EE6550 Machine Learning

Lecture Twelve – Deep Network I
Deep Feedforward Networks

Chung-Chin Lu

Department of Electrical Engineering

National Tsing Hua University

May 22, 2017

Neural Networks

- A neural network [a] can be described as a directed graph whose nodes correspond to neurons and edges correspond to links between them.
- Each neuron receives as input a weighted sum of the outputs of the neurons connected to its incoming edges.
- Feedforward network: a neural network whose underlying graph does not contain cycles.

[a]See Chapter 20 in S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithm*. New York: Cambridge University Press, 2014.

The Contents of This Lecture

- Feedforward neural networks
- VC dimensions of hypothesis sets of neural network predictors
- Expressive power of neural networks
- SGD and backpropagation

Feedforward Neural Networks

- A feedforward neural network is described by a directed acyclic graph, $G = (V, E)$, and a weight function over the edges, $w : E \to \mathbb{R}$.
  - Nodes of the graph correspond to neurons.
  - Each (directed) edge in the graph links the output of some neuron to the input of another neuron.
- Each single neuron is modeled as a simple scalar function, $\sigma : \mathbb{R} \to \mathbb{R}$, which is called the activation function of the neuron.
  - The activation function of a neuron describes the input-output relation of the neuron.
  - There are three general types of activation functions considered.

- $\sigma(a) = \text{sgn}(a)$: the sign function;
- $\sigma(a) = 1_{[a>0]}$: the threshold function;
- $\sigma(a) = 1/(1 + \exp(-a))$: the sigmoid function, which is a smooth approximation to the threshold function.

- The input of a neuron is obtained by taking a weighted sum of the outputs of all the neurons connected to it, where the weighting is according to the weight function w.

Layered Feedforward Neural Networks

- $V = \cup_{t=0}^{T} V_t$: a partition of the node set V.
  - Each V_t forms a layer.
- $E \subseteq \cup_{t=1}^{T} (V_{t-1} \times V_t)$: every edge in E connects some node in V_{t-1} to some node in V_t for some $t \in [1, T]$.
- V_0: the input layer, which consists of $n + 1$ neurons, where n is the dimensionality of the input space.
  - For every $i \in [1, n]$, the output of neuron i in V_0 is simply x_i.
  - The last neuron in V_0 is the constant neuron, which always outputs 1.
- $v_{t,i}$: the ith neuron of the tth layer.

- $o_{t,i}(\boldsymbol{x})$: the output of $v_{t,i}$ when the network is fed with the input vector $\boldsymbol{x}$.
  - For $i \in [1, n]$, we have $o_{0,i}(\boldsymbol{x}) = x_i$ and for $i = n + 1$, we have $o_{0,n+1}(\boldsymbol{x}) = 1$.
- Calculation in a layer by layer manner: suppose we have calculated the outputs of the neurons at layer t. Then, we can calculate the outputs of the neurons at layer $t + 1$ as follows.
  - Fix some $v_{t+1,j} \in V_t$. Let $a_{t+1,j}(\boldsymbol{x})$ denote the total input to $v_{t+1,j}$ when the network is fed with the input vector $\boldsymbol{x}$,

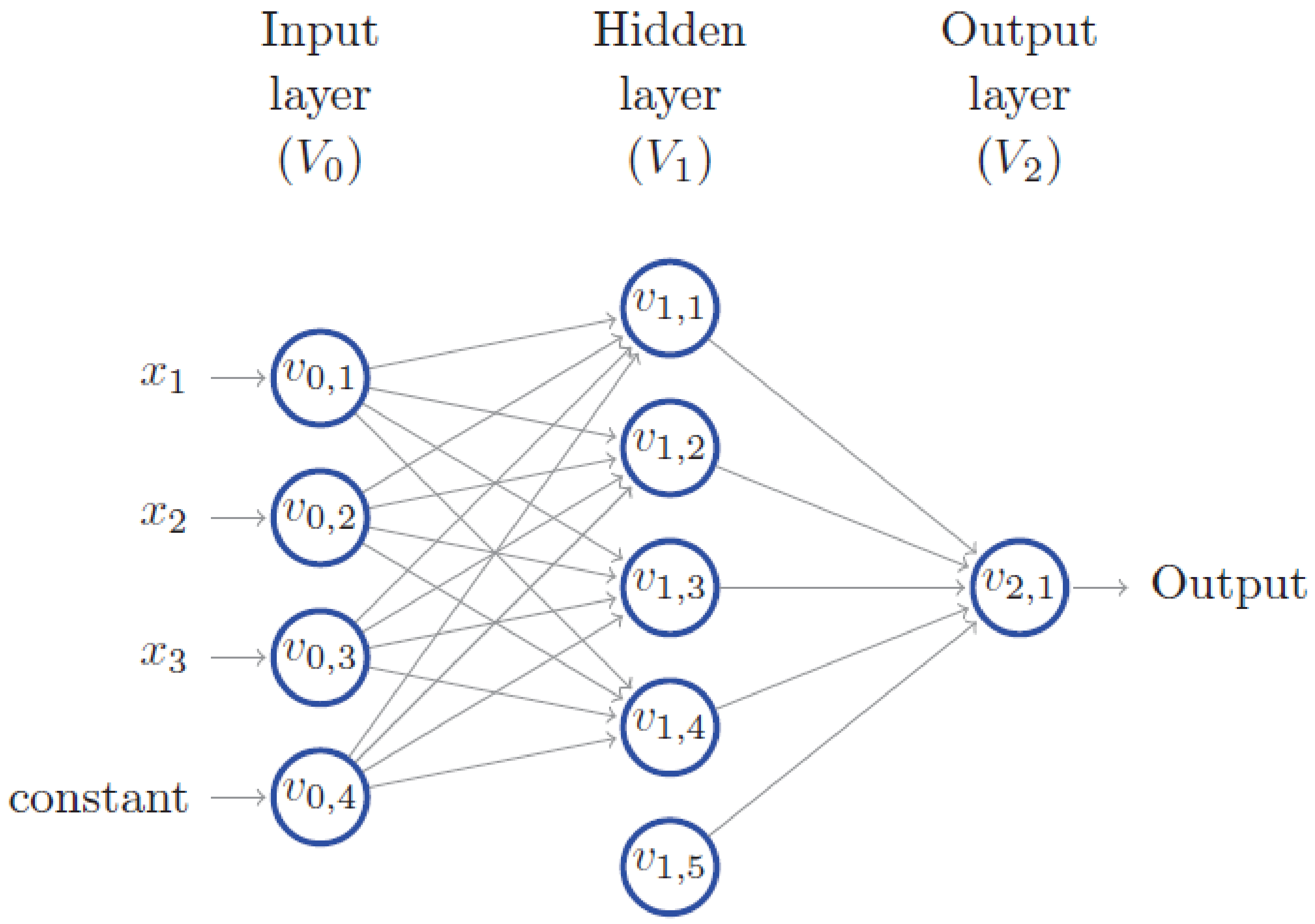
$$a_{t+1,j}(\boldsymbol{x}) = \sum_{r:(v_{t,r},v_{t+1,j})\in E} w((v_{t,r}, v_{t+1,j}))o_{t,r}(\boldsymbol{x}).$$

