Modélisation et Reconnaissance des formes
Regression avec les MLP
(Perceptron multi-couches)

Marie-Odile Berger
http://members.loria.fr/moberger

3 août 2023

Regression non linéaire

- La régression vise, à partir de couples de données (x_i, y_i), à prédire ce que serait pour un x donné la valeur correspondante de y
- Dans le cas de la regression linéaire, le lien entre y et x est de la forme $y = ax + b$ et on détermine a et b en minimisant la distance entre vameur prédite et valeur attendue
- si le lien $y = f(x)$ entre (x_i, y_i) est non linéaire mais connu paramétriquement, par exemple $y = e^{-ax} * ln(x + b)$, on peut toujours minimiser sur a et b $\sum(y_i - e^{-ax} * ln(x_i + b))^2$.. mais le calcul de a et b n'est pas direct mais sera fait par minimisation intérative (descente de gradient)
- Très souvent, on ne connait pas de forme paramétrique de la fonction f : reconstruction d'un environemment à partir de données 3d, observation de systèmes physique complexes...
- $\Rightarrow$ utilisation fréquente de **perceptrons multicouches** pour la régression

Idée générale

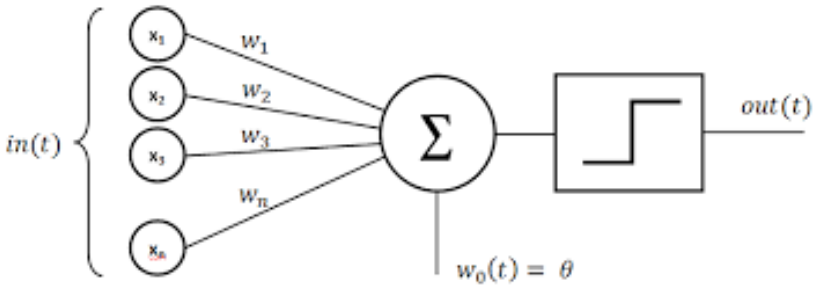
Utiliser un réseau de neurones (un perceptron multi-couches) comme fonction pour la regression
Questions :

- Ca fait beaucoup de paramètres à calculer non ?
- Cela ne risque pas d'être sensible au bruit ?
- cela marche pour quelles fonctions ?

En fait, non...

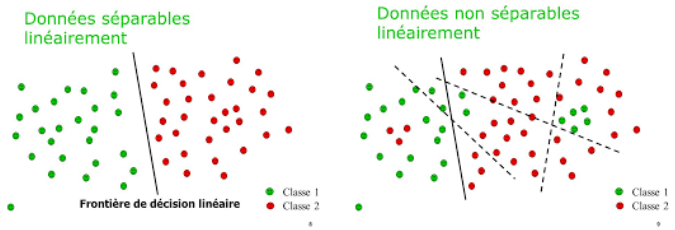
- De nombreux travaux utilisent la regresion pour les MLP en modélisation, vision par ordinateur, physique.
- ils permettent de regression des formes complètes sans besoin d'une fonction paramétrique pour les décrire
- On parlera aussi des PINs (Physically informed networks)

