CS2109s Tutorial 9

by Lee Zong Xun

Recap: CNN and RNN

Convolutional Neural Network

A convolutional neural network is a feed-forward neural network.

- Even with domain knowledge, it is hard to define features that are useful for the task at hand.
- Exploits spatial structure of the data. Performs "**Automated**" feature engineering.
- The power of a convolutional neural network comes from a special kind of layer called the **convolutional** layer.

Convolutional Layer

- A convolutional layer is a layer that performs a convolution operation on its input.
- Performs as element-wise multiply-sum operation on a small window of the input.

Formula given by:

$$\lfloor \frac{N - K + 2P}{s} \rfloor + 1$$

Note: Kernels are learned automatically through backpropagation.

Pooling Layer (downsampling)

Extract the most relevanat features, reduce dimensionality parameters, and noise from previous convolutional layers.

- Unlike kernel, pooling layer does not have weights!
- Max pooling and average pooling are the most common types of pooling layers.

Fully Connected Layer

There can be multiple fully connected layers. An activation function is applied to the output of each fully connected layer.

Softmax Layer

The output of the last fully connected layer is passed through a softmax layer to obtain the final output.

By normalizing the output of the last fully connected layer, we can interpret the output as a probability distribution over the classes.

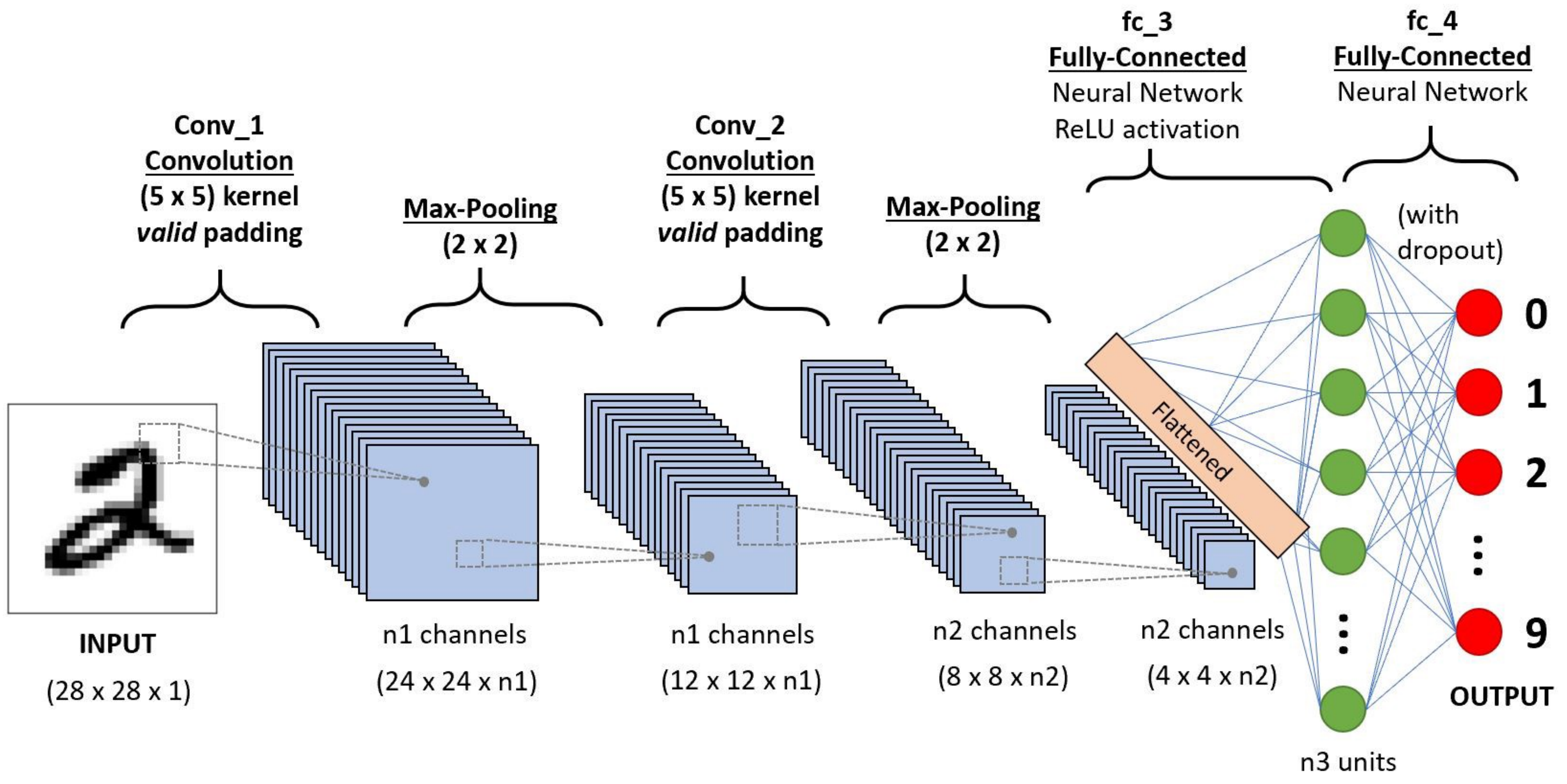
- Avoid binary classification and accomodate as many classes as needed.

