

# Neural Networks I

Wei Li

Syracuse University

Spring 2024

# OVERVIEW

Projection Pursuit Regression

Single Hidden Layer Neural Networks

Deep Neural Network

Stochastic Gradient Descent

Backpropagation

# Projection Pursuit Regression

# Projection Pursuit Regression

Let  $X$  be a  $p$ -dimensional predictors.  $Y$  be some target response. The PPR function takes the form

$$f(X) = \sum_{m=1}^M g_m(\omega_m^\top X)$$

- ▶  $\omega_m : 1, \dots, M$  unit  $p$ -vector (directions) unknown,  $\|\omega\| = 1$
- ▶  $g_m$  unknown,  $g_m(\omega_m^\top X)$  is called a ridge function in  $\mathbb{R}^p$
- ▶ The scalar variable  $V_m = \omega_m^\top X$  is the projection of  $X$  onto the unit vector  $\omega_m$

The parameters are  $\{g_m, \omega_m\}_{m=1}^M$ .

The model takes the form

$$Y_i = \sum_{m=1}^M g_m (\omega_m^\top X_i) + \epsilon_i$$

Given data  $\{Y_i, X_i\}_{i=1}^n$ , the estimates are defined by

$$\min_{g_m, \omega_m} \sum_{i=1}^n \left( Y_i - \sum_{m=1}^M g_m (\omega_m^\top X_i) \right)^2$$

Note: the PPR model is a generalization of the so-called *one-hidden layer neural network* when  $g_m$ 's are assumed to be known

$$g_m (\omega_m^\top X) = \beta_m \sigma (\alpha_{m0} + \alpha_m^\top X) = \beta_m \sigma (\alpha_{m0} + \|\alpha_m\| (\omega_m^\top X))$$

where  $\omega_m = \alpha_m / \|\alpha_m\|$  is the  $m$ -th unit-vector.

# Single-index model

When  $m = 1$  (single-index model), the parameters can be estimated via a recursive algorithm:

1. Given  $\omega$ , forms  $\nu_i = \omega^\top x_i$ , solve for  $g$  (one-dimensional smoothing).
2. Given  $g$ , minimize the loss over  $\omega$  through iteration (quasi-Newton): note that

$$g(\omega^\top x_i) \approx g(\omega_{\text{old}}^\top x_i) + g'(\omega_{\text{old}}^\top x_i) (\omega - \omega_{\text{old}})^\top x_i$$

Thus  $\sum_{i=1}^n (y_i - g(\omega^\top x_i))^2 \approx$

$$\sum_{i=1}^n g'(\omega_{\text{old}}^\top x_i)^2 \left( \left( \omega_{\text{old}}^\top x_i + \frac{y_i - g(\omega_{\text{old}}^\top x_i)}{g'(\omega_{\text{old}}^\top x_i)} \right) - \omega^\top x_i \right)^2$$

$m > 1$

Forward stage-wise manner: adding a pair  $(\omega_m, g_m)$  at each stage (Forward Stagewise Algorithm 10.2). It is hinted that the backfitting algorithm (Algorithm 9.1) may be also applied for each step  $m$  to readjust previous  $g$  terms.

- ▶ Backfitting algorithm: for each  $m$  in turn, minimize

$$\min_{g_m, \omega_m} \sum_{i=1}^n (r_i - g_m(\omega_m^\top X_i))^2$$

where  $r_i = Y_i - \sum_{l \neq m} g_l(\omega_l^\top X_i)$ .

- ▶ The number of terms  $M$ 
  - ▶ The model building stops when the next term does not appreciably improve the fit
  - ▶ Or chosen using cross-validation.

# Single Hidden Layer Neural Networks



# Single Hidden Layer Neural Networks

- ▶ Input:  $X = (X_1, \dots, X_p)$
- ▶ Output:  $\hat{Y} = (\hat{Y}_1, \dots, \hat{Y}_K)$ 
  - ▶ Continuous outcomes: typically  $K = 1$ , but it can handle multiple responses.
  - ▶  $K$ -class classification:  $\hat{Y}_k$  be the probability of class  $k$ , or  $\hat{Y}_k \in \{0, 1\}$ .