Au départ, il y a le perceptron de Rosenblatt

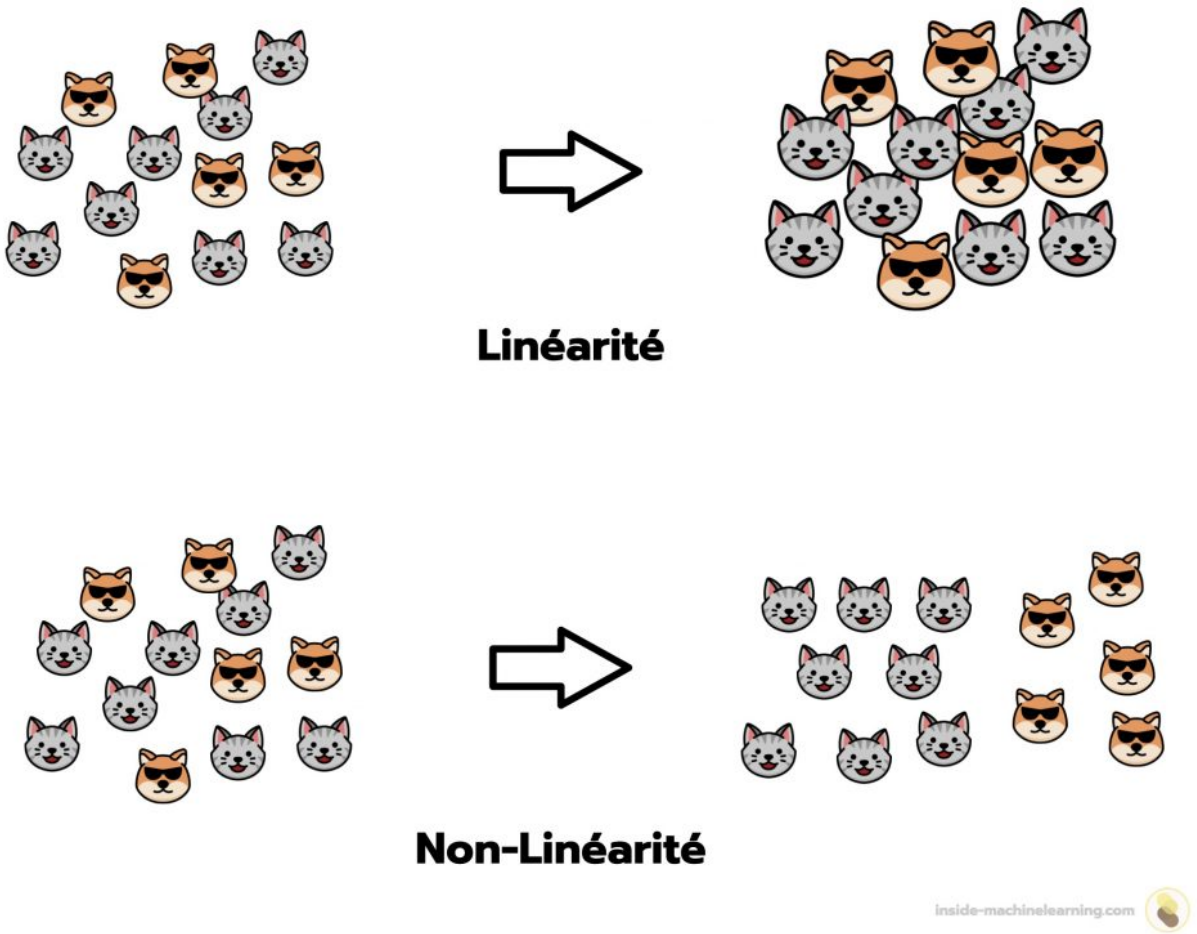


- Le perceptron est un algorithme d'apprentissage supervisé de classifieurs binaires
- il a n entrées $(x_1, \ldots, x_n)$
- la sortie o est définie par la donnée de n poids $w_1, \ldots, w_n$ et un biais (ou seuil) θ :
  $o = 1$ *si* $\sum_i w_i x_i \geq \theta, 0$ *sinon*.
  Qu'on peut aussi voir comme $H(\sum_i w_i x_i - \theta)$ où H est la fonction de Heaviside.

- Rosenblatt a proposé en 1957 un algorithme pour trouver les paramètres w_i séparant les deux classes de données (doc)
- Théorème de Novikoff : L'algorithme du Perceptron de Rosenblatt converge ssi l'échantillon de données est linéairement séparable.



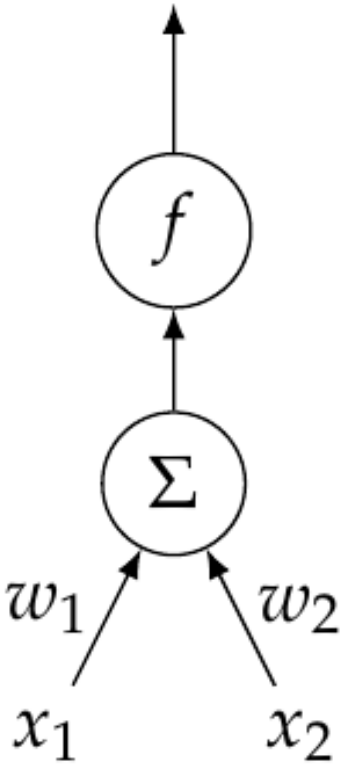
Mais la linéarité est très limitante



Tiré de https://inside-machinelearning.com/fonction-dactivation-comment-ca-marche-une-explication-simple/

Le perceptron multicouches et les fonctions d'activation

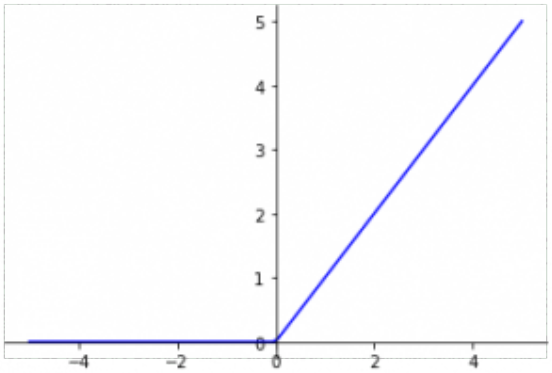
Un référence générale [GBC16]



- Comme pour les neurones biologiques, la fonction d'activation f permet à chaque neurone de lisser ou de normaliser les données d'entrée qu'elle reçoit avant de les transmettre ou non.
- $\hat{y} = f(\sum w_i x_i + b)$
- Cela permet de sortir du cas linéaire !

