Neural Networks I

Wei Li

Syracuse University

Spring 2021

Projection Pursuit Regression

Single Hidden Layer Neural Networks

Deep Neural Network

Stochastic Gradient Descent

Back-propagation

Projection Pursuit Regression

Projection Pursuit Regression

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

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

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

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

The model takes the form

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

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

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

Single-index model

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

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

$$g\left(\omega^T x_i\right) \approx g\left(\omega_{\text{old}}^T x_i\right) + g'\left(\omega_{\text{old}}^T x_i\right)\left(\omega - \omega_{\text{old}}\right)^T x_i$$

Thus

$$\sum_{i=1}^{n}\left[y_i - g\left(\omega^T x_i\right)\right]^2 \approx$$

$$\sum_{i=1}^{n} g'\left(\omega_{\text{old}}^T x_i\right)^2\left[\left(\omega_{\text{old}}^T x_i + \frac{y_i - g\left(\omega_{\text{old}}^T x_i\right)}{g'\left(\omega_{\text{old}}^T x_i\right)}\right) - \omega^T x_i\right]^2$$

m>1

Forward stage-wise manner: adding a pair (ω_m, g_m) at each stage (see backfitting algorithm Algorithm 9.1, and Forward Stagewise Algorithm 10.2).

- Backfitting algorithm: for each m in turn, minimize

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

  where $r_i = Y_i - \sum_{l \neq m} g_l \left(\omega_l^T X_i \right)$.
- 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: $X = (X_1, \cdots, X_p)^T$
- Output: $Y_1, \cdots, Y_K$
  - Continuous outcomes: typically $K = 1$, but it can handle multiple responses.
  - K -class classification: Y_k be the probability of class k, or $Y_k \in \{0, 1\}$.

$$
\begin{aligned}
Z_m &= \alpha_{m0} + \alpha_m^T X, m = 1, \ldots, M \\
A_m &= \sigma\left(Z_m\right), m = 1, \ldots, M \\
T_k &= \beta_{k0} + \beta_k^T A, k = 1, \ldots, K, \text{ here } A = (A_1, \ldots, A_M)^T \\
f_k(X) &= g_k(T), k = 1, \ldots, K
\end{aligned}
$$

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

Activation function

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

i. sigmoid $\sigma(v) = 1/\left(1 + e^{-v}\right)$
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) $\sigma(v) = \left(e^{v} - e^{-v}\right) / \left(e^{v} + e^{-v}\right)$;
v. Gaussian radial basis function
$\sigma(x; \mu_m, \lambda_m) = \exp(-\|x - \mu_m\|^2 / 2\lambda_m)$.

Output function

The output function $g_k(T)$ allows 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 (multilogit model)

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}, k = 1, \ldots, K$$

which gives a vector of estimated probabilities. The corresponding classifier is $G(X) = \arg\max_k g_k(X)$.

Relationship between PPR and NN

The PPR model is a generalization of the one-hidden layer neural network, with nonparametric function g_m instead of $\sigma(\cdot)$.

$$\begin{aligned} g_m\left(\omega_m^T X\right) &= \beta_m \sigma\left(\alpha_{m0} + \alpha_m^T X\right) \\ &= \beta_m \sigma\left(\alpha_{m0} + \|\alpha_m\| \left(\omega_m^T X\right)\right) \end{aligned}$$

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

We denote the complete set of weights by θ, which consists of

$$\begin{aligned} \{\alpha_{m0}, \alpha_m; m = 1, 2, \ldots, M\} \quad & M(p+1) \text{ weights} \\ \{\beta_{k0}, \beta_k; k = 1, 2, \ldots, K\} \quad & K(M+1) \text{ weights} \end{aligned}$$

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

The generic approach to minimizing $R(\theta)$ is by gradient descent, called **back-propagation**

- derived using the chain rule of differentiation.

A single-hidden layer NN

$$R(\theta) \equiv \sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \sum_{k=1}^{K} \left(y_{ik} - f_k\left(x_i\right)\right)^2,$$

For ease of notation, let $\tilde{x}_i = \left(1, x_i^T\right)^T$, $a_{mi} = \sigma\left(\alpha_m^T \tilde{x}_i\right)$,,
$a_i = \left(a_{1i}, a_{2i}, \ldots, a_{Mi}\right)^T$, $\tilde{a}_i = \left(1, a_{1i}, a_{2i}, \ldots, a_{Mi}\right)^T$.

