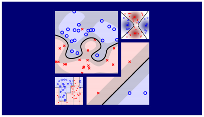
Machine Learning Soundings
(機器學習深測)



Lecture 2: Activation and Initialization

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. Deep Learning Foundations

Lecture 1: Neural Network

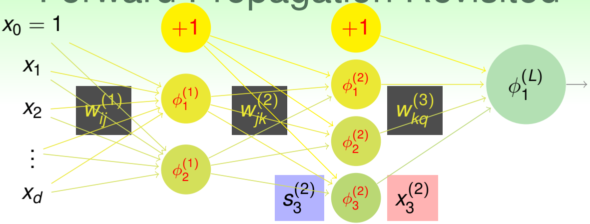
automatic **pattern feature extraction** from **layers of neurons** with **backprop** for GD/SGD

Lecture 2: Activation and Initialization

- Traditional Neural Networks Revisited
- Modern Activation Functions
- Initialization w.r.t. Activation

2. Deep Learning Models

Forward Propagation Revisited



$d^{(0)}$-$d^{(1)}$-$d^{(2)}$-$\cdots$-$d^{(L)}$ Neural Network (NNet)

$$w_{ij}^{(\ell)} \quad : \quad \left(1 + d^{(\ell-1)}\right) \times \left(d^{(\ell)}\right) \text{ matrix before layer } \ell$$

$$\text{score } s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)}, \text{ transformed } x_j^{(\ell)} = \phi_j^{(\ell)}\left(s_j^{(\ell)}\right)$$

$\phi_j^{(\ell)}$: transformation (activation) function, e.g.

$\phi_1^{(L)} = /$ for regression;

$\phi_j^{(\ell)} = \tanh$ for traditional NNet

Backpropagation Revisited

$$
\begin{aligned}
\delta_j^{(\ell)} &= \frac{\partial e_n}{\partial s_j^{(\ell)}} \\
&= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\
&= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\phi'\left(s_j^{(\ell)}\right)\right) \\
&= \sum_k \left(\sum_m \left(\delta_m^{(\ell+2)}\right) \left(w_{km}^{(\ell+2)}\right) \phi'\left(s_k^{(\ell+1)}\right)\right) \left(w_{jk}^{(\ell+1)}\right) \left(\phi'\left(s_j^{(\ell)}\right)\right)
\end{aligned}
$$

$\delta_i^{(1)}$: $\delta_1^{(L)}$ multiplied by many $w_{??}^{(\ell+1)}$ and $\phi'(s_?^{(\ell)})$ for $\ell = 1, 2, \ldots, L-1$

Backpropagation Revisited

$$
\begin{aligned}
\delta_j^{(\ell)} &= \frac{\partial e_n}{\partial s_j^{(\ell)}} \\
&= \sum_{k=1}^{d^{(\ell+1)}} \underbrace{\frac{\partial e_n}{\partial s_k^{(\ell+1)}}}_{\delta_k^{(\ell+1)}} \underbrace{\frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}}}_{w_{jk}^{(\ell+1)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\
&= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\phi'\left(s_j^{(\ell)}\right)\right)
\end{aligned}
$$

$\delta_i^{(1)}$: $\delta_1^{(L)}$ multiplied by many $w_{??}^{(\ell+1)}$ and $\phi'(s_?^{(\ell)})$ for $\ell = 1, 2, \ldots, L-1$

The Vanishing Gradient Issue

gradient $\nabla_{ij}^{(\ell)} = x_i^{(\ell-1)} \cdot \delta_j^{(\ell)} = x_i^{(\ell-1)} \sum\sum\sum \delta_1^{(L)} www\phi'\phi'\phi'$

when $\phi = \tanh \Rightarrow x_i^{(\ell-1)} \in (-1, 1)$

- $\phi(s) \to \pm 1$ when $s \to \pm\infty$: saturation
- $\phi'(s) = 1 - \tanh^2(s)$
  - $\to 0$ when $s \to \pm\infty$
- vanishing gradient: $w_{??}^{(\ell)}$ too big/small $\Rightarrow |s|$ too big/small $\Rightarrow \phi'(s)$ too small $\Rightarrow \delta_?^{(1)}$ too small $\Rightarrow \nabla_{??}^{(1)}$ too small

vanishing gradient: early weights not updated
$\Rightarrow$ cannot train 'deep' network

Possible Cures for Vanishing Gradient

- better-behaved network
  - skip connection (escape some $w\phi'$)
- better-behaved weights
  - small-random initialization (next in Lecture 302)
- better-behaved network + better-behaved weights
  - layer-wise pre-training (see MLTech Lecture 213)
- better-behaved (hidden) inputs
  - internal normalization (scale $x_j^{(\ell)}$)
- better-behaved update direction
  - gradient normalization (next in Lecture 303)
- better-behaved activation functions
  - next :-)

vanishing/varying gradients: difficulty of deep learning optimization

Rectified Linear Unit

$$\phi(s) = \max(s, 0)$$

- Rectified Linear Unit (ReLU):
  $\phi(s) = s$ for $s > 0$, 0 for $s = 0$, 0 for $s < 0$
  —continuous
- $\phi'(s) = 1$ for $s > 0$, 0 for $s < 0$
  —**less gradient vanishing** ($\approx$ 'half' of the time)
- $\phi'(s) =$ undefined for $s = 0$
  —floating point 0.0 hardly encountered, replace by 'sub-gradient' usually okay
- **sparse** network per example
- **fast** arithmetic computation

