2

Back Propagation and Initialization

2.1 Review

- Neural-net and formulation; (section 1.3)
- Training difficulty; (section 1.4)

Example 2.1. Consider the multi-layer ($L = 7$) linear neural network with scalar input. The function shape of the loss function $y(w) \triangleq (w^7 - 1)^2$ is presented in Figure (2.1)

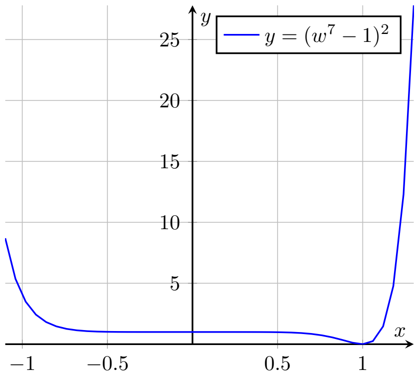


Figure 2.1: Function Shape of $(w^7 - 1)^2$

From Figure (2.1) we can see that when $x \in [-0.5, 0.5]$, the gradient of the loss function *nearly* vanishes; when $x > 1.2$, the gradient explodes into infinite. These two bad region makes the training (optimization) process of such neural network very difficult.

How to rescue the gradient vanishing/explosion during DL training? The "good" region of the loss function is small. There are two ways to rescue this phenomenon:

1. By proper initialization, it's possible to find the good region;

2. By techniques such as *Batch Normalization*, we can change the landscape of the loss function.

2.2 Back Propagation

Suppose that the loss function is of finite-sum form:

$$F(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

with $f_\theta(x_i) = W^L(\phi(W^{L-1}\phi(\cdots\phi(W(x)))))$, and the weight matrices W^ℓ are parameterized by θ. The direct motivation of *back propagation* is to apply gradient descent [1] to minimize the loss function:

$$\theta(t+1) = \theta(t) - \alpha_t \nabla F(\theta(t)).$$

The non-trival part during this process is how to tuning parameters α_t and how to compute $\nabla F(\theta(t))$. The *back propagation* (BP) technique is one efficient strategy to compute the gradient by chain rule, since it avoids repeating the same computations.

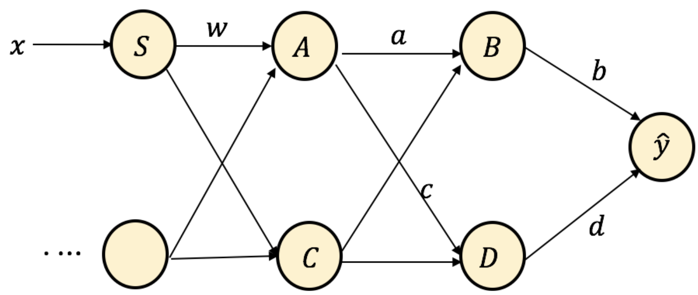
Understanding BP in Level I: Scalar Form of Gradient Most courses/blogs teach how to do BP in scalar version, i.e., to compute the derivative of a scalar-valued function over a scalar variable, which are based on two rules:

[1] Usually we use stochastic gradient descent method in DL since this method is more efficient

- Chain Rule: $f(g(w))$ with $f, g \in \mathbb{R}$,
$$\frac{\mathrm{d}f(g(w))}{\mathrm{d}w} = \frac{\mathrm{d}f}{\mathrm{d}g}\frac{\mathrm{d}g}{\mathrm{d}w}$$
- Sum rule: $g(w) \triangleq f_1(w) + f_2(w)$ with $w \in \mathbb{R}$,
$$\frac{\mathrm{d}g}{\mathrm{d}w} = \frac{\mathrm{d}f_1}{\mathrm{d}w} + \frac{\mathrm{d}f_2}{\mathrm{d}w}$$

We give an example on how to apply these two rules to compute the scalar form of the gradient of the loss function:

Example 2.2. Consider a 2-layer neural network with scalar output. We are interested in computing the derivative of this output $\hat{y}$ over a scalar parameter w. This function w.r.t. w can be represented in graph:



The computation of $\frac{\partial \hat{y}}{\partial w}$ can be summarized as follows:

