Trustable Machine Learning Systems

Jay Morgan

30th March 2021

Hello, and thank you for joining me today. My name is Jay Morgan, I'm a doctoral candidate at swansea university, and today I will be talking about one research method to create trustable machine learning systems.

of ML...

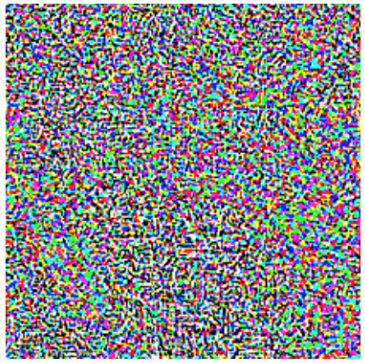
Or perhaps as I will put it today: the good, bad and ugly of Machine Learning

I hope that our future aspirations that, not just of our project, but with Machine Learning in general, that formal methods can become the norm and uncover the uglyness and transform it into all positives, all goods.

./images/self-driving-car.gif

Let us begin with the good.
Machine Learning, and specifically Deep Learning, has achieved a level of speed of computation, a level of accuracy, and perceived intelligence, that its actually becoming very useful in our daily lives.
Over the last few years we're seeing a transformation the automotive industry, where a big few car manufacturers are rushing to bring us unprecidented levels of autonomous driving.
We've already seen how useful big data and anomoly detection is for detecting frautulant bank transactions.
And finally how Machine Learning may revolutionise how chemistry and drug discovery is performed.

$+ .007 \times$



$=$

$\boldsymbol{x}$

"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Figure: `https://pytorch.org/tutorials/beginner/fgsm_tutorial.html`

But theres a twist. And the twist is this: Machine Learning and the 'intelligent' models they create are not intelligent.
Instead, it is well known with the community of Machine Learning Researchers, and beyond that these ML models are subject to some very critical flaws.
These flaws occur when specially crafted modifications are made to the input of the image. With these modifications are applied, the ML model will usually output a miss-classification with high confidence.
These types of attacks, called adversarial attacks, have been shown to be very effective at reducing the state of the classifiers to almost 0 zero accuracy.

"stop"
to "30m speed limit"

"80m speed limit"
to "30m speed limit"

"go right"
to "go straight"

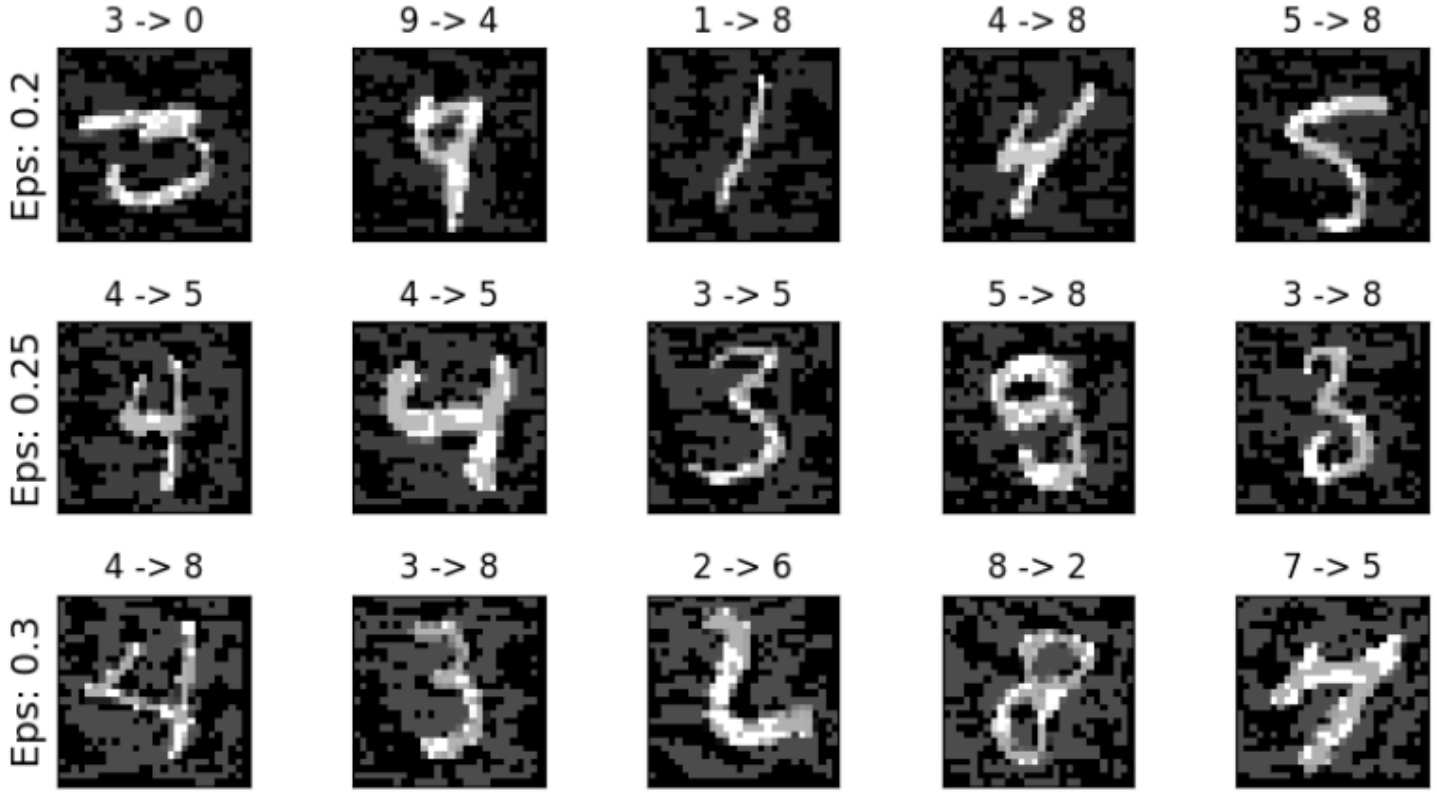
Figure: Huang, X., Kwiatkowska, M., Wang, S., & Wu, M. (2017, July). Safety verification of deep neural networks. In International conference on computer aided verification (pp. 3-29). Springer, Cham.

