Convolution Neural Networks

Jonathan Guymont

Fin ML

Convolution Neural Network

Task where CNN were employed with success

- NLP
  - Speech recognition
  - Document classification
  - Sentiment analysis
  - etc.
- Time series
  - Human activity recognition
  - MRI classification (cancer detection)
  - etc.
- Biology
  - Protein-protein interaction (predication)
  - Drug design
  - etc.
- etc.

Deep Learning Development and Vision Task

- CNNs were developped specifically for solving vision task, although they are now used for solving a larger number of problem
- Large image datasets are very common and benchmarked
- Distribution are often constant which makes the construction of large dataset easier then with dynamical system like financial market where the distribution often change
- There is a lot of competition on vision tasks compared to other machine learning tasks.
  - ImageNet[1]
  - COCO [2]
  - And a lot lot more...

[1] http://www.image-net.org/challenges/LSVRC/
[2] http://cocodataset.org/home

Some Notation

- Training set: X^{train}
- An example: $x^{(i)}$
- A target $y^{(i)}$
- A prediction $\hat{y}^{(i)}$
- A one-hot target: $\boldsymbol{y}^{(i)}$
- Number of training examples: N
- Size of a batch: m
- Loss function: L
- Loss: $L(\hat{y}^{(i)}, y^{(i)})$
- Number of training iterations (epochs): T

What is an image?

$$\begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \mathbf{255} & \mathbf{211} & 111 & \cdots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 1 & \mathbf{246} & \mathbf{244} & 4 & \cdots & 0 \\ 0 & \cdots & 12 & \mathbf{233} & \mathbf{213} & 34 & \cdots & 0 \\ 0 & \cdots & 22 & \mathbf{252} & \mathbf{234} & 2 & \cdots & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a0 & \cdots & 4 & \mathbf{233} & \mathbf{222} & 33 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Figure: Conversion of a greyscale image to an matrix

Black and white image

- Black and white image are said to have one channel which means they are represented by one $H \times W$ dimension matrix. Each element of the matrix represents the darkness of the picture.

Colored Image

- Colored image are said to have 3 channels which means they are represented by a $3 \times H \times W$ tensor. Each element of the $H \times W$ matrix represents the intensity in red, green, and blue, respectively.

Converting an image to an array

Python code

```
from PIL import Image
def image_to_array(image_path, mode):
    """transform an image file (eg a .PNG) into an array
    Arguments:
        image_path: (String) path to an image (jpg, png, ...)
        mode: (String) "L" for greyscale or "RGB" for color
    """
    with Image.open(image_path) as img:
        image = img.convert(mode)
    array_image = np.asarray(image, np.float)
    return array_image
```

Preprocessing

- Pixels value are usually preprocess either by dividing by 255 or by removing the mean and dividing by the variance.
- Usually, having features with similar distributions improve gradient stability.

Preprocessing

First, it is important to note that better backpropagation is obtained if the gradients have the same distribution throughout the network. Stability of the gradient's distribution depends mainly on two things: good weight initialization and inputs with constant variance (see Glorot and Bengio, 2010[3]).

As for the justification of why the inputs must be centered in 0, note that the derivative of the loss function is proportional to the inputs. When doing minibatch training, the variance of your gradient is proportional to the variance of the features in the minibatch. If the means of the inputs are not constant, you risk to overcorrect weights when the minibatch has large feature and to undercorrect otherwise. One easy way of insuring a constant mean is to center all the features.

