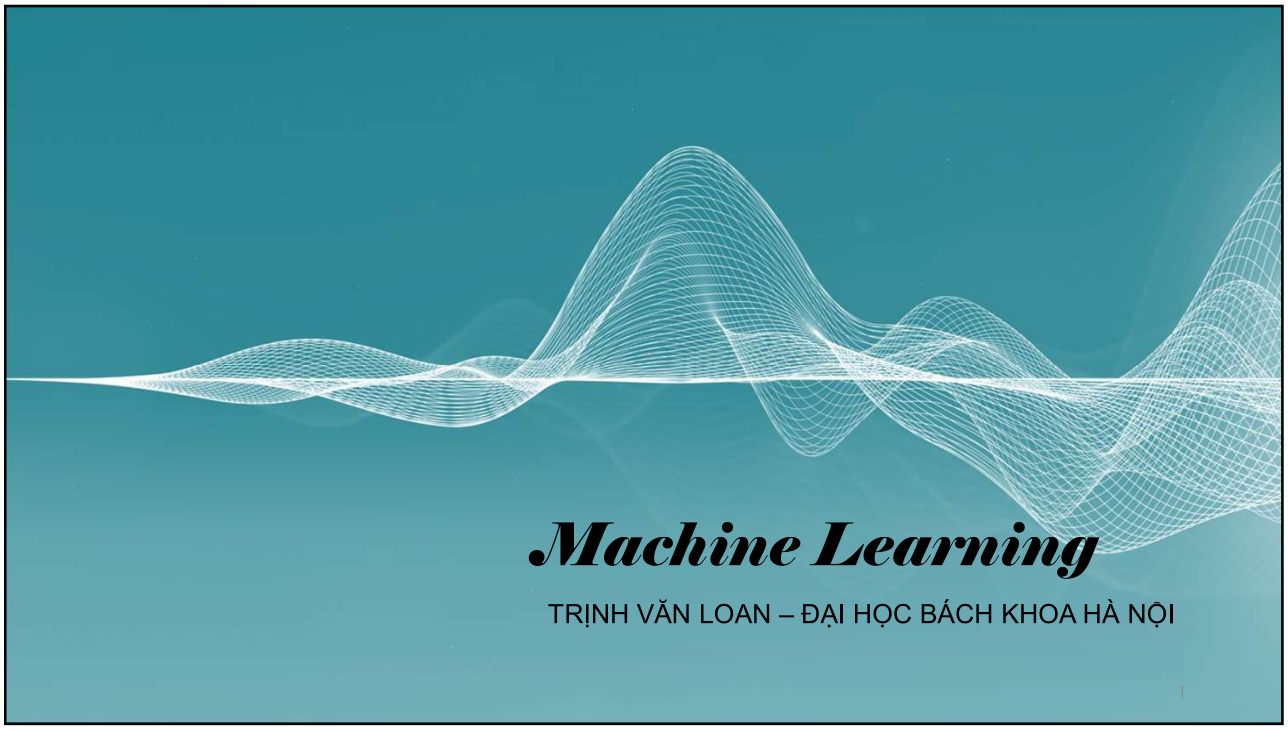
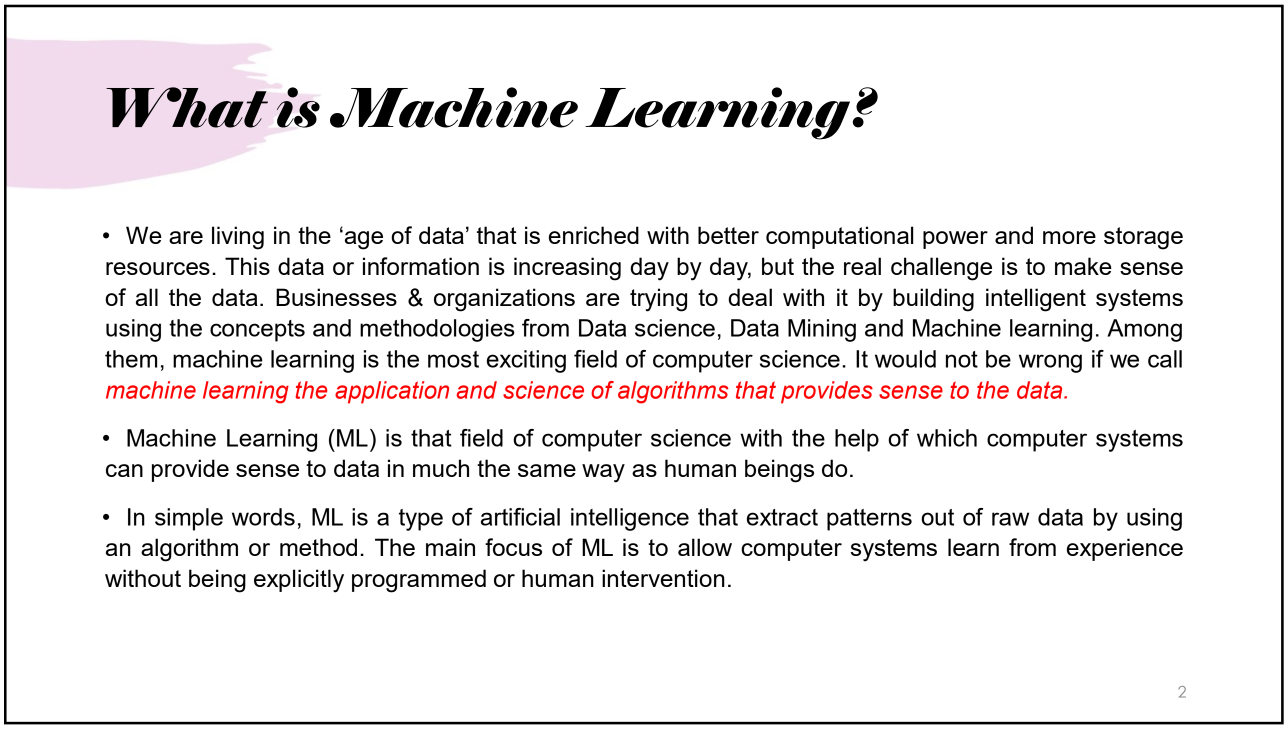


Machine Learning

TRỊNH VĂN LOAN – ĐẠI HỌC BÁCH KHOA HÀ NỘI

1

1



What is Machine Learning?

