Understanding_1dconv

July 17, 2023

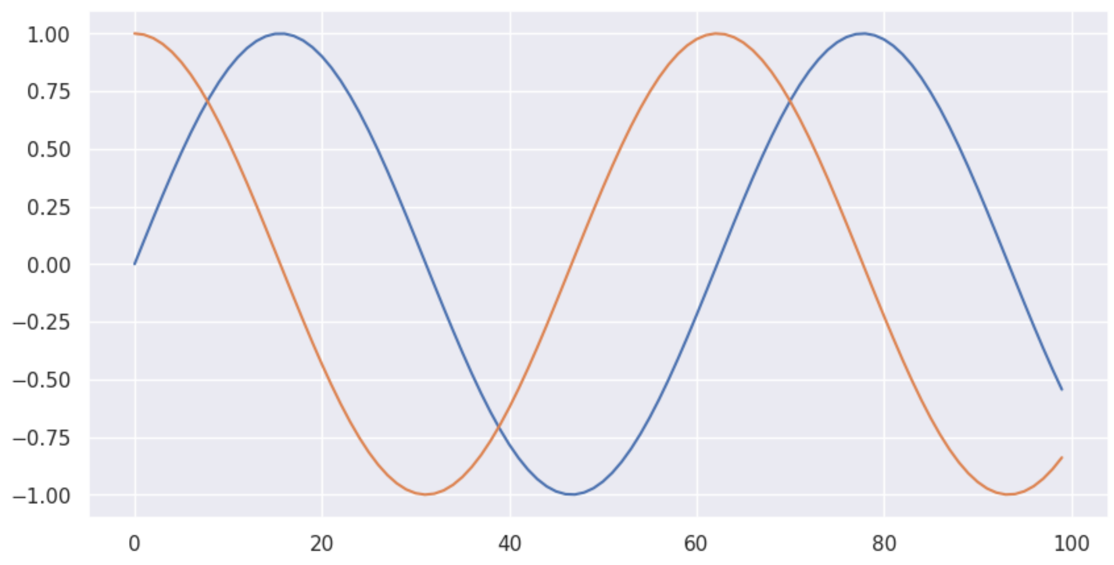
```
[ ]: #import necessary libraries
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     import torch
     import torch.nn as nn
     import torch.nn.functional as F

     #set figure size
     sns.set(rc={'figure.figsize':(10,5)})
     #darkgrid, whitegrid, dark, white, and ticks
     sns.set_style("darkgrid")
```

```
[ ]: #1d signal
     x = torch.linspace(0, 10, 100)
     y_sin = torch.sin(x)
     y_cos = torch.cos(x)
     #make two columns
     y = torch.stack((y_sin, y_cos), dim=1)
     #add batch dimension
     y = y.unsqueeze(0)
     #swap 1st and 2nd dimensions
     y = y.permute(0, 2, 1)
     print(y.shape)
```

```
torch.Size([1, 2, 100])
```

0.0.1 So, 1d conv in Pytorch takes input as (batch_size, channels, length) and outputs as (batch_size, channels, length). So, we used permute to change the order of the dimensions.

