Neural Networks Part 2

Jim Rehg

Based on slides prepared by Dr. Fuxin Li, Oregon State Univ.

With materials from Zsolt Kira, Roger Grosse, Nitish Srivastava, Michael Nielsen

XOR problem and linear classifier

- 4 points: X = [(-1,-1), (-1,1), (1,-1), (1,1)]
- Y=[-1 1 1 -1]
- Try using binomial log-likelihood loss:

$\min_{w} \sum_{i}^{▒} \log(1+e^{2 y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b)})$

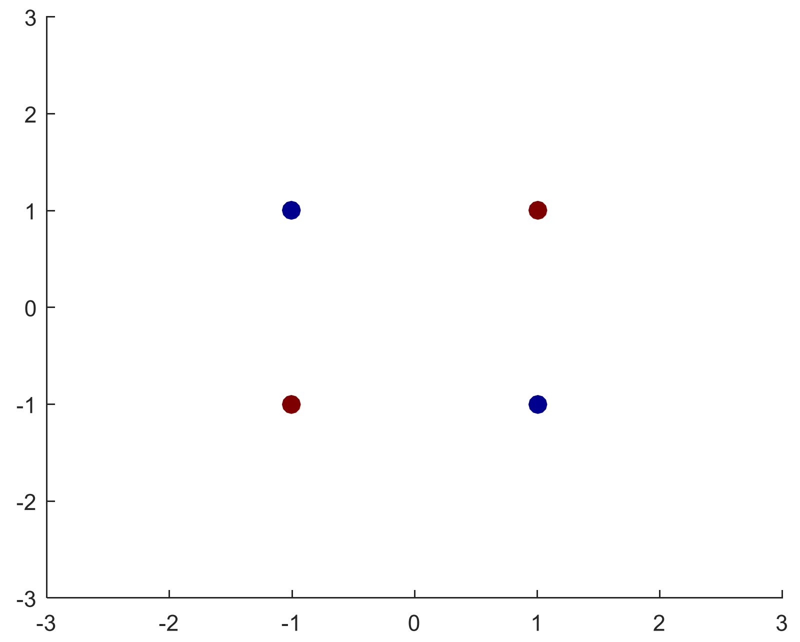
$\min_{w} \sum_{i}^{▒} \log(1+e^{2 y_i (

- Gradient:

$\nabla \boldsymbol{w} = \sum_{i}^{▒} 2 y_i e^{2 y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b)} /$

$\nabla \boldsymbol{w} = \sum_{i}^{▒} 2 y_i e^{2 y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b)} /$

$\nabla b = \sum_{i}^{▒} 2 y_i e^{2 y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b)} / 1 + e^{2 y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b)}$



Try $\boldsymbol{w}=0, b=0$, what

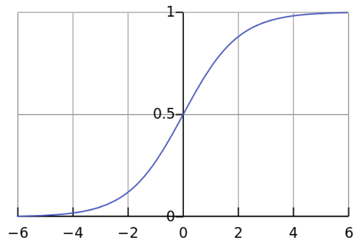
Try $\boldsymbol{w}=0, b=0$, what

With 1 hidden layer

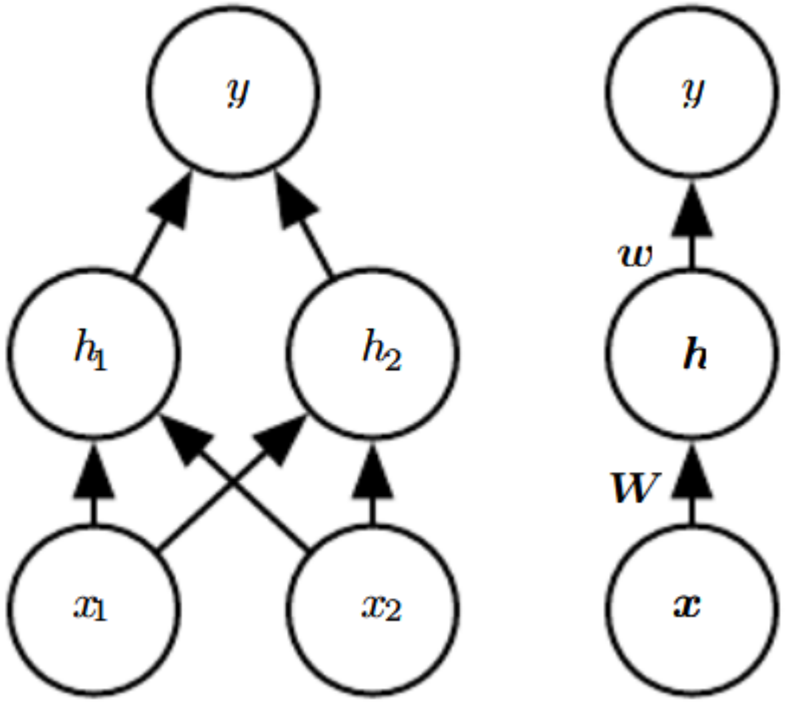
- A hidden layer makes a nonlinear classifier