- We are living in the 'age of data' that is enriched with better computational power and more storage resources. This data or information is increasing day by day, but the real challenge is to make sense of all the data. Businesses & organizations are trying to deal with it by building intelligent systems using the concepts and methodologies from Data science, Data Mining and Machine learning. Among them, machine learning is the most exciting field of computer science. It would not be wrong if we call *machine learning the application and science of algorithms that provides sense to the data.*
- Machine Learning (ML) is that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do.
- In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The main focus of ML is to allow computer systems learn from experience without being explicitly programmed or human intervention.

2

2

Applications of Machines Learning

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML −

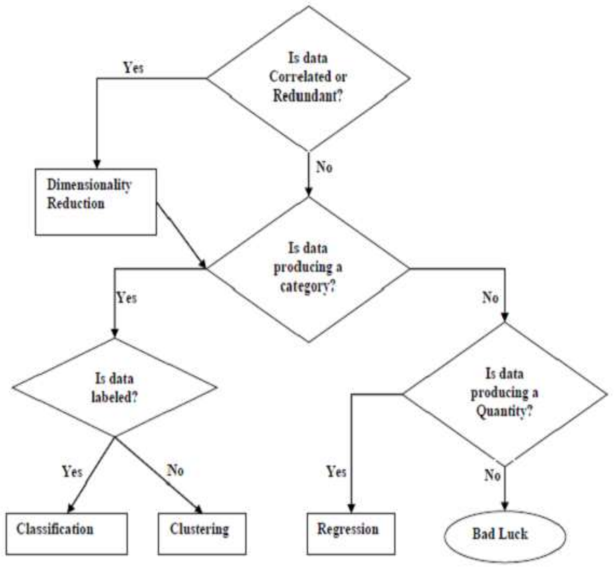
- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping

3

3

Tasks Suited for Machine Learning

- The following diagram shows what type of task is appropriate for various ML problems



4

4

Types of Learning

Learning can be supervised, semi-supervised, unsupervised and reinforcement.

In **supervised learning**, the **dataset** is the collection of **labeled examples** *{(*$\mathbf{x}_i$*,* y_i*)}* *i=1..N*. Each element $\mathbf{x}_i$ among *N* is called a **feature vector**. A feature vector is a vector in which each dimension *j* = 1*, . . . , D* contains a value that describes the example somehow. That value is called a **feature** and is denoted as $x^{(j)}$. For instance, if each example **x** in our collection represents a person, then the first feature, $x^{(1)}$, could contain height in cm, the second feature, $x^{(2)}$, could contain weight in kg, $x^{(3)}$ could contain gender, and so on. For all examples in the dataset, the feature at position *j* in the feature vector always contains the same kind of information. It means that if $x^{(2)}_i$ contains weight in kg in some example $\mathbf{x}_i$, then $x^{(2)}$ will also contain weight in kg in every example $\mathbf{x}_k$, *k* = 1*, . . . , N*. The **label** y_i can be either an element belonging to a finite set of **classes** *{*1*,* 2*, . . . , C}*, or a real number, or a more complex structure, like a vector, a matrix, a tree, or a graph. y_i is either one of a finite set of classes or a real number. You can see a class as a category to which an example belongs. For instance, if your examples are email messages and your problem is spam detection, then you have two classes *{spam, not_spam}*.

Types of Learning

- In **unsupervised learning**, the dataset is a collection of **unlabeled examples** *{*$\mathbf{x}_i$*} i = 1..N* Again, **x** is a feature vector, and the goal of an **unsupervised learning algorithm** is to create a **model** that takes a feature vector **x** as input and either transforms it into another vector or into a value that can be used to solve a practical problem. For example, in **clustering**, the model returns the id of the cluster for each feature vector in the dataset.
  In **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input **x**; in **outlier detection**, the output is a real number that indicates how **x** is di!erent from a “typical” example in the dataset.

Types of Learning

- In **semi-supervised learning**, the dataset contains both labeled and unlabeled examples.
  Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a **semi-supervised learning algorithm** is the same as the goal of the supervised learning algorithm. The hope here is that using many unlabeled examples can help the learning algorithm to find (we might say “produce” or “compute”) a better model

7

7

Types of Learning

- **Reinforcement** learning is a subfield of machine learning where the machine “lives” in an environment and is capable of perceiving the **state** of that environment as a vector of features. The machine can execute **actions** in every state. Different actions bring different **rewards** and could also move the machine to another state of the environment. The goalnof a reinforcement learning algorithm is to learn a **policy**. A policy is a function *f* (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the **expected average reward**.
  Reinforcement learning solves a particular kind of problems where decision making is sequential, and the goal is long-term, such as game playing, robotics, resource management, or logistics.

8

8

How Supervised Learning Works

Supervised learning is used as an example because it's the type of machine learning most frequently used in practice.

The supervised learning process starts with gathering the data. The data for supervised learning is a collection of pairs (input, output). Input could be anything, for example, email messages, pictures, or sensor measurements. Outputs are usually real numbers, or labels (e.g."spam", "not_spam", "cat", "dog", "mouse", etc). In some cases, outputs are vectors (e.g., four coordinates of the rectangle around a person on the picture), sequences (e.g. ["adjective","adjective", "noun"] for the input "big beautiful car"), or have some other structure.
Let's say the problem that you want to solve using supervised learning is spam detection. You gather the data, for example, 10,000 email messages, each with a label either "spam" or "not_spam" (you could add those labels manually or pay someone to do that for us). Now, you have to convert each email message into a feature vector.

How Supervised Learning Works

The data analyst decides, based on their experience, how to convert a real-world entity, such as an email message, into a feature vector. One common way to convert a text into a feature vector, called **bag of words**, is to take a dictionary of English words (let's say it contains 20,000 alphabetically sorted words) and stipulate that in our feature vector:
• the first feature is equal to 1 if the email message contains the word "a"; otherwise, this feature is 0;
• the second feature is equal to 1 if the email message contains the word "aaron"; otherwise, this feature equals 0;
• . . .
• the feature at position 20,000 is equal to 1 if the email message contains the word "zulu"; otherwise, this feature is equal to 0.
You repeat the above procedure for every email message in our collection, which gives us 10,000 feature vectors (each vector having the dimensionality of 20,000) and a label ("spam"/"not_spam").

Neural Network

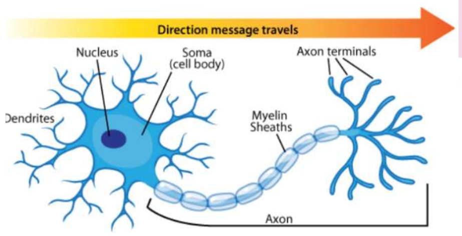
Khái niệm Mạng nơ-ron (Neural Network - NN) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác.

Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites)và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác. => Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không. Giống hàm sigmoid.

Tuy nhiên NN chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình này để giải quyết các bài toán chúng ta cần.

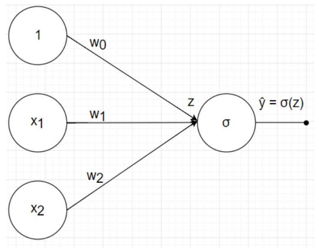


Neural Network Model

Có 2 bước:

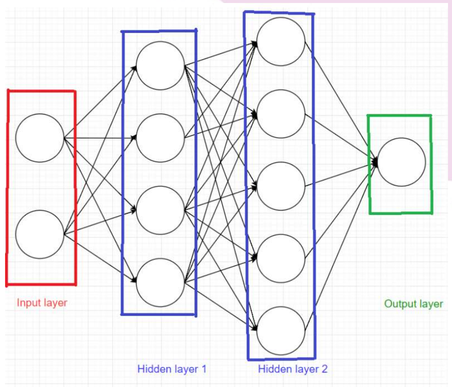
- Tính tổng linear: $z = 1 * w_0 + x_1 * w_1 + x_2 * w_2$
- Áp dụng hàm kích sigmoid function: $\hat{y} = \sigma(z)$

Gộp hai bước trên thành một trên biểu đồ



Neural Network Model

- Hệ số w_0 được gọi là bias (độ lệch). Nếu không có $w0$, phương trình đường thẳng sẽ trở thành $y = w_1 * x$, luôn đi qua gốc tọa độ nên không tổng quát hóa và khó tìm được phương trình mong muốn. Hàm sigmoid được gọi là activation function (hàm kích hoạt).
- **Mô hình tổng quát** Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node (nút).
- Mỗi mô hình luôn có 1 input layer, 1 output layer, các hidden layer có thể có hoặc không. Tổng số layer trong mô hình được quy ước là số layer - 1 (Không tính input layer).
- Ví dụ hình bên có 1 input layer, 2 hidden layer và 1 output layer. Số lượng layer của mô hình là 3 layer .
- Mỗi nút trong hidden layer và output layer :
  - Liên kết với tất cả các nút ở layer trước đó với các hệ số *w* riêng.
  - Mỗi nút có 1 hệ số bias b riêng.
  - Diễn ra 2 bước: tính tổng linear và áp dụng activation function.



13

Neural Network Model

- Số nút trong hidden layer thứ *i* là $l^{(i)}$

Ma trận $W^{(k)}$ kích thước $l^{(k-1)} * l^{(k)}$ là ma trận hệ số giữa layer (k-1) và layer k, trong đó $w_{ij}^{(k)}$ là hệ số kết nối từ node thứ i của layer k-1 đến node thứ j của layer k.

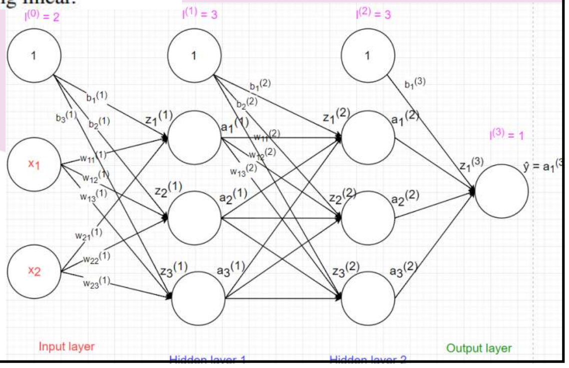
Vector $b^{(k)}$ kích thước $l^k * 1$ là hệ số bias của các node trong layer k, trong đó $b_i^{(k)}$ là bias của node thứ i trong layer k.

Với node thứ i trong layer l có bias $b_i^{(l)}$ thực hiện 2 bước:

- Tính tổng linear: $z_i^{(l)} = \sum_{j=1}^{l^{(l-1)}} a_j^{(l-1)} * w_{ji}^{(l)} + b_i^{(l)}$, là tổng tất cả các node trong layer trước nhân với hệ số w tương ứng, rồi cộng với bias b.
- Áp dụng activation function: $a_i^{(l)} = \sigma(z_i^{(l)})$

Vector $z^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị các node trong layer k sau bước tính tổng linear.

Vector $a^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị của các node trong layer k sau khi áp dụng hàm acti- vation function.



Neural Network Model

Mô hình neural network trên gồm 3 layer. Input layer có 2 node ($l^{(0)} = 2$), hidden layer 1 có 3 node, hidden layer 2 có 3 node và output layer có 1 node.

Do mỗi node trong hidden layer và output layer đều có bias nên trong input layer và hidden layer cần thêm node 1 để tính bias (nhưng không tính vào tổng số node layer có).

Tại node thứ 2 ở layer 1, ta có:

- $z_2^{(1)} = x_1 * w_{12}^{(1)} + x_2 * w_{22}^{(1)} + b_2^{(1)}$
- $a_2^{(1)} = \sigma(z_2^{(1)})$

Hay ở node thứ 3 layer 2, ta có:

- $z_3^{(2)} = a_1^{(1)} * w_{13}^{(2)} + a_2^{(1)} * w_{23}^{(2)} + a_3^{(1)} * w_{33}^{(2)} + b_3^{(2)}$
- $a_3^{(2)} = \sigma(z_3^{(2)})$

15

15

Neural Network Model

Ma trận $W^{(k)}$ kích thước $l^{(k-1)} * l^{(k)}$ là ma trận hệ số giữa layer (k-1) và layer k, trong đó $w_{ij}^{(k)}$ là hệ số kết nối từ node thứ i của layer k-1 đến node thứ j của layer k.

