Adversarial examples in deep learning

Grégory Châtel

DISAITEK

Intel Software Innovator

@rodgzilla
github.com/rodgzilla

28/02/2018

Machine learning is a subfield of artificial intelligence.

Intuitively We want to *learn from* and *make predictions on* data.

Technically We want to build a model that approximate well (*e.g.* minimize a loss function) an unknown function.

It is important to note that the function we want to approximate may or may not have a closed form.

- Regression

Polynomial	$(x, y, z) \rightarrow f(x, y, z)$
House price	(surface, nb rooms, city) $\rightarrow$ price

- Classification

Image classification	pixel values $\rightarrow$ cat or dog
Text classification	list of words $\rightarrow$ spam or valid email

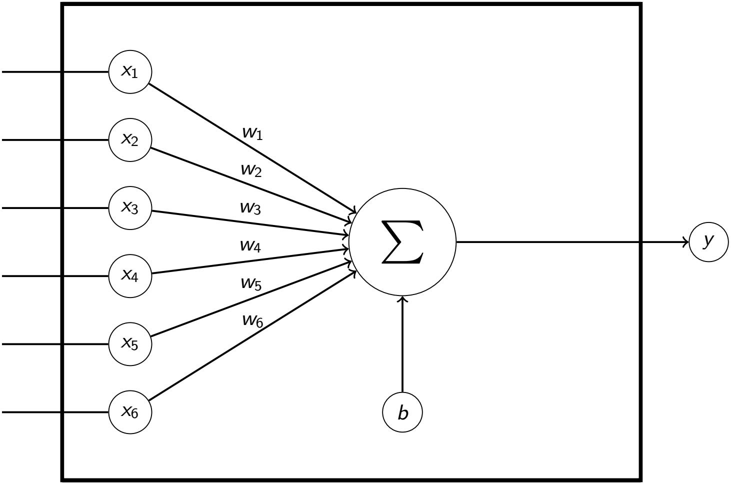
Deep learning

Deep learning is a subfield of machine learning in which we use artificial neural networks to make predictions.

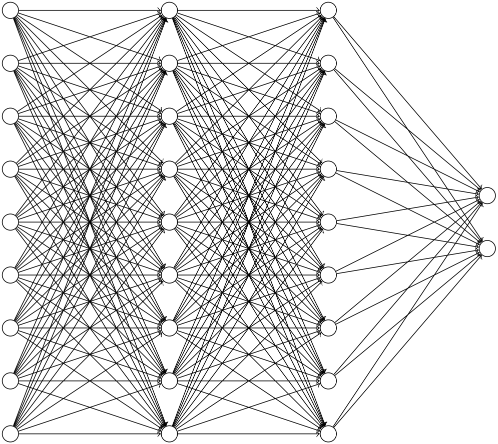
An artificial neural networks is a computation model loosely based on the human brain. It aims to mimic electric signals travelling through neurons in order to make computations.

