COM3025 - Deep Learning and Advanced AI

Revision Notes

Pascal U

Last updated: May 29, 2025

Contents

Lecture 1 – *Attention is All You Need* **2**
1.1 What is a Transformer? . . . 2
1.2 Attention!!! . . . 2

Lecture 2 – *Attention Cont.* **4**
2.1 Positional Encoding Cont. . . . 4

Lecture 3– *LLMs, VLMs and other multimodal models* **6**
3.1 Large Language Models . . . 6
3.2 Mixture of Experts . . . 6
3.3 Parameter-Efficient Fine Tuning . . . 7
3.4 Longer Contexts in Transformers . . . 7
3.5 Parametric-Efficient Fine-Tuning . . . 8
3.6 Normalization . . . 9

Lecture 4 – *Vision Transformers and Autoencoders* **10**
4.1 Vision Transformers . . . 10
4.2 Autoencoders . . . 12
4.3 U-Net . . . 14

Lecture 5 – *Generative Models 1* **14**
5.1 Evidence Lower-Bound . . . 15
5.2 Diffusion . . . 15

Lecture 6 – *Generative Models 2* **16**
6.1 Diffusion Cont. . . . 16
6.2 GANs . . . 18

6.3 CycleGAN 19

※ Lecture 1

1.1 What is a Transformer?

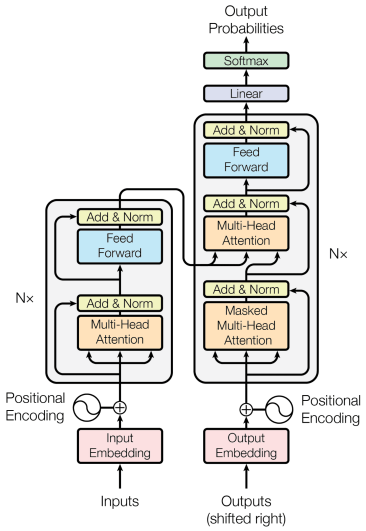


Figure 1: The Transformer model architecture.

The Transformer Model is a framework for making **sequence to sequence** predictions, using an **encoder-decoder** architecture with **self-attention** implemented within itself to capture context in short and long distances.

The encoder maps an input sequence into an abstract continuous representation that holds all the learned information of that input. The decoder then takes that continuous representation and generates single output tokens step-by-step feeding in the previous generated token.

1.2 Attention!!!

The attention function is a mapping of a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

$$q_i = W^q e_i, k_i = W^k e_i, v_i = W^v e_i$$

Attention scores can be calculated as example; $a_{1,i} = \frac{q_1 \cdot k_i}{\sqrt{d_k}}$. where dimension d_k is scaled to its square root because the dot product can grow quite large. This is not the whole sequence however, we still need to apply the Softmax() function to it to scale the scores as probabilities from 0 to 1. The output of this function is then multiplied by the value $v_i \in V$ vector to get the output vector. The higher softmax scores keep the value of the words the model learns as more important and words with lower scores get drowned out as irrelevant. The output is then fed into a linear layer to process.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Positional Encoding (PE)

In the original "attention is all you need" paper, the transformer uses **fixed sinusoidal vectors** added to token embeddings to allow the model to **infer relative positions** and extrapolate to longer sequences.

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}),$$
$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}})$$

The final input embeddings are the concatenation of the learnable embedding and the positional encoding.

Multi-Head Attention

If we take;

$$\mathbf{c}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Then applying the Scaled Dot-Product Attention multiple times on linearly transformed inputs is;

$$\text{Multi-Head}(Q, K, V) = \text{concat}(\mathbf{c}_1, \ldots, \mathbf{c}_h)W^O.$$

Residual Connections

The multi-headed attention output vector is added to the original positional input embedding, in what is called a residual connection. The output of this residual connection goes through a layer normalization.

Layer Norm

An operation for normalizing the values in each column of the matrix separately. This is to improve the stability of the model during training, by making

the average value in each column equal to 0 and the standard deviation equal to 1.

Multi-Layer Perceptron

The normalized residual output gets projected through a pointwise feed-forward network for further processing which consists of linear layers with a ReLU activation in between. The output of that is then fed to the input of the pointwise feed-forward network and further normalized.

Encoder

Is the input to a continuous representation with attention information which helps the decoder focus on the appropriate words in the input during the decoding process. This can be stacked however many times to further encode information where each layer has the opportunity to learn different attention representations.

Decoder

The decoder on the other hand generates text sequences and as such has two multi-headed attention layers, a pointwise feed-forward layer, and residual connects and layer normalizations after each sub-layer. These sub-layers behave similarly to those in the encoder but each multi-headed attention layer has a different job. The decoder is capped with a linear layer that acts as the classifier with a softmax to get the word probabilities.