[3] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

High level CNN Classifier

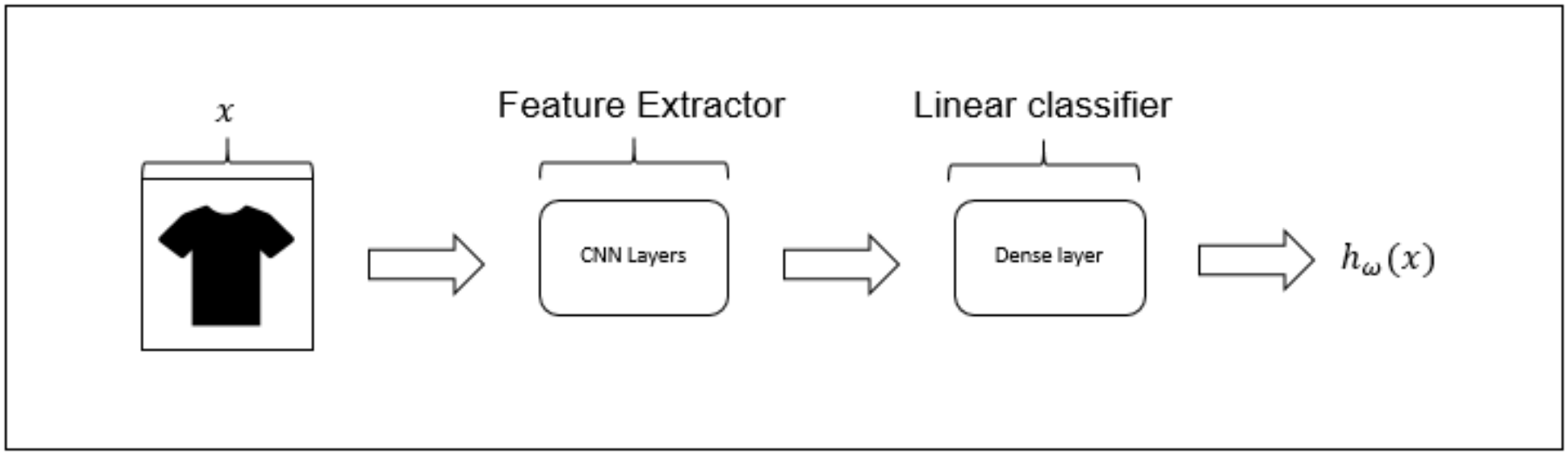


Figure: Typical structure of a CNN

Convolution Operation

The position (i, j) of a *discrete convolution* between an input x and a filter k

$$s(i,j) = (x \star k)(i,j) = \sum_m \sum_n x(m,n)k(i-m, j-n)$$

Machine learning libraries implement *cross-correlation* instead:

$$s(i,j) = (x \star k)(i,j) = \sum_m \sum_n x(i+m, j+n)k(m,n)$$

Pytorch code

```
out = F.conv2d(x, filters, stride=1)
```

Feature Extraction: Filters

The parameters of a CNN consist of filters. Filters are 2 dimensional vector that are initialized randomly and optimized to detect useful features in the inputs.

When we configure a CNN, we need to decide the size of each filter at each layer.

When computing the output of a convolution layer, we compute the cross-correlation between the filters and the input.

Feature Extraction

Consider the following black and white image

$$\mathbf{x} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Feature Extraction: Filters

Consider the following filter:

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

the cross-correlation of $\mathbf{x}$ with W would be a 7×6 matrix

$$\mathbf{x} \star W = \begin{pmatrix} 0 & 0 & 100 & 100 & 100 & 100 \\ 0 & 100 & 100 & 100 & 100 & 100 \\ 0 & 200 & 100 & 100 & 0 & 100 \\ 0 & 300 & 100 & 0 & 0 & 0 \\ 0 & 400 & 0 & 0 & 0 & 0 \\ 0 & 400 & 100 & 100 & 100 & 100 \\ 0 & 300 & 100 & 100 & 100 & 100 \end{pmatrix}$$

Feature Extraction: Filters

The first element of the output is the cross-correlation of the first 4×4 sub-matrix in $\mathbf{x}$ with W

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Feature Extraction: Filters

The first element of the output is the cross-correlation of the first 4×4 sub-matrix in $\mathbf{x}$ with W

$$
\begin{aligned}
\text{out}[1,1] =& \mathbf{x}_{:4,:4} \star W \\
=& \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 \\ 0 & 0 & 100 & 0 \end{pmatrix} \star \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \\
=& \sum_{i=1}^{4} \sum_{j=1}^{4} \mathbf{x}_{:4,:4}[i,j] \cdot W[i,j] \\
=& 0
\end{aligned}
$$

Fin·ML

Feature Extraction: Filters

The second element of the output is the cross-correlation of W with the second 4×4 sub-matrix in $\mathbf{x}$ when moving of one column to the left

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 100 & 100 & 100 & 100 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Feature Extraction: Filters

The second element of the output is the cross-correlation of the second 4×4 sub-matrix in $\mathbf{x}$ when moving of one column to the left with W

$$
\begin{aligned}
\text{out}[1,2] =& \mathbf{x}_{:4,:4} \star W \\
=& \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 \\ 0 & 0 & 100 & 0 \\ 0 & 100 & 0 & 0 \end{pmatrix} \star \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \\
=& \sum_{i=1}^{4} \sum_{j=1}^{4} \mathbf{x}_{:4,1:5}[i,j] \cdot W[i,j] \\
=& 0
\end{aligned}
$$

Fin·ML

Feature Extraction: Filters

Example of filters that could be learned

Diagonal edge detector:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Horizontal edge detector:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Feature Extraction: Filters

Why are we talking about edge detector

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

$$\mathbf{x} \star W = \begin{pmatrix} 0 & 0 & 100 & 100 & 100 & 100 \\ 0 & \mathbf{100} & 100 & 100 & 100 & 100 \\ 0 & \mathbf{200} & 100 & 100 & 0 & 100 \\ 0 & \mathbf{300} & 100 & 0 & 0 & 0 \\ 0 & \mathbf{400} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{400} & 100 & 100 & 100 & 100 \\ 0 & \mathbf{300} & 100 & 100 & 100 & 100 \end{pmatrix}$$

Hyperparameters of a Convolution Layer

Stride

A stride of 1 means that we always move of 1 element after during cross-correlation

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Fin·ML

Hyperparameters of a Convolution Layer

Stride

A stride of 2 means that we always move of 2 elements during cross-correlation

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 100 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Main hyperparameters of a Convolution Layer

Padding

$$\begin{pmatrix}
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 100 & \cdots & 100 & 0 & 0 & 0 \\
0 & 0 & 100 & 0 & \cdots & 0 & 100 & 0 & 0 \\
0 & 0 & 100 & 0 & \cdots & 0 & 100 & 0 & 0 \\
0 & 0 & 100 & 0 & \cdots & 0 & 100 & 0 & 0 \\
0 & 0 & 100 & 0 & \cdots & 0 & 100 & 0 & 0 \\
0 & 0 & 100 & 100 & \cdots & 100 & 100 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0
\end{pmatrix}$$

Main hyperparameters of a Convolution Layer

Kernel Size

Size of the weights of the convolution layer, i.e. size of the filters.

