Advanced Deep Learning

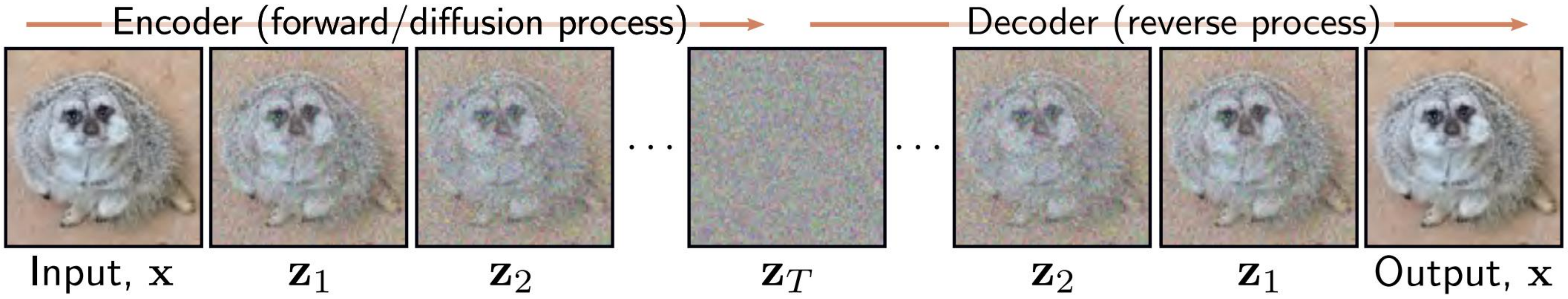
DATA.ML.230

Diffusion models

From: Simon J. D. Prince, Chapter 18 – Understanding Deep Learning, MIT Press (19 May 2025)

Diffusion models

- Approximate the data likelihood using a lower bound based on an encoder that maps to the latent variable:
  - The encoder is predetermined (noising process)
- Probabilistic models that define a nonlinear mapping from latent variables to observed data (decoder)
  - The decoder learns to inverse the encoding process



Diffusion models

Encoder (diffusion or forward process)

- Map a data example **x** through a series of variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$

$$\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\
\mathbf{z}_t &= \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t \in 2,\ldots,T
\end{aligned}$$

How quickly the noise is blended (noise schedule)

Noise drawn from a standard Normal distribution

Diffusion models

Encoder (diffusion or forward process)

- Map a data example **x** through a series of variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$

$$
\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\
\mathbf{z}_t &= \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t \in 2,\ldots,T
\end{aligned}
$$

$\Big\Downarrow$ Forward process

$$
\begin{aligned}
q(\mathbf{z}_1|\mathbf{x}) &= \text{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right] \\
q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right] \qquad \forall\, t \in \{2,\ldots,T\}
\end{aligned}
$$

Diffusion models

Encoder (diffusion or forward process)

- Map a data example **x** through a series of variables $\mathbf{z}_1$, $\mathbf{z}_2$, ..., $\mathbf{z}_T$

$$\begin{aligned} \mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\ \mathbf{z}_t &= \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t\in 2,\ldots,T \end{aligned}$$

$\Downarrow$ Forward process

$$\begin{aligned} q(\mathbf{z}_1|\mathbf{x}) &= \mathrm{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right] \\ q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \mathrm{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right] \qquad \forall\, t\in\{2,\ldots,T\} \end{aligned}$$

Markov chain:
Each probability of $\mathbf{z}_t$ is determined by the preceding variable $\mathbf{z}_{t-1}$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$

time-consuming process

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$
- Diffusion kernel leads to a closed-form expression for $q(\mathbf{z}_t|\mathbf{x})$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$
- Diffusion kernel leads to a closed-form expression for $q(\mathbf{z}_t|\mathbf{x})$

$$
\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1 \\
\mathbf{z}_2 &= \sqrt{1-\beta_2}\cdot\mathbf{z}_1 + \sqrt{\beta_2}\cdot\boldsymbol{\epsilon}_2
\end{aligned}
$$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$
- Diffusion kernel leads to a closed-form expression for $q(\mathbf{z}_t|\mathbf{x})$

$$
\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x}+\sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1\\
\mathbf{z}_2 &= \sqrt{1-\beta_2}\cdot\mathbf{z}_1+\sqrt{\beta_2}\cdot\boldsymbol{\epsilon}_2
\end{aligned}
$$

