3.
Diffusion Models

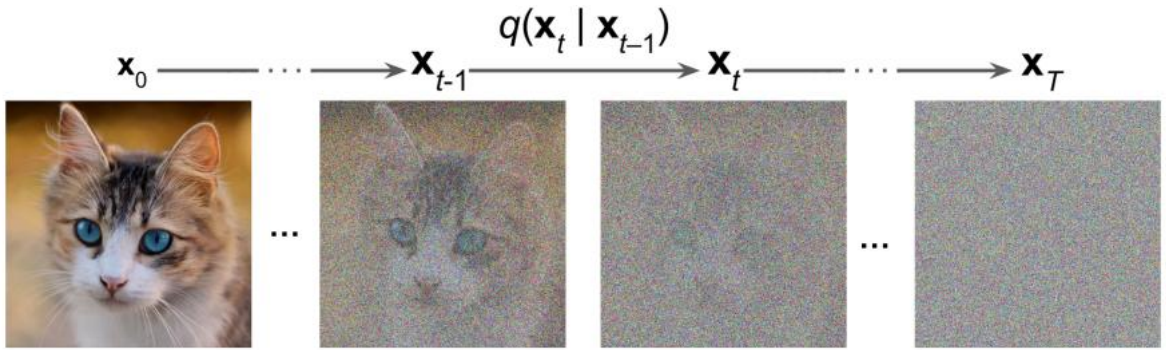
Diffusion Models

- In 2015, Sohl-Dickstein et al. applied tools from thermodynamics to model a diffusion process, similar to a drop of milk diffusing in a cup of tea.
  - The core idea is to train a model to learn the reverse process: start from the completely mixed state, and gradually "unmix" the milk from the tea.
- In 2020, Ho et al. mana… model capable of generating highly realistic images, …*noising diffusion probabilistic model* (DDPM).
- In 2021 OpenAI researc… architecture and proposed several improvements t… …ally beat GANs.
  - DDPMs are much eas…
  - Their generated imag… …igher quality.
  - The main downside … …e a very long time to generate images, compared to …

Forward Process

- *Forward process*: start with a picture $\mathbf{x}_0$, and at each time step t add a little bit of Gaussian noise to the image, with mean 0 and variance β_t.
  - This noise is independent for each pixel: we call it *isotropic*.
  - Rescale the pixel values slightly at each step, by a factor of $\sqrt{1-\beta_t}$.

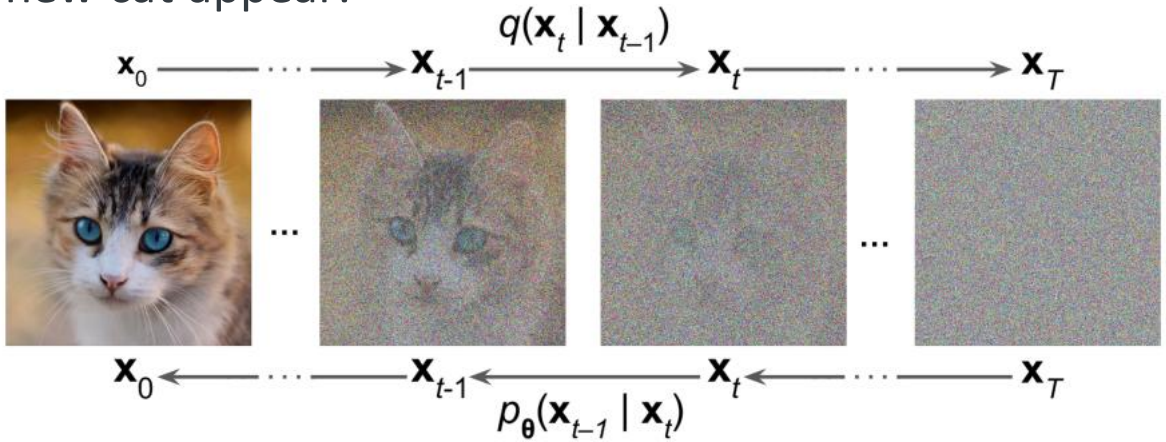
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = N(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$



$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = N(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t\mathbf{I})$ similar to $\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t$ where $\boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \boldsymbol{I})$

Reverse Process

- Train a model that can perform the *reverse process* from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$.
  - We can then use it to remove a tiny bit of noise from an image, and repeat the operation many times until all the noise is gone.
- If we train the model on a dataset of cat images, then we can give it a picture full of Gaussian noise, and the model will gradually make a brand new cat appear.



