In [1]:
```
from IPython.display import Image
```

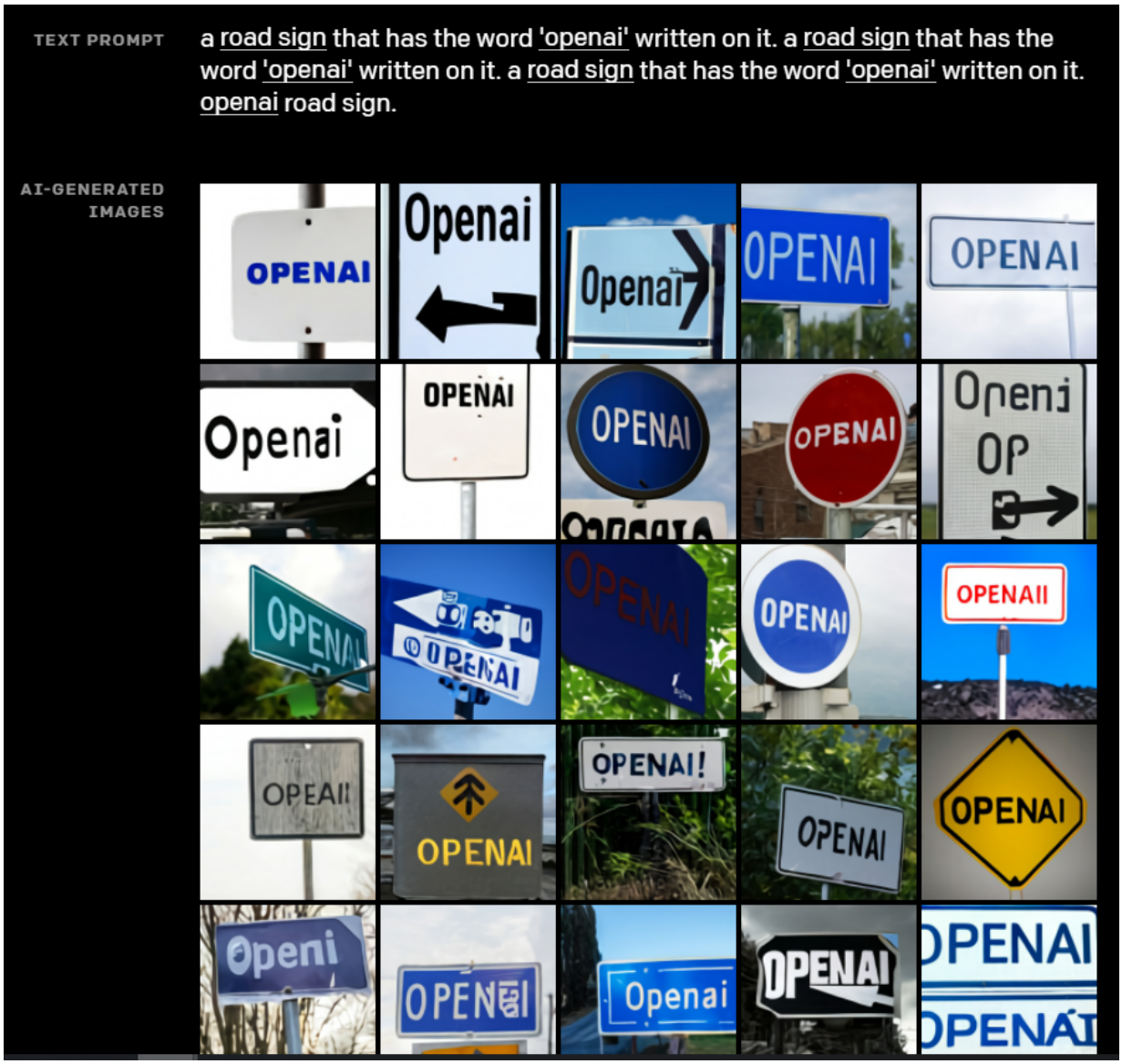
DALL. E

https://openai.com/blog/dall-e/

Zero-Shot Text-to-Image Generation

In [2]:
```
Image('img/01.png')
```

Out[2]:



1. Abstract

In [3]:
```
Image('img/Abstact.jpg')
```

Out[3]:

Abstract

Text-to-image generation has traditionally focused on finding better modeling assumptions for training on a fixed dataset. These assumptions might involve complex architectures, auxiliary losses, or side information such as object part labels or segmentation masks supplied during training. We describe a simple approach for this task based on a transformer that autoregressively models the text and image tokens as a single stream of data. With sufficient data and scale, our approach is competitive with previous domain-specific models when evaluated in a zero-shot fashion.