Masking

The first layer of the decoder operates different to the rest, to prevent learning from **future** tokens (autoregression), masking is applied before calculating the softmax such that each query token has only access to key tokens that are positioned before it, not after. The mask of -inf essentially leaves the matrix with zeroes after applying the `softmax()` function.

※ Lecture 2

2.1 Positional Encoding Cont.

RoPE

Another method for PE is using Rotary Position Embeddings. By encoding **absolute position** with a rotation matrix, the positional encoding multiplies **Key** and **Value** matrices of **every attention layer**. Given rotation matrix R as;

$$R_{\theta_{p,k}} = \begin{bmatrix} \cos\theta_{p,k} & -\sin\theta_{p,k} \\ \sin\theta_{p,k} & \cos\theta_{p,k} \end{bmatrix}$$

where p is the token's absolute position and $k = 0, \ldots, \frac{d}{2} - 1$ indexes each 2D sub-space, and d is the model/attention dimension.

The positional encoding is performed as;

$$\mathbf{q}'_p = \mathbf{q}_p R_p, \qquad \mathbf{k}'_p = \mathbf{k}_p R_p, \qquad \mathbf{v}'_p = \mathbf{v}_p R_p$$

element-wise for each 2-D pair $(2k, 2k+1)$ as;

$$q'_{p,2k} = q_{p,2k}\cos\theta_{p,k} - q_{p,2k+1}\sin\theta_{p,k},$$
$$q'_{p,2k+1} = q_{p,2k}\sin\theta_{p,k} + q_{p,2k+1}\cos\theta_{p,k},$$

Thus RoPE injects position by rotating each 2D sub-vector through an angle that grows linearly with the absolute position while remaining distance-equivariant. Attention scores depend only on the relative offset $p_i - p_j$, which helps generalisation for very long sequences.

ALiBi

Another PE technique, the core idea is that for each **a**ttention head h, add a monotonic **li**near **bi**as to the raw dot-product score between Query position i and Key position j;

$$\text{score}^{(h)}_{i,j} = Q^{(h)}_i \cdot K^{(h)}_i - m_h|i-j|,$$

where $m_h \geq 0$ is a head-specific slope (larger for lower heads, smaller for higher heads). And, $|i-j|$ is the relative distance in timesteps/tokens.

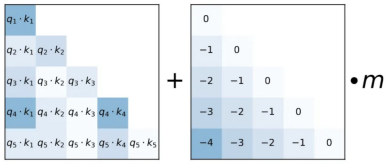


Figure 2: Diagram of matrix score calculation.

※ Lecture 3

3.1 Large Language Models

These are deep learning models primarily based on the transformer architecture. They are trained on vast amounts of text data to predict subsequent words or tokens, capturing complex patterns of language.

Architecture

Transformers use self-attention mechanisms to process input sequences **simultaneously** rather than sequentially.

Notable Models

- **GPT Series** - Known for powerful generative abilities and extensive fine-tuning capabilities (GPT-3; 175B parameters)
- **LLaMA 1** - trained on publicly available data, and more efficient with much more compact model at 13B parameters
- **LLaMA 1** - with an extended context length (4096 tokens), introduces group-query attention (GQA), RMSNorm normalization, SwiGLU activation, RoPE positional encoding.

3.2 Mixture of Experts

Or MoE, enhances transformer architectures by integrating specialized neural network modules ("experts") selectively engaged based on input data, increasing efficiency and capability.

Concept

Expert modules are specialized subnetworks activated selectively for processing different data subsets. The router network assigns inputs dynamically to appropriate experts based on relevance.

E.g., Mixtral 8x7B

Sparse MoE model with 8 feedforward expert modules per layer. Its tokens are dynamically routed through 2 experts per layer, effectively leveraging 47B parameters while activating only 13B per inference pass. Mixtral excels in benchmarking areas like math, code and multilingual tasks.

3.3 Parameter-Efficient Fine Tuning

PEFT techniques enable efficient adaptation of large pretrained models to various downstream applications. By only fine-tuning a small number of (extra) model parameters instead of all the model's parameters.

LoRA (Low-Rank Adaptation)

Introduces low-rank decomposition matrices into model layers. Adjustments restricted to these low-rank matrices (A and B) significantly reduce computational demands.

Prompt Tuning

describes freezing the model weights and fine-tuning small trainable vectors ("soft prompts") prepended to the input sequences. Particularly effective when minimal updates are required, leveraging pretrained capabilities efficiently.

3.4 Longer Contexts in Transformers

Standard transformers suffer from fixed, limited context lengths. Longer-context transformers solve these constraints, allowing models to capture dependencies over extended text sequences.

