Deep Learning

Power of Depth

Lionel Fillatre

2025-2026

Outline

- One hidden-layer
- Limits of a single hidden layer
- Vanishing gradients
- Excess error
- Conclusion

One hidden-layer

ReLU Deep Networks

- Let us remember that a feed-forward deep neural network has the form :

$$\hat{y} = f(x;\theta) = \left(f_{\theta_L} \circ f_{\theta_{L-1}} \circ \cdots \circ f_{\theta_1}\right)(x)$$

where $\theta = \{\theta_k : k = 1, \cdots, L\}$ denotes the model parameters

- Les us assume that $\theta_k = (W_k, b_k)$ and, for any $h \in \mathbb{R}^{n_{k-1}}$,

$$f_{\theta_k}(h) = \sigma(W_k h + b_k)$$

where, for any $z \in \mathbb{R}^n$,

$$\sigma(z) = \begin{pmatrix} \mathrm{ReLU}(z_1) \\ \vdots \\ \mathrm{ReLU}(z_n) \end{pmatrix} = \begin{pmatrix} \max\{0, z_1\} \\ \vdots \\ \max\{0, z_n\} \end{pmatrix}$$

Approximation with ReLU nets

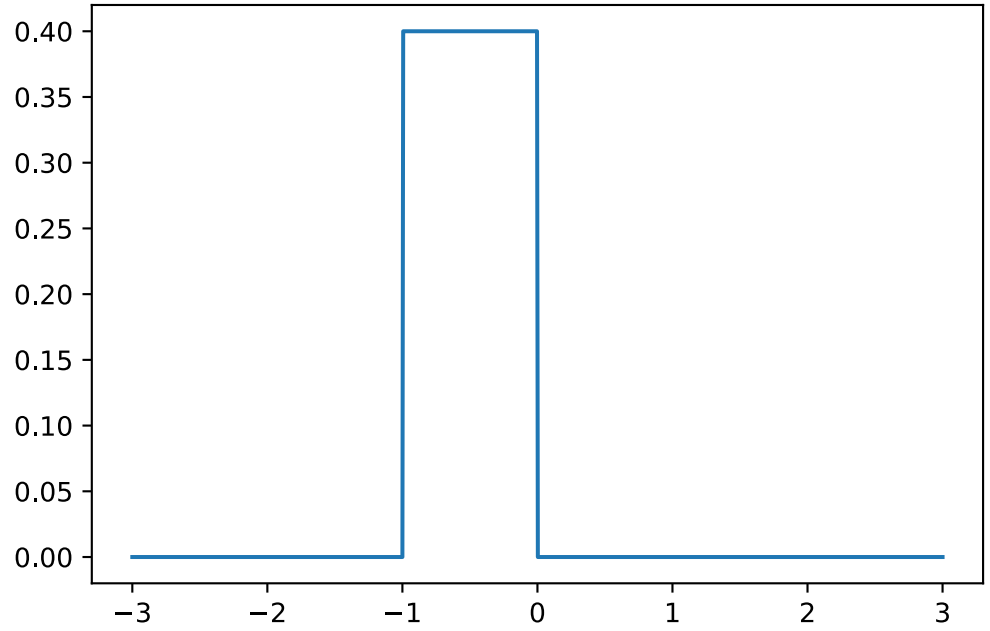
```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
        return np.maximum(x, 0)

def rect(x, a, b, h, eps=1e-7):
        return h / eps * (
                relu(x - a)
                - relu(x - (a + eps))
                - relu(x - b)
                + relu(x - (b + eps)))

x = np.linspace(-3, 3, 1000)
y = ( rect(x, -1, 0, 0.4))

plt.plot(x, y)
```



Approximation with ReLU nets

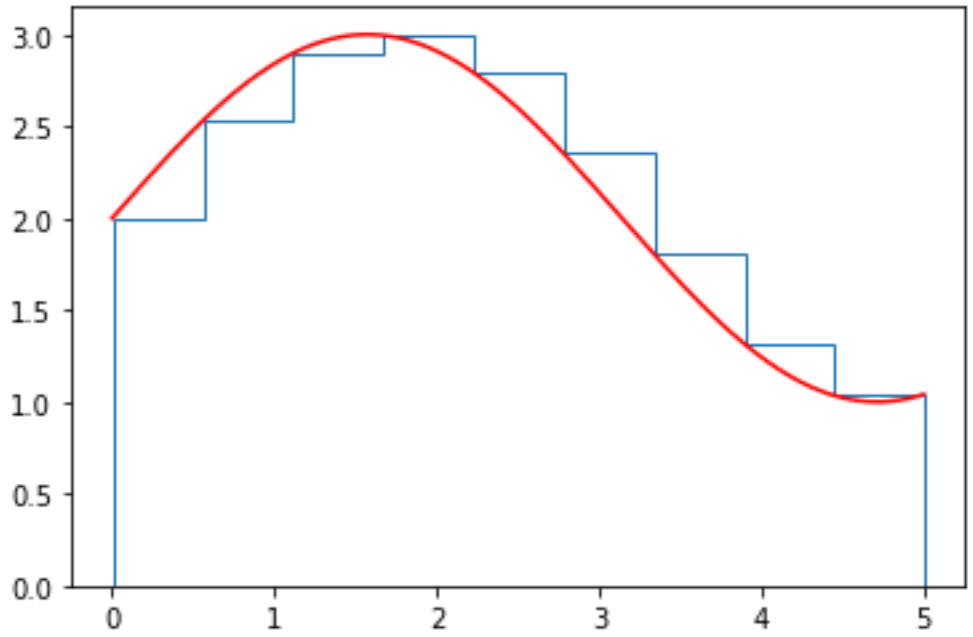
```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
        return np.maximum(x, 0)

def rect(x, a, b, h, eps=1e-7):
        return h / eps * (
                relu(x - a)
                - relu(x - (a + eps))
                - relu(x - b)
                + relu(x - (b + eps)))

x = np.linspace(0.0,5.0,10) # 10 samples on the x-axis
z = np.linspace(0.0,5.0,500) # high resolution x-axis
sin_approx = np.zeros_like(z) # approximation of the sinusoid
for i in range(0, x.size-1):
   sin_approx = sin_approx + rect(z,x[i],x[i+1], 2+np.sin(x[i]))
plt.plot(z, 2+np.sin(z),'r-')
plt.stairs(sin_approx[0:z.size-1], z)
plt.show()
```

Approximation of $f(x) = 2 + \sin(x)$



Conner Davis, [Quora: Is a single layered ReLU network still a universal approximator?]

Approximation with ReLU nets

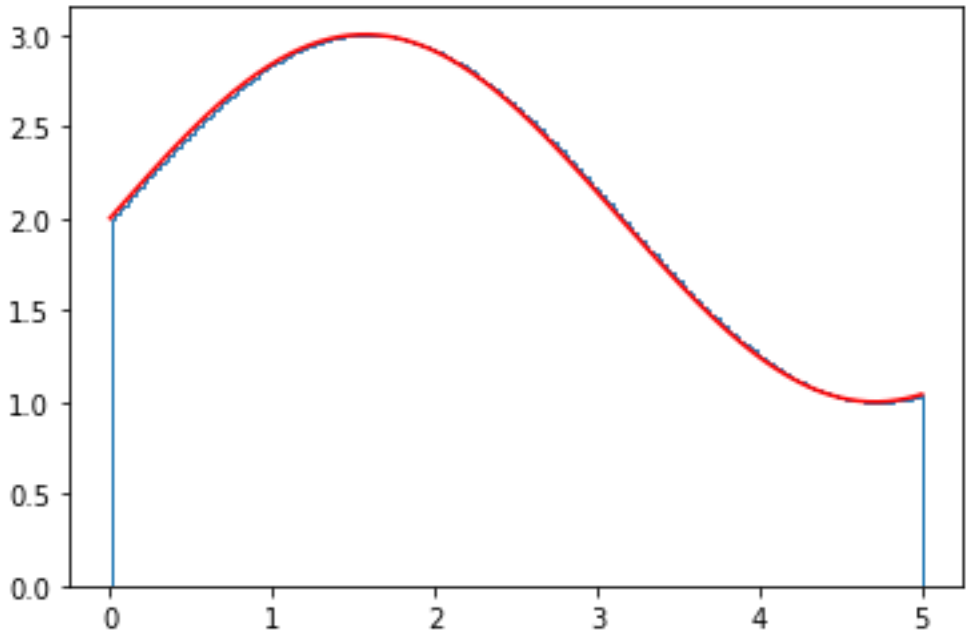
```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
        return np.maximum(x, 0)

def rect(x, a, b, h, eps=1e-7):
        return h / eps * (
                relu(x - a)
                - relu(x - (a + eps))
                - relu(x - b)
                + relu(x - (b + eps)))

x = np.linspace(0.0,5.0,200) # 200 samples on the x-axis
z = np.linspace(0.0,5.0,500) # high resolution x-axis
sin_approx = np.zeros_like(z) # approximation of the sinusoid
for i in range(0, x.size-1):
    sin_approx = sin_approx + rect(z,x[i],x[i+1], 2+np.sin(x[i]))
plt.plot(z, 2+np.sin(z),'r-')
plt.stairs(sin_approx[0:z.size-1], z)
plt.show()
```

Approximation of $f(x) = 2 + \sin(x)$



Conner Davis, [Quora: Is a single layered ReLU network still a universal approximator?]

