Lecture 11: Multi-layer Perceptron

Introduction to Machine Learning [25737]

Sajjad Amini

Sharif University of Technology

Contents

References

Except explicitly cited, the reference for the material in slides is:

- Murphy, K. P. (2022). *Probabilistic machine learning: an introduction.* MIT press.

Section 1

Approach Definition

Approach Definition

Linear Models

- Multinomial logistic regression assume the following model:

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathrm{Cat}(y|\mathcal{S}(\boldsymbol{W}\boldsymbol{x}))$$

- Linear regression assume the following model:

$$p(y|\boldsymbol{x}, \boldsymbol{w}, \sigma^2) = \mathcal{N}(y|\boldsymbol{w}^T\boldsymbol{x}, \sigma^2)$$

One shared feature among both model is linearity.

Increasing Flexibility

To increase fexibility, we can replace input features $\boldsymbol{x}$ with transformed version $\boldsymbol{\phi}(\boldsymbol{x})$ known as **basis function expansion**. Then we have the following model:

$$f(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x})$$

The above model is linear in weight matrix $\boldsymbol{W}$ which makes the estimation easy.

Approach Definition

Toward Automating Transformation (Deep Learning)

- Parameterizing Transformation: $\boldsymbol{\phi}(\boldsymbol{x}) \Rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{\theta})$
  - $\boldsymbol{\phi}([x_1, x_2]^T; [\theta_1, \theta_2]^T = [(\theta_1 + x_1)^2 + (\theta_2 + x_2)^2, \sin(\theta_1 x_1 + \theta_2 x_2)]$
- Applying the transformations in a hierarchical manner:

$$\begin{aligned}
\boldsymbol{z}_1 &= \boldsymbol{\phi}_1(\boldsymbol{z}_0, \boldsymbol{\theta}_1), \boldsymbol{z}_0 = \boldsymbol{x} \\
\boldsymbol{z}_2 &= \boldsymbol{\phi}_2(\boldsymbol{z}_1, \boldsymbol{\theta}_2) \\
&\vdots \\
\boldsymbol{z}_L &= \boldsymbol{\phi}_L(\boldsymbol{z}_{L-1}, \boldsymbol{\theta}_L)
\end{aligned}$$

Altogether we have $\boldsymbol{z}_L = \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\phi}_L(\boldsymbol{\phi}_{L-1}(\ldots \boldsymbol{z}_0 \ldots, \boldsymbol{\theta}_{L-1}), \boldsymbol{\theta}_L)$ where:

$$\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_L)$$

and $\boldsymbol{\phi}_l(\cdot, \boldsymbol{\theta}_l)$ is transformation at layer l.

Section 2

Perceptron Algorithm

Perceptron Algorithm

Binary Logistic Regression

In binary logistic regression, the posterior distribution over labels is modeled as:

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathrm{Ber}(y|\sigma(\boldsymbol{w}^T \boldsymbol{x}))$$

Perceptron

Perceptron is deterministic version of logistic regression (Why??) where the posterior is modeled as:

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathrm{Ber}(y|H(\boldsymbol{w}^T \boldsymbol{x}))$$

where $H(\boldsymbol{w}^T \boldsymbol{x}) = \mathbb{I}(\boldsymbol{w}^T \boldsymbol{x} \geq 0)$ is heaviside step function.

Perceptron Algorithm

Learning Algorithm

The update rule proposed by Rosenblatt for Perceptron is:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t(\widehat{y}_n - y_n)\boldsymbol{x}_n$$

We have seen before the update rule for BLR as:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t(\mu_n - y_n)\boldsymbol{x}_n$$

Perceptron Vs BLR

- Perceptron:
  - No need to compute the probability
  - Convergent when the problem is linearly separable
- BLR
  - $\boldsymbol{\mu}$ is needed for update
  - Always convergent to minimizer of MLE

Perceptron Algorithm

Intuition

Consider Perceptron learning algorithm as:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t(\widehat{y}_n - y_n)\boldsymbol{x}_n$$

Four different cases can occure (assume $\eta_t = 1$):

$$y_n = 1, \widehat{y}_n = 0 \Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{x}_n$$
$$y_n = 0, \widehat{y}_n = 1 \Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{x}_n$$
$$y_n = 0, \widehat{y}_n = 0 \Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t$$
$$y_n = 1, \widehat{y}_n = 1 \Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t$$

Section 3

Multi-layer Perceptron

Perceptron Learning Limitation

XOR Function

Assume XOR function defined as:

$$y = x_1 \oplus x_2 = \begin{cases} 0 & \text{if } x_1 = 0, x_2 = 0 \\ 0 & \text{if } x_1 = 1, x_2 = 1 \\ 1 & \text{if } x_1 = 1, x_2 = 0 \\ 1 & \text{if } x_1 = 0, x_2 = 1 \end{cases}$$