  $f(x)=\boldsymbol{w}^{\top} g(\boldsymbol{W}^{\top} \boldsymbol{x}+\boldsymbol{c})+b$

  - g needs to be nonlinear
  - Sigmoid: $\mathrm{Sigm}(x)=1/(1+e^{x})$



  - RELU: g(x) = max(0,x)



Taking gradient

$$\min_{W,w} E(f) = \sum_{i} L(f(x_i), y_i)$$

$$f(x) = \boldsymbol{w}^\top g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}) + b$$

- What is $\partial E / \partial W$?
- Consider chain rule: $dz/dx = dz/dy \; dy/dx$

Note: Vectorized Computations

On the left are the computations performed by a network. Write them in terms of matrix and vector operations. Let $\sigma(\mathbf{v})$ denote the logistic sigmoid function applied elementwise to a vector $\mathbf{v}$. Let $\mathbf{W}$ be a matrix where the (i, j) entry is the weight from visible unit j to hidden unit i.

$$z_i = \sum_j w_{ij} x_j$$

$$h_i = \sigma(z_i)$$

$$y = \sum_i v_i h_i$$

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$y = \mathbf{v}^T \mathbf{h}$$

Backpropagation

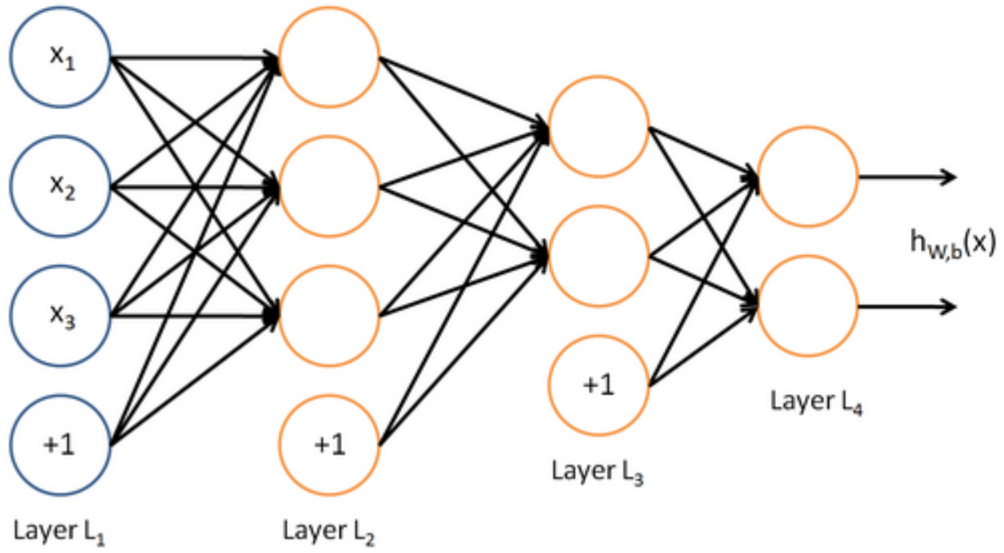
- Save the gradients and the gradient products that have already been computed to avoid computing multiple times
- In a multiple layer network:
  - (Ignore constant terms)

$$f(x)=\boldsymbol{w}_{\boldsymbol{n}}^{\top}\, g(\boldsymbol{W}_{\boldsymbol{n-1}}^{\top}\, \boldsymbol{g}(\boldsymbol{W}_{\boldsymbol{n-2}}^{\top}\, \boldsymbol{g}(\ldots(\boldsymbol{W}_{\boldsymbol{1}}^{\top}\, \boldsymbol{g}(\boldsymbol{x})))))$$

$$\frac{\partial E}{\partial \boldsymbol{W}_k} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial f_k}\, g(f_{k-1}(x))$$

$$= \frac{\partial E}{\partial f_{k+1}}\frac{\partial f_{k+1}}{\partial f_k}\, g(f_{k-1}(x))$$

