ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

LƯU QUANG NHIÊN

BAYSIAN CONVOLUTION NEURAL NETWORK

LUẬN VĂN THẠC SĨ TOÁN HỌC

Hồ Chí Minh - 2019

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

LƯU QUANG NHIÊN

BAYSIAN CONVOLUTION NEURAL NETWORK

Chuyên ngành : **CHUYÊN NGÀNH XÁC SUẤT THỐNG KÊ**
Mã ngành : **60460106**
Mã học viên : **16C23010**

Người hướng dẫn 1 : TS.DƯƠNG ĐẶNG XUÂN THÀNH
Người hướng dẫn 2 : TS. Mai Hoàng Bảo Ân

Hồ Chí Minh - 2019

Lời cảm ơn

Tôi xin được bày tỏ lòng biết ơn chân thành và sự kính trọng sâu sắc đến Tiến sĩ Dương Đặng Xuân Thành, Tiến sĩ Mai Hoàng Bảo Ân, hai thầy đã trực tiếp giảng dạy, hướng dẫn và tạo mọi điều kiện trong quá trình học tập và nghiên cứu để tôi có thể hoàn thành luận văn này một cách tốt nhất. Tôi xin chân thành cảm ơn Ban giám hiệu, Phòng Đào tạo sau đại học, Khoa xác suất và thống kê cùng quý thầy cô giáo đã trực tiếp giảng dạy, giúp đỡ tôi trong quá trình học tập tại trường. Mặc dù rất cố gắng nhưng do hạn chế về thời gian, trình độ và kinh nghiệm nghiên cứu nên bên cạnh những kết quả đã đạt được, luận văn không thể tránh khỏi những hạn chế và thiếu sót. Tôi rất mong nhận được sự góp ý của quý thầy cô và các bạn đồng nghiệp để luận văn được hoàn thiện hơn.

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Giới thiệu

Những năm gần đây, AI-Artifical Intelligence (Trí tuệ nhân tạo), và cụ thể hơn là Machine Learning (Học máy) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư. Trí tuệ nhân tạo len lỏi vào mọi lĩnh vực trong cuộc sống mà có thể chúng ta không nhận ra. Xe tự lái, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm trên các trang buôn bán điện tử, ... chỉ là một vài trong vô vàn ứng dụng của AI/Machine Learning.

Machine Learning(Học máy) là một tập con của AI, nó có khả năng tự học hỏi dựa trên những dữ liệu từ lịch sử mà không cần phải lập trình cụ thể. Những năm gần đây, khi mà khả năng tính toán của máy tính được nâng lên tầm cao mới và bộ dữ liệu khổng lồ được thu tập từ các hãng công nghệ mới, Machine Learning đã tiến thêm một bước dài và lĩnh vực mới ra đời gọi là Deep Learning (Học sâu). Deep Learning đã giúp máy tính thực hiện những việc tưởng chừng không thể như phân loại cả ngàn vật thể khác nhau trong một bức ảnh, tự tạo chú thích cho ảnh, chuẩn đoán bệnh thông qua hình ảnh, giao tiếp với con người. Convolutional neural network (CNNs: Mạng thần kinh tích chập) là một biến thể của Deep Neural Network (DNNs: Mạng thần kinh sâu) đã đạt được độ chính xác vượt con người trong nhiệm vụ phân loại ảnh.

Cùng với sự đa dạng của dữ liệu phi tuyến tính, xu hướng các NN hiện nay là càng ngày sâu, càng chồng nhiều lớp với nhau để có thể mô hình hóa được những đặc trưng tốt nhất của dữ liệu. Vì vậy, số lượng tham số hay thể tích của mô hình ngày càng lớn khiến chúng phải cần một khối lượng dữ liệu đào tạo lớn để tránh tình trạng overfitting. Khối lượng lớn ở đây ngoài số lượng còn muốn nói đến sự đa dạng, tổng quát trong dữ liệu và việc bao nhiêu là đủ lớn rất khó để xác định được. Vì vậy, việc mô hình bị overfitting hay không khó để kiểm

soát. Cùng với đó, những mô hình NN hiện tại không có khả năng ước lượng độ không chắc chắn của dự đoán của chúng dẫn đến việc quá tự tin quyết định, đưa ra những dự đoán. Trong luận văn này, tôi sẽ giới thiệu về Baysians Neural Network hay cụ thể hơn là Bayesian Convolutional Neural Network (BayesCNN), thay vì tìm một bộ trọng số tối ưu, BayesCNN ước lượng mỗi trong số một phân phối cụ thể giúp mô hình trở nên tổng quát hơn. BayesCNN không chỉ đạt hiệu suất tương đương mà còn đưa ra một ước lượng về độ không chắc chắn cho từng dự đoán, giúp việc quyết định việc có nên tin vào kết quả dự đoán đó của mô hình hay không?

Nội dung luận văn gồm 6 chương:

Chương 1: Giới thiệu.

Chương 2: Kiến thức chuẩn bị.

Chương 3: Suy luận Baysian

Chương 4: Bayesian Convolutional Neural Networks

Chương 5: Ứng dụng vào bài toán phân loại.

Chương 6: Kết luận và triển vọng.

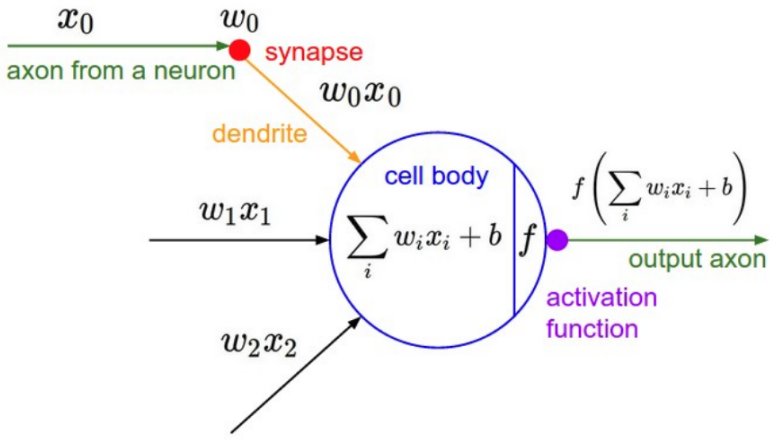
Chương 2

Kiến thức chuẩn bị

2.1 Neural Networks

2.1.1 Brain Analogies

Theo Rosenblatt, một nhà tâm lý học nổi tiếng, tri giác (perceptron) được hình thành như một mô hình toán học về cách mà những tế bào thần kinh (neural) hoạt động trong bộ não của chúng ta. Theo Rosenblatt, một neural nhận một bộ đầu vào nhị phân (từ những neuron gần trước đó), nhân mỗi đầu vào với một trọng số nhất định rồi lấy tổng. Nếu tổng thu được đủ lớn thì đầu ra sẽ là 1, ngược lại là 0 (cách neuron chọn lọc những thông tin quan trọng)

