Mục lục

Chương 1

Practical Aspects of Deep Learning

1.1 Deep L - Layer Neural Network

1.1.1 Notation

1.1.2 Chiều của các Ma trận

1.2 Setting up your Machine Learning Application

1.2.1 Train/Dev/Test sets

1.2.2 Bias/Variance

1.2.3 Basic Recipe for Machine Learning ("Công thức cơ bản" để xử lý High Bias với High Variance)

1.3 Regularizing your Neural Network

1.3.1 Regularization

1.3.2 Cách Regularization giảm Overfitting

1.3.3 Dropout Regularization

1.3.4 Một số cách để giảm Overfitting khác

1.4 Setting up your Optimization Problem

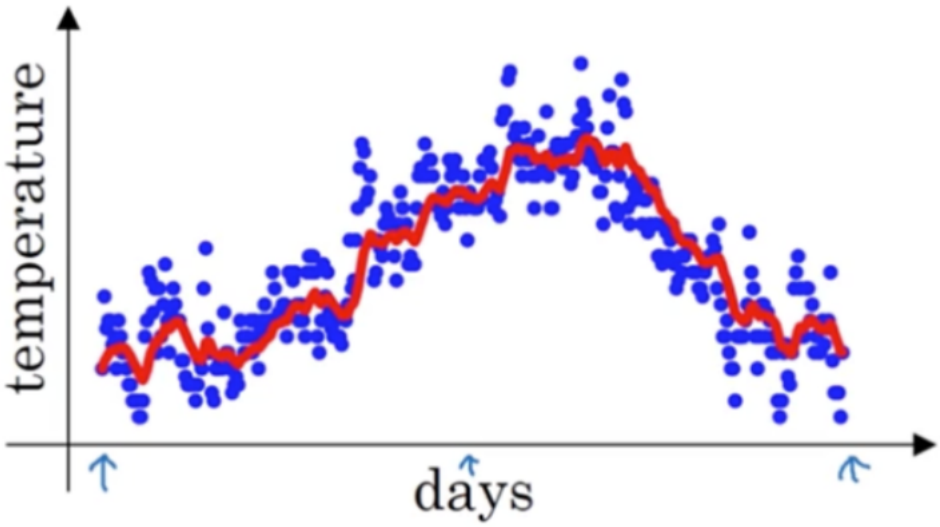
Chương 2

Optimization Algorithms

2.1 Mini-batch Gradient Descent

2.2 Exponentially weighted averages

- Ý tưởng: Khiến cho phân bố của các giá trị trở nên "smooth" hơn
- Tại sao cần đến: Khiến cho thuật toán chạy nhanh hơn, train model nhanh hơn
- Hình ảnh trực quan



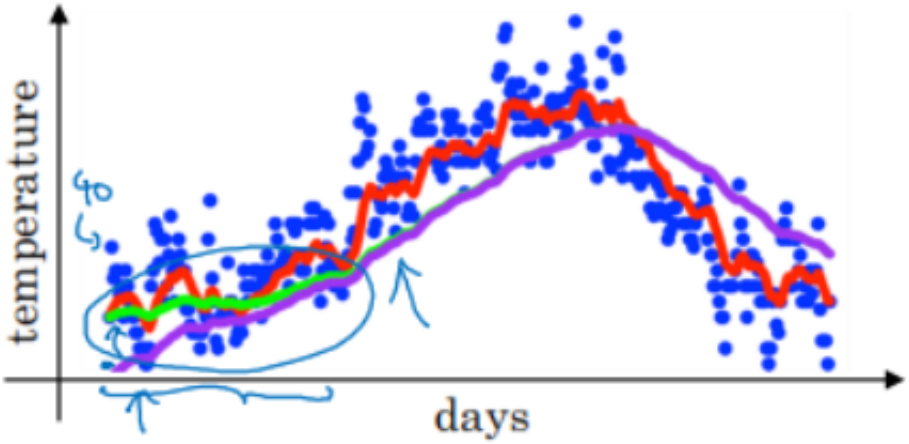
- Công thức:

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

  - β là 1 hyperparameter
  - β tăng thì đồ thị càng "smooth", giảm thì độ thị càng "noisy"
  - θ_t là giá tương ứng tại t

- Implementation:
  $v_\theta = 0$
  repeat {
  get next θ_t
  $v_\theta := \beta v_\theta + (1 - \beta)\theta_t$
  }

- Bias correction:
  - Vấn đề: đồ thị bị thấp hơn so với bao đầu

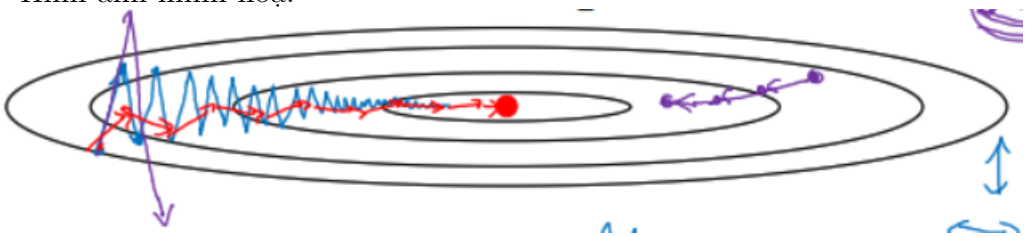


Đường màu tím có điểm bắt đầu rất thấp do khởi tạo $v_\theta = 0$

  - Khắc phục: chia v_θ cho $1 - \beta^t$

2.3 Gradient Descent with Momentum

- Ý tưởng: Thay vì áp dụng trực tiếp Gradient Descent, ta áp dụng Exponential weighted averages lên $d\omega$ và db để khiến thuật toán Gradient Descent chạy nhanh hơn.
  Hình ảnh minh hoạ:



  Biến đường màu xanh thành đường màu đỏ

- Implemetation:
  On iteration t:
  Compute $d\omega, db$ on current mini-batch
  $V_{d\omega} = \beta V_{d\omega} + (1 - \beta)d\omega$
  $V_{db} = \beta V_{db} + (1 - \beta)db$
  $\omega := \omega - \alpha V_{d\omega}, b := b - \alpha V_{db}$

2.4 RMSprop

- Tên đầy đủ: Root Mean Square prop
- Ý tưởng: tương tự như **Gradient Descent with Momentum**
- Implementation:
  On iteration t:
  Compute $d\omega, db$ on current mini-batch

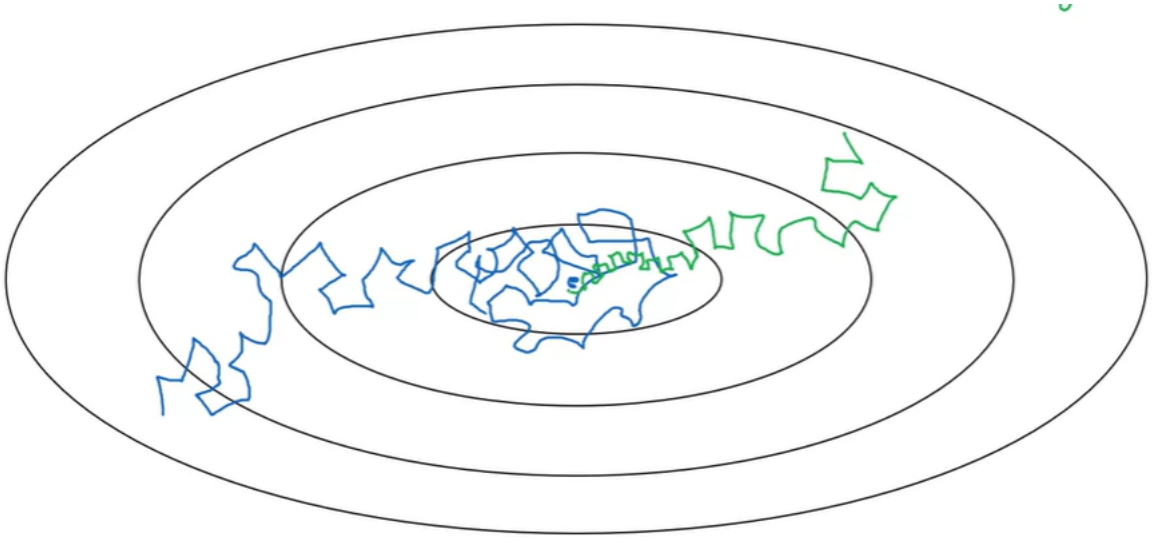
$S_{d\omega} = \beta S_{d\omega} + (1-\beta) d_\omega^2$ (square element-wise)
$S_{db} = \beta S_{db} + (1-\beta) d_b^2$
$\omega := \omega - \alpha \frac{d\omega}{\sqrt{S_{d\omega}}}, b := b - \alpha \frac{db}{\sqrt{S_{db}}}$

