Lecture 6: Impact of the Hyper-parameters of DNNs

Soufiane Hayou

Monday 29$^{\text{th}}$ May, 2023

Training a NN

Let $S = \{(x_i, y_i), i = 1, \dots, N\}$ be the training dataset. The empirical risk minimization is given by

$$\min_{f \in \mathcal{H}} L_{\mathcal{S}}[f] = \min_{\theta=(w,v,b) \in \mathcal{R}^p} L_{\mathcal{S}}[f(\theta)] = \min_{\theta \in \mathcal{R}^p} \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i).$$

where ℓ is the loss function, e.g. $\ell(z, z') = \frac{1}{2}\|z - z'\|^2$.

$\rightarrow$ Run GD/SGD/mGD/... to optimize the empirical loss.

Issues with DNNs

We simply iterate the structure of shallow neural networks T times. T is the *depth* of the DNN. Concretely, deep neural networks make up the following hypothesis space

$$\begin{aligned}
&\mathcal{H}_{\text{dnn}} = \left\{ f : f(x) = v^\top f_T(x), v \in \mathcal{R}^{d_T} \right\} \\
&\text{where} \\
&f_{t+1}(x) = \sigma(W_t f_t(x) + b_t), \quad W_t \in \mathcal{R}^{d_{t+1} \times d_t}, \quad b_t \in \mathcal{R}^{d_{t+1}}, \\
&\text{for} t = 0, \ldots, T-1, \quad \text{with} \quad d_0 = d, \quad f_0(x) = x.
\end{aligned} \tag{1}$$

- $\theta = \{W_0, \ldots, W_{T-1}\} \cup \{b_0, \ldots, b_{T-1}\} \cup \{v\}$.

Fully-Connected DNNs

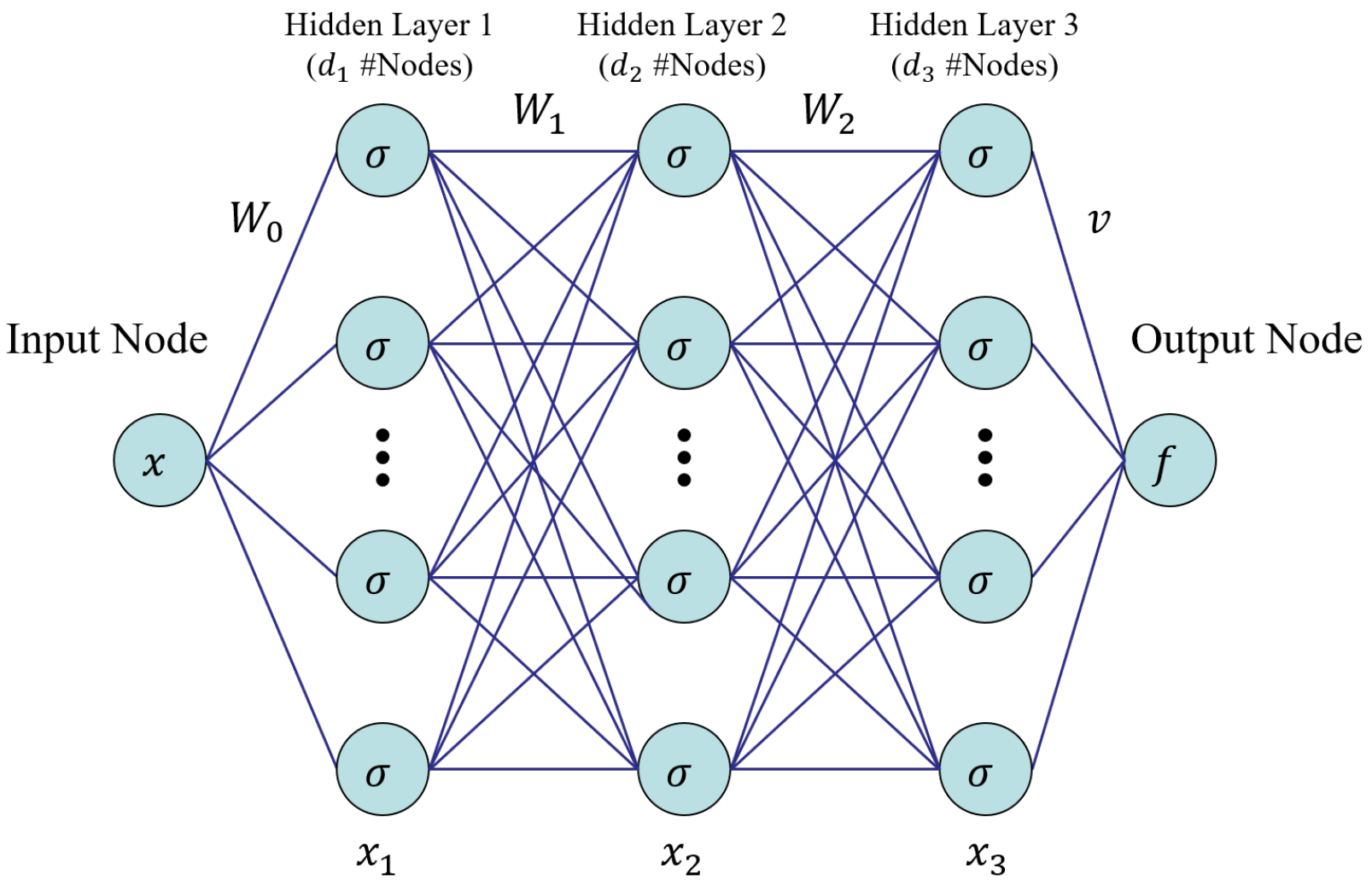

Figure: Illustration of a function parameterized by a DNN with three hidden layers.