This presents a very serious problem for the use of ML in systems, like fully autonomous vehicles, where safety is paramount.
If such a modification is made to the images used by the ML models, even as a result of sensor defects, the result could cost human lives.

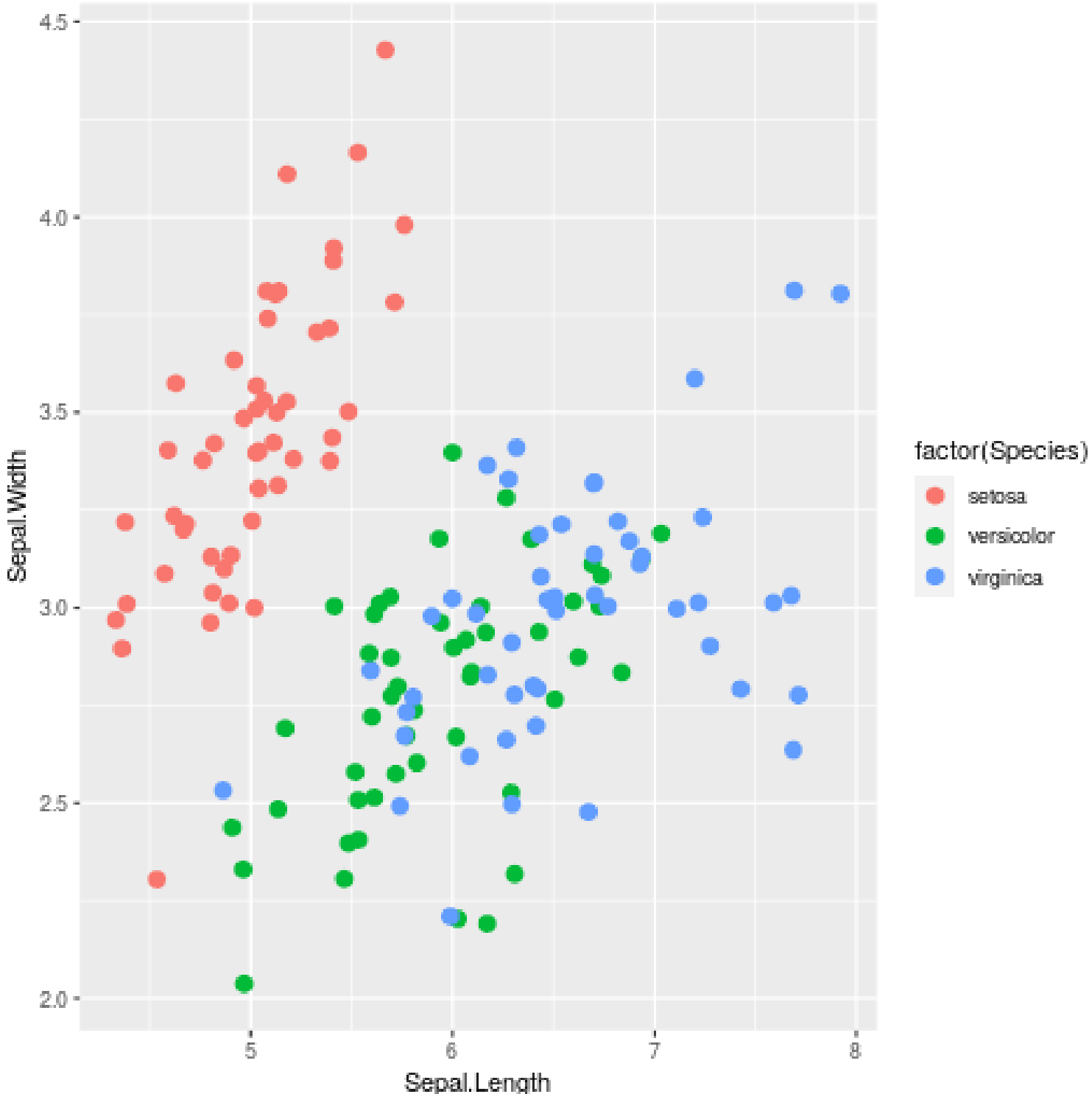
Given some classifier model $\mathcal{F}$ and some input $\mathbf{x}$, and adversarial is created by the modification ϵ within the range of r (i.e. $\epsilon \leq r$) that will result in a miss-classification: $\mathcal{F}(\mathbf{x}) \neq \mathcal{F}(\mathbf{x} + \epsilon)$.

Here we have a more formal definition of what an adversarial is. If we have some classifier F, and an input x, the adversarail example will be some modification ϵ to this x where the result will be a different output from the classifier. Typically, this ϵ value will be bounded by some norm value. In this example we have an r. I.e. this maximum amount of change to pixels will be bounded by this r.

