

Neural Network

A Selective Overview of Deep Learning

Tao Huang

中山大學
SUN YAT-SEN UNIVERSITY



數學學院
篤行工作室

Section 1

Feed-forward neural networks

Model setup

Deep neural networks (DNNs) use composition of a series of simple nonlinear functions to model nonlinearity

$$\mathbf{h}^{(L)} = \mathbf{g}^{(L)} \circ \mathbf{g}^{(L-1)} \circ \dots \circ \mathbf{g}^{(1)}(\mathbf{x}), \quad (1)$$

Model setup

Deep neural networks (DNNs) use composition of a series of simple nonlinear functions to model nonlinearity

$$\mathbf{h}^{(L)} = \mathbf{g}^{(L)} \circ \mathbf{g}^{(L-1)} \circ \dots \circ \mathbf{g}^{(1)}(\mathbf{x}), \quad (1)$$

for $\ell = 1, \dots, L$, define

$$\mathbf{h}^{(\ell)} = \mathbf{g}^{(\ell)}(\mathbf{h}^{(\ell-1)}) \triangleq \sigma(\mathbf{W}^{(\ell)} \mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}), \quad (2)$$

Model setup

Deep neural networks (DNNs) use composition of a series of simple nonlinear functions to model nonlinearity

$$\mathbf{h}^{(L)} = \mathbf{g}^{(L)} \circ \mathbf{g}^{(L-1)} \circ \dots \circ \mathbf{g}^{(1)}(\mathbf{x}), \quad (1)$$

for $\ell = 1, \dots, L$, define

$$\mathbf{h}^{(\ell)} = \mathbf{g}^{(\ell)}(\mathbf{h}^{(\ell-1)}) \triangleq \sigma(\mathbf{W}^{(\ell)} \mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}), \quad (2)$$

The activation function $\sigma(\cdot)$ is usually applied element-wise, and a popular choice is the ReLU (Rectified Linear Unit) function:

$$[\sigma(\mathbf{z})]_j = \max\{z_j, 0\}. \quad (3)$$

Model setup

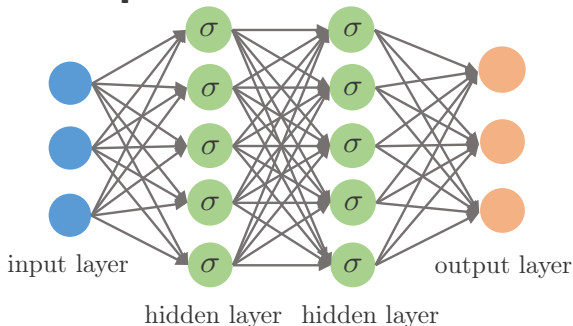


Figure 1: A feed-forward neural network with an input layer, two hidden layers and an output layer. The input layer represents raw features $\{\mathbf{x}_i\}_{1 \leq i \leq n}$. Both hidden layers compute an affine transform (a.k.s. indices) of the input and then apply an element-wise activation function $\sigma(\cdot)$. Finally, the output returns a linear transform followed by the softmax activation (resp. simply a linear transform) of the hidden layers for the classification (resp. regression) problem.

Model setup

The *soft-max* function:

$$f_k(\mathbf{x}; \boldsymbol{\theta}) \triangleq \frac{\exp(z_k)}{\sum_k \exp(z_k)}, \forall k \in [K], \quad (4)$$

where $\mathbf{z} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)} \in \mathbb{R}^K$.

Model setup

The *soft-max* function:

$$f_k(\mathbf{x}; \boldsymbol{\theta}) \triangleq \frac{\exp(z_k)}{\sum_k \exp(z_k)}, \forall k \in [K], \quad (4)$$

where $\mathbf{z} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)} \in \mathbb{R}^K$.

Loss

$$\mathcal{L}(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), y) = - \sum_{k=1}^K \mathbb{1}\{y = k\} \log p_k, \quad (5)$$

, where $\boldsymbol{\theta} \triangleq \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)} : 1 \leq \ell \leq L+1\}$.