2. Method

In [14]:
```
Image('img/method01.jpg')
```

Out[14]:

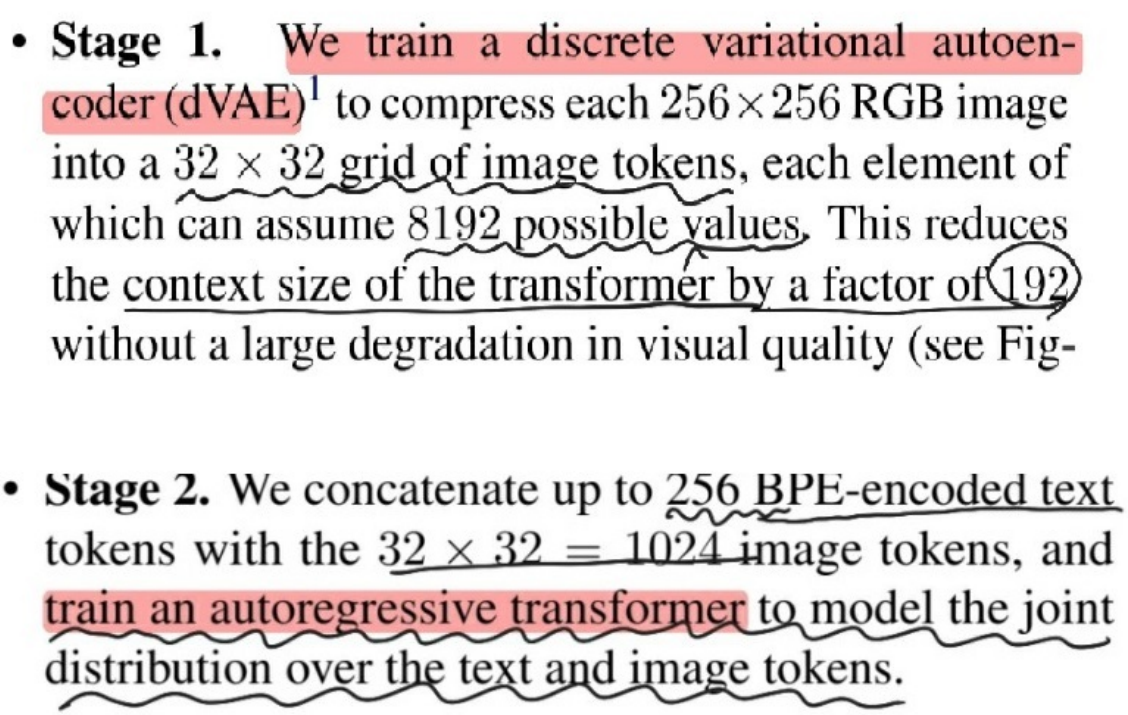
2. Method

Our goal is to train a transformer (Vaswani et al., 2017) to autoregressively model the text and image tokens as a single stream of data. However, using pixels directly as image tokens would require an inordinate amount of memory for high-resolution images. Likelihood objectives tend to prioritize modeling short-range dependencies between pixels (Salimans et al., 2017), so much of the modeling capacity would be spent capturing high-frequency details instead of the low-frequency structure that makes objects visually recognizable to us.

We address these issues by using a two-stage training procedure, similar to (Oord et al., 2017; Razavi et al., 2019):

In [5]:
```
Image('img/method02.jpg')
```

Out[5]:



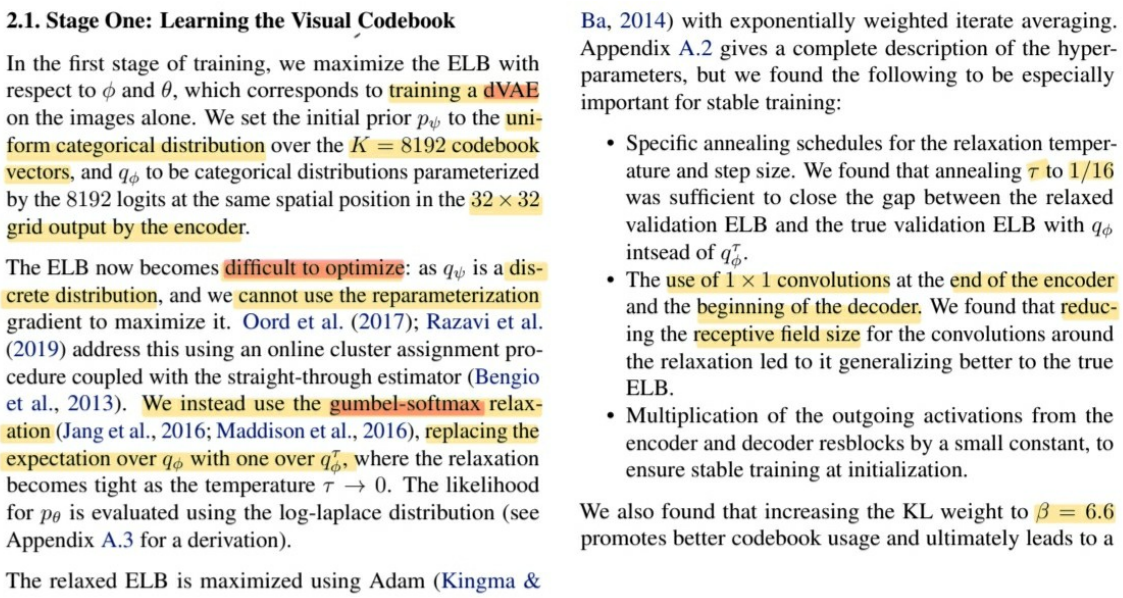
- **Stage 1.** We train a discrete variational autoencoder (dVAE)[1] to compress each 256×256 RGB image into a 32 × 32 grid of image tokens, each element of which can assume 8192 possible values. This reduces the context size of the transformer by a factor of 192 without a large degradation in visual quality (see Fig-

- **Stage 2.** We concatenate up to 256 BPE-encoded text tokens with the 32 × 32 = 1024 image tokens, and train an autoregressive transformer to model the joint distribution over the text and image tokens.

2.1 Stage One

In [6]:
```
Image('img/stage1.jpg')
```

Out[6]:



2.1. Stage One: Learning the Visual Codebook

In the first stage of training, we maximize the ELB with respect to ϕ and θ, which corresponds to training a dVAE on the images alone. We set the initial prior p_ψ to the uniform categorical distribution over the $K = 8192$ codebook vectors, and q_ϕ to be categorical distributions parameterized by the 8192 logits at the same spatial position in the 32×32 grid output by the encoder.

The ELB now becomes difficult to optimize: as q_ψ is a discrete distribution, and we cannot use the reparameterization gradient to maximize it. Oord et al. (2017); Razavi et al. (2019) address this using an online cluster assignment procedure coupled with the straight-through estimator (Bengio et al., 2013). We instead use the gumbel-softmax relaxation (Jang et al., 2016; Maddison et al., 2016), replacing the expectation over q_ϕ with one over q_ϕ^τ, where the relaxation becomes tight as the temperature $\tau \to 0$. The likelihood for p_θ is evaluated using the log-laplace distribution (see Appendix A.3 for a derivation).

The relaxed ELB is maximized using Adam (Kingma & Ba, 2014) with exponentially weighted iterate averaging. Appendix A.2 gives a complete description of the hyperparameters, but we found the following to be especially important for stable training:

- Specific annealing schedules for the relaxation temperature and step size. We found that annealing τ to $1/16$ was sufficient to close the gap between the relaxed validation ELB and the true validation ELB with q_ϕ intsead of q_ϕ^τ.
- The use of 1×1 convolutions at the end of the encoder and the beginning of the decoder. We found that reducing the receptive field size for the convolutions around the relaxation led to it generalizing better to the true ELB.
- Multiplication of the outgoing activations from the encoder and decoder resblocks by a small constant, to ensure stable training at initialization.

