Lecture 4: (Deep) Neural Networks

Soufiane Hayou

Wednesday 24th May, 2023

So far, we discussed,

So far, we discussed,

- Optimization (GD, SGD, Momentum GD, ...)

which is “model agnostic”! (general hypothesis class $\mathcal{H}$)

So far, we discussed,

- Optimization (GD, SGD, Momentum GD, ...)

which is “model agnostic”! (general hypothesis class $\mathcal{H}$)

$\rightarrow$ Today, we discuss a specific hypothesis class: Neural Networks.

Shallow Neural Networks

$$\mathcal{H}_{\mathrm{nn},M} = \left\{ f : f(x) = \sum_{j=1}^{M} v_j \sigma(w_j^T x + b_j), w_j \in \mathcal{R}^d, v_j \in \mathcal{R}, b_j \in \mathcal{R} \right\} \tag{1}$$

where

- x is the input
- v_j, w_j's are the weights, and b_j's are the bias.
- σ is the activation function

We can define the hypothesis class of general-width neural networks by

$$\mathcal{H}_{\mathrm{nn}} = \cup_{M \geq 1} \mathcal{H}_{\mathrm{nn},M}$$

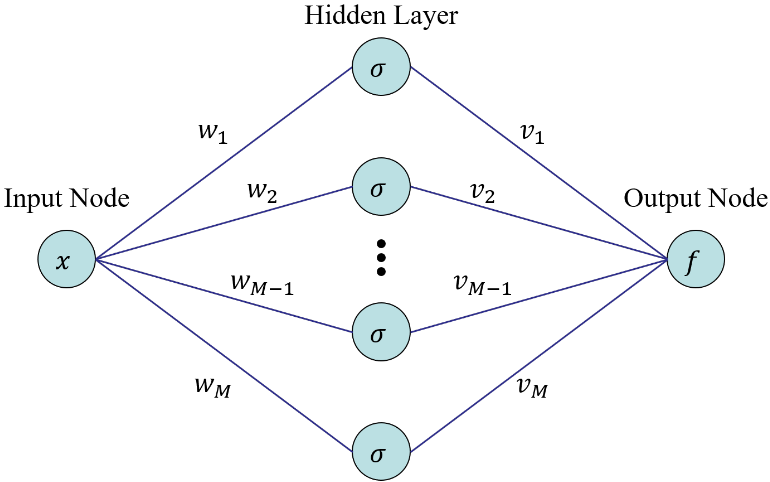

Figure: Illustration of a function parameterized by a shallow neural network with one hidden layer. For convenience we ignore the biases.

Activation functions

$$
\begin{aligned}
&\text{ReLU (Rectified Linear Unit)} && \sigma(z) = \max(0, z) && (2)\\
&\text{Leaky ReLU} && \sigma(z) = \max(0, z) + \delta \min(0, z) && (3)\\
&\text{Tanh} && \sigma(z) = \tanh(z) && (4)\\
&\text{Sigmoid} && \sigma(z) = \frac{1}{1 + e^{-z}} && (5)\\
&\text{Soft-plus} && \sigma(z) = \log(1 + e^{z}) && (6)
\end{aligned}
$$

Theorem (Universal Approximation Theorem for NNs)

Let $K \subset \mathcal{R}^d$ be closed and bounded and $f^* : K \to \mathcal{R}$ be continuous. Assume that the activation function σ is one of the activations above. Then, for every $\epsilon > 0$ there exists $f \in \mathcal{H}_{nn}$ such that

$$\|f - f^*\|_{C(K)} = \max_{x \in K} |f(x) - f^*(x)| < \epsilon \tag{7}$$

Training a NN

Let $S = \{(x_i, y_i), i = 1, \ldots, N\}$ be the training dataset. The empirical risk minimization can be written as

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

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.

Deep Neural Networks

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

$$\begin{aligned}
&\mathcal{H}_{\text{dnn}} = \left\{ f : f(x) = v^\top f_T(x), v \in \mathcal{R}^{d_T} \right\} \\
&\text{where} \\
&f_{t+1}(x) = \sigma(W_t f_t(x) + b_t), \quad W_t \in \mathcal{R}^{d_{t+1} \times d_t}, \quad b_t \in \mathcal{R}^{d_{t+1}}, \\
&\text{for } t = 0, \ldots, T-1, \quad \text{with} \quad d_0 = d, \quad f_0(x) = x.
\end{aligned} \tag{8}$$