Universal function approximation

Theorem. (Hornik et al, 1991)

- Let σ be a nonconstant, bounded, and monotonically-increasing continuous function.
- For any $f \in C\left([0,1]^d\right)$ and $\varepsilon > 0$, there exists $q \in \mathbb{N}$, real constants $v_i, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^d$ such that:

$$\left| \sum_{i=1}^{q} v_i \sigma\left(w_i^T x + b_i\right) - f(x) \right| < \varepsilon$$

- In other words, neural nets are dense in $C\left([0,1]^d\right)$.

- It guarantees that even a single hidden-layer network can represent any classification problem in which the boundary is locally linear (smooth);
- It does not inform about good/bad architectures, nor how they relate to the optimization procedure.
- The universal approximation theorem generalizes to any non-polynomial (possibly unbounded) activation function, including the ReLU (Leshno, 1993).

K. Hornik et al., Approximation Capabilities of Multilayer Feedforward Networks, 1991

Upper-Bound for one-hidden layer network

Theorem (Barron, 1994)

- The mean integrated square error $\int \left(\hat{f}(x) - f(x)\right)^2 dx$ between the estimated network $\hat{f}(x) = \sum_{i=1}^{q} v_i \sigma(w_i^T x + b_i) + v_0$ and the target function f is bounded by

$$O\left(\frac{C_f^2}{q}\right) + O\left(\frac{qn}{N}\log N\right)$$

where N is the number of training points, q is the number of neurons, n is the input dimension, and C_f measures the global smoothness of f.

- Provided enough data, it guarantees that adding more neurons will result in a better approximation.

- For your information,

$$C_f = \int \|\omega\|_1 \tilde{f}(\omega) d\omega$$

where $\tilde{f}(\omega)$ is the Fourier transform of $f(x)$.

Barron, A.R., Approximation and estimation bounds for artificial neural networks, Mach Learn, 1994

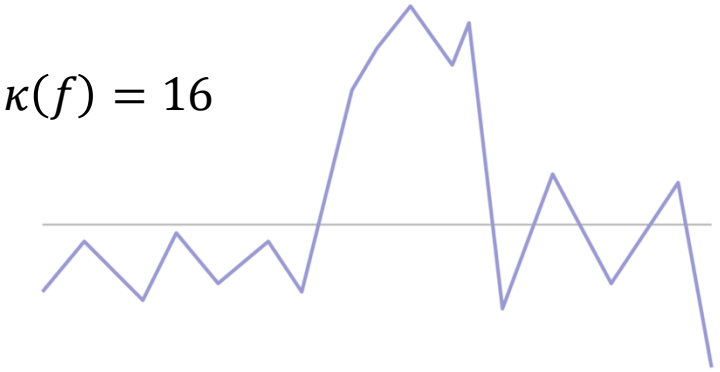
Problem solved?

- Universal function approximation theorems do not tell us:
  - The number q of hidden units is small enough to have the network fit in RAM.
  - There is no constructive way to find an optimal solution
  - The optimal function parameters can be found in finite time by minimizing the Empirical Risk with SGD and the usual random initialization schemes.

- Going deeper?
  - Why would it be a good idea to stack more layers?
  - Are there any theoretical insights for doing his? Empirical ones?

Limits of a single hidden layer

Number of linear pieces



- Let $\mathcal{F}$ be the set of piecewise linear mappings on $[0,1]$
- Let $\kappa(f)$ be the minimum number of linear pieces needed to represent $f \in \mathcal{F}$.
- Let $\sigma: \mathbb{R} \to \mathbb{R}$ be the ReLU function

$$\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$$

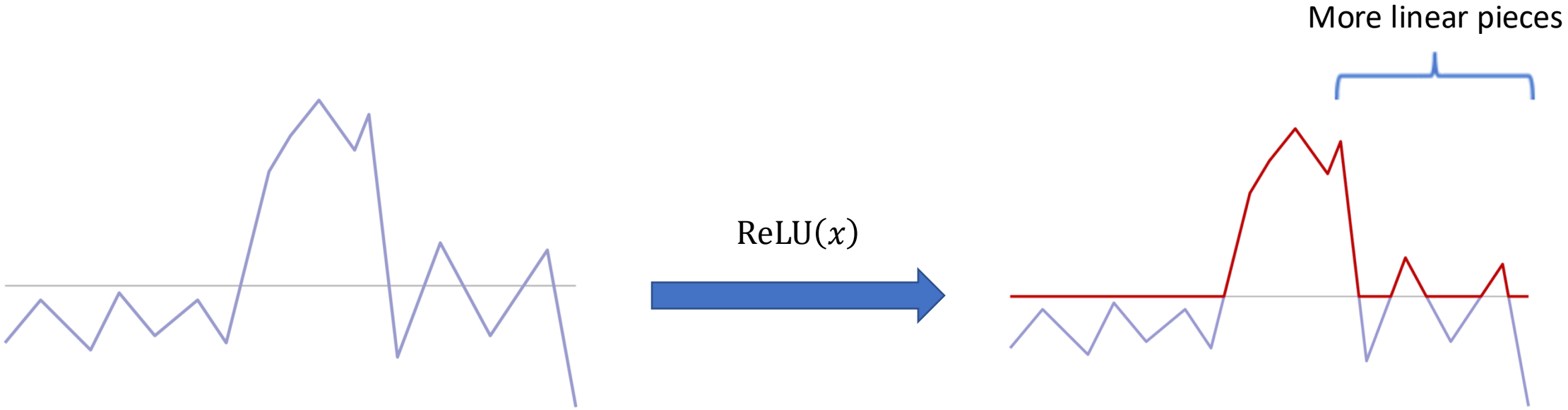
- If we compose σ and $f \in \mathcal{F}$, any linear piece that does not cross 0 remains a single piece or disappears, and one that does cross 0 breaks into two, hence

$$\forall f \in \mathcal{F}, \kappa\big(\sigma(f)\big) \leq 2\,\kappa(f)$$

- We also have

$$\forall (f, g) \in \mathcal{F}^2, \kappa(f + g) \leq\ \kappa(f) + \kappa(g)$$

Illustration



Bound on the number of linear pieces

- Consider a MLP with ReLU, D layers, a single input unit, and a single output unit.
  - Single unit input layer: $h^0 = x \in \mathbb{R}^{n_0=1}$
  - Hidden layers: $h^d = \left(h_1^d, \dots, h_{n_d}^d\right),\ \forall d = 1, \dots, D$, with $h_i^d = \sigma\left(z_i^{d-1}\right)$ and $z_i^{d-1} = \sum_{j=1}^{n_{d-1}} w_{ij}^{d-1} h_j^{d-1} + b_i^{d-1}$
  - Single unit output layer: $\hat{y} = f(x) = h^D \in \mathbb{R}^{n_D=1}$

- Then, we get

$$\forall i, \ell, \qquad \kappa\left(h_i^d\right) = \kappa\left(\sigma\left(z_i^{d-1}\right)\right) \leq 2\kappa\left(z_i^{d-1}\right) \leq 2 \sum_{j=1}^{n_{d-1}} \kappa\left(h_j^{d-1}\right)$$

- It follows that

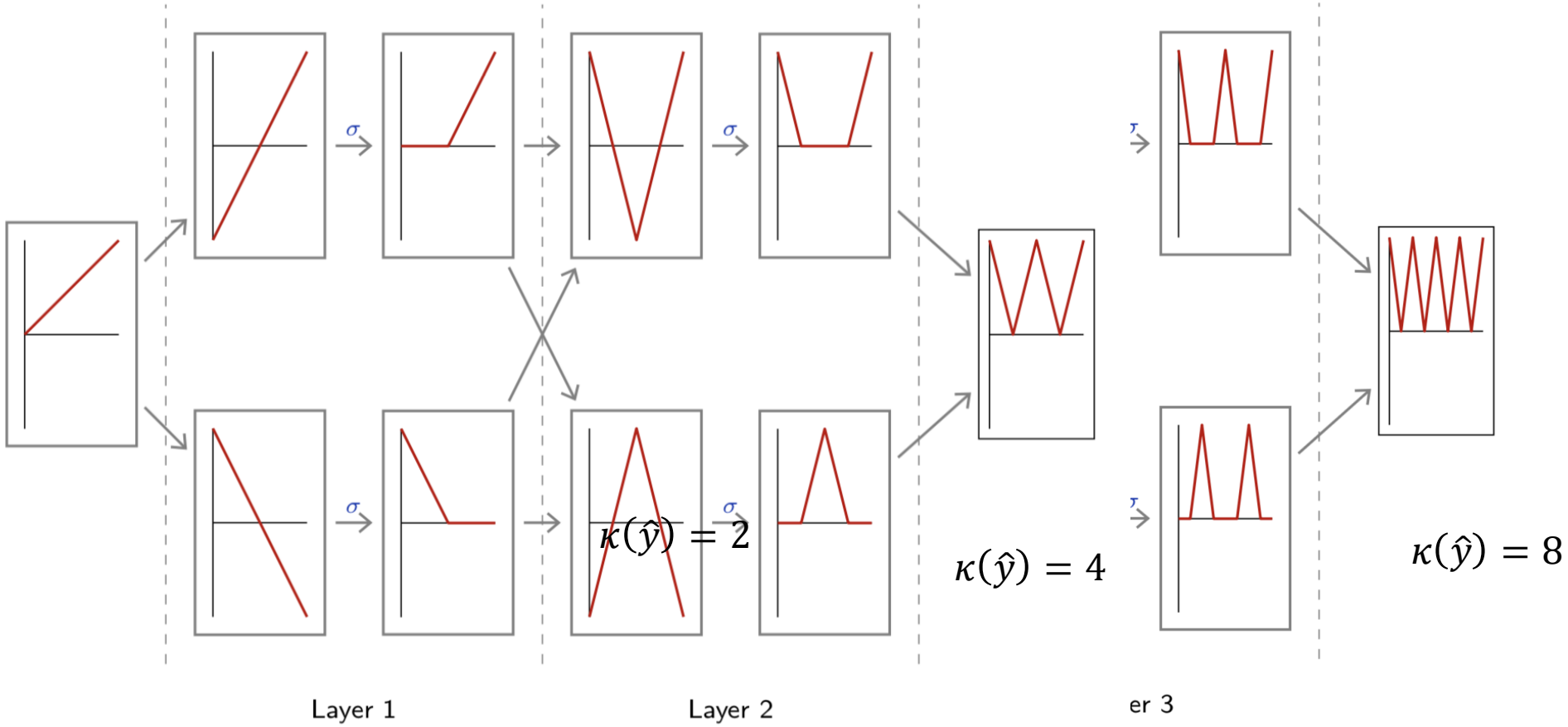
$$\forall d, \qquad \max_i \kappa\left(h_i^d\right) \leq 2n_{d-1} \max_j \kappa\left(h_j^{d-1}\right)$$