Step 1: Decompose into multiple paths The path from the parameter w to the output $\hat{y}$ undergoes two paths:
$$w \to A \to B \to \hat{y}$$
$$w \to A \to D \to \hat{y}$$

Step 2: Take gradient of each path by Chain rule These paths corresponds to the functions (w.r.t. w) as follows:
$$f_1(w) = b \cdot \phi(a \cdot \phi(w \cdot x)$$
$$f_2(w) = d \cdot \phi(c \cdot \phi(w \cdot x)$$

The derivative of $f_1(w)$ is computed by the Chain rule:

$$\frac{\partial f_1}{\partial w} = [b \cdot \phi'(a \cdot \phi(w \cdot x_1)] \cdot [a \cdot \phi'(w \cdot x)] \cdot [x]$$

The derivative of $f_2(w)$ can be computed similarly.

The coding is doable in this understanding level.

Understanding BP in Level II: Matrix Form of Gradient Firstly let's review some matrix calculus knowledge by an example.

Example 2.3. Consider a 2-layer linear network[2] $f_\theta(x) = UVx$. Given n data points (x_i, y_i), the goal is to minimize the loss function

$$F \triangleq \frac{1}{n} \sum_{i=1}^{n} \|UVx_i - y_i\|^2,$$

with U, V to be determined. The question is how to take gradient of F w.r.t. the matrix V? Or even simpler, how to compute $\frac{\partial F}{\partial V}$ with $F \triangleq \|UV - Y\|_F^2$? Here suppose that $U \in \mathbb{R}^{d_y \times d_1}$, $V \in \mathbb{R}^{d_1 \times d_x}$, $Y \in \mathbb{R}^{d_y \times d_x}$.

- Let's try to compute the gradient by "standard" Chain rule. Define $H = U \cdot V$, $E = H - Y$, and $F = \|E\|_F^2$.

$$\begin{aligned}\frac{\partial F}{\partial V} &= \frac{\partial F}{\partial E} \frac{\partial E}{\partial H} \frac{\partial H}{\partial V} \\ &= (2E) \cdot I \cdot (U)\end{aligned}$$

  Then check the dimension. We find $E \in \mathbb{R}^{d_y \times d_x}$ and $U \in \mathbb{R}^{d_y \times d_1}$. The matrix-multiplication is undefined! If we want to make the dimension matched, we should write

$$\frac{\partial F}{\partial V} = 2U^{\mathrm{T}}E.$$

Sometimes it's problematic to write gradient by checking matrix dimensions. For instance, if $d_y = d_x$ in practice, this method is invalid.

[2]The weight matrices U, V are parameterized by θ

Another way is to write down scalar-input scalar-output derivatives and then form the whole matrix[3]. However, this way is tedious in practice.

The reason why our method is problematic is probably that we applied the Chain rule incorrectly. Wikipedia provides the Chain rule for vector-valued functions:

Proposition 2.1 (Vector-Function Chain Rule)**.** For vector-input vector-output functions

$$x \in \mathbb{R}^n \to g(x) \in \mathbb{R}^m \to F(x) \triangleq f(g(x)) \in \mathbb{R}^k,$$

the chain rule is

$$\frac{\partial F}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x},$$

where

$$\frac{\partial f(g(x))}{\partial x} = \left[\frac{\partial f_i(g(x_j))}{\partial x_j}\right]_{ij} \in \mathbb{R}^{k\times m}, \quad \frac{\partial g(x)}{\partial x} = \left[\frac{\partial g_i}{\partial x_j}\right]_{ij} \in \mathbb{R}^{m\times n}$$

denotes the Jacobian matrices.

- Consider the objective function $F = \|UVx - y\|_F^2$. The goal is to apply proposition (2.1) to write $\frac{\partial F}{\partial V}$.