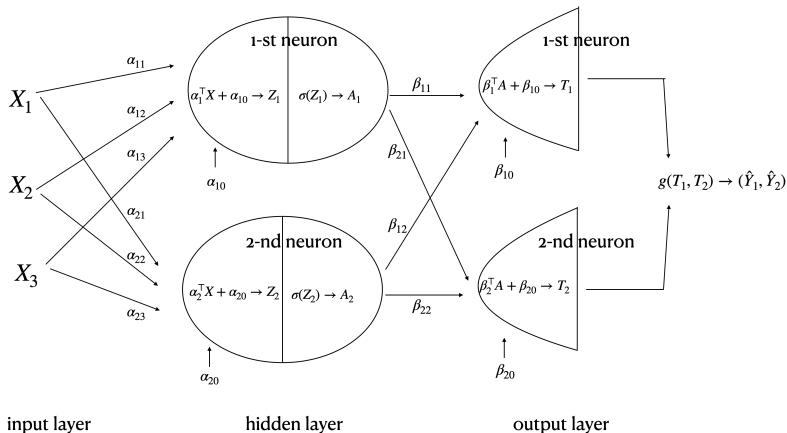
$$Z_m = \alpha_{m0} + \alpha_m^\top X, m = 1, \dots, M$$

$$A_m = \sigma(Z_m), m = 1, \dots, M$$

$$T_k = \beta_{k0} + \beta_k^\top A, k = 1, \dots, K, \text{ here } A = (A_1, \dots, A_M)$$

$$\hat{Y}_k = f_k(X) := g_k(T), k = 1, \dots, K, \text{ here } T = (T_1, \dots, T_K)$$

- ▶  $Z = (Z_1, Z_2, \dots, Z_M)$ : pre-activation values (net input)
- ▶  $A = (A_1, \dots, A_M)$ : activation values (derived features)
- ▶  $T = (T_1, T_2, \dots, T_K)$ : pre-output values
- ▶  $f = (f_1(X), \dots, f_K(X))$ : output (prediction).



### Example

- ▶  $p = 3$ : three predictors
- ▶  $M = 2$ : two hidden units in the hidden layer
- ▶  $K = 2$ : two output units in the output layer
- ▶ softmax output

# Activation function

$\sigma(\cdot)$  is an activation function:

- i. sigmoid  $\sigma(v) = 1 / (1 + e^{-v})$
- ii. ReLU  $\sigma(v) = \max(v, 0)$
- iii. leakyReLU  $\sigma(v) = v1\{v > 0\} + av1\{v \leq 0\}$  (say,  $a = 0.01$ )
- iv. tanh (Hyperbolic Tangent)  $\sigma(v) = (e^v - e^{-v}) / (e^v + e^{-v})$
- v. Gaussian radial basis function  
 $\sigma(x; \mu_m, \lambda_m) = \exp(-\|x - \mu_m\|^2 / 2\lambda_m)$

See next few slides for more details.

# Output function $g$

The output  $g(T) = (\dots, g_k(T), \dots)$  is a final transformation of the vector of outputs  $T$ .

- ▶ For regression, just an identity function  $g_1(T) = T_1$ , ( $K = 1$ ).
- ▶ Binary classification:  $g_1(T) = \text{sigmoid}(T_1)$ , ( $K = 1$ ).
- ▶ For  $K$ -class classification, the softmax function

$$g_k(T) = \mathbf{S}_k := \frac{e^{T_k}}{\sum_{\ell=1}^K e^{T_\ell}}, k = 1, \dots, K$$

which gives a vector of estimated probabilities.

- ▶ The vector  $T$ : **logits**.
- ▶ The corresponding classifier:  $G(X) = \arg \max_k g_k(T)$ .

The complete set of parameters  $\theta$  consists of

$\{\alpha_{m0}, \alpha_m; m = 1, 2, \dots, M\}$   $M(p+1)$  parameters

$\{\beta_{k0}, \beta_k; k = 1, 2, \dots, K\}$   $K(M+1)$  parameters

- ▶ regression:  $\mathcal{L}(\theta) = \sum_{k=1}^K \sum_{i=1}^n (y_{ik} - f_k(x_i))^2$
- ▶ classification:  $\mathcal{L}(\theta) = - \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log f_k(x_i)$ ,  
 $y_{ik} := 1(y_i = k)$ .
  - ▶ The corresponding classifier is  $G(x) = \operatorname{argmax}_k f_k(x)$ .

Find minimizer of  $\mathcal{L}(\theta)$  by **gradient descent**:

$$\theta \leftarrow \theta - \gamma \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

- ▶ *gradients* derived using the chain rule of differentiation
  - ▶ called “**back-propagation**”

## A single-hidden layer NN

$K$  outputs, squared error loss

$$\mathcal{L}(\theta) \equiv \sum_{i=1}^n \mathcal{L}_i = \sum_{i=1}^n \sum_{k=1}^K (y_{ik} - f_k(x_i))^2,$$

Let  $\tilde{x}_i = (1, x_i^\top)^\top$ ,  $a_{mi} = \sigma(\alpha_m^\top \tilde{x}_i)$ ,

$a_i = (a_{1i}, a_{2i}, \dots, a_{Mi})^\top$ ,  $\tilde{a}_i = (1, a_{1i}, a_{2i}, \dots, a_{Mi})^\top$ .