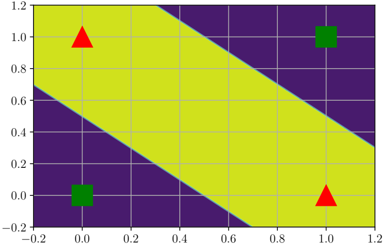

Figure: XOR problem

Quest for Linearly Separable Features

XOR Function

Assume the following transformations:

$$h_1 = x_1 \wedge x_2 = \boldsymbol{w}_1^T \boldsymbol{x} + b_1, \begin{cases} \boldsymbol{w}_1 = [1, 1]^T \\ b_1 = -1.5 \end{cases}$$

$$h_2 = x_1 \vee x_2 = \boldsymbol{w}_2^T \boldsymbol{x} + b_2, \begin{cases} \boldsymbol{w}_2 = [1, 1]^T \\ b_2 = -0.5 \end{cases}$$

Then we can show that

$$y = \overline{h}_1 \wedge h_2 = \overline{(x_1 \wedge x_2)} \wedge (x_1 \vee x_2) = \boldsymbol{w}_2^T \boldsymbol{x} + b_2, \begin{cases} \boldsymbol{w}_3 = [-1, 1]^T \\ b_3 = -0.5 \end{cases}$$

The resulting model is called *Multi-Layer Perceptron* (MLP).

MLP

XOR Function

The final model consist of three Perceptrons, denoted h_1, h_2 and y.

- Hidden unit: h_1 and h_2 are hidden units (Perceptrons) since they are not observed in the training data.
- Output unit: y is output unit (Perceptron).

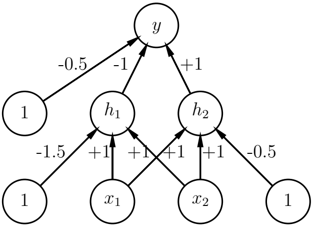

Figure: MLP model for XOR problem

Section 4

Differentiable MLPs

Differentiable MLPs

Problem with MLPs

Training MLP as a stack of Perceptrons is difficult due to non-differentiable Heaviside function.

Differentiable MLPs

Differentiable MLPs are classical MLPs while Heaviside function is replaced with a differentiable function $\varphi : \mathbb{R} \to \mathbb{R}$ known as *Activation Function*.

Differentiable MLPs

Model

Assume the following definitions:

l	Layer number
$\boldsymbol{z}_l$	Hidden units at layer l
$\varphi_l(\cdot)$	Activation function at layer l
K_l	Feature dimension at at layer l

Then the mapping in layer l is:

$$\boldsymbol{z}_l = \boldsymbol{\phi}_l(\boldsymbol{z}_{l-1}, \boldsymbol{\theta}_l) = \varphi_l(\boldsymbol{b}_l + \boldsymbol{W}_l \boldsymbol{z}_{l-1})$$

Note that the quantity passed to activation function is called *pre-activations* defined as:

$$\boldsymbol{a}_l = \boldsymbol{b}_l + \boldsymbol{W}_l \boldsymbol{z}_{l-1}$$

MLP

The term MLP refer to the differentiable MLP rather than non-differentiable version based on Heaviside step function.

Section 5

Activation Functions

Activation Functions

Linear Activation Functions

Assume we select $\varphi_l(a) = c_l a$. Then the whole MLP becomes:

$$
\begin{aligned}
\boldsymbol{z}_1 &= \varphi_1(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) = c_1\boldsymbol{W}_1\boldsymbol{x} + c_1\boldsymbol{b}_1 \\
\boldsymbol{z}_2 &= \varphi_2(\boldsymbol{W}_2\boldsymbol{z}_1 + \boldsymbol{b}_2) = c_2\boldsymbol{W}_2\boldsymbol{z}_1 + c_2\boldsymbol{b}_2 = \underbrace{c_1c_2\boldsymbol{W}_2\boldsymbol{W}_1}_{\boldsymbol{W}_{12}}\boldsymbol{x} + \underbrace{c_1c_2\boldsymbol{W}_2\boldsymbol{b}_1 + c_2\boldsymbol{b}_2}_{\boldsymbol{b}_{12}} \\
&\dots \\
\boldsymbol{z}_L &= \varphi_L(\boldsymbol{W}_L\boldsymbol{z}_{L-1} + \boldsymbol{b}_L) = \boldsymbol{W}_{1\dots L}\boldsymbol{x} + \boldsymbol{b}_{1\dots L}
\end{aligned}
$$

Thus linear activation function reduces to regular linear model. Thus it is important to use nonlinear activation functions.

