CSC413 Neural Networks and Deep Learning

Lecture 9

March 15/18, 2021

Generative Models

Generating Images

How to generate new data of certain types

- generate text that looks like our training data
- generate **images** that look like our training data

Models:

- Generative RNNs
- Autoencoder
- Variational Autoencoder (VAE)
- Generative Adversarial Networks

We'll talk about autoencoders and VAEs today

Autoencoders

There are two ways of thinking of an image autoencoder:

- a model that will eventually help us generate new images
- a model that finds a **low-dimensional representation** of images

Both are considered **unsupervised learning** tasks, since no labels are involved.

However, we do have a dataset of unlabelled images.

Image Autoencoder

Idea: In order to learn to generate images, we'll learn to **reconstruct** images from a low-dimensional representation.

An image autoencoder has two components:

Image Autoencoder

Idea: In order to learn to generate images, we'll learn to **reconstruct** images from a low-dimensional representation.

An image autoencoder has two components:

1. An **encoder** neural network that takes the image as input, and produces a low-dimensional embedding.

Image Autoencoder

Idea: In order to learn to generate images, we'll learn to **reconstruct** images from a low-dimensional representation.

An image autoencoder has two components:

1. An **encoder** neural network that takes the image as input, and produces a low-dimensional embedding.
2. A **decoder** neural network that takes the low-dimensional embedding as input, and reconstructs the image.

A good, low-dimensional representation should allow us to reconstruct everything about the image.

The components of an autoencoder

Encoder:

- Input = image
- Output = low-dimensional embedding

Decoder:

- Input = low-dimensional embedding
- Output = image

Why autoencoders?

- Dimension reduction:
  - find a low dimensional representation of the image
- Image Generation:
  - generate new images not in the training set
  - (Any guesses on how we can do this?)

Image Encoder Architecture

What would the architecture of the encoder look like?

- We could use a MLP, but there are some issues (recall: what are these issues?)
- But we can also use a convolutional neural network!

We can use downsampling to reduce the dimensionality of the data

Image Decoder Architecture

What would the architecture of the decoder look like?

We need to be able to **increase** the image resolution.

We haven't learned how to do this yet!

Transpose Convolution

Transpose Convolution

Used to increase the resolution of a feature map.

This is useful for:

- image generation problems
- pixel-wise prediction problems

Pixel-wise prediction

A prediction problem where we label the content of each pixel is known as a **pixel-wise prediction problem**

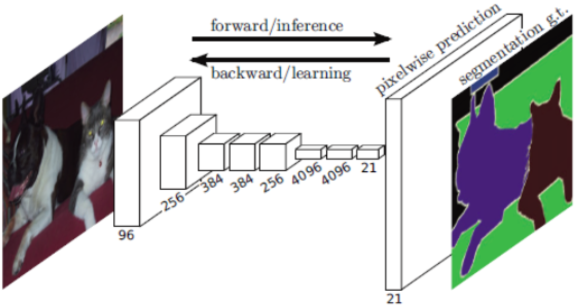


Figure 1: http://deeplearning.net/tutorial/fcn_2D_segm.html

Q: How do we generate pixel-wise predictions?

What we need:

We need to be able to **up-sample** features, i.e. to obtain high-resolution features from low-resolution features

- Opposite of max-pooling OR
- Opposite of a strided convolution

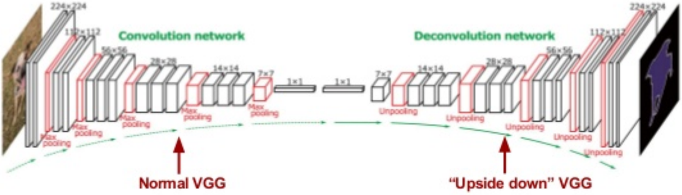
We need an **inverse** convolution – a.k.a a **deconvolution** or **transpose convolution**.

Architectures with Transpose Convolution

More than one upsampling layer

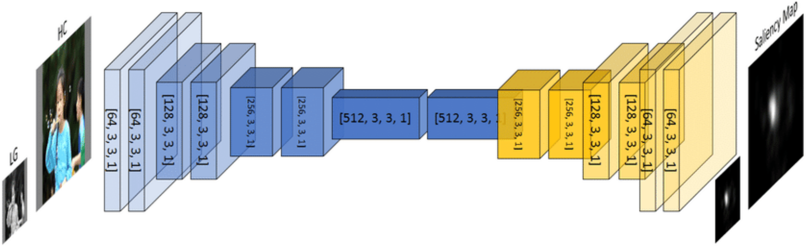
DeconvNet:

VGG-16 (conv+Relu+MaxPool) + mirrored VGG (Unpooling+'deconv'+Relu)



Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Architectures with Transpose Convolution 2



Inverse Convolution

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape
```

Inverse Convolution

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape

>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5)
>>> x = convt(y)
>>> x.shape
```

Inverse Convolution

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape

>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5)
>>> x = convt(y)
>>> x.shape
```

should get the same shape back!

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape
```

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape

>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            padding=2)
>>> x = convt(y)
>>> x.shape
```

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            padding=2)
>>> x = convt(y)
>>> x.shape
```

should get the same shape back!

Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            stride=2)
>>> x = convt(y)
>>> x.shape
```

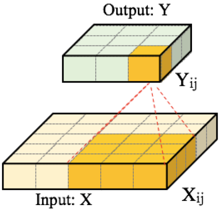
Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

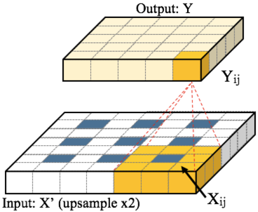
```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            stride=2)
>>> x = convt(y)
>>> x.shape
```

. . . almost the same shape . . .

Transpose Convolution Layer



(**a**) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Yis of size $W_2 = H_2 = 3$.



(**b**) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Yis of size $W_2 = H_2 = 5$.

Figure 2: https://www.mdpi.com/2072-4292/9/6/522/htm

More at `https://github.com/vdumoulin/conv_arithmetic`

Output Padding

```
nn.ConvTranspose2d(in_channels=8,
                   out_channels=8,
                   kernel_size=5,
                   stride=2,
                   output_padding=1) # +1 to output
                                     # width/height
```

Autoencoder

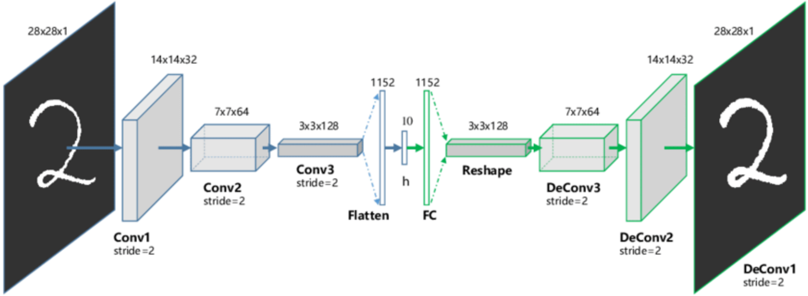
Let's get back to the autoencoder

Recall that we want a model that **generates images** that looks like our training data

Idea:

- In order to learn to generate images, we'll learn to **reconstruct** images from a low-dimensional representation.
- A good, low-dimensional representation should allow us to reconstruct everything about the image.

The components of an autoencoder



Encoder:

- Input = image
- Output = low-dimensional embedding

Decoder:

- Input = low-dimensional embedding
- Output = image

Why autoencoders?

- Dimension reduction:
  - find a low dimensional representation of the image
- Image Generation:
  - generate new images not in the training set

Autoencoders are not used for **supervised learning**. The task is *not* to predict something about the image!

Autoencoders are considered a **generative model**.

How to train autoencoders?

- Loss function:
  - How close were the reconstructed image from the original?
  - **Mean Square Error Loss**: look at the mean square error across all the pixels.
- Optimizer:
  - Just like before!
  - Commonly used for other network architectures too
- Training loop:
  - Just like before!

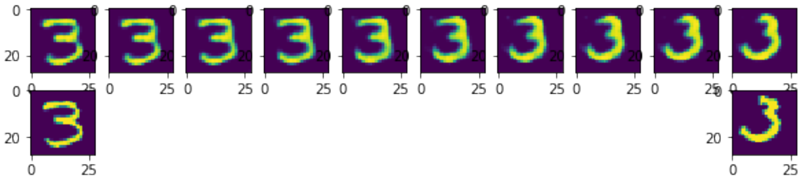
Let's train an autoencoder for MNIST

Structure in the Embedding Space

The dimensionality reduction means that there will be structure in the embedding space.

If the dimensionality of the embedding space is not too large, similar images should map to similar locations.

Interpolating in the Embedding Space



Generating New Images

Q: Can we pick a random point in the embedding space, and decode it to get an image of a digit?

A: Unfortunately not necessarily. Can we figure out why not?

Autoencoder Overfitting

Overfitting can occur if the size of the embedding space is too large.

If the dimensionality of the embedding space is small, then the neural network needs to map similar images to similar locations.

If the dimensionality of the embedding space is **too large**, then the neural network can simply memorize the images!

Blurry reconstructions

Q: Why do autoencoders produce blurry images?

Hint: it has to do with the use of the MSELoss.