2.5 Adam Optimization Algorithm

- Tên đầy đủ: Adaptive Moment Estimation
- ý tưởng: Kết hợp **Gradient Descent with Momentum** và **RMSprop**
- Implementation:
  $V_{d\omega} = 0, S_{d\omega} = 0$
  $V_{db} = 0, S_{db} = 0$
  On iterate t:
  Compute $d\omega, db$ using current mini-batch
  $V_{d\omega} = \beta_1 V_{d\omega} + (1-\beta_1) d\omega, V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ (*moment* with hyperpara β_1)
  $S_{d\omega} = \beta_2 S_{d\omega} + (1-\beta_2) d\omega^2, S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ (*RMSprop* with hyperpara β_2)
  $V_{d\omega}^{corrected} = \frac{V_{d\omega}}{(1-\beta_1^t)}, V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$
  $S_{d\omega}^{corrected} = \frac{S_{d\omega}}{(1-\beta_2^t)}, S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$
  $\omega := \omega - \alpha \frac{V_{d\omega}^{corrected}}{\sqrt{S_{d\omega}^{corrected}} + \epsilon}, b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \epsilon}$
- Hyperarameter:
  - α : cần được điều chỉnh
  - β_1 : 0.9 (default)
  - β_2 : 0.999 (default)
  - ϵ : 10^{-8} (default)

2.6 Learning Rate Decay

- Ý tưởng: thay vì cố định learning rate α, ta giảm dần α theo 1 cách nào đó để khiến Gradient Descent hội tụ nhanh hơn
- Hình ảnh minh hoạ:



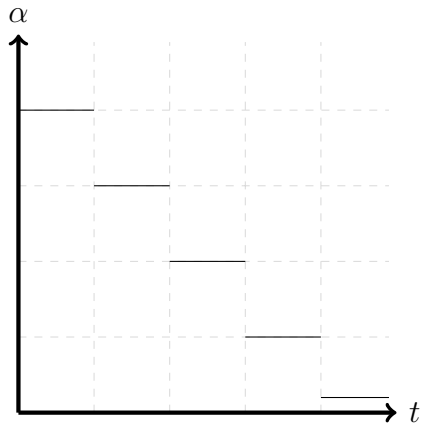
Đường màu xanh - learning rate α cố định
Đường màu xanh lá - learning rate α giảm dần

- Công thức:

$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

  - α_0 : Giá trị learning rate ban đầu
  - decay-rate: Hyperparameter cần tuning
  - epoch-num: 1 epoch là 1 lần duyệt qua toàn bộ dữ liệu, epoch-num là số lần duyệt qua toàn bộ dữ liệu

- Một số công thức khác:
  - $\alpha = 0.95^{\text{epoch-num}} \times \alpha_0$ – exponentically decay
  - $\alpha = \frac{k}{\text{epoch-num}} \alpha_0$ or $\frac{k}{\sqrt{t}} \alpha_0$
  - Discrete Staircase:



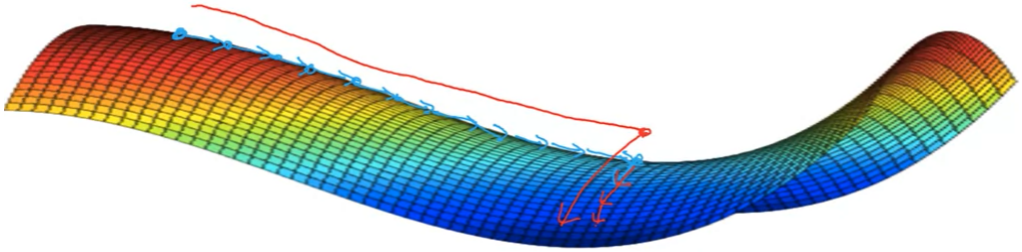
  - Manual Decay: vừa train vừa tự giảm learning rate bằng "cơm"

2.7 The Problem of Local Optima

Chỉ cần biết nhớ là:

- Gần như không thể kẹt ở **Bad local optima**

- **Plateaus** làm chậm learning



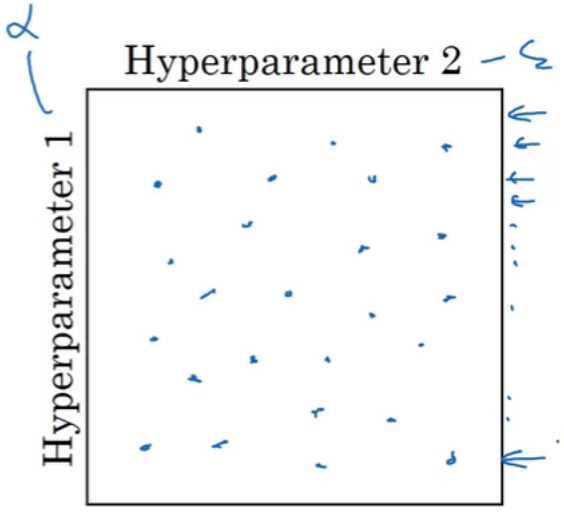
Chương 3

Hyperparameter Tuning, Batch Normalization and Programming Frameworks

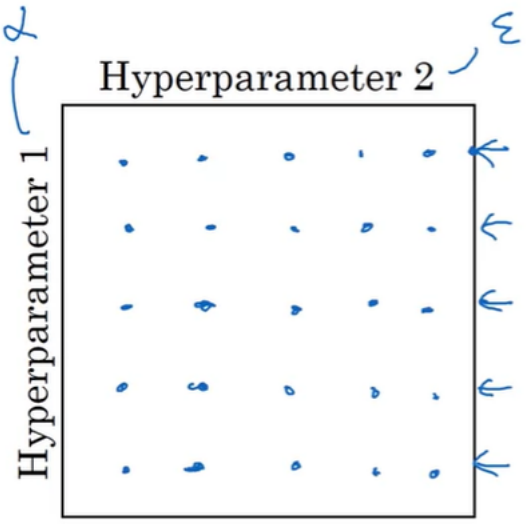
3.1 Hyperparameter Tuning

3.1.1 Tuning Process

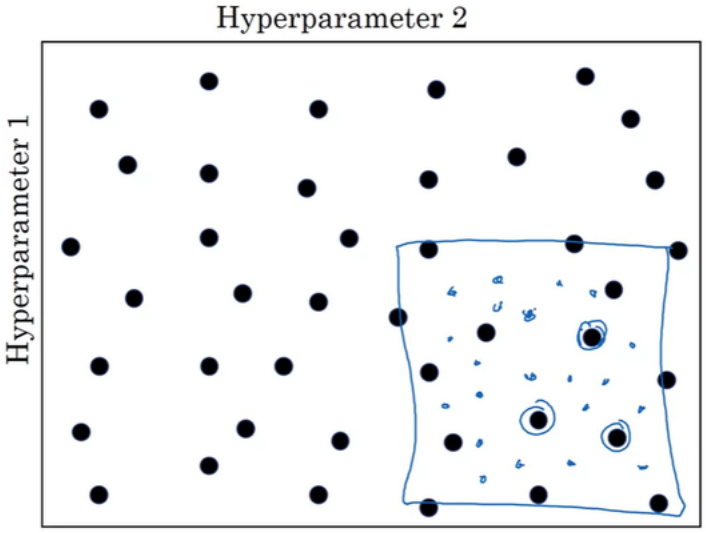
- Ý tưởng: để chọn các giá trị phù hợp của Hyperparameter, ta vẽ theo cặp điểm (ví dụ: (α, ϵ)) lên trục toạ độ (nhiều chiều hơn thì chỉ xử lý theo phương pháp đại số)
- Ví dụ:



Lưu ý: các điểm được khởi tạo ngẫu nhiên vì: Độ quan trọng của Hyperparameter.
Tức là hyperparameter α quan trọng hơn ϵ hay các giá trị ϵ khác nhau không ảnh hưởng đến thuật toán như α thì nếu như xét ví dụ bên dưới (khởi tạo theo grids) sẽ thử được rất ít giá trị của α so với khởi tạo ngẫu nhiên.



- Coarse to fine: Zoom in vào khu vực có các điểm hoạt hoạt động tốt, rồi lại khởi tạo thêm các giá trị **ngẫu nhiên**, rồi lại chạy thử.



3.1.2 Using an Appropriate Scale to pick Hyperparameters

- Ý tưởng: chọn khoảng cho Hyperparameter rồi khởi tạo ngẫu nhiên trong khoảng đó
- Nếu như cứ dải đều trên khoảng được chọn thì rất cả thể 1 khoảng nào đó sẽ có ít giá trị hơn các khoảng còn lại.
  Để khoắc phục thì trên khoảng được chọn, chia nhỏ hơn nữa (theo hàm số mũ $10^{-1}, 10^{-2}, ...$), rồi lấy các giá trị ngẫu nhiên trên các khoảng đó

3.1.3 Hyperparameters Tuning in Practice: Pandas vs. Caviar

- Pandas: train 1 model, mỗi ngày thay đổi hyperparameter 1 tí (phù hợp với dữ liệu lớn, ít phần cứng)
- Caviar: train nhiều model cùng lúc với các hyperparameter giống nhau (phù hợp với nhiều phần cứng)