Sample Activation Functions

Sample Activation Functions

- Sigmoid:

$$\varphi(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$

- Hyperbolic tangent:

$$\varphi(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Rectified linear unit:

$$\varphi(a) = \mathrm{ReLU}(a) = \max(a, 0) = aH(a)$$

Sample Activation Functions

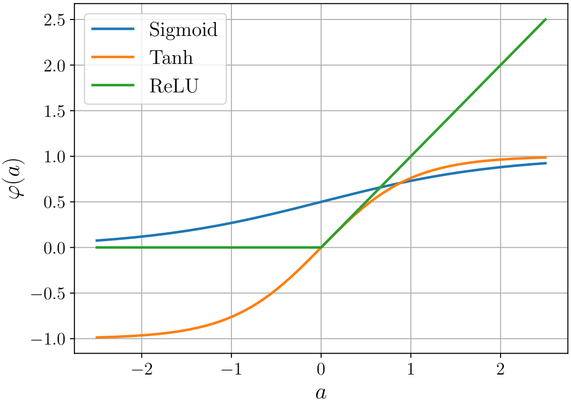

Figure: Sample Activation Functions

Sample MLP

Binary Classification

Consider a binary classification problem with $y \in \{0, 1\}$ and $\boldsymbol{x} \in \mathbb{R}^2$. Assume MLP model with the following features:

- Two hidden layers as:

$$\boldsymbol{z}_1 = \tanh(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1), \begin{cases} \boldsymbol{x} \in \mathbb{R}^2 \\ \boldsymbol{W}_1 \in \mathbb{R}^{4\times 2} \\ \boldsymbol{b}_1, \boldsymbol{z}_1 \in \mathbb{R}^4 \end{cases}$$

$$\boldsymbol{z}_2 = \tanh(\boldsymbol{W}_2 \boldsymbol{z}_1 + \boldsymbol{b}_2), \begin{cases} \boldsymbol{W}_2 \in \mathbb{R}^{3\times 4} \\ \boldsymbol{b}_2, \boldsymbol{z}_2 \in \mathbb{R}^3 \end{cases}$$

- Output layer as:

$$a_3 = \boldsymbol{w}_3^T \boldsymbol{z}_2 + b_3, \begin{cases} \boldsymbol{w}_3 \in \mathbb{R}^3 \\ b_3, a_3 \in \mathbb{R} \end{cases}$$

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathrm{Ber}(y|\sigma(a_3))$$

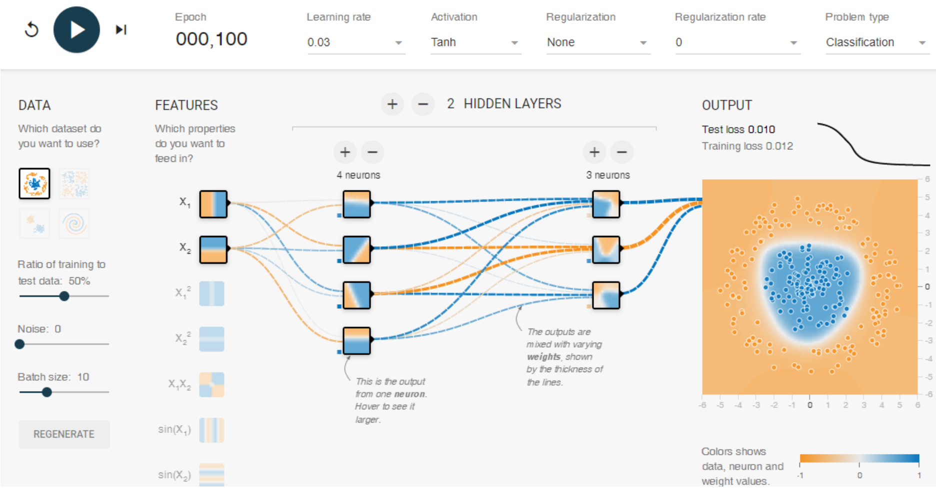

Figure: MLP Visualization

Sample MLP

Multi-class Classification

Consider classifying MNIST dataset [1] where $y \in \{0, 1, \dots, 9\}$ and $\boldsymbol{X} \in \mathbb{R}^{28\times 28}$ (we use the vectorized version of images as $\boldsymbol{x} = \text{vec}(\boldsymbol{X}) \in \mathbb{R}^{784}$). Assume MLP model with the following features:

- Two hidden layers as:

$$\boldsymbol{z}_1 = \tanh(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1), \begin{cases} \boldsymbol{x} \in \mathbb{R}^{784} \\ \boldsymbol{W}_1 \in \mathbb{R}^{128\times 784} \\ \boldsymbol{b}_1, \boldsymbol{z}_1 \in \mathbb{R}^{128} \end{cases}$$

$$\boldsymbol{z}_2 = \tanh(\boldsymbol{W}_2\boldsymbol{z}_1 + \boldsymbol{b}_2), \begin{cases} \boldsymbol{W}_2 \in \mathbb{R}^{128\times 128} \\ \boldsymbol{b}_2, \boldsymbol{z}_2 \in \mathbb{R}^{128} \end{cases}$$