Transformer-XL

Conventional transformers operate inside a fixed window L. Every new window is processed from scratch so **tokens near the front** of a window cannot attend to tokens just before it → fragmenting long-range dependencies. Further, evaluation **latency is quadratic** in L for every step because **past keys/values** are **recomputed**.

Transformer-XL divides a long stream into successive segments x_t of length L and stores the hidden states of each segment as an external memory M^l_{t-1} for the next segment, maintaining these hidden states from previous segments. This also preserves the parallel computation benefits of self-attention within each segment.

For layer l at segment t:

$$\tilde{H}^{\ell}_{t-1} = \text{stop\_grad}\left(H^{\ell}_{t-1}\right); \qquad H^{\ell}_{t} = \text{TransformerBlock}^{\ell}([\tilde{H}^{\ell}_{t-1}; H^{\ell-1}_{t}])$$

Recurrence allows Transformer-XL to access information from much longer contexts compared to standard transformers, **extending their context lengths**.

Only the **keys** and **values** of the self-attention receive the memory. Queries

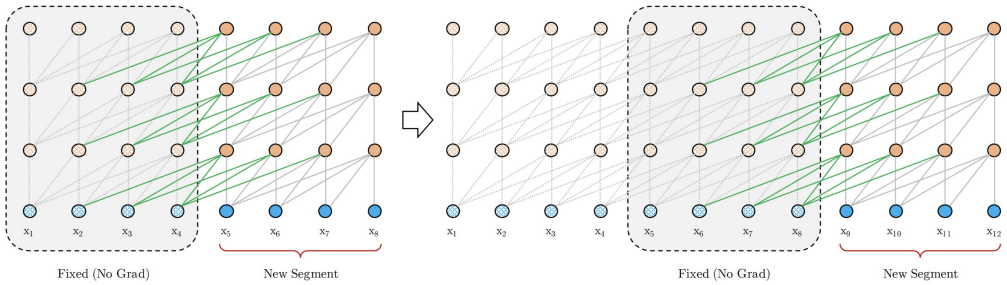


Figure 3: Segment recurrence in Transformer-XL architecture.

come solely from the current segment. Computational cost is $O(L(L+M))$ where M is the memory size. However, M is reused and so inference time per token falls by 180x on WikiText-103 compared with a naive sliding window.

The **relative positional encoding** allows it to encode the relative distance between positions (e.g., token positions i and j; relative distance $(i-j)$) rather than their absolute positions (as in a standard Transformer). Allowing Transformer-XL to generalise better across different sequence lengths and positions.

3.5 Parametric-Efficient Fine-Tuning

PEFT techniques address computational constraints by minimally adjusting or freezing most parameters of pretrained models during fine-tuning, updating only small subsets of parameters.

Low-Rank Adaptation

LoRA introduces low-rank decomposition matrixes into model layers. Adjustments to the model are restricted to these low-rank matrices (A and B), significantly reducing computational demands.

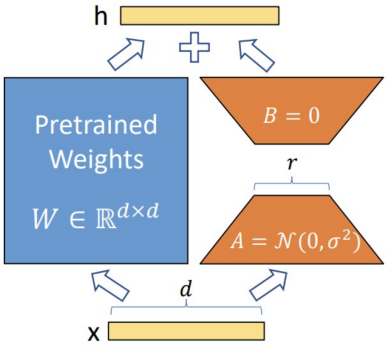


Figure 4: LoRA PEFT method.

ReLoRA

ReLoRA uses sequential low-rank updates with periodic (optimiser) resets to approximate full high-rank training. Starts with full-rank training but then switches to low-rank updates with optimizer resets and a jagged learning rate schedule (Fig 5) for stability.

Outperforms LoRA at larger scales, though gains in fine-tuning are less pronounced.

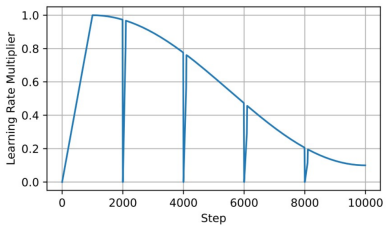


Figure 5: LoRA PEFT method.

Prompt Tuning

Another type of PEFT approach for transformers, it works by inserting a soft-prompt in a frozen transformer, prepended to the input sequence for fine-tuning.

3.6 Normalization

Activation functions propogating through very deep nets can drift toward saturated ends of non-linearities. This slows learning and causes vanishing and exploding gradients.

Normalization **rescales/recenters** intermediate representations so every layer receives a more stable distribution, enabling faster convergence, regularisation and smoother loss landscapes.