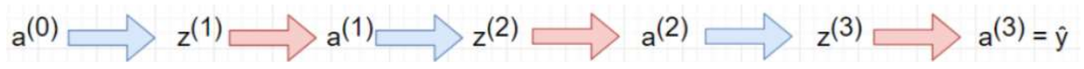
- $W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} (W^{(1)})^{\mathrm{T}} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix}$
- $(W^{(1)})^{\mathrm{T}} * a^{(0)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix} * \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \end{bmatrix} = \begin{bmatrix} w_{11}a_1^{(0)} + w_{21}a_2^{(0)} + w_{31}a_3^{(0)} \\ w_{12}a_1^{(0)} + w_{22}a_2^{(0)} + w_{32}a_3^{(0)} \\ w_{13}a_1^{(0)} + w_{23}a_2^{(0)} + w_{33}a_3^{(0)} \end{bmatrix}$
- $z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = (W^{(1)})^{\mathrm{T}} * a^{(0)} + b^{(1)} = \begin{bmatrix} w_{11}a_1^{(0)} + w_{21}a_2^{(0)} + w_{31}a_3^{(0)} + b_1^{(1)} \\ w_{12}a_1^{(0)} + w_{22}a_2^{(0)} + w_{32}a_3^{(0)} + b_2^{(1)} \\ w_{13}a_1^{(0)} + w_{23}a_2^{(0)} + w_{33}a_3^{(0)} + b_3^{(1)} \end{bmatrix}$

16

16

Neural Network Model

- $a^{(1)} = \sigma(z^{(1)})$ nên tương tự ta có:
- $z^{(2)} = (W^{(2)})^{T} * a^{(1)} + b^{(2)}$
- $a^{(2)} = \sigma(z^{(2)})$
- $z^{(3)} = (W^{(3)})^{T} * a^{(2)} + b^{(3)}$
- $\hat{y} = a^{(3)} = \sigma(z^{(3)})$



$$\begin{tikzcd} a^{(0)} \arrow[r] & z^{(1)} \arrow[r] & a^{(1)} \arrow[r] & z^{(2)} \arrow[r] & a^{(2)} \arrow[r] & z^{(3)} \arrow[r] & a^{(3)} = \hat{y} \end{tikzcd}$$

Feedforward

Image Processing

- Hệ màu RGB: red (đỏ), green (xanh lục), blue (xanh lam), là ba màu cơ bản. Khi trộn ba màu này theo tỉ lệ nhất định có thể tạo thành các màu khác nhau. Với mỗi bộ 3 số r, g, b nguyên trong khoảng [0, 255] sẽ cho ra một màu khác nhau. Do có 256 cách chọn r, 256 cách chọn màu g, 256 cách chọn b => tổng số màu có thể tạo ra bằng hệ màu RGB là: 256 * 256 * 256 = 16777216 màu
- Kích thước ảnh tính bằng pixel, ví dụ 800x600 pixel (chiều rộng 800 pixel, chiều cao 600 pixel), mỗi pixel là 1 điểm ảnh, mỗi điểm ảnh trên ảnh màu có màu sắc nhất định tương ứng với tỷ lệ 3 thành phần R,G,B.

Image Processing

- Tương ứng có 3 ma trận điểm ảnh ứng với 3 màu cơ bản:

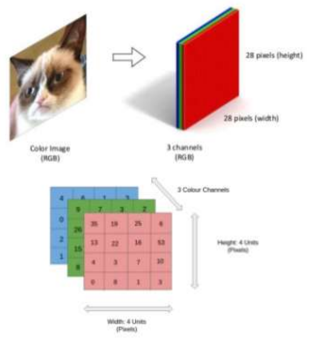
$$\begin{bmatrix} 100 & 101 & 131 \\ 150 & 10 & 111 \\ 10 & 200 & 100 \end{bmatrix}, \begin{bmatrix} 100 & 112 & 20 \\ 210 & 120 & 120 \\ 260 & 20 & 20 \end{bmatrix}, \begin{bmatrix} 50 & 3 & 80 \\ 130 & 130 & 130 \\ 30 & 30 & 3 \end{bmatrix}$$

R G B

- Mỗi phần tử của ma trận có giá trị từ 0 đến 255.
- Khái niệm tensor: Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột. Khi dữ liệu dạng 2 chiều, sẽ có dạng ma trận, kích thước là số hàng * số cột. Khi dữ liệu nhiều hơn 2 chiều thì sẽ được gọi là tensor, ví dụ như dữ liệu có 3 chiều. Xếp k ma trận kích thước m*n lên nhau sẽ được tensor kích thước m*n*k.

Image Processing

color image is 3rd-order tensor



- Ví dụ ảnh màu kích thước 28*28, biểu diễn dưới dạng tensor 28*28*3
- Khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ.

$$\begin{bmatrix} 0 & 215 & \dots & 250 \\ 12 & 156 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 244 & 255 & \dots & 12 \end{bmatrix}$$

- Khi chuyển từ ảnh màu sang ảnh xám có thể dùng công thức:

$x = r * 0.299 + g * 0.587 + b * 0.114.$

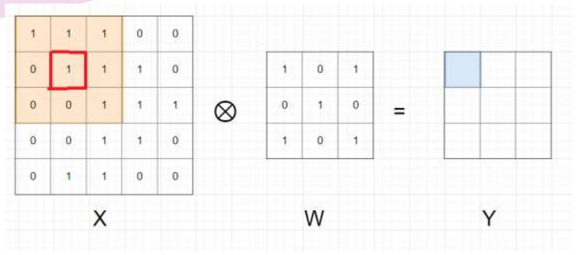
Image Processing

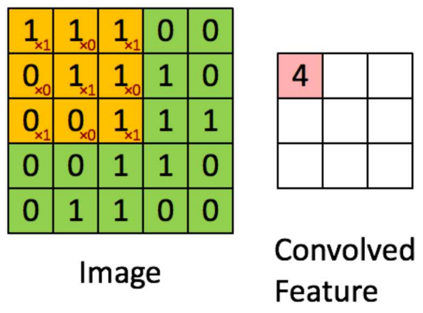
- Phép tính Convolution: Định nghĩa kernel là một ma trận vuông kích thước k*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9,... Ví dụ kernel kích thước 3*3

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- Kí hiệu phép tính convolution ($\otimes$), kí hiệu $Y = X \otimes W$
- Với mỗi phần tử $x$ij trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử x_{ij} làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính element-wise của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y

Image Processing



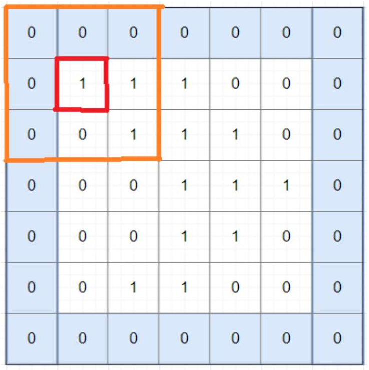


- Ví dụ khi tính tại x22 (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có x22 làm trung tâm có màu nền da cam như trong hình. Sau đó tính y11 = sum(A⊗W) = x11 ∗w11 +x12 ∗w12 +x13 ∗ w13 +x21 ∗ w21 +x22 ∗ w22 +x23 ∗ w23 +x31 ∗ w31 +x32 ∗ w32 +x33 ∗ w33 = 4.

