3. Basic Feedforward Network

CS 519: Deep Learning, Winter 2017

Fuxin Li

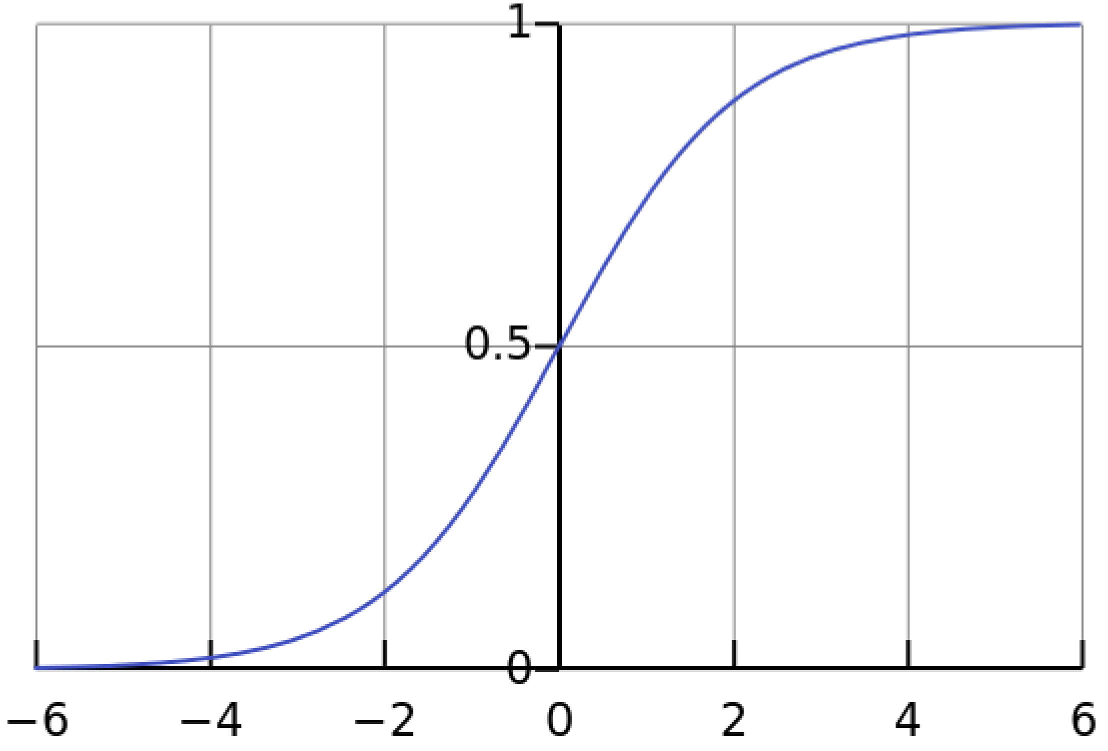
With materials from Zsolt Kira, Roger Grosse, Nitish Srivastava, Michael Nielsen

Linear Classifier and the Perceptron Algorithm

- $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$
- σ: Sigmoid function:$\sigma(x) = \frac{1}{1+e^{-x}}$
- The connection to logistic regression:
  - Assume binomial distribution with parameter $\hat{p}$
  - Assume the logit transform is linear:

$$\log \frac{\hat{p}}{1-\hat{p}} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

$$\Rightarrow \quad \hat{p} = \sigma(f(\boldsymbol{x}))$$



Maximum Log-Likelihood

- MLE of the binomial likelihood:

$$\sum_{i=1}^{n} \mathrm{y_i^*} \log \hat{p} + (1 - y_i^*) \log(1 - \hat{p})$$

  - where $y_i^* \in \{0,1\} = \frac{1+y_i}{2}$

$$\log \hat{p} = -\log(\,1 + e^{-f(x)})$$

$$\log(1 - \hat{p}) = -\log(\,1 + e^{f(x)})$$

$$\mathrm{y_i^*} \log \hat{p} + (1 - y_i^*) \log(1 - \hat{p}) = -\log(1 + e^{-yf(x)})$$