3.2 Batch Normalization

3.2.1 Normalizing Activations in a Network

- Ý tưởng: không chỉ normalize Input mà còn normalize cả input Z ở các layer để tính ω, b tương ứng nhanh hơn
- Với một giá trị $z^{[l](i)}$ bất kì trong Network:

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$
$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$
$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

  - γ, β là learnable parameters của model
  - Nếu $\gamma = \sqrt{\sigma^2 + \epsilon}$ và $\beta = \mu$ thì $\tilde{z}^{(i)} = z^{(i)}$

3.2.2 Fitting Batch Norm into Neural Network

- Thêm Batch Norm và Network

$$X \xrightarrow{\omega^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{\omega^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{BN}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \to a^{[2]} \to \dots$$

- Xử lý với Mini-batches

$$X^{\{1\}} \xrightarrow{\omega^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{\omega^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{BN}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \to a^{[2]} \to \dots$$

$$X^{\{2\}} \xrightarrow{\omega^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \dots$$

$$X^{\{3\}} \to \dots$$

- Khi áp dụng Batch Norm thì hyperparameter b sẽ bị triệt tiêu nên có thể bỏ luôn từ đầu
- Implementing Gradient Descent
  for $t = 1 \dots$ numMiniBatches
    Compute forward pop on $X^{\{t\}}$
      In each hidden layrer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$
    Use backpop to compute $d\omega^{[l]}, d\beta^{[l]}, d\gamma^{[l]}$ (đã bỏ qua b)
    Update parameters: $\omega^{[l]} = \omega^{[l]} - \alpha d\omega^{[l]}$ (tương tự với $\beta^{[l]}, \gamma^{[l]}$)
  Hoạt động với cả Moment, RMSprop, Adam

