Class Notes on GANs Models

Jack Li

October 25, 2024

Contents

1 Introduction **2**

2 Basic Math **2**
- 2.1 Probability Theory . . . 2
- 2.2 Sampling from a Probability Distribution: Why $P(x)\,dx$? . . . 2
  - 2.2.1 Mathematical Expression . . . 2
  - 2.2.2 Why $P(x)\,dx$? . . . 2
  - 2.2.3 Example of Sampling . . . 2
- 2.3 Standard Sampling (Non-Differentiable) . . . 3
- 2.4 Reparameterization Trick (Differentiable) . . . 3
- 2.5 Gumbel-Softmax . . . 3
  - 2.5.1 Definition . . . 3
  - 2.5.2 Proof of Differentiability . . . 4
- 2.6 Concrete Distribution . . . 4
  - 2.6.1 Definition . . . 4
  - 2.6.2 Differentiability . . . 4
- 2.7 Pros and Cons . . . 4
- 2.8 Use Cases . . . 4
  - 2.8.1 Gumbel-Softmax . . . 4
  - 2.8.2 Concrete Distribution . . . 4
- 2.9 Equations Summary . . . 5
  - 2.9.1 Gumbel-Softmax Equation . . . 5
  - 2.9.2 Concrete Distribution Equation . . . 5
- 2.10 Linear Algebra . . . 5
- 2.11 Optimization . . . 5

3 PyTorch Basics **5**
- 3.1 Tensors . . . 5

4 PyTorch training gradients **5**
- 4.1 Autograd . . . 6
- 4.2 Building Neural Networks . . . 7
- 4.3 Convolutional Neural Networks (CNNs) basics . . . 8

5 GANs Models **9**
- 5.1 Basic GAN . . . 9
- 5.2 DCGAN . . . 9
- 5.3 WGAN . . . 9
- 5.4 CycleGAN . . . 9

6 Loss Functions **10**

7 Evaluation Metrics **10**
- 7.1 Frechet Inception Distance (FID) . . . 10
- 7.2 Inception Score (IS) . . . 11

8 Fourier Series and Fast Fourier Transform (FFT): Mathematical Explanation and Comparison 11
8.1 1. Fourier Series . . . 11
8.2 2. Fast Fourier Transform (FFT) . . . 11
8.3 Comparison . . . 11
8.4 Python Code for FFT and Inverse FFT . . . 11
8.5 Example: FFT and Inverse FFT Calculation . . . 12
8.5.1 Given Signal . . . 12
8.5.2 FFT Computation (Step-by-Step) . . . 12
8.5.3 Inverse FFT Computation . . . 12

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 Sampling from a Probability Distribution: Why $P(x)\,dx$?

In continuous probability theory, the **probability density function** (PDF) $P(x)$ represents the density of the probability at a particular point x. However, the actual probability of the random variable X falling within a small interval around x, say $[x, x+dx]$, is given by the product of the PDF at x and the small interval dx.

2.2.1 Mathematical Expression

The probability of X falling within the interval $[x, x+dx]$ is approximately:

$$P(X \in [x, x+dx]) \approx P(x)\,dx$$

Where: - $P(x)$ is the probability density function evaluated at x, - dx is an infinitesimally small interval around x.

2.2.2 Why $P(x)\,dx$?

- The PDF $P(x)$ by itself does not give the actual probability for any specific value of x, because for continuous random variables, the probability of any single point is zero:

$$P(X = x) = 0 \quad \text{for continuous variables.}$$

- Instead, the **probability** is found by integrating the PDF over an interval:

$$P(X \in [a, b]) = \int_a^b P(x)\,dx$$

For a very small interval dx, the integral simplifies to:

$$P(X \in [x, x+dx]) \approx P(x)\,dx$$

This shows that the product $P(x)\,dx$ gives the probability mass within the small region $[x, x+dx]$.