Gradient descent optimization

- Optimize $\boldsymbol{w}, b$ with gradient descent

$$\min_{\boldsymbol{w},b} \sum_i \log(1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)})$$

$$\nabla \boldsymbol{w} = \sum_i \frac{-y_i e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)}}{1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)}} \boldsymbol{x_i}$$

$$= \sum_i -y_i^* \big(1 - \hat{p}(\boldsymbol{x}_i)\big) - (1 - y_i^*)\hat{p}(\boldsymbol{x_i})\boldsymbol{x}_i$$

$$\nabla b = \sum_i \frac{-y_i e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)}}{1 + e^{-y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)}}$$

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:

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

- Gradient:

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

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

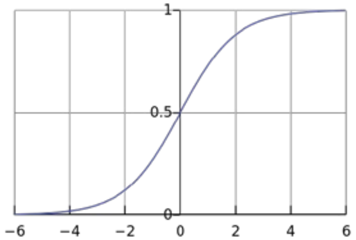
Try $\boldsymbol{w} = 0, b = 0$, what do you see?

With 1 hidden layer

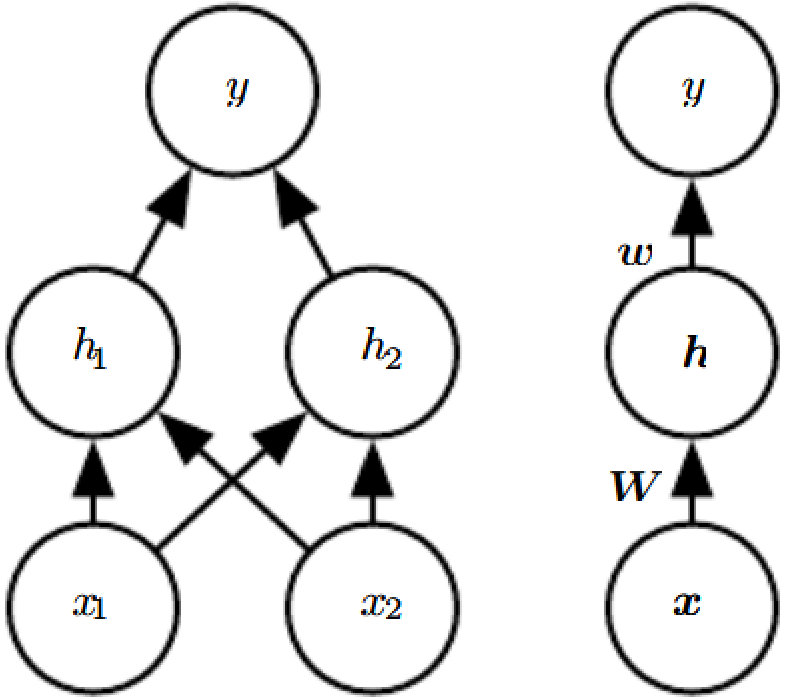
- A hidden layer makes a nonlinear classifier

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

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



  - 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 $\frac{\partial E}{\partial W}$?
- Consider chain rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

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.