ReLU (with or without some tanh): arguably the most widely-used for deep learning

Dead Neuron Issue

$$\phi(s) = \max(s, 0)$$

- $s < 0$: $\phi(s) = 0$ and $\phi'(s) = 0$ per example
- $s < 0$ for every example (dead neuron) if
  - very negative $w_{0?}^{(\ell)}$ (e.g. update from a very big gradient step)
  - all positive $x_i^{(0)}$ (e.g. images without shifting) + negative weights $w_{ij}^{(1)}$

dead neurons worrisome (but not always serious)

Leaky Rectified Linear Unit

$$\phi(s) = \max(s, 0.01s) = \max(s, 0) + 0.01 \min(s, 0)$$

- $s > 0$: $\phi(s) = s$ and $\phi'(s) = 1$ per example
- $s < 0$: $\phi(s) = 0.01s$ and $\phi'(s) = 0.01$ per example

less likely to have dead neurons, but why 0.01?

Parametric Rectified Linear Unit

$$\phi(\alpha, s) = \max(s, \alpha \cdot s)$$

- ReLU: $\alpha = 0$, Leaky ReLU: fixed α
- optimizable α:

$$\begin{aligned}\frac{\partial e_n}{\partial \alpha_j^{(\ell)}} &= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial \alpha_j^{(\ell)}} \\ &= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\frac{\partial \phi(\alpha_j^{(\ell)}, s_j^{(\ell)})}{\partial \alpha_j^{(\ell)}}\right)\end{aligned}$$

with $\frac{\partial \phi(\alpha, s)}{\partial \alpha} = s$ if $\alpha s > s$, or 0 otherwise.

'power' of deep learning: anything (loosely) differentiable is 'learnable'

Weight Initialization

- all 0: too symmetric for `tanh`, not differentiable for ReLU
- constants: ‘cloned’ neurons
- too large: saturation/gradient vanishing for `tanh`, (some dying, some overfitting) for ReLU

want

- random: avoid all-0 or constants
- small: ‘well-behaved initialization

next small random initialization with zero-mean
(easier to analyze)

Small Random Initialization for Forwarding tanh

score $\boxed{s_j^{(\ell)}} = \sum_{i=0}^{d^{(\ell-1)}} \boxed{w_{ij}^{(\ell)}} \cdot x_i^{(\ell-1)}$, transformed $\boxed{x_j^{(\ell)}} = \phi_j^{(\ell)}\left(\boxed{s_j^{(\ell)}}\right)$

- $w_{ij}^{(\ell)}$ small $\Rightarrow s_j^{(\ell)}$ small $\Rightarrow \tanh'(s_j^{(\ell)}) \approx 1$ (approximately linear)

$$
\begin{aligned}
var(x_j^{(\ell)}) &\approx var(s_j^{(\ell)}) = var\left(\sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)}\right) \\
(independence) &= var(w_{0j}^{(\ell)}) + \sum_{i=1}^{d^{(\ell-1)}} var(w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)})
\end{aligned}
$$

- (ind.): $var(wx) = E(x)^2 var(w) + E(w)^2 var(x) + var(w)var(x)$
  - $E(w) = 0$ by construction $\Rightarrow E(x_i^{(\ell)}) = 0$
  - $E(x_i^{(0)}) = 0$ for mean-shifted features

'ideal' $var(w) = 1/d^{(\ell-1)}$ so
$var(x_j^{(\ell)}) \approx var(x_i^{(\ell-1)})$

Small Random Initialization for Backward `tanh`

$$\delta_j^{(\ell-1)} = \sum_k \left(\delta_k^{(\ell)}\right) \left(w_{jk}^{(\ell)}\right) \left(\phi'\left(s_j^{(\ell-1)}\right)\right)$$

- assume approximately linear: $\phi' \approx 1$
- to keep $var(\delta_j^{(\ell-1)})$ similar to $var(\delta_k^{(\ell)})$:

$$var(w) = \frac{1}{d^{(\ell)}}$$

Xavier initialization: let $var(w) = \frac{2}{d^{(\ell-1)}+d^{(\ell)}}$

Small Random Initialization for ReLU

-

$$\begin{aligned} var(s_j^{(\ell)}) &= var\left(\sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)}\right) \\ (independence) &= var(w_{0j}^{(\ell)}) + \sum_{i=1}^{d^{(\ell-1)}} var(w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)}) \end{aligned}$$

- (ind.): $var(wx) = E(x)^2 var(w) + E(w)^2 var(x) + var(w) var(x)$
  - $E(w) = 0$ by construction
  - $var(wx) = var(w) E(x^2)$
  - $E(x^2) = 0.5 var(s^2)$ because of 'Re'LU

He initialization: $var(w) = 2/d^{(\ell-1)}$ so
$var(s_j^{(\ell)}) \approx var(s_i^{(\ell-1)})$