Artificial neural network

Neuron

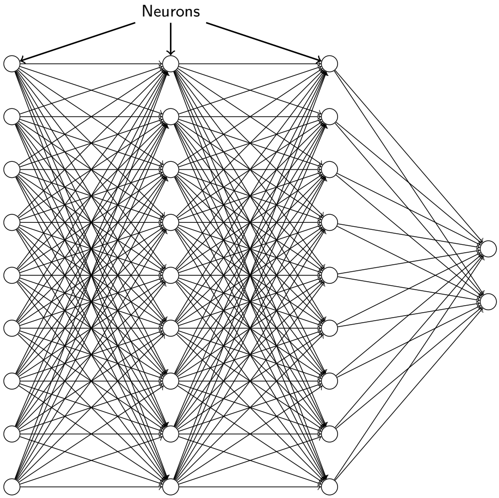


$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + b$$



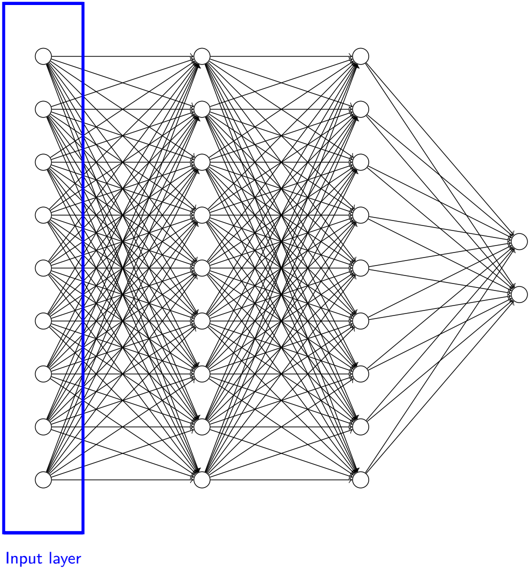
Artificial neural network

Network



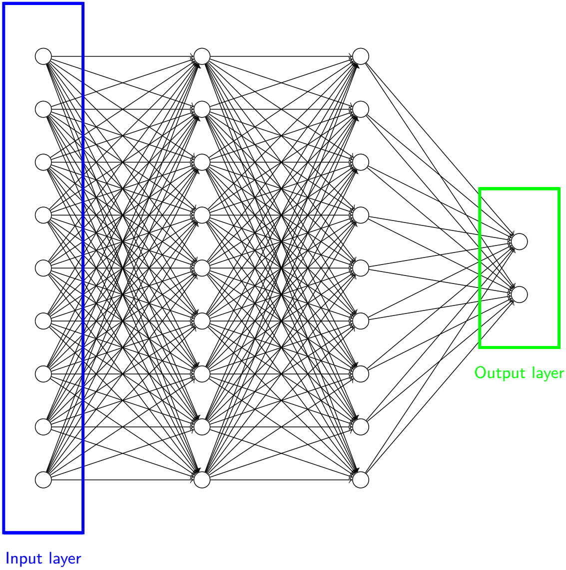
Artificial neural network

Network



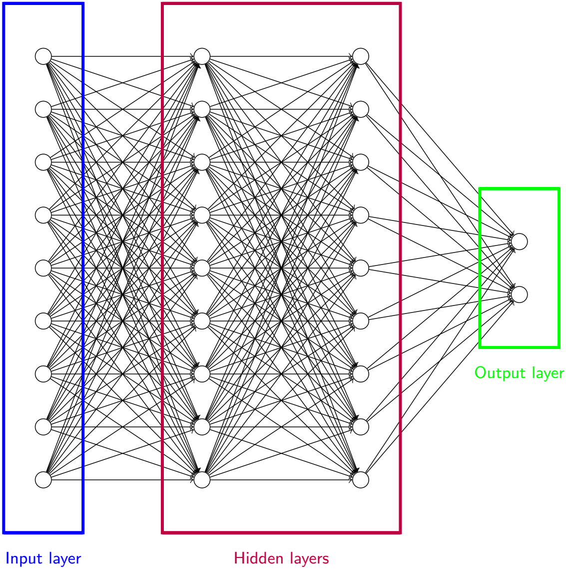
Artificial neural network

Network



Artificial neural network

Network



Problem with this definition

We now have a quite complicated framework to compute linear functions.

Neuron	linear function
Neural network	linear combination of neuron outputs

To approximate non linear functions, we would like to have a non linear model.

Problem with this definition

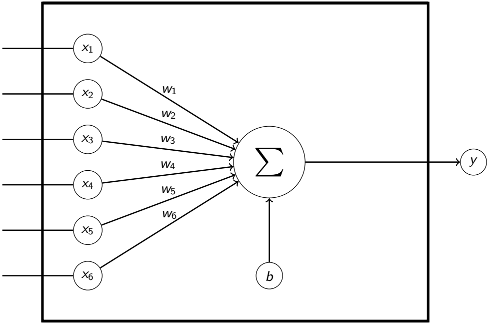
We now have a quite complicated framework to compute linear functions.

Neuron	linear function
Neural network	linear combination of neuron outputs

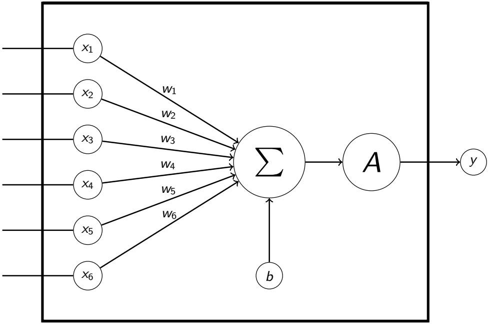
To approximate non linear functions, we would like to have a non linear model.

Solution: add nonlinearity to neurons.

Neuron with activation

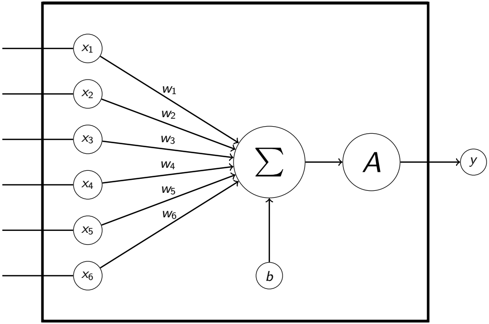


Neuron with activation



$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

Neuron with activation

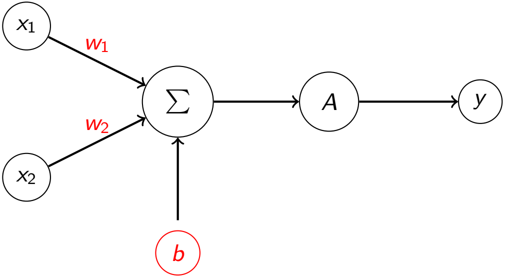


$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$y = A(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + b)$$