Main hyperparameters of a Convolution Layer

Number of output Channels

It is common to increase the number of channels by applying different filter to the same input.

High level CNN Classifier

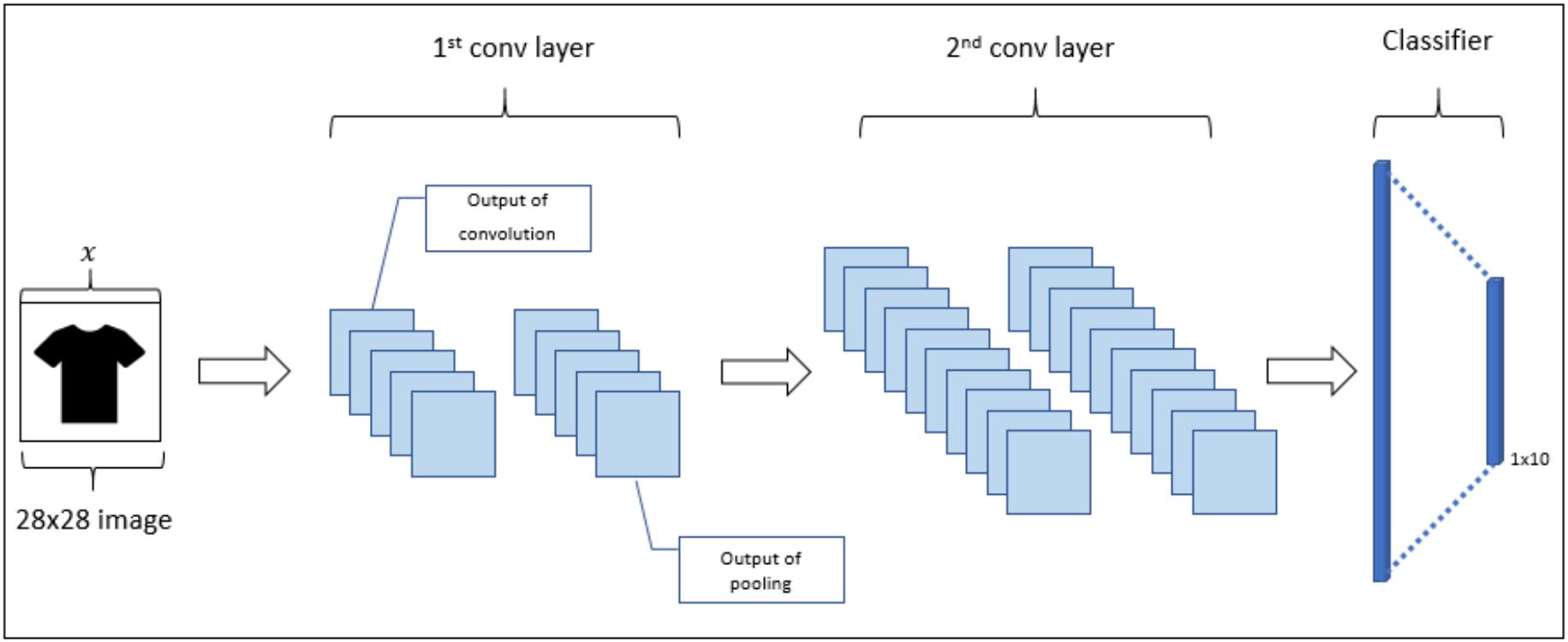


Figure: Example of CNN architecture

Increasing the number of channels

from one channel to N channels

Let x be a $H \times W$ image. To increase the number of channel from one to N you need N different filters. Each cross-correlation between the input and a filter will become a channel.

Increasing the number of channels

from N channel to M channels

Let x be a $N \times H \times W$ input. To increase the number of channel from N to M you need M set of N different filters. The sum of the cross-correlation between all the input channel and a set of N filters will become a channel.

Convolution Layer

Let $\mathbf{x}$ be a $H \times W$ image. The output value of a layer with input size (N, C_{in}, H, W) and output size $(N, C_{out}, H_{out}, W_{out})$ can be described has

$$\begin{aligned} \text{out}[i] &= \text{Conv}_k(\mathbf{x}) && (1) \\ &= \text{bias}[i] + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(i,k) \star \text{input}[k] && (2) \end{aligned}$$

