Neural Networks II

Wei Li

Syracuse University

Spring 2021

Some Issues in Training Deep Neural Networks

Approximation and Statistical Properties

Some Issues in Training Deep Neural Networks

Some Issues in Training Deep Neural Networks

The objective function $\mathcal{L}(\theta)$ is nonconvex, possessing many local minima; with deep networks, vanishing/exploding gradients are also issues

- the choice of learning rates
- standardization of all inputs
- the initialization of the weights

The complexity of the model needs special attention

- to avoid overfitting, regularization methods are used
- choice of the width/depth, the architecture of the network

Learning Rates

$$\mathcal{L}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(f\left(x^{(i)},\theta\right),y^{(i)}\right) + \lambda\Omega(\theta)$$

How to set ϵ_t?

GD update: at the t-th iteration

$$\theta^{<t+1>} = \theta^{<t>} - \epsilon_t\frac{1}{B}\sum_{i\in\mathcal{B}}\left\{\nabla_\theta\mathcal{L}\left(f\left(x^{(i)},\theta^{<t>}\right),y^{(i)}\right) + \lambda\nabla_\theta\Omega\left(\theta^{<t>}\right)\right\}$$

In training deep neural networks, the line search method is generally difficult due to the use of SGD.

A sufficient condition for SGD to converge is the Robbins-Monro Condition:

$$\frac{\sum_{t=1}^{\infty} \epsilon_t^2}{\sum_{t=1}^{\infty} \epsilon_t} \to 0 \quad \text{as } \epsilon_t \to 0$$

- Step decay: Reduce the learning rate by some factor every few epochs.
- Exponential decay: $\epsilon_t = \epsilon_0 e^{-kt}$ where ϵ_0 and k are hyperparameters and t is the iteration number or epoch number.
- Polynomial decay: $\epsilon_t = \frac{\epsilon_0}{(1+kt)^a}$ where , ϵ_0 and a, k are the hyperparameters and t is the iteration number or epoch number.

- learning rate warmup (quickly starts with increasing learning rate, then decreases it slowly)
- cyclical learning rate
- iterate averaging

$$\theta^{<t+1>} = \frac{1}{t}\sum_{i=1}^{t}\theta^{<i>} = \frac{1}{t}\theta^{<t>} + \frac{t-1}{t}\bar{\theta}^{<t>}$$

Per-Parameter Adaptive Learning Rates

- Adagrad (adaptive gradient method)
- RMSProp (root mean square propgation)
- Adam (Adaptive Moment Estimation).

All are improvements of Gradient Descent with Momentum.

As of now, Adam (and its variant) is one of the most popular and successful algorithms in deep learning.

Input normalization

- The whole training input are often normalized before they feed into the network.
- **batch normalization**: the normalization is restrained to each mini-batch for each layer separately in the training process.

Weight initialization

Weight initialization has the following impact

- convergence or not, speed of convergence
- small values cause diminishing signals passed through
- large values cause exploding gradients
- large values cause saturated activation (vanishing gradients)

Three most common weight initialization:

- Initialize weights to zero
- Initialize weights to $\mathcal{N}(\mu = 0, \sigma^2)$
- Initialized weights to uniform$(-u, u)$

However, uniform and deterministic weight initialization should be avoided. Random initialization can break the unit symmetry and is often used.

LeCun/He/Xavier/Glorot and Bengio Initialization: The underlying idea is to preserve the variance of activation values between layers.

LeCun initialization:

$$\sigma\left(w_{i,j}^{[l]}\right) = \sqrt{\frac{1}{n^{[l-1]}}}$$

He initialization:

$$\sigma\left(w_{i,j}^{[l]}\right) = \sqrt{\frac{2}{n^{[l-1]}}}$$

Xavier initialization:

$$\sigma(w_{i,j}^{[l]}) = \sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}.$$

Glorot and Bengio initialization:

$$w_{i,j}^{[l]} \sim \mathrm{U}\left(-\frac{6}{n^{[l-1]} + n^{[l]}}, \frac{6}{n^{[l-1]} + n^{[l]}}\right)$$

Regularization

As for any machine learning algorithm/methods, overfitting may occur if a model becomes too complex.

- early stopping, ℓ_2-regulalization, and dropout are the most common regularization methods.

Early stopping: stop the trainining process when the error on the validation set starts to increase. This is heuristic approach.

The ℓ_2-regulalization:

$$\mathcal{L}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}, \theta\right), y^{(i)}\right) + \lambda\Omega(\theta)$$

where

$$\Omega(\theta) = \sum_k \sum_i \sum_j \left(w_{i,j}^{[k]}\right)^2 = \sum_k \left\|W^{[k]}\right\|_F^2$$

Dropout is applied to a layer randomly, on a per-example basis, some percentage of its neurons are randomly "dropped out", and their connections are deactivated.