2.2.3 Example of Sampling

When sampling a value x from a distribution, the probability that a sample lies within a small range $[x, x+dx]$ is proportional to $P(x)\,dx$. For example, in the case of a Gaussian (Normal) distribution, the probability density is given by:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

To find the probability of the random variable X falling within the range $[x, x+dx]$, we compute:

$$P(X \in [x, x + dx]) \approx P(x)\, dx$$

2.3 Standard Sampling (Non-Differentiable)

In the standard sampling method, we sample directly from a normal distribution, which does not allow gradients to propagate:

```
# Non-differentiable direct sampling
mu = torch.tensor(0.0, requires_grad=True)
sigma = torch.tensor(1.0, requires_grad=True)

# Direct sampling from a normal distribution
z = torch.normal(mu, sigma)

# Loss function
loss = (z - 5) ** 2
loss.backward()  # This breaks the gradient flow
```

This will not compute gradients correctly because the sampling is a discrete, non-differentiable operation.

2.4 Reparameterization Trick (Differentiable)

In the reparameterization trick, we express the random variable z as a function of μ, σ, and a noise term ϵ, which allows backpropagation to compute gradients:

$$z = \mu + \sigma \cdot \epsilon$$

```
# Differentiable sampling using reparameterization trick
mu = torch.tensor(0.0, requires_grad=True)
log_sigma = torch.tensor(0.0, requires_grad=True)
sigma = torch.exp(log_sigma)

# Sample epsilon from N(0, 1)
epsilon = torch.randn_like(sigma)

# Reparameterization: z = mu + sigma * epsilon
z = mu + sigma * epsilon

# Loss function
loss = (z - 5) ** 2
loss.backward()  # Gradient flows through mu and sigma
```

Here, ϵ is sampled from a standard normal distribution $\mathcal{N}(0, 1)$, and the parameters μ and σ can be updated by gradient descent because the whole process is now differentiable.

2.5 Gumbel-Softmax

2.5.1 Definition

The Gumbel-Softmax trick is a method for sampling from a categorical distribution in a differentiable way. It provides a continuous approximation to a categorical distribution, making it suitable for use in backpropagation.

Let $\pi_1, \pi_2, \ldots, \pi_k$ be the probabilities of each category in a categorical distribution. The Gumbel-Softmax trick works by introducing Gumbel noise g_i for each category i, sampled from a Gumbel distribution:

$$g_i = -\log(-\log(u_i)), \quad u_i \sim \text{Uniform}(0, 1)$$

The continuous approximation to the categorical distribution is given by the softmax function:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)}$$

where τ is the temperature parameter controlling the sharpness of the approximation. As $\tau \to 0$, the distribution becomes more discrete (closer to one-hot), and as $\tau \to \infty$, the distribution becomes more uniform.

2.5.2 Proof of Differentiability

Let $\pi = (\pi_1, \pi_2, \dots, \pi_k)$ be the parameter of the categorical distribution. The reparameterization with Gumbel noise is differentiable because:

$$\frac{\partial y_i}{\partial \pi_j} = \frac{\partial}{\partial \pi_j} \left(\frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \right)$$

Since g_i is independent of π_j, the function remains differentiable.

2.6 Concrete Distribution

2.6.1 Definition

The Concrete distribution is similar to the Gumbel-Softmax but is defined for both binary and multinomial distributions. The Concrete distribution for binary random variables introduces a continuous relaxation by using the logistic sigmoid function for binary variables.

For a binary random variable with probability p, the Concrete distribution samples:

$$z = \frac{\log(p) + g}{\tau}$$

where $g \sim \text{Gumbel}(0, 1)$ and τ is the temperature parameter. The output z is passed through the sigmoid function to produce a value between 0 and 1:

$$y = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

2.6.2 Differentiability

As in the Gumbel-Softmax case, the Concrete distribution is differentiable because the sampling is reparameterized using Gumbel noise, and the final step uses differentiable functions like the softmax or sigmoid.

2.7 Pros and Cons

Method	**Pros**	**Cons**
Gumbel-Softmax	- Differentiable approximation of categorical variables. - Control over discreteness via temperature τ. - Easy to implement for multiclass problems.	- Approximation error due to continuous relaxation. - Gradients vanish as $\tau \to 0$. - Limited to categorical (one-hot) outputs.
Concrete Distribution	- Differentiable for both binary and categorical variables. - More flexibility than Gumbel-Softmax for binary decisions.	- Same vanishing gradient issue as Gumbel-Softmax for low τ. - Still an approximation of a discrete distribution, not exact.