High level CNN Classifier

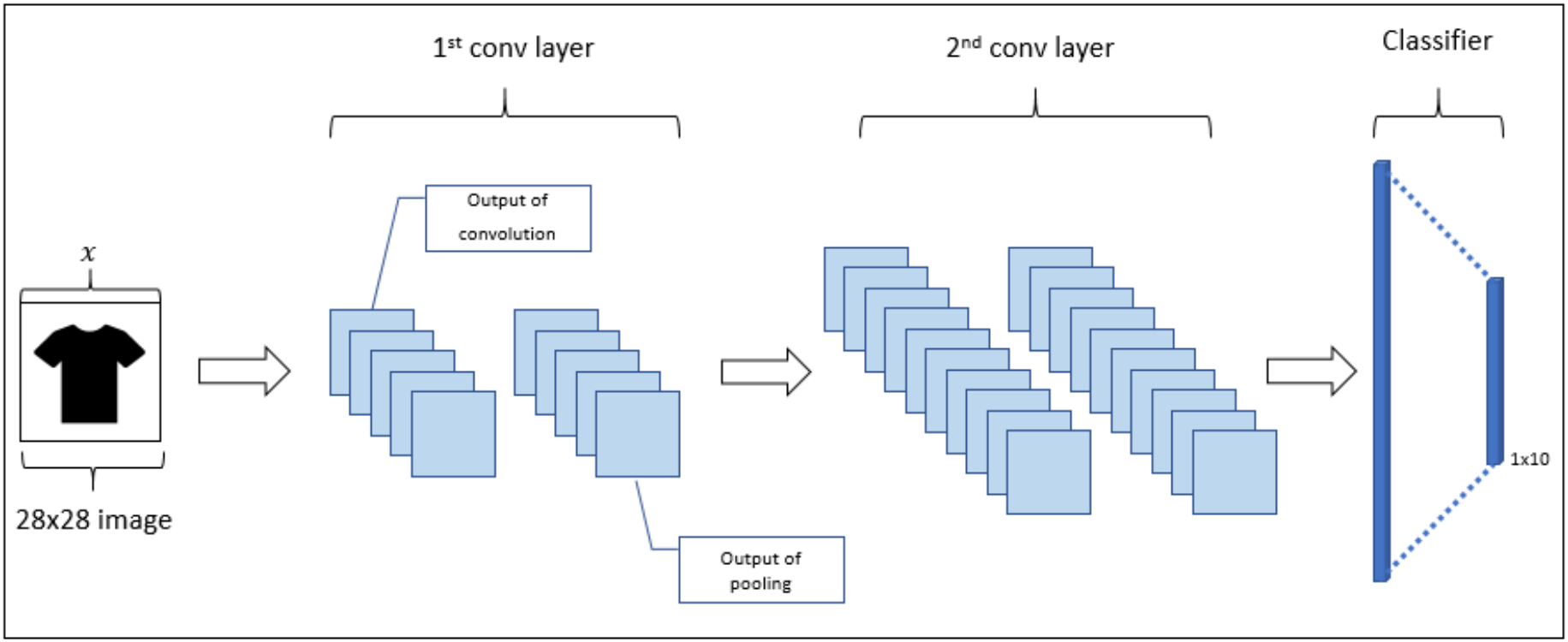


Figure: Example of CNN architecture

Simple CNN Classifier

$$
\begin{aligned}
\mathbf{out} &= \mathrm{relu}(\mathrm{Conv}(\mathbf{x})) \\
\mathbf{h} &= \mathrm{Cat}(\mathrm{out}) \\
\mathbf{y} &= \mathrm{softmax}(W\mathbf{h} + \mathbf{b})
\end{aligned}
$$