Les principales fonctions d'activation

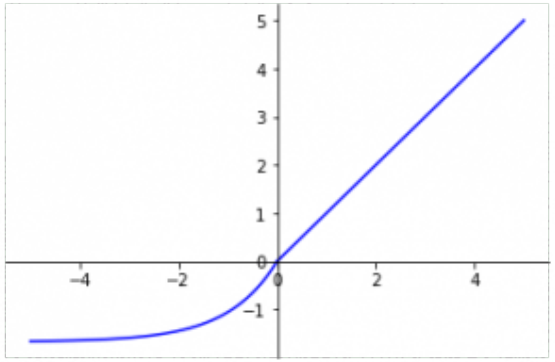
devraient être dérivables pour permettre le calcul des poids du réseau par retropropagation du gradient.



max(x, 0)

RELU

RectifiedLinearUnit

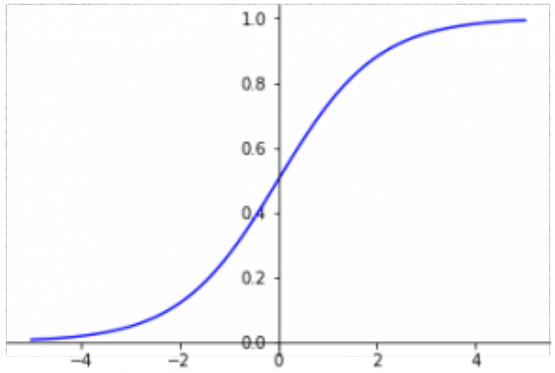


$if\ x > 0 : x$

$if\ x < 0 : \alpha * (e^x - 1)$

ELU

ExponentialLinearUnit

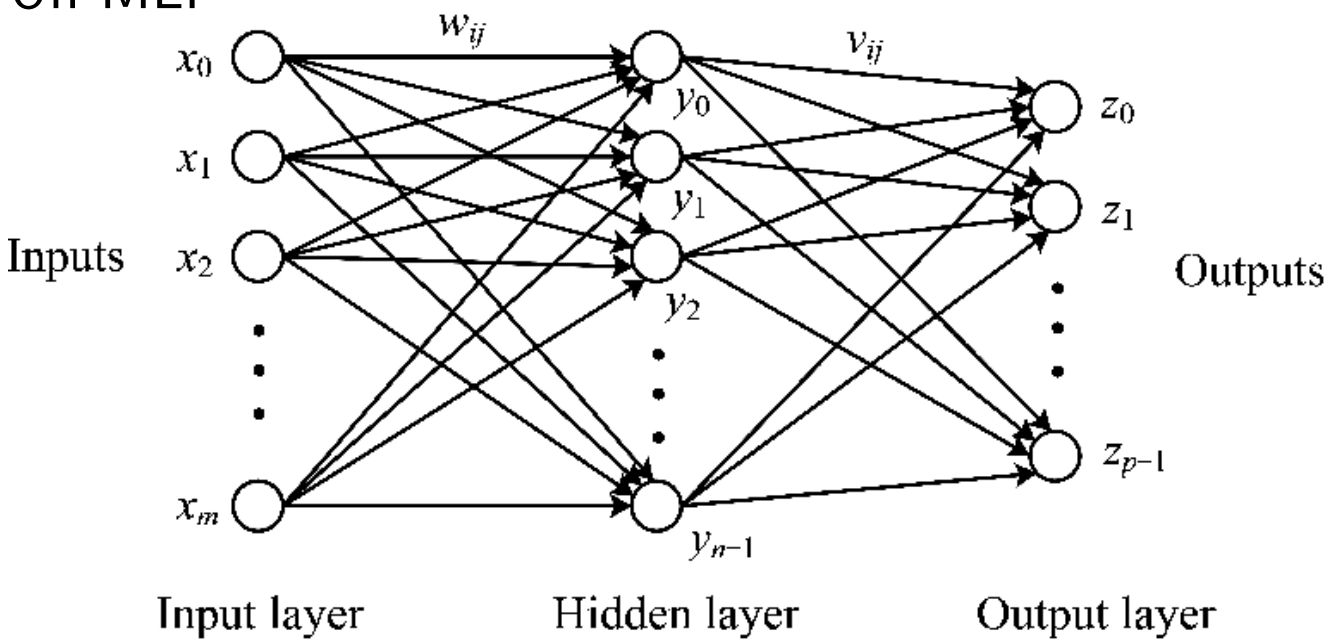


$1\ /\ (1 + e^{-x})$

Sigmoid

Le perceptron multicouches (multilayer perceptron, MLP)