Table 1: Comparison of Gumbel-Softmax and Concrete distribution

2.8 Use Cases

2.8.1 Gumbel-Softmax

- **Generative Models**: Used in **Variational Autoencoders (VAEs)** for categorical latent variables, allowing for discrete sampling while maintaining differentiability. - **Reinforcement Learning**: In policy gradient methods, Gumbel-Softmax can be used for discrete action selection.

2.8.2 Concrete Distribution

- **Binary Decision Making**: Useful in models where decisions are binary, like in the **Binary Variational Autoencoder (Binary VAE)**. - **Binary Latent Variables**: Concrete distributions work well for models with binary latent variables, where a differentiable approximation is required.

2.9 Equations Summary

2.9.1 Gumbel-Softmax Equation

For categorical variables, the Gumbel-Softmax approximation is given by:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)}$$

where $g_i \sim \text{Gumbel}(0, 1)$.

2.9.2 Concrete Distribution Equation

For binary variables, the Concrete distribution is given by:

$$z = \frac{\log(p) + g}{\tau}, \quad g \sim \text{Gumbel}(0, 1)$$

and the relaxed binary sample is:

$$y = \frac{1}{1 + \exp(-z)}$$

2.10 Linear Algebra

- Vectors, matrices, eigenvalues, eigenvectors, etc.

2.11 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 $dim = 1$.