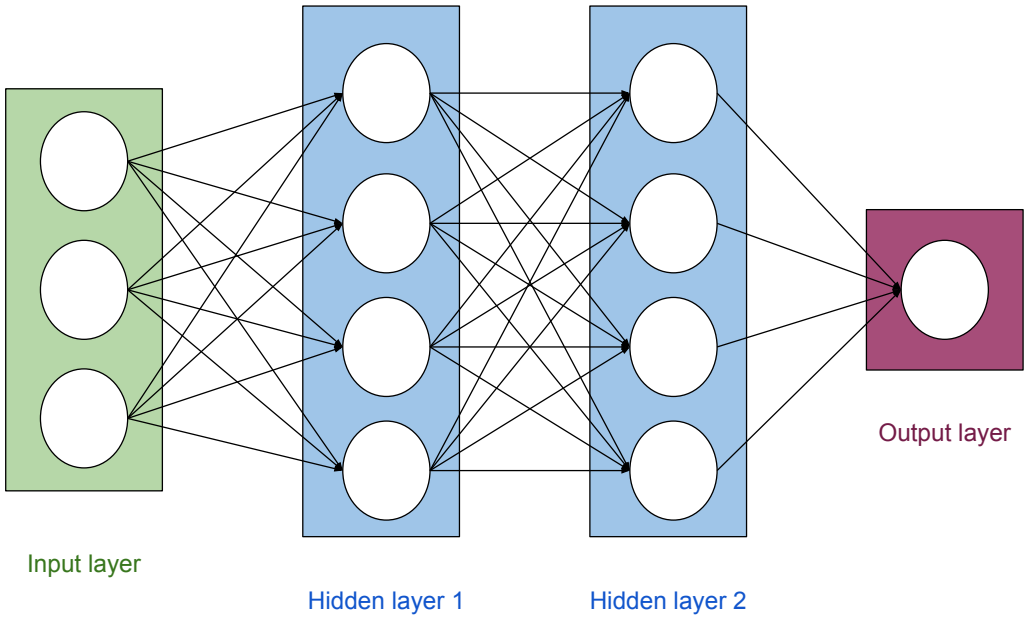


Hình 2.1: Biologically inspired Neural Network [Karparthy, Andrej, 2016]

2.1.2 Mạng lưới thần kinh nhân tạo

Lấy ý tưởng từ mạng lưới thần kinh sinh học, cấu trúc của mạng lưới thần kinh nhân tạo (Neural Network: NN) được phát triển để xử lý thông tin tương tự cách bộ não xử lý thông tin. Một khối lượng lớn các quy trình kết nối các phần tử (neural) làm việc cùng nhau làm cho NN có thể giải quyết các vấn đề phức tạp. Giống như con người học từ các ví dụ, NN cũng vậy. Việc học trong hệ thống sinh học là những điều chỉnh các kết nối khớp thần kinh (synaptic) tương tự như việc cập nhật các trọng số trong NN. Một NN bao gồm 3 lớp: lớp đầu vào (input layer) nơi tiếp nhận dữ liệu, lớp ẩn (hidden layer) để học hỏi cách biểu diễn dữ liệu hợp lý và lớp đầu ra (output) để đưa ra kết quả, dự đoán.Mạng nơ-ron có thể được

coi là hệ thống từ đầu đến cuối để tìm ra các đặc trưng trong dữ liệu quá phức tạp để con người có thể nhận ra để dạy cho máy.