$\alpha_m := \{\alpha_{m0}, \alpha_{m1}, \alpha_{m2}, \dots, \alpha_{mp}\}$

$\beta_k := \{\beta_{k0}, \beta_{k1}, \beta_{k2}, \dots, \beta_{kM}\}$

The derivatives are given by, for  $i = 1, \dots, n$ ,

$$\frac{\partial \mathcal{L}_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g'_k(\beta_k^\top \tilde{a}_i) \tilde{a}_{mi}$$

$$\frac{\partial \mathcal{L}_i}{\partial \alpha_{ml}} = - \sum_{k=1}^K 2(y_{ik} - f_k(x_i)) g'_k(\beta_k^\top \tilde{a}_i) \beta_{km} \sigma'(\alpha_m^\top \tilde{x}_i) \tilde{x}_{il}$$

The “errors” (wrt net input) at the output layer and hidden layer

$$\partial t_{ki} := \partial \mathcal{L}_i / \partial T_k = -2 (y_{ik} - f_k(x_i)) g'_k(\beta_k^\top \tilde{a}_i)$$

$$\partial z_{mi} := \partial \mathcal{L}_i / \partial Z_m = \sigma'(\alpha_m^\top \tilde{x}_i) \sum_{k=1}^K \beta_{km} \partial t_{ki}$$

The gradients w.r.t parameters can be written as

$$\frac{\partial \mathcal{L}_i}{\partial \beta_{km}} = (\partial t_{ki}) \tilde{a}_{mi}, \quad \frac{\partial \mathcal{L}_i}{\partial \alpha_{ml}} = (\partial z_{mi}) \tilde{x}_{il}$$

In the  $r$ -th iteration,

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma^{(r)} \sum_{i=1}^n \frac{\partial \mathcal{L}_i}{\partial \beta_{km}^{(r)}}, \quad \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma^{(r)} \sum_{i=1}^n \frac{\partial \mathcal{L}_i}{\partial \alpha_{ml}^{(r)}}$$

# Forward and Backpropagation Algorithm

At the  $r$ -th iteration:

1. Forward pass: Given current estimate  $\beta_k^{(r)}, \alpha_m^{(r)}$ , compute  $a_{mi}^{(r)} = \sigma \left( \tilde{x}_i^\top \alpha_m^{(r)} \right)$  and  $f_k^{(r)}(x_i)$ .
2. Backward pass:

$$\partial t_{ki}^{(r)} = -2 \left( y_{ik} - f_k^{(r)}(x_i) \right) g'_k \left( \beta_k^{(r)\top} \tilde{a}_i^{(r)} \right)$$

$$\partial z_{mi}^{(r)} = \sigma' \left( \tilde{x}_i^\top \alpha_m^{(r)} \right) \sum_{k=1}^K \beta_{km}^{(r)} \partial t_{ki}^{(r)}.$$

3. Update

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma^{(r)} \sum_{i=1}^n (\partial t_{ki}^{(r)}) \tilde{a}_{mi}^{(r)},$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma^{(r)} \sum_{i=1}^n (\partial z_{mi}^{(r)}) \tilde{x}_{il}$$



# Issues concerning activation functions

When there are multiple hidden layers (*deeper network*)...

**Exploding gradients:** large error gradients accumulate, resulting in very large updates; may be alleviated through *gradient clipping*.

**Vanishing gradients:** small error gradients accumulate, resulting in very small (or zero!) updates; may be alleviated through careful choice of activation functions.

- ▶ sigmoid: the output values between  $(0, 1)$ , would be very near 0 when the input values are strongly negative and vice versa; gradient almost near to zero.
- ▶ tanh: the output values between  $(-1, 1)$ , has stronger gradients than sigmoid, still have vanishing gradient issues
- ▶ Relu: preserves the gradient for positive  $Z$ , thus alleviate the vanishing gradient problem; only saturates when the input is negative
- ▶ leaky Relu: improves upon Relu by recovering the negative region

# activation functions

## Sigmoid

$$\sigma(Z) = \frac{1}{1 + e^{-Z}}, \quad \frac{d}{dZ}\sigma(Z) = \sigma(Z)(1 - \sigma(Z))$$