Let $\alpha_m := \{\alpha_{m0}, \alpha_{m1}, \alpha_{m2}, \ldots, \alpha_{mp}\}$ and
$\beta_k := \{\beta_{k0}, \beta_{k1}, \beta_{k2}, \ldots, \beta_{kM}\}$

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

$$\frac{\partial R_i}{\partial \beta_{km}} = -2\left\{y_{ik} - f_k\left(x_i\right)\right\} \dot{g}_k\left(\beta_k^T \tilde{a}_i\right) \tilde{a}_{mi}$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2\left\{y_{ik} - f_k\left(x_i\right)\right\} \dot{g}_k\left(\beta_k^T \tilde{a}_i\right) \beta_{km} \dot{\sigma}\left(\alpha_m^T \tilde{x}_i\right) \tilde{x}_{il}$$

Let the "errors" from the current model at the output layer and hidden layer be

$$\delta_{ki} := \partial R_i / \partial T_k = -2 \left\{ y_{ik} - f_k \left(x_i \right) \right\} \dot{g}_k \left(\beta_k^T \tilde{a}_i \right)$$

$$s_{mi} := \partial R_i / \partial Z_m = \dot{\sigma} \left(\alpha_m^T \tilde{x}_i \right) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

We can write the above two expressions as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} \tilde{a}_{mi}$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} \tilde{x}_{il}$$

In the r-th iteration,

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \quad \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}$$

Forward and Backpropagation Algorithm

At the r-th iteration:

1. Forward pass:

- Given current estimate $\hat{\beta}_k^{(r)}, \hat{\alpha}_m^{(r)}$, compute $\tilde{a}_{mi}^{(r)} = \sigma\left(\tilde{x}_i^T \hat{\alpha}_m^{(r)}\right)$ and $\hat{f}_k^{(r)}(x_i)$.

2. Backward pass:

- Compute

$$\hat{\delta}_{ki}^{(r)} = -2\left\{y_{ik} - \hat{f}_k^{(r)}(x_i)\right\} \dot{g}_k\left(\hat{\beta}_k^{(r)T} \tilde{a}_i^{(r)}\right)$$

$$\hat{s}_{mi}^{(r)} = \dot{\sigma}\left(\tilde{x}_i^T \hat{\alpha}_m^{(r)}\right) \sum_{k=1}^{K} \hat{\beta}_{km}^{(r)} \hat{\delta}_{ki}^{(r)}.$$

3. Update $\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^n \hat{\delta}_{ki}^{(r)} \tilde{a}_{mi}^{(r)}$, and $\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^n \hat{s}_{mi}^{(r)} \tilde{x}_{il}$