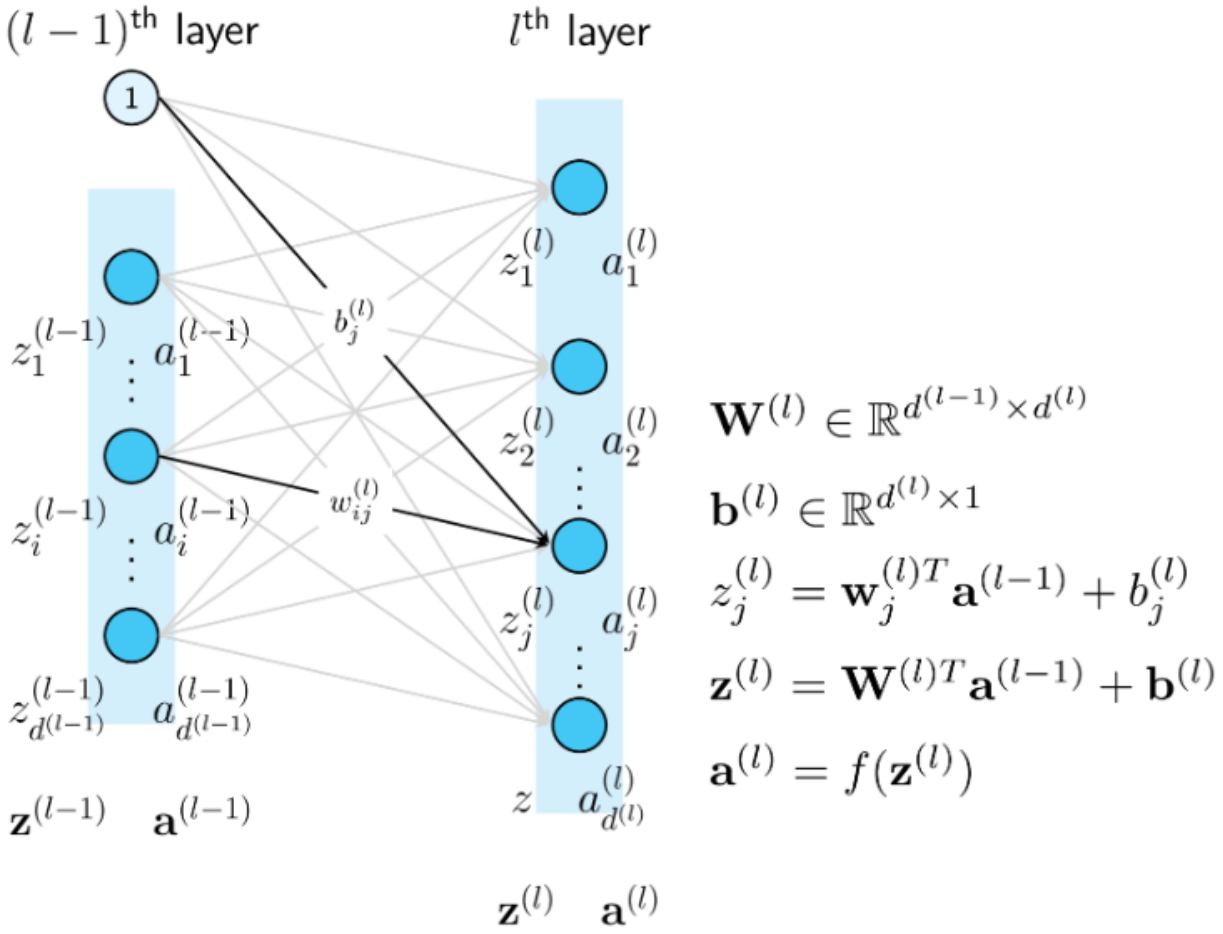
Hình 2.2: MLP với hai lớp ẩn

2.1.2.1 Multi-layer Perceptron (MLP)

Multi-layer Perceptron (MLP) là dạng đơn giản NN, ta đi qua các khái niệm và ký hiệu:

1. Lớp (Layer): Ngoài lớp đầu vào và lớp đầu ra, một MLP có thể có nhiều lớp ẩn (hidden layers) ở giữa. Các lớp ẩn theo thứ tự từ đầu vào đến đầu ra được đánh số thứ tự là Hidden layer 1, Hidden layer 2, ... Hình (??) là một ví dụ về MLP với 2 lớp ẩn

2. Units: Một node hình tròn trong một lớp gọi là unit. Unit ở các lớp đầu vào, lớp ẩn và lớp đầu ra. Đầu vào của các lớp ẩn được ký hiệu bởi z, đầu ra của mỗi unit thường được ký hiệu bằng a (thể hiện hàm activation, tức là giá trị của mỗi unit sau khi ta áp dụng hàm activation). Đầu ra của unit thứ i được ký hiệu là $a_i^{(l)}$. Giả sử thêm rằng số unit trong lớp thứ (l) là $d^{(l)}$. Vector biểu diễn đầu ra của lớp l được ký hiệu là $\mathbf{a}^{(l)} \in \mathbb{R}^{d^{(l)}}$.



Hình 2.3: Các ký hiệu sử dụng trong MLP.

3. Trọng số và Độ lệch (Weight and Bias): Có **L** ma trận trọng số cho một MLP có **L** lớp. Các ma trận này được ký hiệu là $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}, l = 1, 2, ..., L$ trong đó $\mathbf{W}^{(l)}$ thể hiện kết nối từ lớp thứ $l = 1$ tới lớp thứ l (nếu ta coi lớp đầu vào là lớp 0). Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của lớp thứ $(l-1)$ tới node thứ j của lớp thứ (L). Các biases của lớp thứ (l) được ký hiệu là $\mathbf{b}^{(L)} \in \mathbb{R}^{d^{(l)}}$. Các trọng số này được

ký hiệu như trên ??. Khi tối ưu một MLP cho một công việc nào đó, chúng ta cần tìm các trọng số và biases này. Tập hợp các trọng số và biases lần lượt được ký hiện là $\mathbf{W}$ và $\mathbf{b}$.

4. Các hàm kích hoạt (Activation Functions): Mỗi đầu ra của một unit được tính dựa vào công thức:

$$a_i^{(l)} = f(\mathbf{w}_i^{(l)T} a^{(l-1)} + b_i^{(l)}) \tag{2.1.1}$$

Trong đó $f(.)$ là một (không tuyến tính: nonlinear) hàm kích hoạt. Ở dạng vector, biểu thức bên trên được viết là:

$$a^{(l)} = f(\mathbf{W}^{(l)T} a^{(l-1)} + b^{(l)}) \tag{2.1.2}$$

Khi hàm kích hoạt $f(.)$ được áp dụng cho một ma trận (hoặc vector), ta hiểu rằng nó được áp dụng cho từng thành phần của ma trận đó. Sau đó các thành phần này được sắp xếp lại đúng theo thứ tự để được một ma trận có kích thước bằng với ma trận đầu vào.

2.1.2.2 Lan truyền ngược(Backpropagation)

Phương pháp phổ biến nhất để tối ưu MLP vẫn là Gradient Descent(GD). Để áp dụng GD, chúng ta cần tính được gradient của hàm mất mát theo từng ma trận trọng số $\mathbf{W}^{(l)}$ và vector bias $\mathbf{b}^{(l)}$. Trước hết, chúng ta cần tính dự đoán đầu ra $\hat{y}$ với một đầu vào x:

$$\begin{aligned}
\mathbf{a}^{(0)} &= x \\
z_i^{(l)} &= \mathbf{w}_i^{(l)T} a^{(l-1)} + b_i^{(l)} \\
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)T} a^{(l-1)} + b^{(l)}, l = 1, 2, ..., L \\
a^{(l)} &= f(\mathbf{z}^{(l)}), l = 1, 2, ..., L \\
\hat{\mathbf{y}} &= \mathbf{a}^{(L)}
\end{aligned}$$

Bước này gọi là feedforward vì cách tính toán được thực hiện từ đầu đến cuối của mạng. Giả sử $J(\mathbf{W,b,X,Y})$ là hàm mất mát của bài toán, trong đó $\mathbf{W,b}$ là tập hợp tất cả các ma trận trọng số giữa các lớp và biases của mỗi lớp. $\mathbf{X,Y}$ là cặp dữ liệu huấn luyện với mỗi cột

tương ứng với một điểm dữ liệu. Để có thể áp dụng các phương pháp gradient-based (mà Gradient Descent là một ví dụ). chúng ta cần tính được:

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}}; \frac{\partial J}{\partial \mathbf{b}^{(l)}}, l = 1, 2, ..., L$$

Một ví dụ của hàm mất mát là hàm Mean Square Error (MSE) tức là *trung bình của bình phương lỗi*.

$$\begin{aligned} J(\mathbf{W,b,X,Y}) &= \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{y}_n - \hat{\mathbf{y}}_n||_2^2 \\ &= \frac{1}{N}\sum_{n=1}^{N} ||\mathbf{y}_n - \mathbf{a}_n^{(L)}||_2^2 \end{aligned}$$

Với N là số cặp dữ liệu ($\mathbf{x}$,$\mathbf{y}$) trong tập huấn luyện.