- Output layer as:

$$\boldsymbol{a}_3 = \boldsymbol{W}_3\boldsymbol{z}_2 + \boldsymbol{b}_3, \begin{cases} \boldsymbol{W}_3 \in \mathbb{R}^{10\times 128} \\ \boldsymbol{b}_3, \boldsymbol{a}_3 \in \mathbb{R}^{10} \end{cases}$$

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Cat}(y|\mathcal{S}(\boldsymbol{a}_3))$$

Sample MLP

```
  Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0

_________________________________________________________________
dense (Dense)                (None, 128)               100480

_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512

_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Figure: MLP structure for MNIST classification

Sample MLP

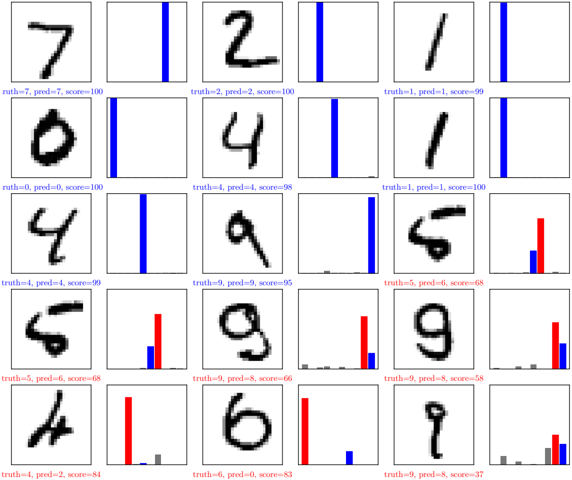

Figure: MLP results for MNIST classification after 1 epoch

Sample MLP

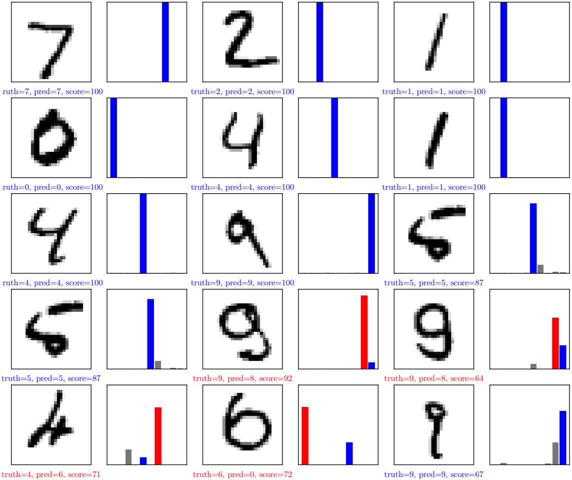

Figure: MLP results for MNIST classification after 2 epoch

Section 6

Backpropagation

NLL for Multi-class Classification

For classification problem using MLP, we assume the following model:

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathrm{Cat}(y|\underbrace{\mathcal{S}(\overbrace{\boldsymbol{W}_L^T\boldsymbol{z}_{L-1} + \boldsymbol{b}_L}^{\boldsymbol{a}_L})}_{\boldsymbol{\mu}_n})$$

Thus the NLL can be formulated as:

$$\mathrm{NLL}(\boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) = -\log \prod_{n=1}^{N}\prod_{c=1}^{C}\mu_{nc}^{y_{nc}} = -\sum_{n=1}^{N}\sum_{c=1}^{C} y_{nc}\log\mu_{nc}$$
$$= \sum_{n=1}^{N}\mathbb{H}(\boldsymbol{y}_n, \boldsymbol{\mu}_n)$$

where $\boldsymbol{y}_n$ is one-hot encoding of the label.

NLL for Regression

For regression problem using MLP, we assume the following model:

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\overbrace{\boldsymbol{w}_L^T \boldsymbol{z}_{L-1} + b_L}^{a_L = \widehat{y}}, \sigma^2)$$

Thus the NLL can be formulated as:

$$\begin{aligned}
\text{NLL}(\boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) &= -\log \prod_{i=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) \\
&= -\log \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_n - \widehat{y})^2\right) \\
&= \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \widehat{y}_n)^2 + \frac{N}{2} \log(2\pi\sigma^2)
\end{aligned}$$

Challenge

Challenge

To minimize $\mathrm{NLL}(\boldsymbol{\theta})$, you need to evaluate the gradient with respect to all parameters. Calculating the gradient when the MLP mapping is complex becomes challenging.

MLP Structure

The structure of MLP is hierarchical. Thus we can reformulate $\mathrm{NLL}(\boldsymbol{\theta})$ in a hierarchical form. Assume a multi-class classification MLP with 2 hidden layers. Then $\mathrm{NLL}(\boldsymbol{\theta})$ can be formulated as:

$$\boldsymbol{f} = f_4 \circ \boldsymbol{f}_3 \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1 \begin{cases} \boldsymbol{f}_1 : \boldsymbol{x} \to \boldsymbol{z}_1 \\ \boldsymbol{f}_2 : \boldsymbol{z}_1 \to \boldsymbol{z}_2 \\ \boldsymbol{f}_3 : \boldsymbol{z}_2 \to \boldsymbol{\mu} \\ f_4 : \boldsymbol{\mu} \to \mathrm{NLL}(\boldsymbol{\theta}) \end{cases}$$

Backpropagation

Backpropagation is an algorithm to compute the gradient of a loss function applied to the output of the network with respect to the parameters in each layer.

Forward vs Reverse Mode Differentiation

Consider mapping $\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{o} \in \mathbb{R}^m$ is defined as:

$$\boldsymbol{f} = \boldsymbol{f}_4 \circ \boldsymbol{f}_3 \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1, \begin{cases} \boldsymbol{f}_1 : \mathbb{R}^n \to \mathbb{R}^{m_1} & \boldsymbol{x}_2 = \boldsymbol{f}_1(\boldsymbol{x}) \\ \boldsymbol{f}_2 : \mathbb{R}^{m_1} \to \mathbb{R}^{m_2} & \boldsymbol{x}_3 = \boldsymbol{f}_2(\boldsymbol{x}_2) \\ \boldsymbol{f}_3 : \mathbb{R}^{m_2} \to \mathbb{R}^{m_3} & \boldsymbol{x}_4 = \boldsymbol{f}_3(\boldsymbol{x}_3) \\ \boldsymbol{f}_4 : \mathbb{R}^{m_3} \to \mathbb{R}^{m} & \boldsymbol{o} = \boldsymbol{f}_4(\boldsymbol{x}_4) \end{cases}$$

Using the chain rule, we have:

$$\begin{aligned} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} &= \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}_4} \frac{\partial \boldsymbol{x}_4}{\partial \boldsymbol{x}_3} \frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{x}_2} \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}} \\ &= \boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4) \boldsymbol{J}_{\boldsymbol{f}_3}(\boldsymbol{x}_3) \boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2) \boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}) \in \mathbb{R}^{m \times n} \end{aligned}$$

Forward vs Reverse Mode Differentiation

$\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})$ matrix can be written in term of columns and row vectors as:

$$\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} - & \nabla f_1(\boldsymbol{x})^T & - \\ & \vdots & \\ - & \nabla f_m(\boldsymbol{x})^T & - \end{bmatrix} = \begin{bmatrix} | & & | \\ \frac{\partial \boldsymbol{f}}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{f}}{\partial x_n} \\ | & & | \end{bmatrix} \in \mathbb{R}^{m \times n}$$

- Reverse Mode Differentiation: Assume $\boldsymbol{e}_i \in \mathbb{R}^m$ to be the unit basis vector. Then the i-th row from $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})$ can be extracted by using vector Jacobian product as:

$$\nabla f_i(\boldsymbol{x})^T = \boldsymbol{e}_i^T \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{e}_i^T \boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4) \boldsymbol{J}_{\boldsymbol{f}_3}(\boldsymbol{x}_3) \boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2) \boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x})$$

- Forward Mode Differentiation: Assume $\boldsymbol{e}_j \in \mathbb{R}^n$ to be the unit basis vector. Then the j-th row from $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})$ can be extracted by using vector Jacobian product as:

$$\frac{\partial \boldsymbol{f}}{\partial x_j} = \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}) \boldsymbol{e}_j = \boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4) \boldsymbol{J}_{\boldsymbol{f}_3}(\boldsymbol{x}_3) \boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2) \boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x}) \boldsymbol{e}_j$$

Forward Mode Differentiation

Forward Mode Differentiation (FMD)

In forward mode differentiation, we are interested in computing each column of $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})$ at query point $\boldsymbol{x}_q$.

- When $n < m$, then it is efficient to use FMD.

Algorithm 1: Forward Mode Differentiation

Initialization: $\boldsymbol{x}_1 = \boldsymbol{x}_q$

$\boldsymbol{v}_j = \boldsymbol{e}_j \in \mathbb{R}^n, \; j = 1, \ldots, n$

begin

  for $k = 1 : K$ **do**

    $\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k)$

    for $j = 1 : n$ **do**

      $\boldsymbol{v}_j = \boldsymbol{J}_{\boldsymbol{f}_k}(\boldsymbol{x}_k)\boldsymbol{v}_j$

    end

  end

end

Output : $\boldsymbol{o} = \boldsymbol{x}_{K+1}, \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$

Forward Mode Differentiation

Forward Mode Differentiation

Consider the following functions:

$$\boldsymbol{f}_1 : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 x_2 \\ x_1 + x_2 \end{bmatrix}, \boldsymbol{f}_2 : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 x_2^2 \\ x_1^2 + x_2^2 \\ \frac{x_1}{x_2} \end{bmatrix}$$

Assume $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_2 \circ \boldsymbol{f}_1$. Compute $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q)$ for $\boldsymbol{x}_q = [1, 1]^T$.

Solution: In this example, $m = 3$ and $n = 2$. Thus $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) \in \mathbb{R}^{3 \times 2}$ and we have the following initializations:

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{v}_1 = \boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{v}_2 = \boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

We also have:

$$\boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_2 & x_1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_2^2 & 2x_1 x_2 \\ 2x_1 & 2x_2 \\ \frac{1}{x_2} & -\frac{x_1}{x_2^2} \end{bmatrix}$$

Forward Mode Differentiation