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

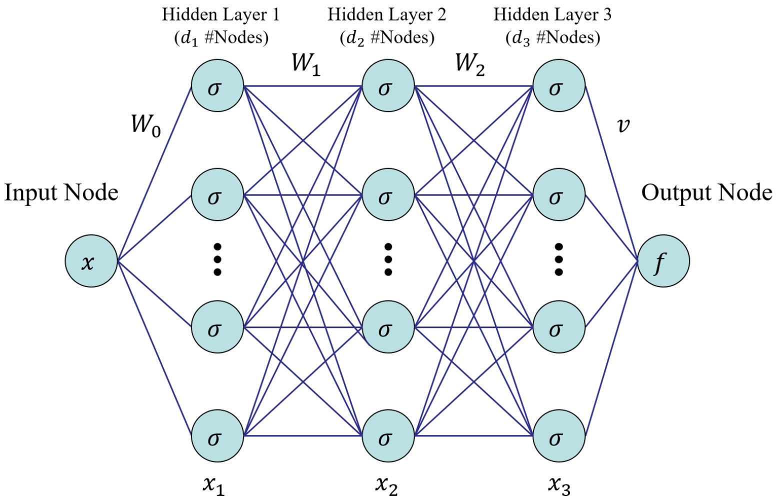

Figure: Illustration of a function parameterized by a DNN with three hidden layers.

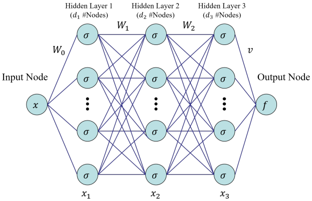

Figure: Illustration of a function parameterized by a DNN with three hidden layers.

- Can represent hierarchical features naturally $\rightarrow$ automated multi-layer feature engineering.
- "Richness" of the hypothesis space increases with width and depth

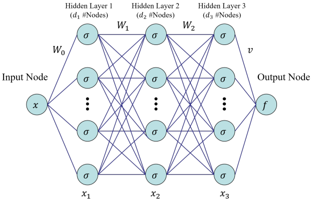

Figure: Illustration of a function parameterized by a DNN with three hidden layers.

- Can represent hierarchical features naturally $\rightarrow$ automated multi-layer feature engineering.
- "Richness" of the hypothesis space increases with width and depth

How do we compute the gradients(for training)?

Gradient Backpropagation (GB)

Like shallow NNs, DNNs can be trained using GD/SGD/... We compute the gradients using Gradient Backpropagation which is just the chain rule applied to DNNs. We write $x_{t+1} = g_t(x_t, W_t)$ ($x_t = f_t(x)$ for FC-DNN), with this we have

$$\begin{aligned} \nabla_{W_t} \ell(x_{T+1}, y) &= [\nabla_{W_t} x_{t+1}]^\top \nabla_{x_{t+1}} \ell(x_{T+1}, y) \\ &= [\nabla_{W_t} g_t(x_t, W_t)]^\top \nabla_{x_{t+1}} \ell(x_{T+1}, y). \end{aligned} \tag{9}$$

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

$$\nabla_{W_t} \ell(x_{T+1}, y) = [\nabla_{W_t} g_t(x_t, W_t)]^\top p_{t+1}. \tag{10}$$

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.

GB algorithm

Algorithm 1: Backpropagation for FC-DNN

$x_0 = x \in \mathcal{R}^d$ **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 $\{\nabla_{W_t} \ell(x_{T+1}, y) : t = 0, \ldots, T\}$

Question

- Is the loss function of a DNN convex?
- If not, then what happens to GD?

Fully-Connected NNs are structure-agnostic: the input coordinates are interpreted in a similar fashion.

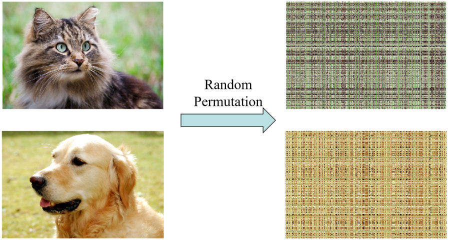

Figure: Illustration of the effect of permutation on data with spatial structure. Applying a permutation on the spatial indices destroys such structure, but fully connected neural networks treats these two cases equivalently, thus unsuitable to process such data types.

Fully-Connected NNs are structure-agnostic: the input coordinates are interpreted in a similar fashion.

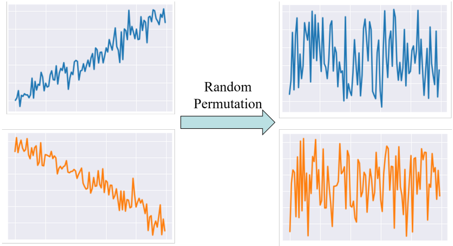

Figure: Another illustration with temporal data.