Tutorial 10

Gidon Rosalki

2025-12-31

1 Fine tuning

Training a model from scratch has a few challenges. It takes an incredibly long time, since improvement is a slow process, and there are many weights (millions) to train. It needs as a result huge amounts of electricity, and data, which can be expensive, and hard to attain accordingly. The idea of fine tuning is to take a model that has already been trained on a task, and adjust it to our intended task. This way, we need far less data, and time to train.

For example, we could start with a model trained on a large dataset (such as ImageNet or MNIST), keep the learned features (edges, shapes, textures), and then train it with a smaller dataset. We need to be careful to only train it a little, in order to avoid overfitting.

2 Low Rank Adaption - LoRA

The key idea is to adapt big models with small changes. This is a method for fine tuning large models *efficiently*. Instead of updating all weights, LoRA adds small trainable matrices (which are low rank updates). The original model weights stay frozen, it simply learns a few extra parameters. This is much faster, and needs less memory than full fine tuning. The resultant model involves:

$$\text{Frozen: } W \in \mathbb{R}^{n\times m} \tag{1}$$

$$\text{Trained: } A \in \mathbb{R}^{r\times m},\ B \in \mathbb{R}^{n\times r} \tag{2}$$

$$r \text{ is usually very small, so 4 or 8: } r \ll n, m \tag{3}$$

$$h = (W + BA)\, x \tag{4}$$

3 Generative models

There are two main types of model. Discriminative (aimed at classification or regression) which attempts to learn $p(y|x)$, and maps inputs to labels / values. There are also **generative** models, which learn $p(x)$ or $p(x, y)$, where the idea is how to model, or *generate* new data.

3.1 Loss functions

For a model with the output of an image, rather than a single number or class label, we need to use different types of loss functions. Firstly we have L_1:

$$L_1(I_t, I_o) = ||I_t - I_o||_1 = \frac{1}{N}\sum_{x,y} |I_t[x, y] - I_o[x, y]| \tag{5}$$

I_t is the target image, and I_o is the model output image. This gives sharper, but possibly noisier results. We also have L_2:

$$L_1(I_t, I_o) = ||I_t - I_o||_2^2 = \frac{1}{N}\sum_{x,y} (I_t[x, y] - I_o[x, y])^2 \tag{6}$$

L_2 will give blurrier, but globally accurate results.

There is also perceptual loss, which is feature based. It measures the difference between images in a deep feature space (pre trained VGG / ResNet), encouraging the generated image to look perceptually similar to the target, rather than just matching the pixel values:

$$L_{VGG}(I_t, I_o) = ||\phi_l(I_t) - \phi_l(I_o)||_2^2$$

We may see the differences in the below example:

Figure 1: Loss differences

3.2 Unsupervised learning

Generally, we use something called unsupervised learning. We generate new samples from the same distribution of the training data. A generative model needs to be Sufficient: for every image x there must be $z : G(z) = x$, and compact, for every z, $G(z)$ should be a valid image in X. There are many ways to address this problem:

- Auto-regressive pixel generation
- VAE – variational autoencoders
- GAN – generative adversarial networks
- ViT – vision transformers
- Diffusion models

3.2.1 Auto-Regressive pixel generation

There exists PixelCNN, which uses all previously generated pixels to predict the next one, using auto-regressive convolutions. This creates nice results, but is very slow, since it only generates one pixel at a time:

$$p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

3.2.2 VAE - Variational autoencoders

An autoecnoder compresses the input to a low dimensional latent vector. This is done with unsupervised learning, adn the loss is

$$||x - \widehat{x}||^2$$

The latent vector can be the input to a supervised classification network.

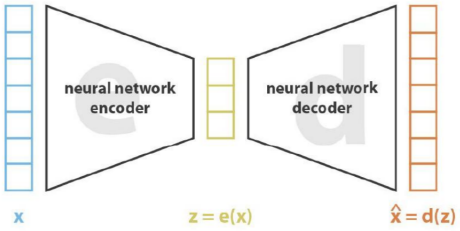

Figure 2: Autoencoders

So, for VAE we sample latent vectors to create new images. We need to enforce some "arrangement" on the latent space, so we encourage the latent vector to be normally distributed. Instead of learning the latent vector, we learn the mean standard deviation, and use them to sample from the normal distribution. The loss would now be both the reconstruction loss $||x - \widehat{x}||^2$, and the regularisation loss $KL[N(\mu_x, \sigma_x) \| N(0, 1)]$.