Image Processing

Padding: Mỗi lần thực hiện phép tính convolution xong, kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên nếu muốn ma trận Y thu được có kích thước bằng ma trận X => Tìm cách giải quyết cho các phần tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.

Phép tính này gọi là convolution với padding=1. Padding=k nghĩa là thêm k vector 0 vào mỗi phía của ma trận



0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

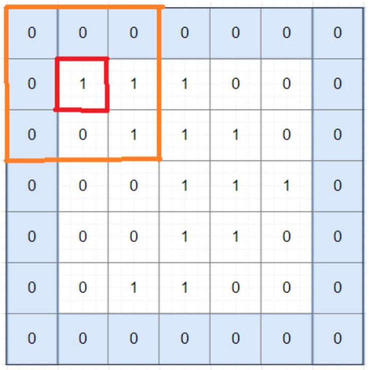
Image Processing

Stride: Thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

Tuy nhiên nếu stride=k ($k > 1$) thì chỉ thực hiện phép tính convolution trên các phần tử $x_{1+i*k; 1+j*k}$.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước m*n với kernel kích thước $k*k$, stride = s, padding = p ra ma trận Y kích thước

$$(\frac{m-k+2p}{s}+1)*(\frac{n-k+2p}{s}+1)$$



0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

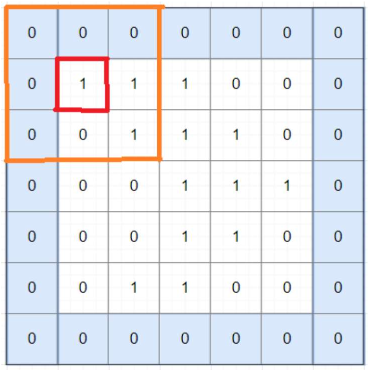
Image Processing

Stride: Thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

Tuy nhiên nếu stride=k ($k > 1$) thì chỉ thực hiện phép tính convolution trên các phần tử $x_{1+i*k;1+j*k}$.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước m*n với kernel kích thước $k*k$, stride = s, padding = p ra ma trận Y kích thước

$$\left(\frac{m-k+2p}{s}+1\right)*\left(\frac{n-k+2p}{s}+1\right)$$

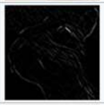
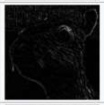
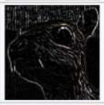
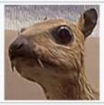
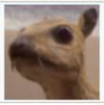


0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

25

Image Processing

- Ý nghĩa của phép tính convolution: Mục đích của phép tính convolution trên ảnh là làm mờ, làm nét ảnh; xác định các đường;... Với mỗi kernel khác nhau phép tính convolution sẽ có ý nghĩa khác nhau

Operation	Kernel ω	Image result g(x
Identity	$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$	
Edge detection	$\begin{bmatrix}1&0&-1\\0&0&0\\-1&0&1\end{bmatrix}$	
	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	
	$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$	
Sharpen	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	

26

Convolutional Neural Network

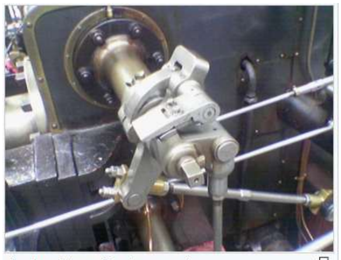
- CNN (Mạng nơ-ron lấy chập)

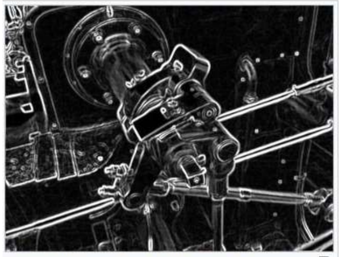
Vấn đề đặt ra với mạng nơ-ron kết nối đầy đủ: Ví dụ ảnh màu kích thước 64x64 sẽ được biểu diễn dưới dạng tensor 64x64x3 = 12 288 pixel. Vậy input layer (lớp vào) sẽ cần có 12288 nút.

Giả thiết số nút của lớp ẩn thứ nhất là 1000. Do có kết nối đầy đủ giữa lớp vào và lớp ẩn thứ nhất này nên số lượng trọng số W giữa lớp vào và lớp ẩn thứ nhất sẽ là 12 288 x 1000 = 12 288 000, số lượng độ lệch bias là 1000, tổng số tham số sẽ là 12 289 000. Đây mới chỉ là số tham số giữa lớp vào và lớp ẩn đầu tiên. Nếu kích thước ảnh đầu vào tăng và số lớp ẩn tăng thì tổng số tham số còn tăng nhanh nữa. Cần có giải pháp giảm số lượng tham số song đặc trưng bức ảnh vẫn đảm bảo cho nhận dạng.

Nhận xét:

- Các pixel kề nhau trong ảnh thường có độ tương quan lớn so với các pixel khác ở xa chúng. Để làm rõ đường viền có thể dùng các kernel (bộ lọc) kiểu sobel trên mỗi vùng có kích thước 3x3. Để làm nét ảnh có thể dung kernel kiểu sharpen.
- Trong phép lấy chập, chỉ 1 kernel dùng chung cho toàn bức ảnh, như vậy các pixel đã dùng chung hay chia sẻ hệ số với nhau.