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

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

```python
# 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.

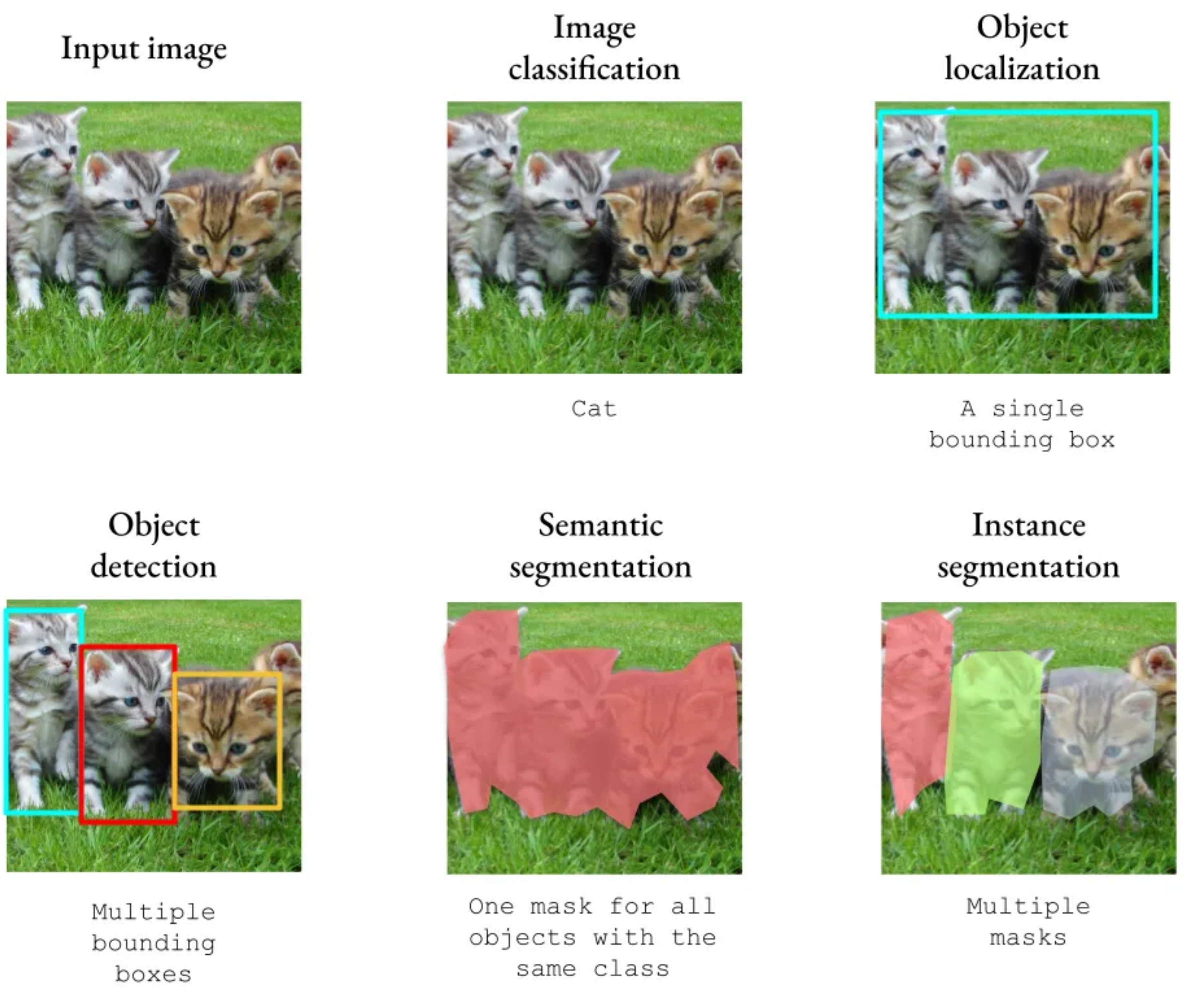

Figure 1: An example image illustrating segmentation.

4.3 Convolutional Neural Networks (CNNs) basics

— PyTorch Conv2d Equation

The output size of a Conv2d layer can be calculated using the following equation:

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} + 2 \times \text{Padding} - \text{Kernel Size}}{\text{Stride}} \right\rfloor + 1$$

Where: - Input Size is the size of the input feature map (height or width). - Padding is the number of zero-padding added to both sides of the input. - Kernel Size is the size of the convolution kernel (height or width). - Stride is the stride of the convolution.

PyTorch ConvTranspose2d Equation

The output size of a ConvTranspose2d (transposed convolution) layer can be calculated using the following equation:

$$\text{Output Size} = (\text{Input Size} - 1) \times \text{Stride} - 2 \times \text{Padding} + \text{Kernel Size} + \text{Output Padding}$$

Where: - Input Size is the size of the input feature map (height or width). - Stride is the stride of the convolution. - Padding is the number of zero-padding added to both sides of the input. - Kernel Size is the size of the convolution kernel (height or width). - Output Padding is the additional size added to the output (usually used to ensure the output size matches a specific value).

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 Loss Functions

Model	Loss Function
Basic GAN (2014)	$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$
CGAN(2014)	$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x\|y)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z\|y)))]$$
DCGAN (2015)	$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$ $$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}[\log D(G(z))]$$
WGAN (2017)	$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))]$$
LSGAN (2017)	$$\mathcal{L}_D = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}(x)}[(D(x) - 1)^2] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[D(G(z))^2]$$ $$\mathcal{L}_G = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z)) - 1)^2]$$
WGAN-GP (2017)	$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[D(x)] - \mathbb{E}_{z\sim p_z(z)}[D(G(z))] + \lambda \mathbb{E}_{\hat{x}\sim p_{\hat{x}}(\hat{x})}\left[(\\|\nabla_{\hat{x}} D(\hat{x})\\|_2 - 1)^2\right]$$
CycleGAN (2017)	$$\mathcal{L}_{\text{cyc}}(G,F) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\\|F(G(x)) - x\\|_1] + \mathbb{E}_{y\sim p_{\text{data}}(y)}[\\|G(F(y)) - y\\|_1]$$
F-GAN (2018)	$$\mathcal{L}_D = \mathbb{E}_{x\sim p_{\text{data}}(x)}[f'(D(x))] - \mathbb{E}_{z\sim p_z(z)}[f'(D(G(z)))]$$ $$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}[f(D(G(z)))]$$

Table 2: Loss functions for various generative models

7 Evaluation Metrics

7.1 Frechet Inception Distance (FID)

Description: FID measures the distance between the feature vectors of real and generated images, capturing both the mean and covariance differences.

Equation:

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

where:

- μ_r, Σ_r are the mean and covariance of the real images' feature vectors.
- μ_g, Σ_g are the mean and covariance of the generated images' feature vectors.

7.2 Inception Score (IS)

Description: IS evaluates the quality and diversity of generated images by using the Inception v3 network to classify them. High-quality images should have a high confidence score for a single class, and diverse images should be spread across many classes.

Equation:

$$\text{IS} = \exp\left(\mathbb{E}_{x\sim p_g}\left[D_{\text{KL}}(p(y|x)\|p(y))\right]\right)$$

where:

- $p(y|x)$ is the conditional label distribution given an image x.
- $p(y)$ is the marginal label distribution.
- D_{KL} is the Kullback-Leibler divergence.

8 Fourier Series and Fast Fourier Transform (FFT): Mathematical Explanation and Comparison

Fourier Series and the Fast Fourier Transform (FFT) are tools for representing signals in the frequency domain.

8.1 1. Fourier Series

The **Fourier Series** decomposes a *periodic continuous function* $f(t)$ with period T into a sum of sine and cosine functions with different frequencies:

$$f(t) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\left(\frac{2\pi nt}{T}\right) + b_n \sin\left(\frac{2\pi nt}{T}\right)\right)$$

where a_0 is the mean (DC) component, and a_n and b_n represent the amplitude of each frequency component.

8.2 2. Fast Fourier Transform (FFT)

The **Fast Fourier Transform (FFT)** is an algorithm for computing the *Discrete Fourier Transform (DFT)* of a discrete, finite-length signal. The DFT is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n]\cdot e^{-i2\pi kn/N}$$

where $x[n]$ is the input signal (with N samples) and $X[k]$ is the DFT output for each frequency k. The FFT reduces the complexity to $O(N\log N)$.

8.3 Comparison

Feature	Fourier Series	FFT
Input Type	Continuous, periodic functions	Discrete, finite-length signals
Output Type	Sine and cosine terms for harmonics	Complex frequency components
Computational Complexity	Integral calculations	$O(N\log N)$ algorithm
Applications	Continuous signals	Discrete signals

8.4 Python Code for FFT and Inverse FFT