Back-propagation in computational graphs

The key to the above training procedure, namely SGD, is the calculation of the gradient $\nabla \ell_{\mathcal{B}}(\theta)$, where

$$\ell_{\mathcal{B}}(\theta) \triangleq |\mathcal{B}|^{-1} \sum_{i \in \mathcal{B}} \mathcal{L}(\mathbf{f}(\mathbf{x}_i; \theta), y_i). \quad (6)$$

Back-propagation in computational graphs

The key to the above training procedure, namely SGD, is the calculation of the gradient $\nabla \ell_{\mathcal{B}}(\boldsymbol{\theta})$, where

$$\ell_{\mathcal{B}}(\boldsymbol{\theta}) \triangleq |\mathcal{B}|^{-1} \sum_{i \in \mathcal{B}} \mathcal{L}(\mathbf{f}(\mathbf{x}_i; \boldsymbol{\theta}), y_i). \quad (6)$$

$$\begin{aligned}
 \frac{\partial \ell_{\mathcal{B}}}{\partial \mathbf{h}^{(\ell-1)}} &= \frac{\partial \mathbf{h}^{(\ell)}}{\partial \mathbf{h}^{(\ell-1)}} \cdot \frac{\partial \ell_{\mathcal{B}}}{\partial \mathbf{h}^{(\ell)}} \\
 &= (\mathbf{W}^{(\ell)})^{\top} \text{diag} \left(\mathbb{1} \{ \mathbf{W}^{(\ell)} \mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)} \geq \mathbf{0} \} \right) \frac{\partial \ell_{\mathcal{B}}}{\partial \mathbf{h}^{(\ell)}}
 \end{aligned} \quad (7)$$

Back-propagation in computational graphs

$$\mathbf{w}^{(\ell)} \leftarrow \mathbf{w}^{(\ell)} - \eta \frac{\partial \ell_{\mathcal{B}}}{\partial \mathbf{w}^{(\ell)}}, \text{ where } \frac{\partial \ell_{\mathcal{B}}}{\partial w_{jm}^{(\ell)}} = \frac{\partial \ell_{\mathcal{B}}}{\partial h_j^{(\ell)}} \cdot \sigma' \cdot h_m^{(\ell-1)}, \quad (8)$$

Back-propagation in computational graphs

$$\mathbf{w}^{(\ell)} \leftarrow \mathbf{w}^{(\ell)} - \eta \frac{\partial \ell_{\mathcal{B}}}{\partial \mathbf{w}^{(\ell)}}, \text{ where } \frac{\partial \ell_{\mathcal{B}}}{\partial w_{jm}^{(\ell)}} = \frac{\partial \ell_{\mathcal{B}}}{\partial h_j^{(\ell)}} \cdot \sigma' \cdot h_m^{(\ell-1)}, \quad (8)$$

MLP with a single hidden layer and an ℓ_2 regularization:

$$\ell_{\mathcal{B}}^{\lambda}(\boldsymbol{\theta}) = \ell_{\mathcal{B}}(\boldsymbol{\theta}) + r_{\lambda}(\boldsymbol{\theta}) = \ell_{\mathcal{B}}(\boldsymbol{\theta}) + \lambda \left(\sum_{j,j'} (w_{j,j'}^{(1)})^2 + \sum_{j,j'} (w_{j,j'}^{(2)})^2 \right), \quad (9)$$

Back-propagation in computational graphs

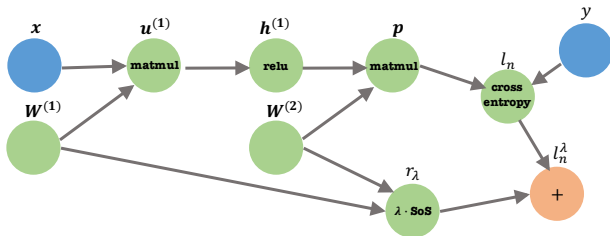


Figure 2: The computational graph illustrates the loss (9). For simplicity, we omit the bias terms. Symbols inside nodes represent functions, and symbols outside nodes represent function outputs (vectors/scalars). **matmul** is matrix multiplication, **relu** is the ReLU activation, **cross entropy** is the cross entropy loss, and **SoS** is the sum of squares.