Forward Mode Differentiation

- $k = 1$:

$$\boldsymbol{x}_2 = \boldsymbol{f}_1(\boldsymbol{x}_1) = \boldsymbol{f}_1\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}1\\2\end{bmatrix},\ \boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$$

$$\boldsymbol{v}_1^{new} = \boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right)\boldsymbol{v}_1^{old} = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$$

$$\boldsymbol{v}_2^{new} = \boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right)\boldsymbol{v}_2^{old} = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$$

Forward Mode Differentiation

Forward Mode Differentiation

- $k = 2$:

$$\boldsymbol{x}_3 = \boldsymbol{f}_2(\boldsymbol{x}_2) = \boldsymbol{f}_2\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}4\\5\\0.5\end{bmatrix}, \; \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix}$$

$$\boldsymbol{v}_1^{new} = \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right)\boldsymbol{v}_1^{old} = \begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}8\\6\\0.25\end{bmatrix}$$

$$\boldsymbol{v}_2^{new} = \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right)\boldsymbol{v}_2^{old} = \begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}8\\6\\0.25\end{bmatrix}$$

Thus we have:

$$\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] = \begin{bmatrix}8 & 8\\6 & 6\\0.25 & 0.25\end{bmatrix}$$

Reverse Mode Differentiation (RMD)

In reverse mode differentiation, we are interested in computing each row of $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x})$ at query point $\boldsymbol{x}_q$.

- When $m < n$, then it is efficient to use RMD.

Algorithm 2: Reverse Mode Differentiation

Initialization: $\boldsymbol{x}_1 = \boldsymbol{x}_q$

$\boldsymbol{u}_i = \boldsymbol{e}_i \in \mathbb{R}^m, \; j = 1, \ldots, m$

begin
    for $k = 1 : K$ **do**
        $\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k)$
    end
    for $k = K : 1$ **do**
        for $i = 1 : m$ **do**
            $\boldsymbol{u}_i^{T,new} = \boldsymbol{u}_i^{T,old} \boldsymbol{J}_{\boldsymbol{f}_k}(\boldsymbol{x}_k)$
        end
    end
end

Output : $\boldsymbol{o} = \boldsymbol{x}_{K+1}, \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) = \begin{bmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_m^T \end{bmatrix}$

Forward Mode Differentiation

Consider our previous functions as:

$$\boldsymbol{f}_1 : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1 x_2 \\ x_1 + x_2 \end{bmatrix}, \boldsymbol{f}_2 : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1 x_2^2 \\ x_1^2 + x_2^2 \\ \frac{x_1}{x_2} \end{bmatrix}$$

Again $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_2 \circ \boldsymbol{f}_1$. Compute $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q)$ for $\boldsymbol{x}_q = [1, 1]^T$.

Solution: In this example, $m = 3$ and $n = 2$. Thus $\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) \in \mathbb{R}^{3\times 2}$ and we have the following initializations:

$$\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{u}_1 = \boldsymbol{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{u}_2 = \boldsymbol{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \boldsymbol{u}_3 = \boldsymbol{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We also have:

$$\boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_2 & x_1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_2^2 & 2x_1 x_2 \\ 2x_1 & 2x_2 \\ \frac{1}{x_2} & -\frac{x_1}{x_2^2} \end{bmatrix}$$

Forward Mode Differentiation

Forward Mode Differentiation

- Forward loop:

$$\boldsymbol{x}_2 = \boldsymbol{f}_1(\boldsymbol{x}_1) = \boldsymbol{f}_1\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}1\\2\end{bmatrix},\ \boldsymbol{x}_3 = \boldsymbol{f}_2(\boldsymbol{x}_2) = \boldsymbol{f}_2\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}4\\5\\0.5\end{bmatrix}$$

- $k = 2$:

$$\boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2) = \boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix}$$

$$\boldsymbol{u}_1^{T,new} = \boldsymbol{u}_1^{T,old}\boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix} = \begin{bmatrix}4 & 4\end{bmatrix}$$