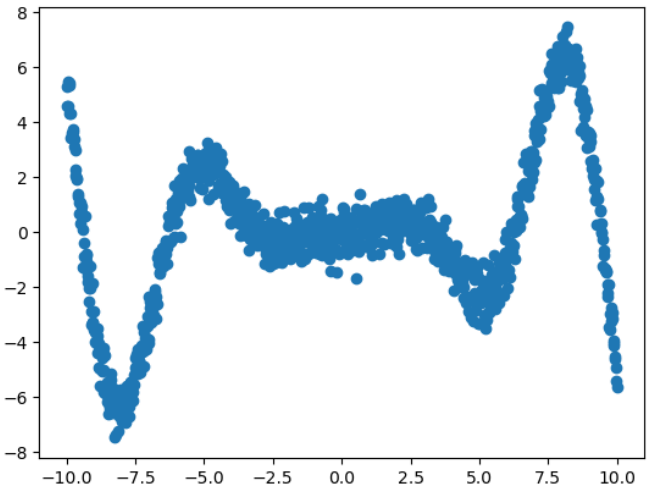
Un MLP

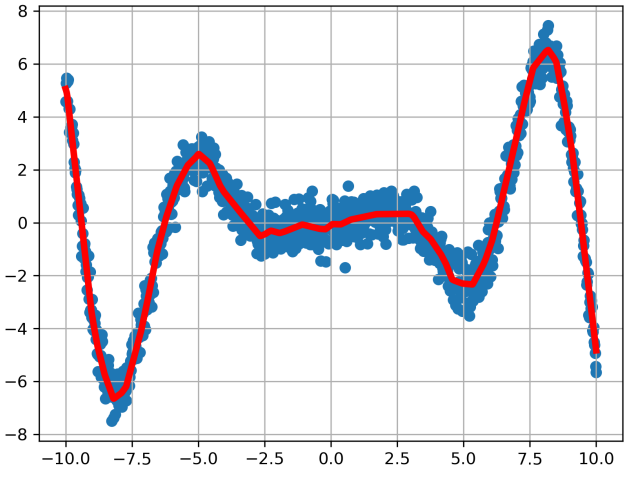


- une ou plusieurs couches cachées avec une fonction d'activation (identique pour tous les neurones de la couche)

Exemple

Le notebook correspondant (avec tensorflow) est ici
Les données à approximer (1000 points)





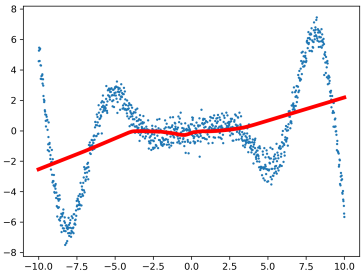
Code

Réseau à 2 couches avec 64 noeuds et une activation 'relu'.
500 epochs (on passe 500 fois sur l'ensemble des données
mse=mean squares error : $\frac{1}{N}\sum(\hat{y}_i - y_i)^2$ optimisation : adam (voir le cours sur l'optimisation)

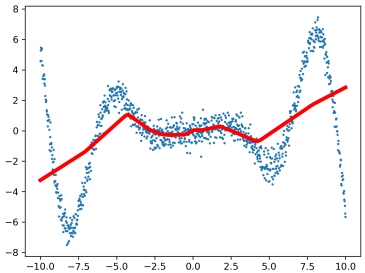
```
# Create the model
 model = keras.Sequential()
model.add(keras.layers.Dense(units = 1, activation = 'linear', input_shape=[1]))
model.add(keras.layers.Dense(units = 64, activation = 'relu'))
model.add(keras.layers.Dense(units = 64, activation = 'relu'))
model.add(keras.layers.Dense(units = 1, activation = 'linear'))
model.compile(loss='mse', optimizer="adam")
# Training
model.fit( x_data, y_data, epochs=500, verbose=1)
# Compute the output
y_predicted = model.predict(x_data)

# Display the result
plt.scatter(x_data[::1], y_data[::1])
plt.plot(x_data, y_predicted, 'r', linewidth=4)
plt.grid()
plt.show()
```

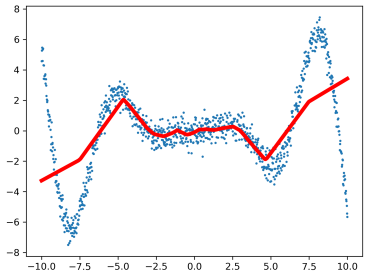
Evolution de la solution au cours des itérations