Numerical experiments

Section 2

Convolutional neural networks



$$O_{ij}^k = \langle [\mathbf{X}]_{ij}, \mathbf{F}_k \rangle = \sum_{i'=1}^w \sum_{j'=1}^w \sum_{l=1}^{d_3} [\mathbf{X}]_{i+i'-1, j+j'-1, l} [\mathbf{F}_k]_{i', j', l}. \quad (10)$$

$$O_{ij}^k = \langle [\mathbf{X}]_{ij}, \mathbf{F}_k \rangle = \sum_{i'=1}^w \sum_{j'=1}^w \sum_{l=1}^{d_3} [\mathbf{X}]_{i+i'-1, j+j'-1, l} [\mathbf{F}_k]_{i', j', l}. \quad (10)$$

$$\tilde{X}_{ijk} = \sigma(O_{ijk}), \forall i \in [d_1 - w + 1], j \in [d_2 - w + 1], k \in [\tilde{d}_3]. \quad (11)$$

Convolutional layer

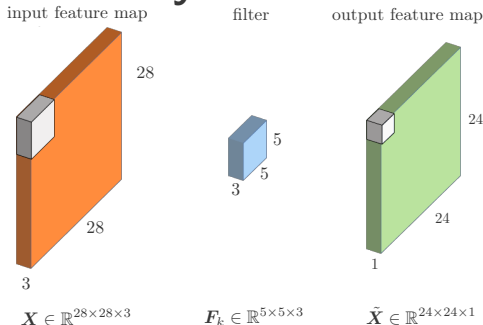


Figure 3: $\mathbf{X} \in \mathbb{R}^{28 \times 28 \times 3}$ represents the input feature consisting of 28×28 spatial coordinates in a total number of 3 channels / feature maps. $\mathbf{F}_k \in \mathbb{R}^{5 \times 5 \times 3}$ denotes the k -th filter with size 5×5 . The third dimension 3 of the filter automatically matches the number 3 of channels in the previous input. Every 3D patch of \mathbf{X} gets convolved with the filter \mathbf{F}_k and this as a whole results in a single output feature map $\tilde{\mathbf{X}}_{:, :, k}$ with size $24 \times 24 \times 1$. Stacking the outputs of all the filters $\{\mathbf{F}_k\}_{1 \leq k \leq K}$ will lead to the output feature with size $24 \times 24 \times K$.

Pooling layer (POOL)

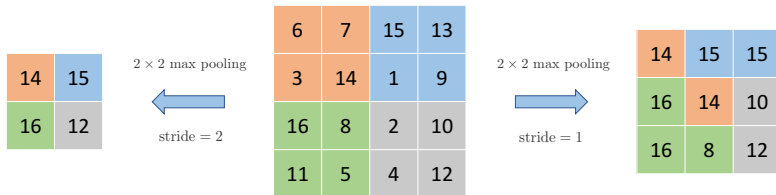


Figure 4: A 2×2 max pooling layer extracts the maximum of 2 by 2 neighboring pixels / features across the spatial dimension.

LeNet

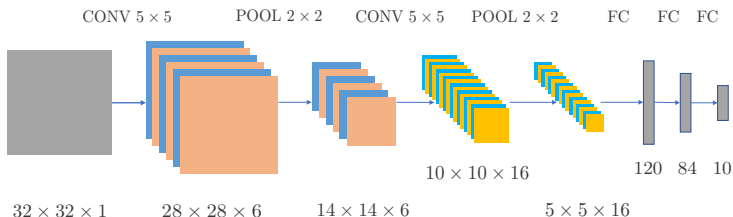


Figure 5: LeNet is composed of an input layer, two convolutional layers, two pooling layers and three fully-connected layers. Both convolutions are valid and use filters with size 5×5 . In addition, the two pooling layers use 2×2 average pooling.