High level CNN Classifier

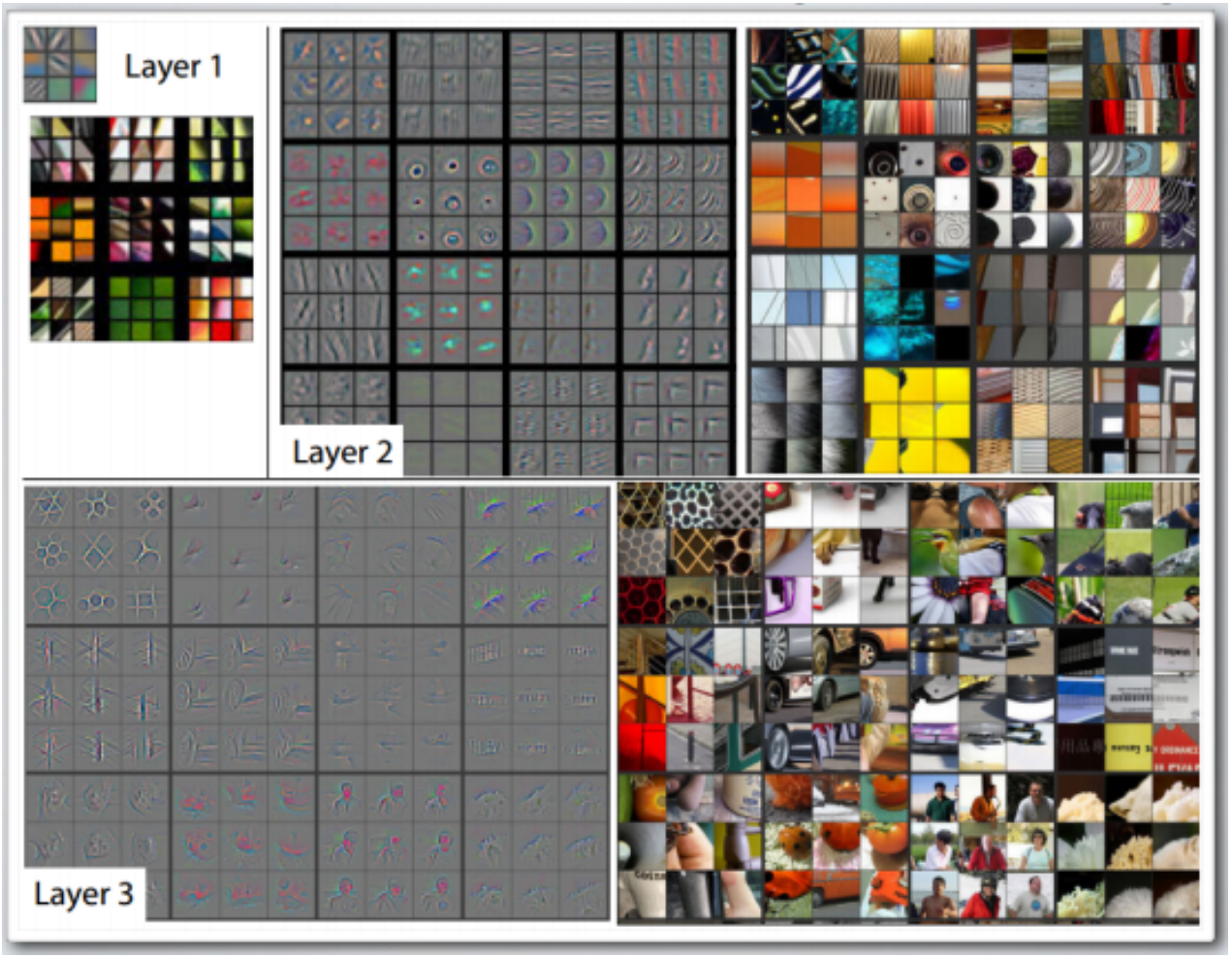


Figure: Visualizong CNN (Image from Zeiler and Fergus, 2013)