  - The output $o_{t+1,j}(\boldsymbol{x})$ of the neuron $v_{t+1,j}$ is

$$o_{t+1,j}(\boldsymbol{x}) = \sigma(a_{t+1,j}(\boldsymbol{x})).$$

- Layers $V_1, \ldots, V_{T-1}$: the $T - 1$ hidden layers.

- V_T: the output layer.
  - In simple prediction problems, the output layer contains a single neuron whose output is the output of the network.
- $|V| = \sum_{t=0}^{T} |V_t|$: the size of the network.
- T: the number of layers in the network (excluding V_0), called the depth of the network.
- $\max_{0 \le t \le T} |V_t|$: the width of the network.



A layered feedforward neural network of depth 2, size 10, and width 5.

The Contents of This Lecture

- Feedforward neural networks
- VC dimensions of hypothesis sets of neural network predictors
- Expressive power of neural networks
- SGD and backpropagation

Hypothesis Sets of Neural Network Predictors

- A neural network can be specified by a quadruple (V, E, σ, w).
- $h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$: the hypothesis associated with a neural network specified by (V, E, σ, w).
- $\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} \mid w \text{ is a mapping from } E \text{ to } \mathbb{R}\}$: the hypothesis set of neural network predictors by fixing the directed graph $G = (V, E)$ as well as the activation function σ but letting the weight function w arbitrary.
  - The triplet (V, E, σ) is often called the architecture of the hypothesis set of neural network predictors.
  - The parameters specifying a hypothesis in the hypothesis set $\mathcal{H}_{V,E,\sigma}$ are the weights over the edges of the network.

VC-Dimension of $\mathcal{H}_{V,E,\mathbf{sgn}}$ with Single Output Neuron

Theorem 1: The VC-Dimension of $\mathcal{H}_{V,E,\text{sgn}}$ with single output neuron is $O(|E| \log |E|)$.

VC-Dimension of $\mathcal{H}_{V,E,\sigma}$ with Single Output Neuron

Theorem 2: The VC-Dimension of $\mathcal{H}_{V,E,\sigma}$ with single output neuron, where σ is the sigmoid function, is both $\Omega(|E|^2)$ and $O(|V|^2|E|^2)$.

The Contents of This Lecture

- Feedforward neural networks
- VC dimensions of hypothesis sets of neural network predictors
- Expressive power of neural networks
- SGD and backpropagation

Expressive Power of Neural Networks

- Fixing an architecture, (V, E, σ), what functions hypotheses in $\mathcal{H}_{V,E,\sigma}$ can implement ?
- Which type of Boolean functions (i.e., functions from $\{\pm 1\}^n$ to $\{\pm 1\}$) can be implemented by $H_{V,E,\text{sgn}}$?
  - Since real numbers are stored using b bits in every digital computer, whenever we calculate a function $f : \mathbb{R}^n \to \mathbb{R}$ on such a computer we in fact calculate a function $g : \{\pm 1\}^{nb} \to \{\pm 1\}^b$.
  - Therefore, studying which Boolean functions can be implemented by $H_{V,E,\text{sgn}}$ can tell us which functions can be implemented on a computer that stores real numbers using b bits.

Neural Networks Are Universal for Boolean Functions

Proposition 1: For every n, there exists a graph (V, E) of depth 2, such that $H_{V,E,\mathrm{sgn}}$ contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$.

Proof.