Computation example

Binary AND gate

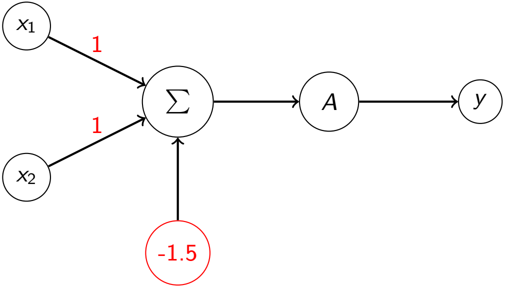


We want to set w_1, w_2 and b such that:

$$A(w_1 x_1 + w_2 x_2 + b) = x_1 \text{ AND } x_2$$

Computation example

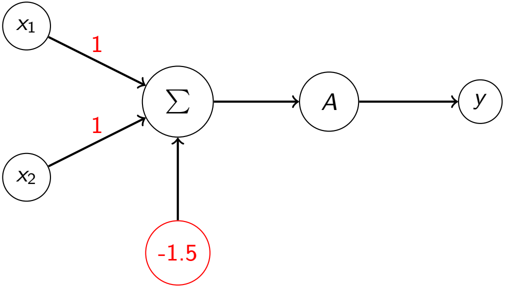
Binary AND gate



We want to set w_1, w_2 and b such that:

$$A(w_1 x_1 + w_2 x_2 + b) = x_1 \text{ AND } x_2$$

Computation example
Binary AND gate



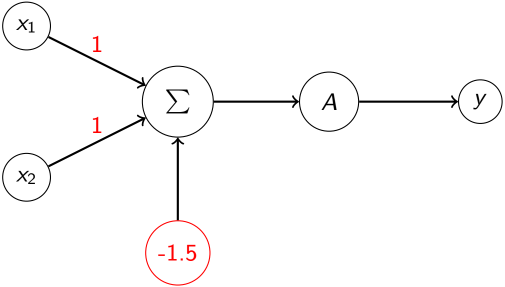
We want to set w_1, w_2 and b such that:

$$A(w_1x_1 + w_2x_2 + b) = x_1 \text{ AND } x_2$$

$$x_0 = 0, x_1 = 1. \quad y = A(0 + 1 - 1.5) = A(-0.5) = 0$$

Computation example

Binary AND gate



We want to set w_1, w_2 and b such that:

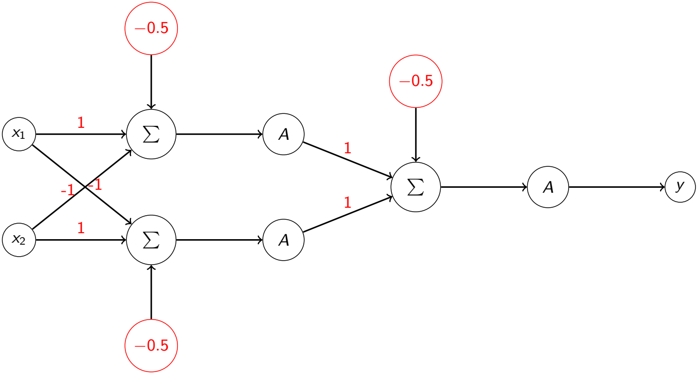
$$A(w_1 x_1 + w_2 x_2 + b) = x_1 \text{ AND } x_2$$

$$x_0 = 0, x_1 = 1. \quad y = A(0 + 1 - 1.5) = A(-0.5) = 0$$

$$x_0 = 1, x_1 = 1. \quad y = A(1 + 1 - 1.5) = A(0.5) = 1$$

Computation example

Binary XOR gate



Model complexity

One way to measure the complexity of a neural network is its number of parameters.

Model complexity

One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters

Model complexity

One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters
- 1000 classes image classification (VGG16 network): 138,357,544 parameters

Model complexity

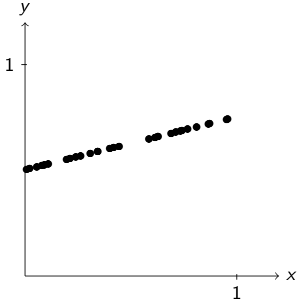
One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters
- 1000 classes image classification (VGG16 network): 138,357,544 parameters

We should probably look for a way to tune these parameters automatically otherwise it is going to get *really* boring *really* quickly.

Parameter tuning
Line approximation

We have a few sample points and we want to find a line that approximate these points.