## tanh

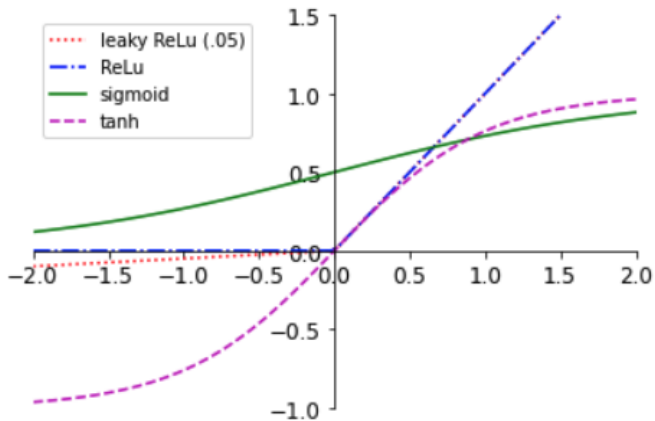
$$\tanh(Z) = \frac{e^Z - e^{-Z}}{e^Z + e^{-Z}}, \quad \frac{d}{dZ}\tanh(Z) = 1 - \tanh^2(Z)$$

## Rectified Linear Unit

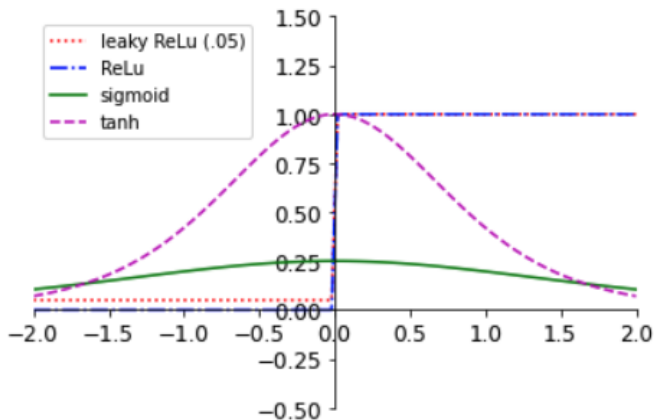
$$\text{ReLu}(Z) = \max(Z, 0), \quad \frac{d}{dZ}\text{relu}(Z) = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{otherwise} \end{cases}$$

## Leaky Rectified Linear Unit

$$\text{LReLu}(Z) = \begin{cases} Z & \text{if } Z > 0 \\ aZ & \text{otherwise} \end{cases}, \quad \frac{d}{dZ}\text{LReLu}(Z) = \begin{cases} 1 & \text{if } Z > 0 \\ a & \text{otherwise} \end{cases}$$



Plots of sigmoid, tanh, ReLu and leaky ReLu.



Plots of the derivatives of sigmoid, tanh, ReLu and leaky ReLu.

See here for more options: [https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)

## softmax activation

- ▶  $K$ -categories
- ▶ an  $K$ -dimensional vector  $z = (z_1, z_2, \dots, z_K)$  (**logits**)
- ▶ the softmax function produces another  $K$ -dimensional vector with values in the range  $[0,1]$

$$z \mapsto \mathbf{S}(z) = (p_1, \dots, p_K)$$

where  $p_i := P(y = i|z) = \mathbf{S}(z)_i := \frac{\exp^{z_i}}{\sum_{k=1}^K \exp^{z_k}}, \forall j \in 1, \dots, K$ .

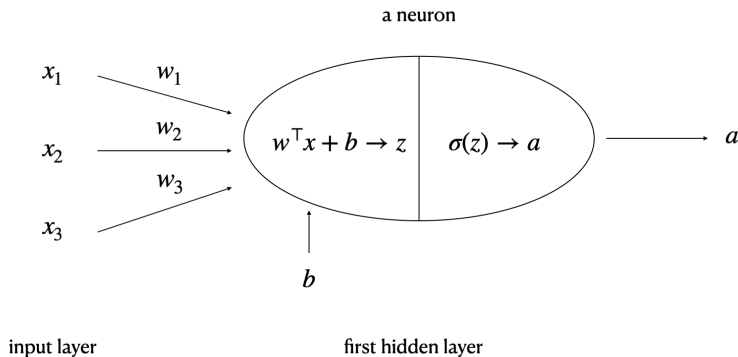
Derivatives are

$$\frac{\partial p_i}{\partial z_j} = \begin{cases} p_i(1 - p_j) & \text{if } i = j \\ -p_i p_j & \text{if } i \neq j \end{cases}$$

Note: softmax activation is one example of the so-called *vector activation*, as opposed to the elementwise activations. It is most commonly used in the output layer for multi-class classification.