- Construct a layered graph with node set $V = V_0 \cup V_1 \cup V_2$, where $|V_0| = n + 1$, $|V_1| = 2^n + 1$, and $|V_2| = 1$, and edge set E consisting of all possible edges between adjacent layers.
- $f : \{\pm 1\}^n \to \{\pm 1\}$: an arbitrary Boolean function.
- We need to show that $f = h_{V,E,\mathrm{sgn},w}$ by specifying a weight function w.
- $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$: all vectors in $\{\pm 1\}^n$ on which f outputs 1.
- Observe that for every $i \in [1, k]$ and every $\boldsymbol{x} \in \{\pm 1\}^n$, if $\boldsymbol{x} \neq \boldsymbol{u}_i$, then $\boldsymbol{x} \cdot \boldsymbol{u}_i \leq n - 2$ and if $\boldsymbol{x} = \boldsymbol{u}_i$, then $\boldsymbol{x} \cdot \boldsymbol{u}_i = n$.

- The function $g_i(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{x} \cdot \boldsymbol{u}_i - n + 1)$ equals 1 if and only if $\boldsymbol{x} = \boldsymbol{u}_i$.
- We can adapt the weights between V_0 and V_1 so that for every $i \in [1, k]$, the neuron $v_{1,i}$ implements the function $g_i(\boldsymbol{x})$.
  - $w((v_{0,j}, v_{1,i})) = u_{i,j}$ for all $j \in [1, n]$ and $w((v_{0,n+1}, v_{1,i})) = -n + 1$.
  - The total input $a_{1,i}(\boldsymbol{x})$ of $v_{1,i}$ is $\boldsymbol{x} \cdot \boldsymbol{u}_i - n + 1$.
  - $o_{1,i}(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{x} \cdot \boldsymbol{u}_i - n + 1) = g_i(\boldsymbol{x})$.
- Observe that $f(\boldsymbol{x})$ is the disjunction of the functions $g_i(\boldsymbol{x})$, and therefore can be written as

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{k} g_i(\boldsymbol{x}) + k - 1\right)$$

  - $w((v_{1,i}, v_{2,1})) = 1 \ \forall\ i \in [1, k]$ and $w((v_{1,2^n+1}, v_{2,1})) = k - 1$.

This concludes the proof. $\square$

Exponential Size Is Inevitable

Theorem 3: For every n, let $s(n)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(n)$ such that the hypothesis set $H_{V,E,\mathrm{sgn}}$ contains all the functions from $\{\pm 1\}^n$ to $\{\pm 1\}$. (This implies that there are $n + 1$ input neurons and a single output neuron.) Then, $s(n)$ is exponential in n. Similar results hold for $H_{V,E,\sigma}$ where σ is the sigmoid function.

Proof.

- Suppose that for some (V, E) with $|V_0| = n + 1$ and $|V_T| = 1$, we have that $\mathcal{H}_{V,E,\mathrm{sgn}}$ contains all functions from $\{\pm 1\}^n$ to $\{\pm 1\}$.
- Since $\mathcal{H}_{V,E,\mathrm{sgn}}$ can shatter the set $\{\pm 1\}^n$ of 2^n vectors, the VC dimension of $\mathcal{H}_{V,E,\mathrm{sgn}}$ is lower bounded by 2^n.
- We know that the VC dimension of $\mathcal{H}_{V,E,\mathrm{sgn}}$ is $O(|E| \log |E|)$

and is upper bounded by $O(|V|^3)$. Thus we have

$$2^n \leq \text{VC-dim}(\mathcal{H}_{V,E,\text{sgn}}) \leq M|V|^3$$

for some constant $M > 0$.

- This implies that $|V|$ is $\Omega(2^{n/3})$.

This concludes the proof for the case of neural networks with the sign activation function. The proof for the sigmoid case is analogous. $\square$

Remarks

- It is possible to derive a similar theorem for $H_{V,E,\sigma}$ with single output neuron for any activation function σ, as long as we restrict the weights so that it is possible to express every weight using a number of bits which is bounded by a universal constant.
- We can even consider hypothesis sets where different neurons can employ different activation functions, as long as the number of allowed activation functions is also finite.
- We will show that all Boolean functions that can be calculated in time $O(T(n))$ can also be expressed by a neural network of size $O(T(n)^2)$.

A Lemma of Logical Implementation

Lemma 1: Suppose that a neuron v, that implements the sign activation function, has k incoming edges, connecting it to neurons whose outputs are in $\{\pm 1\}$. Then, by adding one more edge, linking a constant neuron to v, and by adjusting the weights on the edges to v, the output of v can implement the conjunction or the disjunction of its inputs.

Proof.

- The conjunction function $f : \{\pm 1\}^n \to \{\pm 1\}$ is $f(\boldsymbol{x}) = \wedge_i x_i$, which can be implemented as $f(\boldsymbol{x}) = \mathrm{sgn}(\sum_{i=1}^{k} x_i - k + 1)$.
- The disjunction function $f : \{\pm 1\}^n \to \{\pm 1\}$ is $f(\boldsymbol{x}) = \vee_i x_i$, which can be implemented as $f(\boldsymbol{x}) = \mathrm{sgn}(\sum_{i=1}^{k} x_i + k - 1)$.

This completes the proof. $\square$

Neural Networks of Polynomial Size

Theorem 4: Let $T : \mathbb{N} \to \mathbb{N}$ and for every n, let $\mathfrak{F}_n$ be the set of functions that can be implemented using a Turing machine using runtime of at most $T(n)$. Then, there exist constants $b, c > 0$ such that for every n, there is a graph (V_n, E_n) of size at most $cT(n)^2 + b$ such that $H_{V_n,E_n,\mathrm{sgn}}$ contains $\mathfrak{F}_n$.