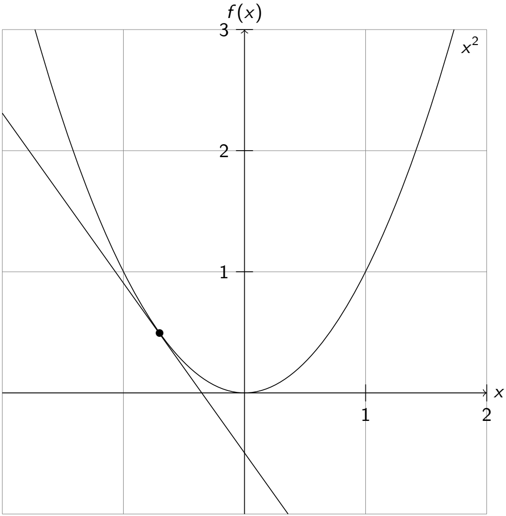
Our model is just a line

$$y_{pred} = {\color{red}a}x + {\color{red}b}$$

We want to find *a* and *b* to best match our samples.

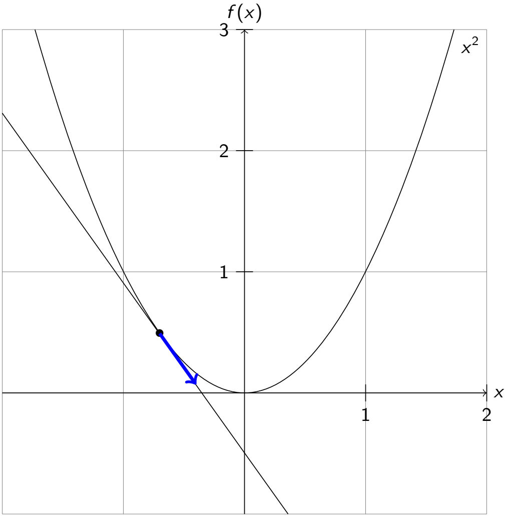
Parameter tuning

Gradient descent concept



Parameter tuning

Gradient descent concept



Optimization

Parameter tuning

Loss definition

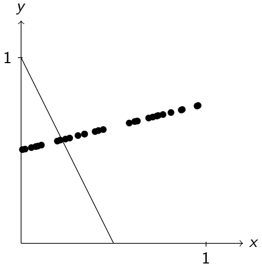
First we have a define a loss that measures how good our predictions are.

$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

and now, we compute how the loss is affected by small changes of a and b:

$$\frac{dl}{da} = 2x(ax + b - y) \qquad \frac{dl}{db} = 2(ax + b - y)$$

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

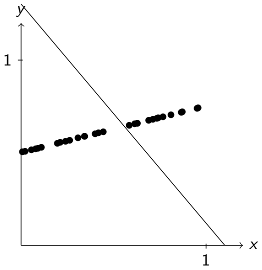
$$a = -2.00 \quad b = 1.00 \quad \overline{\frac{dl}{da}} = -1.07 \quad \overline{\frac{dl}{db}} = -1.37 \quad \overline{l(x, y, a, b)} = 0.860367$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

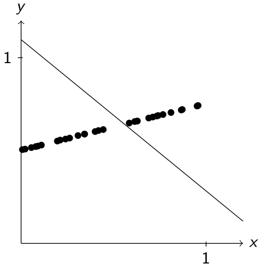
Then, we compute the average of the gradient along all (x, y) couples

$$a = -1.18 \quad b = 1.30 \quad \overline{\frac{dl}{da}} = -0.17 \quad \overline{\frac{dl}{db}} = 0.09 \quad \overline{l(x, y, a, b)} = 0.159420$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning
Gradient descent application

We start with random values: $a = -2$ and $b = 1$



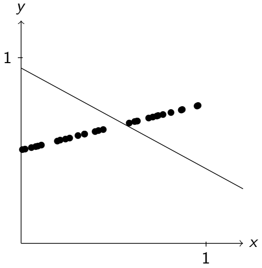
$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = -0.82 \quad b = 1.10 \quad \overline{\frac{dl}{da}} = -0.13 \quad \overline{\frac{dl}{db}} = 0.07 \quad \overline{l(x, y, a, b)} = 0.088204$$

And update a and b by substracting a small proportion of their gradient.

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

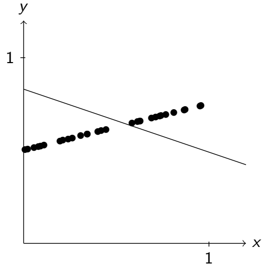
$$a = -0.54 \quad b = 0.94 \quad \overline{\frac{dl}{da}} = -0.09 \quad \overline{\frac{dl}{db}} = 0.05 \quad \overline{l(x, y, a, b)} = 0.048802$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

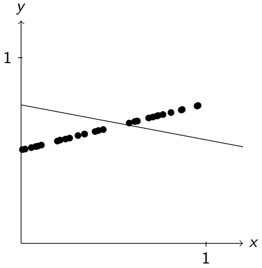
Then, we compute the average of the gradient along all (x, y) couples

$$a = -0.34 \quad b = 0.83 \quad \overline{\frac{dl}{da}} = -0.07 \quad \overline{\frac{dl}{db}} = 0.04 \quad \overline{l(x, y, a, b)} = 0.027001$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning
Gradient descent application

We start with random values: $a = -2$ and $b = 1$



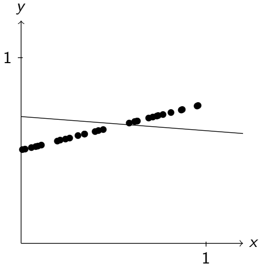
$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = -0.19 \quad b = 0.75 \quad \overline{\frac{dl}{da}} = -0.05 \quad \overline{\frac{dl}{db}} = 0.03 \quad \overline{l(x, y, a, b)} = 0.014939$$

And update a and b by substracting a small proportion of their gradient.