High level CNN Classifier

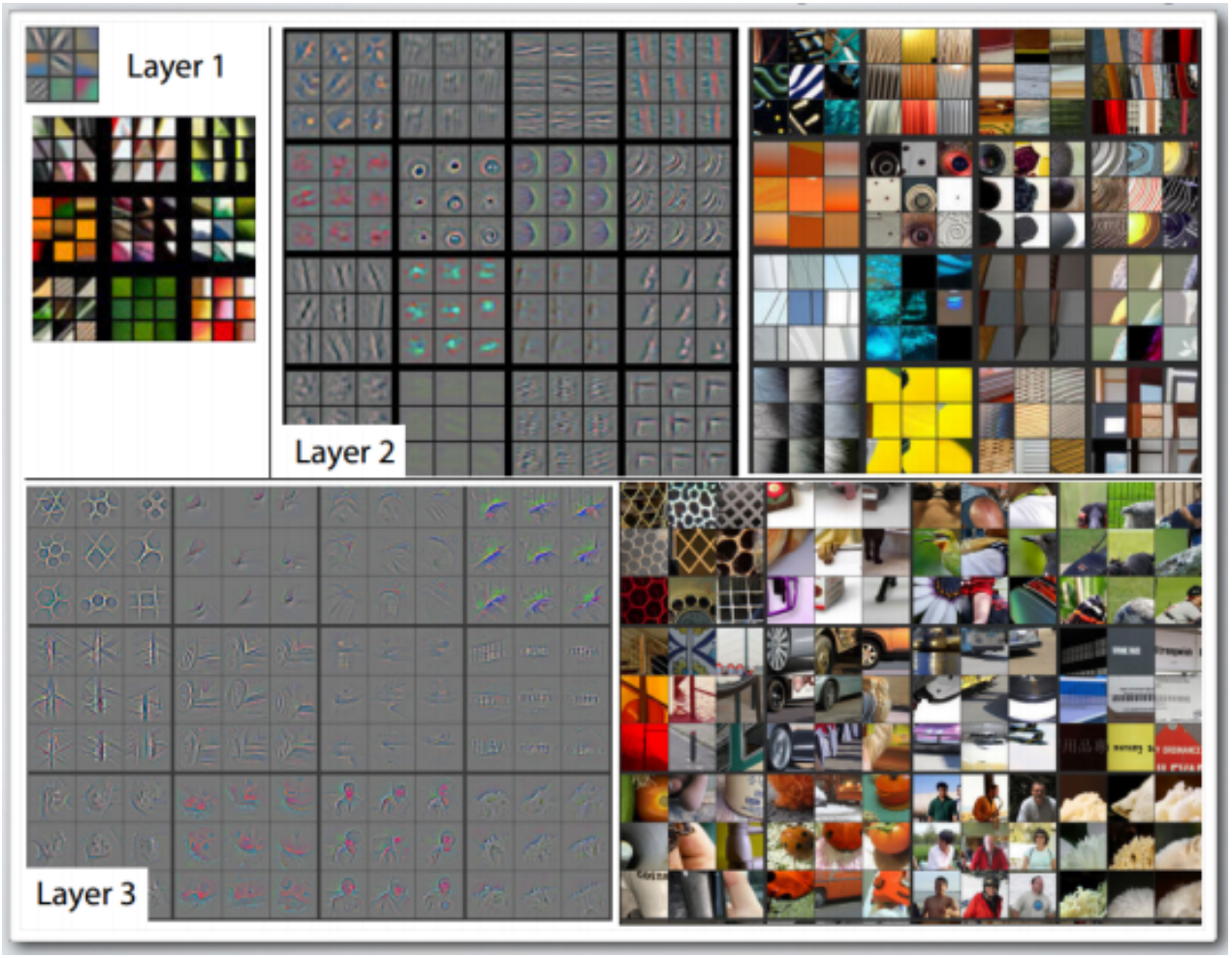


Figure: Visualizong CNN (Image from Zeiler and Fergus, 2013)

High level CNN Classifier

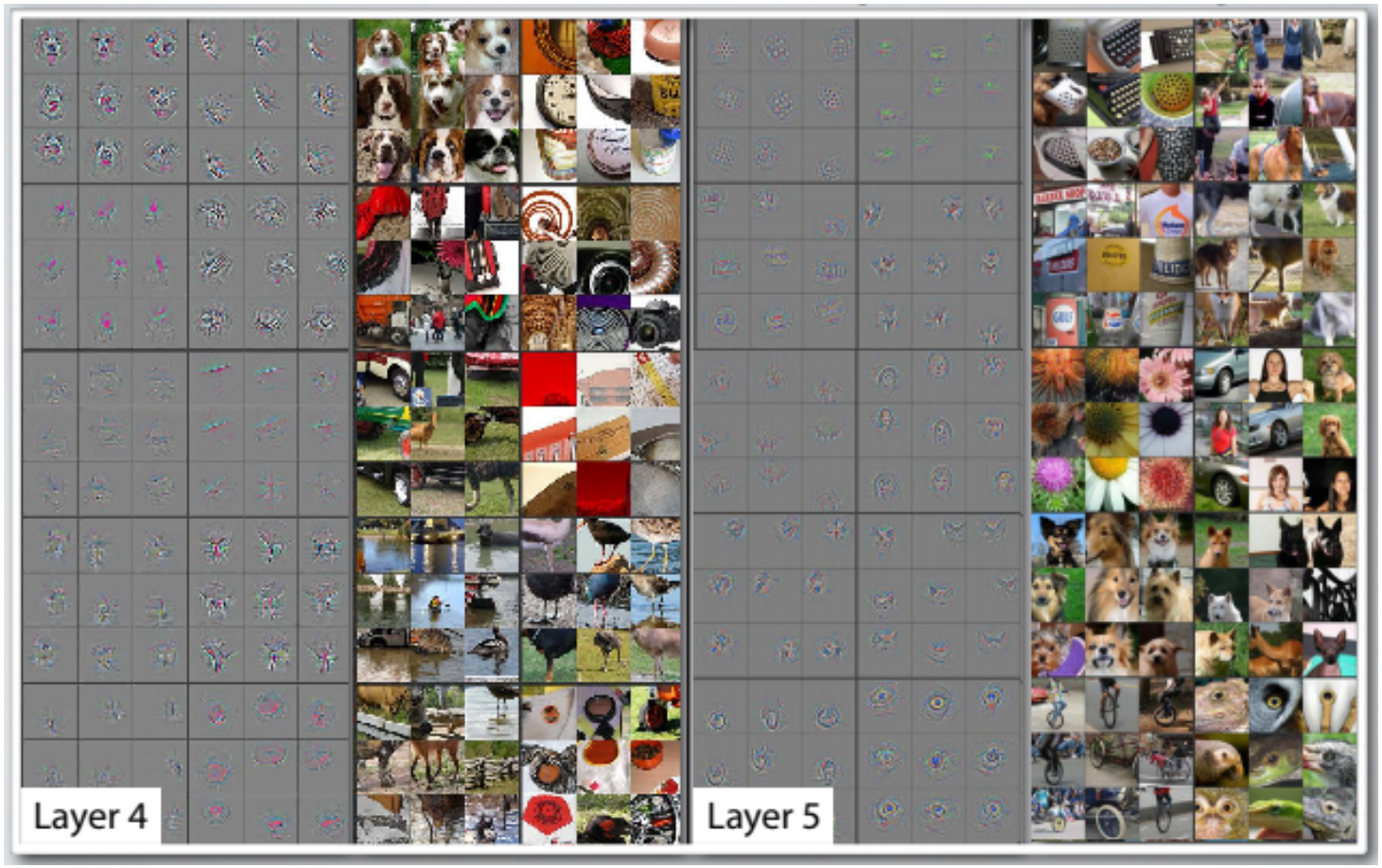


Figure: Visualizong CNN (Image from Zeiler and Fergus, 2013)

Pooling

Different pooling types

- Max pooling (most popular)
- Average pooling
- etc.

Example: Apply a 2×2 max pooling to a 4×4 CNN activation.