Dropout is applied in both during forward and backward propagation during training phase for better learning of the parameters, but is not used during test phase.

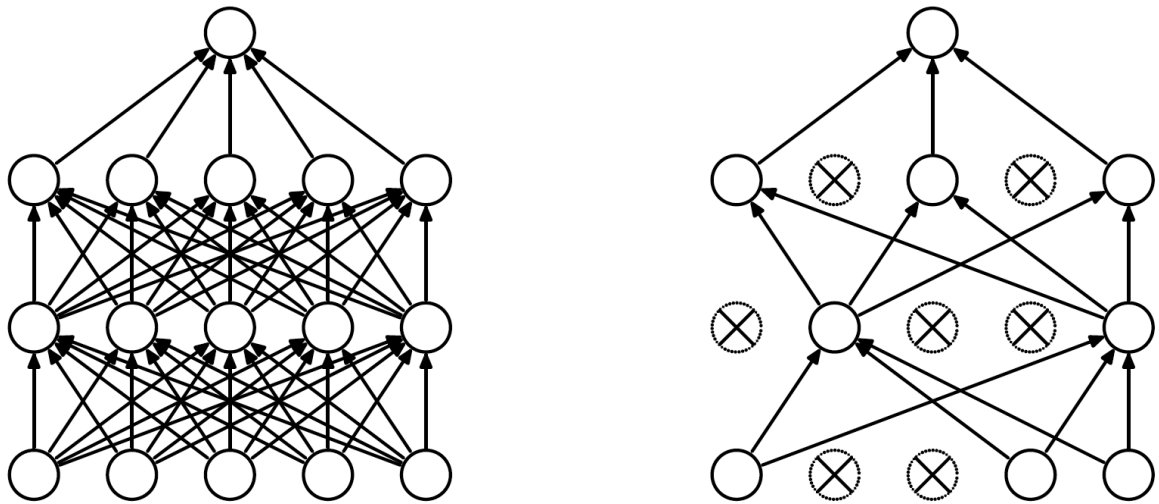

Image Source: Kevin Murphy (2022) Fig. 13.18.

Approximation and Statistical Properties

Approximation properties

The **Universal Approximation Theorem**:

A feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g., sigmoid, Relu) can approximate any suitably smooth function arbitrarily well, provided that the network is given enough hidden units.

(Hornik et al.,1989; 1990; Leshno et al. 1993)

Wider or deeper? (Montufar et al 2014)

A feedforward network with a single layer is sufficient to represent any function, but the layer size may be infeasibly large. Shallow networks may fail to learn and generalize correctly if "too shallow".

In the asymptotic limit of many hidden layers, deep networks are able to separate their input space into exponentially more linear response regions than their shallow counterparts, with the same number of units.

Empirically, deeper networks seem to result in better generalization.

Convergence rates (Schmidt-Hieber 2020)

$$Y_i = f(X_i) + \varepsilon_i, \quad X_i \in \mathbb{R}^p$$

Assume a general composition assumption on the true regression function f^*.

$$f^* = g_q \circ g_{q-1} \circ \cdots \circ g_1 \circ g_0$$

- $g_i : [a_i, b_i]^{d_i} \to [a_{i+1}, b_{i+1}]^{d_{i+1}}$.
- each g_i is a t_i-variate, α_i-smooth function.
  - t_i is not growing as levels deepens (required by minimax)

Using a sparse multi-layer feedforward network for f with

- relu activation
- bounded networks parameters values
- number of parameters exceed the sample size
  - with suitably tuning sparsity level

The empirical minimizer can achieves L_2 (test MSE) minimax rates (up to log factor)

$$\alpha_j^* := \alpha_j \prod_{\ell=j+1}^{q} (\alpha_\ell \wedge 1)$$

$$\phi_n := \max_{j=0,\ldots,q} n^{-\frac{2\alpha_j^*}{2\alpha_j^*+t_j}}$$

with depth scales with the sample size $\log n$.

If the true function is of the generalized additive form:

$$f^*(X_1, \ldots, X_p) = \sum_{j=1}^{p} f_j(X_j)$$

or tensor-product form:

$$f^*(X) = \sum_{\ell=1}^{M} b_\ell \prod_{j=1}^{p} f_{j\ell}(X_j)$$

assuming each component function is α-smooth, the sparse deep NN achieves the rate $n^{-\frac{2\alpha}{2\alpha+1}}$ up to log factor (p does not show up in the exponent).

A (well-tuned) sparse deep NN in theory can circumvent the curse of dimensionality.