- We get the following bound for any ReLU MLP

$$\kappa(\hat{y}) \ \leq 2^D \prod_{d=1}^{D} n_d$$

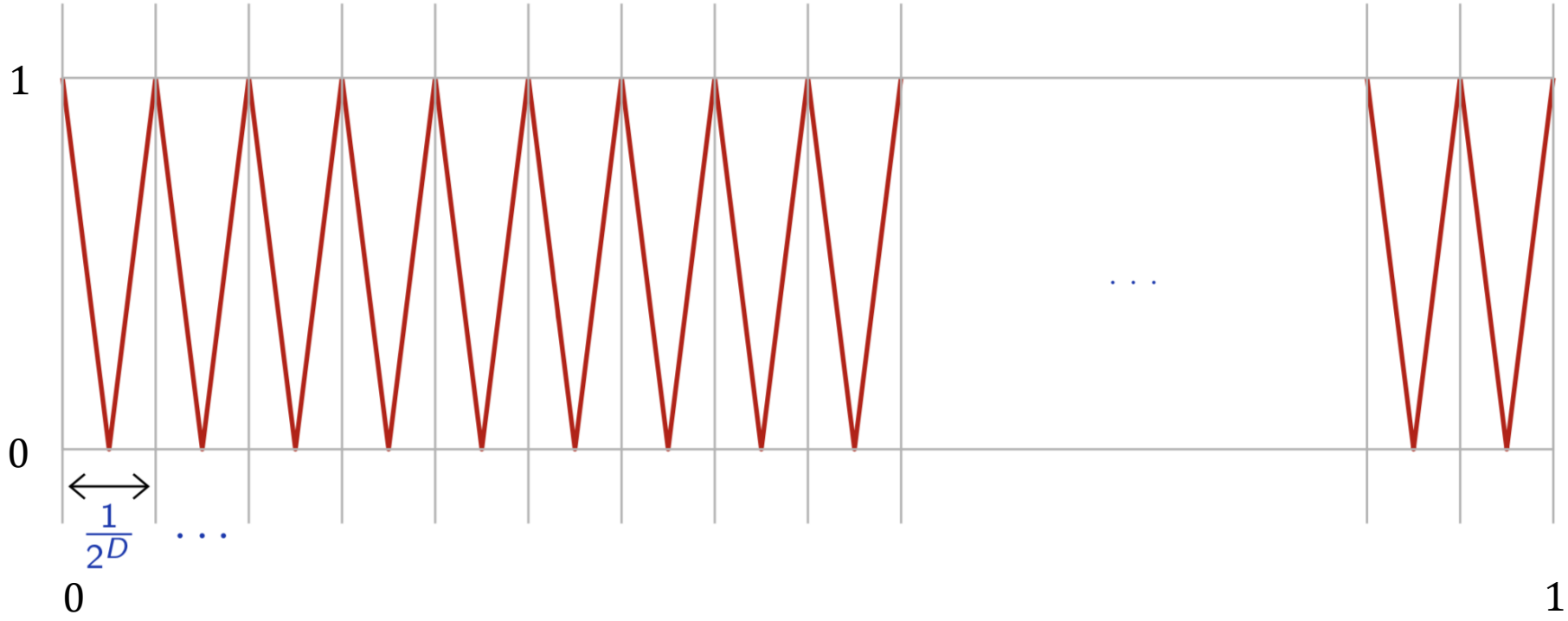
Tight bound?

- Although this seems quite a pessimist bound, we can hand-design a network that (almost) reaches it



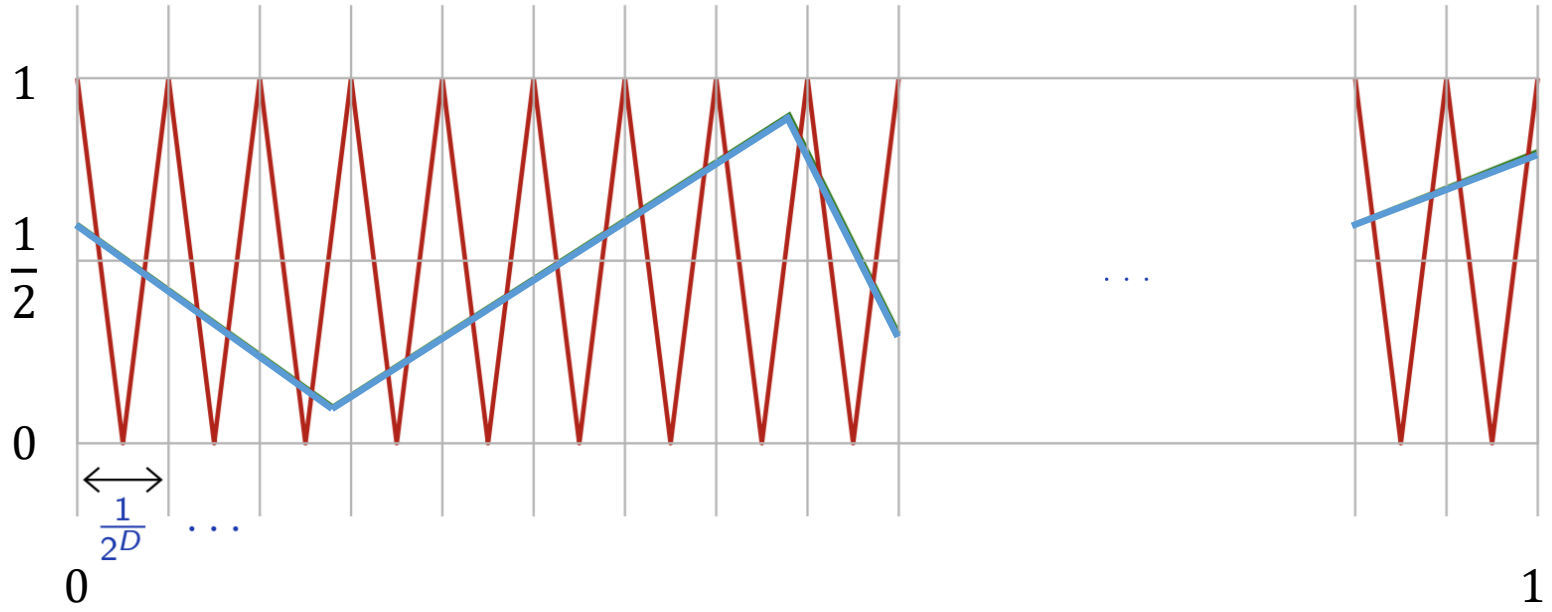
Triangle wave

- So for any D, there is a network with D hidden layers and $2D$ hidden units which computes a function $f: [0,1] \to [0,1]$ of period $\frac{1}{2^D}$



