Convolutional Neural Networks (CNNs)

CNNs are a special kind of neural network for processing data with grid-like structures

↳ • 1-D grid: time series with regular intervals
  • 2-D grid: image data

Basic Structure: CNNs are neural networks that use convolution instead of matrix multiplication in (at least) one layer

The Convolution Operator

Def

Defn The convolution $s(t) = (x * w)(t)$ of $x : \mathbb{R} \to \mathbb{R}$ and $w : \mathbb{R} \to \mathbb{R}$ is $s(t) = \int_{-\infty}^{\infty} x(a)\, w(t-a)\, da$

Terminology:
- x is often called the input
- w ... kernel
- s is sometimes called the featuremap

in CNNs

The mathematical definition of convolution is for integrals

But in ML, we are typically more interested in discrete functions

$$\hookrightarrow s(t) = (x * w)(t) = \sum_a x(a)\, w(t-a) = (w * x)(t)$$

similar to a (unnormalized) weighted moving average

For 2-D (image) data: $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$

$$= (K * I)(i,j)$$

CNNs attempt to learn the kernel

ex: $I = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 3 \\ 2 & 0 & 1 & 0 \\ 0 & 2 & 1 & 0 \end{pmatrix}, \quad K = \begin{pmatrix} 0 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 0 \end{pmatrix}$

$$\Rightarrow (I * K) = \begin{pmatrix} 2 & 1 \\ -2 & 5 \end{pmatrix}$$

$$2 = 1 \times 0 + 0 \times -1 + 0 \times 1 + 0 \times 0 + 0 \times 1 + 0 \times 0 + 2 \times 1 + 0 \times -1 + 1 \times 0$$

Motivation for CNNs

- Sparse interactions

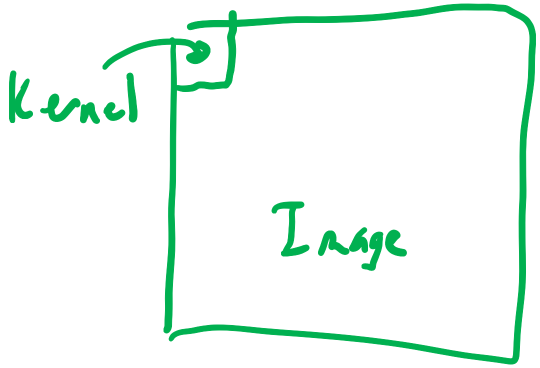
Images can have millions of pixels

Each pixel is a feature for the neural network

Thus without "sparse" network designs, any neural network would be too large to run on local hardware

If the kernel is smaller than the input, then we only need to learn a few parameters to generate the output "image"

↪ Output comes from rolling the kernel over the entire image



- <u>Parameter Sharing</u>

With the sparse structure, we repeatedly use the same kernel parameters many times

↪ Recall that the kernel slides over the entire image

⌈A single convolutional layer can include multiple kernels
This is similar to how the weight <u>matrix</u> consists of multiple linear transforms⌋

Pooling

Often we apply an additional operator to reduce dimensionality further

This is called pooling

Idea: Special structures (often) result in observations with similar values

This is a form of redundancy

↪ Commonly take the average or maximum of nearby observations

Time-Delay Neural Networks (TDNNs)

TDNNs can be thought of as a 1-D convolutional network applied to time series

Also called: temporal convolutional networks (TCN)

or dilated convolutional networks

Idea: Recall the (linear) autoregressive model

$$x_t = \beta_0 + \beta_1 x_{t-1} + \cdots + \beta_p x_{t-p} + \varepsilon_t$$

↳ 1) Replace the next time step prediction with a general signal

$$y_t = \beta_0 + \beta_1 x_{t-1} + \cdots + \beta_p x_{t-p} + \varepsilon_t \quad \text{(often } y_t = x_t\text{)}$$