We also found that increasing the KL weight to $\beta = 6.6$ promotes better codebook usage and ultimately leads to a

- d VAE(VQ-VAE:Neural Discrete Representation Learning)

In [7]:
```
Image('img/vqvae.jpg')
```

Out[7]:

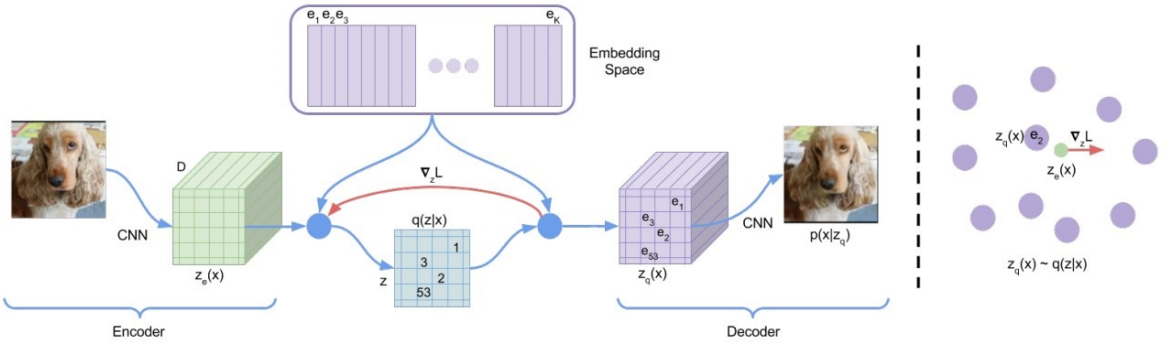


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

$$L = \log p(x|z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \text{sg}[e]\|_2^2,$$

- Gumbel softmax relaxation

(CATEGORICAL REPARAMETERIZATION WITH GUMBEL-SOFTMAX)

In [8]:
```
Image('img/gumbel.jpg')
```

Out[8]: The Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2014) provides a simple and efficient way to draw samples z from a categorical distribution with class probabilities π:

$$z = \text{one\_hot}\left(\arg\max_i [g_i + \log \pi_i]\right) \tag{1}$$

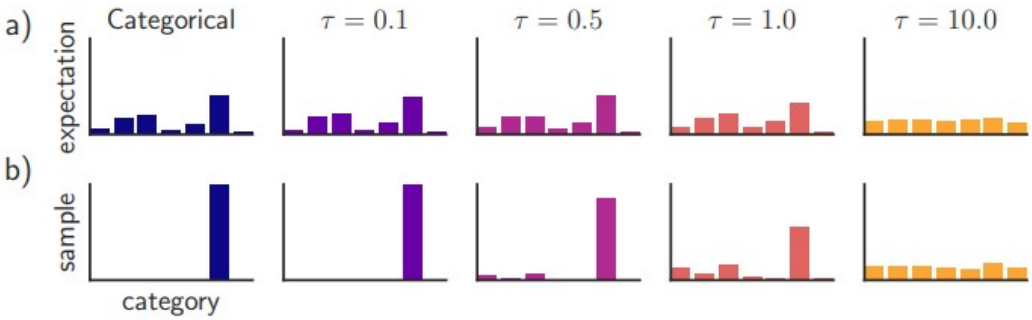
where $g_1 ... g_k$ are i.i.d samples drawn from Gumbel$(0, 1)$[1]. We use the softmax function as a continuous, differentiable approximation to $\arg\max$, and generate k-dimensional sample vectors $y \in \Delta^{k-1}$ where

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, ..., k. \tag{2}$$

The density of the Gumbel-Softmax distribution (derived in Appendix B) is:

$$p_{\pi,\tau}(y_1, ..., y_k) = \Gamma(k)\tau^{k-1}\left(\sum_{i=1}^{k} \pi_i / y_i^{\tau}\right)^{-k} \prod_{i=1}^{k} (\pi_i / y_i^{\tau+1}) \tag{3}$$

This distribution was independently discovered by Maddison et al. (2016), where it is referred to as the concrete distribution. As the softmax temperature τ approaches 0, samples from the Gumbel-Softmax distribution become one-hot and the Gumbel-Softmax distribution becomes identical to the categorical distribution $p(z)$.



In [9]:
```
Image('img/vqvae_result.jpg')
```

Out[9]:

Figure 1. Comparison of original images (top) and reconstructions from the discrete VAE (bottom). The encoder downsamples the spatial resolution by a factor of 8. While details (e.g., the texture of the cat's fur, the writing on the storefront, and the thin lines in the illustration) are sometimes lost or distorted, the main features of the image are still typically recognizable. We use a large vocabulary size of 8192 to mitigate the loss of information.