The Contents of This Lecture

- Feedforward neural networks
- VC dimensions of hypothesis sets of neural network predictors
- Expressive power of neural networks
- SGD and backpropagation

Learning Neural Networks Is NP Hard

Theorem 5: Let $k \geq 3$. For every n, let (V, E) be a three-layered directed graph with $n+1$ input nodes, where one of them is the constant neuron, $k+1$ nodes at the (single) hidden layer, where one of them is the constant neuron, and a single output node. Then, it is NP hard to implement the ERM rule with respect to the hypothesis set $H_{V,E,\mathrm{sgn}}$.

Proof.

- The k-coloring problem can be reduced to the ERM problem with respect to $H_{V,E,\mathrm{sgn}}$.
- The k-coloring problem is known to be NP hard for all $k \geq 3$ [a].
- The ERM problem with respect to $H_{V,E,\mathrm{sgn}}$ is NP hard. $\square$

[a] R.M. Karp, *Reducibility Among Combinatorial Problems*. Springer, 1972.

Neural Network Learning Problem

- (V, E, σ): the architecture of the hypothesis set $\mathcal{H}_{V,E,\sigma}$ of neural network predictors.
- $\boldsymbol{w} \in \mathbb{R}^{|E|}$: the weight vector which represents the weight function $w : E \to \mathbb{R}$.
- $n = |V_0|$: the number of input neurons.
- $k = |V_T|$: the number of output neurons.
- $\boldsymbol{h_w} : \mathbb{R}^n \to \mathbb{R}^k$: the vector function calculated by the neural network if the weight function is represented by $\boldsymbol{w}$.
- $\mathscr{I} = \mathbb{R}^n$: the input space, associated with a probability space $(\mathbb{R}^n, \mathcal{B}_n, P)$ where P is unknown.
- $\mathscr{Y} = \mathbb{R}^k$: the label space.

- $\boldsymbol{c} : \mathbb{R}^n \to \mathbb{R}^k$: an unknown target concept.
- $\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$: the loss of predicting $\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x})$ when the label is $\boldsymbol{c}(\boldsymbol{x})$.
  - $\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})) = \frac{1}{2}\|\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}) - \boldsymbol{c}(\boldsymbol{x})\|^2$: squared loss will be assumed.
- $R(\boldsymbol{w}) = \underset{\boldsymbol{x} \sim P}{E}[\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))]$: the risk or generalization error or true error of the neural network predictor represented by $\boldsymbol{w}$.

Stochastic Gradient Descent Algorithm for Neural Networks

```
StochasticGradientDescent($N, \{\eta_i\}_{i=1}^{N}, \lambda, V, E, \sigma, \boldsymbol{w}_0, S$)
1. $\boldsymbol{w}^{(1)} \leftarrow$ Random($\boldsymbol{w}_0$)      ▷ a random initial weight vector around $\boldsymbol{w}_0$
2. for $i \leftarrow 1$ to $N$ do
3.     $(\boldsymbol{x}^{(i)}, \boldsymbol{c}(\boldsymbol{x}^{(i)})) \leftarrow$ Sample($S$)    ▷ a labeled item taken from the labeled sample $S$
4.     $\boldsymbol{u}^{(i)} \leftarrow$ Backpropagation($\boldsymbol{x}^{(i)}, \boldsymbol{c}(\boldsymbol{x}^{(i)}), V, E, \sigma, \boldsymbol{w}^{(i)}$)
5.     $\boldsymbol{w}^{(i+1)} \leftarrow \boldsymbol{w}^{(i)} - \eta_i(\boldsymbol{u}^{(i)} + \lambda \boldsymbol{w}^{(i)})$
6. return $\bar{\boldsymbol{w}}$ = the best performing $\boldsymbol{w}^{(i)}$ on a validation set
```

Stochastic Gradient Descent Algorithm for Neural Networks

- Parameters:
  - N: number of iterations.
  - $\{\eta_i\}_{i=1}^{N}$: step size sequence. There are two popular choices:
    - $\eta_i = \eta \; \forall \; i$: fixed step size.
    - $\eta_i = \frac{\alpha}{\sqrt{i}}$: variable step size for some $\alpha > 0$
  - $\lambda \geq 0$: regularization parameter.
    - We try to minimize $R(\boldsymbol{w}) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$.
- Input:
  - $(V = \cup_{t=0}^{T} V_t, E, \sigma)$: architecture of a feedforward neural network.
  - $\boldsymbol{w}_0$: suggested center of the initial weight vector, usually set to be $\mathbf{0}$.

- $S = \{(\boldsymbol{x}_1, \boldsymbol{c}(\boldsymbol{x}_1)), (\boldsymbol{x}_2, \boldsymbol{c}(\boldsymbol{x}_2)), \ldots, (\boldsymbol{x}_m, \boldsymbol{c}(\boldsymbol{x}_m))\}$: a labeled sample of size m.

- Initialization:
  - The initial weight vector $\boldsymbol{w}^{(1)}$ is chosen at random around the center $\boldsymbol{w}_0$.

- Modules:
  - Random($\boldsymbol{w}_0$): an algorithm to randomly choose an initial weight vector $\boldsymbol{w}^{(1)}$ centered around $\boldsymbol{w}_0$.
    - It is common to set $w_j^{(1)} = w_{0,j} + n_j$ where n_j is drawn i.i.d. from a Gaussian distribution with zero mean and variance ε^2.