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

$$h_i = \sigma(z_i)$$

$$y = \sum_i v_i h_i$$

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

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

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

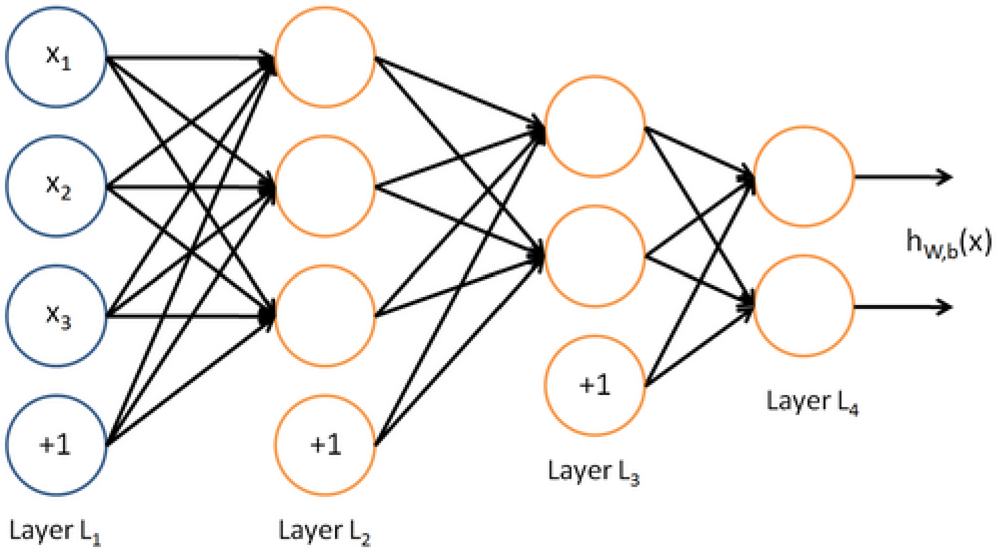
Backpropagation

- 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)

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

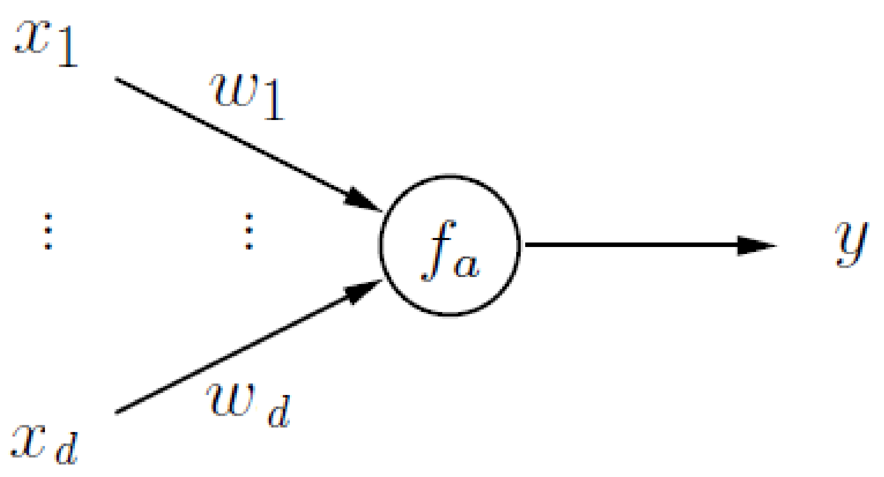
$$\frac{\partial E}{\partial \boldsymbol{W}_k} = \frac{\partial E}{\partial f_k} g\left(f_{k-1}(x)\right)$$
$$= \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_k} g(f_{k-1}(x))$$