Batch-Normalization Normalizes each feature within a batch. During training, each batch computes the mean and variance for normalization. During inferencing, the test feature will be normalized by the mean and variance from the training set.

For each feature $x_{b,c}$ (for sample b, channel c), The forward pass is;

$$\mu_c = \frac{1}{B}\sum_b x_{b,c}, \qquad \sigma_c = \frac{1}{B}\sum_b (x_{b,c} - \mu_c)^2 \hat{x}_{b,c} = \frac{x_{b,c} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} y_{b,c} = \gamma_c \hat{x}_{b,c} + \beta_c \gamma, \beta$$

are learnable scale/shift parameters.

Using running estimates of μ_c, σ_c^2 collected during training for **inferencing**. With small batches, performance degrades between training and inference distributions.

Layer-Normalization

Normalizes **all features for each sample**, operating within a single example across all hidden units of a layer. The forward pass for each hidden vector $\mathbf{h} \in \mathbb{R}^d$:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} h_i, \qquad \sigma^2 = \frac{1}{d}\sum_{i}(h_i - \mu)^2 \hat{h}_i = \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \qquad y_i = \gamma_i \hat{h}_i + \beta_i \gamma, \beta \in \mathbb{R}^d$$

Inferencing is identical to training so no running statistics are needed. It is invariant to the sequence length making it crucial for the transformer encoder/decoders, as self-attention mixes time positions, so per-sample statistics are more stable. LN has slightly higher computational cost than BN in CNNs and offer less regularisation benefit because of no batch noise.

※ Lecture 4

4.1 Vision Transformers

In comparison to CNNs, ViTs create more similar representations in shallow and deep layers. They obtain the global representation from the shallow layers, but the local representation obtained from the shallow layers is also important.

Therefore, skip connections in ViT become even more influential than in CNNs and substantially impact the performance and similarity of representations. ViTs appear to retain more spatial information and can learn high-quality intermediate representations with large amounts of data.

ViTs vs CNNs

Transformers need lots of data for high accuracy. If the size of the dataset is small, CNNs generally perform better than transformers. Training transformers takes less time than CNNs, in terms of computational efficiency and accuracy, transformers can be chosen if time for model training is limited.

Self-attention brings more awareness to the developed model as attention maps can be visualized which may help developers guide how to improve the model.

Self-Supervision

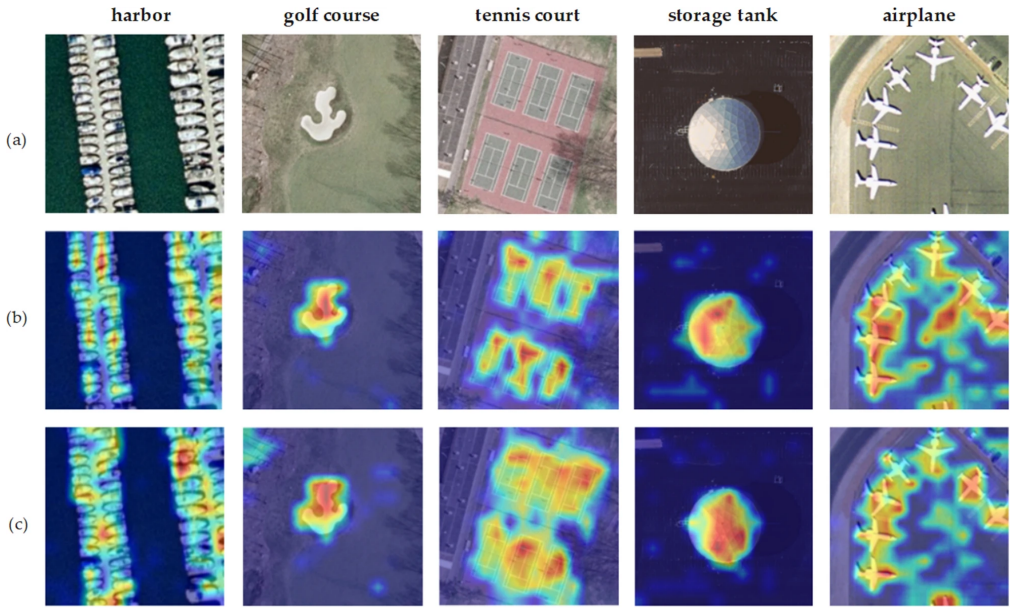


Figure 6: Example attention map for various scene images.

Distillation with **No** Labels (DINO) is a self-supervised learning algorithm that made ViTs competitive (and largely superior to) supervised CNNs.

Simply, Two ViT networks (a Student and Teacher) receive different augmentations of the same image. The student is trained to match the teacher's soft probability distribution over patch tokens; no ground-truth labels are used.