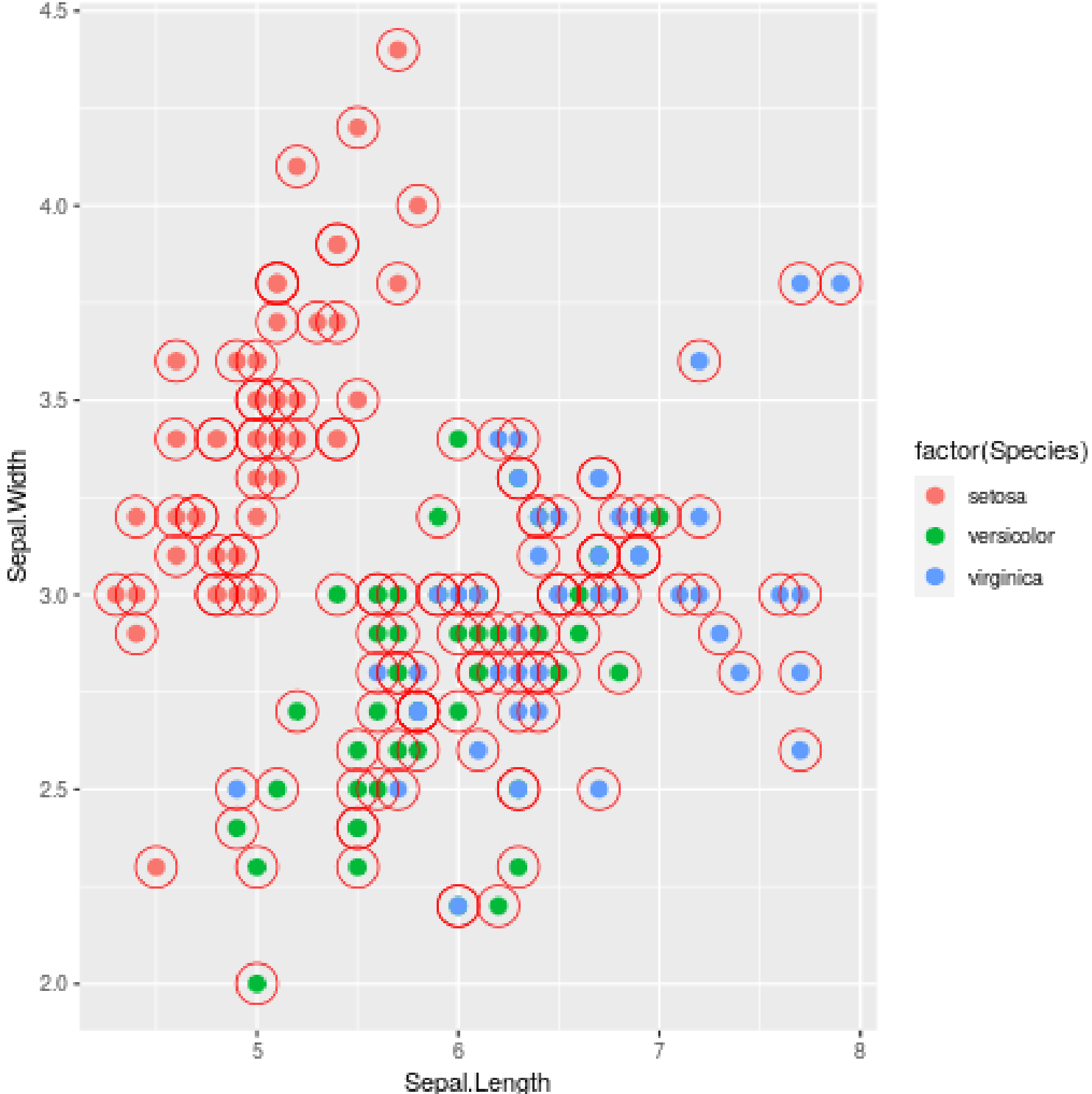
In other words, to create an adversarial, it is necessary to find some, suitably small, modification to the original input image, i.e. change of pixels, that will result in the model outputing an incorrect class. Often, we find that the modifications are not noticable to the human observer, but yet, the model has a high degree of confidence in its prediction.



And by modifying the amount of perturbation one can apply to the input, we have more destructive modifications. But as you can see in these examples, each image still looks like their respective number, despite any perturbation is applied.


4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
factor(Species)
setosa
versicolor
virginica

However, for non-image data, how much modification can we apply in order to search for adversarial examples?
In this toy example we have the Iris dataset. Called because we have 3 different types of plant species indicated by the different coloured points. In this plot we have plotted the Sepal Width against the Sepal Length. While the Setosa class may be almost linearly separated, the Versicolor and virginica classes are interdispersed using these two Sepal features.


4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
factor(Species)
setosa
versicolor
virginica

In this plot we have added a red outline to each point in the training data. This red outline represents the maximum amount of perturbation, our r bound we talked about before.
Even in this case where r is roughly 0.1, many modifications of the each point would push across potential class boundaries. This may be certainly true for the versicolor and viriginica classes. Yet for the Setosa class, we can be more sure that we have not passed any class boundaries.
So we may once again ask the question: which r-bound should we use when search for adversarial examples.