Việc tính toán trực tiếp giá trị này rất phức tạp vè hàm mất mát không phụ thuộc trực tiếp vào các hệ số. Phương pháp phổ biến nhất được dùng có thể là Backpropagation giúp tính gradient từ llopws cuối cùng đến lớp đầu tiên. Lớp cuối cùng được tính toán trước vì nó gần gũi hơn với đầu ra của mô hình và hàm mất mát. Việc tính toán gradient của các lớp trước được thực hiện dựa trên quy tắc đạo hàm của hàm hợp.

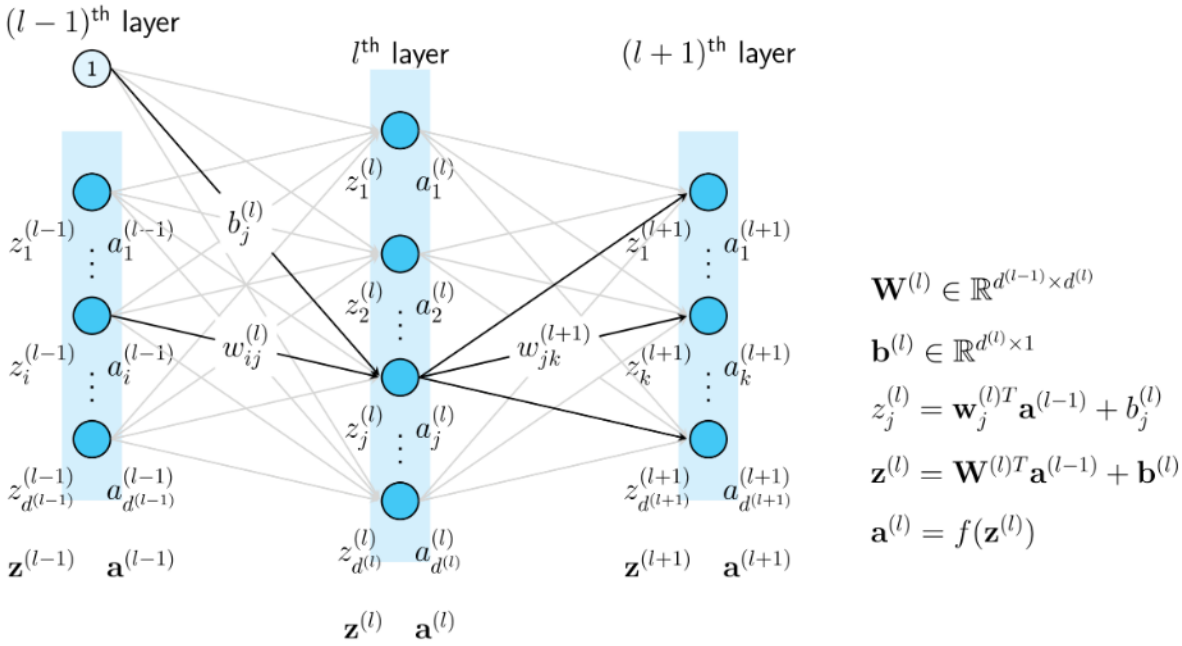
Đạo hàm của hàm mất mát theo chỉ một thành phần của ma trận trọng số của lớp cuối cùng:

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^{(L)}} &= \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} \\ &= e_j^{(L)} a_i^{(L-1)} \end{aligned}$$

Trong đó $e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$ thường là một đại lượng dễ tính toán và $\frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = a_i^{(L-1)}$ vì $z_j^{(L)} = \mathbf{w}_j^{(L)T}\mathbf{a}^{(L-1)} + b_j^{(L)}$. Tương tự như thế, đạo hàm của hàm mất mát theo bias của lớp cuối cùng là:

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$

Dựa vào hình trên, ta có thể tính được:



Hình 2.4: Mô phỏng cách tính backpropagation.

$$\begin{aligned}\frac{\partial J}{\partial w_{ij}^{(l)}} &= \frac{\partial J}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} \\ &= e_j^{(l)} a_i^{(l-1)}\end{aligned}$$

với:

$$\begin{aligned}e_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} &= \frac{\partial J}{\partial a_j^{(l)}} . \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\ &= \Big(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} . \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \Big) f'(z_j^{(l)}) \\ &= \Big(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)} \Big) f'(z_j^{(l)}) \\ &= (\mathbf{w}_j^{(l+1)} \mathbf{e}^{(l+1)}) f'(z_j^{(l)})\end{aligned}$$

trong đó $\mathbf{e}^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, ..., e_{d^{(l+1)}}^{(l+1)}]^T \in \mathbb{R}^{d^{(l+1)}}$ và $\mathbf{w}_{j:}^{(l+1)}$ được hiểu là hàng thứ j của ma

trận $\mathbf{W}^{(l+1)}$ Với cách làm tương tự, ta có thể suy ra:

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}$$

Nhận thấy rằng trong các công thức trên đây, việc tính $e_j^{(l)}$ đóng một vai trò quan trọng, Hơn nữa, để tính được các giá trị này, ta cần tính được các $e_j^{(l+1)}$. Nói cách khác, ta cần tính ngược các giá trị này từ cuối.

2.1.2.3 Convolutional Neural Network

Vấn đề chính của lớp FC là cần quá nhiều trọng số, làm khối lượng của mô hình lớn đối mặt với những vấn đề như thời gian đào tạo (training) và đưa ra dự đoán của mô hình chậm và đặc biệt dễ bị overfitting,... Ví dụ: trọng nhiệm vụ phân loại ảnh, đầu vào là một ảnh có kích thước $64 \times 64 \times 3$, FC cần 12288 trọng số cho một đầu ra trong lớp ẩn đầu tiên. Số lượng trọng số sẽ tăng lên rất nhiều lần khi kích thước ảnh tăng. Ngoài ra, lớp FC còn làm mất thông tin hình học của bức ảnh vì đang xem các điểm ảnh (pixel) độc lập với nhau. Nhưng trong thực tế, các pixel trong từng vùng nhỏ (local) có ảnh hưởng đến nhau. Convolutional Neural Network(CNN) được thiết kế ra để giải quyết các vấn đề của FC trong phân loại ảnh. CNN thường bao gồm các thành phần sau:

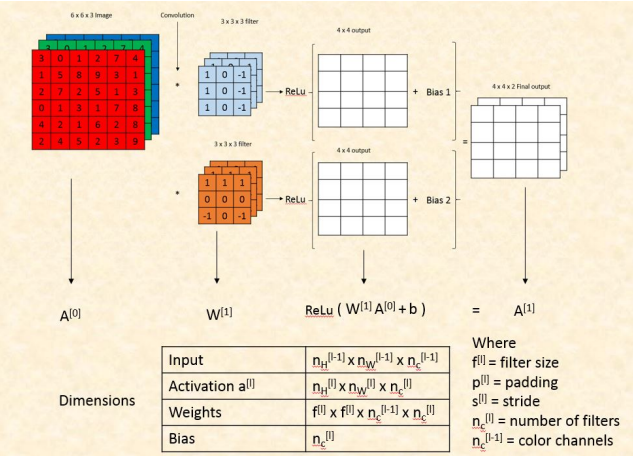
1. Lớp đầu vào: một ảnh RGB có kích thước $H \times W \times 3$ tương ứng $Height \times Width \times Chanels$.

2. Khối convolution: bao gồm

   (a) Lớp convolution: sử dụng một ma trận lọc (kernel hoặc fillters) có kích thước nhỏ, thường là 3×3 hoặc 5×5, trượt qua lần lượt khắp ảnh hoặc bản đồ đặc trưng (feature map), thực hiện phép nhân tích chập cho kernel và một phần nhỏ cùng kích thước với kernel trên ảnh, tổng của ma trận tích vừa thu được được xem như một phần tử đầu ra tương ứng trên lớp đó.

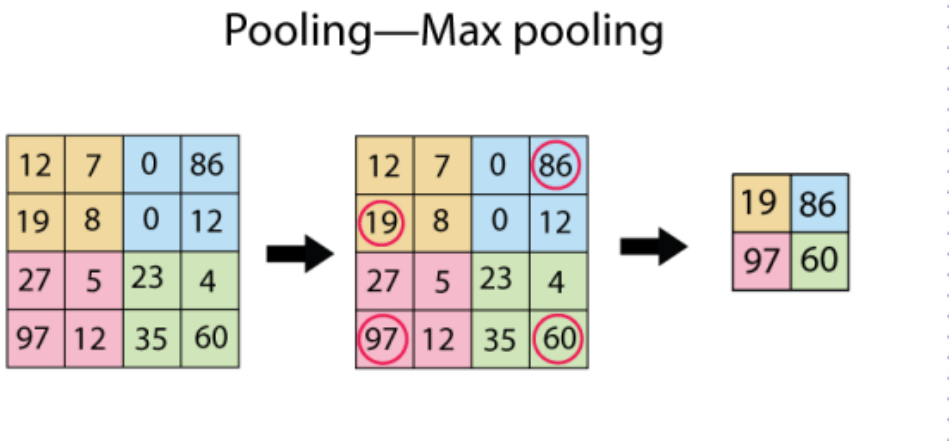
   (b) Hàm kích hoạt (activation function): thường sử dụng hàm ReLU, được mô tả như hàm $max(0, x)$. Có nghĩa là bất kỳ số thực nào bé hơn 0 được chuyển về 0 trong khi những số thực lớn hơn 0 được giữ nguyên giá trị.

   (c) Lớp tổng hợp (Pooling layer): lớp pooling được sử dụng để đơn giản hóa thông tin đầu ra, giảm bớt số lượng neuron. Max-pooling được sử dụng phổ biến nhất,



Hình 2.5: Kích thước tham số của lớp convolution [Karparthy, Andrej, 2016]

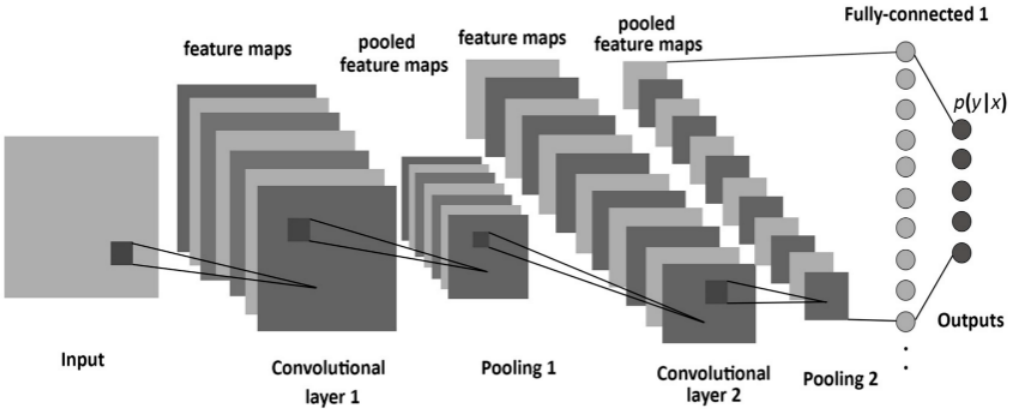
nó chọn giá trị lớn nhất trong vùng đầu vào 2×2. Như vậy, qua lớp max-pooling thì số lượng neuron giảm đi phân nửa. Chúng ta có thể thấy rằng max-pooling lọc những đặc trưng yếu, chỉ giữ lại đặc trưng mạnh nhất.



Hình 2.6: Một ví dụ của lớp max-pooling

3. Lớp FC

Trong một mạng CNN, thông thường các lớp được sắp xếp theo thứ tự: lớp đầu vào + một vài khối convolution + lớp FC + lớp đầu ra.



Hình 2.7: Cấu trúc đầy đủ của một mạng CNN

2.2 Khoảng cách giữa hai phân phối

2.2.1 Entropy và cross-entropy

Entropy là một khái niệm thường gặp trong lý thuyết thông tin. Chúng dùng để đo lượng thông tin mang lại của một sự kiện hay đặc trưng khi biết phân phối xác suất của chúng. Với một phân phối P,

$$Entropy = H(P) = \mathbb{E}_{x \sim P}[-logP(x)]$$

Miến là ta biết phân phối xác suất của bất kỳ cái gì, ta đều có thể tính toán entropy của nó.