The teacher's weights are an exponential-moving average (EMA) of the student (a momentum encoder), providing a slowly evolving target that prevents collapse. This "self-distillation" objective avoids the heavy negative-pair sampling used by contrastive methods while still encouraging view-invariant yet information-rich embeddings.

iBOT

Image **B**ERT Pre-Training with **O**nline **T**okenizer; Reformulates masked-image modelling (MIM) pre-training as knowledge distillation (KD). It learns to distill knowledge from the tokenizer and performs self-distillation for MIM with help of twin teacher as an online tokenizer.

The goal is to let the target network recover each masked patch token to its corresponding tokenizer output.

CLIP

Learned representations are formulated using 3 key features; a ViT, a contrastive objective, and scale. During contrastive pre-training, CLIP learns to associate each image in a batch with its text companion, while dissociating it

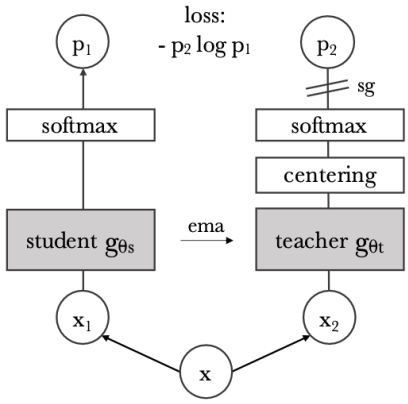


Figure 7: High level overview of the DINO architecture.

(a) Contrastive Pre-Training.

(b) Zero-shot prediction.

Figure 8: MAE reconstruction example.

from the other text snippets.

To use CLIP for specific **classification** tasks, prompts are required where labels of the task's dataset are reformulated to resemble the pre-training set while communicating the underlying context of the task. Then it predicts, among all the encoded prompts, the one which has **minimal contrastive loss** with the encoded image.

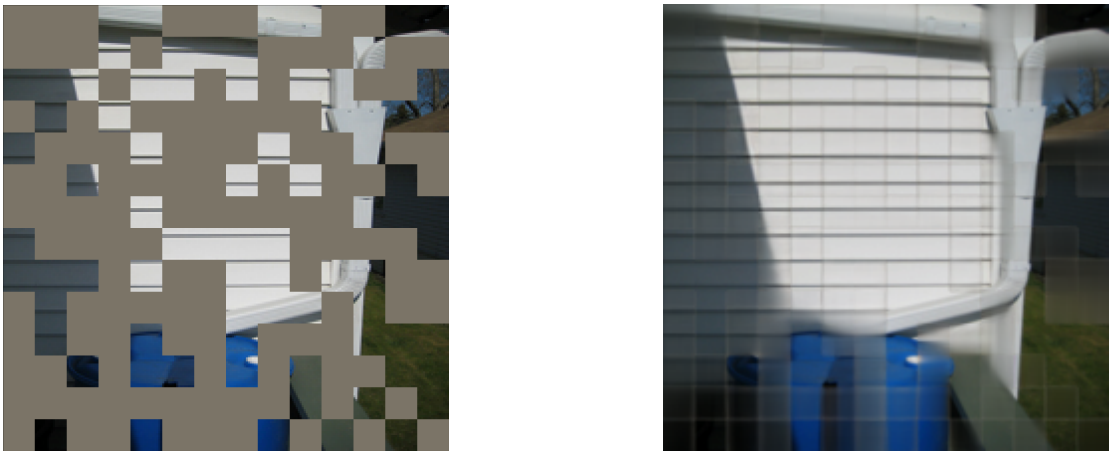
4.2 Autoencoders

An important category of unsupervised machine learning algorithms. Primarily designed to learn efficient encodings of data. They accomplish this by compressing (encoding) the input data into a reduced dimensional representation (in latent space), and attempt reconstructing (decoding) the original data from this compressed representation.

Variational Autoencoders

VAEs introduce probabilistic elements to the AE structure, the encoder of VAE typically models a multivariate Gaussian distribution with diagonal covariance, and a standard multivariate Gaussian often serves as the prior. VAEs can also generate new data points by sampling from the learned distribution.

Masked Autoencoders



(a) Masked Image.

(b) Reconstruction of (a).

Figure 9: MAE reconstruction example.

MAEs are effective in vision-based tasks, often involving a specialized training mechanism. During training, a significant portion (e.g., 75%) of the image patches are randomly masked. Then the encoder processes only the visible (unmasked) patches. Mask tokens are inserted after encoding and a small decoder reconstructs the entire image including the masked parts. After training, the decoder is discarded and the encoder alone is used for image reconstruction.

MAE encoder employs ViT and the decoder involves additional transformer blocks. This combination proves highly scalable and generalizable.