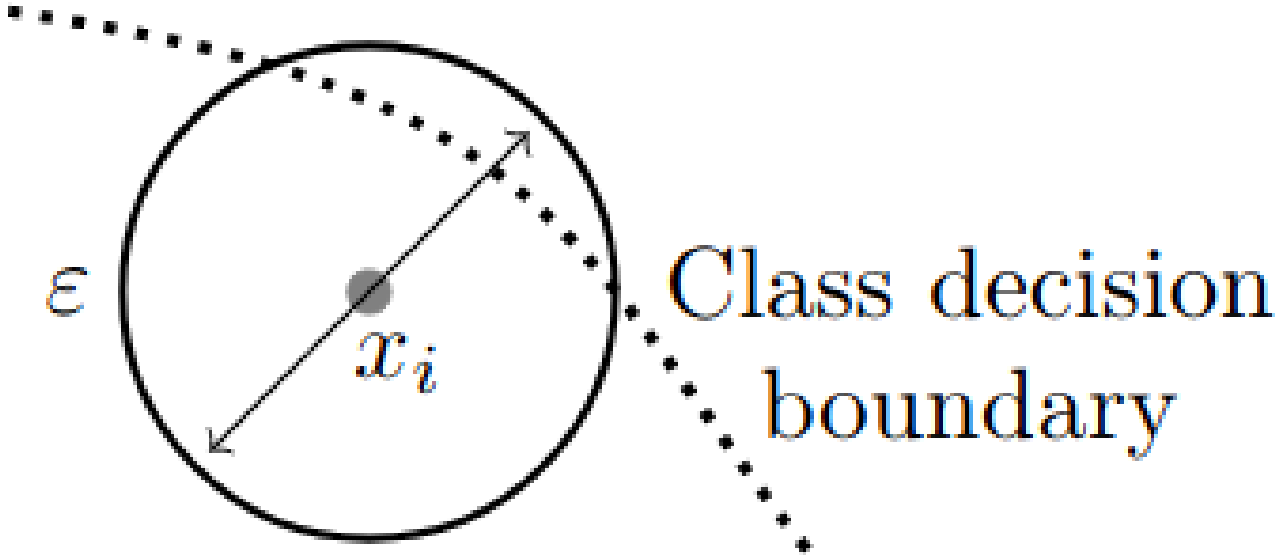
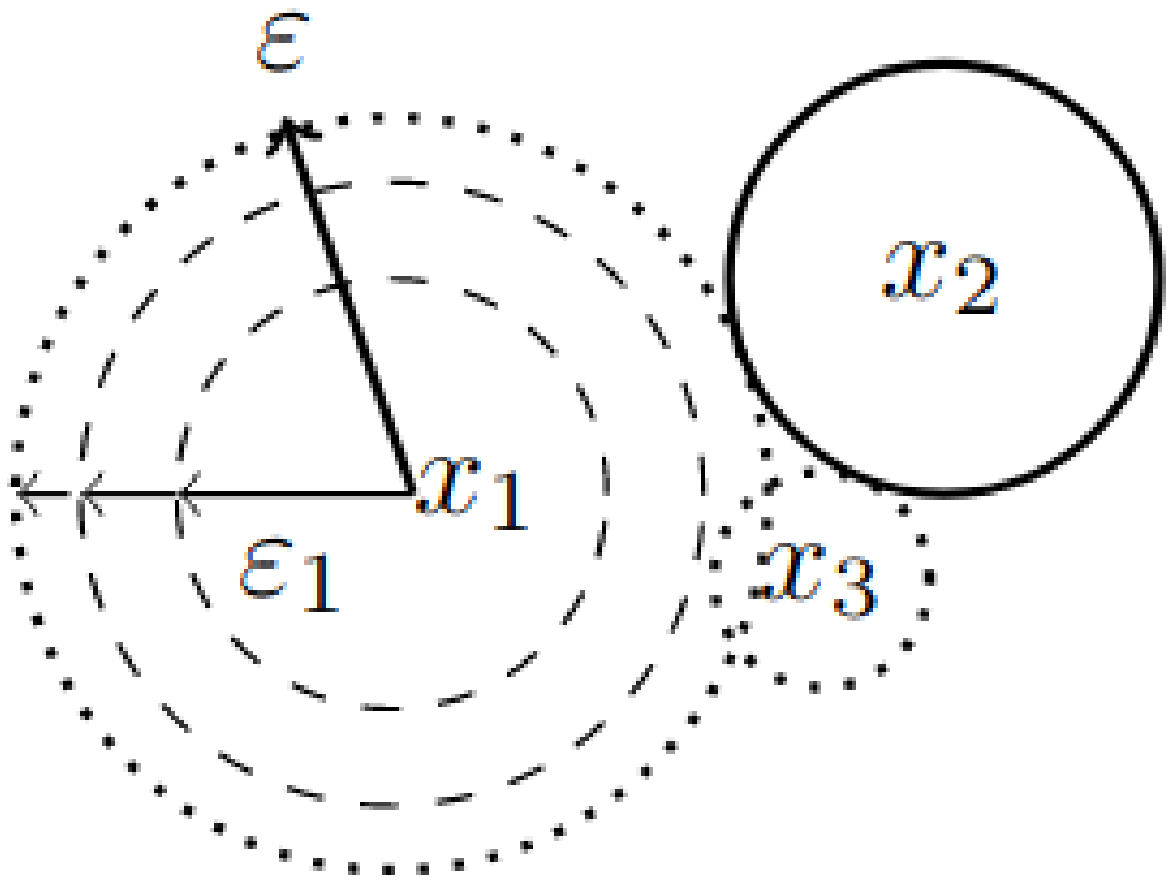


Figure: geometric complexity of class boundaries

Figure: sparsity/density of sampling from data manifold that consistutes the

training data.

Some of my research aims to answer this question, using the information presented in the available data. Given a set of data, a unique r-bound will be computed for each data point that will take into consideration the estimated class bounds, and how much information there is present in the data.
We consider two properties of the data in the process of generating these neighbourhoods. These are:
situations where differently labelled data points lay close together in the topological space, and therefore any perturbation of the data points could result in passing the class boundaries, while wrongly labelling the perturbation the same as the original. We have just seen this with the previous plots of the Iris data.
Our second property is shown in figure 2. It concerns the number of samples from different regions of the data manifold. In sparse regions (small numbers of samples), estimated class boundaries mayseem deceivingly simple, e.g. linear with a wide margin.



In our method we provide an algorithm to iteratively expand the maximum r bound.

Our method consists of iteratively expanding the maximum r bounds for each data point simulateanously. If the bound intersects with a bound of data point from a different class, then these two data points will stop expanding.
In this example we can see 3 data points, x_1, x_2, and x_3. Where x_1, and x_3, are from one class, and x_2 is from another. There are iterative steps for the expansion of x_1 until it collides with the bound of x_2. The bound of x_3 is ignored as its from the same class.