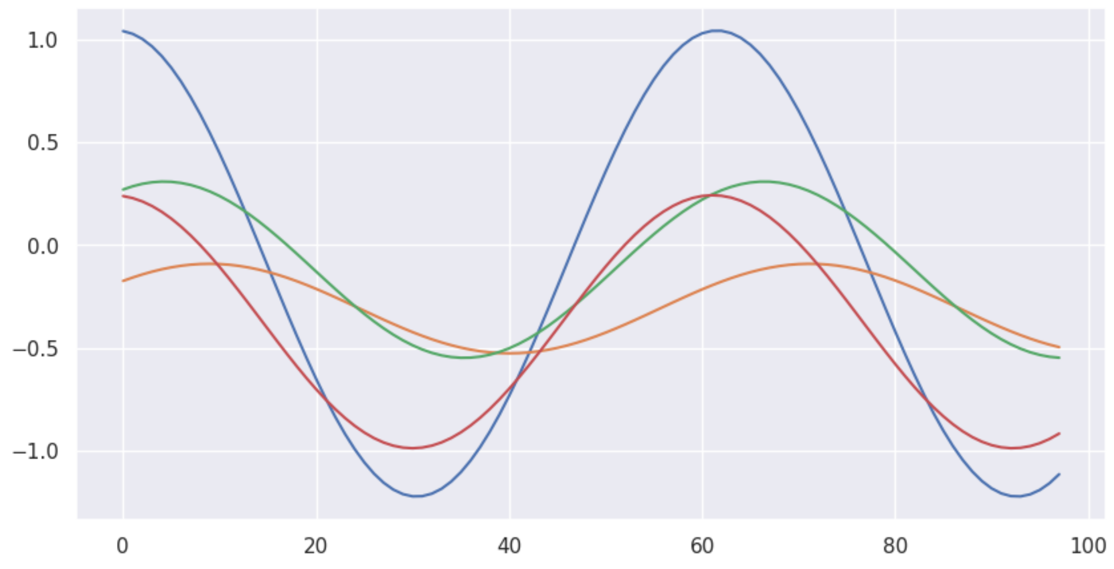
```
[ ]: #plot the signal
     plt.plot(y[0].permute(1, 0));
```



```
[ ]: #1d convolution
     conv = nn.Conv1d(in_channels=2, out_channels=4, kernel_size=3, stride=1,␣
      ↪padding='valid')
```

```
[ ]: y_hat = conv(y)
```

```
[ ]: plt.plot(y_hat[0].permute(1, 0).detach().numpy())
     plt.show()
```



0.0.2 So, what happens is, as the output channel is 4, conv1d creates a weight of 4 different kernel matrix. As the kernel size is 3, and in channel is 2, so the kernel matrix is of size 4x2x3. These 2x3 kernels stride along the length of the input and performs convolutions.

[ ]:
```
conv.weight.shape, conv.bias.shape
```

[ ]:
```
(torch.Size([4, 2, 3]), torch.Size([4]))
```

[ ]:
```
#see the weights
print(conv.weight), print(conv.bias)
```

```
Parameter containing:
tensor([[[ 0.3532, -0.0864, -0.2273],
         [ 0.3906,  0.4004,  0.4054]],

        [[-0.0145,  0.3996, -0.2601],
         [ 0.3777, -0.0403, -0.1950]],

        [[-0.1861,  0.1455,  0.3016],
         [-0.1227,  0.1366,  0.3082]],

        [[-0.2982,  0.2398,  0.0650],
         [ 0.1740,  0.0829,  0.3248]]], requires_grad=True)
Parameter containing:
tensor([-0.0893, -0.3084, -0.1192, -0.3725], requires_grad=True)
```

[ ]:
```
(None, None)
```

[ ]:
```
y.permute(0, 2, 1)[0][:5, :], y_hat.permute(0, 2, 1)[0][:5, :]
```

[ ]:
```
(tensor([[0.0000, 1.0000],
         [0.1008, 0.9949],
         [0.2006, 0.9797],
         [0.2984, 0.9544],
         [0.3931, 0.9195]]),
 tensor([[ 1.0425, -0.1738,  0.2711,  0.2394],
         [ 1.0289, -0.1572,  0.2871,  0.2293],
         [ 1.0040, -0.1421,  0.2989,  0.2131],
         [ 0.9679, -0.1288,  0.3066,  0.1908],
         [ 0.9210, -0.1172,  0.3098,  0.1629]], grad_fn=<SliceBackward0>))
```

[ ]:
```
#one manual convolution
torch.dot(conv.weight[0][0], y[0][0][:3]) + torch.dot(conv.weight[0][1],␣
 ↪y[0][1][:3]) + conv.bias[0]
```

[ ]:
```
tensor(1.0425, grad_fn=<AddBackward0>)
```

0.0.3 So, every kernel combines all the channels with convolution and then adds the bias, and make one output channel. There are 4 kernels, so there are 4 output channels. Weight of the kernel is learned by the model, and that dictates which channel contributes the most to one specific output channel.