Numerical experiments

Section 3

Recurrent neural networks

Vanilla RNNs

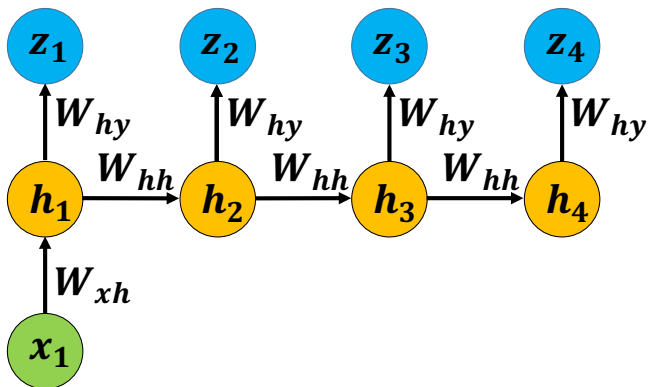


Figure 6: Vanilla RNNs with different inputs/outputs settings. (a) has one input but multiple outputs;

Vanilla RNNs

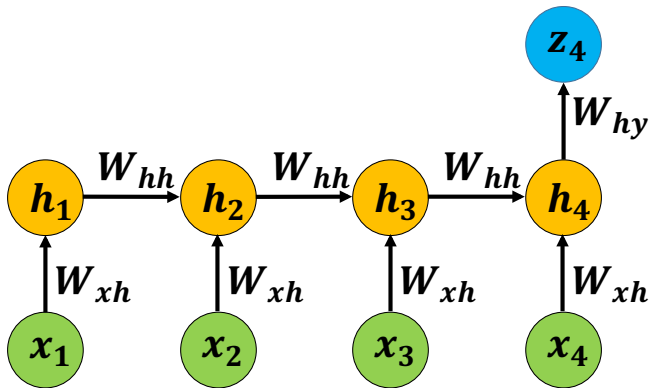


Figure 7: Vanilla RNNs with different inputs/outputs settings. (b) has multiple inputs but one output;

Vanilla RNNs

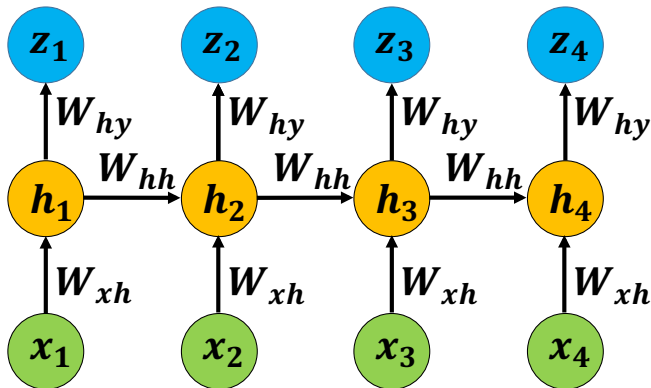


Figure 8: Vanilla RNNs with different inputs/outputs settings. (c) has multiple inputs and outputs. Note that the parameters are shared across time steps.

Vanilla RNNs

$$h_t = f_{\theta}(h_{t-1}, \mathbf{x}_t). \quad (12)$$

Vanilla RNNs

$$\mathbf{h}_t = \mathbf{f}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t). \quad (12)$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h), \quad (13)$$

$$\tanh(a) = \frac{e^{2a} - 1}{e^{2a} + 1}, \quad (14)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_z), \quad (15)$$

Vanilla RNNs

$$\mathbf{h}_t = \mathbf{f}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t). \quad (12)$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_h), \quad (13)$$

$$\tanh(a) = \frac{e^{2a} - 1}{e^{2a} + 1}, \quad (14)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_z), \quad (15)$$

$$\begin{aligned} \ell_{\mathcal{T}}(\boldsymbol{\theta}) &= \sum_{t \in \mathcal{T}} \mathcal{L}(y_t, \mathbf{z}_t) \\ &= - \sum_{t \in \mathcal{T}} \sum_{k=1}^K \mathbb{1}\{y_t = k\} \log \left(\frac{\exp([\mathbf{z}_t]_k)}{\sum_k \exp([\mathbf{z}_t]_k)} \right), \end{aligned} \quad (16)$$

LSTM

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \\ 1 \end{pmatrix}, \quad (17)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \quad (18)$$

$$h_t = o_t \odot \tanh(c_t), \quad (19)$$

Multilayer RNNs

$$\mathbf{h}_t^\ell = \tanh \left[\mathbf{W}^\ell \begin{pmatrix} \mathbf{h}_t^{\ell-1} \\ \mathbf{h}_{t-1}^\ell \\ 1 \end{pmatrix} \right], \quad \text{for all } \ell \in [L], \quad \mathbf{h}_t^0 \triangleq \mathbf{x}_t. \quad (20)$$