Putting them together

A basic convolutional neural network can be viewed as a series of convolutional layers, followed by an activation function, followed by a pooling (downscaling) layer, repeated many times.

When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers.

- Each individual neuron makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.



Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

Recurrent Neural Network

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.

- They are distinguished by their "**memory**" as they take information from prior inputs to influence the current input and output.
- Parameters are **shared across each layer** of the network. While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network.

Backpropagation Through Time (BPTT)

The principles of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer.

- These calculations allow us to adjust and fit the parameters of the model appropriately.
- Differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

Problems

RNNs tend to run into two problems,

1. **Exploding** gradients
   Exploding gradients occur when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN.
2. **Vanishing** gradients
   When the gradient is too small, it continues to become smaller, updating the weight parameters until they become insignificant—i.e. 0. When that occurs, the algorithm is no longer learning.

Solutions

- Proper weight initialization
- Non-saturating activation functions
- Batch Normalization
- Gradient Clipping

Other technniques: **Dropout** and **Early Stopping**

Tutorial 9

Problem 1

You are given an image $\mathbf{x} \in \mathbb{R}^{4\times 4}$ and a filter $\mathbf{W} \in \mathbb{R}^{3\times 3}$. **No extra padding is added**.

0.5	0.2	0.1	0.7
0.1	0.6	0.9	0.5
0.0	0.8	0.2	0.7
0.2	0.4	0.0	0.4

0.1	0.2	0.6
0.4	0.3	0.5
0.9	0.8	0.7

Get the output feature map from this convolution and write down its output dimensions if the kernel moved with a stride of 1×1.

0.5	0.2	0.1	0.7
0.1	0.6	0.9	0.5
0.0	0.8	0.2	0.7
0.2	0.4	0.0	0.4

0.1	0.2	0.6
0.4	0.3	0.5
0.9	0.8	0.7

Solution

Output dimension: 2×2

Output feature map: $\begin{bmatrix} 1.60 & 2.59 \\ 1.51 & 1.91 \end{bmatrix}$

Problem 1(b)

Now, let's say you have access to a very deep, large CNN model. We feed a single image to the network. Each image has 3(C) channels (RGB) with a height and width of 224×224 ($H = 224, W = 224$). Here our input is a 3-dimensional tensor ($H \times W \times C$). The first layer of the big CNN is a Convolutional Layer with 96 (C_1) kernels which has a height and width of 11, and each kernel has the same number of channels as the input channel. The stride is 4×4 and no padding is used. Calculate the output size $H_1 \times W_1 \times C_1$ after the first Convolutional Layer.

Solution

For a single image and a single kernel, the output height $H_1 = \lfloor \frac{H-K+2P}{S} \rfloor + 1 = 54$. Similarly output width is $W_1 = 54$. Hence the output size is $H_1 \times W_1 \times C_1$. Here, H_1 and W_1 are the dimensions of the intermediate feature map, K is the kernel size, P is the padding, and S is the stride.

Problem 1 (c)

In most of Deep Learning libraries such as PyTorch, images are fed together as a batch B. B can take values such as $8, 16, 32, 64$. Comment on the output shape if we feed the large CNN in part (b) with a batch of images. What are the advantages of using a batch of images rather than a single image?

Solution:
The output shape will be $B \times H_1 \times W_1 \times C_1$. Using a batch of images is computationally efficient and more stable in gradient descent convergence.

Problem 2

What type of RNN model does Image captioning use? What characteristics make this a standard model for Image captioning?

Hint: Input: One image. Output: Multiple words as captions.

Problem 2

What type of RNN model does Image captioning use? What characteristics make this a standard model for Image captioning?

One-to-many model. One-to-many sequence problems are sequence problems where the input data has one time-step, and the output contains a vector of multiple values or multiple time-steps. Thus, we have a single input and a sequence of outputs.

What type of RNN model does Text classification use? What characteristics make this a standard model for Text classification?