Encoder-Decoder?

the encoder-decoder architecture is similar to autoencoder conceptually but differ significantly. Autoencoders aim to reconstruct the **same** input data after compression while encoder-decoder models often use data from **different sources** for the input and the output.

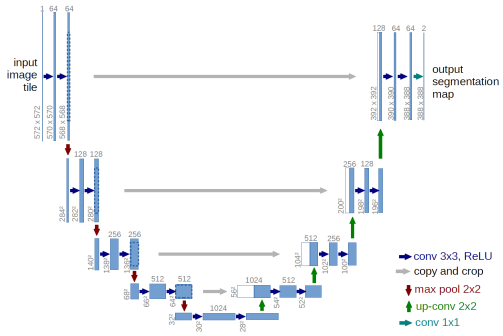


Figure 10: U-Net Architecture.

4.3 U-Net

U-Net is a CNN architecture specifically designed for biomedical image segmentation tasks but is now widely applied across various domains involving pixel-wise classification.

Consisting of two main parts; contracting (Encoder) and expansive (Decoder) paths, both these paths are symmetric, forming the U-shape structure.

The contracting path follows the typical architecture of a convolutional network;

- It consists of the repeated application of two 3×3 (unpadded) convolutions, each followed by a ReLU block and a 2×2 max pool layer with stride 2 for downsampling.
- At each downsampling step it doubles the number of feature channels.

Every step in the expansive path consists of an upsampling of the feature map followed by a 2×2 ("up-)convolution(") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path. And two 3×3 convolutions, each followed by a ReLU block. Cropping is necessary as there is loss of border pixels every convolution.

※ Lecture 5

First, defining the concept of the **latent variable** denoted by random variable z. For many modalities, we can think of the data we observe as represented or generated by this latent variable z.

Mathematically it can be modeled by the joint distribution of $p(x, z)$.

One approach of generative modeling ("likelihood-based") is to learn a model to maximize the likelihood $p(x)$ of all observed x. This likelihood of purely observed data $p(x)$ can be given by;

$$p(x) = \frac{p(x,z)}{p(z|x)}$$

and this evidence is quantified in this case as the log likelihood of the observed data; $\log p(x)$.

5.1 Evidence Lower-Bound

ELBO is the worst-case for the log-likelihood of the distribution $p(x)$. Below, $\mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big]$ is called the evidence for x, and $D_{\mathrm{KL}}\big(q_\phi(z \mid x) \,\|\, p(z)\big)$ is the KL-divergence between q_ϕ and p_θ.

$$\begin{aligned}
\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x \mid z)\, p(z)}{q_\phi(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] \;+\; \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(z)}{q_\phi(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big] \;-\; D_{\mathrm{KL}}\big(q_\phi(z \mid x) \,\|\, p(z)\big).
\end{aligned}$$

The last line can become a **maximising loss function** such that we always try to push the evidence for x closer to 1, while minimizing the KL divergence toward 0.

The encoder in VAE commonly models a multivariate gaussian with diagonal covariance, the prior is often selected to be a standard multivariate gaussian. The diffusion model is very similar to a hierarchical variational autoencoder (HVAE).

5.2 Diffusion

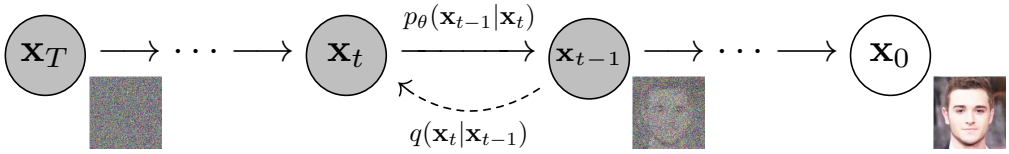


Figure 11: Directed graphical model of the Diffusion model.

Diffusion describes a generative process that generates samples from gaussian noises through a process called denoising. This is described as the reverse

process, the inverse being the forward process which is the gradual **addition** of noise to images as;

$$q(x_t \mid x_{t-1}) := \mathcal{N}\big(x_t;\, \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t I\big)$$

$q(x_0)$ being the distribution of x_0 which generates the sequence of samples $\{x_t\}_{t=1}^{T}$. $[\beta_1, \ldots, \beta_T]$ denotes a variance schedule for the addition of noise and T is a hyperparameter denoting the total number of time steps.