# Deep Neural Network

# Deep Neural Network



a neuron: 3 input features

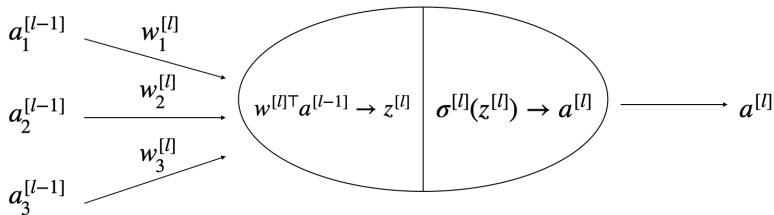
- ▶ parameters
  - ▶ weights:  $w_1, w_2, w_3$
  - ▶ “bias”:  $b$

# notations

1. For inputs  $x \in \mathbb{R}^p$  ( $p$  features)
  - ▶ superscript  $i$ : is a reference to the  $i$  th observation; i.e.,  $x^{(1)}$  is the input value of the first observation.
  - ▶ subscript  $j$ : is a reference to the  $j$ -th feature number; i.e.,  $x_2^{(1)}$  is the first observation of the second feature.
2. For network layers:
  - ▶ superscript  $[l]$ : refers to layer  $l$ ; i.e.,  $z^{[1]}$  is the net input at layer 1.
  - ▶ subscript  $j$ : is a reference to the node number; i.e.,  $z_1^{[2]}$  is the net input at the first node in layer 2.
3.  $n^{[\ell]}$  denotes the number of units in the  $\ell$ -th layer.



a neuron in the  $\ell$ -th layer

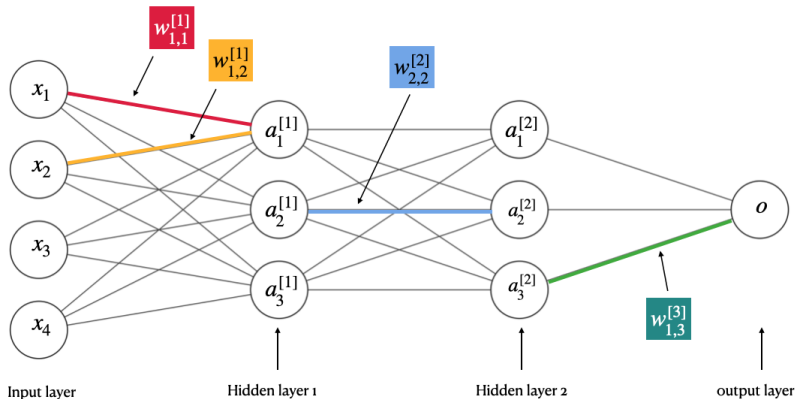


bias unit is omitted for simplicity

a neuron in the  $\ell$ -th layer: 3 (derived) input features

## Example

Assume four features ( $p = 4$ ), two observations ( $n = 2$ )



A three-layer neural network (two-hidden-layer neural network )

►  $n^{[0]} = p, n^{[1]} = 3, n^{[2]} = 3, n^{[3]} = 1.$

## example: bias in layer 1

There are 3 nodes in hidden layer 1. The shape of the bias matrix  $b^{[1]}$  are ( number of nodes in layer 1 ).

The bias matrix:

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

## example: weights in layer 1

A general data matrix  $X$  ( $n \times p$ ):

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_p^{(n)} \end{bmatrix}$$

The weights matrix can be represented as

$$W^{[1]} = \begin{bmatrix} w_1^{[1]\top} \\ w_2^{[1]\top} \\ w_3^{[1]\top} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} & w_{1,4}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} & w_{2,4}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} & w_{3,4}^{[1]} \end{bmatrix}$$

- ▶  $w_i^{[\ell]}$  to denote the vector of weights for the  $i$ -th node in the layer  $\ell$
- ▶  $w_{i,j}^{[\ell]}$  to denote the  $j$ -th coordinate of  $w_i^{[\ell]}$ , i.e., the weight for the connection from the  $j$ -th node in the layer  $\ell - 1$  to the  $i$ -th node in the layer  $\ell$ .

## example: pre-activation in layer 1

We have

$$\mathbf{X}W^{[1]\top} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \end{bmatrix} \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} & w_{1,4}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} & w_{2,4}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} & w_{3,4}^{[1]} \end{bmatrix}^\top$$

The pre-activation at layer-1 can be written as

$$\mathbf{Z}^{[1]} = \mathbf{X}W^{[1]\top} + b^{[1]\top}, \quad (2 \times 3)$$

$$\mathbf{Z}^{[1]} = \begin{bmatrix} z_{1,1}^{[1]} & z_{1,2}^{[1]} & z_{1,3}^{[1]} \\ z_{2,1}^{[1]} & z_{2,2}^{[1]} & z_{2,3}^{[1]} \end{bmatrix}.$$

## example: activation in layer 1

The activation is applied *elementwise*:

The activation matrix  $\mathbf{A}^{[1]}$  has the same shape as  $\mathbf{Z}^{[1]}$

$$\mathbf{A}_{[1]} = \varphi(\mathbf{Z}) = \begin{bmatrix} \varphi(z_{1,1}^{[1]}) & \varphi(z_{1,2}^{[1]}) & \varphi(z_{1,3}^{[1]}) \\ \varphi(z_{2,1}^{[1]}) & \varphi(z_{2,2}^{[1]}) & \varphi(z_{2,3}^{[1]}) \end{bmatrix},$$

where  $\varphi$  is an activation function (i.e., sigmoid, etc.)