Common activiation functions

Sigmoid

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

tanh

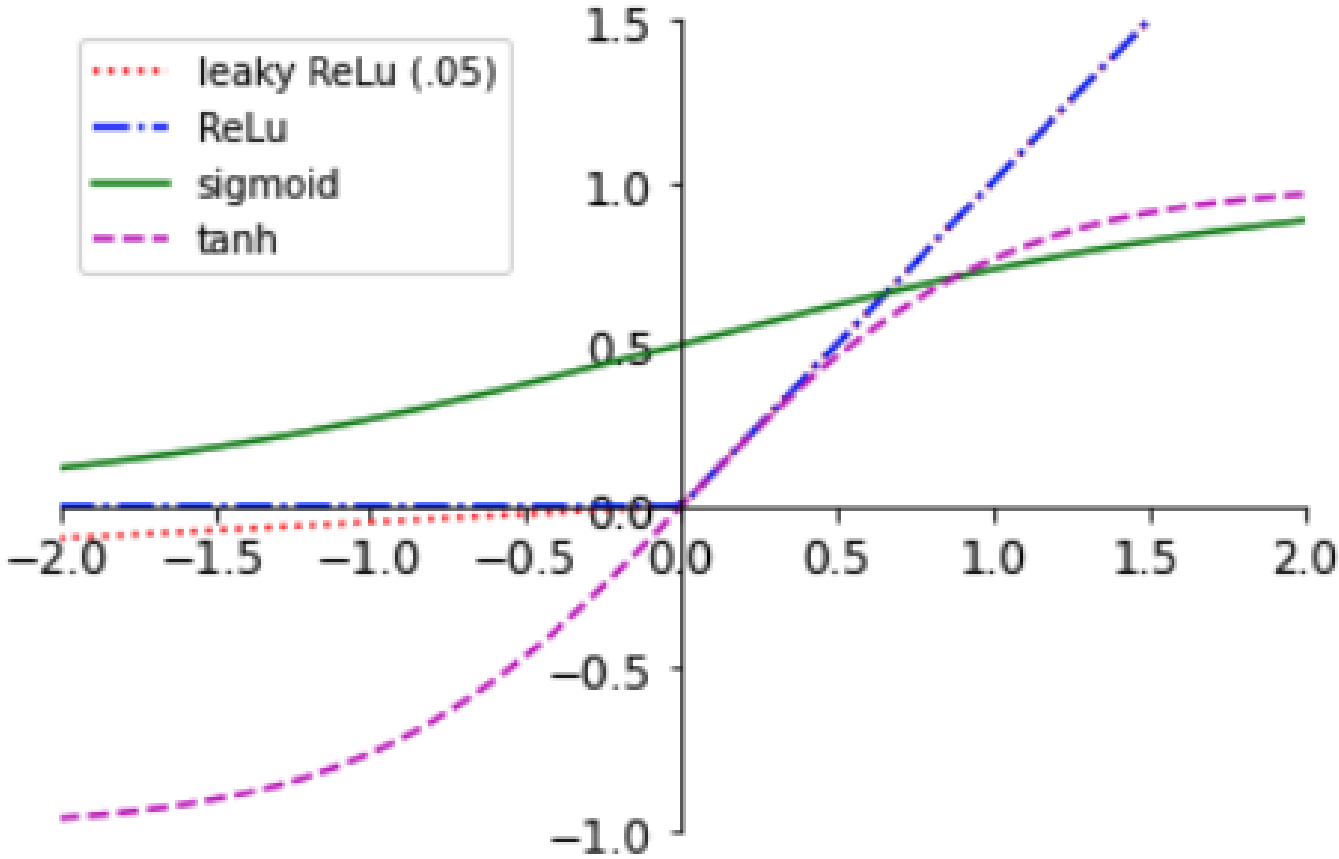
$$\tanh(Z) = \frac{e^{Z}-e^{-Z}}{e^{Z}+e^{-Z}}, \quad \frac{d}{dZ}\tanh(Z) = 1-\tanh^2(Z)$$

Rectified Linear Unit

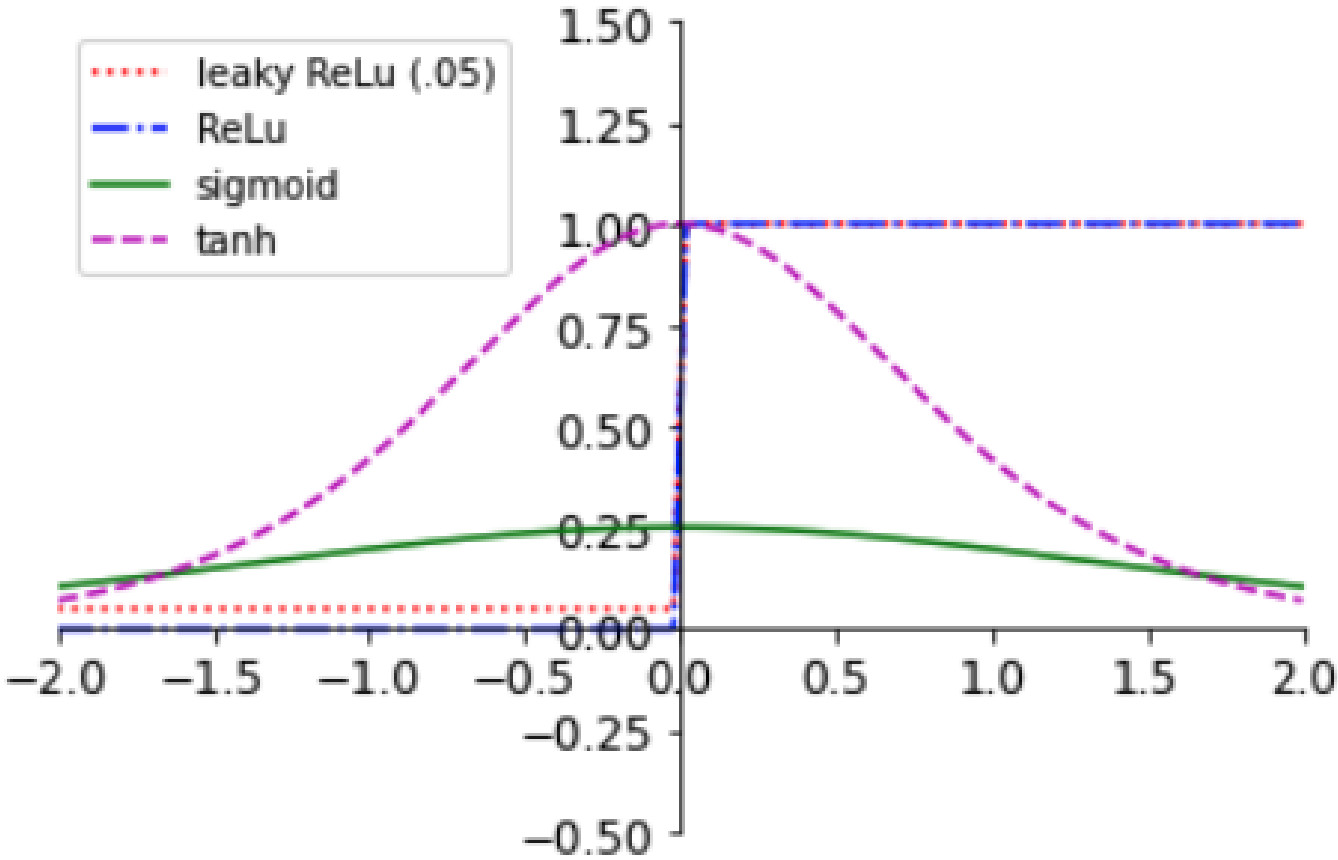
$$\mathrm{ReLu}(Z) = \max(Z,0), \quad \frac{d}{dZ}\mathrm{relu}(Z) = \begin{cases} 1 & \text{if } Z>0 \\ 0 & \text{otherwise} \end{cases}$$