- SAMPLE(S): an algorithm to take a labeled item $(\boldsymbol{x}^{(i)}, \boldsymbol{c}(\boldsymbol{x}^{(i)})$ from the sample S. There are three possible ways:
  * Round-robin: first ordering the sample S at random and once the order is fixed, selecting a labeled item form S in this order and around.
  * Purely random: selecting a labeled item form S at random in each iteration.
  * Round-robin once and then purely random.
- BACKPROPAGATION$(\boldsymbol{x}^{(i)}, \boldsymbol{c}(\boldsymbol{x}^{(i)}), V, E, \sigma, \boldsymbol{w}^{(i)})$: an algorithm to calculate a conditionally unbiased estimate of the gradient $\nabla R(\boldsymbol{w}^{(i)})$ of the risk R at the given weight vector $\boldsymbol{w}^{(i)}$ with a labeled item $(\boldsymbol{x}^{(i)}, \boldsymbol{c}(\boldsymbol{x}^{(i)})$ in S drawn i.i.d. from the unknown distribution P.

- Output:
  - $\bar{\boldsymbol{w}}$: the best performing $\boldsymbol{w}^{(i)}$ on a validation set.
    - * While the training error will continue to decrease as the number of iterations increases, the validation error may decrease in the early iterations but then starts to increase in the later iterations.

The Backpropagation Algorithm

Differentiable Vector Fields

- $\boldsymbol{f} = (f_1, f_2, \ldots, f_m) : \Omega \to \mathbb{R}^m$: a differentiable vector field on an open subset Ω of $\mathbb{R}^n$, i.e., for each point $\boldsymbol{w} \in \Omega$, there is an $m \times n$ matrix $D\boldsymbol{f}(\boldsymbol{w})$ such that

$$\boldsymbol{f}(\boldsymbol{w}+\boldsymbol{u}) = \boldsymbol{f}(\boldsymbol{w}) + D\boldsymbol{f}(\boldsymbol{w})\boldsymbol{u} + o(\|\boldsymbol{u}\|) \text{ as } \boldsymbol{u} \to \mathbf{0}.$$

  - $\lim_{\boldsymbol{u}\to\mathbf{0}} \frac{o(\|\boldsymbol{u}\|)}{\|\boldsymbol{u}\|} = 0.$

- $D\boldsymbol{f}(\boldsymbol{w})$: called the total derivative of $\boldsymbol{f}$ at $\boldsymbol{w}$.

- $J_{\boldsymbol{w}}(\boldsymbol{f})$: the Jocobian matrix of $\boldsymbol{f}$ at $\boldsymbol{w}$,

$$J_{\boldsymbol{w}}(\boldsymbol{f}) = \begin{bmatrix} \partial f_1/\partial w_1 & \partial f_1/\partial w_1 & \cdots & \partial f_1/\partial w_n \\ \partial f_2/\partial w_1 & \partial f_2/\partial w_1 & \cdots & \partial f_2/\partial w_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_m/\partial w_1 & \partial f_m/\partial w_1 & \cdots & \partial f_m/\partial w_n \end{bmatrix}.$$

- $D\boldsymbol{f}(\boldsymbol{w}) = J_{\boldsymbol{w}}(\boldsymbol{f})$.
- $\boldsymbol{g} = (g_1, g_2, \ldots, g_n) : X \to \Omega$: a differentiable vector field on an open subset X of $\mathbb{R}^k$.
- Chain rule: $J_{\boldsymbol{x}}(\boldsymbol{f} \circ \boldsymbol{g}) = J_{\boldsymbol{g}(\boldsymbol{x})}(\boldsymbol{f}) J_{\boldsymbol{x}}(\boldsymbol{g})$, i.e., $D(\boldsymbol{f} \circ \boldsymbol{g})(\boldsymbol{x}) = D\boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))D\boldsymbol{g}(\boldsymbol{x})$.

Special Differentiable Vector Fields

- $\boldsymbol{f}(\boldsymbol{w}) = A\boldsymbol{w}$ for some $A \in \mathbb{R}^{n\times k}$. Then $J_{\boldsymbol{w}}(\boldsymbol{f}) = A$ for all $\boldsymbol{w} \in \mathbb{R}^k$.
- $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n) : \mathbb{R}^n \to \mathbb{R}^n$: $\sigma_i(\boldsymbol{\theta}) = \sigma(\theta_i)$ for all $i \in [1, n]$, where $\sigma(\theta) = \frac{1}{1+e^{-\theta}}$ is the sigmoid function. Then

$$J_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) = \begin{bmatrix} \sigma'(\theta_1) & 0 & \cdots & 0 \\ 0 & \sigma'(\theta_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'(\theta_n) \end{bmatrix} \triangleq \operatorname{diag}(\boldsymbol{\sigma}'(\boldsymbol{\theta})),$$

  where $\sigma'(\theta) = \frac{1}{(1+e^{\theta})(1+e^{-\theta})}$.
- If $\boldsymbol{f}(\boldsymbol{w}) = A\boldsymbol{w}$ for some $A \in \mathbb{R}^{n\times k}$, then

$$J_{\boldsymbol{w}}(\boldsymbol{\sigma} \circ \boldsymbol{f}) = \operatorname{diag}(\boldsymbol{\sigma}'(M\boldsymbol{w}))A.$$