$$\boldsymbol{u}_2^{T,new} = \boldsymbol{u}_2^{T,old}\boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}0 & 1 & 0\end{bmatrix}\begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix} = \begin{bmatrix}2 & 4\end{bmatrix}$$

$$\boldsymbol{u}_3^{T,new} = \boldsymbol{u}_3^{T,old}\boldsymbol{J}_{\boldsymbol{f}_2}\left(\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}4 & 4\\2 & 4\\0.5 & -0.25\end{bmatrix} = \begin{bmatrix}0.5 & -0.25\end{bmatrix}$$

Forward Mode Differentiation

Forward Mode Differentiation

- $k = 1$:

$$\boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x}_1) = \boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$$

$$\boldsymbol{u}_1^{T,new} = \boldsymbol{u}_1^{T,old}\boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}4 & 4\end{bmatrix}\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}8 & 8\end{bmatrix}$$

$$\boldsymbol{u}_2^{T,new} = \boldsymbol{u}_2^{T,old}\boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}2 & 4\end{bmatrix}\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}6 & 6\end{bmatrix}$$

$$\boldsymbol{u}_3^{T,new} = \boldsymbol{u}_3^{T,old}\boldsymbol{J}_{\boldsymbol{f}_1}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}0.5 & -0.25\end{bmatrix}\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix} = \begin{bmatrix}0.25 & 0.25\end{bmatrix}$$

Thus we have:

$$\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_q) = \begin{bmatrix}\boldsymbol{u}_1^T\\ \vdots\\ \boldsymbol{u}_m^T\end{bmatrix} = \begin{bmatrix}8 & 8\\6 & 6\\0.25 & 0.25\end{bmatrix}$$

RMD for MLP

To estimate parameters $\boldsymbol{\theta}$ in MLPs, we have the following optimization problem (for both classification and regression):

$$\widehat{\boldsymbol{\theta}}_{mle} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \;\; \mathrm{NLL}(\boldsymbol{\theta})$$

where $\mathrm{NLL}(\boldsymbol{\theta})$ is a hierarchical mapping. Thus $m = 1$ and $n > 1$ and RMD is more efficient than FMD.

Backpropagation

Hierarchical Structure of MLPs

Assume an MLP with one hidden layer for multi-class classification. Then we can write NLL($\boldsymbol{\theta}$) as:

$$\mathcal{L} = \boldsymbol{f}_4 \circ \boldsymbol{f}_3 \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1$$

where:

$$\begin{aligned} \boldsymbol{x}_2 &= \boldsymbol{f}_1(\boldsymbol{x}, \boldsymbol{W}_1, \boldsymbol{b}_1) = \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1 & \boldsymbol{x}_3 &= \boldsymbol{f}_2(\boldsymbol{x}_2) = \varphi(\boldsymbol{x}_2) \\ \boldsymbol{x}_4 &= \boldsymbol{f}_3(\boldsymbol{x}_3, \boldsymbol{\theta}_3) = \boldsymbol{W}_2 \boldsymbol{x}_3 & \mathcal{L} &= \boldsymbol{f}_4(\boldsymbol{x}_4, \boldsymbol{y}) = \mathbb{H}(\boldsymbol{x}_4, \boldsymbol{y}) \end{aligned}$$

Thus we can compute the gradient with respect MLP parameters using RMD as:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_4} \frac{\partial \boldsymbol{x}_4}{\partial \boldsymbol{W}_2} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_2} \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{W}_1} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_2} \frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{b}_1} \end{aligned}$$

Backpropagation Algorithm

Algorithm 3: Backpropagation for an MLP with K layers

Initialization: $\boldsymbol{x}_1 = \boldsymbol{x}$

begin

  for $k = 1 : K$ **do**

    $\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)$

  end

  $\boldsymbol{u}_{K+1} = 1$

  for $k = K : 1$ **do**

    for $i = 1 : m$ **do**

      $\boldsymbol{g}_k = \boldsymbol{u}_{k+1}^T \frac{\partial \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)}{\partial \boldsymbol{\theta}_k}$

      $\boldsymbol{u}_k^T = \boldsymbol{u}_{k+1}^T \frac{\partial \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)}{\partial \boldsymbol{x}_k}$

    end

  end

end

Output : $\mathcal{L} = \boldsymbol{x}_{K+1}$

$\nabla_{\boldsymbol{x}} \mathcal{L} = \boldsymbol{u}_1$

$\{\nabla_{\boldsymbol{\theta}_k} \mathcal{L} = \boldsymbol{g}_k : k = 1 : K\}$

BP for Common Layers

Cross Entropy Layer