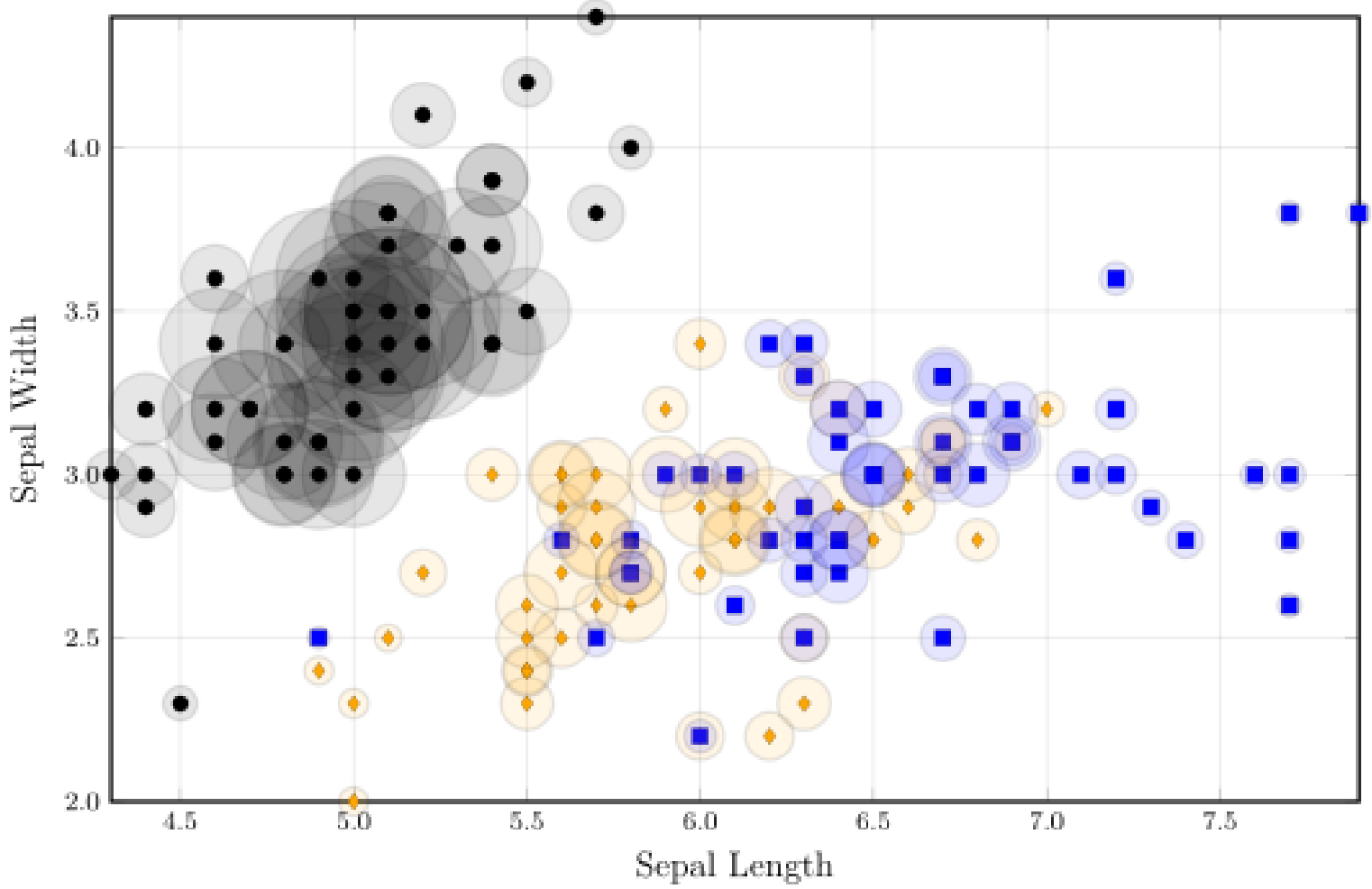
Expansion is modulated by the density data point. Using an inverse multiquadric radial basis function (RBF) to measure the density of each point.

$$\varphi(x;\overline{x}) = \frac{1}{\sqrt{1+(\varepsilon r)^2}}, \text{ where } r = \parallel \overline{x} - x \parallel$$

The density for a single point is the sum of RBFs centered at each point.

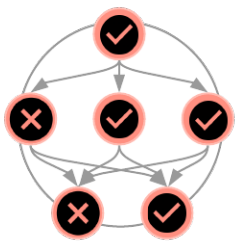
$$\rho_c(x) = \sum_{x_j \in X^c} \varphi(x;x_j)$$

As we noted before, we may not have a lot of information in which to estimate the class boundaries. This lack of information occurs due to the lack of sampling of data. Therefore, we use this information of density of sampling to account for the lack of information of class boundaries.
This density modulates the expansion of the r-bound. If there is not lots of information about class boundaries, then the r-bound expansion will be a lot smaller.



Morgan, J., Paiement, A., Pauly, A., & Seisenberger, M. (2021). Adaptive neighbourhoods for the discovery of adversarial examples. arXiv preprint arXiv:2101.09108.