Approximation of the triangle wave

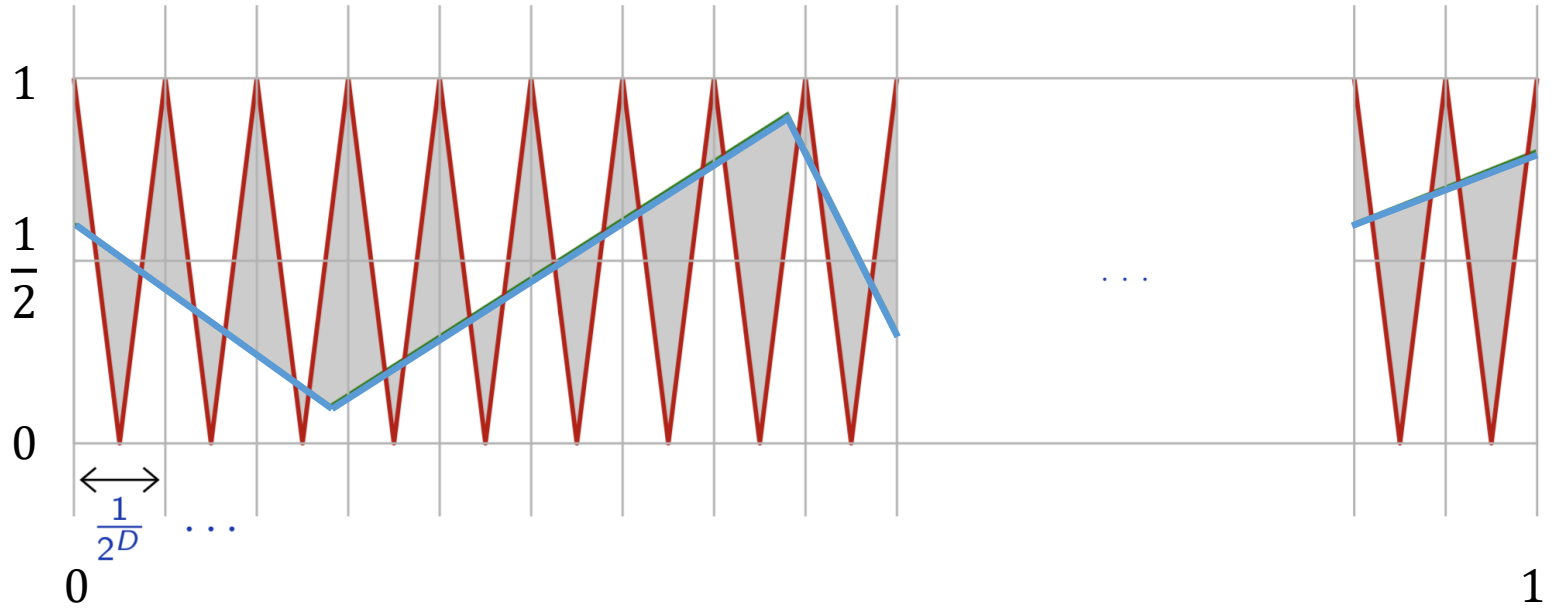
- The unit segment $[0,1]$ is cut into 2^D segments



Is the proposed approximation (in blue) accurate?

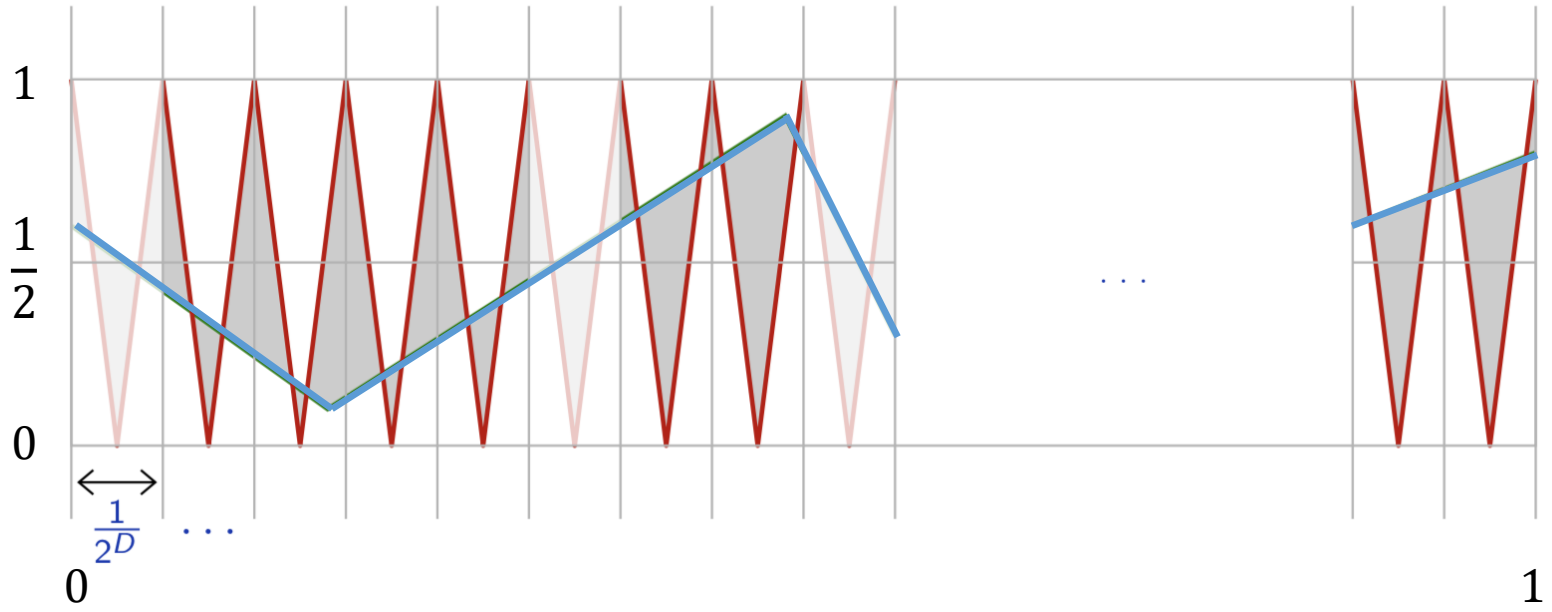
Approximation of the triangle wave

- Given $g \in \mathcal{F}$ (blue curve), it crosses $\frac{1}{2}$ at most $\kappa(g)$ times,



Approximation of the triangle wave

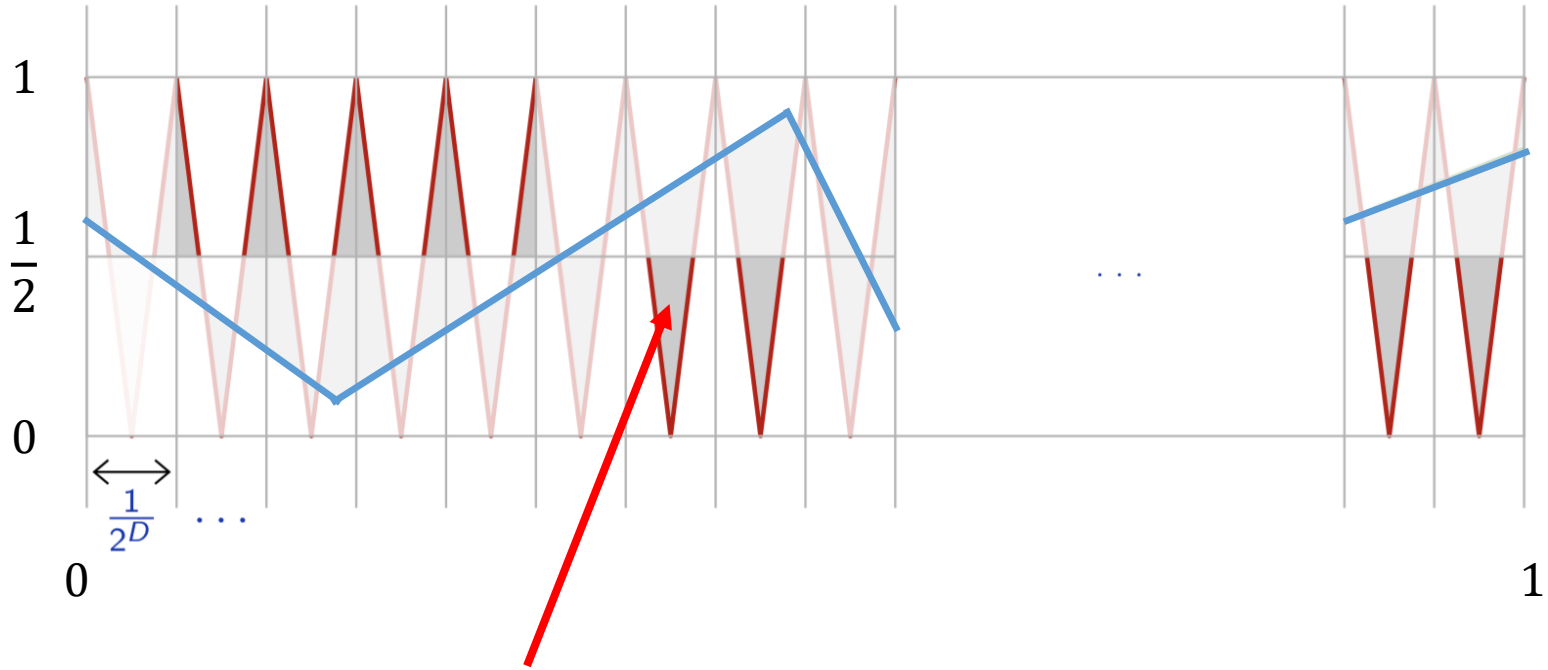
- On at least $2^D - \kappa(g)$ segments of length $\frac{1}{2^D}$, g is on one side of $\frac{1}{2}$

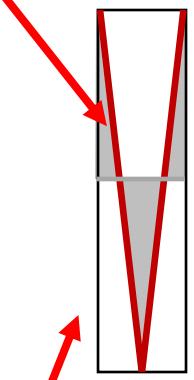


Approximation of the triangle wave

Area of the small grey right-angled triangle:

$$\frac{\frac{1}{2^D}}{\frac{4}{2}} \times \frac{1}{2} = \frac{1}{2^D} \times \frac{1}{16}$$





Area of the grey area:

$$\int_0^{\frac{1}{2^D}} \left| f(x) - \frac{1}{2} \right| dx = \frac{1}{2^D} \times \frac{1}{16} \times 4 = \frac{1}{2^D} \times \frac{1}{4}$$

Area of the isosceles triangle: $\frac{1}{2^D} \times \frac{1}{16} \times 2 = \frac{1}{2^D} \times \frac{1}{8}$