3.2.3 GAN - generative adversarial networks

This is a two player game:

- Generator network G: This generates images that will look real enough to fool D
- Discriminator network D: Distinguish between real images, and images produced by G

Both networks are trained together in a minmax game, the discriminator is trained to maximise the difference between real and fake images, and the generator is trained to minimise it:

$$\min_G \left\{ \max_D \{V(G, D)\} \right\} = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

So the discriminator is trained to maximise the difference between its output on real images (such that for image x, $D(x) \to 1$), to the output on generated images (such that for image $G(z)$ $D(G(z)) \to 0$). In contrast, G is trained to minimise it, such that $D(G(z)) \to 1$.
They are trained using an iterative approach. You begin by fixing the generator, and training the discriminator:

$$\max_D \left\{ \mathbb{E}_{x \sim p_{data}(x)} \left[\log D(x) \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right\}$$

Then you fix the discriminator, and train the generator:

$$\min_G \left\{ \mathbb{E}_{z \sim p_z(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right\}$$

Training GANs is very difficult, and unstab;e. It does not always converge, since parameters are unstable, so they will oscillate when training and may never converge. Additionally, they can undergo mode collapse, where the generator always produces the same kind of images. Finally, a strong discriminator causes small vanishing gradients for the generator. They are also very sensitive to choice of hyperparameters.

3.2.4 StyleGAN

We want better understanding of GANs and latent space. StyleGAN incorporates ideas from style transfer to separate high level attributes (pose, identity, etc.) from stochastic variation (freckles, hair, etc.) in generated images. This only modifies the generator. Instead of using z as the input the to generator, we first map it to an intermediate latent space W using a non-linear mapping $f : Z \to W$. w will be inserted into the generator as a style "hint", and noise inputs are also given to the generator to help it generate stochastic detail.

3.2.5 ViT - vision transformers

Let us begin by discussing CLIP. CLIP stands for Contrastive Language Image Pretraining. It's goal is to teach a model to understand which text matches which image. Instead of predicting models, CLIP learns to match the correct image, with the correct text, and separate mismatched pairs.

The **attention mechanism** was originally developed for NLP tasks. Since not all words in a sentence are equally important, attention lets the model focus on relevant words. Attention works by havign each word create a Query, Key, Value vector. It scores how much one word relates to another, and the weighted sum gives a context vector for each word. This enables the model to capture meaning across the whole sentence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} \right) V \tag{7}$$

$$\tag{8}$$

There is also *cross attention*, which is a mechanism where one set of features attends to another set. This lets information flow between two different sources.

So, ViT has us take an image, and separate it into patches. We then multiply each vectorised patch by a matrix to transfer us to some other space, these are passed through embeddings, and we have build a transformer encoder that classifies the patches.

3.2.6 Diffusion models

Diffusion models learn to invert a parameterised Markovian image noising process. This can generate high quality images. More simply, they take random noise, and slowly de-noise them to generate high quality images.

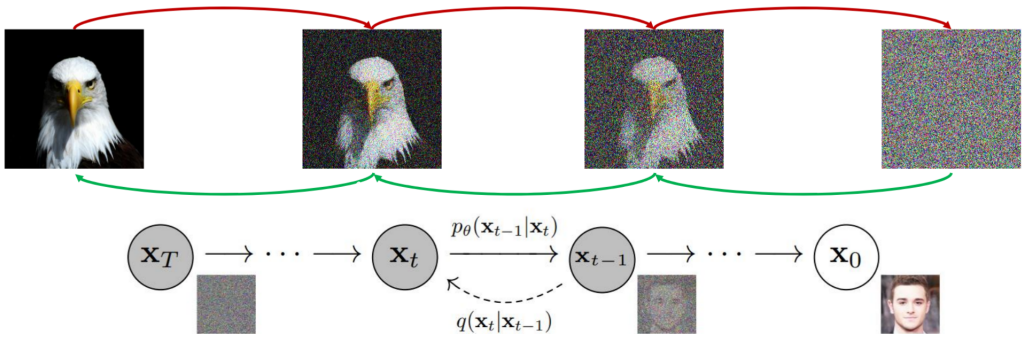

Figure 3: Denoising

In general, in generative models, we try to learn the probability distribution $p(x)$. So in diffusion models, what we learn is the "direction" each point should move in order to reach the true distribution.

There are a couple of problems:

1. The model is unconditioned, so it generates a random image without any prompt or guidance.
2. For large images, the generation process takes a long time.