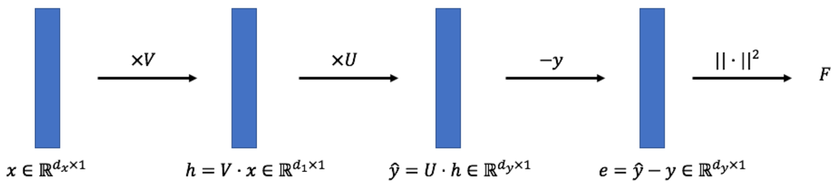


Figure 2.2: Diagram for the operator F

As a result,

$$\frac{\partial F}{\partial V} = \frac{\partial F}{\partial e} \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial V}$$

[3]LeCun, CS224 Note, https://web.stanford.edu/class/cs224n/

In this formula, the LHS is of the form (∂ scalar/∂ matrix), which should be a matrix; the first term in RHS is of the form (∂ scalar/∂ vector), which should be a vector; the second and third term in RHS are of the form (∂ vector/∂ vector), which should be a matrix; the forth term is of the form (∂ vector/∂ matrix), which should be a tensor. Here we discuss the issues for computing these derivatives:

1. Issue 1: computing derivative of a scalar over a vector.
   The issue for computing $\frac{\partial F}{\partial e}$ is on the confusion of the different notions of derivatives.
   - By definition of Jacobian matrices from proposition (2.1), $\frac{\partial F}{\partial e} \in \mathbb{R}^{\text{fan-out}\times\text{fan-in}} = \mathbb{R}^{1\times d_y}$, which is a row vector;
   - By definition of gradient, we assume $\frac{\partial F}{\partial e}$ is a column vector instead, i.e., a vector of dimension $d_y \times 1$.
   - Moreover, the notion of Jacobian and gradient coincides[4] for the case fan-out > 1, e.g.,
   $$\frac{\partial (Wx)}{\partial x} = W.$$

   Based on the issues above, one solution is to define the *general Jacobian* to unify the notions of gradient and Jacobian. Before that, from now on, we define $\frac{\partial f}{\partial x}$ as a row vector if f is scalar-valued, otherwise $\frac{\partial f}{\partial x}$ denotes the Jacobian matrix. Moreover, define the *general Jacobian*
   $$\frac{\tilde{\partial} f}{\tilde{\partial} x} = \begin{cases} \frac{\partial f}{\partial x}, & \text{if fan-out} > 1 \text{ and fan-in} > 1 \\ \left(\frac{\partial f}{\partial x}\right)^{\mathrm{T}}, & \text{if fan-out} = 1 \end{cases}$$

   The proposition (2.1) always holds for *general Jacobian*. Intermediately,
   $$\frac{\tilde{\partial} F}{\tilde{\partial} h} = \frac{\tilde{\partial} F}{\tilde{\partial} e}\frac{\tilde{\partial} e}{\tilde{\partial} h} \implies \left(\frac{\tilde{\partial} F}{\tilde{\partial} h}\right)^{\mathrm{T}} = \left(\frac{\tilde{\partial} e}{\tilde{\partial} h}\right)^{\mathrm{T}} \left(\frac{\tilde{\partial} F}{\tilde{\partial} e}\right)^{\mathrm{T}}$$

[4] At least in some references, e.g., Matrix Differentiation, available at https://atmos.washington.edu/~dennis/MatrixCalculus.pdf

Or equivalently,

$$\begin{array}{cccc} \frac{\partial F}{\partial h} & = & \left(\frac{\partial e}{\partial h}\right)^{\mathrm{T}} & \left(\frac{\partial F}{\partial e}\right) \\ \uparrow & & \uparrow & \uparrow \\ \frac{\partial \text{ scalar}}{\partial \text{ vector}} & & \frac{\partial \text{ vector}}{\partial \text{ vector}} & \frac{\partial \text{ scalar}}{\partial \text{ vector}} \end{array} \tag{2.1}$$

2. Issue 2: computing derivative of a vector over a matrix.

   There are two ways to solve this issue:

   – The first way is to reduce matrix into vectors, i.e., in order to compute $\frac{\partial F}{\partial V}$, it suffices to consider $\frac{\partial F}{\partial V(:,k)}$ and then combine to form a tensor.

   – The other is to use Lemma (2.1) that deals vector-matrix derivative into the vector-vector cases.

   Let's consider the second way in this lecture.