Trong trường hợp phân phối xác suất thật P ta không biết hoặc rất khó để tính toán, việc tìm được một phân phối Q xấp xỉ P rất có ý nghĩa. Từ đó, ta cần phải định nghĩa một độ đo giữa hai phân phối để có thể đánh giá được việc xấp sỉ P băng Q có hiệu quả hạy chưa? Ta định nghĩa:

$$CrossEntropy = H(P, Q) = \mathbb{E}_{x \sim P}[-logQ(x)]$$

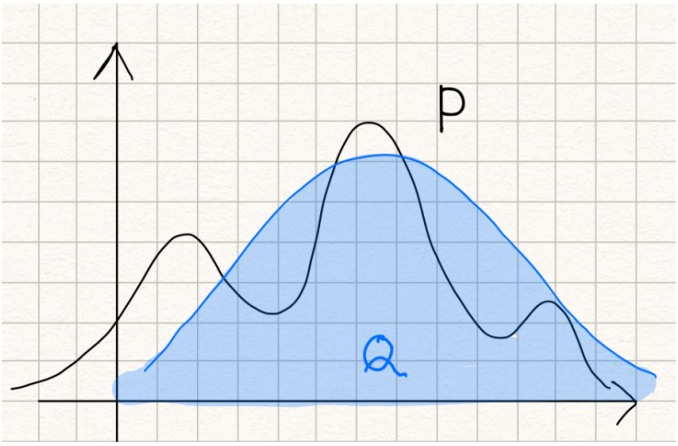
CrossEntropy là lượng thông tin mang lại của của sự kiện có phân phối xấp xỉ Q nhưng được lấy mẫu với phân phối thực P. Lúc này, việc so sánh giữa CrossEntropy và Entropy có ý nghĩa vì chúng đều được lẫy mẫu từ cùng một phân phối.

2.2.2 Phân kỳ Kullback-Leibler

Phân kỳ Kullback-Leibler cho chúng ta biết phân phối Q xấp xỉ phân phối P tốt như thế nào bằng cách tính toán cross-entropy trừ entropy.

$$\begin{aligned} D_K L(P||Q) &= H(P,Q) - H(P) \\ &= \mathbb{E}_{x\sim P}[-logQ(x)] - \mathbb{E}_{x\sim P}[-logP(x)] \\ &= \mathbb{E}_{x\sim P}[logP(x) - logQ(x)] \\ &= \mathbb{E}_{x\sim P}[log\frac{P(x)}{Q(x)}] \end{aligned}$$

Phân kỳ KL đo lượng thông tin chênh lệch hay sự không hiệu quả khi sử dụng phân phối Q để xấp xỉ phân phối P.



Hình 2.8: Xấp xỉ phân phối P bằng Q

Chương 3

Statistical inference

3.1 Bayesian Inference

3.1.1 Định lý Bayes

Định lý Bayes là một định lý cơ bản thường xuyên được sử dụng trong lý thuyết học máy.

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)} \tag{3.1.1}$$

Trong đó: A và B là các sự kiện.

$P(A), P(B)$: Xác suất của sự kiện A, B xảy ra tương ứng.

$P(A/B)$ là xác suất sự kiện A xảy ra nếu biết sự kiện B đã xảy ra

$P(B/A)$ là xác suất sự kiện B xảy ra nếu biết sự kiện A đã xảy ra

Định lý Bayes cho phép chúng ta sử dụng những kiến thức hay niềm tin đã có để tính toán xác suất cho một sự kiện nào đó.

3.1.2 Suy luận Bayesian

Suy luận Bayes là một phương pháp suy luận thống kê mà định lý Bayes được sử dụng để cập nhật xác suất của một giả thuyết khi có thêm nhiều thông tin hay bằng chứng hơn.

Một phần quan trọng của suy luận Bayes là việc thiết lập các tham số và mô hình. Mô hình là một công thức toán học của bộ dữ liệu quan sát được. Tham số là các yếu tố trong mô hình ảnh hưởng đến dữ liệu. Để xây dựng một mô hình, ta cần thiết kế cấu trúc và ước lượng bộ tham số của nó. Giá sử Θ đại diện cho bộ tham số của mô hình, $y = \{y_1, y_2, \ldots, y_n\}$ là bộ dữ liệu mà chúng ta có được. Lúc này, định lý Bayes được viết dưới dạng:

$$P(\Theta/y) = \frac{P(y/\Theta)P(\Theta)}{P(y)} \tag{3.1.2}$$

$P(y/\Theta)$ được gọi là *Likelihood*, mô tả tính hợp lý của giá trị tham số mô hình từ các dữ liệu quan sát được.

$P(\Theta)$ được gọi là phân phối tiền nghiệm hay *prior*, đại diện cho những kiến thức, niềm tin về giá trị thực của tham số từ những kinh nghiệm trước đây khi chưa tiếp xúc với dữ liệu hiện tại.

$P(\Theta/y)$ được gọi là phân phối hậu nghiệm hay *posterior*. Nó đại diện cho phân phối của các tham số sau khi đã quan sát được bộ dữ liệu y.

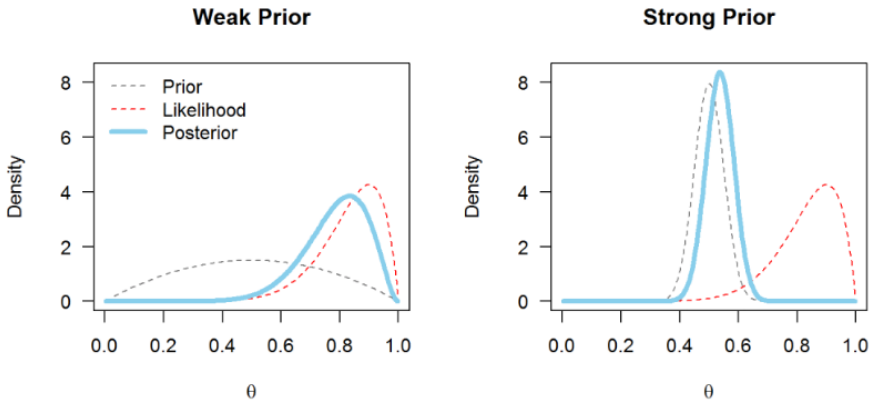
$P(y)$ được gọi là phân phối cận biên hay chứng cứ, nó giúp chuẩn hóa đầu ra của *posterior*, đảm bảo *posterior* thõa điều kiện là một phân phối xác suất.