Approximation of the triangle wave $f(x)$

- Summary:
  - Given $g \in \mathcal{F}$, it crosses $\frac{1}{2}$ at most $\kappa(g)$ times,
  - Which means that, on at least $2^D - \kappa(g)$ segments of length $\frac{1}{2^D}$, it is on one side of $\frac{1}{2}$,
- It follows that

Error on 1 segment w.r.t. the constant function $c(x) = \frac{1}{2}$,
i.e., area of the isoceles triangle in previous slides

$$\int_0^1 |f(x) - g(x)|\, dx \geq \left(2^D - \kappa(g)\right) \times \frac{1}{2} \int_0^{\frac{1}{2^D}} \left| f(x) - \frac{1}{2} \right| dx$$

$$= \left(2^D - \kappa(g)\right) \times \frac{1}{2^D} \times \frac{1}{8}$$

$$= \frac{1}{8}\left(1 - \frac{\kappa(g)}{2^D}\right)$$

- We multiply f by 8 (and also g) to get the final result: $\int_0^1 |f(x) - g(x)|\, dx \geq 1 - \frac{\kappa(g)}{2^D}$

ReLU MLPs with a single input/output

- There exists a network f with D layers, and $2D$ internal units, such that, for any network g with D' layers of sizes $\{n_1, \dots, n_{D'}\}$

$$\int_0^1 |f(x) - g(x)|\, dx \geq 1 - \frac{2^{D'}}{2^D} \prod_{d=1}^{D'} n_d$$

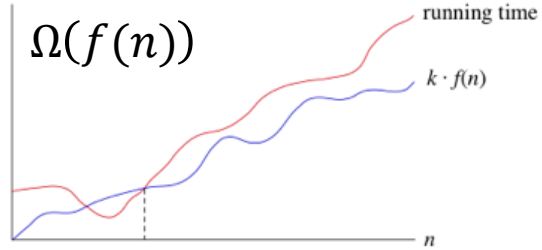
- In particular, with g a single hidden layer network ($D' = 1$)

$$\int_0^1 |f(x) - g(x)|\, dx \geq 1 - \frac{2n_1}{2^D}$$

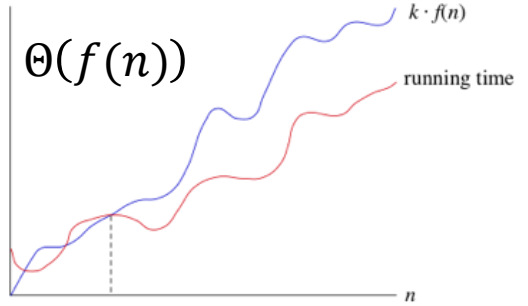
- To approximate f properly, the width n_1 of g's hidden layer has to increase exponentially with f's depth D.
- This is a simplified variant of results by Telgarsky (2015, 2016).

Depth and Parametric Cost

- **Theorem** (Telgarsky, 2016): There exists functions that can be approximated by a deep ReLU network with $\Theta(n^3)$ layers with $\Theta(1)$ units that cannot be approximated by shallower networks with $\Theta(n)$ layers unless they have $\Omega(2^n)$ units.
- Note: the number of parameters of a deep network is typically quadratic with the number of units.
- This also holds for ReLU convnets with max pooling layers.



- Notation:
  - If a running time is $\Omega(f(n))$, then for large enough n, the running time is at least $k \cdot f(n)$ for some constant C
  - If a running time is $\Theta(f(n))$, then for large enough n, the running time is at most $k \cdot f(n)$ for some constant C



The problem of depth

- Although it was known that deeper is better, for decades training deep neural networks was highly challenging and unstable.
- Besides limited hardware and data there were a few algorithmic flaws that have been fixed/softened in the last decade.
- An important issue is to control the amplitude of the gradient, which is tightly related to controlling activations.
- In particular we have to ensure that
  - The gradient does not « vanish » (Bengio et al., 1994; Hochreiter et al., 2001),
  - The gradient amplitude is homogeneous so that all parts of the network train at the same rate (Glorot and Bengio, 2010),
  - The gradient does not vary too unpredictably when the weights change (Balduzzi et al., 2017).

What to do?

- Modern techniques change the functional itself instead of trying to improve training « from the outside » through penalty terms or better optimizers.

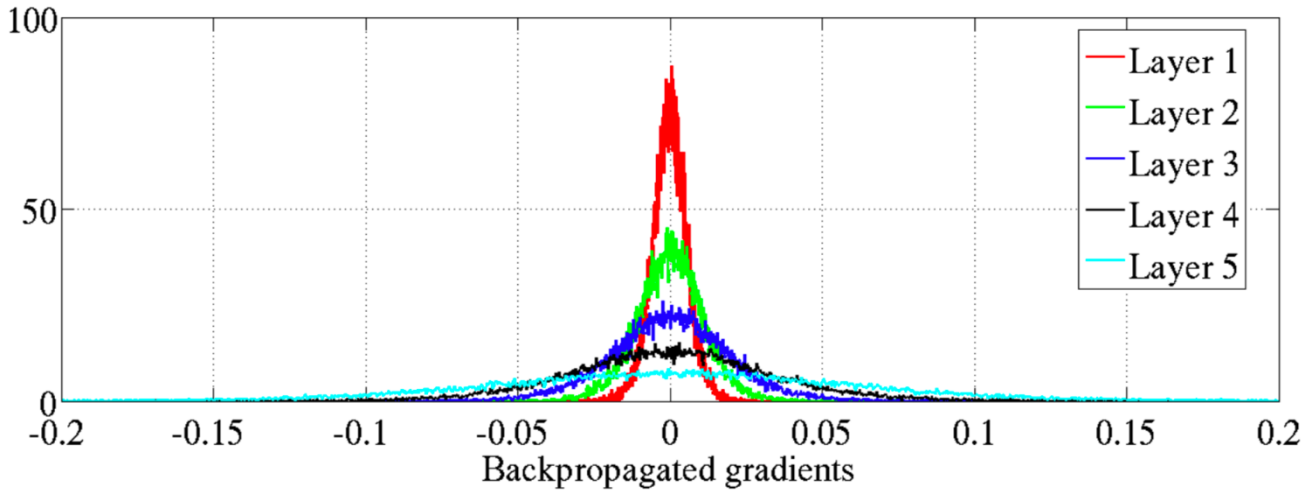
- Our main concern is to make the gradient descent work, even at the cost of engineering substantially the class of functions.

- An additional issue for training very large architectures is the computational cost, which often turns out to be the main practical problem.

Vanishing gradients

Vanishing gradients

- Training deep MLPs with many layers has for long (pre-2011) been very difficult due to the vanishing gradient problem.
  - Small gradients slow down, and eventually block, stochastic gradient descent.
  - This results in a limited capacity of learning.



Backpropagated gradients normalized histograms (Glorot and Bengio, 2010).
Gradients for layers far from the output vanish to zero.

Glorot and Bengio, Understanding the difficulty of training deep feedforward neural networks; AISTAT 2010

Vanishing gradients

- Consider a simplified 3-layer MLP, with $x, w_1, w_2, w_3 \in \mathbb{R}$, such that

$$f(x, w_1, w_2, w_3) = \sigma\Big(w_3\sigma\big(w_2\sigma(w_1 x)\big)\Big)$$

- Under the hood, this would be evaluated as

$$u_1 = w_1 x$$
$$u_2 = \sigma(u_1)$$
$$u_3 = w_2 u_2$$
$$u_4 = \sigma(u_3)$$
$$u_5 = w_3 u_4$$
$$\hat{y} = \sigma(u_5)$$

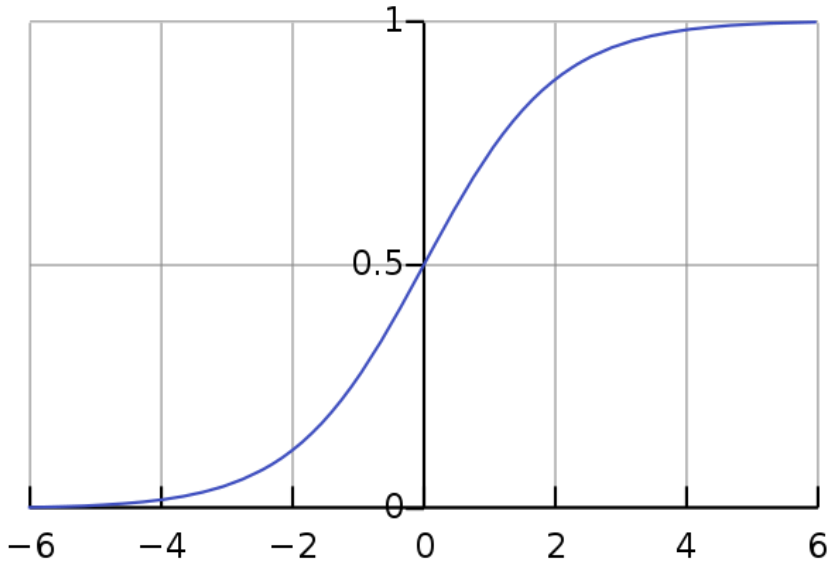
- Its derivative $\frac{d\hat{y}}{dw_1}$ is

