Machine Learning
(機器學習)

Lecture 15: Machine Learning Soundings

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

Deep Learning Initialization

Weight Initialization

- all 0: too symmetric for `tanh`, not differentiable for ReLU
- constants: ‘cloned’ neurons
- too large: saturation/gradient vanishing for `tanh`, half dying 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 $s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} \cdot x_i^{(\ell-1)}$, transformed $x_j^{(\ell)} = \phi_j^{(\ell)}\left(s_j^{(\ell)}\right)$

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

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

- (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 (Xavier Glorot, 2010): let $var(w) = \frac{2}{d^{(\ell-1)} + d^{(\ell)}}$

Small Random Initialization for ReLU

-

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

- (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 (Kaiming He, 2015):
$var(w) = 2/d^{(\ell-1)}$ so $var(s_j^{(\ell)}) \approx var(s_i^{(\ell-1)})$

Questions?

Deep Learning Optimization

Difficulty of Deep Learning Optimization

error surface complicated

- local minima: not as bad as imagined
- saddle points/local maxima: easily escapable (especially with SGD)
- plateau: need larger learning rate η
- ravines: need to avoid oscillation

stability <> computation trade-off

slow computation of gradient (backprop)
$\Rightarrow$ SGD on minibatch
$\Rightarrow$ 'instable' estimate of gradient

getting more stable estimate? **averaging**

Running Average Estimate of Gradient

gradient descent: $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \eta \cdot \mathbf{v}_t$

original minibatch SG

gradient estimate $\mathbf{v}_t = \Delta_t$ from one minibatch SG

averaging by multiple SG

if minibatch SG for M times at t-th iteration, each getting $\Delta_t^{(m)}$, more stable gradient estimate by uniform averaging $\mathbf{v}_t = \frac{1}{M}\sum_{m=1}^{M} \Delta_t^{(m)}$ —needing M times more computation than original minibatch SGD

speedup by reusing each Δ_{t-m+1} as $\Delta_t^{(m)}$

$\mathbf{v}_t = \frac{1}{M}\sum_{m=1}^{M} \Delta_{t-m+1}$ —'moving window' average of SG

issue with 'moving window' average:
uniformly weighted

Averaging SG Non-uniformly

Running Average

- $$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\Delta_t$$

  with $0 \leq \beta < 1$ to control how much history to take
- $\beta = 0$: original SGD

$$\mathbf{v}_t = \sum_{m=1}^{t} \beta^{t-m}(1-\beta)\Delta_t$$

—size-t window, exponentially-decreasing aeveraging

SGD **with momentum**: optimization direction
= current SG (Δ_t) + historical inertia ($\mathbf{v}_{t-1}$)

Benefits of SGD with Momentum

$$\begin{aligned}\mathbf{v}_t &= \beta\mathbf{v}_{t-1} + (1-\beta)\Delta_t \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \eta\mathbf{v}_t\end{aligned}$$

- some variance in SG canceled out
- oscilliation across ravine dampened
- shallow local optima/saddle points escaped

SGD with momentum: ‘stablize’ SG with running average

Per-Component Learning Rate

fixed learning rate : $\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \mathbf{v}_t$

per-component learning rate : $\mathbf{w}_t = \mathbf{w}_{t-1} - \boldsymbol{\eta}_t \odot \mathbf{v}_t$

intuition: scales error surface

want: smaller step for larger gradient component

Running Average of Gradient Magnitude

want: smaller step for larger gradient component, say

$$\eta_t = \frac{1}{\nabla E(\mathbf{w}_t) \odot \nabla E(\mathbf{w}_t)} \text{ per component}$$

- full gradient ∇E not available, SG only
- using $\Delta \odot \Delta$ directly: not very stable

idea: running average of $\Delta_t \odot \Delta_t$

RMSProp

$$\begin{aligned}\mathbf{u}_t &= \beta \mathbf{u}_{t-1} + (1-\beta)\Delta_t \odot \Delta_t \\ \boldsymbol{\eta}_t &= \frac{\eta}{\sqrt{\mathbf{u}_t + \epsilon}} \text{ per component} \\ \mathbf{w}_t &= \mathbf{w}_{t-1} - \boldsymbol{\eta}_t \odot \Delta_t\end{aligned}$$

RMSProp: SGD + per-component learng rate
using running average of magnitude

Adam: Adaptive Moment Estimation

Adam $\approx$ momentum + RMSProp + global decay

$$
\begin{aligned}
\mathbf{v}_t &= \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\Delta_t \\
\mathbf{u}_t &= \beta_2 \mathbf{u}_{t-1} + (1-\beta_2)\Delta_t \odot \Delta_t \\
\boldsymbol{\eta}_t &= \frac{\eta}{\sqrt{\mathbf{u}_t + \epsilon}} \cdot \frac{1}{\sqrt{t/N}} \\
\mathbf{w}_t &= \mathbf{w}_{t-1} - \boldsymbol{\eta}_t \odot \mathbf{v}_t
\end{aligned}
$$

- momentum in $\mathbf{v}_t$
- RMSProp in $\mathbf{u}_t$
- global decay by $\sqrt{t/N}$
- (some minor correction of estimation)

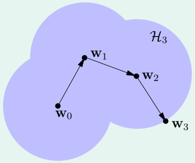
Adam usually more aggressive than original SGD (but can also overfit faster)

Questions?