## example: output layer

Layer-2 can be similarly written...

Turn to output, assume considering a binary classification (sigmoid activation in the output layer).

- ▶ input to the output layer is  $\mathbf{A}^{[2]}$ , i.e.,  $(2, 3)$ .
- ▶ for some  $W$  of shape  $1 \times 3$ :

$$\mathbf{Z}^{[3]} = \mathbf{A}^{[2]} W^{[3]\top}$$

$$g^{[3]} = \sigma(\mathbf{Z}^{[3]})$$

$g^{[3]}$  produces a vector of two predicted values, one for each observation.

## example: in summary

All layers can be generalized as

$$\mathbf{Z}^{[\ell]} = \mathbf{A}^{[\ell-1]} W^{[\ell]\top} + b^{[\ell]\top}$$

where

$$W^{[\ell]} = \begin{bmatrix} w_1^{[\ell]\top} \\ w_2^{[\ell]\top} \\ \vdots \\ w_{n^{[\ell]}}^{[\ell]\top} \end{bmatrix}.$$

- ▶ shape of  $W^{[\ell]}$  is  $(n^{[\ell]}, n^{[\ell-1]})$
- ▶ shape of  $\mathbf{Z}^{[\ell]}$  (or  $\mathbf{A}^{[\ell]}$ ) is  $n \times n^{[\ell]}$



## In summary: 3 layers

The forward propagation equations for a three-layer network for a *single observation* can be represented as

$$\begin{aligned}z^{[1](i)} &= W^{[1]}a^{[0](i)} + b^{[1](i)}, a^{[0](i)} := x^{(i)} \\a^{[1](i)} &= g^{[1]} \left( z^{[1](i)} \right) \\z^{[2](i)} &= W^{[2]}a^{[1](i)} + b^{[2](i)} \\a^{[2](i)} &= g^{[2]} \left( z^{[2](i)} \right) \\z^{[3](i)} &= W^{[3]}a^{[2](i)} + b^{[3](i)} \\a^{[3](i)} &= \hat{y}^{(i)} = g^{[3]} \left( z^{[3](i)} \right), g^{[3]}(\cdot) = \text{sigmoid}(\cdot)\end{aligned}$$

The shape of  $W^{[\ell]}$  is  $(n^{[\ell]}, n^{[\ell-1]})$ .

## In summary: general ( $L$ layers)

- ▶ A multilayer perceptron with  $L - 1$  hidden layers ( $L$  layers):
  - ▶ Input  $a^{[0]}(x) = x$ .
  - ▶ For  $\ell = 1, \dots, L - 1$  (hidden layers),

$$z^{[\ell]}(x) = b^{[\ell]} + W^{[\ell]} a^{[\ell-1]}(x)$$

$$a^{[\ell]}(x) = g^{[\ell]}(z^{[\ell]}(x))$$

where  $g^{[\ell]}$  is the activation function.

- ▶ For  $\ell = L$  (output layer),

$$z^{[L]}(x) = b^{[L]} + W^{[L]} a^{(L-1)}(x)$$

$$f(x, \theta) \equiv a^{[L]}(x) = g^{[L]}(z^{[L]}(x))$$

where  $g^{[L]}$  is the output activation function.

- ▶ Parameters  $\theta : b^{[\ell]}$  (biases) and  $W^{[\ell]}$  (weights),  $\ell = 1, \dots, L$ .  
 $\{b^{[\ell]}, W^{[\ell]}\}$  called parameters in the  $\ell$ -th layer.

Combining  $n$  observations (using matrices),

	W	b	Pre-activation	Z	# of paras
Layer 1	$(n^{[1]}, p)$	$(n^{[1]}, 1)$	$Z^{[1]} = XW^{[1]\top} + b^{[1]\top}$	$(n, n^{[1]})$	$n^{[1]}(1 + p)$
Layer $\ell$	$(n^{[\ell]}, n^{[\ell-1]})$	$(n^{[\ell]}, 1)$	$Z^{[\ell]} = A^{[\ell-1]}W^{[\ell]\top} + b^{[\ell]\top}$	$(n, n^{[\ell]})$	$n^{[\ell]}(1 + n^{[\ell-1]})$
Layer $L$	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = A^{[L-1]}W^{[L]\top} + b^{[L]\top}$	$(n, n^{[L]})$	$n^{[L]}(1 + n^{[L-1]})$

- ▶ In  $L$ -th layer (the output layer), then  $\hat{y} = g^{[L]}(Z^{[L]})$  is of shape  $n \times n^{[L]}$ .
- ▶ Depending on the output function  $A$ 
  - ▶ for LS and binary classification,  $n^{[L]} = 1$
  - ▶ for K-class classification,  $n^{[L]} = K$ .