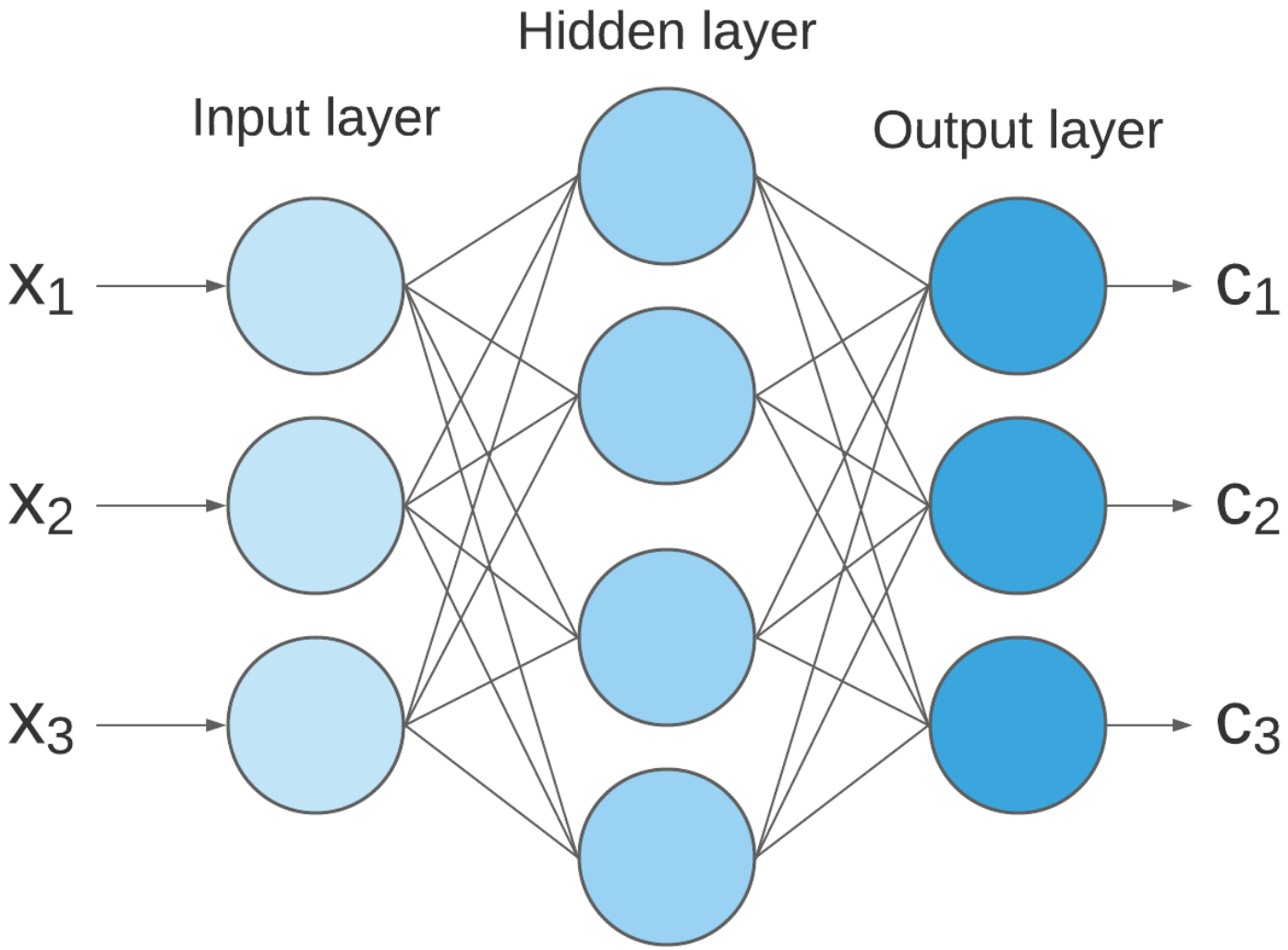
After computing the density of each data point and expanding the neighbourhoods, then we will have an unique r-bound for each data point. This *r* provides the upper-bound with which to search for adversarial examples.
Here today, I have provided the iterative method to compute r-bounds, but we also provide another method using langrangian multipliers to directly compute these bounds. You can find the method in the paper "Adaptive neighbourhoods for the discovery of adversarial examples".



NeuralVerifier

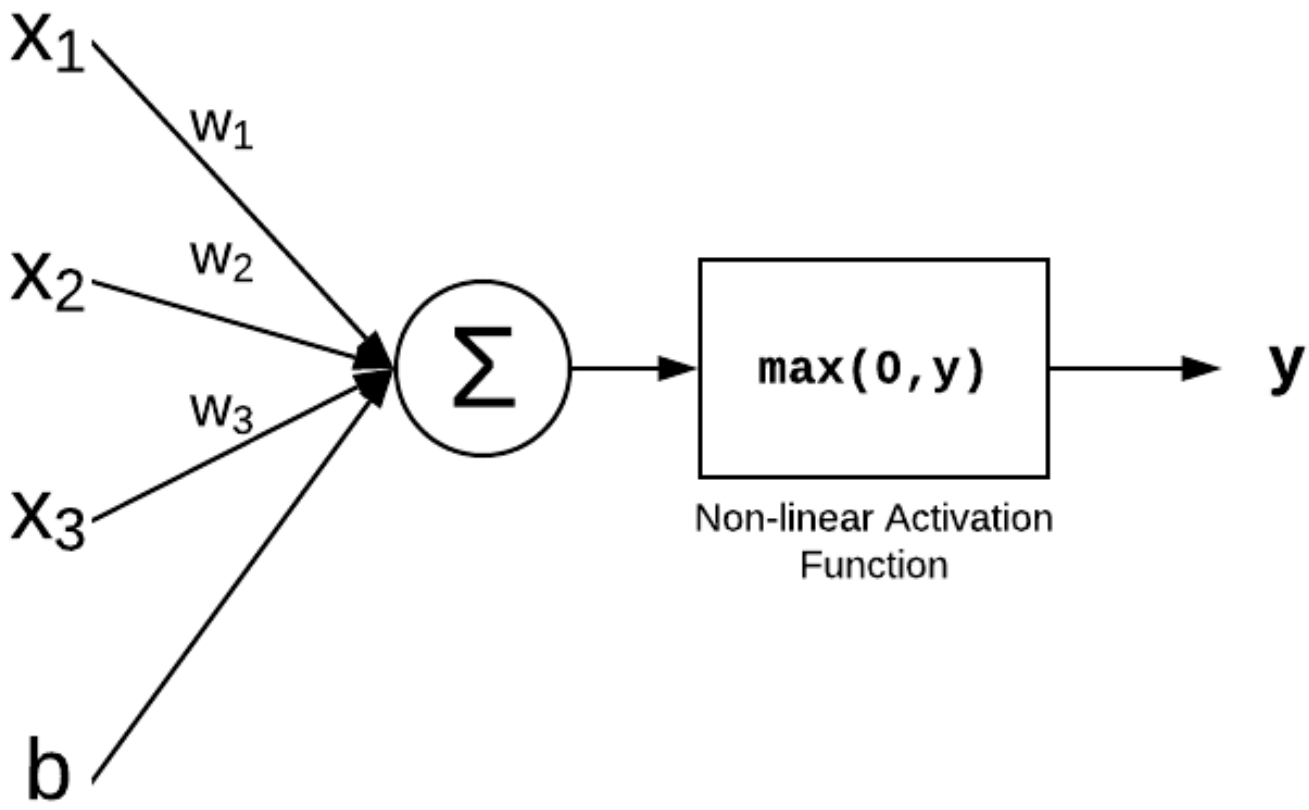
`https://github.com/jaypmorgan/NeuralVerifier.jl` - built on top of Z3 solver to provide an interface to verify Nerual Network properties, such as: output bounds checking and adversarial robustness.