   Lemma 2.1. For $g(V) \triangleq \phi(Vx)$ with $x \in \mathbb{R}^{d\times 1}$ and $V \in \mathbb{R}^{k\times d}$, define $h = Vx$. Then

$$\frac{\partial g}{\partial V} = \frac{\partial \phi}{\partial h} x^{\mathrm{T}}$$

   Now we give an example for applying Lemma (2.1) to compute $\frac{\partial F}{\partial V}$:

$$\begin{aligned} \frac{\partial F}{\partial V} &= \frac{\partial F}{\partial h} x^{\mathrm{T}} && \text{(2.2a)} \\ &= \left(\frac{\partial e}{\partial h}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} && \text{(2.2b)} \\ &= \left(\frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} && \text{(2.2c)} \\ &= (I \cdot U)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} && \text{(2.2d)} \\ &= 2U^{\mathrm{T}} e x^{\mathrm{T}} \end{aligned}$$

   where (2.2a) is because of Lemma (2.1) and $F(V) = F(Vx)$; (2.2b) is by the substitution of (2.1); (2.2c) is by the Chain rule stated in proposition (2.1); (2.2d) is by direct calculation.

Exercise:

$$\frac{\partial \|AWB + C\|_F^2}{\partial W} = 2A^{\mathrm{T}}(AWB + C)B^{\mathrm{T}}$$

BP for General Deep Non-linear Network Now derive the gradient of fully-connected neural network with quadratic loss. The objective f_θ is defined based on the following diagram:

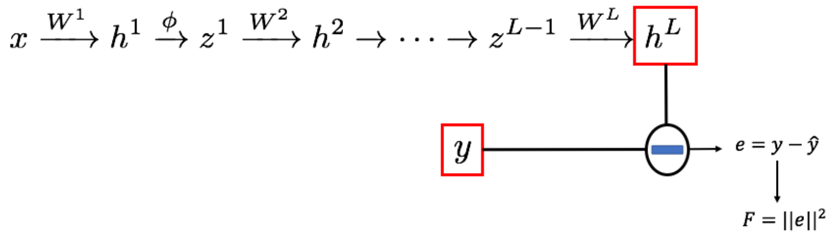


Figure 2.3: Diagram for the operator F

Then the derivative $\frac{\partial F}{\partial W^1}$ is computed as follows:

$$
\begin{aligned}
\frac{\partial F}{\partial W^1} &= \frac{\partial F}{\partial h^1} x^{\mathrm{T}} && \text{(2.3a)}\\
&= \left(\frac{\partial e}{\partial h^1}\right)^{\mathrm{T}} \left(\frac{\partial F}{\partial e}\right) x^{\mathrm{T}} && \text{(2.3b)}\\
&= \left(\frac{\partial e}{\partial h^1}\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} \\
&= \left(\frac{\partial e}{\partial h^L}\frac{\partial h^L}{\partial z^{L-1}} \cdots \frac{\partial h^1}{\partial z^1}\frac{\partial z^1}{\partial h^1}\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} && \text{(2.3c)}\\
&= \left(W^L D^{L-1} W^{L-1} D^{L-2} \cdots W^2 D^1\right)^{\mathrm{T}} 2e \cdot x^{\mathrm{T}} && \text{(2.3d)}
\end{aligned}
$$

where (2.3a) is by Lemma (2.1); (2.3b) follows the similar trick as in (2.1); (2.3c) is by the Chain rule stated in proposition (2.1); in (2.3d) the matrix $D^\ell \triangleq \operatorname{diag}(\phi'(h_i^\ell))_{i=1}^{d_\ell}$, with ϕ' denotes the derivative of ϕ. The general formula $\frac{\partial F}{\partial W^\ell}$ is left as exercise:

$$
\frac{\partial F}{\partial W^\ell} = 2(W^L D^{L-1} \cdots W^{\ell+1} D^\ell)^{\mathrm{T}} \cdot 2e \cdot (z^{\ell-1})^{\mathrm{T}}
$$

This formula can be expressed in a recursive way, which is the mechanism of the BP technique. BP is an efficient way to compute all gradients

$\frac{\partial F}{\partial W^\ell}$ for $\ell = 1, \ldots, L$. The navie computation complexity is $\mathcal{O}(d^2L^2)$; while the BP complexity is $\mathcal{O}(d^2L)$.