Leaky Rectified Linear Unit

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



Plots of sigmoid, tanh, ReLu and leaky ReLu.



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

Deep Neural Network

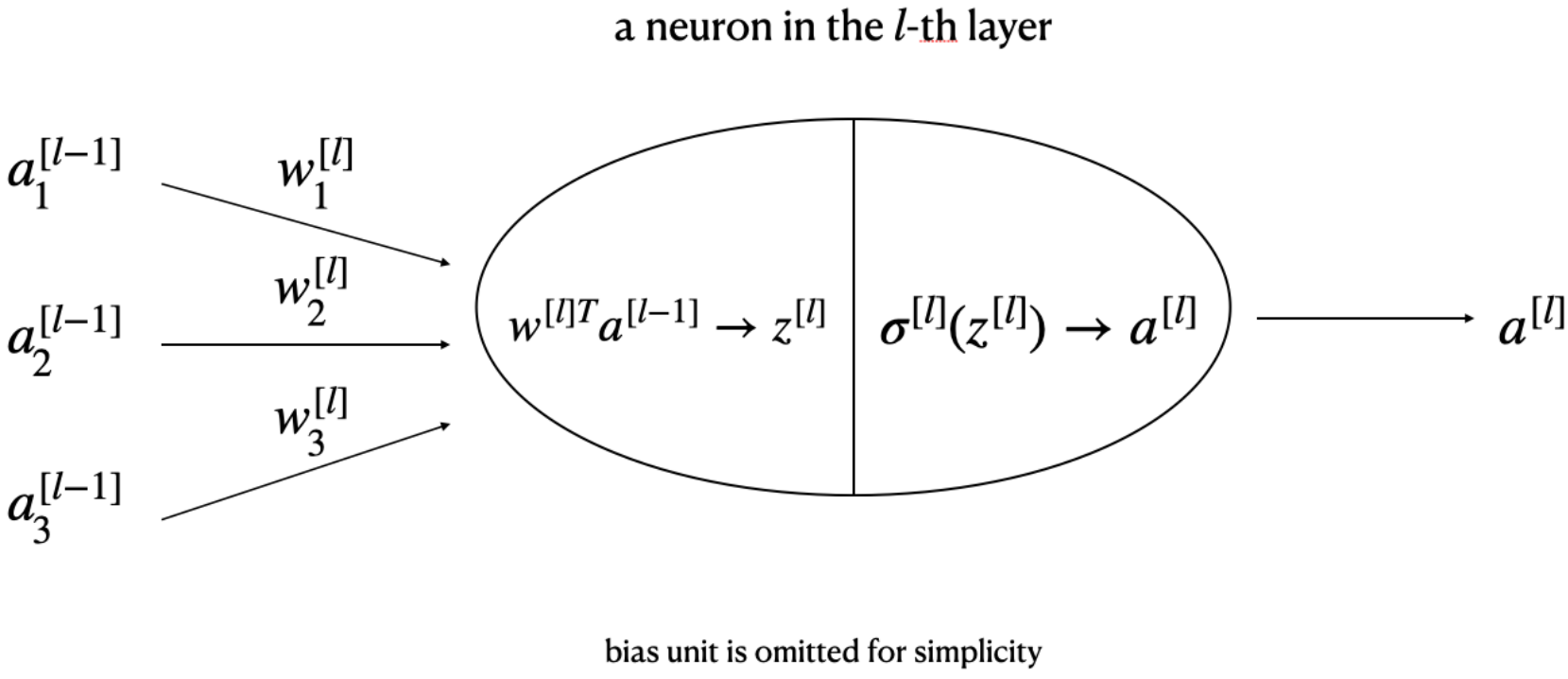
1. For inputs $x \in \mathbb{R}^p$(p features)

