Class Notes on GANs Models

Jack Li

October 10, 2024

Contents

1 Introduction

Provide an introduction to GANs and their importance in machine learning.

2 Basic Math

2.1 Probability Theory

- Definitions of probability, random variables, expectation, etc.

2.2 Linear Algebra

- Vectors, matrices, eigenvalues, eigenvectors, etc.

2.3 Optimization

- Gradient descent, stochastic gradient descent, etc.

3 PyTorch Basics

3.1 Tensors

- Definition and operations on tensors.

4 PyTorch training gradients

Code 1: PyTorch training gradients

```
 #SECTION: Gradient computation

# Step 1: Define a simple model
model = nn.Linear(1, 1)
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Dummy input and target
input = torch.tensor([[1.0]], requires_grad=True)
target = torch.tensor([[2.0]])

# Step 2: Print the initial parameters
print("Initial parameters:")
for param in model.parameters():
    print(param.data)

# Step 3: Forward pass
output = model(input)
loss = (output - target).pow(2).mean()

# Step 4: Zero the gradients
optimizer.zero_grad()

# Step 5: Backward pass
loss.backward()

# Step 6: Update the parameters
optimizer.step()

# Step 7: Print the parameters after the update
print("\nParameters after one training step:")
for param in model.parameters():
    print(param.data)
```

1. **Initialize Parameters:**

- Assume initial weights w and bias b are both 0.
- Model: $y = wx + b$

2. **Forward Pass:**

- Compute the output: $\hat{y} = wx + b$
- Given input $x = 1.0$ and target $y = 2.0$:

$$\hat{y} = 0 \cdot 1.0 + 0 = 0$$

3. **Compute Loss:**

- Loss function: Mean Squared Error (MSE)

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- For our single data point:

$$\text{Loss} = (0 - 2.0)^2 = 4.0$$

4. **Backward Pass (Gradient Calculation):**

- Compute gradients of the loss with respect to w and b:

$$\frac{\partial \text{Loss}}{\partial w} = 2(\hat{y} - y)x = 2(0 - 2.0) \cdot 1.0 = -4.0$$

$$\frac{\partial \text{Loss}}{\partial b} = 2(\hat{y} - y) = 2(0 - 2.0) = -4.0$$

5. **Parameter Update:**
- Using Stochastic Gradient Descent (SGD) with learning rate $\eta = 0.01$:

$$w_{\text{new}} = w - \eta \frac{\partial \text{Loss}}{\partial w} = 0 - 0.01 \cdot (-4.0) = 0.04$$
$$b_{\text{new}} = b - \eta \frac{\partial \text{Loss}}{\partial b} = 0 - 0.01 \cdot (-4.0) = 0.04$$

6. **Updated Parameters:**
- After one training step, the new parameters are:

$$w = 0.04, \quad b = 0.04$$

Summary - Initial parameters: $w = 0, b = 0$ - After one training step: $w = 0.04, b = 0.04$

4.1 Autograd

- Automatic differentiation in PyTorch.

The active selection $gradient.norm(2, dim = 1)$ is a PyTorch operation that computes the L2 norm (Euclidean norm) of the *gradient* tensor along a specified dimension. In this case, the dimension specified is $dim = 1$.

$$\begin{aligned}\Theta = &\text{argmin}_{\Theta} \tfrac{1}{B} \textstyle\sum_{i=1}^{B} \left[D\Big(z_i, \Theta\Big) - D\Big(y_i, \Theta\Big) \right] \\ &+\lambda \Big(\Big\| \tfrac{\partial D(y,\Theta)}{\partial y} \Big\| - 1 \Big)^2\end{aligned}$$

Detailed Explanation:

1. L2 Norm (Euclidean Norm):

- - The L2 norm of a vector is a measure of its magnitude and is calculated as the square root of the sum of the squares of its components. Mathematically, for a vector v, the L2 norm is given by $\|v\|_2 = \sqrt{\sum v_i^2}$.
- - In PyTorch, the *norm* function can compute various types of norms, with the L2 norm being specified by the argument 2.

2. Dimension Specification ($dim = 1$)**:**

- - The *dim* argument specifies the dimension along which the norm is computed. In a multi-dimensional tensor, this allows you to compute norms along specific axes.
- - For example, if *gradient* is a 2D tensor (matrix) with shape $[batchsize, numfeatures]$, setting $dim = 1$ means that the norm is computed for each row independently. This results in a tensor of shape $[batchsize]$, where each element is the L2 norm of the corresponding row in the original tensor.

Code 2: PyTorch gradient sampling example

```
# Define the sampling function
def sample_function(x):
    return torch.sin(x)


# NOTE: Generate sample points with requires_grad=True, and need requires_grad=True
# Generate sample points with requires_grad=True
x = torch.tensor(
    np.linspace(0, 2 * np.pi, 100), dtype=torch.float32, requires_grad=True
)

# Define f by sampling from the sample_function
f = sample_function(x)
```

```

# Compute the gradient of f with respect to x
grad = torch.autograd.grad(outputs=f, inputs=x, grad_outputs=torch.ones_like(f))

print(f"The gradient of f(x) = sin(x) at x = {x} is {grad[0]}")
```

4.2 Building Neural Networks

- Layers, activation functions, loss functions, etc.

5 GANs Models

5.1 Basic GAN

- Architecture: Generator and Discriminator.
- Loss functions: Minimax game.
- Training process.

5.2 DCGAN

- Architecture: Convolutional layers.
- Improvements over basic GAN.
- Training tips.

5.3 WGAN

- Wasserstein distance.
- Critic network.
- Gradient penalty.

5.4 CycleGAN

- Architecture: Cycle consistency loss.
- Applications: Image-to-image translation.

6 Conclusion

Summarize the key points and discuss future directions.