Variance Schedule

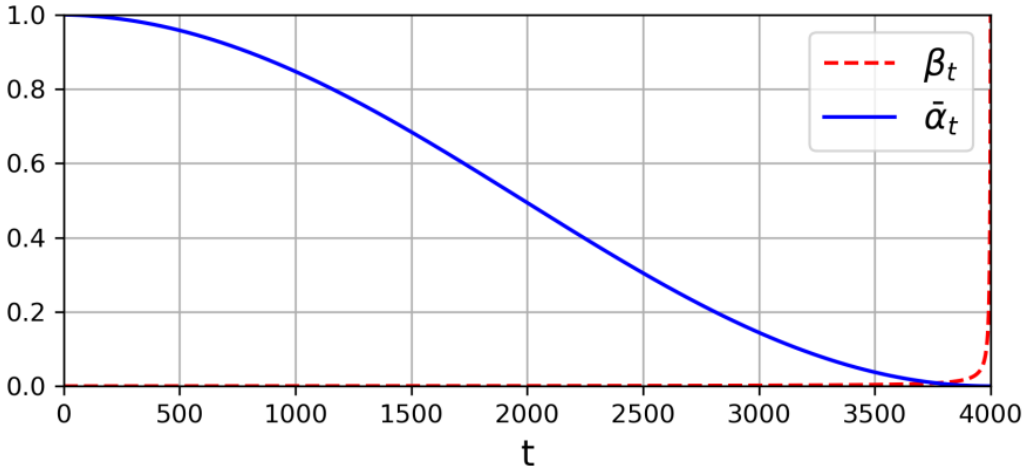
- If we define $\alpha_t = 1 - \beta_t$, after t time steps, the cat signal will have been multiplied by a factor of $\bar{\alpha}_t = \alpha_1 \times \alpha_2 \times \cdots \times \alpha_t = \prod_{i=1}^{t} \alpha_i$
- It's this "cat signal" factor $\bar{\alpha}_t$ that we want to schedule so it shrinks down from 1 to 0 gradually between time steps 0 and *T*.
- Shortcut for the forward diffusion process:

$$q(\mathbf{x}_t|\mathbf{x}_0) = N(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0\,, (1 - \bar{\alpha}_t)\mathbf{I})$$

- Variance schedule equations for the forward diffusion process:

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, \text{ with } \bar{\alpha}_t = \frac{f(t)}{f(0)} \text{ and } f(t) = \cos\left(\frac{t/T + s}{1+s} . \frac{\pi}{2}\right)^2$$

Cosine Variance Schedule



Linear Schedule

Cosine Schedule

Variance Schedule

```python
def variance_schedule(T, s=0.008, max_beta=0.999):
    t = np.arange(T + 1)
    f = np.cos((t / T + s) / (1 + s) * np.pi / 2) ** 2
    alpha = np.clip(f[1:] / f[:-1], 1 - max_beta, 1)
    alpha = np.append(1, alpha).astype(np.float32)  # add α₀ = 1
    beta = 1 - alpha
    alpha_cumprod = np.cumprod(alpha)
    return alpha, alpha_cumprod, beta  # αₜ , ᾱₜ , βₜ for t = 0 to T

T = 4000
alpha, alpha_cumprod, beta = variance_schedule(T)
```

$$\bar{\alpha}_t = \alpha_1 \times \alpha_2 \times \cdots \times \alpha_t$$

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, \text{ with } \bar{\alpha}_t = \frac{f(t)}{f(0)} \text{ and } f(t) = \cos\left(\frac{t/T+s}{1+s} \cdot \frac{\pi}{2}\right)^2$$

Preparing Batch for Training

- To train our model to reverse the diffusion process, we will need noisy images from different time steps of the forward process.
- For this, we create a `prepare_batch()` function that will take a batch of clean images from the dataset and prepare them:

```
def prepare_batch(X):
    X = tf.cast(X[..., tf.newaxis], tf.float32) * 2 - 1  # scale from -1 to +1
    X_shape = tf.shape(X)  # X_shape : (batch_size, height, width, 1)
    t = tf.random.uniform([X_shape[0]], minval=1, maxval=T + 1, dtype=tf.int32)
    alpha_cm = tf.gather(alpha_cumprod, t)
    # reshape alpha_cm to [batch size, 1, 1, 1] for broadcasting:
    alpha_cm = tf.reshape(alpha_cm, [X_shape[0]] + [1] * (len(X_shape) - 1))
    noise = tf.random.normal(X_shape)
    return {
        "X_noisy": alpha_cm ** 0.5 * X + (1 - alpha_cm) ** 0.5 * noise,
        "time": t,
    }, noise
```

$$\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t \text{ where } \boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \boldsymbol{I})$$