3.2.3 Why does Batch Norm work?

- Covariate Shift: vấn đề xảy ra khi phân bố của train set và test set có sự phân bố không đều *(một cách đáng kể)*
- Ảnh hưởng của Covariate Shift lên Neural Network:
  - Không chỉ trên train set và test set, giữa 2 layer của Neural Network cũng xảy ra hiện tượng này.
  - Vì sự biến đối liên tục của hyperparameter ω, b của những layer trước, khiến đầu vào của layer sau chịu ảnh hưởng của Convariate Shift
  - Batch Norm khiến sự thay đổi đó tác động ít hơn, giúp các layer sau có thể học một cách ổn định hơn vì đầu vào ổn định hơn.
- Batch Norm as regularization
  - Batch Norm có một chút khả năng regularization, nhưng không đáng kể.
  - Không thể dựa vào Batch Norm để regularization như Dropout
  - Giống như Dropout, Batch Norm khiến cho $z^{[l]}$ hơi bị "noisy" trong Mini-batch, khiến cho layer sau không thể phụ thuộc hoàn toàn vào unit nào của layer trước cả. **Tuy nhiên vẫn không có ảnh hưởng như Dropout**

3.2.4 Batch Norm at Test Time

- Với test set, chúng ta muốn chạy thử với từng example một chứ không phải là chạy trên từng mini-batch nên không thể dùng các công thức ở mục 3.2.1 để tính toán μ, σ^2 rồi đem đi normalize các input.
- Để khắc phục vấn đề này thì khi train Neural Network với mini-batch, ta keep track các giá trị μ, σ^2 rồi áp dụng exponential weight lên chúng để sau khi train model, ta lưu được 1 cặp giá trị μ và σ^2, rồi đem cặp giá đó đi normalize input ở test set.

3.3 Multi-class Classification

3.3.1 Multi-class Classification

- Khác bài toàn Logistic ở chỗ: **Phân biệt nhiều hơn 2 class**
- Cách làm thông thường:
  - Ở layer cuối của Neural Network sẽ là 1 vecto $(C, 1)$ (C là số class cần phân biệt)
  - Với mỗi phần tử tương ứng với xác suất "Là class đó", phân loại dự trên xác suất cao nhất
  - Ví dụ: Ta có 3 class cần phân biệt: Méo, Cho, Chim, Khác.
    Gán tương ứng: 1, 2, 3, 0
    Thì mỗi phần tử của layer cuối sẽ tương ứng: $P(0|x), P(1|x), P(2|x), P(3|x)$

3.3.2 Softmax Regression

- Là thuật toán dùng trong Multi-class Classification
- Ý tưởng: activation của layer L sẽ trả về 1 vecto $(C, 1)$
- Ở layer L:
  - $z^{[L]} = \omega^{[L]} a^{[L-1]} + b^{[L]}$
  - Activation Function:
    - $t = e^{(z^{[L]})}$
    - $a^{[L]} = \frac{e^{(z^{[L]})}}{\sum_{i=1}^{4} t_i} (a_i^{[L]} = \frac{t_i}{\sum_{i=1}^{4} t_i})$

3.3.3 Train a Softmax Classifier

- Softmax Regression generalizes Logistic Regression to C classes
- Loss function:
  - $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$
  - $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$
  - $J(\omega^{[i]}, b^{[i]}, ...) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
  - $Y = \begin{bmatrix} y^{(1)} & y^{(2)} & ... & y^{(m)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & \\ 1 & 0 & 0 & ... \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix} (4, m)$

    $\hat{Y} = \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} & ... & \hat{y}^{(m)} \end{bmatrix} = \begin{bmatrix} 0.3 & & \\ 0.2 & ... & \\ 0.1 & & \\ 0.4 & & \end{bmatrix} (4, m)$