$\Big\Downarrow$ Substitute $\mathbf{z}_1$ in $\mathbf{z}_2$

$$
\mathbf{z}_2 = \sqrt{(1-\beta_2)(1-\beta_1)}\cdot\mathbf{x}+\sqrt{1-(1-\beta_2)(1-\beta_1)}\cdot\boldsymbol{\epsilon}
$$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$
- Diffusion kernel leads to a closed-form expression for q($\mathbf{z}_t$|$\mathbf{x}$)

$$\begin{aligned}\mathbf{z}_1 &= \sqrt{1-\beta_1}\cdot\mathbf{x}+\sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1\\ \mathbf{z}_2 &= \sqrt{1-\beta_2}\cdot\mathbf{z}_1+\sqrt{\beta_2}\cdot\boldsymbol{\epsilon}_2\end{aligned}$$

Substitute $\mathbf{z}_1$ in $\mathbf{z}_2$

$$\mathbf{z}_2 = \sqrt{(1-\beta_2)(1-\beta_1)}\cdot\mathbf{x}+\sqrt{1-(1-\beta_2)(1-\beta_1)}\cdot\boldsymbol{\epsilon}$$

$$\boxed{\begin{aligned}&\mathbf{z}_t = \sqrt{\alpha_t}\cdot\mathbf{x}+\sqrt{1-\alpha_t}\cdot\boldsymbol{\epsilon}\\ &\text{where}\quad \alpha_t = \textstyle\prod_{s=1}^{t} 1-\beta_s\end{aligned}}$$

Diffusion models

Diffusion kernel

- To train the decoder we use multiple samples $\mathbf{z}_t$ at time t for the same input $\mathbf{x}$
- Diffusion kernel leads to a closed-form expression for $q(\mathbf{z}_t|\mathbf{x})$

$$q(\mathbf{z}_t|\mathbf{x}) = \mathrm{Norm}_{\mathbf{z}_t}\left[\sqrt{\alpha_t}\cdot\mathbf{x}, (1-\alpha_t)\mathbf{I}\right]$$

Diffusion models

Decoder (reverse process)

- Learns a series of probabilistic mappings back from latent variable $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$, starting from t = T, until we reach the data $\mathbf{x}$
- We approximate the reverse distributions by normal distributions:

$$
\begin{aligned}
Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\
Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t], \sigma_t^2\mathbf{I}\Big] \\
Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) &= \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1, \boldsymbol{\phi}_1], \sigma_1^2\mathbf{I}\Big],
\end{aligned}
$$

Diffusion models

Decoder (reverse process)

- Learns a series of probabilistic mappings back from latent variable $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$, starting from t = T, until we reach the data $\mathbf{x}$
- We approximate the reverse distributions by normal distributions:

$$
\begin{aligned}
Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\
Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \phi_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\left[\mathbf{f}_t[\mathbf{z}_t, \underset{\uparrow}{\phi_t}], \sigma_t^2\mathbf{I}\right] \\
Pr(\mathbf{x}|\mathbf{z}_1, \phi_1) &= \text{Norm}_{\mathbf{x}}\left[\mathbf{f}_1[\mathbf{z}_1, \phi_1], \sigma_1^2\mathbf{I}\right],
\end{aligned}
$$

Neural network mapping $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$

Diffusion models

Decoder (reverse process)

- Learns a series of probabilistic mappings back from latent variable $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$, starting from t = T, until we reach the data $\mathbf{x}$
- We approximate the reverse distributions by normal distributions:

$$
\begin{aligned}
Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\
Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \phi_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t, \phi_t], \sigma_t^2\mathbf{I}\Big] \\
Pr(\mathbf{x}|\mathbf{z}_1, \phi_1) &= \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1, \phi_1], \sigma_1^2\mathbf{I}\Big],
\end{aligned}
$$