Creating Training and Validation Sets

- We'll create a training dataset and a validation set that will apply the `prepare_batch`() function to every batch.
- `X_train` and `X_valid` contain the Fashion MNIST images with pixel values ranging from 0 to 1.

```
def prepare_dataset(X, batch_size=32, shuffle=False):
    ds = tf.data.Dataset.from_tensor_slices(X)
    if shuffle:
        ds = ds.shuffle(10_000)
    return ds.batch(batch_size).map(prepare_batch).prefetch(1)

train_set = prepare_dataset(X_train, batch_size=32, shuffle=True)
valid_set = prepare_dataset(X_valid, batch_size=32)
```

Building the Model

- The actual diffusion model can be any model you want, as long as it takes the noisy images and time steps as inputs, and predicts the noise to subtract from the input images.
- The DDPM authors used a modified U-Net architecture which is similar to a fully convolutional network.

```
def build_diffusion_model():
    X_noisy = tf.keras.layers.Input(shape=[28, 28, 1], name="X_noisy")
    time_input = tf.keras.layers.Input(shape=[], dtype=tf.int32, name="time")
    [...] # build the model based on the noisy images and the time steps
    outputs = [...] # predict the noise (same shape as the input images)
    return tf.keras.Model(inputs=[X_noisy, time_input], outputs=[outputs])
```

Training the Model

- The authors noted that using the MAE loss worked better than the MSE during model training.
  - You can also use the Huber loss.

```
model = build_diffusion_model()
model.compile(loss=tf.keras.losses.Huber(), optimizer="nadam")

# add a ModelCheckpoint callback
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("my_diffusion_model",
                                                   save_best_only=True)

history = model.fit(train_set, validation_data=valid_set, epochs=100,
                    callbacks=[checkpoint_cb])
```

Generating New Images

- No shortcut in the reverse diffusion process: we have to sample $\mathbf{x}_T$ randomly from a Gaussian distribution with mean 0 and variance 1, then pass it to the model to predict the noise; and subtract it from the image to get $\mathbf{x}_{T-1}$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right) + \sqrt{\beta_t}\mathbf{z}$$

- $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ represents the noise predicted by the model given the input image $\mathbf{x}_t$ and the time step t.
  - $\boldsymbol{\theta}$ represents the model parameters.
- $\boldsymbol{z}$ is Gaussian noise with mean 0 and variance 1.
  - This makes the reverse process stochastic: if you run it multiple times, you will get different images.

Generating New Images

```
def generate(model, batch_size=32):
    X = tf.random.normal([batch_size, 28, 28, 1])
    for t in range(T - 1, 0, -1):
        noise = (tf.random.normal if t > 1 else tf.zeros)(tf.shape(X))
        X_noise = model({"X_noisy": X, "time": tf.constant([t] * batch_size)})
        X = (
            1 / alpha[t] ** 0.5
            * (X - beta[t] / (1 - alpha_cumprod[t]) ** 0.5 * X_noise)
            + (1 - alpha[t]) ** 0.5 * noise
        )
    return X

X_gen = generate(model)  # generated images
```

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right) + \sqrt{\beta_t}\mathbf{z}$$

Generating New Images

Latent Diffusion Model

- *Latent diffusion models*: where the diffusion process takes place in latent space, rather than in pixel space.
- A powerful autoencoder is used to compress each training image into a much smaller latent space, where the diffusion process takes place, then the autoencoder is used to decompress the final latent representation, generating the output image.
  - This considerably speeds up image generation, and reduces training time and cost dramatically.
  - The quality of the generated images is outstanding.
- A powerful pretrained latent diffusion model named *Stable Diffusion* was open sourced in August 2022 by a collaboration between LMU Munich and a few companies, including StabilityAI.

https://stablediffusion.com

Stable Diffusion
Free. Simple. Unlimited!

Create | Log in

Home | Playground | WebUI | SD 3.5 Large | Free Tools | Best Prompts | Free ChatGPT | Monetize Art | Guides

Free Online

AI Image Generator

AI Design tools

Create stunning AI art, images, anime, and realistic photos with our advanced, free-to-use AI tools.

Simple Prompts, Unlimited Creativity!

No Cost!

Try it now:

Generate AI Image For Free

7,656,253+ IMAGES GENERATED BY 1,043,200+ USERS