2) As these are neural networks, we want nonlinear relationships

$$y_t = \phi(x_{t-1}, \dots, x_{t-p}) + \varepsilon_t$$

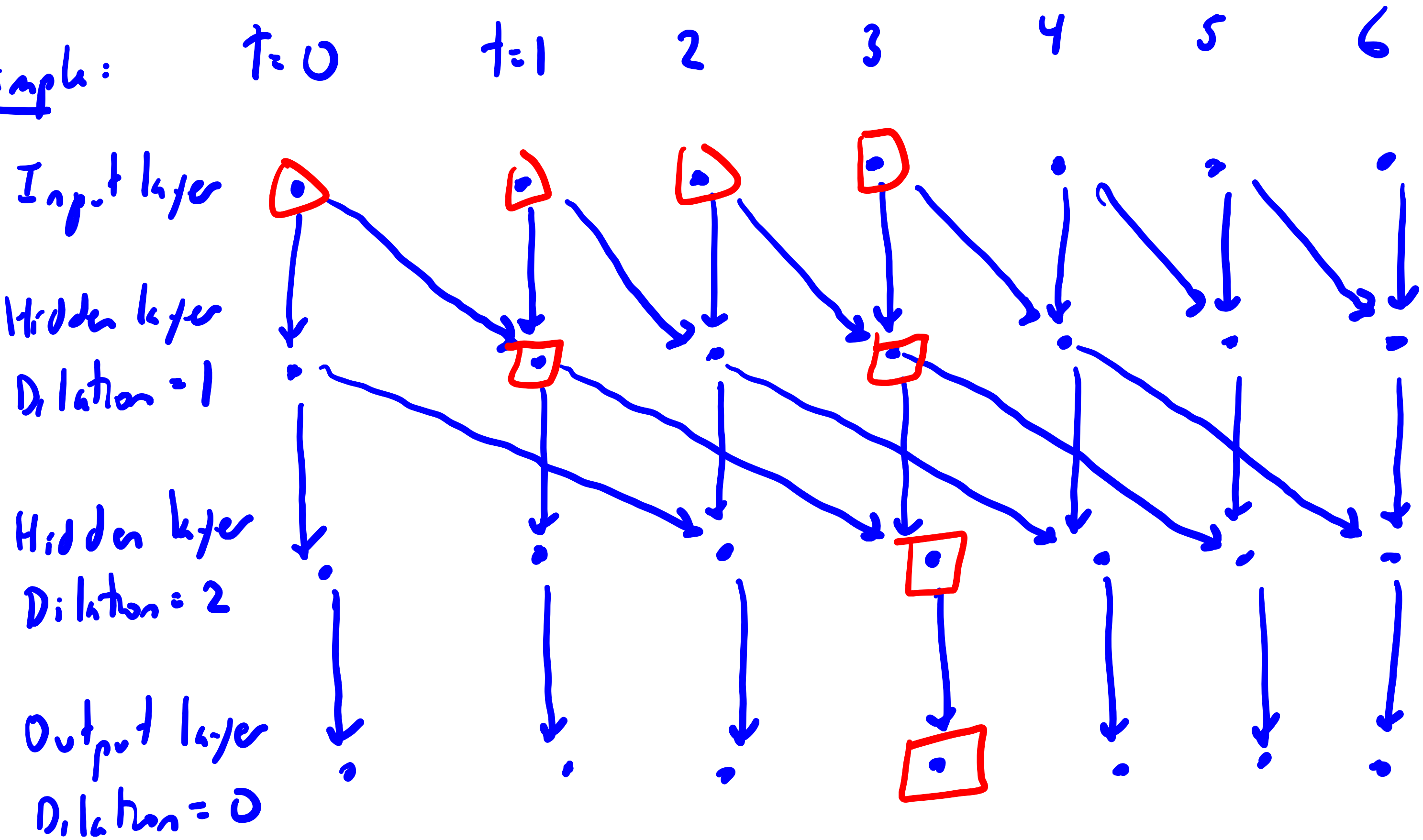
where ϕ follows a sequential structure (based on convolutions)

⌜Similar to recurrent neural networks which we will discuss in 2 weeks⌟

Basic Structure

Use <u>dilated convolutions</u> in which the kernel is applied to every d^{th} input

<u>example</u>:



$$y_3 = \phi(x_3, x_2, x_1, x_0)$$

$$= g^{(3)}\left(\beta_0^{(3)} + \beta^{(3)} h_3^{(2)}\right)$$

$$h_3^{(2)} = g^{(2)}\left(b_0^{(2)} + w_2^{(2)} h_3^{(1)} + w_0^{(2)} h_1^{(1)}\right)$$

~~bad notation~~

$$h_3^{(1)} = g^{(1)}\left(b_0^{(1)} + w_1^{(1)} x_3 + w_0^{(1)} x_2\right)$$
$$h_1^{(1)} = g^{(1)}\left(b_0^{(1)} + w_1^{(1)} x_1 + w_0^{(1)} x_0\right)$$

Note that the same weights and bias are used in this layer due to convolutional structure

__Note__: TDNNs can be thought of as a CNN with the full time series as inputs __or__ as a specific structure of a feed-forward neural net

Choice of number of hidden layers + dilations can matter a lot

[__Typical__: use dilation = 1 unless strong feeling otherwise
__ex__: to capture seasonality]

Using dilations allows the output y to be influenced by more predictors x with a sparser representation (compared to typical feed-forward neural net)

Remapping Data as an Image

If some features of the data are expected to be similar, you can remap the data set from input vectors (1-D) to input images (2-D) etc.

examples: Seasonality

Today's data is similar to data chronologically similar (yesterday) + from the same time last year

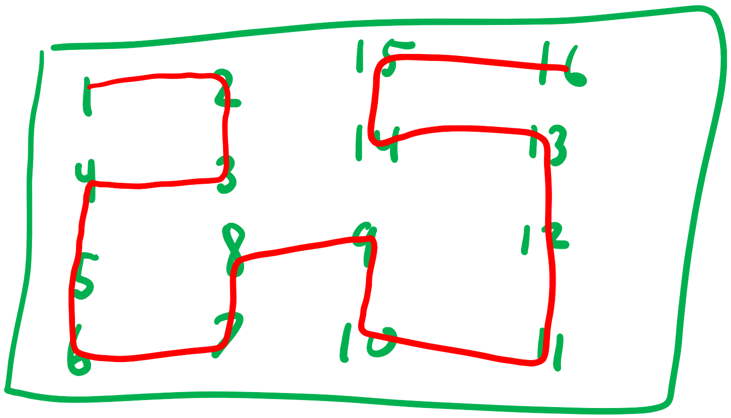
↳ Form an image by stacking rows of data based on seasonality

Ratios

Form the matrix from the ratios of the features

Sometimes use "space filling curves"

ex: 16 features



Example Financial Applications

1) Financial time series predictions

a) Use a TDNN to generalize linear autoregressive/factor models

Can be used for predicting returns/values or direction of movement

Same interpretability problem as other neural networks

b) Remap data to 2+ dimensions

Be careful because the method used to remap can greatly influence performance

c) There have been recent studies that use time series graphs

as image inputs in order to predict future movement

This is natively a 2-D input

2) Credit / ESG rating predictions

Recall: Many bonds are not rated or it is costly to rate them

Goal: Predict ratings based on balance sheet information

Because of interrelation between different features in the balance sheet,

remapping into 2-D data is often successful

3) Image analysis

There are now companies that use satellite imaging to track economic indicators like:

- crop yields
- traffic (such as at ports)
- construction

These can be vital statistics for making informed decisions