Neural network mapping $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$

- Data generation using ancestral sampling

Diffusion models

Decoder (reverse process)

- Training by maximizing the log-likelihood of the training data $\{\mathbf{x}_i\}$:

$$\hat{\boldsymbol{\phi}}_{1\ldots T} = \underset{\boldsymbol{\phi}_{1\ldots T}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i | \boldsymbol{\phi}_{1\ldots T}) \right] \right]$$

where $Pr(\mathbf{x}|\boldsymbol{\phi}_{1\ldots T}) = \int Pr(\mathbf{x}, \mathbf{z}_{1\ldots T} | \boldsymbol{\phi}_{1\ldots T}) d\mathbf{z}_{1\ldots T}$

Diffusion models

Decoder (reverse process)

- Training by maximizing the log-likelihood of the training data $\{\mathbf{x}_i\}$:

$$\hat{\phi}_{1\ldots T} = \underset{\phi_{1\ldots T}}{\operatorname{argmax}} \left[\sum_{i=1}^{I} \log\left[Pr(\mathbf{x}_i|\phi_{1\ldots T})\right]\right]$$

where $Pr(\mathbf{x}|\phi_{1\ldots T}) = \int Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\phi_{1\ldots T}) d\mathbf{z}_{1\ldots T}$

Intractable integral

➔ Use ELBO

$$\mathrm{ELBO}\left[\phi_{1\ldots T}\right] = \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \log\left[\frac{Pr(\mathbf{x}, \mathbf{z}_{1\ldots T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right] d\mathbf{z}_{1\ldots T}$$

Diffusion models

Decoder (reverse process)

$$
\begin{aligned}
\mathrm{ELBO}\left[\boldsymbol{\phi}_{1\ldots T}\right]
&= \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \log\left[\frac{Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\boldsymbol{\phi}_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right] d\mathbf{z}_{1\ldots T} \\
&\approx \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \left(\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\right] + \sum_{t=2}^{T}\log\left[\frac{Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\right]\right) d\mathbf{z}_{1\ldots T} \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\Big[\underbrace{\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\right]}_{\text{Reconstruction accuracy}}\Big] - \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\Big[\mathrm{D}_{KL}\Big[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\,\Big|\Big|\,Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)\Big]\Big]
\end{aligned}
$$

- ELBO will be larger if the model prediction matches the data
- We use Monte Carlo estimate as in VAE

Diffusion models

Decoder (reverse process)

$$
\begin{aligned}
\mathrm{ELBO}\big[\boldsymbol{\phi}_{1\ldots T}\big] \\
&= \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \log\left[\frac{Pr(\mathbf{x},\mathbf{z}_{1\ldots T}|\boldsymbol{\phi}_{1\ldots T})}{q(\mathbf{z}_{1\ldots T}|\mathbf{x})}\right] d\mathbf{z}_{1\ldots T} \\
&\approx \int q(\mathbf{z}_{1\ldots T}|\mathbf{x}) \left(\log\big[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\big] + \sum_{t=2}^{T} \log\left[\frac{Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)}{q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}\right]\right) d\mathbf{z}_{1\ldots T} \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\Big[\log\big[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\big]\Big] - \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\Big[\underbrace{\mathrm{D}_{KL}\Big[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\Big|\Big|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)\Big]}_{\text{Distance between normal distributions}}\Big]
\end{aligned}
$$

$$
\frac{1}{2\sigma_t^2}\left\|\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\,\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x} - \mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t]\right\|^2 + C
$$

Diffusion models

Diffusion loss function

$$L[\phi_{1\ldots T}] = \sum_{i=1}^{I}\Bigg(\overbrace{-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\phi_1],\sigma_1^2\mathbf{I}\right]\Big]}^{\text{reconstruction term}}$$

$$+\sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\Bigg\|\underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_{it}+\frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_i}_{\text{target, mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})}-\underbrace{\mathbf{f}_t[\mathbf{z}_{it},\phi_t]}_{\text{predicted } \mathbf{z}_{t-1}}\Bigg\|^2\Bigg)$$