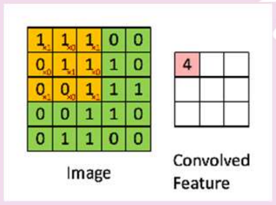

A color picture of a steam engine

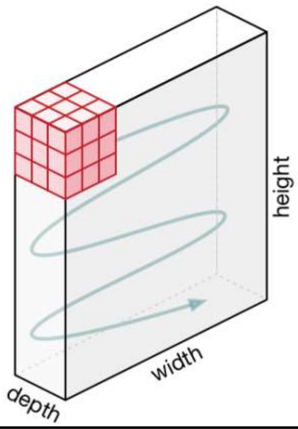

The Sobel operator applied to that image

27

27

Convolutional Neural Network



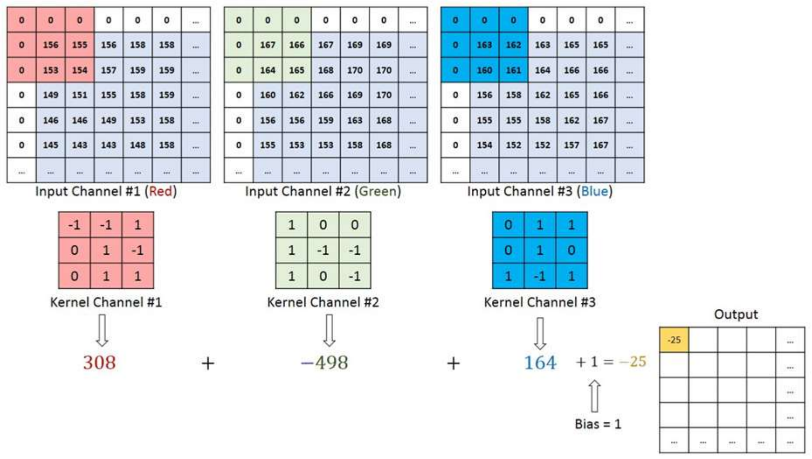


Ảnh màu có 3 kênh R, G, B nên khi biểu diễn ảnh dưới dạng Tensor 3 chiều cũng sẽ định nghĩa kernel là tensor 3 chiều có kích thước k x k x 3.
Độ sâu (depth) của kernel lúc này bằng 3.

28

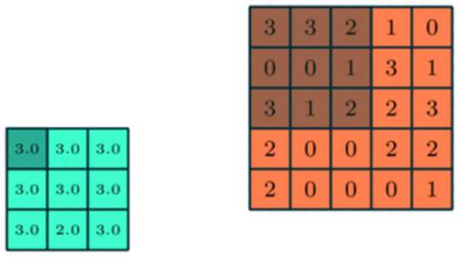
28

Convolution for color image



Pooling Layer

- Max Pooling với size = (3,3), stride = 1, padding = 0



One-hot Encode

- **Why One-Hot Encode Data in Machine Learning?**

What is Categorical Data?

Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Categorical variables are often called nominal.Some examples include:

A “*pet*” variable with the values: “*dog*” and “*cat*“.

A “*color*” variable with the values: “*red*“, “*green*” and “*blue*“.

A “*place*” variable with the values: “*first*”, “*second*” and “*third*“.

Each value represents a different category.

Some categories may have a natural relationship to each other, such as a natural ordering.

The “*place*” variable above does have a natural ordering of values. This type of categorical variable is called an ordinal variable.

One-hot Encode

- **What is the Problem with Categorical Data ?**

What is Categorical Data?

Some algorithms can work with categorical data directly. For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation). Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

One-hot Encode

- **How to Convert Categorical Data to Numerical Data?**

This involves two steps:

Integer Encoding

One-Hot Encoding

1. Integer Encoding

As a first step, each unique category value is assigned an integer value.For example, “red” is 1, “green” is 2, and “blue” is 3.This is called a label encoding or an integer encoding and is easily reversible. For some variables, this may be enough.

The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.

For example, ordinal variables like the “place” example above would be a good example where a label encoding would be sufficient.

One-hot Encode

- **How to Convert Categorical Data to Numerical Data?**

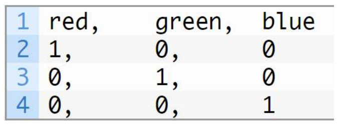
2. One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors.

For example:

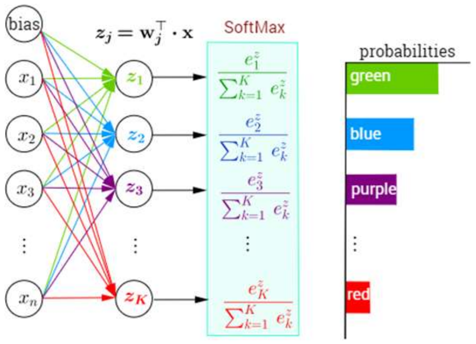


```
red,    green,  blue
1,      0,      0
0,      1,      0
0,      0,      1
```

In summary: you discovered why categorical data often must be encoded when working with machine learning algorithms. Specifically:

- That categorical data is defined as variables with a finite set of label values.
- That most machine learning algorithms require numerical input and output variables.
- That an integer and one hot encoding is used to convert categorical data to integer data.

Softmax



bias
x_1
x_2
x_3
x_n
$z_j = \mathbf{w}_j^\top \cdot \mathbf{x}$
z_1
z_2
z_3
z_K
SoftMax
$\frac{e_1^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_2^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_3^z}{\sum_{k=1}^K e_k^z}$
$\frac{e_K^z}{\sum_{k=1}^K e_k^z}$
probabilities
green
blue
purple
red

What is the Softmax Function?
The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always

Many multi-layer neural networks end in a penultimate layer which outputs real-valued scores that are not conveniently scaled and which may be difficult to work with. Here the softmax is very useful because it converts the scores to a normalized probability distribution, which can be displayed to a user or used as input to other systems. For this reason it is usual to append a softmax function as the final layer of the neural network.

35

35

Softmax

- Softmax Formula

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- Mathematical definition of the softmax function

where all the z_i values are the elements of the input vector and can take any real value. The term on the bottom of the formula is the normalization term which ensures that all the output values of the function will sum to 1, thus constituting a valid probability distribution.

36

36

Softmax

- Softmax Formula Symbols Explained

$\vec{z}$	The input vector to the softmax function, made up of (z0, ... zK)
z_i	All the zi values are the elements of the input vector to the softmax function, and they can take any real value, positive, zero or negative. For example a neural network could have output a vector such as (-0.62, 8.12, 2.53), which is not a valid probability distribution, hence why the softmax would be necessary.
e^{z_i}	The standard exponential function is applied to each element of the input vector. This gives a positive value above 0, which will be very small if the input was negative, and very large if the input was large. However, it is still not fixed in the range (0, 1) which is what is required of a probability.
$\sum_{j=1}^{K} e^{z_j}$	The term on the bottom of the formula is the normalization term. It ensures that all the output values of the function will sum to 1 and each be in the range (0, 1), thus constituting a valid probability distribution.
K	The number of classes in the multi-class classifier.

37

Softmax

- Calculating the Softmax: Imagine we have an array of three real values. These values could typically be the output of a machine learning model such as a neural network. We want to convert the values into a probability distribution.

$$\begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix}$$

- First we can calculate the exponential of each element of the input array. This is the term in the top half of the softmax equation.