Pytorch code

```
out = F.max_pool2d(x, kernel_size=(2,2))
```

CNN Classifier

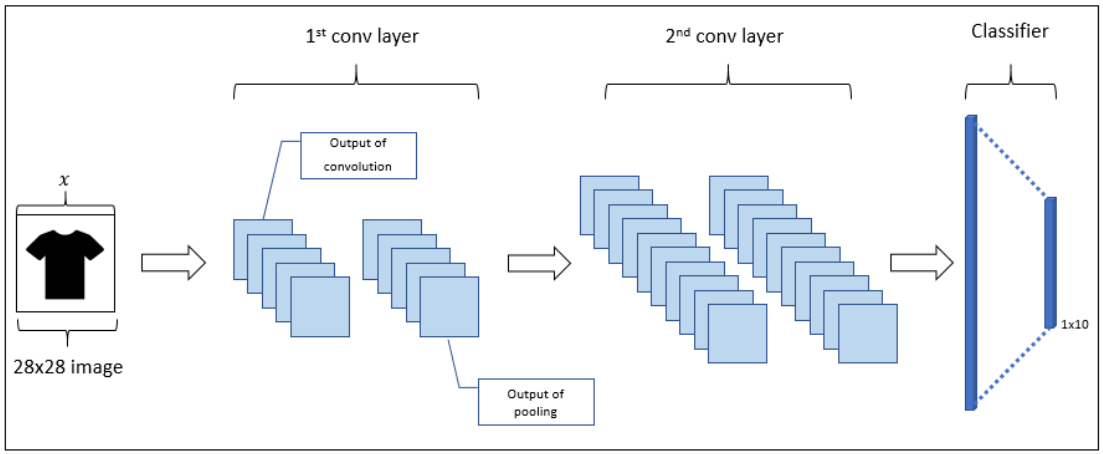


Figure: Example of CNN architecture

Questions:

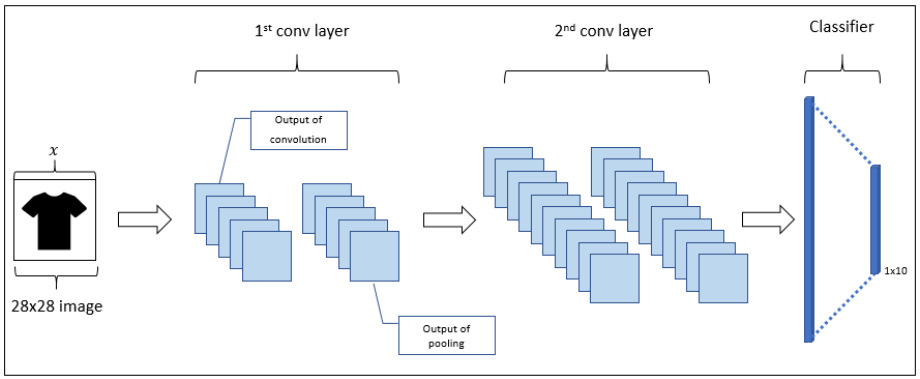
- How many filters are needed to go from the input to the 1st conv. layer?
- How many filters are needed to go from the 1st conv. layer to the second?
- How do you go from the output of the CNN to the classifier?

CNN Classifier with Pytorch

Pytorch code

```
import torch.nn.functional as F

def forward(x):
    conv_layer1 = F.relu(F.conv2d(x, filters1))
    pool_layer1 = F.max_pool2d(conv_layer1, pool_size)
    conv_layer2 = F.relu(F.conv2d(pool_layer1, filters2))
    pool_layer2 = F.max_pool2d(conv_layer2, pool_size)
    dense_features = reshape_to_vector(pool_layer2, 1440)
    fc1 = dense_features @ dense_weight + dense_bias
    return F.softmax(fc1, dim=1)
```



Why ReLu?

- Increase capacity without adding parameters
- Offer efficient optimization (no second order effect)

Weight Initialization

Xavier Glorot's Initialization

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

where n_j is the number of neurons on the previous layer and n_{j+1} is the number of neurons on the current layer.

Pytorch code

```
def glorot_initialization(n_in, n_out):
    xavier_stddev = np.sqrt(6) / np.sqrt(n_in + n_out)
    w = torch.randn(n_in, n_out) * xavier_stddev
    return Variable(w, requires_grad=True)
```

Weight Initialization

Proper weight initialization is very important for gradient stability throughout the network (see Glorot and Bengio, 2010)[4]). One can show that, assuming inputs are standardized, the pre-activation at each hidden layer can maintain an average of 0 and a variance of 1, if the weights are initialized as follow:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

where n_j is the number of neurons on the previous layer and n_{j+1} is the number of neurons on the current layer. This initialization scheme allows for the gradients at each layer to have the same distribution. This prevents multiplicative effects when applying the chain rule to compute the derivative.