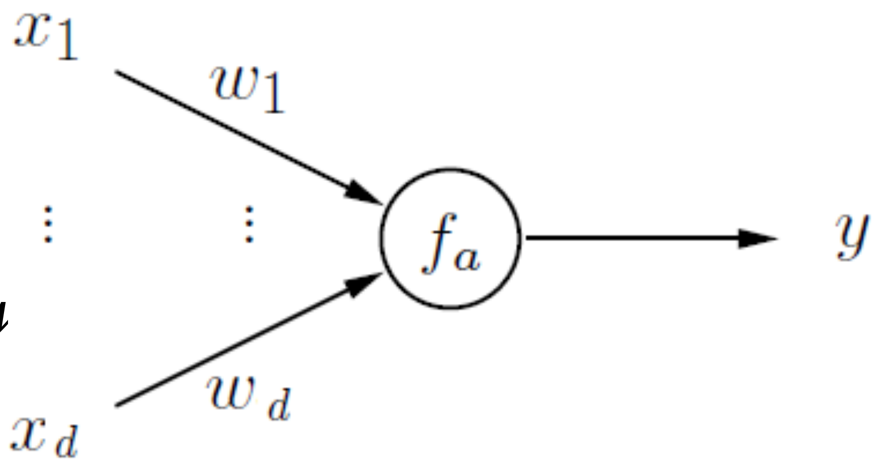
$$f_k(x)=w_k^{\top}\, g(f_{k-1}(x)), f_0(x)=x$$



Modules

- Each layer can be seen as a module
- Given input, return
  - Output $f_a(x)$
  - Network gradient $\partial f_a / \partial x$
  - Gradient of module parameters $\partial f_a / \partial w$
- During backprop, propagate/update
  - Backpropagated gradient

$$\partial E / \partial f_a$$

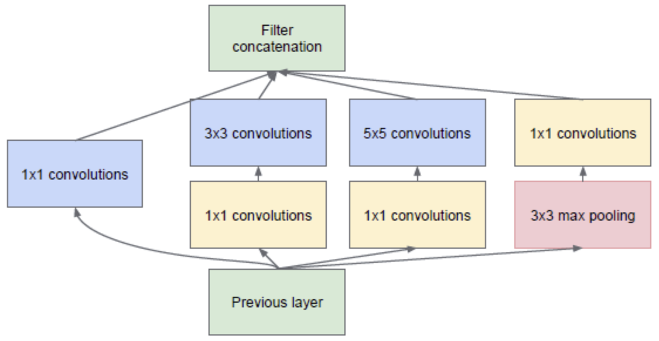


$$\partial E / \partial \boldsymbol{W}_k = \partial E / \partial f \;\; \partial f / \partial f_k \;\; g(f_{k-1}(x))$$
$$= \partial E / \partial f_{k+1} \;\; \partial f_{k+1} / \partial f_k \;\; g(f_{k-1}(x))$$

Different DAG structures

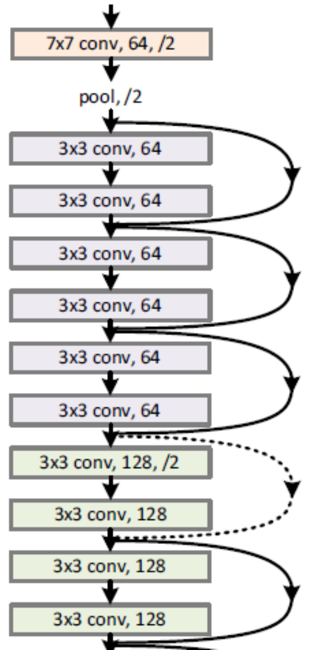
- The backpropation algorithm would work for any DAGs
- So one can imagine different architectures than the plain layerwise one