Multilayer RNNs

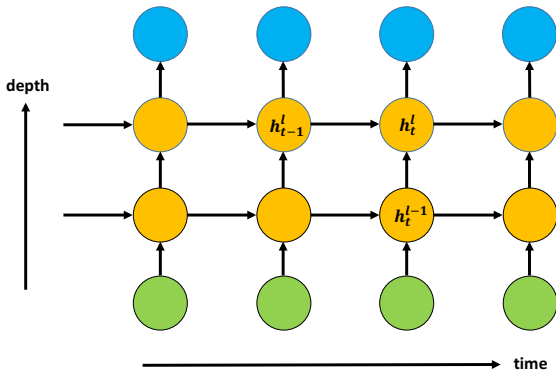
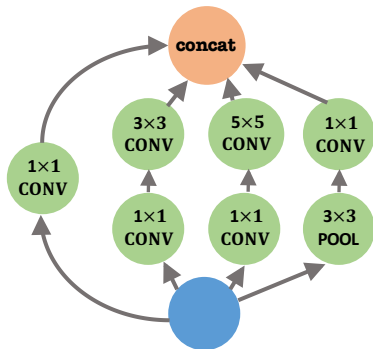


Figure 9: A vanilla RNN with two hidden layers. Higher-level hidden states h_t^{ℓ} are determined by the old states h_{t-1}^{ℓ} and lower-level hidden states $h_t^{\ell-1}$. Multilayer RNNs generalize both feed-forward neural nets and one-hidden-layer RNNs.

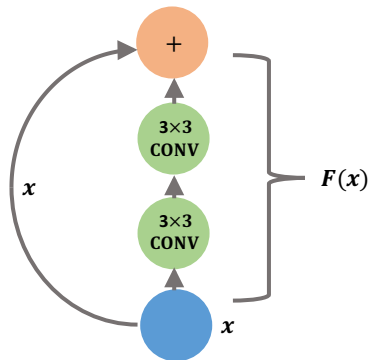
Section 4

Modules

Skip connections



(a) “Inception” module



(b) Skip connections

Figure 10: (a) The “Inception” module from GoogleNet. Concat means combining all features maps into a tensor. (b) Skip connections are added every two layers in ResNets.

Skip connections

$$x \mapsto \sigma(x + \mathbf{F}(x)) = \sigma(x + \mathbf{W}'\sigma(\mathbf{W}x + \mathbf{b}) + \mathbf{b}'), \quad (21)$$

Section 5

Autoencoders

Autoencoders

$$f(x) = W_f x \triangleq h \quad (22)$$

$$g(h) = W_g h, \quad (23)$$

$$W_f \in \mathbb{R}^{k \times d} \text{ and } W_g \in \mathbb{R}^{d \times k}. \quad (24)$$

Autoencoders

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}_f \mathbf{x} \triangleq \mathbf{h} \quad (22)$$

$$\mathbf{g}(\mathbf{h}) = \mathbf{W}_g \mathbf{h}, \quad (23)$$

$$\mathbf{W}_f \in \mathbb{R}^{k \times d} \text{ and } \mathbf{W}_g \in \mathbb{R}^{d \times k}. \quad (24)$$

$$\min_{\mathbf{f}, \mathbf{g}} \quad \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\mathbf{x}_i, \mathbf{g}(\mathbf{h}_i)) \text{ with } \mathbf{h}_i = \mathbf{f}(\mathbf{x}_i), \forall i \in [n]. \quad (25)$$

Autoencoders

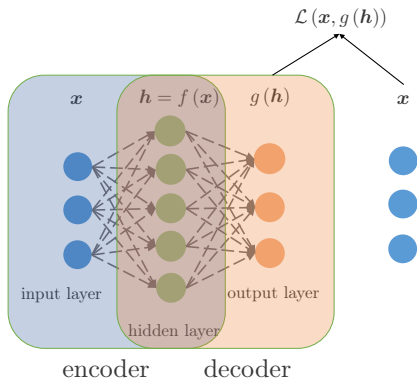


Figure 11: First an input \mathbf{x} goes through the decoder $\mathbf{f}(\cdot)$, and we obtain its hidden representation $\mathbf{h} = \mathbf{f}(\mathbf{x})$. Then, we use the decoder $\mathbf{g}(\cdot)$ to get $\mathbf{g}(\mathbf{h})$ as a reconstruction of \mathbf{x} . Finally, the loss is determined from the difference between the original input \mathbf{x} and its reconstruction $\mathbf{g}(\mathbf{f}(\mathbf{x}))$.