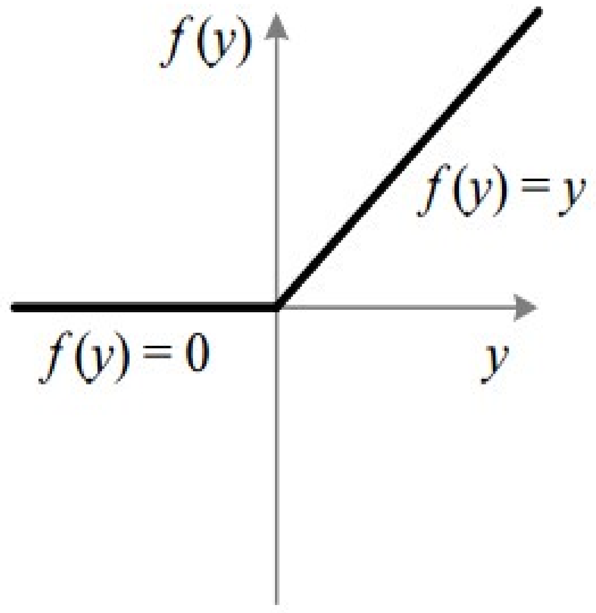
Many-to-one model. In many-to-one sequence problems, we have a sequence of data as input, and we have to predict a single output. Sentiment analysis or text classification is one such use case.
Input: Many words. Output: Which class this text belongs to.

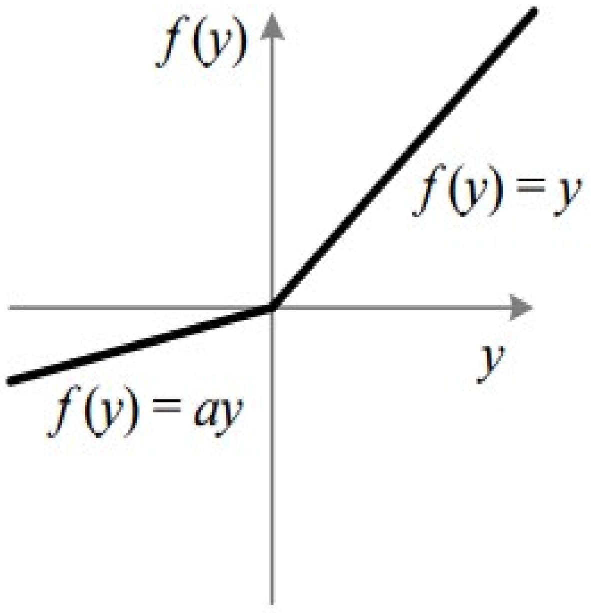
What type of RNN model does Language translation use? What characteristics make this a standard model for Language translation?

Many-to-many model. Many-to-Many sequence learning can be used for machine translation where the input sequence is in some language, and the output sequence is in some other language.
Input: Many words / code of language A. Output: Many words / code of language B.

Problem 3

The $Dying\ ReLU\ Problem$ occurs when majority of the activations are 0 (meaning the underlying pre-activations are mostly negative), resulting in the network dying midway. The gradients passed back are also 0 which leads to poor gradient descent and hence, poor learning. How does Leaky ReLU fix this?





Solution

If a ReLU activation is dead, it will always output 0 for any input. When a ReLU unit ends up this in state, it is unlikely to recover because the function gradient at 0 is 0. Leaky ReLU has a small positive gradient a for negative inputs. Having a small positive gradient for the negative inputs gives the network a chance to recover. However, this depends on the dataset being used and the value of a being set prior to training.

Problem 4

Let's apply what you learnt to a specific example, performing sentiment analysis on Covid-19 posts on twitter. Explain what characteristics of RNN make it a standard model for sentiment analysis and which RNN model you want to use to tackle this problem.

RNN is the standard method for dealing with any sequential (e.g., time series) input. As the final state of the RNN encodes the representation of the entire sentence, we use this final state to yield a classification of the sentiment of the sentence. This corresponds to a many-to-one RNN model.

Given the same context in part (a), would it be possible to perform sentiment analysis using CNN? Explain why or why not.

The key thing about sentiment analysis is that we do not just need to capture what words appear, but we also need to capture the general context of the whole sentence. The CNN model is also capable of doing sentence classification because as we add more convolution layers and make the network deeper, we will be able to detect higher-level features and capture the general context of the whole sentence.

Now let's talk about another application - image recognition. Suppose we now want to recognize whether the image contains Chihuahua or muffin, briefly explain why CNN is good for image recognition.

CNN allows us to exploit spatial structure. Convolution allows us to capture not just one pixel, but a group of pixels at a time; different convolution filters can be used to extract different features; and methods like max pooling can be used to capture higher level features and reduce dimensionality. Last but not least, kernels are learned automatically through weight updates.

We also know RNNs take in sequences (temporally-related collection of tokens). If we had to treat an image as a sequence, what would you do to allow RNNs to classify images? In other words, how can you tokenize an image? Your final method of "tokenizing" an image should ensure an RNN can be fed these tokens as inputs.

Express your creativity in coming up with different ways to go about doing so.

Possible ways are to treat each row/column as a token, or use overlapping / non-overlapping patches, or treat windows of 10 pixels as a token.
Among these, the best (as in, practical) way is to use patches as they retain locality, allowing important features (like ears or eyes) to be kept together.

Thank you

Please take your attendance here