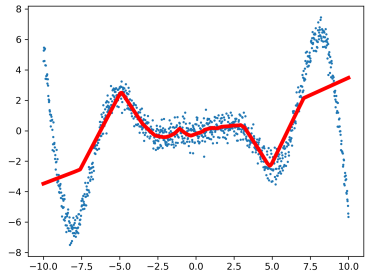
initial



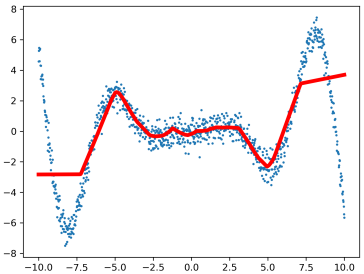
20 *epochs*



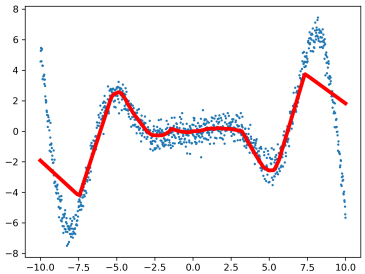
40



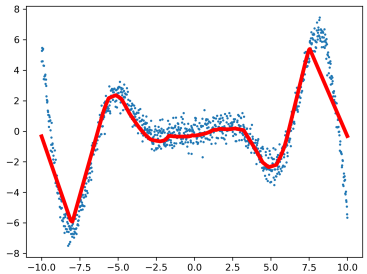
60



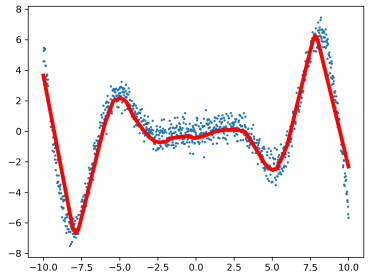
80



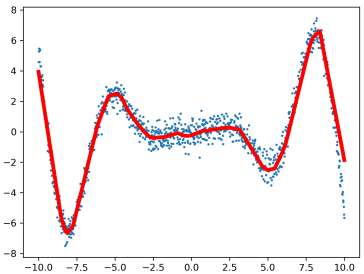
100



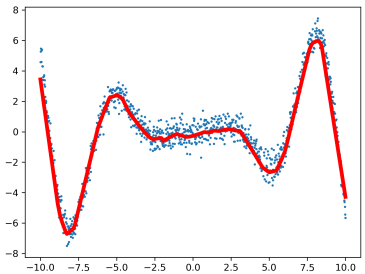
120



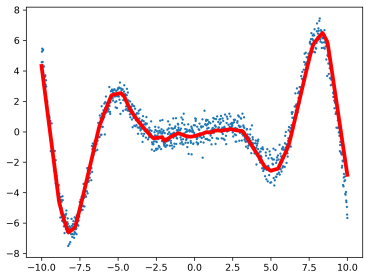
140



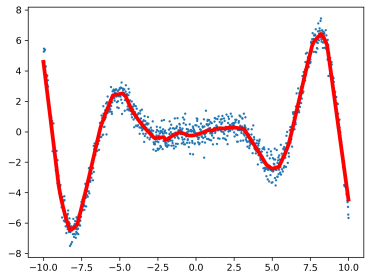
160



180



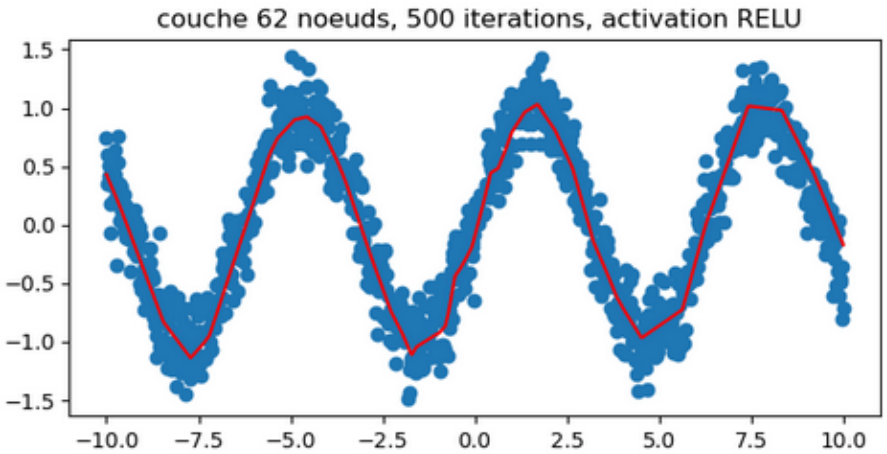
200

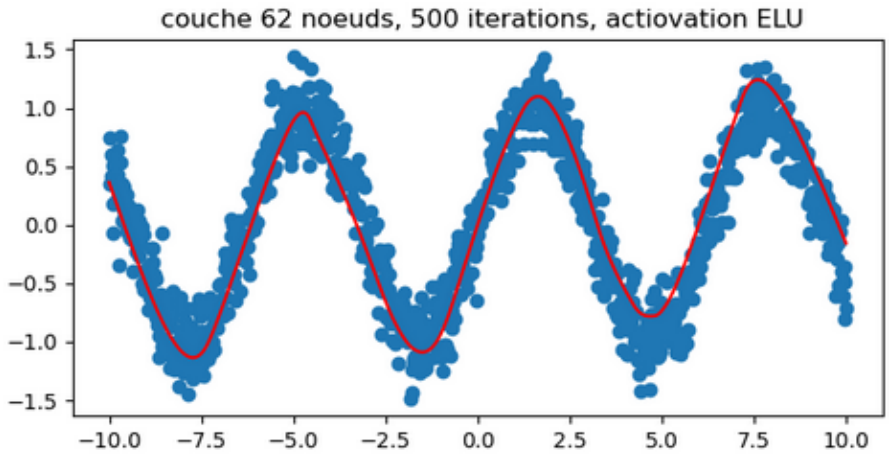


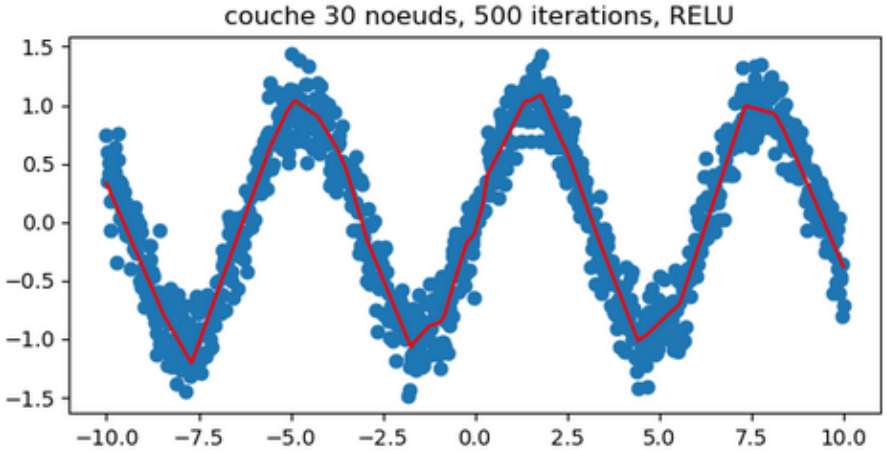
220

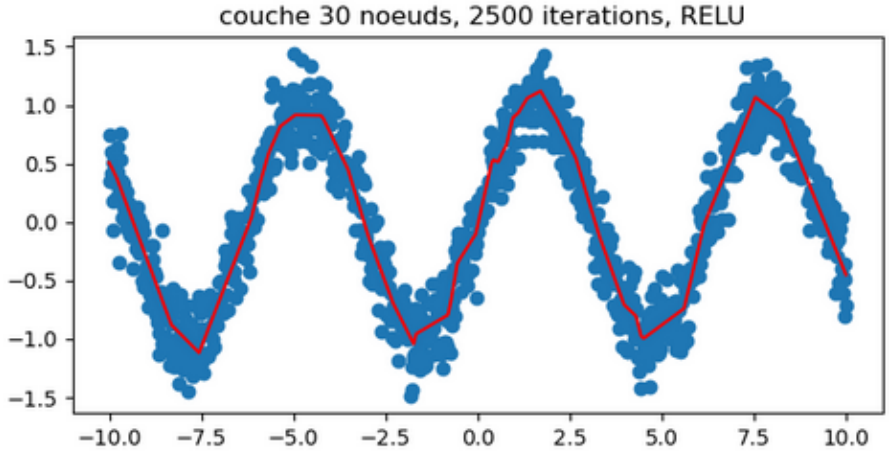
Faisons varier les paramètres

Toujours 2 couches, nombre d'époques, de noeuds, d'activation différents



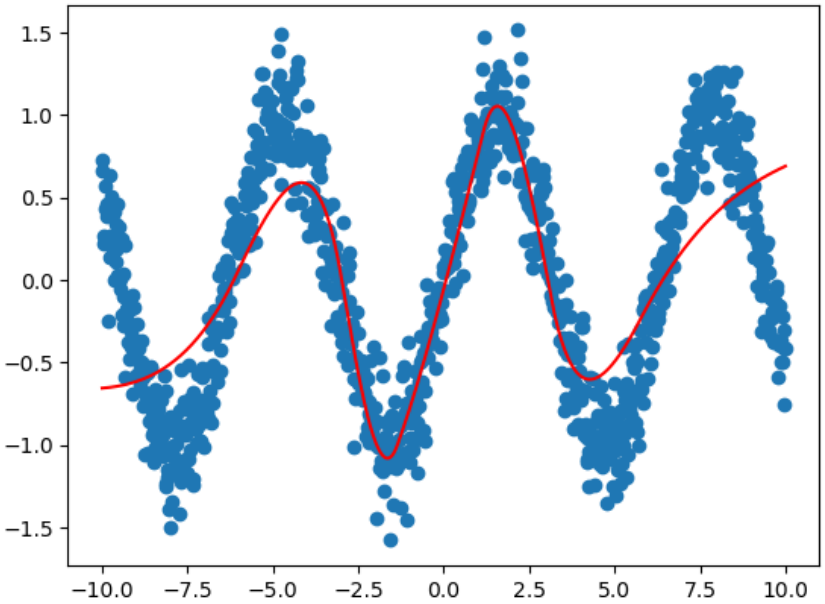




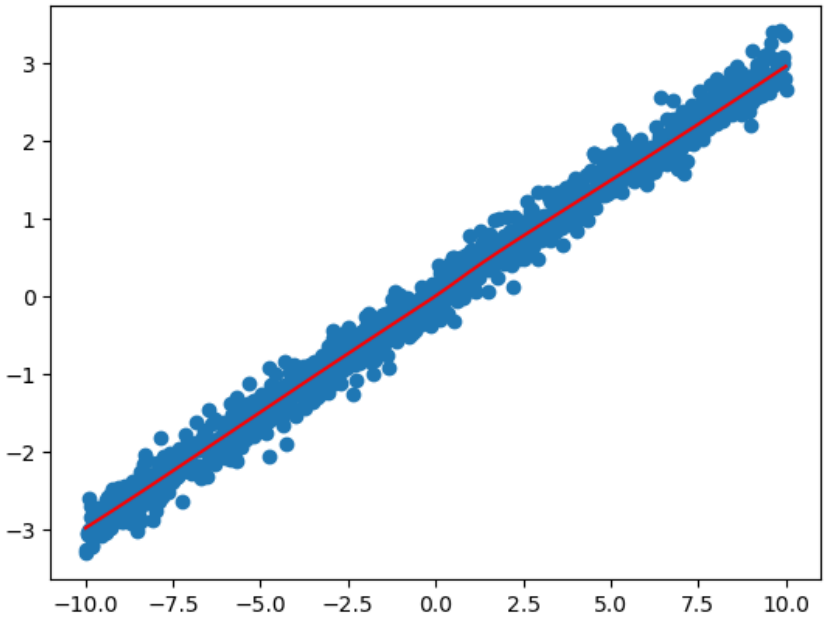


suite

1 *couche mais* 120*noeuds*



droite



Théorème d'approximation universelle

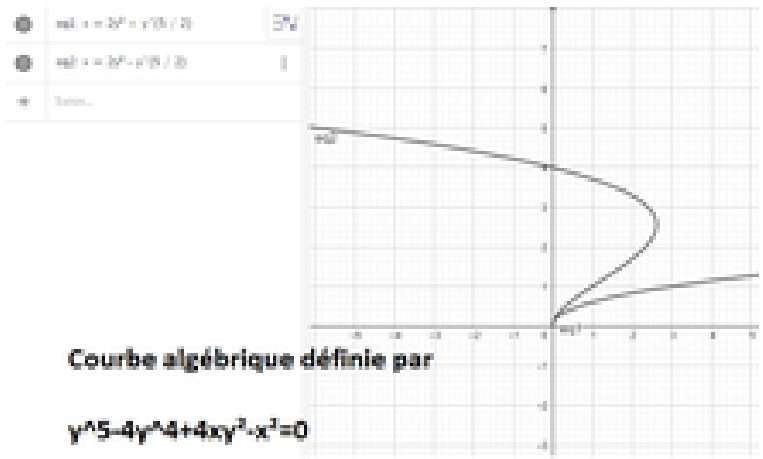
- Cela a l'air de marcher mais est ce que cela marche pour n'importe quelle fonction ?
- Plusieurs travaux ont contribué à montrer le théorème d'approximation universelle *un réseau à propagation avant d'une seule couche cachée contenant un nombre fini de neurones (c'est-à-dire, un perceptron multicouche) peut approximer des fonctions continues sur des sous-ensembles compacts de R^n.*