Gradient Backpropagation (GB)

Let $x_{t+1} = g_t(x_t, W_t)$ ($x_t = f_t(x)$ for FC-DNN), with this we have

$$\begin{aligned} \nabla_{W_t} \ell(x_{T+1}, y) &= [\nabla_{W_t} x_{t+1}]^\top \nabla_{x_{t+1}} \ell(x_{T+1}, y) \\ &= [\nabla_{W_t} g_t(x_t, W_t)]^\top \nabla_{x_{t+1}} \ell(x_{T+1}, y). \end{aligned} \tag{2}$$

Let us define $p_t = \nabla_{x_t} L(x_{T+1}, y)$, then

$$\nabla_{W_t} \ell(x_{T+1}, y) = [\nabla_{W_t} g_t(x_t, W_t)]^\top p_{t+1}. \tag{3}$$

Gradient Backpropagation (GB)

Therefore, we only need $\{p_t\}$ to compute the gradients readily. Observe that

$$p_t = [\nabla_{x_t} g_t(x_t, W_t)]^\top p_{t+1}, \qquad p_{T+1} = \nabla_{x_{T+1}} \ell(x_{T+1}, W_T).$$

$\rightarrow$ This provides a recursive way to compute gradients in a single backward pass. In summary GB is performed as follows:

1. Forward pass to compute x_t's.
2. Backward pass to compute the gradients.

Algorithm 1: back-propagation Algorithm

$x_0 = x \in \mathcal{R}^d$ **for** $t = 0, 1, \ldots, T$ **do**

  $x_{t+1} = g_t(x_t, W_t) = \sigma(W_t^\top x_t)$;

end

Set $p_{T+1} = \nabla_{x_{T+1}} \ell(x_{T+1}, y)$;

for $t = T, T-1, \ldots, 1$ **do**

  $\nabla_{W_t} \ell(x_{T+1}, y) = p_{t+1}^\top \nabla_{W_t} g_t(x_t, W_t)$;

  $p_t = [\nabla_{x_t} g_t(x_t, W_t)]^\top p_{t+1}$;

end

return $\{\nabla_{W_t} \ell(x_{T+1}, y) : t = 0, \ldots, T\}$

Gradient Vanishing/Exploding

Consider the simple case of a FC-DNN with a neuron per layer, i.e. $x_{t+1} = g_t(x_t, w_t) = \sigma(w_t \times x_t)$ where $x_t, w_t \in \mathbb{R}$.

Gradient Vanishing/Exploding

Consider the simple case of a FC-DNN with a neuron per layer, i.e. $x_{t+1} = g_t(x_t, w_t) = \sigma(w_t \times x_t)$ where $x_t, w_t \in \mathbb{R}$. Hence

$$p_t = w_t \sigma'(w_t x_t) p_{t+1}, \qquad p_{T+1} = \nabla_{x_{T+1}} \ell(x_{T+1}, y).$$

$\rightarrow$ This can lead to gradient vanishing (or exploding) in the limit of large depth

1. Choice of the activation function
2. Choice of initialization
3. Choice of architecture?

Activation functions

ReLU (Rectified Linear Unit)	$\sigma(z) = \max(0, z)$	(4)
Leaky ReLU	$\sigma(z) = \max(0, z) + \delta \min(0, z)$	(5)
Tanh	$\sigma(z) = \tanh(z)$	(6)
Sigmoid	$\sigma(z) = \frac{1}{1 + e^{-z}}$	(7)
Soft-plus	$\sigma(z) = \log(1 + e^{z})$	(8)

Choice of Activation function

If $|\sigma'(x)| < 1$ for all x, we might have a gradient vanishing problem

Choice of Activation function

If $|\sigma'(x)| < 1$ for all x, we might have a gradient vanishing problem
Example in a simple case $d_1 = d_2 = \cdots = d_L = 1$,

$$p_t = w_t \sigma'(w_t x_t) p_{t+1}, \qquad p_{T+1} = \nabla_{x_{T+1}} \ell(x_{T+1}, W_T).$$

$\rightarrow$ This can lead to gradient vanishing (GV)/ gradient exploding (GE) in the limit of large depth

1. $\sigma(x) = \max(x, 0), \quad \sigma'(x) = 1_{x>0}$ (no GV)
2. $\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$ (potential GV)

$\rightarrow$ Activations with $|\sigma'(x)| > 1$ are not used in practice as it might lead to numerical instability (gradient exploding)

Choice of the Initialization weights

Can $|w_t|$ be initialized to avoid a gradient vanishing problem (only at initialization)?

$$r_t = \frac{p_t}{p_{t+1}} = w_t \sigma'(w_t x_t),$$

$\rightarrow$ We would like to control the growth rate $\frac{p_t}{p_{t+1}}$. Assume that σ is ReLU and $w_t \sim \mathcal{N}(0, \gamma^2)$. What is a good choice for γ?

1. $\mathbb{E}_W[r_t] = 0$
2. $\mathbb{E}_W[r_t^2] = \frac{\gamma^2}{2}$

$\rightarrow$Hence, we should choose $\gamma^2 = 2$. What about general widths $d_1, d_2, \ldots, d_L$?

In DNNs with width d, this becomes $\gamma_d^2 = 2/d$. This is known as Kaiming (or He) initialization scheme. More generally, with different widths d_t, we have

$$W_t^{ij} \sim \mathcal{N}(0, \frac{2}{d_t}).$$

$\rightarrow$ This also solves a problem of vanishing/exploding during forward propagation!!