Inception

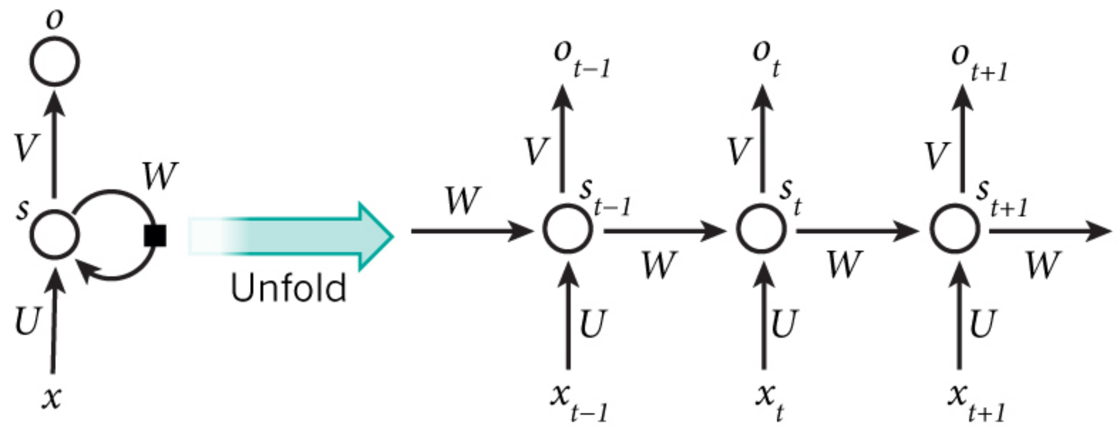


(b) Inception module with dimension reductions

Residual



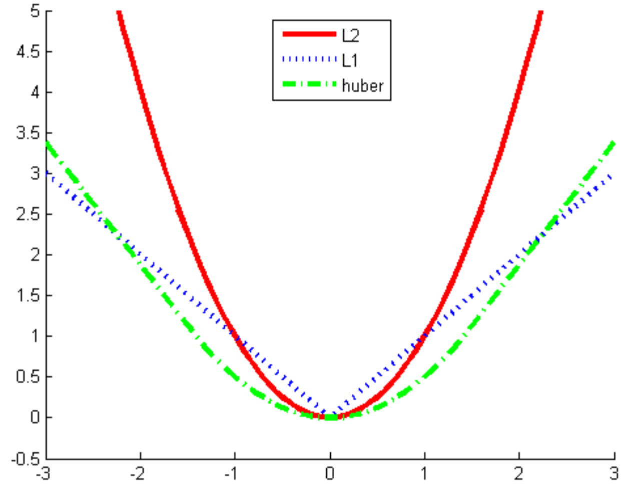
RNN



Loss functions

- Regression:
  - Least squares $L(f)=(f(x)-y)\uparrow 2$
  - L1 loss $L(f)=|f(x)-y|$
  - Huber loss $L(f)=\{■1/2\ (f(x)-y)\uparrow 2\ ,|f(x)-y|<\delta\delta($ otherwise

- Binary Classification
  - Hinge loss $L(f)=\max(1-yf(x),0)$
  - Binomial log-likelihood$L(f)=\ln(1+\exp(-2yf(x)))$
  - Cross-entropy $L(f)=y\uparrow * \ \ln\mathrm{sigm}(f)+(1-y\uparrow * \ )\ln(1-\mathrm{sigm}(f))$,
    - $y\uparrow * =(y+1)/2$