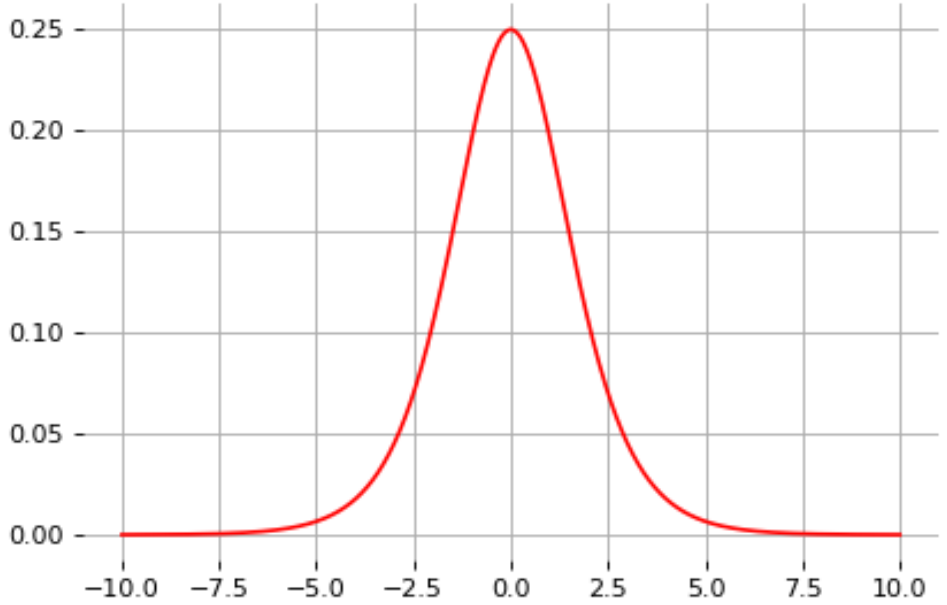
$$\frac{d\hat{y}}{dw_1} = \frac{d\hat{y}}{du_5}\frac{du_5}{du_4}\frac{du_4}{du_3}\frac{du_3}{du_2}\frac{du_2}{du_1}\frac{du_1}{dw_1} = \frac{\partial\sigma(u_5)}{\partial u_5} w_3 \frac{\partial\sigma(u_3)}{\partial u_3} w_2 \frac{\partial\sigma(u_1)}{\partial u_1} x$$

Derivative of the sigmoid

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- $\frac{d\sigma(x)}{dx} = \sigma(x)\big(1-\sigma(x)\big)$
- Hence, we get

$0 \leq \frac{d\sigma(x)}{dx} \leq \frac{1}{4}$ for all x.





Bound on the derivative

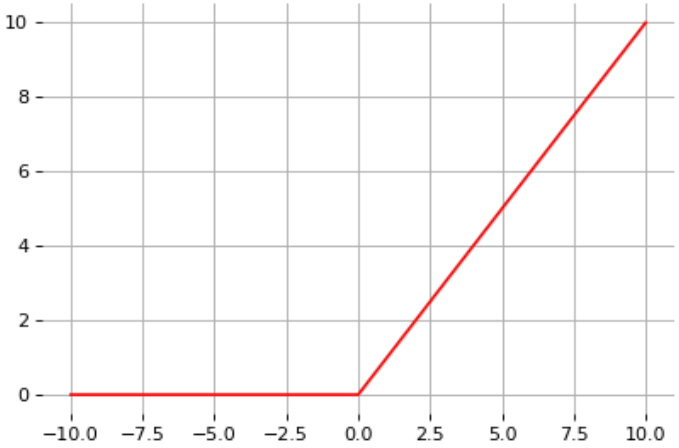
- Assume that weights w_1, w_2, w_3 are initialized randomly from a Gaussian with zero-mean and small variance, such that with high probability $-1 \leq w_i \leq 1$.
- Then,

$$\left|\frac{d\hat{y}}{dw_1}\right| = \left|\frac{\partial\sigma(u_5)}{\partial u_5}\right| |w_3| \left|\frac{\partial\sigma(u_3)}{\partial u_3}\right| |w_2| \left|\frac{\partial\sigma(u_1)}{\partial u_1}\right| |x| \leq \left(\frac{1}{4}\right)^3 |x|$$

- This implies that the gradient $\frac{d\hat{y}}{dw_1}$ exponentially shrinks to zero as the number of layers in the network increases. This is the vanishing gradient problem.
- In general, bounded activation functions (sigmoid, tanh, etc) are prone to the vanishing gradient problem.
- Note also the importance of a proper initialization scheme.

Derivative of the ReLU function

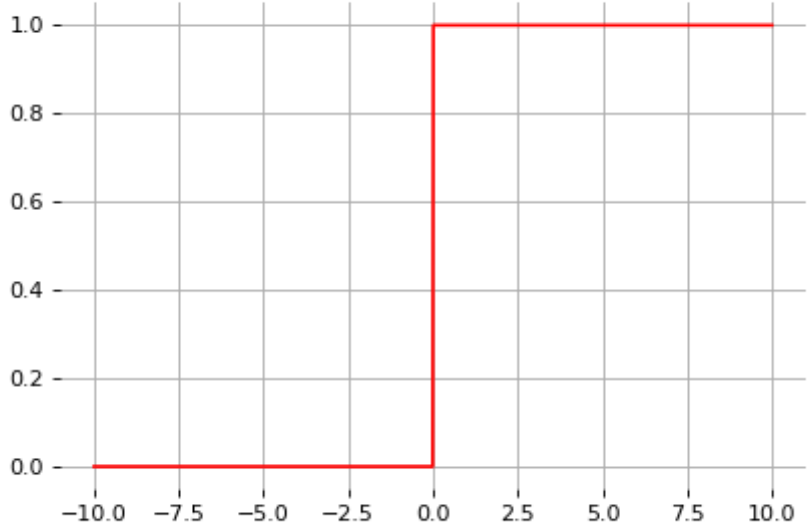
$$\mathrm{ReLU}(x) = \max(0, x)$$



- Note that the derivative of the ReLU function is

$$\frac{d\mathrm{ReLU}(x)}{dx} = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ if } x > 0 \end{cases}$$

- For $x = 0$, the derivative is undefined. In practice, it is set to zero.



Solving the gradient vanishing problem?

- Assume again that weights w_1, w_2, w_3 are initialized randomly from a Gaussian with zero-mean and small variance, such that with high probability $-1 \leq w_i \leq 1$.

- We get

$$\left|\frac{d\hat{y}}{dw_1}\right| = \left|\frac{\partial\sigma(u_5)}{\partial u_5}\right| |w_3| \left|\frac{\partial\sigma(u_3)}{\partial u_3}\right| |w_2| \left|\frac{\partial\sigma(u_1)}{\partial u_1}\right| |x| \leq |x|$$

- This solves the vanishing gradient problem, even for deep networks! (provided proper initialization)

- Note that:
  - The ReLU unit dies when its input is negative, which might block gradient descent.
  - This is actually a useful property to induce sparsity.
  - This issue can also be solved using leaky ReLUs, defined as

$$\text{LeakyReLU}(x) = \max(\alpha x, x)$$

  for small $\alpha > 0$

Excess error

Learning (reminder)

- We observe some samples $\mathcal{D} = (x_i, y_i)_{i=1,\cdots,N}$ following the distribution of (X, Y)
  - We are assuming that the samples are independent (iid assumption)

- We are considering a family $\mathcal{F}$ of functions f_θ parameterized by $\theta \in \Theta$
  - The parameter θ generally belongs to an Euclidean space (e.g., $\theta \in \mathbb{R}^n$)

- We are expecting to solve

$$\theta^* \in \underset{\theta \in \Theta}{\operatorname{argmin}} \, \mathbb{E}\big(L(f_\theta(X), Y)\big)$$

but, in practice, a naïve approach consists in minimizing the empirical risk

$$\hat{\theta} = \hat{\theta}(\mathcal{D}) \in \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i)$$

Regularized Learning (recall)

- Generally, we prefer to solve

$$\hat{\theta} = \hat{\theta}(\mathcal{D}) \in \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i) + \lambda \Omega(f_\theta)$$

where $\Omega(f_\theta)$ is the regularization term and $\lambda \geq 0$ is an hyperparameter

- Examples:
  - Squared ℓ_2–norm: $\Omega(f_\theta) = \Omega(\theta) = \|\theta\|_2^2$
  - Sparsity: $\Omega(f_\theta) = \Omega(\theta) = \|\theta\|_1$