```python
import numpy as np

def fft(x):
    N = len(x)
    if N <= 1:
        return x
    even = fft(x[0::2])
    odd = fft(x[1::2])
    T = np.exp(-2j * np.pi * np.arange(N) / N)[:N // 2] * odd
    return np.concatenate([even + T, even - T])

```

```
def ifft(X):
    N = len(X)
    if N <= 1:
        return X
    X_conj = np.conjugate(X)
    X_fft = fft(X_conj)
    return np.conjugate(X_fft) / N
```

8.5 Example: FFT and Inverse FFT Calculation

8.5.1 Given Signal

Consider a simple 4-point signal $x = [1, 2, 3, 4]$. We?ll compute the FFT to transform this signal into the frequency domain and then apply the inverse FFT to retrieve the original signal.

8.5.2 FFT Computation (Step-by-Step)

For an $N = 4$-point signal, the DFT formula is:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i2\pi kn/N}$$

1. Compute $X[0]$:

$$X[0] = x[0] \cdot e^{-i2\pi \cdot 0 \cdot 0/4} + x[1] \cdot e^{-i2\pi \cdot 0 \cdot 1/4} + x[2] \cdot e^{-i2\pi \cdot 0 \cdot 2/4} + x[3] \cdot e^{-i2\pi \cdot 0 \cdot 3/4}$$

$$= 1 + 2 + 3 + 4 = 10$$

2. Compute $X[1]$:

$$X[1] = x[0] \cdot e^{-i2\pi \cdot 1 \cdot 0/4} + x[1] \cdot e^{-i2\pi \cdot 1 \cdot 1/4} + x[2] \cdot e^{-i2\pi \cdot 1 \cdot 2/4} + x[3] \cdot e^{-i2\pi \cdot 1 \cdot 3/4}$$

$$= -2 + 2i$$

And so on. The FFT of $x = [1, 2, 3, 4]$ is $X = [10, -2 + 2i, -2, -2 - 2i]$.

8.5.3 Inverse FFT Computation

To verify the result, compute the inverse FFT:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{i2\pi kn/N}$$

1. Compute $x[0]$:

$$x[0] = \frac{1}{4}(10 + (-2 + 2i) + (-2) + (-2 - 2i)) = 1$$

Similarly, $x[1] = 2$, $x[2] = 3$, and $x[3] = 4$.
Thus, we reconstruct $x = [1, 2, 3, 4]$.