We start with random values: $a = -2$ and $b = 1$



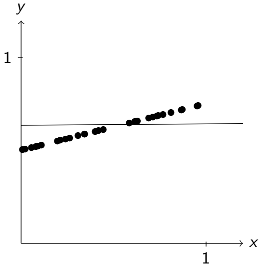
$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = -0.08 \quad b = 0.68 \quad \overline{\frac{dl}{da}} = -0.04 \quad \overline{\frac{dl}{db}} = 0.02 \quad \overline{l(x, y, a, b)} = 0.008266$$

And update a and b by substracting a small proportion of their gradient.

We start with random values: $a = -2$ and $b = 1$



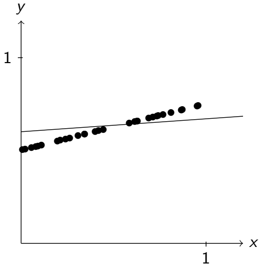
$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = 0.01 \quad b = 0.64 \quad \overline{\frac{dl}{da}} = -0.03 \quad \overline{\frac{dl}{db}} = 0.02 \quad \overline{l(x, y, a, b)} = 0.004573$$

And update a and b by substracting a small proportion of their gradient.

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

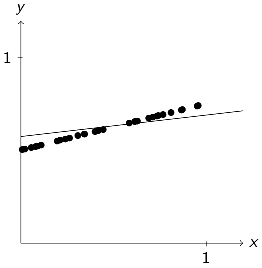
$$a = 0.07 \quad b = 0.60 \quad \overline{\frac{dl}{da}} = -0.02 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.002530$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

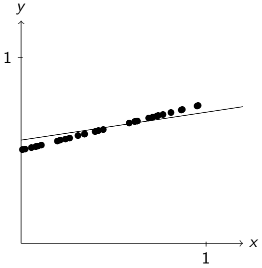
$$a = 0.12 \quad b = 0.58 \quad \overline{\frac{dl}{da}} = -0.02 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.001400$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

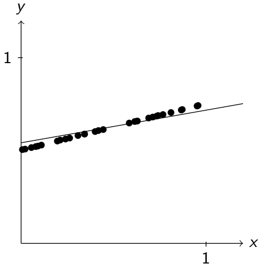
$$a = 0.15 \quad b = 0.56 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.000775$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

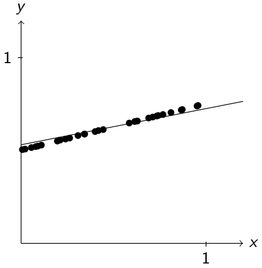
$$a = 0.18 \quad b = 0.54 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000429$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

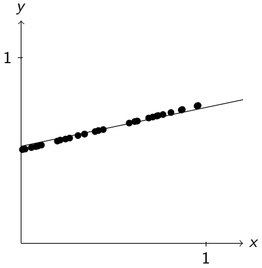
$$a = 0.19 \quad b = 0.53 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000237$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

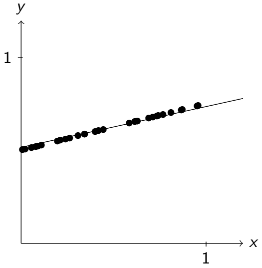
$$a = 0.21 \quad b = 0.52 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000131$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

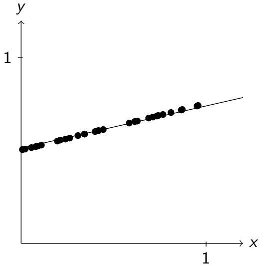
$$a = 0.22 \quad b = 0.52 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000073$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



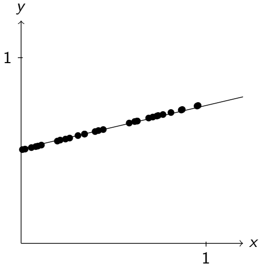
$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = 0.23 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000040$$

And update a and b by substracting a small proportion of their gradient.