L[$\boldsymbol{\phi}_{1\ldots T}$] is used to train a network for each diffusion time step

Diffusion models

Reparameterization of loss function

Idea: Learn to predict the noise that was mixed with the original data example to create the current variable

$$\mathbf{z}_t = \sqrt{\alpha_t}\cdot\mathbf{x} + \sqrt{1-\alpha_t}\cdot\boldsymbol{\epsilon} \quad \Longrightarrow \quad \mathbf{x} = \frac{1}{\sqrt{\alpha_t}}\cdot\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\cdot\boldsymbol{\epsilon}$$

Substituting **x** to the loss function:

$$\begin{aligned} L[\boldsymbol{\phi}_{1\ldots T}] \quad = \quad & \sum_{i=1}^{I}\Bigg(-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big] \\ & +\sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\left\|\left(\frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_{it} - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\boldsymbol{\epsilon}_{it}\right) - \mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]\right\|^2\Bigg) \end{aligned}$$

Diffusion models

Replace the model $\hat{\mathbf{z}}_{t-1} = \mathbf{f}_t[\mathbf{z}_t, \phi_t]$ with a new model $\hat{\boldsymbol{\epsilon}} = \mathbf{g}_t[\mathbf{z}_t, \phi_t]$ predicting the noise ϵ mixed with **x** to create $\mathbf{z}_t$:

$$\mathbf{f}_t[\mathbf{z}_t, \phi_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \phi_t]$$

Diffusion models

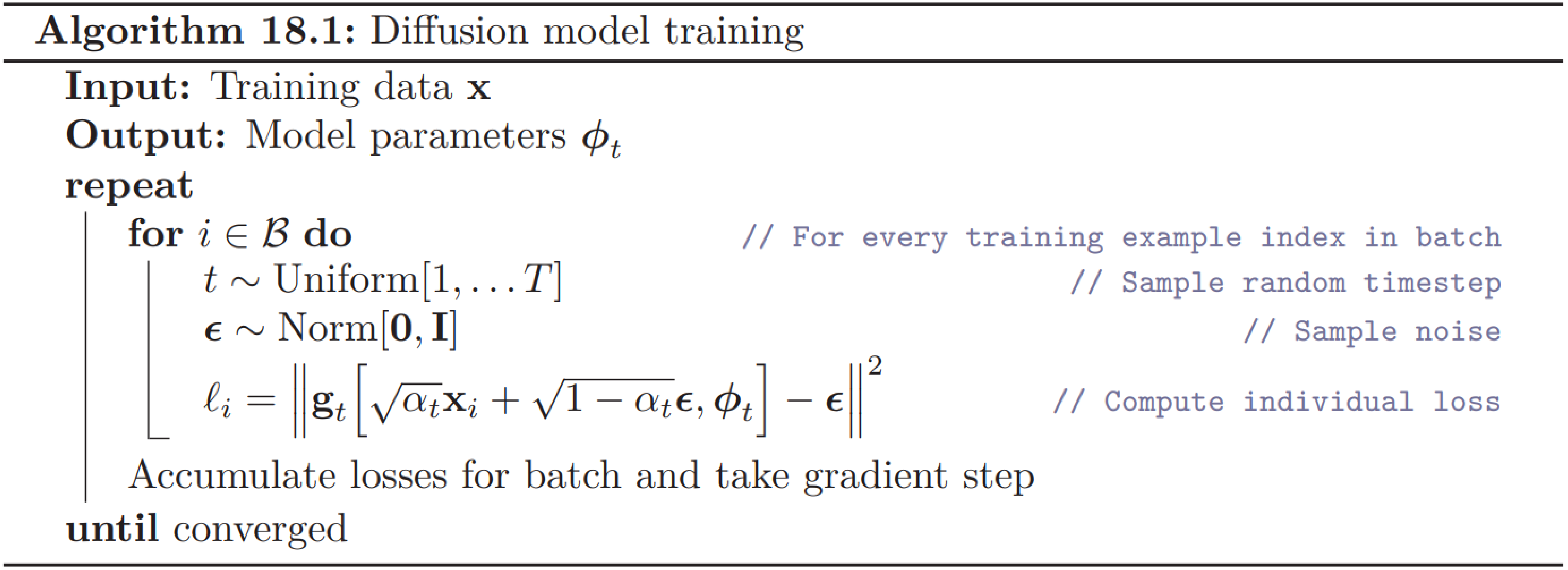
Replace the model $\hat{\mathbf{z}}_{t-1} = \mathbf{f}_t[\mathbf{z}_t, \phi_t]$ with a new model $\hat{\boldsymbol{\epsilon}} = \mathbf{g}_t[\mathbf{z}_t, \phi_t]$ predicting the noise ϵ mixed with **x** to create $\mathbf{z}_t$:

$$\mathbf{f}_t[\mathbf{z}_t, \phi_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \phi_t]$$

Substituting to the loss function leads to:

$$L[\phi_{1\ldots T}] = \sum_{i=1}^{I}\sum_{t=1}^{T} \underbrace{\frac{\beta_t^2}{(1-\alpha_t)(1-\beta_t)2\sigma_t^2}}_{\text{In practice the scaling factors are ignored}} \left\| \mathbf{g}_t[\mathbf{z}_{it}, \phi_t] - \boldsymbol{\epsilon}_{it} \right\|^2$$

Diffusion models



Algorithm 18.1: Diffusion model training

```
Input: Training data $\mathbf{x}$
Output: Model parameters $\boldsymbol{\phi}_t$
repeat
    for $i \in \mathcal{B}$ do                                      // For every training example index in batch
        $t \sim \text{Uniform}[1, \ldots T]$                         // Sample random timestep
        $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$    // Sample noise
        $\ell_i = \left\| \mathbf{g}_t\left[\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon}\right\|^2$    // Compute individual loss
    Accumulate losses for batch and take gradient step
until converged
```

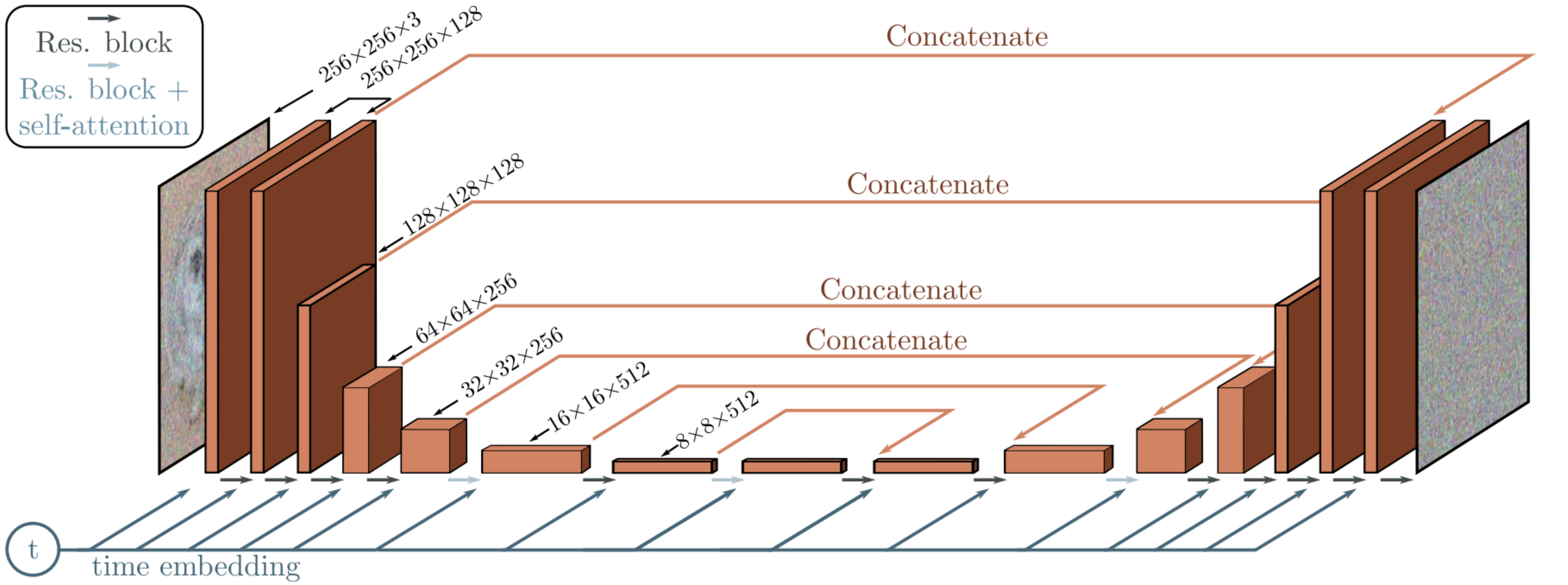
Diffusion models

Algorithm 18.2: Sampling

```
Input: Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$
Output: Sample, $\mathbf{x}$