- le résultat est théorique. Il ne dit rien sur l'architecture à adopter, ni les fonctions d'activations spécifiques
- il existe cependant des travaux (voir par exemple [KH91, Pin99]) visant à analyser mathématiquement les compétence de certains réseaux

Avantages et inconvénients des MLP

- Quand on connait la forme paramétrique des données, utiliser un moindres carrés linéaire ou non linéaire
- quand la forme des données est inconnue, un MLP se révèle très efficace.
  - utilisation massive en modélisation de scène en vision ou computers graphics (NERF et compagnie) [MST+20, TLY+21, OCD+22]
- un MLP reste dépendant de l'architecture du réseau (combien de couches ? de noeuds par couches ?) et des paramètres de la minimisation. Mais pas tant que cela.

Les représentations implicites

- Les données (x, y) des exemple précédent vient d'une représentation fonctionnelle de type $y = f(x)$ (à une valeur de x correspond une valeur de y) $\rightarrow$ difficile à utiliser pour créér un modèle de scène urbain, ou un plan cadastral !
- Les représentations implicites sont plus générales : un ensemble y est défini pat $\{(x, y)|f(x, y) = 0\}$
- exemples : $x^2 +^y 2/4 - 1 = 0$ est la représentation implicite d'une ellipse



Représentation par SDF (signed distance function)

- La distance signée à un ensemble $\mathcal{S}$ est la distance orthogonale d'un point à $\mathcal{S}$

$$f(x) = \begin{cases} d(x, \delta\Omega) if x \in \Omega \\ -d(x, \delta\Omega) if x \notin \Omega \end{cases}$$

- La forme correspond à $\{x | f(x) = 0\}$

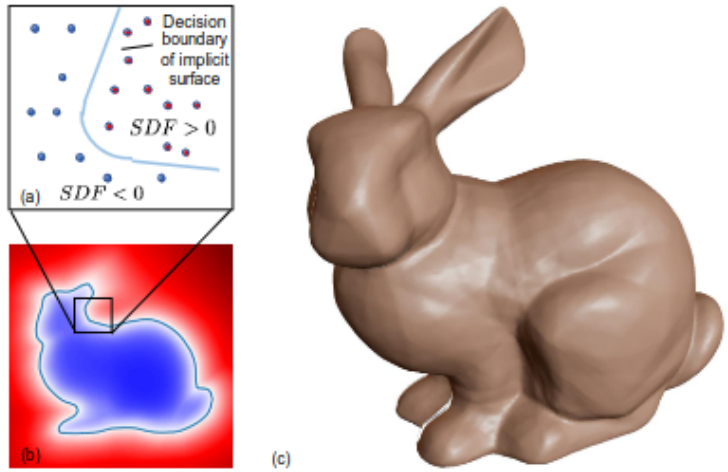
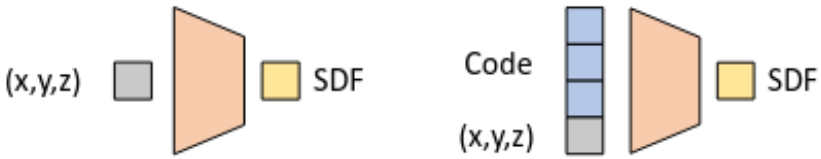


Figure 2: Our DeepSDF representation applied to the Stanford Bunny: (a) depiction of the underlying implicit surface $SDF = 0$ trained on sampled points inside $SDF < 0$ and outside $SDF > 0$ the surface, (b) 2D cross-section of the signed distance field, (c) rendered 3D surface recovered from $SDF = 0$. Note that (b) and (c) are recovered via DeepSDF.

tiré de [PFS+19]

L'exemple de DeepSDF [PFS+19]

- input : nuage de pouints 3D (LiDar, camera RGBD,...)
- objectif : reconstruire une forme à partir des points par MLP mais avec un réseau conçu non pas pour une forme donnée mais pour une grande varité de formes
- méthode : un réseau commun à toutes les formes (appris sur un vaste ensemble de formes) + un code de la forme considérée (figure b)