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

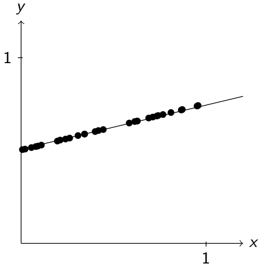
Then, we compute the average of the gradient along all (x, y) couples

$$a = 0.23 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000022$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning
Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

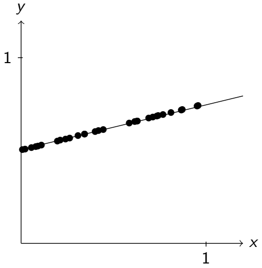
$$a = 0.24 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000012$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

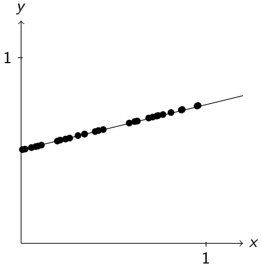
$$a = 0.24 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000007$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Gradient descent application

We start with random values: $a = -2$ and $b = 1$



$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

Then, we compute the average of the gradient along all (x, y) couples

$$a = 0.24 \quad b = 0.50 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000004$$

And update a and b by substracting a small proportion of their gradient.

Parameter tuning

Problem with ANN

Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

Problem: The function computed by our neural network is not differentiable because of the activation function.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

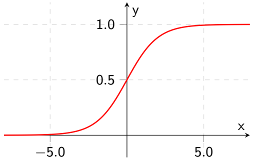
Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

Problem: The function computed by our neural network is not differentiable because of the activation function.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

Solution: We replace it by a differentiable function that does the same job.

$$A(x) = \frac{1}{1+e^{-x}}$$

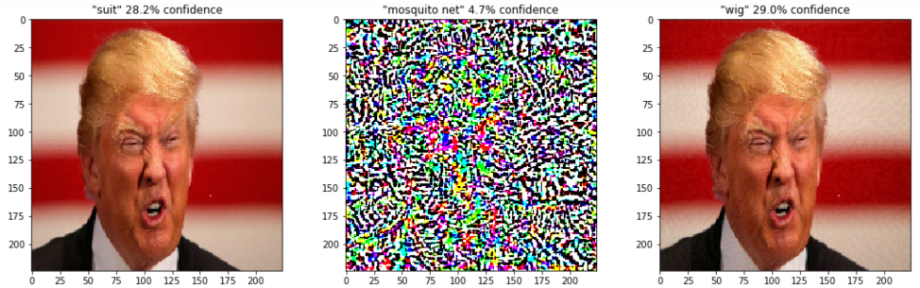


What is an adversarial example?

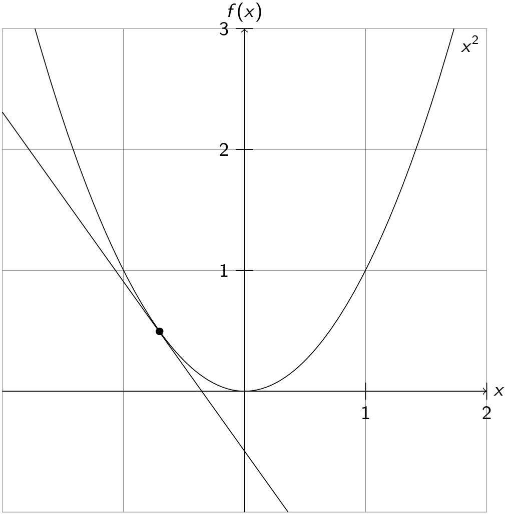
An *adversarial example* is a sample of input data which has been modified *very slightly* in a way that is intended to cause a machine learning classifier to misclassify it.

What is an adversarial example?

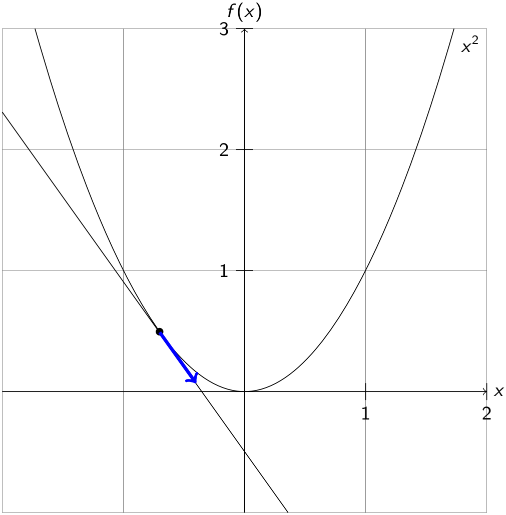
An *adversarial example* is a sample of input data which has been modified *very slightly* in a way that is intended to cause a machine learning classifier to misclassify it.