$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$    // Sample last latent variable
for $t = T \ldots 2$ do
    $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$    // Predict previous latent variable
    $\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$    // Draw new noise vector
    $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t \boldsymbol{\epsilon}$    // Add noise to previous latent variable
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$    // Generate sample from $\mathbf{z}_1$ without noise
```

Applications of Diffusion models

Image generation



Instead of training multiple networks, train one and enhance its input with a (predetermined) vector representing the time step

Applications of Diffusion models

Conditional generation using classifier guidance:

- Modifies the denoising update to using class information c
- Uses a classifier on the latent variable $\mathbf{z}_t$
- Gradients of the classifier affect the parameters of the generating neural network

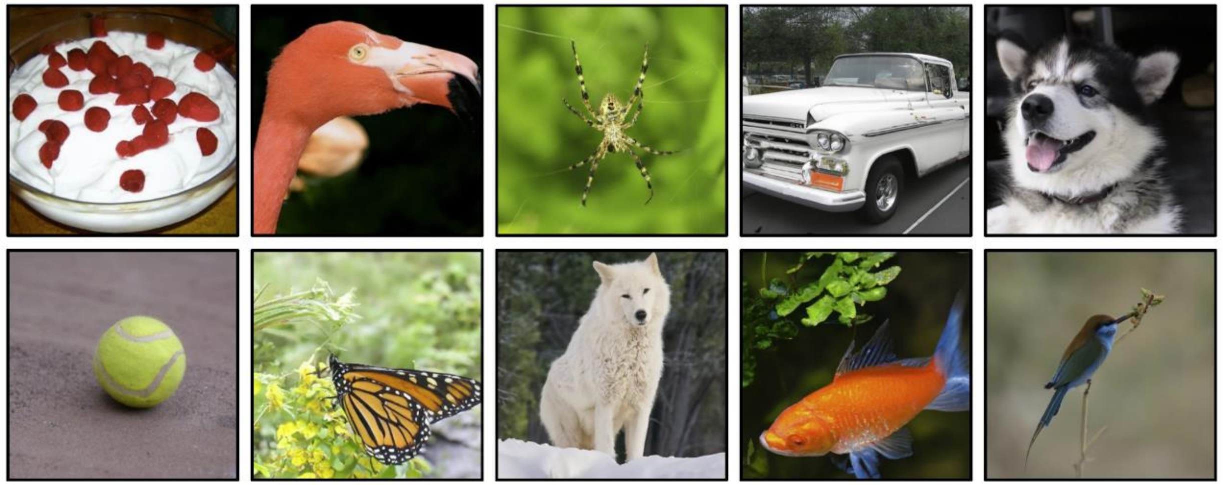


Figure 18.12 Conditional generation using classifier guidance. Image samples conditioned on different ImageNet classes. The same model produces high quality samples of highly varied image classes. Adapted from Dhariwal & Nichol (2021).