Layered Feedforward Neural Network

- $(V = \cup_{t=0}^{T} V_t, E, \sigma, \boldsymbol{w})$: a layered feedforward neural network.
  - $|V_t| = k_t$.
  - $V_t = \{v_{t,1}, v_{t,2}, \ldots, v_{t,k_t}\}$.
  - $E \subseteq \cup_{t=0}^{T-1}(V_t \times V_{t+1})$.
- $W_t \in \mathbb{R}^{k_{t+1} \times k_t}$: a matrix which gives a weight to every potential edge between V_t and V_{t+1}.
  - If the edge $(v_{t,j}, v_{t+1,i})$ exists in E, then we set $W_{t,i,j}$ to be the weight of this edge according to $\boldsymbol{w}$.
  - Otherwise, we add a "phantom" edge and set its weight to be zero, $W_{t,i,j} = 0$.
  - We can assume, without loss of generality, that all edges exist, that is, $E = \cup_{t=0}^{T-1}(V_t \times V_{t+1})$.

Conditionally Unbiased Estimate of the Gradient ∇R of the Risk R

- $\mathscr{I} = \mathbb{R}^n$: the input space, associated with a probability space $(\mathbb{R}^n, \mathcal{B}_n, P)$ where P is unknown, $n = |V_0| - 1$.
- $\mathscr{Y} = \mathbb{R}^k$: the label space, $k = |V_T|$.
- $\boldsymbol{c} : \mathbb{R}^n \to \mathbb{R}^k$: an unknown target concept.
- $\boldsymbol{h_w} : \mathbb{R}^n \to \mathbb{R}^k$: the vector function calculated by the neural network if the weight function is represented by $\boldsymbol{w}$.
- $\Delta(\boldsymbol{h_w}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$: the loss of predicting $\boldsymbol{h_w}(\boldsymbol{x})$ when the label is $\boldsymbol{c}(\boldsymbol{x})$. The squared loss will be assumed, $\Delta(\boldsymbol{h_w}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})) = \frac{1}{2}\|\boldsymbol{h_w}(\boldsymbol{x}) - \boldsymbol{c}(\boldsymbol{x})\|^2$.
- $R(\boldsymbol{w}) = \underset{\boldsymbol{x}\sim P}{E}[\Delta(\boldsymbol{h_w}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))]$: the risk or generalization error or true error of the neural network predictor represented by $\boldsymbol{w}$.

- At the ith iteration of the stochastic gradient descent algorithm, there is a newly updated weight vector $\boldsymbol{w}^{(i)}$ and a new input point $\boldsymbol{x}^{(i)}$ sampled from the unknown distribution P, independent of $\boldsymbol{w}^{(i)}$.
- The gradient $\nabla_{\boldsymbol{w}}\Delta(\boldsymbol{h}_{\boldsymbol{w}^{(i)}}(\boldsymbol{x}^{(i)}), \boldsymbol{c}(\boldsymbol{x}^{(i)}))$ of the loss $\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}), \boldsymbol{c}(\boldsymbol{x}^{(i)}))$ at $\boldsymbol{w}^{(i)}$ is a conditionally unbiased estimate of the gradient $\nabla R(\boldsymbol{w}^{(i)})$ of the risk R at $\boldsymbol{w}^{(i)}$ when given $\boldsymbol{w}^{(i)}$, i.e.,

$$\underset{\omega_t \sim P}{E}[\nabla_{\boldsymbol{w}}\Delta(\boldsymbol{h}_{\boldsymbol{w}^{(i)}}(\boldsymbol{x}^{(i)}), \boldsymbol{c}(\boldsymbol{x}^{(i)}))] = \nabla R(\boldsymbol{w}^{(i)}).$$

- The backpropagation algorithm is used to calculate the gradient $\nabla_{\boldsymbol{w}}\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$ of the loss $\Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$ with respective to $\boldsymbol{w}$.

Hypotheses Associated with Subnetwroks

- $(\cup_{i=t}^{T} V_i, E \cap (\cup_{i=t}^{T-1}(V_i \times V_{i+1})), \sigma)$: the architecture of the subnetwork from layer t to layer T.
- $\boldsymbol{w}_{[t,T]}$: consisting of all weights in the subnetwork with architecture $(\cup_{i=t}^{T} V_i, E \cap (\cup_{i=t}^{T-1}(V_i \times V_{i+1})), \sigma)$
- $\boldsymbol{h}_{\boldsymbol{w}_{[t,T]}}$: the hypothesis associated with the subnetwork $(\cup_{i=t}^{T} V_i, E \cap (\cup_{i=t}^{T-1}(V_i \times V_{i+1})), \sigma, \boldsymbol{w}_{[t,T]})$.
- $\boldsymbol{o}_t(\boldsymbol{x}) = (o_{t,1}(\boldsymbol{x}), o_{t,2}(\boldsymbol{x}), \ldots, o_{t,k_t}(\boldsymbol{x}))^t$: the input vector of the subnetwork $(\cup_{i=t}^{T} V_i, E \cap (\cup_{i=t}^{T-1}(V_i \times V_{i+1})), \sigma, \boldsymbol{w}_{[t,T]})$, which is just the output vector of layer t.