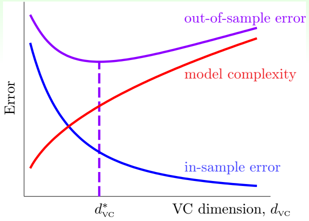
Deep Learning Regularization

A Basic Trick: Early Stopping

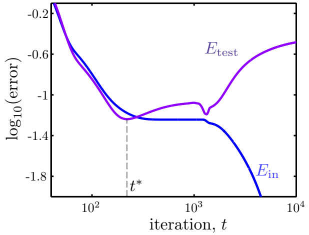
- **GD/SGD (backprop)** visits more weight combinations as t increases



- smaller t effectively decrease d_{VC}
- better 'stop in middle': **early stopping**



(d^*_{VC} **in middle, remember? :-)**)



when to stop? **validation!**

Co-Adaptation Issue of Deep Learning

co-adaptation

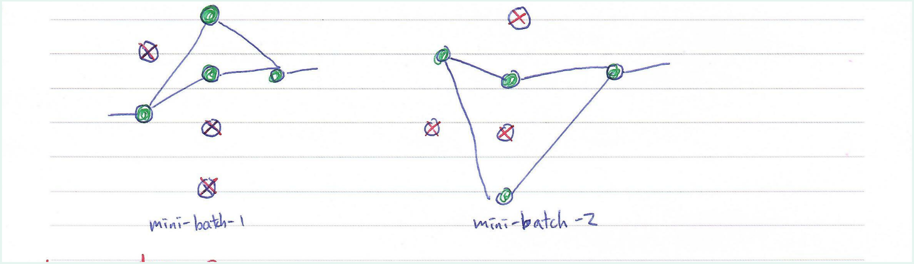
- consistent mistakes from some neurons: like noise
- corrected by fitting other neurons: like overfit

—unhealthy dependence between neurons

co-adaptation harms generalization of deep learning

Breaking the Dependence

idea: shut down some neurons randomly

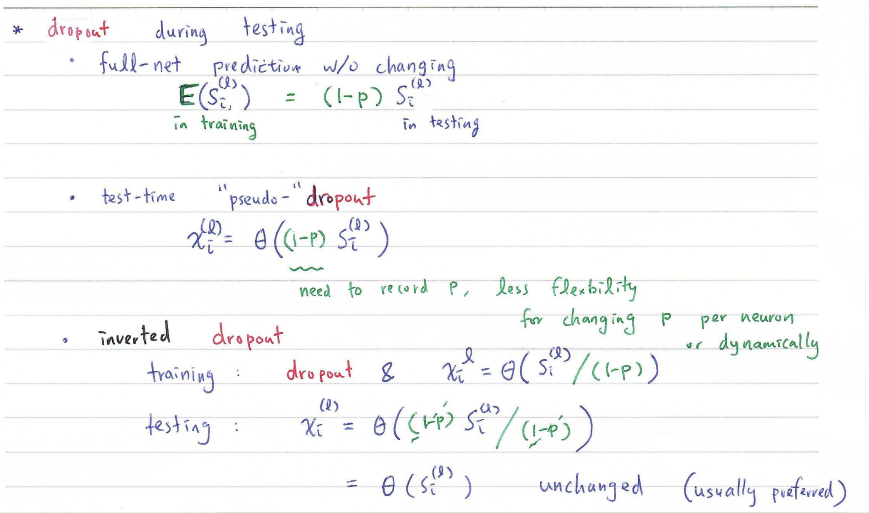


dropout:

- drop p, keep $1 - p$
- implicit aggregation of many thinner networks
- slow down convergence, but faster per-iteration

dropout: simple yet effective technique for deep learning regularization

Dropout During Testing



* dropout during testing

- full-net prediction w/o changing

$$\underset{\text{in training}}{E(S_i^{(l)})} = \underset{\text{in testing}}{(1-p)\,S_i^{(l)}}$$

- test-time "pseudo-" dropout

$$x_i^{(l)} = \theta\Big(\underset{\sim}{(1-p)}\,S_i^{(l)}\Big)$$

need to record p, less flexibility for changing p per neuron or dynamically

- inverted dropout

training : dropout & $x_i^{l} = \theta\big(S_i^{(l)}/(1-p)\big)$

testing : $x_i^{(l)} = \theta\big(\cancel{(1-p)}\,S_i^{(l)}/\cancel{(1-p)}\big)$

$= \theta(S_i^{(l)})$ unchanged (usually preferred)

Questions?