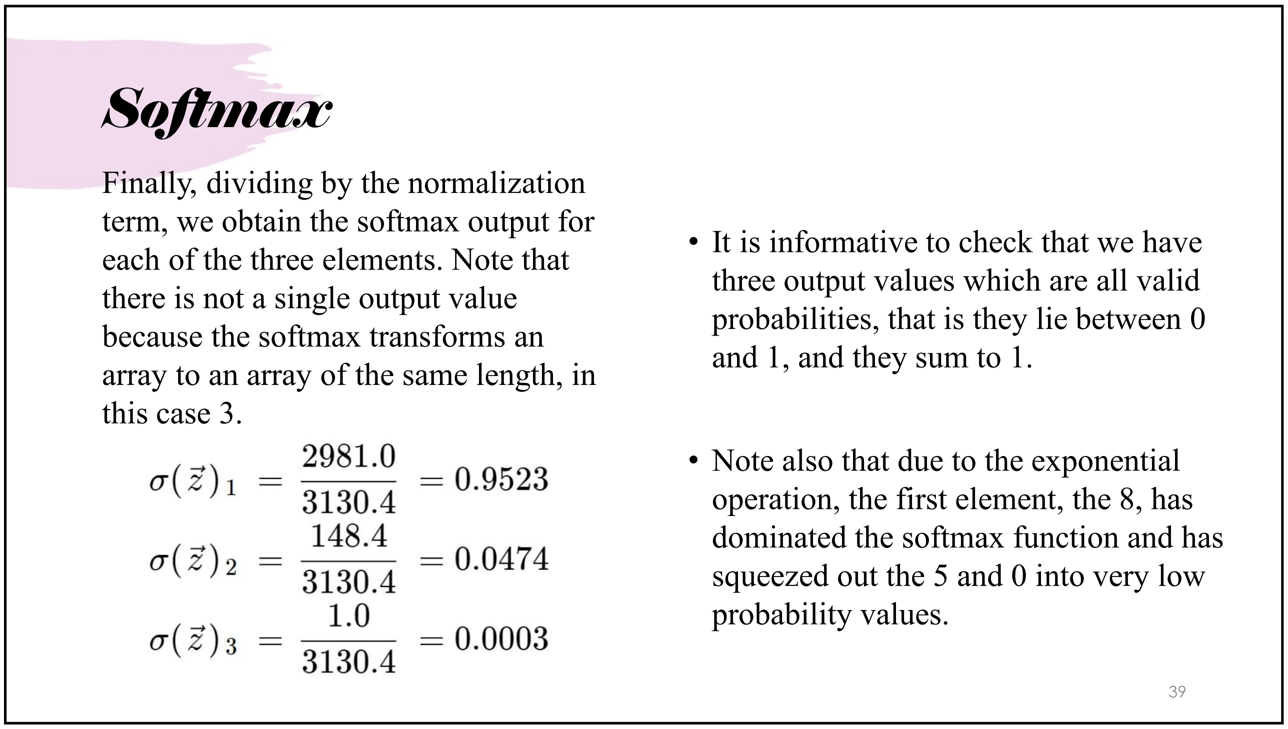
$$e^{z_1} = e^{8} = 2981.0$$
$$e^{z_2} = e^{5} = 148.4$$
$$e^{z_3} = e^{0} = 1.0$$

- These values do not look like probabilities yet. Note that in the input elements, although 8 is only a little larger than 5, 2981 is much larger than 148 due to the effect of the exponential. We can obtain the normalization term, the bottom half of the softmax equation, by summing all three exponential terms:

$$\sum_{j=1}^{K} e^{z_j} = e^{z_1} + e^{z_2} + e^{z_3} = 2981.0 + 148.4 + 1.0 = 3130.4$$

38

38



Softmax

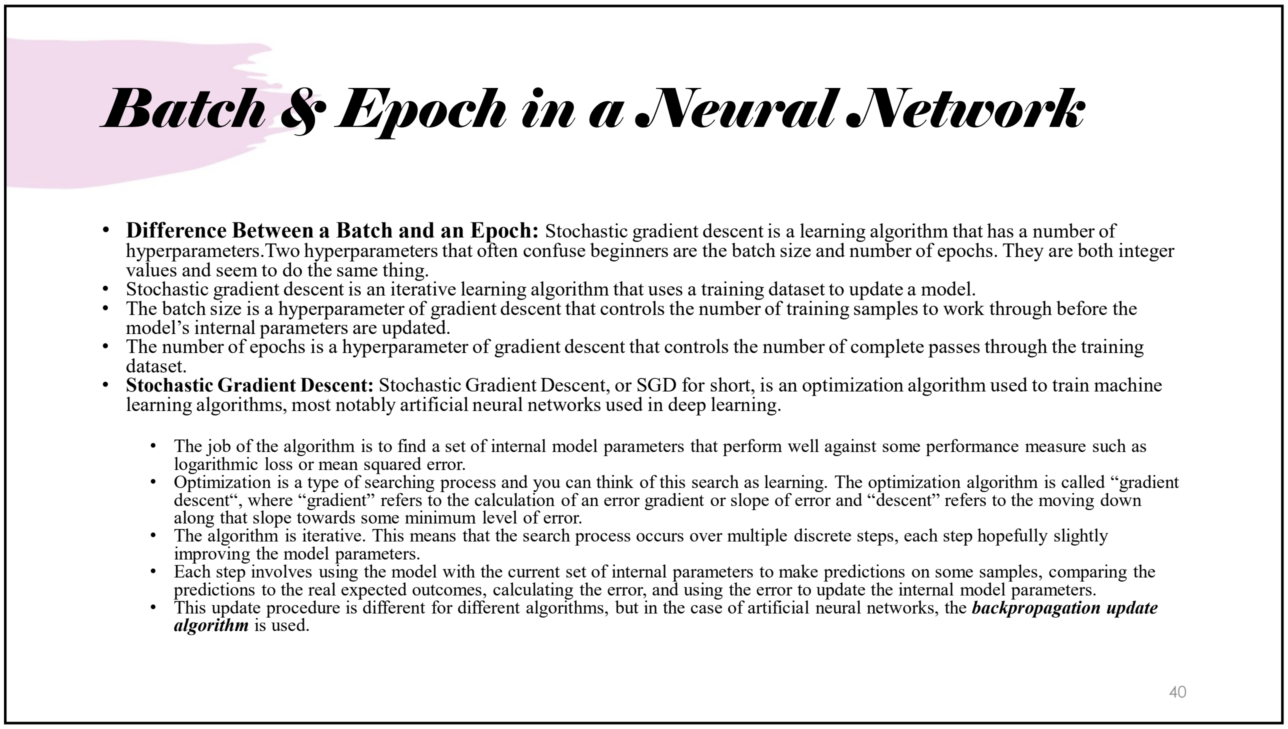
Finally, dividing by the normalization term, we obtain the softmax output for each of the three elements. Note that there is not a single output value because the softmax transforms an array to an array of the same length, in this case 3.

$$\sigma(\vec{z})_1 = \frac{2981.0}{3130.4} = 0.9523$$
$$\sigma(\vec{z})_2 = \frac{148.4}{3130.4} = 0.0474$$
$$\sigma(\vec{z})_3 = \frac{1.0}{3130.4} = 0.0003$$

- It is informative to check that we have three output values which are all valid probabilities, that is they lie between 0 and 1, and they sum to 1.
- Note also that due to the exponential operation, the first element, the 8, has dominated the softmax function and has squeezed out the 5 and 0 into very low probability values.