2.2 Stage Two

In [10]:
```
Image('img/stage2.jpg')
```

Out[10]:

2.2. Stage Two: Learning the Prior

In the second stage, we fix ϕ and θ, and learn the prior distribution over the text and image tokens by maximizing the ELB with respect to ψ. Here, p_ψ is represented by a 12-billion parameter sparse transformer (Child et al., 2019).

Given a text-image pair, we BPE-encode (Sennrich et al., 2015) the lowercased caption using at most 256 tokens[5] with vocabulary size 16,384, and encode the image using $32 \times 32 = 1024$ tokens with vocabulary size 8192. The image tokens are obtained using argmax sampling from the dVAE encoder logits, without adding any gumbel noise.[6] Finally, the text and image tokens are concatenated and modeled autoregressively as a single stream of data.

↳ GPT

The transformer is a decoder-only model in which each image token can attend to all text tokens in any one of its 64 self-attention layers. The full architecture is described in Appendix B.1. There are three different kinds of self-attention masks used in the model. The part of the attention masks corresponding to the text-to-text attention is the standard causal mask, and the part for the image-to-image attention uses either a row, column, or convolutional attention mask.[7]

↳ Sparse attention

We limit the length of a text caption to 256 tokens, though it is not totally clear what to do for the "padding" positions in between the last text token and the start-of-image token. One option is to set the logits for these tokens to $-\infty$ in the self-attention operations. Instead, we opt to learn a special padding token separately for each of the 256 text positions. This token is used only when no text token is available. In preliminary experiments on Conceptual Captions (Sharma et al., 2018), we found that this resulted in higher validation loss, but better performance on out-of-distribution captions.

We normalize the cross-entropy losses for the text and image

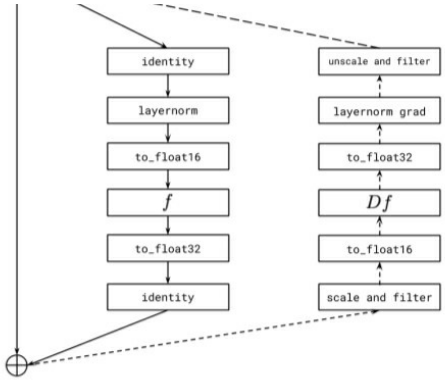


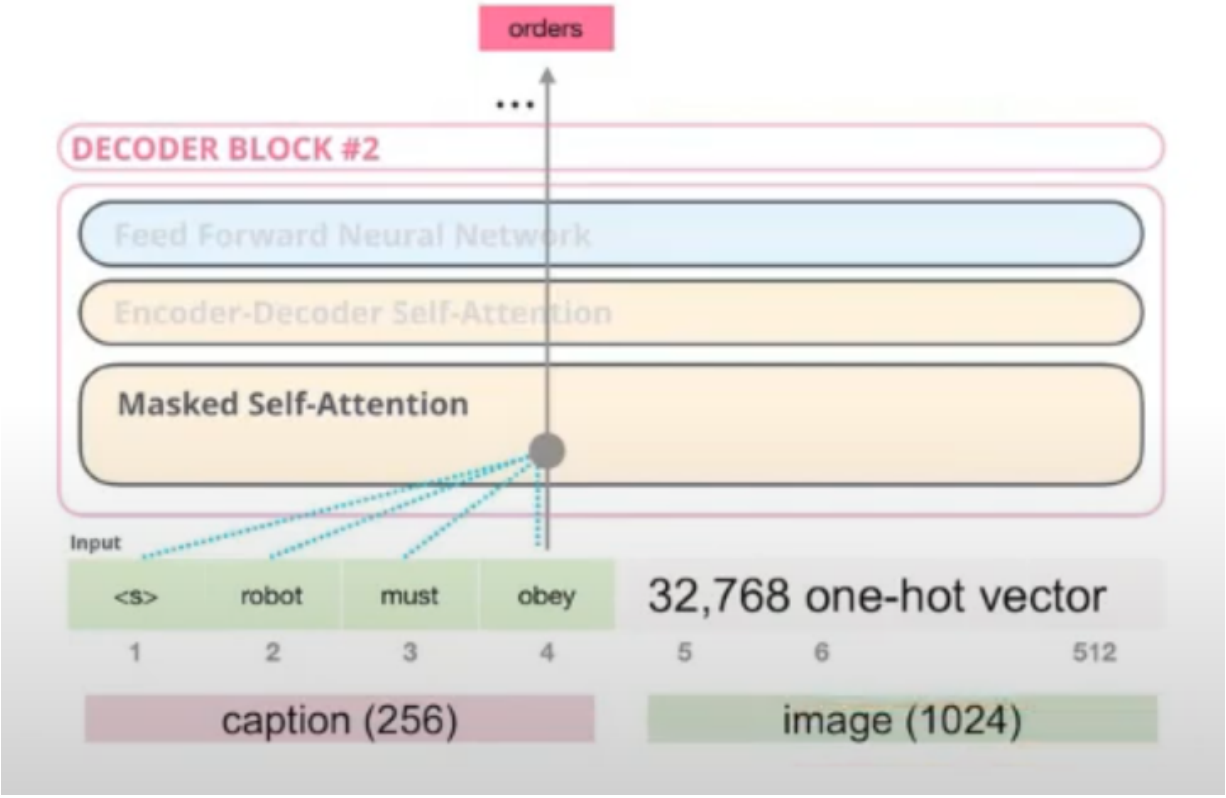
Figure 4. Illustration of per-resblock gradient scaling for a transformer resblock. The solid line indicates the sequence of operations for forward propagation, and the dashed line the sequence of operations for backpropagation. We scale the incoming gradient for each resblock by its gradient scale, and unscale the outgoing gradient before it is added to the sum of the gradients from the successive resblocks. The activations and gradients along the identity path are stored in 32-bit precision. The "filter" operation sets all Inf and NaN values in the activation gradient to zero. Without this, a nonfinite event in the current resblock would cause the gradient scales for all preceding resblocks to unnecessarily drop, thereby resulting in underflow.