The forward process samples x_t at an arbitrary timestep t as given by

$$q(x_t \mid x_0) = \mathcal{N}\big(x_t;\, \sqrt{\alpha_t}\, x_0,\, (1-\alpha_t)I\big)$$

where $\alpha_t = \Pi_{i=1}^{t}(1-\beta_i)$, similarly the reverse process $p(x_{t-1}|x_t)$ is approximated as

$$p_\theta(x_{t-1} \mid x_t) := \mathcal{N}\big(x_{t-1};\, \mu_\theta(x_t, t),\, \tilde{\beta}_t I\big)$$

where $\tilde{\beta}_t = \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t$, and θ represents the learnable parameters. Here the reverse process may be **reformulated** as

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\,\epsilon_\theta(x_t, t)\right)$$

where ϵ_θ is the noise estimation network. The **objective (loss) function** of diffusion can be given by reformulating the forward process;

$$\mathcal{L}_\theta(x_0, t) = \mathbb{E}_{x_0,\epsilon,t}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\alpha_t}\, x_0 + \sqrt{1-\alpha_t}\,\epsilon,\, t\right)\right\|^2\right]$$

Algorithms

Seperate to the Training and Sampling of the model (Algorithms 12a, 12b), diffusion is divided into two parts: forward and reverse diffusion. Forward diffusion is done using the closed-form formula while reverse diffusion is usually the goal, to be used with a trained neural network (likely U-Net).

Approximating the denoising q, we need to approximate noise ϵ_t using the neural network ϵ_θ

※ Lecture 6

6.1 Diffusion Cont.

Stable Diffusion

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}\{1, \dots, T\}$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take a gradient-descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon},\ t\right)\right\|^2$
6: **until** converged

(a) Training

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ **if** $t > 1$ **else** $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\,\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

(b) Sampling

Aims to accelerate the diffusion process by conducting it within the latent space. Known as Latent Diffusion Model (LDM), it performs faster than standard diffusion.

It works by training an autoencoder to learn to compress the image data into lower-dimensional representations. Then, using the trained encode E, encode the full image into lower dimensional latent data. And using the decoder D, decode the latent data back into an image.

After encoding the images into latent data, forward and reverse diffusion processes are conducted in this latent space.

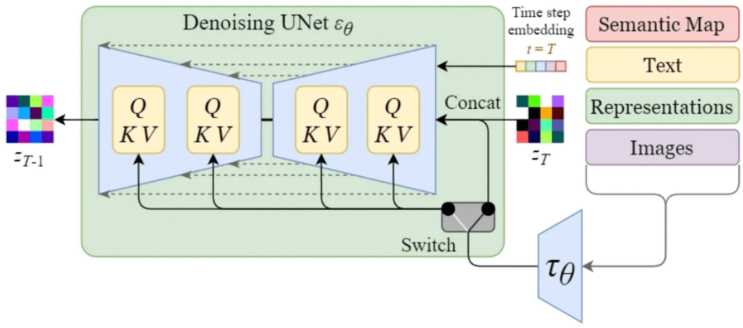


Figure 12: Conditioning mechanism within Stable Diffusion's U-Net.

The real powerful aspect about Stable Diffusion is its ability to generate images with text prompts by modifying the inner diffusion model to accept conditioning inputs. By augmenting denoising U-Net with cross-attention, the model is turned into a conditional image generator.

The switch in Diagram 12 controls the different types of conditioning inputs;

- **Text inputs** - first converted into embeddings (vectors) using a language model τ_θ, then mapped into the U-Net via the multi-head attention layer.

- **Other spatial inputs** - (e.g., semantic maps, images, inpainting) the conditioning can be done using concatenations.

Training is similar to normal diffusion, the loss function uses latent data z_t instead of image x_t, and the conditioning input $\tau_\theta(y)$ is added to the U-Net.

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{t,z_0,\varepsilon,y}\left[\left\|\varepsilon - \varepsilon_\theta(z_t, t, \tau_\theta(y))\right\|^2\right]$$

6.2 GANs

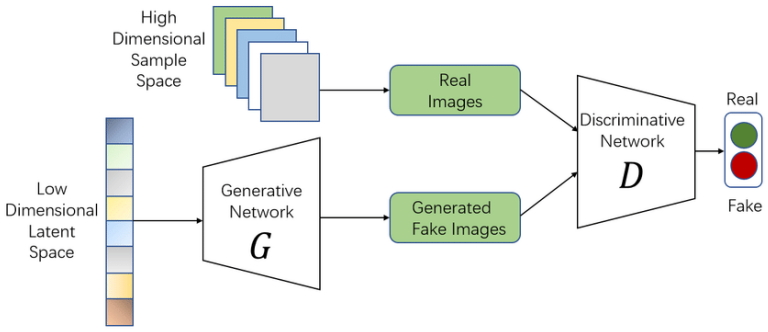


Figure 13: The most basic form of the GAN architecture.

Generative Adversarial Networks - The big picture for this class of models is generating data from scratch. Mainly focused on images, there is growing application in other domains.