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



Modules

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

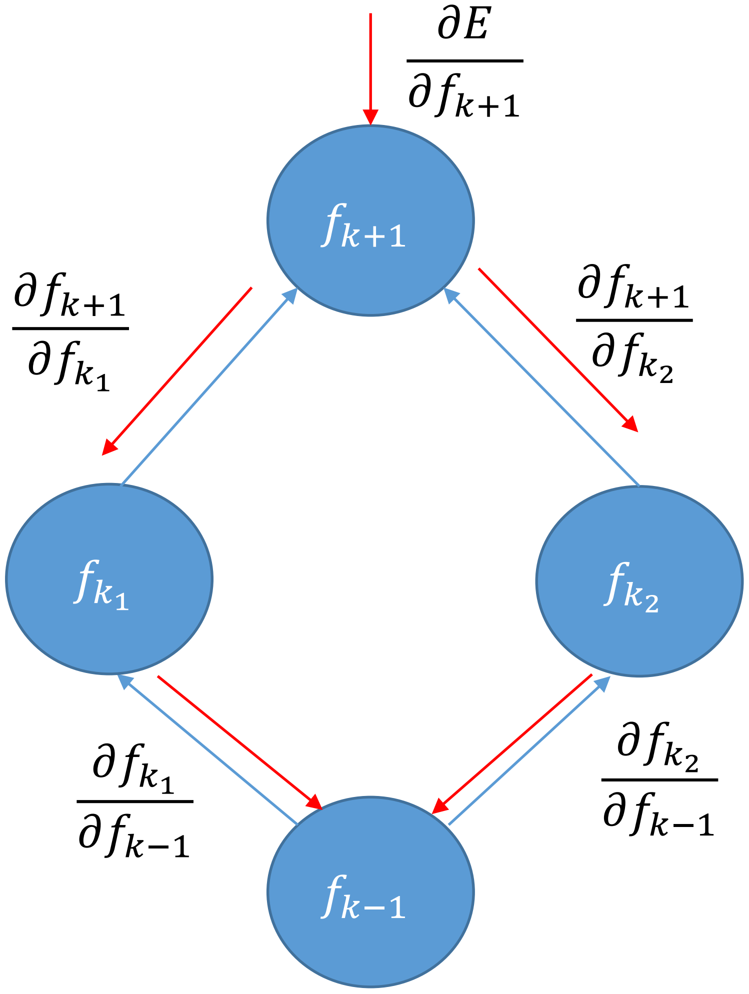


$$\frac{\partial E}{\partial \boldsymbol{W}_k} = \frac{\partial E}{\partial f_k} g\big(f_{k-1}(x)\big) = \underbrace{\frac{\partial E}{\partial f_{k+1}}}_{\text{Backprop signal}} \underbrace{\frac{\partial f_{k+1}}{\partial f_k}}_{\text{Network gradient}} g(\underbrace{f_{k-1}(x)}_{\text{Gradient of parameters}})$$

Note: $\frac{\partial E}{\partial f_k} = \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_k}$

Multiple Inputs and Multiple Outputs

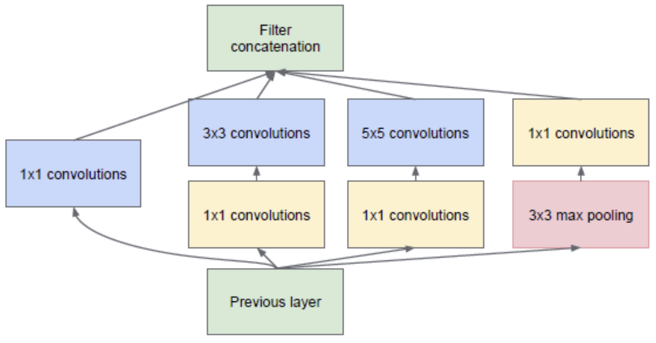
- $$\frac{\partial E}{\partial f_{k-1}} = \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_{k_1}} \frac{\partial f_{k_1}}{\partial f_{k-1}} + \frac{\partial E}{\partial f_{k+1}} \frac{\partial f_{k+1}}{\partial f_{k_2}} \frac{\partial f_{k_2}}{\partial f_{k-1}}$$



Different DAG structures

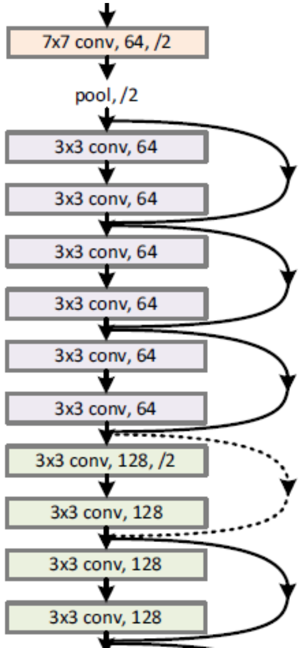
- The backpropation algorithm would work for any DAGs
- So one can imagine different architectures than the plain layerwise one