Section 6

Generative adversarial networks

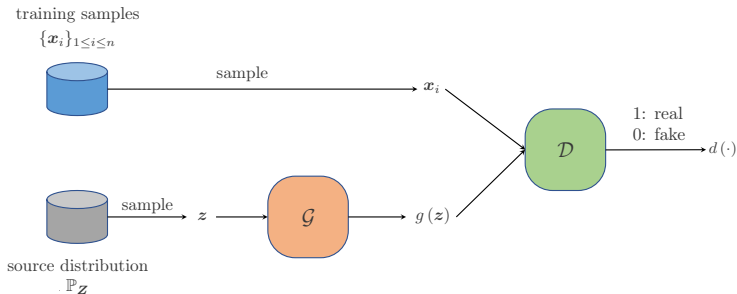


Figure 12: GANs consist of two components, a generator \mathcal{G} which generates fake samples and a discriminator \mathcal{D} which differentiate the true ones from the fake ones.

GAN

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_X} [\log (d(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_Z} [\log (1 - d(\mathbf{g}(\mathbf{z})))] . \quad (26)$$

GAN

$$\min_{\mathbb{P}_{\mathcal{G}}} \max_{d(\cdot)} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{X}}} [\log (d(\mathbf{x}))] + \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{G}}} [\log (1 - d(\mathbf{x}))]. \quad (27)$$

$$\min_{\mathbb{P}_{\mathcal{G}}} \max_{d(\cdot)} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{X}}} [\log (d(\mathbf{x}))] + \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{G}}} [\log (1 - d(\mathbf{x}))]. \quad (27)$$

$$d^*(\mathbf{x}) = \frac{\mathbb{P}_{\mathcal{X}}(\mathbf{x})}{\mathbb{P}_{\mathcal{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}. \quad (28)$$

$$\min_{\mathbb{P}_{\mathcal{G}}} \max_{d(\cdot)} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathbf{X}}} [\log (d(\mathbf{x}))] + \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{G}}} [\log (1 - d(\mathbf{x}))]. \quad (27)$$

$$d^*(\mathbf{x}) = \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}. \quad (28)$$

$$\min_{\mathbb{P}_{\mathcal{G}}} \text{JS}(\mathbb{P}_{\mathbf{X}} \parallel \mathbb{P}_{\mathcal{G}}), \quad (29)$$

where $\text{JS}(\cdot \parallel \cdot)$ denotes the Jensen–Shannon divergence between two distributions

$$\text{JS}(\mathbb{P}_{\mathbf{X}} \parallel \mathbb{P}_{\mathcal{G}}) = \frac{1}{2} \text{KL}(\mathbb{P}_{\mathbf{X}} \parallel \frac{\mathbb{P}_{\mathbf{X}} + \mathbb{P}_{\mathcal{G}}}{2}) + \frac{1}{2} \text{KL}(\mathbb{P}_{\mathcal{G}} \parallel \frac{\mathbb{P}_{\mathbf{X}} + \mathbb{P}_{\mathcal{G}}}{2}). \quad (30)$$

it minimizes the Wasserstein distance between $\mathbb{P}_{\mathbf{X}}$ and $\mathbb{P}_{\mathcal{G}}$:

$$\min_{\theta_{\mathcal{G}}} \text{WS}(\mathbb{P}_{\mathbf{X}} \parallel \mathbb{P}_{\mathcal{G}}) = \min_{\theta_{\mathcal{G}}} \sup_{f: f \text{ 1-Lipschitz}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathbf{X}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathcal{G}}} [f(\mathbf{x})], \quad (31)$$

Questions