Given a set of training data, GANs learn to estimate the underlying probability distribution of the data. Then generate samples from the learnt probability distribution that may not be present in the original training set.

The generator $G(x)$ takes a random vector as input and tries to produce images similar to the training set x. The discriminator network $D(x)$ is a binary classifier that tries to distinguish between the real images according to the training set x and the fake images as produced by $G(x)$.

With this discriminator, the model is able to use it as a loss function to guide the generator to create images close to indistinguishable to the real training samples.

Training

This means that both networks are trained in alternating steps and are locked in a competition to improve themselves. The discriminator ideally eventually identifies tiny differences between real and generated samples, and the generator

creates images that the discriminator cannot tell the difference between. The GAN model should then converge and produce natural looking images.

Backpropagation The output of the discriminator $D(x)$ is the probability of x being a real image. The goal is to maximise the change of recognising real and fake images.

On the generator $G(x)$, its objective function should model the generation of samples with the highest value of $D(x)$ to fool the discriminator. Once both objectives are defined, they are learned jointly by altering gradient descent to fix the generator model's parameters with the output of the discriminator per iteration of gradient descent. The same is true vice-versa with the discriminator. Fixing the discriminator and training the generator for a single iteration.

6.3 CycleGAN

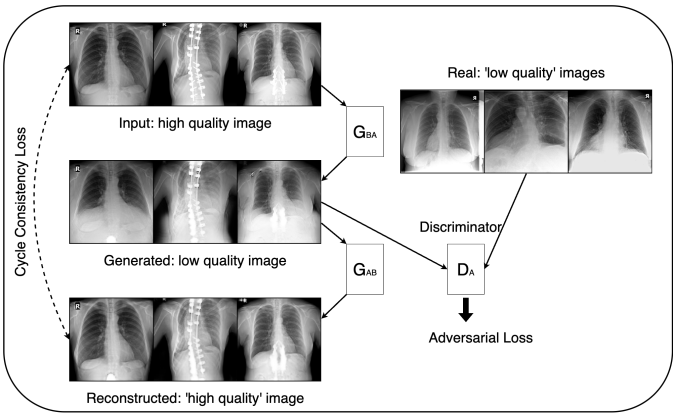


Figure 14: CycleGAN Architecture

A variant of GANs originally proposed for unpaired image-to-image translation. Its overall architecture (stolen from the group coursework) is shown in Figure 14.

Simply put, generator G_{BA} maps sample images B to A. and G_{AB} maps the inverse of this. The discriminator functions the same, trying to distinguish which images are real and fake.

$$\mathcal{L}_{\text{GAN}}(G_{\text{BA}}, D_{\text{A}}, B, A) = \mathbb{E}_{a\sim p_{\text{data}}(a)}\big[\log D_{\text{A}}(a)\big] + \mathbb{E}_{b\sim p_{\text{data}}(b)}\big[\log\big(1 - D_{\text{A}}\big(G_{\text{BA}}(b)\big)\big)\big]$$

The real innovation of CycleGAN is the cycle consistency loss which describes the absolute error between the input and reconstructed images. With the objective function;

$$\mathcal{L}_{\text{cyc}}(G_{\text{BA}}, G_{\text{AB}}) = \mathbb{E}_{b \sim p_{\text{data}}(b)}\left[\| G_{\text{AB}}\big(G_{\text{BA}}(b)\big) - b\|_1\right] + \mathbb{E}_{a \sim p_{\text{data}}(y)}\left[\| G_{\text{BA}}\big(G_{\text{AB}}(a)\big) - a\|_1\right]$$

This objective ensures the image translation cycle should be able to bring the generated image back to the original which is called forward cycle consistency.

The full combined objective is therefore;

$$\begin{aligned}\mathcal{L}(G_{\text{BA}}, G_{\text{AB}}, D_{\text{B}}, D_{\text{A}}) = \mathcal{L}_{\text{GAN}}(G_{\text{BA}}, D_{\text{A}}, B, A) \\ + \mathcal{L}_{\text{GAN}}(G_{\text{AB}}, D_{\text{B}}, A, B) \\ + \lambda \mathcal{L}_{\text{cyc}}(G_{\text{BA}}, G_{\text{AB}})\end{aligned}$$

where λ controls the relative importance of either objectives. The optimal solution can be given by solving for;

$$G^*_{\text{BA}},\ G^*_{\text{AB}} = \arg \min_{G_{\text{BA}}, G_{\text{AB}}} \max_{D_{\text{B}}, D_{\text{A}}} \mathcal{L}(G_{\text{BA}}, G_{\text{AB}}, D_{\text{B}}, D_{\text{A}})$$