Multi-class: Softmax layer

- Multi-class logistic loss function

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$

- Log-likelihood:
  - Loss function is minus log-likelihood

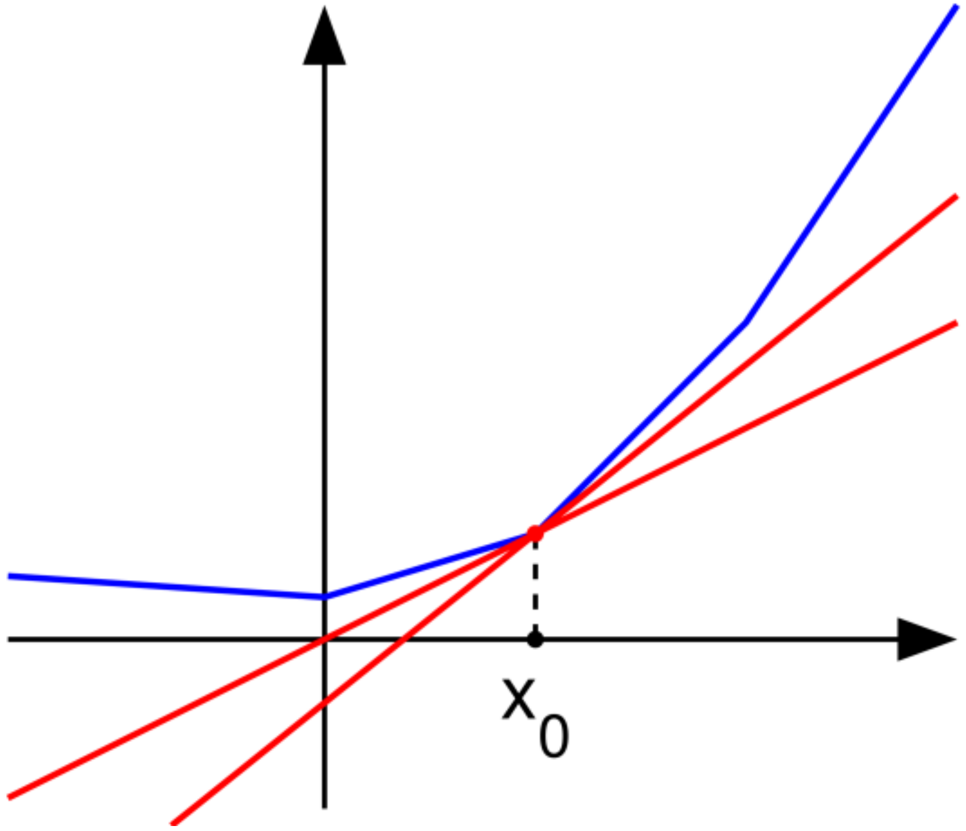
$$-\log P(y=j|x) = -\boldsymbol{x}^\top \boldsymbol{w}_j + \log \sum_k e^{\boldsymbol{x}^\top \boldsymbol{w}_k}$$

Subgradients

- What if the function is non-differentiable?
- Subgradients:
  - For **convex** $f(x)$ at x_0 :
  - If for any x,

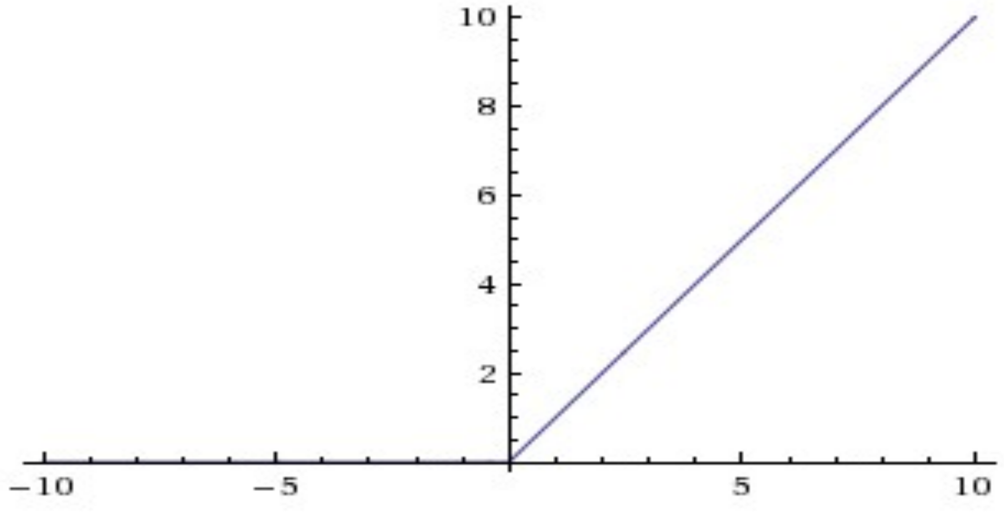
    $$f(y) \geq f(x) + g^\top (y - x)$$

  - g is called a subgradient
- Subdifferential: ∂f: set of all subgradients
- Optimality condition: $0 \in \partial f$



The RELU unit

- f(x) = max(x,0)
- Convex
- Non-differentiable
- Subgradient:$df/dx = \{$■$1, x > 0 [0$