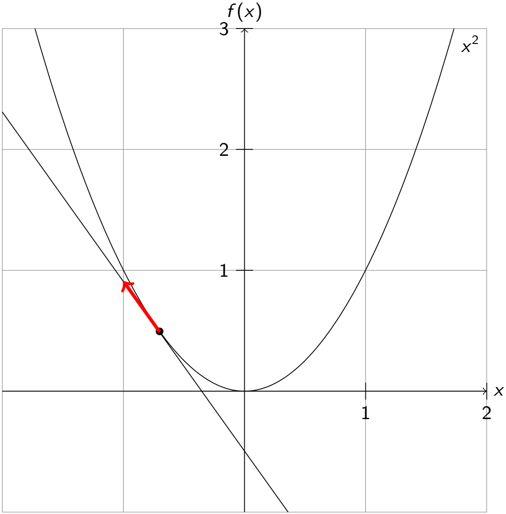
Gradient descent



Gradient descent



Gradient descent



De-optimization

Gradient descent

Being evil

In our previous example, we have modified the model in order to minimize the loss.

$$y = ax + b$$

In our previous example, we have modified the model in order to minimize the loss.

$$y = ax + b$$

Now suppose we are an attacker who wants to maximize the loss of a model, its parameters being fixed. The only thing we can modify are the inputs.

$$L(x, y, a, b) = (y - (ax + b))^2$$

In our previous example, we have modified the model in order to minimize the loss.

$$y = ax + b$$

Now suppose we are an attacker who wants to maximize the loss of a model, its parameters being fixed. The only thing we can modify are the inputs.

$$L(x, y, a, b) = (y - (ax + b))^2$$

In order to do this, we compute how the loss is affected by small changes of the input.

$$\frac{\mathrm{d}L}{\mathrm{d}x} = 2a(ax + b - y)$$

We can now make *imperceptible* changes to an input that will increase the loss value.

Neural networks

Everything works the same way when working with a neural network on an image classification task.

We also have a differentiable *loss function* (often *categorical cross entropy*), model *parameters* (the *weights* of the neural network) and *inputs* (*pixel values* in the case of images) that we can modify to increase the loss.

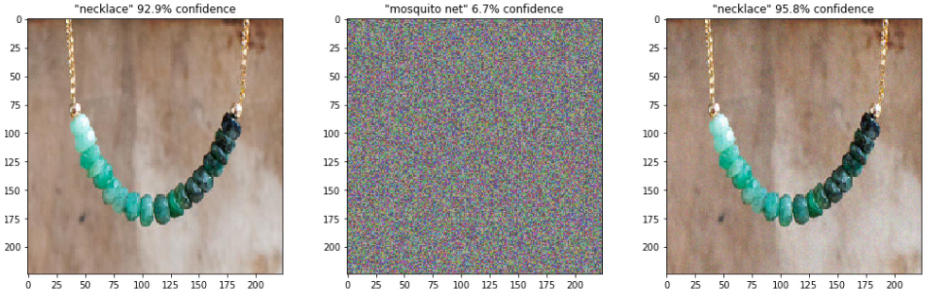
The ultimate goal is to make our input cross the *decision boundary*.

Random noise perturbation

Do we really need to do that?

Random noise perturbation

Yep, we really do



Fast Gradient Sign Method [Goodfellow et al. 2015]

Let x be the original image, θ the parameters of the model, y the target associated with x and $L(\theta, x, y)$ the loss function.

We compute the gradient of the loss function according to the input pixels.

$$\nabla_x L(\theta, x, y)$$

The perturbation is the signs of these derivatives multiplied by a small number ε.

$$\eta = \varepsilon \ \mathrm{sign}(\nabla_x L(\theta, x, y))$$

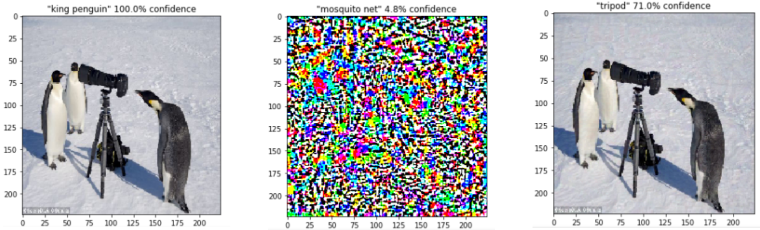
The final adversarial sample is the sum of the original image and the perturbation.

$$x_{adv} = x + \eta$$

Fast Gradient Sign Method

VGG16 network

$$x \quad + \quad \varepsilon \, \mathrm{sign}(\nabla_x L(\theta, x, y)) \quad = \quad x_{\text{adv}}$$



Black-box attack [Papernot et al. 2016]

or good luck getting gradients out of your self-driving car

Black-box attack [Papernot et al. 2016]

Transferability of adversarial samples

We can train a new model M' to solve the same classification task as the target model M.

Black-box attack [Papernot et al. 2016]

Transferability of adversarial samples

We can train a new model M' to solve the same classification task as the target model M.

Once trained, we can create an adversarial sample x'_{adv} for the M' model and experiences have shown that x'_{adv} will also fool M very often.

Black-box attack [Papernot et al. 2016]

Transferability of adversarial samples

We can train a new model M' to solve the same classification task as the target model M.

Once trained, we can create an adversarial sample x'_{adv} for the M' model and experiences have shown that x'_{adv} will also fool M very often.

What if we do not have a training set for the target network? Well... build one using M predictions.

Black-box attack [Papernot et al. 2016]

Transferability of adversarial samples

We can train a new model M' to solve the same classification task as the target model M.