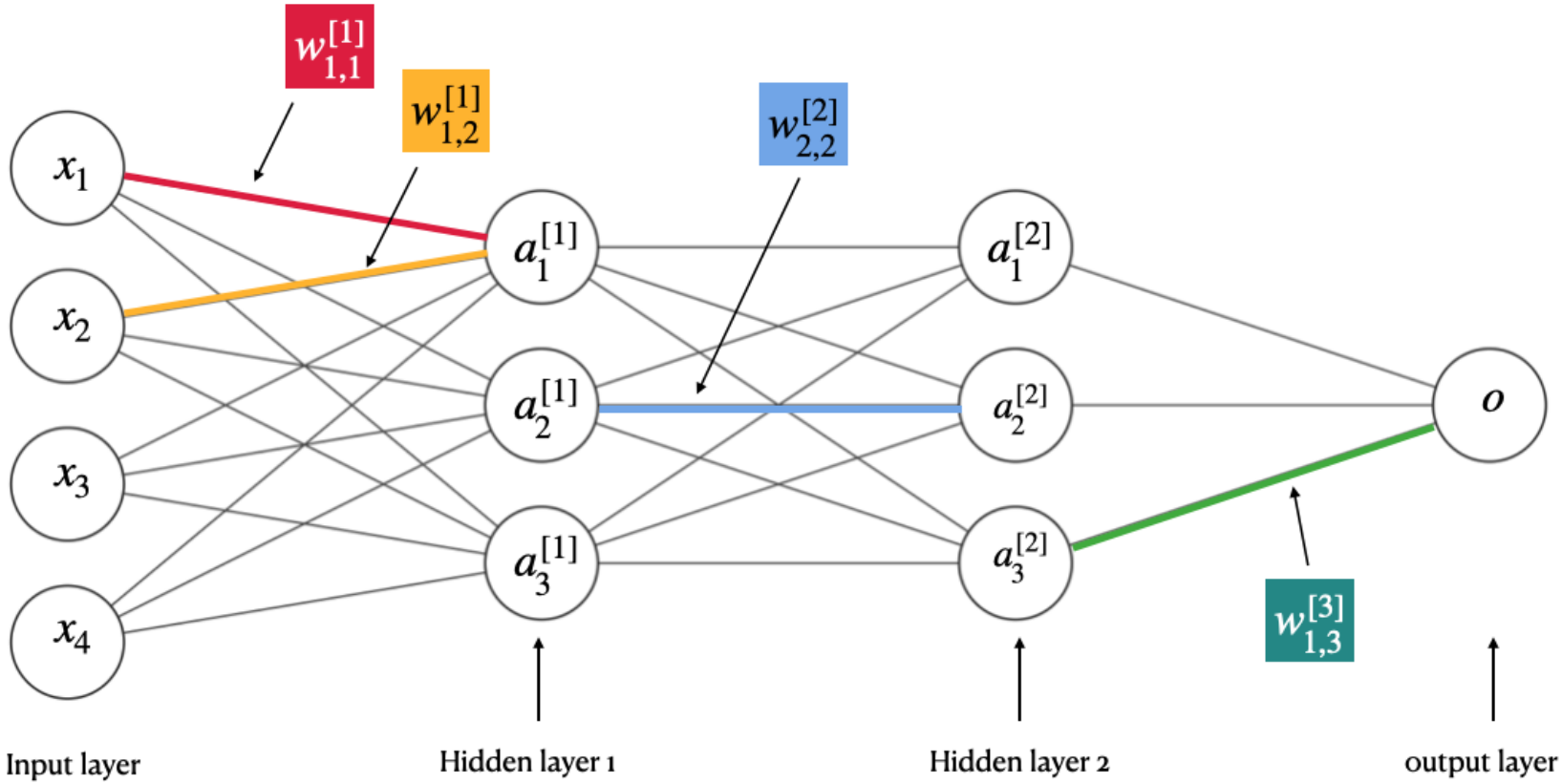
- superscript (i): is a reference to the i th observation; i.e., $x^{(1)}$ is the input value of the first observation.
- subscript j : is a reference to the j-th feature number; i.e., $x_2^{(1)}$ is the first observation of the second feature.