## Penalized empirical risk

For  $\ell = L$  (output layer),

$$\begin{aligned}z^{[L]}(x) &= b^{[L]} + W^{[L]}a^{(L-1)}(x) \\f(x, \theta) &\equiv a^{[L]}(x) = g^{[L]}(z^{[L]}(x))\end{aligned}$$

where  $g^{[L]}$  is the output activation function.

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

where  $\Omega(\theta)$  is some penalty function of  $\theta$ , say  $\ell_1$  or  $\ell_2$ -penalty of the weights.

## Loss functions (classification)

**Cross-entropy** (CE) loss, or log-loss, measures the performance of a classification model whose output has a probability value between 0 and 1.

As the predicted probability diverges from the actual value the CE loss consequently increases.

The cross-entropy loss is

$$H(p, \hat{p}) = \sum_k^K p_k \log \frac{1}{\hat{p}_k} = - \sum_k p_k \log (\hat{p}_k)$$

where,  $p_k$  is the true probability distribution and  $\hat{p}_k$  is the computed probability distribution.

Binary classification:

$$CE_{\text{Loss}} = H(p, \hat{p}) = -(p \log(\hat{p}) + (1-p) \log(1-\hat{p})) \text{ binary classification}$$

Multi-class classification:  $K(\geq 2)$ -labels,

$$CE_{\text{Loss}} = H(p, \hat{p}) = - \sum_{k=1}^K p_k \log(\hat{p}_k) \text{ multiclass classification}$$

Minimize the CE loss is closely related to minimizing negative log-likelihood.

## Negative Log-likelihood (deviance)

The **binomial deviance** (negative loglikelihood) is

$$\mathcal{L}_n = \frac{1}{n} \sum_{i=1}^n \mathcal{L}^{(i)} = -\frac{1}{n} \sum_{i=1}^n \left( y^{(i)} \log \left( \hat{y}^{(i)} \right) + \left( 1 - y^{(i)} \right) \log \left( 1 - \hat{y}^{(i)} \right) \right)$$

The K-class **multinomial deviance** (negative loglikelihood) is

$$\mathcal{L}_n = \frac{1}{n} \sum_{i=1}^n \mathcal{L}^{(i)} = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_k^{(i)} \log \left( \hat{y}_k^{(i)} \right)$$

where  $y_k^{(i)} = 1(y^{(i)} = k)$  and  $\hat{y}^{(i)} = (\hat{y}_1^{(i)}, \dots, \hat{y}_K^{(i)})$  is a vector of estimated probabilities for  $i$ -th observation.

Some authors simply called them binary cross-entropy and K-class cross-entropy respectively.

For the binary CE loss with sigmoid output function  $\hat{y} = \sigma(z)$ ,

$$\begin{aligned}\mathcal{L}_{\text{sigmoid}} &= -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \\ \frac{\partial \mathcal{L}_{\text{sigmoid}}}{\partial z} &= \frac{\partial \mathcal{L}_{\text{sigmoid}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = - \left( \frac{y}{\hat{y}} - \frac{(1 - y)}{1 - \hat{y}} \right) \sigma(z)(1 - \sigma(z))\end{aligned}$$

For K-class CE loss with the softmax output  $\hat{y} = \text{softmax}(z)$ :

$$\begin{aligned}\mathcal{L}_{\text{softmax}} &= - \sum_k^K y_k \log(\hat{y}_k) \\ \frac{\partial \mathcal{L}_{\text{softmax}}}{\partial z_j} &= - \sum_k y_k \frac{\partial \log(\hat{y}_k)}{\partial z_j} = \hat{y}_j - y_j\end{aligned}$$

Note that  $y$  is a one-hot encoded K-dim vector.



# Stochastic Gradient Descent

# Stochastic Gradient Descent

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

- ▶ Let  $\theta := (W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$ . Initialize  $\theta^{<0>}$ .
- ▶ GD update: at the  $t$ -th iteration

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

- ▶  $\mathcal{B}$  is a subset or a batch of cardinality  $B$  (**batch size**)
  - ▶ (taken at random without replacement from training data)
- ▶ **(full batch) gradient descent**:  $B = n$ .
- ▶ **stochastic gradient descent**:  $B = 1$ .
- ▶ **mini batch learning**: the number of batches  $= n/B > 1$ .

In the **mini-batch gradient descent** method, the parameters are updated based only on the current mini-batch (*one step*), and continue iterating over *all* the mini-batches till we have seen the entire data set.