Inception

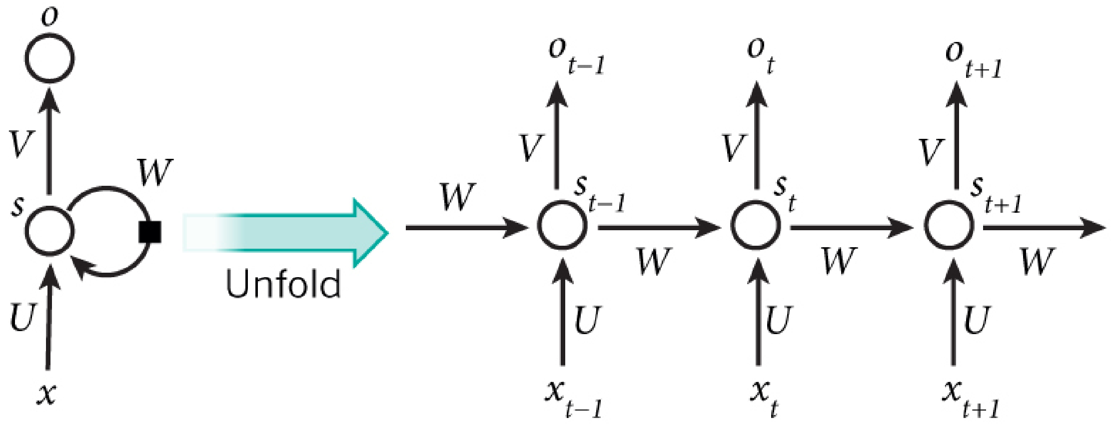


(b) Inception module with dimension reductions

Residual

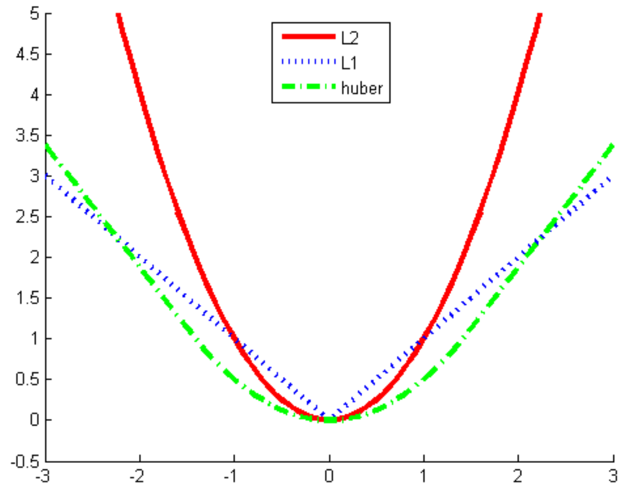


RNN



Loss functions

- Regression:
  - Least squares $L(f) = (f(x) - y)^2$
  - L1 loss $L(f) = |f(x) - y|$
  - Huber loss $L(f) = \begin{cases} \frac{1}{2}(f(x) - y)^2, |f(x) - y| < \delta \\ \delta\left(|f(x) - y| - \frac{1}{2}\delta\right), \text{otherwise} \end{cases}$