tokens by the total number of each kind in a batch of data. Since we are primarily interested in image modeling, we multiply the cross-entropy loss for the text by 1/8 and the cross-entropy loss for the image by 7/8. The objective is optimized using Adam with exponentially weighted iterate averaging; Appendix B.2 describes the training procedure in more detail. We reserved about 606,000 images for validation, and found no signs of overfitting at convergence.

GPT-3

In [11]:
```
Image('img/gpt3.png')
```

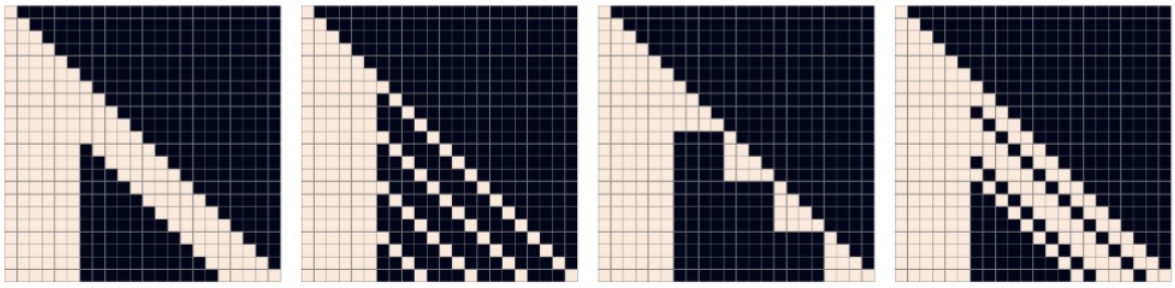
Out[11]:



Sparse transformer (sparse attention mask)

In [12]:
```
Image('img/sparseattentionmask.jpg')
```

Out[12]:



(a) Row attention mask. (b) Column attention mask. (c) Column attention mask with transposed image states. (d) Convolutional attention mask.

Figure 11. Illustration of the three types of attention masks for a hypothetical version of our transformer with a maximum text length of 6 tokens and image length of 16 tokens (i.e., corresponding to a 4×4 grid). Mask (a) corresponds to row attention in which each image token attends to the previous 5 image tokens in raster order. The extent is chosen to be 5, so that the last token being attended to is the one in the same column of the previous row. To obtain better GPU utilization, we transpose the row and column dimensions of the image states when applying column attention, so that we can use mask (c) instead of mask (b). Mask (d) corresponds to a causal convolutional attention pattern with wraparound behavior (similar to the row attention) and a 3×3 kernel. Our model uses a mask corresponding to an 11×11 kernel.

In [ ]:
```
```