Now that we have our upper-bound for our search, we must now find instances of adversarial examples.



Take a very simple example of a 3-layer neural network.

$$z = \sigma(Wx + b)$$

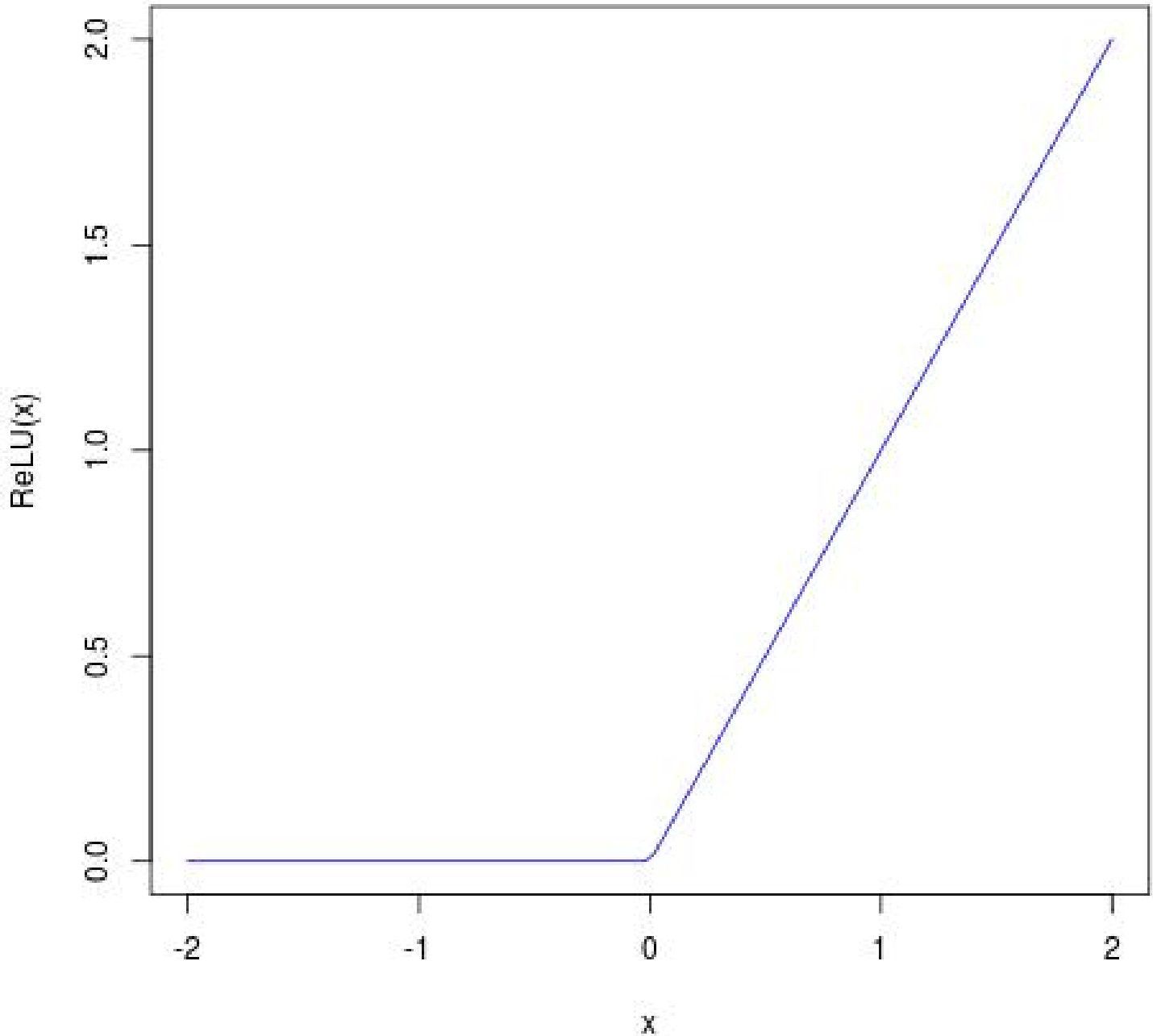


Where σ is some non-linear function to increase the model's complexity to allow it to model non-linear relationships. One of the most common