Loss of Subnetwroks

- $\ell_t : \mathbb{R}^{k_t} \to \mathbb{R}$: the loss function of the subnetwork $(\cup_{i=t}^{T} V_i, E \cap (\cup_{i=t}^{T-1} (V_i \times V_{i+1})), \sigma, \boldsymbol{w}_{[t,T]})$ with respective to the unknown concept $\boldsymbol{c}(\boldsymbol{x})$, defined as a function of the outputs $\boldsymbol{o}_t(\boldsymbol{x})$ of the neurons in V_t,

$$\ell_t(\boldsymbol{o}_t(\boldsymbol{x})) = \Delta(\boldsymbol{h}_{\boldsymbol{w}_{[t,T]}}(\boldsymbol{o}_t(\boldsymbol{x})), \boldsymbol{c}(\boldsymbol{x})).$$

- It can be seen that

$$\ell_T(\boldsymbol{o}_T(\boldsymbol{x})) = \cdots = \ell_t(\boldsymbol{o}_t(\boldsymbol{x})) = \cdots = \ell_0(\boldsymbol{o}_0(\boldsymbol{x})) = \Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})).$$

Calculation of the Partial Derivatives of the Loss $\Delta(\boldsymbol{h}_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$ with Respect to the Weights in W_t

- When calculating the partial derivatives of the loss $\Delta(\boldsymbol{h}_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$ with respect to the weights in W_t, we fix all other weights of the network. It follows that the outputs of all the neurons in V_t are fixed numbers which do not depend on the weights in W_t.
- In this case, we have

$$\frac{\partial}{\partial W_{t,i,j}} \Delta(\boldsymbol{h}_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})) = \frac{\partial}{\partial W_{t,i,j}} \Delta(\boldsymbol{h}_{\boldsymbol{W}_{[t,T]}}(\boldsymbol{o}_t(\boldsymbol{x})), \boldsymbol{c}(\boldsymbol{x})) = \frac{\partial \ell_t(\boldsymbol{o}_t(\boldsymbol{x}))}{\partial W_{t,i,j}}.$$

- $\boldsymbol{a}_{t+1} = W_t \boldsymbol{o}_t$: the input to the neurons of the $(t+1)$th layer V_{t+1}.
- $\boldsymbol{o}_{t+1} = \boldsymbol{\sigma}(\boldsymbol{a}_{t+1})$: the output of the neurons of the $(t+1)$th layer V_{t+1}.

- To represent the loss $\Delta(\boldsymbol{h}_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x}))$ as a function of weights in W_t, we define

$$g_{t+1}(W_t) \triangleq \ell_t(\boldsymbol{o}_t) = \ell_{t+1}(\boldsymbol{o}_{t+1}) = \ell_{t+1}(\boldsymbol{\sigma}(\boldsymbol{a}_{t+1})) = \ell_{t+1}(\boldsymbol{\sigma}(W_t \boldsymbol{o}_t)).$$

- Now we have

$$\frac{\partial}{\partial W_{t,i,j}} \Delta(\boldsymbol{h}_{\boldsymbol{W}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})) = \frac{\partial g_{t+1}(W_t)}{\partial W_{t,i,j}} = \frac{\partial}{\partial W_{t,i,j}} \ell_{t+1}(\boldsymbol{\sigma}(W_t \boldsymbol{o}_t)).$$

- It is convenient to represent the weights in the matrix W_t as a column vector $\boldsymbol{w}_t = (W_{t,1,1}, \ldots, W_{t,1,k_t}, W_{t,2,1}, \ldots, W_{t,2,k_t}, \ldots, W_{t,k_{t+1},1}, \ldots, W_{t,k_{t+1},k_t})^t$.

- Define a $k_{t+1} \times (k_t k_{t+1})$ matrix,

$$O_t \triangleq \begin{bmatrix} \boldsymbol{o}_t^t & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{o}_t^t & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{o}_t^t \end{bmatrix}.$$

- Then $W_t \boldsymbol{o}_t = O_t \boldsymbol{w}_t$ and $g_{t+1}(\boldsymbol{w}_t) = \ell_{t+1}(\boldsymbol{\sigma}(O_t \boldsymbol{w}_t))$.
- The Jocobian (i.e., gradient) of g_{t+1} can now be calculated by chain rule:

$$\begin{aligned} J_{\boldsymbol{w}_t}(g_{t+1}) &= J_{\boldsymbol{\sigma}(O_t \boldsymbol{w}_t)}(\ell_{t+1}) J_{\boldsymbol{w}_t}(\boldsymbol{\sigma}(O_t \boldsymbol{w}_t)) \\ &= J_{\boldsymbol{o}_{t+1}}(\ell_{t+1}) \mathrm{diag}(\boldsymbol{\sigma}'(\boldsymbol{a}_{t+1})) O_t. \end{aligned}$$

- Define $\boldsymbol{\delta}_t = J_{\boldsymbol{o}_t}(\ell_t)$. Then we have
$J_{\boldsymbol{w}_t}(g_{t+1}) = (\delta_{t+1,1}\sigma'(a_{t+1,1})\boldsymbol{o}_t^t, \ldots, \delta_{t+1,k_{t+1}}\sigma'(a_{t+1,k_{t+1}})\boldsymbol{o}_t^t)$.