Subgradient descent

- Similar to gradient descent

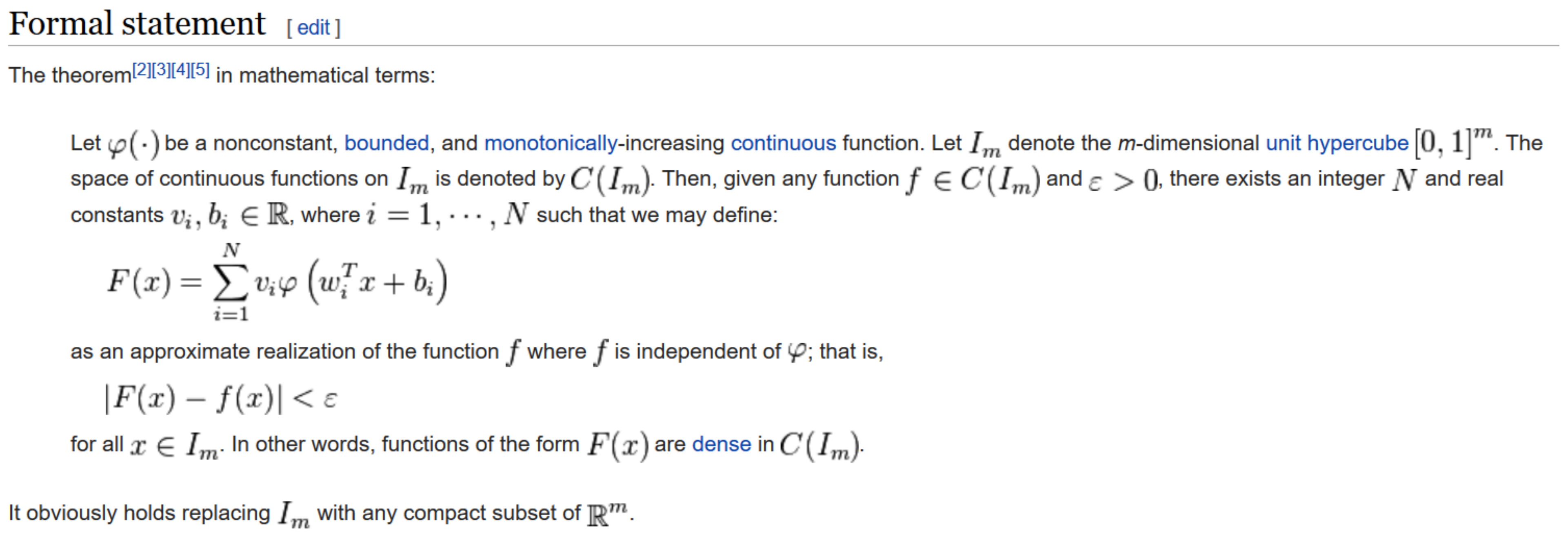
$$x^{(k+1)} = x^{(k)} - \alpha_k g^{(k)}$$

- Step size rules:
  - Constant step size: $\alpha_k = \alpha.$
  - Square summable:
    $$\alpha_k \geq 0, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \qquad \sum_{k=1}^{\infty} \alpha_k = \infty.$$

  - Usually, a large constant that drops slowly after a long while
    - e.g. $100/100+k$

Universal Approximation Theorems

- Many universal approximation theorems proved in the 90s
- Simple statement: for every continuous function, there exist a function that can be approximated by a 1-hidden layer neural network with arbitrarily high precision



Formal statement [edit]

The theorem[2][3][4][5] in mathematical terms:

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exists an integer N and real constants $v_i, b_i \in \mathbb{R}$, where $i = 1, \cdots, N$ such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

It obviously holds replacing I_m with any compact subset of $\mathbb{R}^m$.

Universal Approximation Theorems

- The approximation does not need many units if the function is kinda nice. Let

$$C_f = \int_{\mathbf{R}^d} |\omega| |\hat{f}(\omega)| \, d\omega$$

- Then for a 1-hidden layer neural network with n hidden nodes, we have for a finite ball with radius r,

$$\int_{B_r} (f(x) - f_n(x))^2 \, d\mu(x) \leq 4 r^2 C_f^2 / n$$