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

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

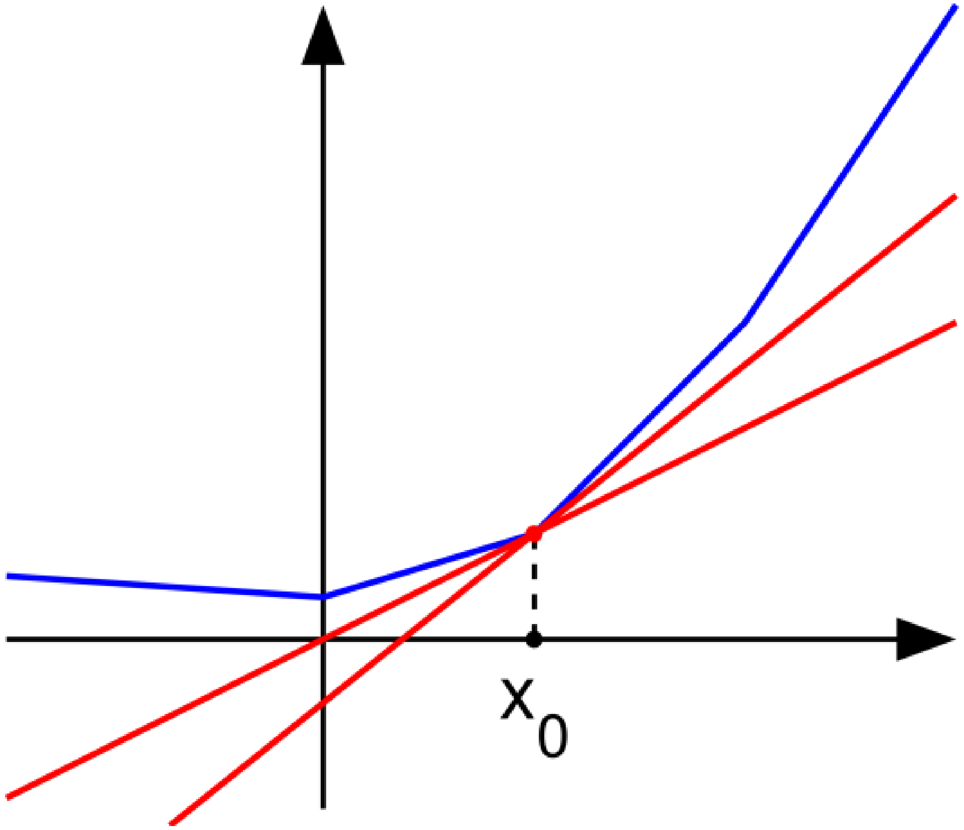
$$-\log P(y = j|x) = -\boldsymbol{x}^\top \boldsymbol{w}_j + \log \sum_k e^{\boldsymbol{x}^\top \boldsymbol{w}_k}$$

Subgradients

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

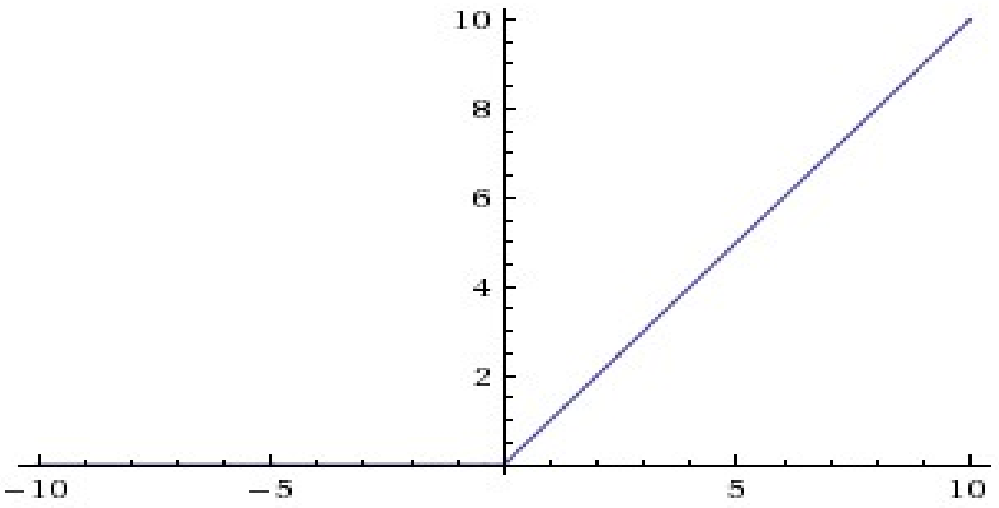
$$f(y) \geq f(x) + g^{\top}(y - x)$$

  - 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: $\frac{df}{dx} = \begin{cases} 1, x > 0 \\ [0,1], x = 0 \\ 0, x < 0 \end{cases}$



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. $\frac{100}{100+k}$

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

$$C_f = \int_{\mathbf{R}_d} ||\omega|||\tilde{f}(\omega)|d\omega$$

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

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