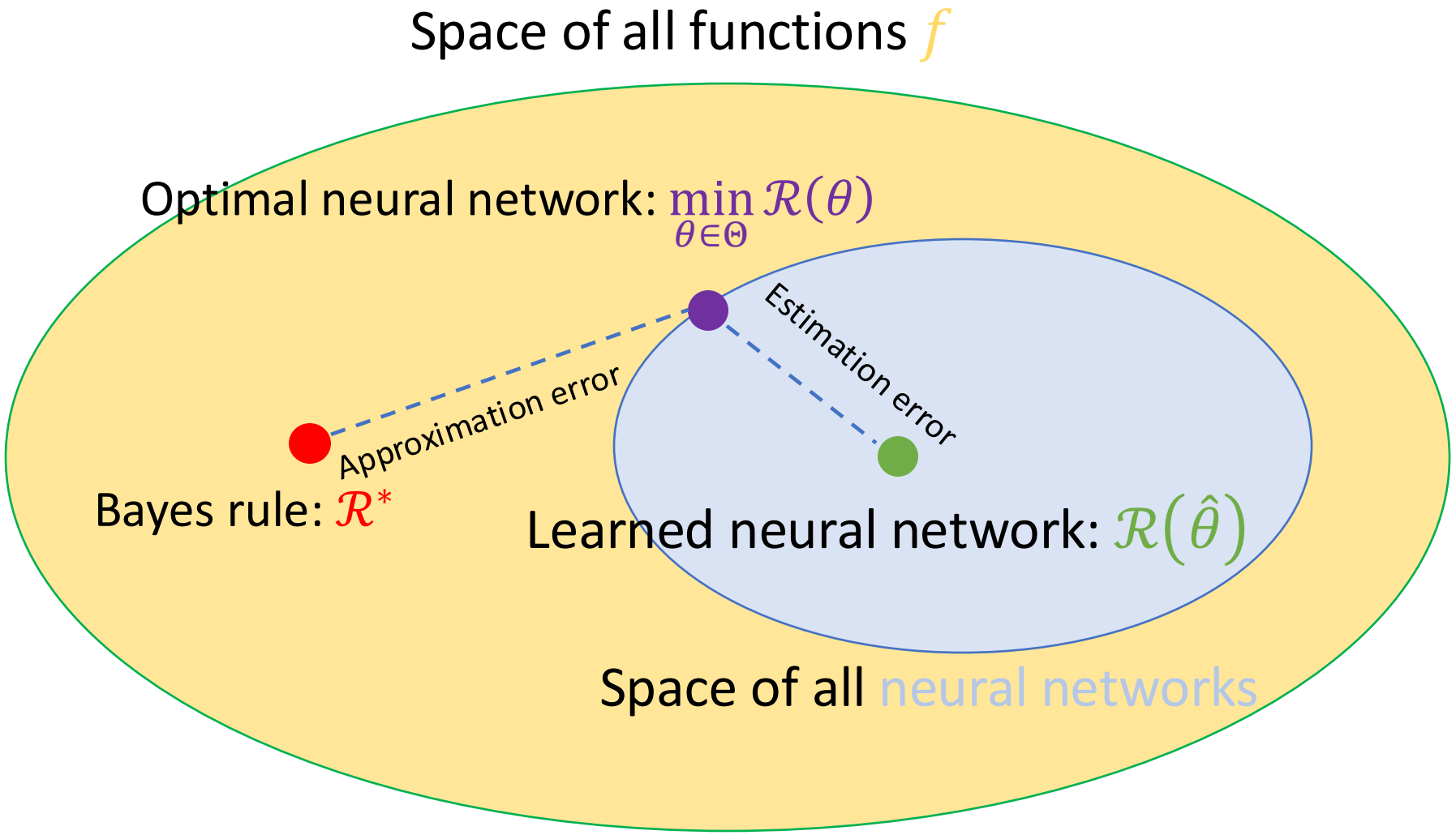
Decomposition of the error

- Let $\mathcal{R}^* = \min_f \mathbb{E}\big(L(f(X), Y)\big)$ the Bayes risk (minimum error)
- Let $\mathcal{R}(\theta) = \mathbb{E}\big(L(f_\theta(X), Y)\big)$ the expected risk at parameter θ for function $f_\theta \in \mathcal{F}$
- Let $\hat{\mathcal{R}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(f_\theta(x_i), y_i)$ the empirical risk at parameter θ
- Let $\hat{\theta} = \hat{\theta}(\mathcal{D}) \in \operatorname{argmin}_\theta \hat{\mathcal{R}}(\theta)$
- We get the famous equality

$$\mathcal{R}(\hat{\theta}) - \mathcal{R}^* = \Big(\mathcal{R}(\hat{\theta}) - \min_{\theta\in\Theta} \mathcal{R}(\theta)\Big) + \Big(\min_{\theta\in\Theta} \mathcal{R}(\theta) - \mathcal{R}^*\Big)$$

Excess error | Estimation error | Approximation error

Decomposition of the error



In practice, decomposition of the error

- Let us assume that our minimization algorithm returns an approximate solution $\tilde{\theta}$ such that

$$\hat{\mathcal{R}}(\tilde{\theta}) \leq \hat{\mathcal{R}}(\hat{\theta}) + \varepsilon$$

where $\varepsilon \geq 0$ is a predefined tolerance (early stopping for example)

- Then, we get the equality

$$\mathcal{E} = \underbrace{\mathcal{R}(\tilde{\theta}) - \mathcal{R}^*}_{\text{Excess error}} = \underbrace{\left(\mathcal{R}(\hat{\theta}) - \min_{\theta \in \Theta} \mathcal{R}(\theta)\right)}_{\text{Estimation error}} + \underbrace{\left(\min_{\theta \in \Theta} \mathcal{R}(\theta) - \mathcal{R}^*\right)}_{\text{Approximation error}} + \underbrace{\left(\mathcal{R}(\tilde{\theta}) - \mathcal{R}(\hat{\theta})\right)}_{\text{Optimization Error}}$$

- The optimization is generally small with respect to the other errors
- Take home message: the computation of the parameters may have an impact of the classifier.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In Advances in Neural Information Processing Systems (NIPS), 2008.

Interpretation

- The approximation error measures how closely functions in $\mathcal{F}$ can approximate the optimal solution f^b.

- The estimation error measures the effect of minimizing the empirical risk $\hat{\mathcal{R}}(\theta)$ instead of the expected risk $\mathcal{R}(\theta)$.
  - The estimation error is determined by the number of training examples and by the capacity of the family of functions (see next slides).

- Large families of functions $\mathcal{F}$ have smaller approximation errors but lead to higher estimation errors.

- This tradeoff has been extensively discussed and lead to excess error that scale between the inverse $\left(\frac{1}{N}\right)$ and the inverse square root $\left(\frac{1}{\sqrt{N}}\right)$ of the number of examples.

Analysis of the Approximation error

- Approximation error:

$$\min_{\theta \in \Theta} \mathcal{R}(\theta) - \mathcal{R}^* = \min_{f_\theta \in \mathcal{F}} \mathbb{E}\big(L(f_\theta(X), Y)\big) - \min_{f} \mathbb{E}\big(L(f(X), Y)\big)$$

- Main properties:
  - This error is non-random
  - It depends on the richness of $\mathcal{F}$
  - If the optimal classifier belongs to $\mathcal{F}$, this error vanishes
  - This error is related to the functional analysis (approximation of function)

Analysis of the Estimation error

- Warning:
  - $\hat{\theta}$ and $\hat{\mathcal{R}}$ are random so all the terms depending on them (or just one of them) are random

- We have

$$\mathcal{R}(\hat{\theta}) - \mathcal{R}(\theta^*) = \mathcal{R}(\hat{\theta}) - \hat{\mathcal{R}}(\hat{\theta}) + \hat{\mathcal{R}}(\hat{\theta}) - \mathcal{R}(\theta^*)$$

- Note that $\hat{\mathcal{R}}(\hat{\theta}) \leq \hat{\mathcal{R}}(\theta^*)$. Hence,

$$\mathcal{R}(\hat{\theta}) - \mathcal{R}(\theta^*) \leq \mathcal{R}(\hat{\theta}) - \hat{\mathcal{R}}(\hat{\theta}) + \hat{\mathcal{R}}(\theta^*) - \mathcal{R}(\theta^*) \leq \mathcal{R}(\hat{\theta}) - \hat{\mathcal{R}}(\hat{\theta}) + \sup_{\theta \in \Theta} \left| \mathcal{R}(\theta) - \hat{\mathcal{R}}(\theta) \right|$$

- It follows that

$$\mathcal{R}(\hat{\theta}) - \mathcal{R}(\theta^*) \leq 2 \sup_{\theta \in \Theta} \left| \mathcal{R}(\theta) - \hat{\mathcal{R}}(\theta) \right|$$

Generalization error

- The term $\left|\mathcal{R}(\theta) - \hat{\mathcal{R}}(\theta)\right|$ is called the generalization error: it is the difference between the expected loss and the empirical loss

- Affine classifier: $f_\theta(x) = \theta^T x = \sum_{i=1}^{n} \theta_i x_i$
  - $\left|\mathcal{R}(\theta) - \hat{\mathcal{R}}(\theta)\right| \approx \sqrt{\frac{n}{N}}$

- One-layer ReLU networks with regularization: $f_\theta(x) = \sum_{i=1}^{q} v_i \max\left(0, w_i^T x + b_i\right)$
  - $\left|\mathcal{R}(\theta) - \hat{\mathcal{R}}(\theta)\right| \approx q\sqrt{\frac{n}{N}}$ where q is the number of hidden neurons

Growth function, VC-dimension, shattering

- Let $\mathcal{F}$ denote a class of functions from $\mathcal{X}$ to $\{0,1\}$ (the classification rules).
- For any non-negative integer m, we define the growth function of $\mathcal{F}$ as

$$\Pi_{\mathcal{F}}(m) = \max_{x_1,\dots,x_m \in \mathcal{X}} \left|\left\{\left(f(x_1),\dots,f(x_m)\right): f \in \mathcal{F}\right\}\right|$$