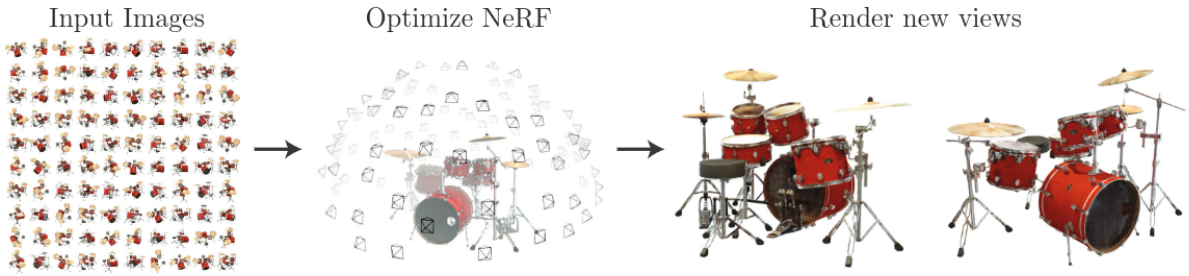
(a) Single Shape DeepSDF **(b)** Coded Shape DeepSDF

- article à lire pour la semaine prochaine

Les NeRF (neural radiance fields) et compagnie I

D'autres travaux dans la continuité avec pour la plupart une représentation de forme par MLP destinée à une seule forme

- Pour accélérer le procédé une représentation par octree est utilisée et un descripteur est attaché à chaque noeud [ZPL+22]
- Les NeRF [MST+20] visent à produire un représentation d'une forme pour la synthèse de vues à partir d'images calibrées (position des caméras connues)
  - input (x,y,z,)+ direction de vue (2 angles)
  - output : densité et radiance



  - méthode : MLP ; Loss : erreur entre couleur simulée et couleur vérité terrain (disponibles dans les images)

Les NeRF (neural radiance fields) et compagnie II

- A noter : l'entrée du réseau n'est pas uniquement la position d'un point p mais utilise du "positional encoding" $\gamma(p) = sin(2^0\pi p), cos(2^0\pi p), ..., sin(2^L\pi p), cos(2^L\pi p))$ ce qui apporte une bien meilleure qualité à la surface reconstruite. Voir [TSM+20] pour une explication théorique (les réseaux sont biaisés vers l'apprentissage de fonctions basses fréquence)

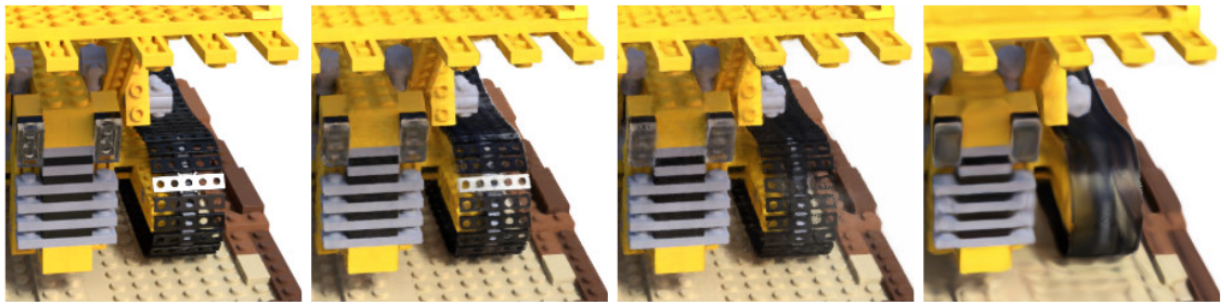


Fig. 4: Here we visualize how our full model benefits from representing view-dependent emitted radiance and from passing our input coordinates through a high-frequency positional encoding. Removing view dependence prevents the model from recreating the specular reflection on the bulldozer tread. Removing the positional encoding drastically decreases the model's ability to represent high frequency geometry and texture, resulting in an oversmoothed appearance.

Quelques idées sur les PINs

Bibliographie I


Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, Cambridge, MA, USA, 2016.
http://www.deeplearningbook.org.


Kurt and Hornik.
Approximation capabilities of multilayer feedforward networks.
Neural Networks, 4(2) :251–257, 1991.


Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng.
Nerf : Representing scenes as neural radiance fields for view synthesis.
In *eccv2020*, 2020.

Bibliographie II


Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam.
isdf : Real-time neural signed distance fields for robot perception.
In *Robotics : Science and Systems (RSS) 2022*, 2022.


Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove.
Deepsdf : Learning continuous signed distance functions for shape representation.
CoRR, abs/1901.05103, 2019.


Allan Pinkus.
Approximation theory of the mlp model in neural networks.
Acta Numerica, 8 :143–195, 1999.

Bibliographie III

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler.

Neural geometric level of detail : Real-time rendering with implicit 3D shapes.

2021.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng.

Fourier features let networks learn high frequency functions in low dimensional domains.

CoRR, abs/2006.10739, 2020.

Bibliographie IV


Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys.
NICE-SLAM : neural implicit scalable encoding for SLAM.
In *CVPR2022*, 2022.