in reverse order
   for each i = 1 to layerSize[layer] {
      g = [gradient_upper, gradient_lower] → interval of lower and upper values
      g = hadamard_product(R[layer][i], g) → Interval of hadamard product
      g = matrix_multiplication(weights[layer], g) → Interval matrix multiplication
   }
}
return g
```

4. Understanding the logic:

Although the symbolic interval analysis approach yields rather tight boundaries for neural network output, it may not produce sufficiently tight intervals for property verification, especially when dealing with huge input intervals that lead to several concretizations. Influence analysis and monotonicity are two optimization techniques that are introduced to handle this problem, coupled with the iterative interval refinement technique. Until the output intervals fulfill the required security property or a timeout happens, input intervals are split iteratively in the baseline iterative refinement process. To improve the output boundaries, this procedure, which is depicted as a bisection tree, continuously divides input intervals into subintervals. Concrete outputs are verified against the security property to identify adversarial cases. This process results in the tagging of particular sub-intervals that contain adversarial instances. Symbolic interval analysis on these sub-intervals completes the procedure. The adjustments also try to lower average bisection depths. Influence analysis emphasizes the input features that have the most influence on the output by ranking which input intervals to bisect first using gradients or Jacobians. By identifying the output's monotonic behavior within input intervals, monotonicity analysis reduces the number of splits required for security property checks by allowing intervals to be replaced with concrete values when the output acts monotonically within that range. These methods are intended to improve the effectiveness and precision of iterative interval refining in neural network security property verification, particularly in the case of complex input intervals.

Pseudo Code of Influence analysis: choose the most influential feature to split

1. Inputs:
    a. network: tested a neural network
    b. input: input interval
    c. g: gradient interval calculated by backward propagation
2. Logic:

```
for each i=1 to input.length {
  r = w(input[i]) → range of each input interval
  e = gup[i] * r → influence from each input to output
  if e > largest then → Check if e is greater than the largest influence
    largest = e → Update the largest influence
    splitFeature = i → Update the splitFeature
return splitFeature → Return the most influential feature to split
```

3. Understanding:

The weight parameters, which are independent of the input, fully determine the Jacobian matrix. When input is positive, a ReLU node's gradient can be 1, and when it is negative, it can be 0. In backward propagation, as seen in Algorithm 2, we employ intervals to track and propagate the constraints on the gradients of the ReLU nodes.
To compute the smear function for an input feature, we further employ the estimated gradient interval [26, 27]. The formula $Si(X) = \max 1 \leq j \leq d |Jij| w(Xj)$ indicates that Jij represents the gradient of input X j regarding output Yi. Algorithm 3 illustrates how we lower the over-approximation error for each refining stage by bisecting the Xj with the highest smear value.

The paper provides an evaluation of the ReluVal technique using two general categories of deep neural networks (DNNs), each designed for specific tasks:

Airborne Collision Avoidance System (ACAS) Xu Models:

- Task: Collision avoidance in unmanned aircraft.
- Network Structure: Input layer with five inputs, an output layer with five outputs, and six hidden layers with fifty neurons each.
- Input Parameters: {ρ, θ, ψ, vown, vint}, representing distance, heading angles, and speeds.
- Output Parameters: {COC, weak left, weak right, strong left, strong right}.
- Properties: Safety properties for ACAS models, including conditions for collision avoidance actions.

MNIST Models:

- Task: Handwritten digit recognition from the MNIST dataset.
- Network Structure: 784 inputs, 10 outputs, and two hidden layers with 512 neurons each.
- Properties: Classification accuracy and robustness to adversarial attacks on MNIST images.

Comparison Tools/Techniques:

- Reluplex: A solver-based verification tool used for comparison, known for its application in verifying the safety properties of neural networks.
- Carlini-Wagner Attack (CW): A state-of-the-art gradient-based adversarial attack used for comparison in terms of adversarial input discovery.

Main Evaluation Results:

1. ACAS Xu Models:
    a. ReluVal consistently outperforms Reluplex in terms of verification time, achieving up to a 200× speedup.

    b. The comparison with the Carlini-Wagner attack shows that ReluVal consistently finds 50% more adversarial examples on average.
2. MNIST Models:
    a. Preliminary tests on an MNIST model demonstrate ReluVal's ability to prove certain images as safe under different L∞ norm bounds.
    b. For adversarial input discovery, ReluVal outperforms the Carlini-Wagner attack, consistently finding more adversarial examples.
3. Optimizations:
    a. Symbolic interval analysis, influence analysis, and monotonicity collectively enhance the efficiency of ReluVal in verifying the properties of neural networks.
    b. Symbolic interval analysis significantly reduces the deepest and average depth of the bisection tree.
    c. Influence analysis prioritizes influential inputs, reducing the average depth and running time.
    d. Monotonicity provides improvements in running time, particularly when the average depth is high.

Benchmarks Used: Common benchmarks such as MNIST and CIFAR-10 are not specifically included in the table. Rather, properties designated as Φ1 through Φ15 are listed. These attributes, which are probably unique to the networks or models under review, might stand in for particular traits or performance standards like robustness, safety, or functional correctness. "Security Properties from [25]" is stated as the source for some of these features, implying a connection to another study or paper. Additionally, the table splits the attributes into two categories, which can represent various sets or classifications of assessments. The 'Networks' column indicates how many networks are tested for each attribute.

Principal Assessment Findings: The time in seconds that each tool takes to evaluate the properties is shown in the columns labeled "Reluplex Time" and "ReluVal Time." The 'Speedup' column appears to be comparing ReluVal's performance to that of Reluplex, indicating the number of times faster ReluVal is. ReluVal outperforms Reluplex by almost 30 times for property Φ1, for instance. The times vary: some properties (example 4729x for Φ5) show a large speedup, while other qualities (other example: 1x for Φ2) show little to no speedup. With a few notable exceptions, the data show that ReluVal typically completes evaluations more quickly than Reluplex for the assets listed.

Overall, ReluVal demonstrates superior performance compared to Reluplex in terms of speed and adversarial input discovery, showcasing its effectiveness in verifying safety properties and handling adversarial scenarios for different neural network models.