39

39



Batch & Epoch in a Neural Network

- **Difference Between a Batch and an Epoch:** Stochastic gradient descent is a learning algorithm that has a number of hyperparameters.Two hyperparameters that often confuse beginners are the batch size and number of epochs. They are both integer values and seem to do the same thing.
- Stochastic gradient descent is an iterative learning algorithm that uses a training dataset to update a model.
- The batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.
- The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.
- **Stochastic Gradient Descent:** Stochastic Gradient Descent, or SGD for short, is an optimization algorithm used to train machine learning algorithms, most notably artificial neural networks used in deep learning.
  - The job of the algorithm is to find a set of internal model parameters that perform well against some performance measure such as logarithmic loss or mean squared error.
  - Optimization is a type of searching process and you can think of this search as learning. The optimization algorithm is called "gradient descent", where "gradient" refers to the calculation of an error gradient or slope of error and "descent" refers to the moving down along that slope towards some minimum level of error.
  - The algorithm is iterative. This means that the search process occurs over multiple discrete steps, each step hopefully slightly improving the model parameters.
  - Each step involves using the model with the current set of internal parameters to make predictions on some samples, comparing the predictions to the real expected outcomes, calculating the error, and using the error to update the internal model parameters.
  - This update procedure is different for different algorithms, but in the case of artificial neural networks, the ***backpropagation update algorithm*** is used.

40

40

Batch & Epoch in a Neural Network

- **What Is a Sample?** A sample is a single row of data. It contains inputs that are fed into the algorithm and an output that is used to compare to the prediction and calculate an error. A training dataset is comprised of many rows of data, e.g. many samples. A sample may also be called an instance, an observation, an input vector, or a feature vector.
- What Is a Batch? The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

A training dataset can be divided into one or more batches. When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

- **Batch Gradient Descent**. Batch Size = Size of Training Set
- **Stochastic Gradient Descent**. Batch Size = 1
- **Mini-Batch Gradient Descent**. 1 < Batch Size < Size of Training Set

In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples.

41

41

Batch & Epoch in a Neural Network

- **What if the dataset does not divide evenly by the batch size?**
- This can and does happen often when training a model. It simply means that the final batch has fewer samples than the other batches. Alternately, you can remove some samples from the dataset or change the batch size such that the number of samples in the dataset does divide evenly by the batch size.
- **What Is an Epoch?** The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm. You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified "batch size" number of samples.
- The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.
- It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset.

42

42

Batch & Epoch in a Neural Network

- **What Is the Difference Between Batch and Epoch?**

The batch size is a number of samples processed before the model is updated. The number of epochs is the number of complete passes through the training dataset. The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset. The number of epochs can be set to an integer value between one and infinity. You can run the algorithm for as long as you like and even stop it using other criteria besides a fixed number of epochs, such as a change (or lack of change) in model error over time. They are both integer values and they are both hyperparameters for the learning algorithm, e.g. parameters for the learning process, not internal model parameters found by the learning process. You must specify the batch size and number of epochs for a learning algorithm. There are no magic rules for how to configure these parameters. You must try different values and see what works best for your problem.

- **Worked Example**

Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.
This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples.
This also means that one epoch will involve 40 batches or 40 updates to the model.
With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

43

43

Normalization

- **Normalization** is the process of converting an actual range of values which a numerical feature can take, into a standard range of values, typically in the interval [-1, 1] or [0, 1]. For example, suppose the natural range of a particular feature is 350 to 1450. By subtracting350 from every value of the feature, and dividing the result by 1100, one can normalize those values into the range [0, 1].
  More generally, the normalization formula looks like this:

$$\bar{x}^{(j)} = \frac{x^{(j)} - min^{(j)}}{max^{(j)} - min^{(j)}},$$

where *min*(*j*) and *max*(*j*) are, respectively, the minimum and the maximum value of the feature *j* in the dataset.
Why do we normalize? Normalizing the data is not a strict requirement. However, in practice, it can lead to an increased speed of learning. Remember the gradient descent example. Imagine you have a two-dimensional feature vector. When you update the parameters of *w*(1) and *w*(2), you use partial derivatives of the average squared error with respect to *w*(1) and *w*(2). If *x*(1) is in the range [0, 1000] and *x*(2) the range [0, 0.0001], then the derivative with respect to a larger feature will dominate the update.
Additionally, it's useful to ensure that our inputs are roughly in the same relatively small range to avoid problems which computers have when working with very small or very big
numbers (known as numerical overflow)

44

44

Standardization

- **Standardization** (or **z-score normalization**) is the procedure during which the feature values are rescaled so that they have the properties of a *standard normal distribution* with $\mu = 0$ and $\sigma = 1$, where μ is the mean (the average value of the feature, averaged over all examples in the dataset) and σ is the standard deviation from the mean.
  Standard scores (or z-scores) of features are calculated as follows:

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}}$$

- You may ask when you should use normalization and when standardization. There's no definitive answer to this question. Usually, if your dataset is not too big and you have time,you can try both and see which one performs better for your task.
- unsupervised learning algorithms, in practice, more often benefit from standardization than from normalization;standardization is also preferred for a feature if the values this feature takes are distributed close to a normal distribution (so-called bell curve);
- again, standardization is preferred for a feature if it can sometimes have extremely high or low values (outliers); this is because normalization will "squeeze" the normal valuesinto a very small range;
- in all other cases, normalization is preferable.
  Modern implementations of the learning algorithms, which you can find in popular libraries, are robust to features lying in di!erent ranges. Feature rescaling is usually beneficial to most learning algorithms, but in many cases, the model will still be good when trained from the original features.