Trong một vài trường hợp, chúng ta không quan tâm đến việc liệu phân phối có được chuẩn hóa hay không. Khí đó, chúng ta có thể viết lại định lý Bayes dưới dạng rút gọn hơn:

$$P(\Theta/y) \propto P(y/\Theta)P(\Theta) \tag{3.1.3}$$

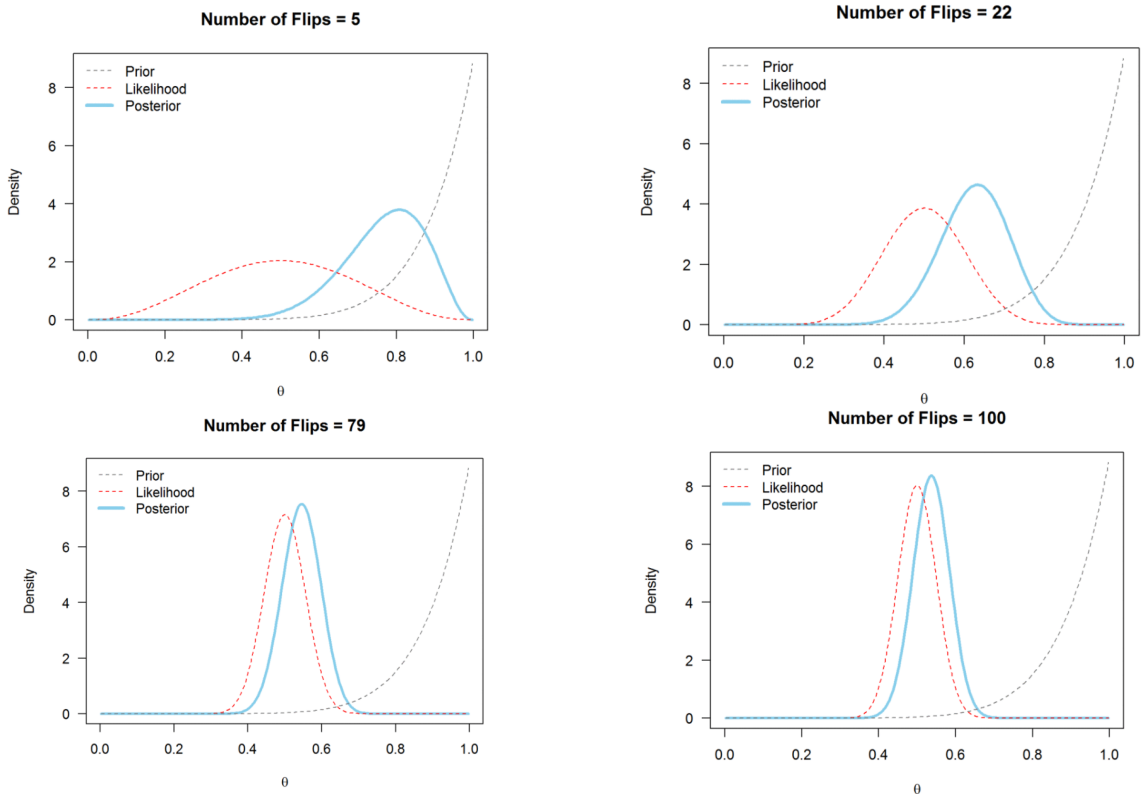
Một số lưu lý về prior *Prior* là một sự đặc biệt trong suy luận Bayes, giúp đưa những kiến thức đã biết hay kiến thức của chuyên gia về Θ vào việc hình thành nên mô hình, làm cho mô hình thu được không phụ thuộc hoàn toàn dữ liệu, ngăn chặn được tình trạng *overfitting* khi có ít dữ liệu.

Prior có nhiệm vụ bảo vệ (một phần nào đó) kiến thức đã có về mô hình khỏi những dữ liệu nhiễu. *Prior* có sức ảnh hưởng mạnh nhất khi mô hình chưa được học với quá nhiều dữ liệu. Trong thực thế, đặc biệt là trong việc đào tạo mô hình, việc nhận định sai về Θ hay



Hình 3.1: Sự ảnh hưởng của một *prior* yếu và một *prior* mạnh khi tung đồng xu được 9/10 mặt ngửa

Prior sai rất dễ xảy ra. Trong trường hợp này, chỉ cần thu thập thật nhiều dữ liệu. Khi ấy, sự ảnh hưởng của Prior ngày càng giảm dần, Posterior vẫn sẽ hội tụ đến đến điểm tối ưu. Điều này được minh họa ở hình ??



Hình 3.2: Cùng với một prior khá lệch, khi ta tung đồng xu càng nhiều thì posterior càng hội tụ đến LikeLihood

3.2 Variational Inference

3.2.1 Probabilistic model

Một mạng thần kinh có thể được xem như mộng mô hình xác suất $p(y/x, w)$. Trong nhiệm vụ phân loại, y là tập hợp các phân lớp và $p(y/x, w)$ là một phân phối phân loại. Trong nhiệm vụ hồi quy, y là một biến liên tục và $p(y/x, w)$ là một phân phối Gaussian.

Cho tập dữ liệu đào tạo $\mathcal{D} = \mathbf{x}^{(i)}, y^{(i)}$, chúng ta có thể xây dựng Likelihood $p(\mathcal{D}/\mathbf{w}) = \prod_i p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$ là một hàm số của $\mathbf{w}$. Cực đại hóa Likelihood (MLE) cũng là một phương pháp hay dùng để tìm $\mathbf{w}$ tối ưu nhưng nó dễ dẫn đến onverfiting vì phụ thuộc hoàn toàn vào dữ liệu.

Nhân Likelihood với một Prior $p(\mathbf{w})$ sẽ tỷ lệ với phân phối Posterior $p(\mathbf{w}/\mathcal{D}) \propto p((D/\mathbf{w})p(\mathbf{w}))$. Cực đại hóa $p((D/\mathbf{w})p(\mathbf{w}))$ tương đương bởi tìm $\mathbf{w}$ để Posterior $p(\mathbf{w}/\mathcal{D})$ đạt giá trị lớn nhất (hay được gọi là MAP). Làm việc với MAP cho hiệu quả tổng quát hơn và có thể ngăn chặn

overfitting.