Recursive Calculation of the Jocobian $\boldsymbol{\delta}_t = J_{\boldsymbol{o}_t}(\ell_t)$

- Since $\ell_T(\boldsymbol{o}_T) = \Delta(\boldsymbol{o}_T - c(\boldsymbol{x})) = \frac{1}{2}\|\boldsymbol{o}_T - c(\boldsymbol{x})\|^2$, we have

$$\boldsymbol{\delta}_T = J_{\boldsymbol{o}_T}(\ell_T) = \boldsymbol{o}_T^t - c(\boldsymbol{x})^t.$$

- Note that

$$\ell_t(\boldsymbol{o}_t) = \ell_{t+1}(\boldsymbol{o}_{t+1}) = \ell_{t+1}(\boldsymbol{\sigma}(\boldsymbol{a}_{t+1})) = \ell_{t+1}(\boldsymbol{\sigma}(W_t\boldsymbol{o}_t)).$$

- By chain rule,

$$\begin{aligned}\boldsymbol{\delta}_t &= J_{\boldsymbol{o}_t}(\ell_t) = J_{\boldsymbol{\sigma}(W_t\boldsymbol{o}_t)}(\ell_{t+1})J_{\boldsymbol{o}_t}(\boldsymbol{\sigma}(W_t\boldsymbol{o}_t)) \\ &= J_{\boldsymbol{o}_{t+1}}(\ell_{t+1})\mathrm{diag}(\boldsymbol{\sigma}'(W_t\boldsymbol{o}_t))W_t \\ &= \boldsymbol{\delta}_{t+1}\mathrm{diag}(\boldsymbol{\sigma}'(\boldsymbol{a}_{t+1}))W_t.\end{aligned}$$

The Backpropagation Algorithm

```
$\textsc{Backpropagation}(\boldsymbol{x}, \boldsymbol{c}(\boldsymbol{x}), V = \cup_{t=0}^{T} V_t, E, \sigma, \boldsymbol{w})$
 1. for $t \leftarrow 1$ to $T$ do        $\triangleright$ initialization
                                          $V_t = \{v_{t,1}, v_{t,2}, \ldots, v_{t,k_t}\}, 0 \le t \le T$
 2.     for $i \leftarrow 1$ to $k_t$ do
 3.         for $j \leftarrow 1$ to $k_{t-1}$ do
 4.             if $(v_{t-1,j}, v_{t,i}) \in E$ then
 5.                 $W_{t-1,i,j} \leftarrow w((v_{t-1,j}, v_{t,i}))$
 6.             else $W_{t-1,i,j} \leftarrow 0$
 7. $\boldsymbol{o}_0 \leftarrow \boldsymbol{x}$            $\triangleright$ starting forward propagation
 8. for $t \leftarrow 1$ to $T$ do
 9.     for $i \leftarrow 1$ to $k_t$ do
10.         $a_{t,i} \leftarrow \sum_{j=1}^{k_{t-1}} W_{t-1,i,j} o_{t-1,j}$
11.         $o_{t,i} \leftarrow \sigma(a_{t,i})$
12. $\boldsymbol{\delta}_T \leftarrow \boldsymbol{o}_T^t - \boldsymbol{c}(\boldsymbol{x})^t$            $\triangleright$ starting backward propagation
13. for $t \leftarrow T-1$ down to $0$ do
14.     for $i \leftarrow 1$ to $k_t$ do
15.         $\delta_{t,i} \leftarrow \sum_{j=1}^{k_{t+1}} \delta_{t+1,j} \sigma'(a_{t+1,j}) W_{t,j,i}$
```

```
16. for $t \leftarrow 1$ to $T$ do        ▷ generating the gradient
17.     for $i \leftarrow 1$ to $k_t$ do
18.         for $j \leftarrow 1$ to $k_{t-1}$ do
19.             $u_{t-1,i,j} \leftarrow \delta_{t,i}\sigma'(a_{t,i})o_{t-1,j}$
20. return  $\boldsymbol{u}$     ▷ a conditionally unbiased estimate of $\nabla R(\boldsymbol{w})$
    with an input-output pair $(\boldsymbol{x}, \boldsymbol{c}(\boldsymbol{x}))$ of the target concept $\boldsymbol{c}$
```

The Backpropagation Algorithm

- Input:
  - $(\boldsymbol{x}, \boldsymbol{c}(\boldsymbol{x}))$: an input-output pair of the target concept $\boldsymbol{c}$.
  - $(V = \cup_{t=0}^{T} V_t, E, \sigma, \boldsymbol{w})$: a feedforward neural network.
- Initialization:
  - $V_t = \{v_{t,1}, v_{t,2}, \ldots, v_{t,k_t}\}$: the tth layer.
  - $W_{t-1,i,j} = w((v_{t-1,j}, v_{t,i}))$ if $(v_{t-1,j}, v_{t,i}) \in E$ and $W_{t-1,i,j} = 0$ if $(v_{t-1,j}, v_{t,i}) \notin E$.
- Forward iteration: computing the input $\boldsymbol{a}_t$ to and the output $\boldsymbol{o}_t$ from the neurons in the tth layer forward from $t = 0$ to $t = T$.
- Backward iteration: computing the gradient $\boldsymbol{\delta}_t = J_{\boldsymbol{O}_t}(\ell_t)$ of the loss function ℓ_t at $\boldsymbol{o}_t$ backward from $t = T$ to $t = 0$.

- Output: for each each $(v_{t-1,j}, v_{t,i}) \in E$, set the partial derivatives as

$$u_{t-1,i,j} = \frac{\partial}{\partial W_{t-1,i,j}} \Delta(\boldsymbol{h}_{\boldsymbol{w}}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{x})) = \delta_{t,i} \sigma'(a_{t,i}) o_{t-1,j}.$$

Model Selection by Cross-Validation

- The depth and width of a feedforward neural network.
- The regularization parameter λ.
- The step size, fixed or variable.
- These model parameters can be determined by cross-validation.