- If we define $\boldsymbol{p} = \mathcal{S}(\boldsymbol{x})$ then the Mapping is:

$$z = f(\boldsymbol{x}) = \mathbb{H}(\boldsymbol{y}, \boldsymbol{x}) = -\sum_{c} y_c \log(\mathcal{S}(\boldsymbol{x})_c) = -\sum_{c} y_c \log p_c$$

where $m = 1$, $n = C$ and $\boldsymbol{J}_f(\boldsymbol{x}) \in \mathbb{R}^{1\times C}$.

- Assume the target label is c, then:

$$z = f(\boldsymbol{x}) = -\log(p_c) = -\log\left(\frac{e^{x_c}}{\sum_j e^{x_j}}\right) = \log\left(\sum_j e^{x_j}\right) - x_c$$

$$\frac{\partial z}{\partial x_i} = \frac{\partial}{\partial x_i} \log \sum_j e^{x_j} - \frac{\partial}{\partial x_i} x_c = \frac{e^{x_i}}{\sum_j e^{x_j}} - \mathbb{I}(i = c)$$

$$\Rightarrow \boldsymbol{J}_f(\boldsymbol{x}) = (\boldsymbol{p} - \boldsymbol{y})^T$$

BP for Common Layers

Elementwise Nonlinearity

- The Mapping is:

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}) = \varphi(\boldsymbol{x}) \Rightarrow z_i = \varphi(x_i),\ i = 1, \ldots, p$$

  where $m = p$, $n = p$ and $\boldsymbol{J}_f(\boldsymbol{x}) \in \mathbb{R}^{p \times p}$.

- The (i, j) element of Jacobian matrix is:

$$\frac{\partial z_i}{\partial x_j} = \begin{cases} \varphi'(x_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \Rightarrow \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}) = \operatorname{diag}(\varphi'(\boldsymbol{x}))$$

BP for Common Layers

Linear layer

- The Mapping is:

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

  where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{z} \in \mathbb{R}^m$ and $\boldsymbol{J}_f(\boldsymbol{x}) = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \in \mathbb{R}^{m \times n}$.
- We know that $z_i = \sum_{k=1}^{n} W_{ik} x_k$, thus (i, j) element of Jacobian matrix is:

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k=1}^{n} W_{ik} x_k = \sum_{k=1}^{n} W_{ik} \frac{\partial}{\partial x_j} x_k = \sum_{k=1}^{n} W_{ik} \mathbb{I}(k = j) = W_{ij}$$

$$\Rightarrow \boldsymbol{J_f}(\boldsymbol{x}) = \boldsymbol{W}$$

BP for Common Layers

Linear layer (Continue)

- Calculating $\frac{\partial \mathcal{L}}{\partial\, \mathrm{vec}(\boldsymbol{W})} = \boldsymbol{u}^T \frac{\partial \boldsymbol{z}}{\partial\, \mathrm{vec}(\boldsymbol{W})}$ where $\boldsymbol{u} \in \mathbb{R}^m$ and $\frac{\partial \boldsymbol{z}}{\partial\, \mathrm{vec}(\boldsymbol{W})} \in \mathbb{R}^{m\times(m\times n)}$
  First, we calculate an arbitrary column in $\frac{\partial \boldsymbol{z}}{\partial\, \mathrm{vec}(\boldsymbol{W})}$ vector:

$$z_k = \sum_{l=1}^{m} W_{kl}x_l + b_k \Rightarrow \frac{\partial z_k}{\partial W_{ij}} = \sum_{l=1}^{m} x_l \frac{\partial}{\partial W_{ij}} W_{kl} = \sum_{l=1}^{m} x_l \mathbb{I}(i=k, j=l)$$

$$\Rightarrow \frac{\partial \boldsymbol{z}}{\partial W_{ij}} = x_j \times \boldsymbol{e}_i = (0, \ldots, x_j, \ldots, 0)^T \in \mathbb{R}^m$$

  Thus the corresponding column in $\frac{\partial \mathcal{L}}{\partial\, \mathrm{vec}(\boldsymbol{W})}$ is:

$$\boldsymbol{u}^T \frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \sum_{k=1}^{m} u_k \frac{\partial z_k}{\partial W_{ij}} = u_i x_j$$

  If we use inverse vectorizing operator, we have:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \boldsymbol{u}\boldsymbol{x}^T \in \mathbb{R}^{m\times n}$$

BP for Common Layers

Linear layer (Continue)

- Calculating $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \boldsymbol{u}^T \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$ where $\boldsymbol{u} \in \mathbb{R}^m$ and $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \in \mathbb{R}^{m \times m}$
  We know:

$$z_k = \sum_{l=1}^{m} W_{kl} x_l + b_k \Rightarrow \frac{\partial z_k}{\partial b_j} = \frac{\partial}{\partial b_j} b_k = \mathbb{I}(j = k) \Rightarrow \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{I} \in \mathbb{R}^{m \times m}$$

  Thus we have:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \boldsymbol{u}^T \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{u}^T \boldsymbol{I} = \boldsymbol{u}^T$$

References I

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner,
"Gradient-based learning applied to document recognition,"
Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.