[4] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Batch Normalization and Dropout

- Batch normalization paper:
  https://arxiv.org/pdf/1502.03167.pdf
- Dropout paper:
  https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf

Fashion MNIST

Figure: Examples from Fashion MNIST dataset

Fashion MNIST

- Dataset of Zalando's article images
- Training set of 60,000 examples
- Test set of 10,000 examples
- Each 28×28 grayscale image is associated with a label from 10 classes
- Human performance (Non fashion expert) $\sim 84\%$[5]

Label	0	1	2	3	4
Desc.	T-shirt	Trouser	Pullover	Dress	Coat
Label	5	6	7	8	9
Desc.	Sandal	Shirt	Sneaker	Bag	Boot

Table: Fashion MNIST classes

[5] According to https://github.com/zalandoresearch/fashion-mnist

Classification Task

A classifier is a function f_ω parametrized by ω that maps an input $\boldsymbol{x}$ (in our case, $\boldsymbol{x}$ is an image) to its category, i.e.

$$y = f_\omega(\boldsymbol{x})$$

Classification Task

A typical deep learning algorithm outputs a probability distribution rather than a category. For instance, the output could be

$$h_\omega(x) = [0.5, 0, 0.1, 0, 0, 0.1, 0, 0.2, 0.1, 0]$$

where 0.5 could be interpreted as the probability of the category being a t-shirt given the feature of the image x, i.e.

$$P(\text{"t-shirt"}|\boldsymbol{x}) = 0.5$$

Prediction

In the previous case, since the class "t-shirt" had the highest probability, our prediction would be "t-shirt", i.e.

$$\text{prediction} = \arg\max h_\omega(x) = 0$$

Conditional Maximum Likelihood

The best parameters are given by

$$\boldsymbol{\omega}_{ML} = \arg\max_{\boldsymbol{\omega}} p(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\omega})$$

where $\boldsymbol{X}$ represents all the inputs and $\boldsymbol{Y}$ represents all the targets. If the inputs are assumed to be iid, we have

$$\boldsymbol{\omega}_{ML} = \arg\max_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\omega})$$

where m is the number of inputs.

Conditional Maximum Likelihood

Example: Suppose h_ω is a MLP classifier. We have an image $\boldsymbol{x}$ of a *coat* which correspond to label 4. The corresponding one hot encoding is

$$\boldsymbol{y} = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$$

Suppose the output of the MLP is

$$h_\omega(x) = [0.5, 0, 0.1, 0, \mathbf{0.2}, 0.1, 0, 0, 0.1, 0]$$

the likelihood of this example is

$$p(y|\mathbf{x}) = \langle \boldsymbol{y}, h_\omega(\boldsymbol{x}) \rangle = 0.2$$

Loss function in classification

From conditional MLE we have

$$
\begin{aligned}
\boldsymbol{\omega}_{ML} &= \arg\max_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\omega}) \\
&= -\arg\min_{\boldsymbol{\omega}} \sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\omega}) \\
&= -\arg\min_{\boldsymbol{\omega}} \frac{1}{N} \sum_{i=1}^{N} \log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\omega}) \\
&= -\arg\min_{\boldsymbol{\omega}} \frac{1}{N} \sum_{i=1}^{N} \log \langle \boldsymbol{y}^{(i)}, h_{\omega}(\boldsymbol{x}^{(i)}) \rangle
\end{aligned}
\tag{3}
$$

Loss function in classification

Thus, we need to find ω such that

$$L(Y, h_\omega(X)) = -\frac{1}{N}\sum_{i=1}^{N} \log\langle \boldsymbol{y}^{(i)}, h_\omega(\boldsymbol{x}^{(i)})\rangle$$

is minimized. How?

Surrogate Loss Functions

Note that the classification accuracy does not affect the loss function:

$$L(\boldsymbol{y}^{(i)}, h_\omega(\boldsymbol{x}^{(i)})) = -\frac{1}{N}\sum_{i=1}^{N} \log\langle \boldsymbol{y}^{(i)}, h_\omega(\boldsymbol{x}^{(i)})\rangle$$

Batch Gradient Descent

Algorithm 1 Pseudocode for Batch Gradient Descent

Require: Learning rate ϵ_k
Require: Initial parameter $\boldsymbol{w}_0$
Require: Number of epochs T
  for $i = 1$ to T **do**
    Compute gradient $\boldsymbol{g}_t = \frac{1}{m} \nabla_w \sum_i L(h_{w_{t-1}}(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \epsilon \boldsymbol{g}_t$
  end for

Stochastic Gradient Descent

Algorithm 2 Pseudocode for Stochastic Gradient Descent

Require: Learning rate ϵ_k
Require: Initial parameter $\boldsymbol{w}_0$
Require: Number of epochs T
  for $i = 1$ to T **do**
    $X = X^{train}$.copy() and $Y = Y^{train}$.copy()
    while X is not empty **do**
      Sample $\{x^{(1)}, ..., x^{(m)}\}$ from X and $\{y^{(1)}, ..., y^{(m)}\}$ from Y
      Remove samples from X and Y
      Compute gradient $\boldsymbol{g}_t = \frac{1}{m} \nabla_w \sum_i L(h_{w_{t-1}}(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$
      Apply update: $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \epsilon \boldsymbol{g}_t$
    end while
  end for

Learning rate decay

Sufficient conditions for convergence of SGD:

- $\sum \epsilon_k = \infty$
- $\sum \epsilon_k^2 < \infty$

Common practice:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

with $\alpha = k/\tau$.