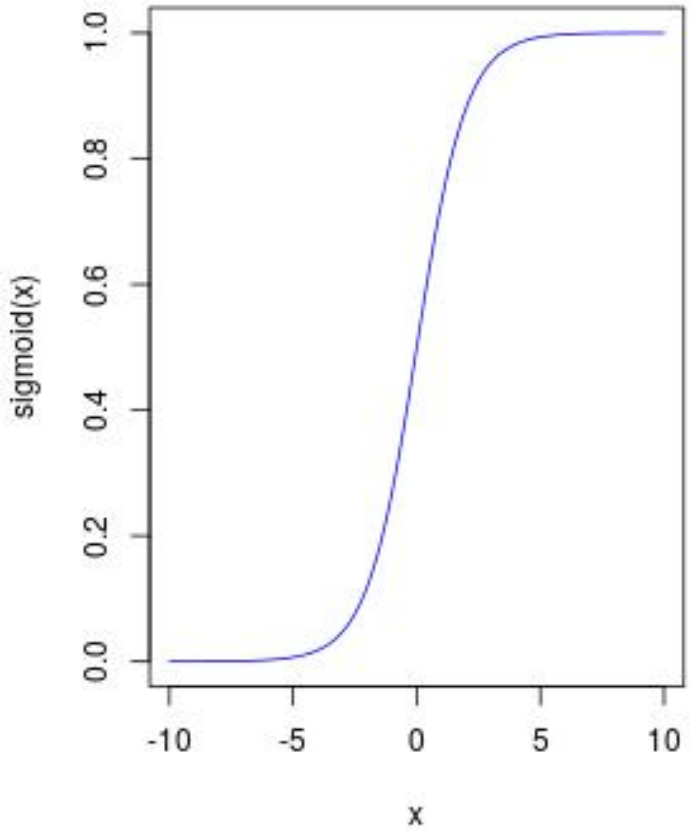
non-linear functions when training neural networks is the Rectified Linear activation function (ReLU): $\max(Wx + b, 0)$.
Z3 provides support for real linear arithmetic and provides operations for the basic multiplication and addition. Thus, we need only to apply these elementwise.

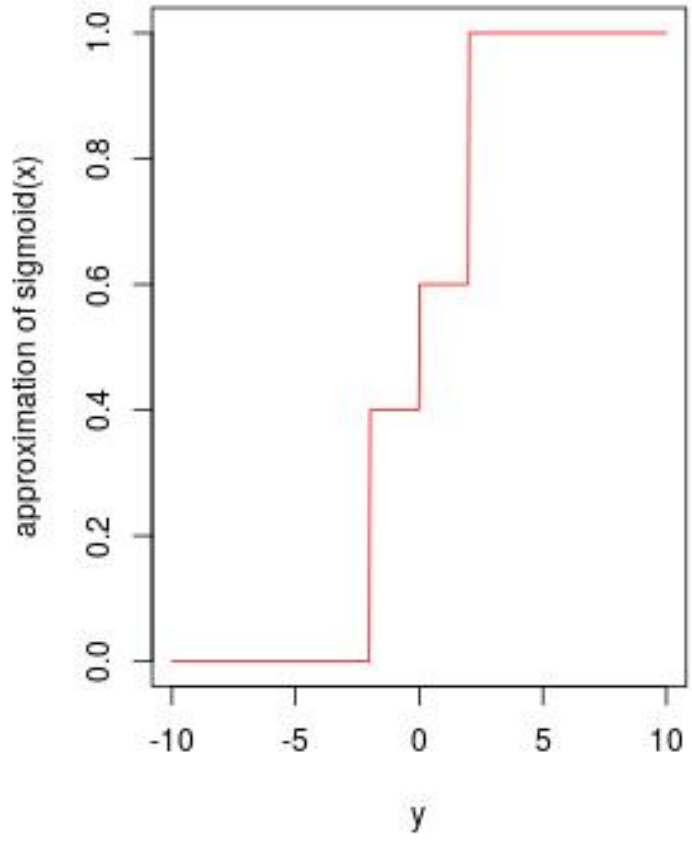
```
function dense(x, W, b)
    out = fill!(Array(undef, size(W,1), size(x,2)), 0)

    for i = 1:size(out,1), j = 1:size(W,2)
        out[i] += W[i,j] * x[j]
    end

    out = out .+ b
    return out
end
```

Moving onto non-linear functions, we must consider how such non-linearities are encoded in the model. For some of the activation functions, it could be as easy as simple boolean logic.
`If(x > 0, x, 0)`


ReLU(x)
0.0
0.5
1.0
1.5
2.0
-2
-1
0
1
2
x

However, encoding more complex activation functions can be reduced via piecewise linear approximation with the same boolean arithmetic. Increasing the precision of approximation will make satisfiability slower, but the encoding will be more true with the original network.






```
function sigmoid(x)
    If(x < 0,
        If(x < -2, 0.0, 0.4),
        If(x >  2, 1.0, 0.6))
end

function dense(x, W, b)
```

```
    out = fill!(Array(undef, size(W,1), size(x,2)), 0)

    for i = 1:size(out,1), j = 1:size(W,2)
        out[i] += W[i,j] * x[j]
    end

    out = out .+ b
    return out
end

function relu(x)
    If(x > 0, x, 0)
end

y = relu(dense(x, W, b))

encoding(x) = begin
    y = dense(x,
              neural_network[1].W,
              neural_network[1].b) |> relu;
    y = dense(y,
              neural_network[1].W,
              neural_network[1].b) |> relu;
```

```
    y = dense(y,
              neural_network[2].W,
              neural_network[2].b) |> softmax;
end
```

$$\min_{\epsilon}(\mathcal{F}(x) \neq \mathcal{F}(x+\epsilon)) \wedge \epsilon \leq r$$

```
adv_examples = []

for (idx, (x_i, r_i)) in enumerate(zip(x, r))
    m = Optimize()  # create an optimisation procedure (model)

    add!(m, (eps > 0) && (eps <= r_i))

    y = encoding(x_i)  # get initial condition of y given our en

    add!(m, y != f(xi)) # add the adversarial example condition

    minimize!(m, eps)  # find the smallest eps

    check(m) # check for satisfiability

    if m.is_sat == "sat"
```

```
            push!(adv_examples, [variable(m, var) for var in vars])
            @info "Input number: $(idx), Adversarial found!"
        end
    end
```

```
[ Info: Input number: 1, Adversarial found!
[ Info: Input number: 3, Adversarial found!
[ Info: Input number: 4, Adversarial found!
[ Info: Input number: 5, Adversarial found!
[ Info: Input number: 7, Adversarial found!
...
```

You can find these slides on my personal website below. Additionally follow the github link for more documentation and usage on NeuralVerifier.jl

- `https://blog.morganwastaken.com`
- `https://github.com/jaypmorgan/NeuralVerifier.jl`