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 ℓ-th layer.


a neuron in the l-th layer
$a_1^{[l-1]}$
$a_2^{[l-1]}$
$a_3^{[l-1]}$
$w_1^{[l]}$
$w_2^{[l]}$
$w_3^{[l]}$
$w^{[l]T}a^{[l-1]} \to z^{[l]}$
$\sigma^{[l]}(z^{[l]}) \to a^{[l]}$
$a^{[l]}$
bias unit is omitted for simplicity



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

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

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

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

$$z^{[k]}(x) = b^{[k]} + W^{[k]} a^{[k-1]}(x)$$
$$a^{[k]}(x) = g^{[k]}\left(z^{[k]}(x)\right)$$

    where $g^{[k]}$ is the activation function.
  - For $k = L$ (output layer),

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

    where $g^{[L]}$ is the output activation function.
  - Parameters $\theta : b^{[k]}$(biases) and $W^{[k]}$(weights), $k = 1, \cdots, L$. $\{b^{[k]}, W^{[k]}\}$ called parameters in the k-th layer.

Combing n observations, the dimensions of the components in a neural network

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

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

- Penalized empirical risk:

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

where

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

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:

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

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

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

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

Negative Log-likelihood

The binary CE loss (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 CE loss (negative loglikelihood) is

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

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

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

$$\mathcal{L}_{\text{sigmoid}} = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

$$\frac{\partial \mathcal{L}_{\text{sigmoid}}}{\partial z} = \frac{\partial \mathcal{L}_{\text{sigmoid}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = -\left(\frac{y}{\hat{y}} - \frac{(1-y)}{1-\hat{y}}\right) \sigma(z)(1-\sigma(z))^{\cdot}$$

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

$$\mathcal{L}_{\text{softmax}} = -\sum_{k}^{K} y_k \log\left(\hat{y}_k\right)$$

$$\frac{\partial \mathcal{L}_{\text{softmax}}}{\partial z_j} = -\sum_{k} y_k \frac{\partial \log\left(\hat{y}_k\right)}{\partial z_j} = \hat{y}_j - y_j$$

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

Stochastic Gradient Descent

Stochastic Gradient Descent

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

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

- Let $\theta := \left(W^{[1]}, b^{[1]}, \cdots, W^{[L]}, b^{[L]}\right)$. 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 (taken at random without replacement from training data) of cardinality B (**batch size**).
- **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 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.

- **Total number of epochs**: the number of times the algorithm "sees" the entire training data. So one epoch would take n/B iterations.
- **Total number of iterations**: the total number of epochs times number of batches (n/B).

In mini-batch gradient descent, the following is involved

- 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 $64, 128, 256, 512, \ldots$, i.e., a power of 2 (due to the computer architecture).

Back-propagation

Back-propagation

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

$$\delta^{[\ell]} := \nabla_{z^{[\ell]}} \mathcal{L}(\hat{f}, y).$$

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

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

  Note $\left(g^{[L]}\right)'\left(z^{[L]}\right)$ denotes the elementwise derivative of $g^{[L]}$ w.r.t. $z^{[L]}$ elementwise.
- For $k = L, L-1, \ldots, 1$, we have

$$\begin{aligned}
\nabla_{W^{[k]}} \mathcal{L} &= \delta^{[k]} a^{[k-1]\top}, \\
\nabla_{b^{[k]}} \mathcal{L} &= \delta^{[k]}, \\
\nabla_{a^{[k-1]}} \mathcal{L} &= W^{[k]T} \delta^{[k]}, \\
\delta^{[k-1]} &= \left(g^{[k-1]}\right)'\left(z^{[k-1]}\right) \otimes \nabla_{a^{[k-1]}} \mathcal{L}, \text{ if } k \geq 2
\end{aligned}$$

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 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