Applications of Diffusion models

Conditional generation using text:

- Condition the generating network with text embeddings

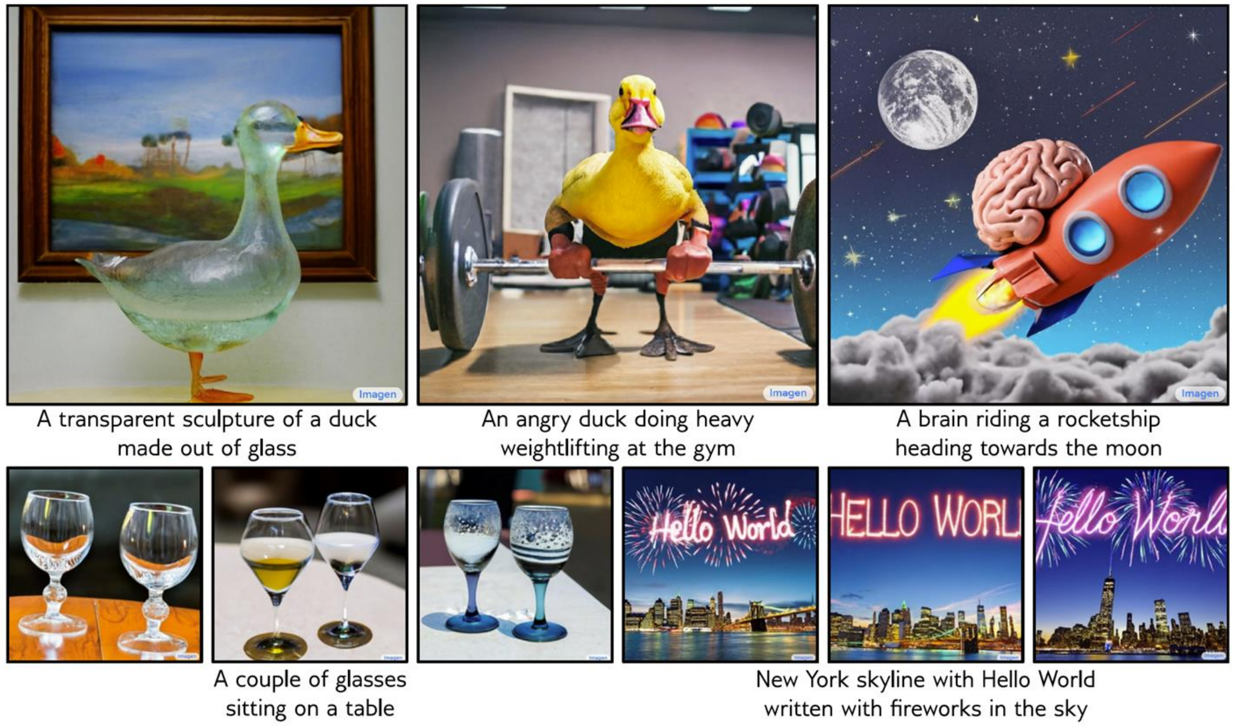
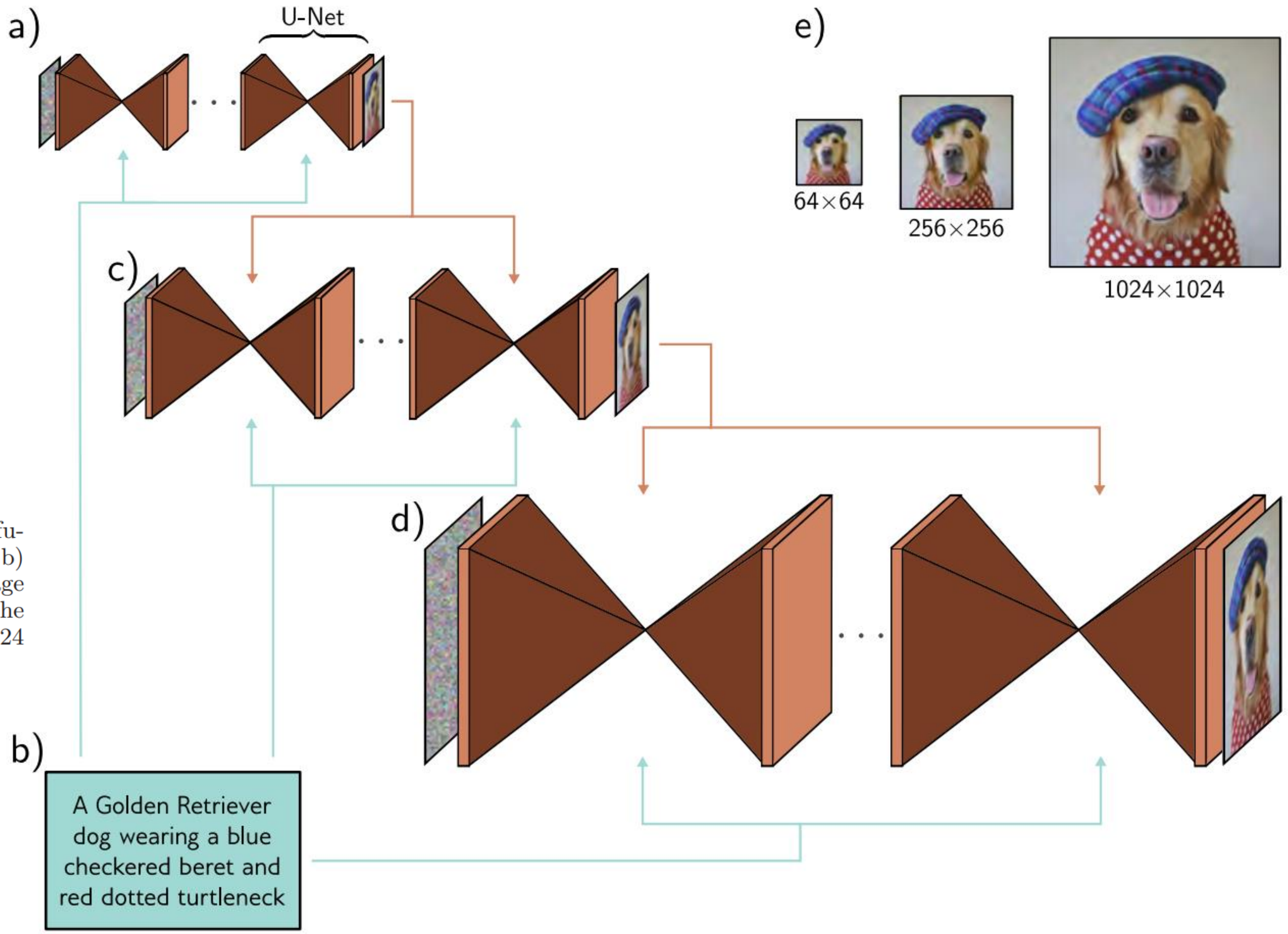


Imagen
A transparent sculpture of a duck made out of glass
An angry duck doing heavy weightlifting at the gym
A brain riding a rocketship heading towards the moon
A couple of glasses sitting on a table
Hello World
HELLO WORL
New York skyline with Hello World written with fireworks in the sky

Figure 18.13 Conditional generation using text prompts. Synthesized images from a cascaded generation framework, conditioned on a text prompt encoded by a large language model. The stochastic model can produce many different images compatible with the prompt. The model can count objects and incorporate text into images. Adapted from Saharia et al. (2022b).

Applications of Diffusion models

Conditional generation of higher resolution images



a)
U-Net
e)
64×64
256×256
1024×1024
c)
d)
b)
A Golden Retriever dog wearing a blue checkered beret and red dotted turtleneck

Figure 18.11 Cascaded conditional generation based on a text prompt. a) A diffusion model consisting of a series of U-Nets is used to generate a 64×64 image. b) This generation is conditioned on a sentence embedding computed by a language model. c) A higher resolution 256×256 image is generated and conditioned on the smaller image *and* the text encoding. d) This is repeated to create a 1024×1024 image. e) Final image sequence. Adapted from Saharia et al. (2022b).