2.3 Initialization methods for handling Training Difficulty

We have discussed the *gradient explosion or vanishing* issue. The step size for the gradient descent method is 1 over the Lipschitz constant, which will be super-small/super-large in gradient explosion/vanishing cases. From the landscape in Fig. (2.1), we can see that w^7 grows more active compared with the input $x = 1$. To solve this problem, the direct idea is to control the "energy" of output compared with the input, i.e., for linear network $y = W^L W^{L-1} \cdots W^1 \cdot x$, we want to have

$$\|W^L W^{L-1} \cdots W^1 \cdot x\| \approx \|x\|.$$

Or even simpler, maybe it's enough to let $\|W^\ell x\| \approx \|x\|$ for $\ell = 1, \ldots, L$. Assume W^ℓ is initialized to be a random matrix. After simulation we found that the energy (ℓ_2 norm) for the output after activation is much larger than the previous input.

```
clear;
d = 100; % dimension for weight matrix W
maxit = 10; % maximum iteration number

x = ones(d,1);  norm0 = norm(x);
for i = 1:maxit
    W = randn(d,d);
    x = W*x;
    rato = norm(x)/norm0
end
```

There are different ways to deal with this problem:

- Sparsity Solution: Set many entries of W to be 0;
- Orthogonalization: Generate orthogonal random weight matrix; (to be discussed in the future)
- Scalization: Normalize each entry of W by some constant C.

We find that if each entry of W (assume to be square matrix first) is divided by $\sqrt{d}$, then the energy of $\|W \cdot x\|$ is very close to $\|x\|$.

Informal Xavier Initialization: for the special case where $d = d_x = d_1 = \cdots = d_{L-1} = d_L$, initialize

$$W_{i,j}^{\ell} \sim \mathcal{N}(0,1) \cdot \frac{1}{\sqrt{d}}$$

Supporting Analysis

1. Claim 1: For fixed x, if entries of W are i.i.d. such that

$$W_{i,j} \sim \mathcal{N}(0, 1/d),^{5} \tag{2.4}$$

   then

$$\mathbb{E}\|Wx\|^2 = \|x\|^2.$$

   Proof. Two-line proof: $\mathbb{E}\|Wx\|^2 = x^{\mathrm{T}}\mathbb{E}[W^{\mathrm{T}}W]x$ and evaluate the term $\mathbb{E}[W^{\mathrm{T}}W]$. □

   Sometimes x are also initialized as random number. Therefore, there is a stronger version of claim 1.

2. Claim 2: if x_i's are i.i.d., and previous conditon holds[6], and x is independent of W, then

$$\mathbb{E}\|Wx\|^2 = \|x\|^2.$$

Remark 2.1.

1. If the input and the output dimension are not the same, there is an in-consistency in (2.4). In this case, we try $W_{i,j}^{\ell} \sim \mathcal{N}(0, 2/(d_{\text{fan-in}} + d_{\text{fan-out}}))$. This is the formal Xavier Initialization.

2. The claims 1 and 2 are only about feed-forward neural netowork. For the back-ward case, i.e., $e^1 = (W^L W^{L-1} \cdots W^1)^{\mathrm{T}} e$, we need to have $W_{ij} \sim \mathcal{N}(0, 1/d_{\text{fan-in}})$.

---

[5] for the case fan-in $\neq$ fan-out, use $W_{i,j} \sim \mathcal{N}(0, 1/d_{\text{fan-out}})$

[6] Again, for the case fan-in $\neq$ fan-out, follow the setting in claim 1.

3. The conditions for claims 1 and 2 can be weakened a little bit, e.g., the Gaussian assumptions of W are not needed but only the mean and variance assumptions.

4. For non-linear activation such as relu function, the He Initialization / Kaming Initialization is needed. The initution is that $\mathbb{E}[\mathrm{Relu}(w^2)] = 1/2$. In this case, initialize

$$\mathbb{E}W_{ij}^{\ell} = 0, \quad \mathrm{Var}(W_{ij}^{\ell}) = \frac{2}{\text{fan-in}} \quad \text{or} \frac{2}{\text{fan-out}}$$