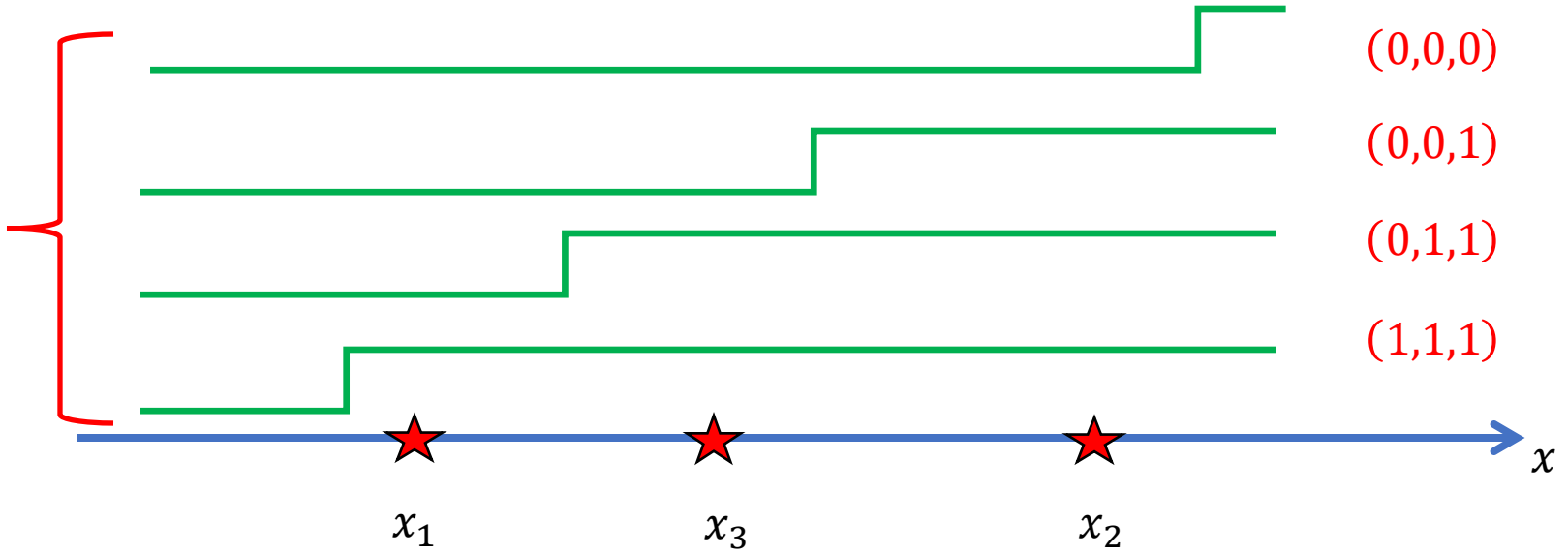
  - Example: see the next slide!

- If $\left|\left\{\left(f(x_1),\dots,f(x_m)\right): f \in \mathcal{F}\right\}\right| = 2^m$, we say $\mathcal{F}$ shatters the set $\{x_1,\dots,x_m\}$.
  - Example: see the next slide!

- The Vapnik-Chervonenkis dimension of $\mathcal{F}$ is denoted $\mathrm{VCdim}(\mathcal{F})$
  - It is the size of the largest shattered set, i.e. the largest m such that $\Pi_{\mathcal{F}}(m) = 2^m$.
  - If there is no largest m, we define $\mathrm{VCdim}(\mathcal{F}) = \infty$.

Example of growth function

4 possible classifications based on $\mathcal{F}$

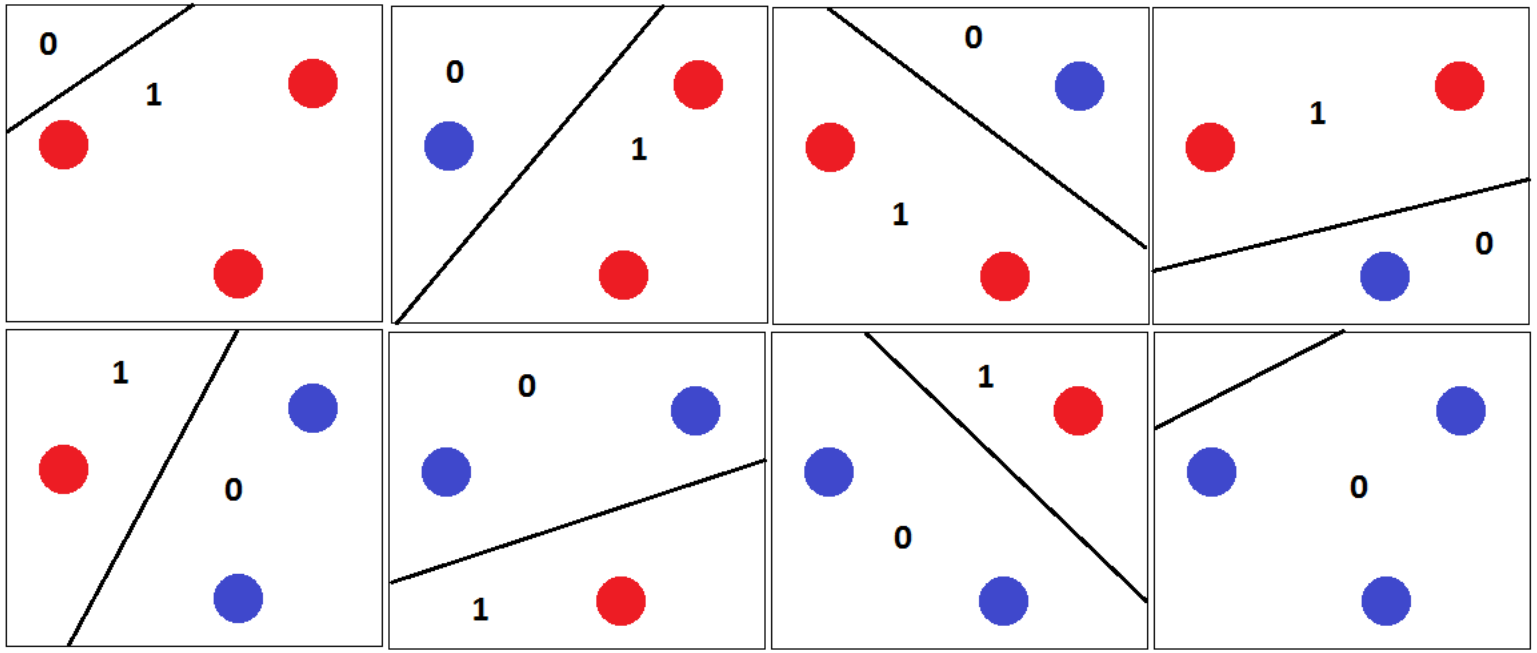


- Set of step classifiers (threshold clasifier): $\mathcal{F} = \{f \in \{0,1\}: f(x) = 1 \Leftrightarrow x > a,\ a \in \mathbb{R}\}$
- Classification results:
  - $\{(f(x_1), f(x_2), f(x_3)): f \in \mathcal{F}\} = \{(0,0,0), (0,0,1), (0,1,1), (1,1,1)\}$
  - $|\{(f(x_1), f(x_2), f(x_3)): f \in \mathcal{F}\}| = 4 < 2^3 = 8$
- $\mathcal{F}$ **does not shatter** the set $\{x_1, x_2, x_3\}$
  - It is impossible to get the classification result $(1,0,1)$ with a function f in $\mathcal{F}$
- $\mathcal{F}$ **does not shatter** the set $\{x_1, x_2\}$
- $\mathcal{F}$ **shatters** the set $\{x_1\}$ whatever x_1 is. Hence $\text{VCdim}(\mathcal{F}) = 1$.

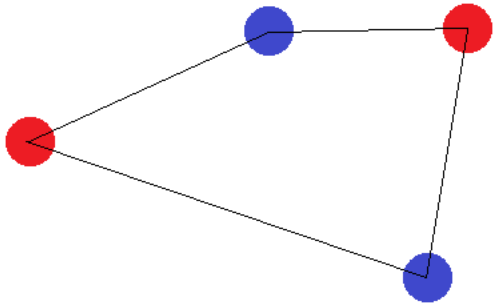
Example of VC dimension

- For an input space with two variables and a linear classifier, the VC dimension is 3.

A linear classifier can always shatter 3 points



A linear classifier may not shatter 4 points



PAC Bound

- In case of binary classfication, we have for example a more precise probabilitics behavior: the « Probably and Approximately Correct » (PAC) bound

- For any set $\mathcal{F}$ with $\mathrm{VCdim}(\mathcal{F}) = d$, with probability $1 - \delta$ over the training dataset with N samples, we have

$$\mathcal{R}(\theta) \leq \hat{\mathcal{R}}(\theta) + O\left(\sqrt{\frac{d}{N}\log\left(\frac{N}{d}\right) - \frac{1}{N}\log(\delta)}\right)$$

Finite-sample expressivity

- As soon as the number of parameters of a networks is greater than N, even simple two-layer neural networks can represent any function of the input sample.

- We say that a neural network $f_\theta(x)$ can represent any function of a sample of size N in n dimensions if for every sample $S \subset (\mathbb{R}^n)^N$ with $|S| = N$ and every function $f: S \to \mathbb{R}$, there exists a setting of the weights θ of $f_\theta(x)$ such that $f_\theta(x) = f(x)$ for every $x \in S$.

Theorem (Zhang, 2016)

- There exists a two-layer neural network with ReLU activations and $2N + n$ weights that can represent any function on a sample of size N in n dimensions.

Corollary

- For every $k \geq 2$, there exists a neural network with ReLU activations of depth k, width $O\left(\frac{N}{k}\right)$ and $O(N + n)$ weights that can represent any function on a sample of size N in n dimensions.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B. & Vinyals, O. Understanding deep learning requires rethinking generalization, 2016.

Conclusion

Conclusion

- Neural networks with several layers provide high capability for approximating multivariate functions
- Depth leads to vanishing gradient, which is an important issue
- Activation functions play an important role
- Theoretical results with neural network representations are in progress.