Once trained, we can create an adversarial sample x'_{adv} for the M' model and experiences have shown that x'_{adv} will also fool M very often.

What if we do not have a training set for the target network? Well... build one using M predictions.

"After labeling 6,400 synthetic inputs to train our substitute (an order of magnitude smaller than the training set used by MetaMind) we find that their DNN misclassifies adversarial examples crafted with our substitute at a rate of 84.24%"

- Papernot et al., about their attack on the MetaMind deep neural network.

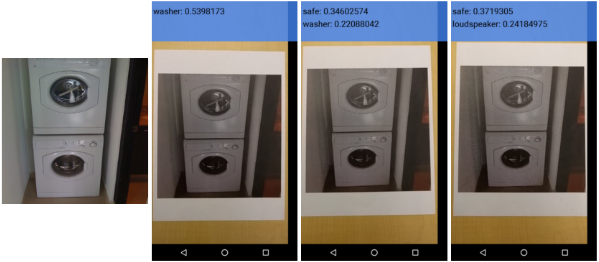
Adversarial examples in the physical world [Kurakin et al. 2017]

or good luck attacking a self-driving car with your USB flash drive

Adversarial examples in the physical world [Kurakin et al. 2017]

In real world scenarios, the target network does not take our image files as input. It acquires the data by the network's system (*e.g.* a camera).

It also works, for free.



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

"We used images taken from a cell-phone camera as a input to an Inception v3 image classification neural network. We showed that in such a set-up, a significant fraction of adversarial images crafted using the original network are misclassified even when fed to the classifier through the camera."

\- Kurakin et al.

White-box defense

Adversarial training

We can generate adversarial samples and train the network to produce the correct classification on these new data points.

- Works against FGSM
- Expensive
- Does not defend subtler white-box attacks (*e.g.* I-FGSM or RAND+FGSM)
- Does not defend against black-box attack

Black-box defense

Ensemble adversarial training [Tramèr et al. May 2017]

Augment training data with adversarial inputs from a number of fixed pre-trained models.

- Best defense so far against black-box adversary
- Does not defend subtle white-box attacks

Error rate from 15.5% to 3.9% between adversarial trained and ensemble adversarial trained model on MNIST for a black-box attack.

Defending machine learning

Wide open problem

"Most defenses against adversarial examples that have been proposed so far just do not work very well at all, but the ones that do work are not adaptive. This means it is like they are playing a game of whack-a-mole: they close some vulnerabilities, but leave others open."

- Ian Goodfellow, Nicolas Papernot, February 2017

References

DISAITEK

Research papers:

1. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
2. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., & Swami, A. (2016). Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697.
3. Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533.
4. Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., & McDaniel, P. (2017). Ensemble Adversarial Training: Attacks and Defenses. arXiv preprint arXiv:1705.07204.
5. Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). The Space of Transferable Adversarial Examples. arXiv preprint arXiv:1704.03453.

Implementations:

1. github.com/tensorflow/cleverhans
2. github.com/rodgzilla/machine_learning_adversarial_examples

Targeted perturbation

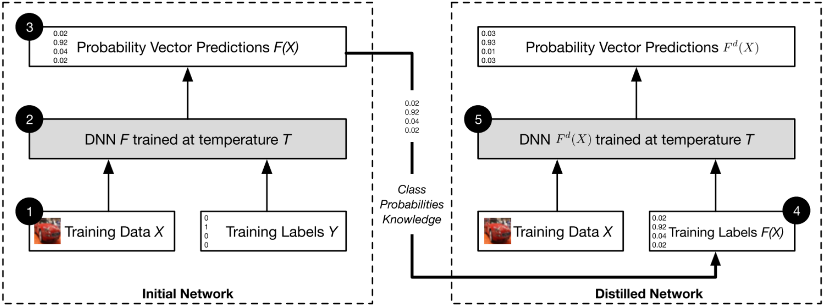
Papernot algorithm

This algorithm iteratively computes the adversarial saliency value $S(x,t)[i]$ of pixel i of the image x according to the class t.

$$S(x,t)[i] = \begin{cases} 0 \quad \text{if} \quad \frac{\mathrm{d}L_t}{\mathrm{d}x_i}(x) < 0 \quad \text{or} \quad \sum_{j\neq t} \frac{\mathrm{d}L_j}{\mathrm{d}x_i}(x) > 0 \\ \frac{\mathrm{d}L_t}{\mathrm{d}x_i}(x) \, \left|\sum_{j\neq t} \frac{\mathrm{d}L_j}{\mathrm{d}x_i}(x)\right| \text{ otherwise.} \end{cases}$$

and use it iteratively to produce x_{adv} classified as t by the network.

Defensive distillation



Training a network with explicit relative information about classes prevents models from fitting too tightly to the data.

Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In Security and Privacy (SP), 2016 IEEE Symposium on (pp. 582-597). IEEE.