MLE và MAP đều ước lượng điểm cho các tham số trong mô hình. Thay vào đó, nếu chúng ta có phân phối Posterior đầy đủ cho tất cả các tham số, chúng ta có thể đưa ra nhiều dự đoán với những tham số được lấy ngẫu nhiên từ Posterior. Khi ấy, không những ta thu được dự đoán có tính tổng quát cao mà còn đánh giá được độ không chắc chắn của dự đoán ấy. Khi đó, mục tiêu tối ưu hóa hay hàm mất mát được viết dưới dạng

$$p(y/\mathbf{x}, \mathcal{D}) = \int p(y/\mathbf{x}, \mathbf{w}) p(\mathbf{w}/\mathcal{D}) d\mathbf{w}$$

Điều này tương đương với trung bình các dự đoán đến từ các mạng thần kinh có trong số được lấy từ phân phối Posterior.

3.2.2 Variational Inference

Thật không may, việc ước lượng Posterior $p(y/\mathbf{x}, \mathcal{D})$ trong mạng thần kinh rất khó khăn vì trong đó, số lượng tham số rất lớn, có thể đến hàng triệu tham số. Vì thế, chúng ta phải ước lượng Posterior bằng một phân phối $p(\mathbf{w}/\theta)$ có dạng đơn giản hơn. Việc cực tiểu hóa KL divergence giữa $p(\mathbf{w}/\theta)$ và phân phối Posterior $p(y/\mathbf{x}, \mathcal{D})$ có thể giúp ta làm việc này. Nhiệm vụ của ta là tìm điểm tối ưu θ^* của θ để KL divergence đạt cực tiểu.

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \\
&= \arg\min_\theta \int q(\mathbf{w}|\theta) log \frac{q(\mathbf{w}|\theta)P(\mathcal{D})}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})} d\mathbf{w} \\
&= \arg\min_\theta \int q(\mathbf{w}|\theta)(log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})} - log(P(\mathcal{D}|\mathbf{w})) + log(P(\mathcal{D})))d\mathbf{w} \\
&= \arg\min_\theta \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] + \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D})] \\
&= \arg\min_\theta \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]
\end{aligned}$$

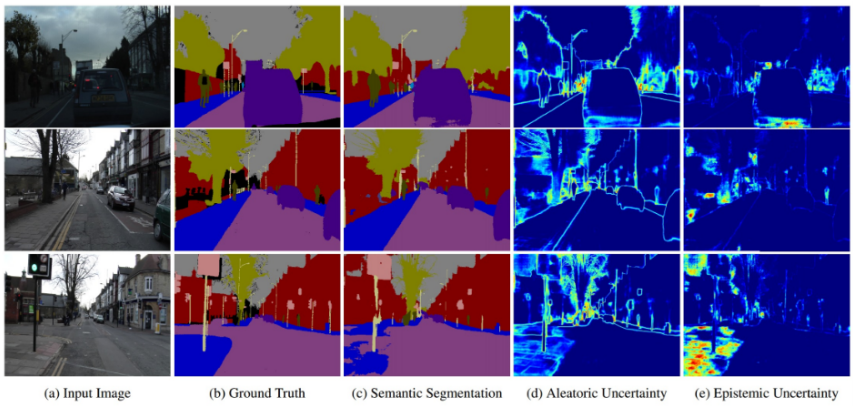
$\log P(\mathcal{D})$ có thể được bỏ qua trong việc tối ưu hóa vì nó là hằng số. Để đơn giản, ta có thể định nghĩa hàm mất mát:

$$\begin{aligned}\mathcal{F}(\mathcal{D},\theta) &= \mathbf{KL}[q(\mathbf{w}|\theta)||p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] \\ &= \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(w|\theta) - \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(w) - \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(\mathcal{D}|\mathbf{w}) \\ &\approx \frac{1}{N}\sum_{i=1}^{N}[\log p(w^{(i)}|\theta) - \log p(w^{(i)}) - \log p(\mathcal{D}|w^{(i)})]\end{aligned}$$

3.3 Uncertainties in Bayesian Learning

Sự không chắc chắn của một mạng thần kinh là độ đo thể hiện sự chắc chắn bao nhiêu của mô hình với dự đoán của nó. Trong mô hình Bayesian, có 2 loại không chắc chắn chính: Aleatoric và Epistemic. Độ không chắc chắn Aleatoric đo tính không chắc chắn có của dữ liệu. Ví dụ, trong các vấn đề với ảnh, các vị trí mà đối tượng bị che khuất hoặc thiếu các đặc điểm nhận dạng thưởng có đô không chắn chắn Aleatoric cao.

Độ không chắc chắn Epistemic thể hiện sự thiếu thông tin về dữ liệu của mô hình. Epistemic cao với những trường hợp mà mô hình chưa được học và ta có thể giảm nó nếu cung cấp đủ dữ liệu.



Hình 3.3: Một ví dụ của uncertainty trong bài toán Semamtation

Tài liệu tham khảo

[Blundell et al., 2015] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

[Der Kiureghian and Ditlevsen, 2009] Der Kiureghian, A. and Ditlevsen, O. (2009). Aleatory or epistemic? does it matter? *Structural Safety*, 31:105–112.

[E. Rumelhart et al., 1986] E. Rumelhart, D., E. Hinton, G., and J. Williams, R. (1986). Learning representations by back propagating errors. *Nature*, 323:533–536.

[Fortunato et al., 2017] Fortunato, M., Blundell, C., and Vinyals, O. (2017). Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*.

[Friston et al., 2007] Friston, K., Mattout, J., Trujillo-Barreto, N., Ashburner, J., and Penny, W. (2007). Variational free energy and the laplace approximation. *Neuroimage*, 34(1):220–234.

[Gal and Ghahramani, 2015] Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

[Graves, 2011] Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.

[Han et al., 2015] Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.

[Hinton and Van Camp, 1993] Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM.

[Houthooft et al., 2016] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arxiv.1605.09674*.

[Karparthy, Andrej, 2016] Karparthy, Andrej (2016). Neural Networks 1. http://cs231n.github.io/neural-networks-1/. Online.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[Lipton et al., 2016] Lipton, Z. C., Gao, J., Li, L., Li, X., Ahmed, F., and Deng, L. (2016). Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*.

[MacKay, 1996] MacKay, D. J. (1996). Hyperparameters: optimize, or integrate out? In *Maximum entropy and bayesian methods*, pages 43–59. Springer.

[Mackay, 1991] Mackay, D. J. C. (1991). A practical bayesian framework for backprop networks.

[Narang et al., 2017] Narang, S., Diamos, G. F., Sengupta, S., and Elsen, E. (2017). Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119.

[Neal and Hinton, 1998] Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[Yedidia et al., 2005] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312.