The process of iterating through the *whole* training dataset is referred to as an **epoch**.

- ▶ **number of epochs:** the number of times the algorithm “sees” the entire training data.
  - ▶ So one epoch would take  $n/B$  steps.
- ▶ **total number of steps:** the total number of epochs multiplies the number of batches ( $n/B$ ).

In mini-batch gradient descent, within each epoch, the following steps are used to create minibatches

- ▶ Shuffle - Shuffle the data set  $(X, Y)$  values randomly.
- ▶ Partition - Partition the data set into the defined mini-batch size set.

The general rule for mini-batch size is

- ▶ use 64, 128, 256, 512,  $\dots$ , i.e., a power of 2

Shuffle is usually required during the training stage, but often not required during the test stage (or *inference*).

# Backpropagation

# Backpropagation

To compute  $\nabla_{\theta} \mathcal{L}(f(X, \theta), Y)$ . Define **error gradient**

$$\nabla_{z^{[\ell]}} \mathcal{L} = \nabla_{z^{[\ell]}} \mathcal{L}(\hat{f}, y), \quad n^{[\ell]} \times 1$$

- For output layer  $L$ , we compute  $\nabla_{z^{[L]}} \mathcal{L} = \nabla_{z^{[L]}} \mathcal{L}(\hat{f}, y)$  by direct computation, or by applying chain rule (if  $g^{[L]}$  is elementwise function)

$$\nabla_{z^{[L]}} \mathcal{L} = \left(g^{[L]}\right)' \left(z^{[L]}\right) \otimes \nabla_{\hat{f}} \mathcal{L}(\hat{f}, y)$$

Note  $\left(g^{[L]}\right)' \left(z^{[L]}\right)$  denotes the elementwise derivative of  $g^{[L]}$  w.r.t.  $z^{[L]}$  elementwise.

- For  $\ell = L, L-1, \dots, 1$ , we have

$$\nabla_{W^{[\ell]}} \mathcal{L} = (\nabla_{z^{[\ell]}} \mathcal{L})(a^{[\ell-1]})^{\top}, \quad n^{[\ell]} \times n^{[\ell-1]}$$

$$\nabla_{b^{[\ell]}} \mathcal{L} = \nabla_{z^{[\ell]}} \mathcal{L}, \quad n^{[\ell]} \times 1$$

$$\nabla_{a^{[\ell-1]}} \mathcal{L} = (W^{[\ell]})^{\top} \nabla_{z^{[\ell]}} \mathcal{L}$$

$$\nabla_{z^{[\ell-1]}} \mathcal{L} = \left(g^{[\ell-1]}\right)' \left(z^{[\ell-1]}\right) \otimes \nabla_{a^{[\ell-1]}} \mathcal{L}, \quad \text{if } \ell \geq 2$$

## vector form

The vectorized form to include **all** observations for gradients of  $\mathcal{L}_n = \frac{1}{n} \sum_i \mathcal{L}^{(i)}$  (denoted by  $d\cdot$ ):

- ▶  $d\mathbf{Z}^{[\ell]}$  be defined so that the  $i$ -th row of  $d\mathbf{Z}^{[\ell]}$  is  $(\nabla_{\mathbf{z}^{[\ell]}(i)} \mathcal{L}^{(i)})^\top$
- ▶ similarly for  $d\mathbf{A}^{[\ell-1]}$

$$d\mathbf{Z}^{[\ell]} = d\mathbf{A}^{[\ell]} \otimes \left(g^{[\ell]}\right)' \left(\mathbf{Z}^{[\ell]}\right), \text{ shape } (n, n^{[\ell]})$$

$$dW^{[\ell]} = \frac{1}{n} (d\mathbf{Z}^{[\ell]})^\top \mathbf{A}^{[\ell-1]}, \text{ shape } (n^{[\ell]}, n) \times (n, n^{[\ell-1]})$$

$$db^{[\ell]} = \frac{1}{n} \text{row Sums} \left( (d\mathbf{Z}^{[\ell]})^\top \right), \text{ shape } (n^{[\ell]}, 1)$$

$$d\mathbf{A}^{[\ell-1]} = d\mathbf{Z}^{[\ell]} W^{[\ell]}, \text{ shape } (n, n^{[\ell-1]})$$

$\otimes$ : elementwise operation

# Some issues in training deep NN

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

- ▶ the choice of activation functions (seen before)
- ▶ the choice of learning rates (or optimization algorithm)
- ▶ standardization of the inputs (and activations)
- ▶ the initialization of the weights

The complexity of the model needs special attention to avoid overfitting,

- ▶ early stopping
- ▶ regularization via penalty
- ▶ dropout
- ▶ residual connection
- ▶ data augmentation
- ▶ the architecture of the network (width/depth)