CS 360: Machine Learning

Sara Mathieson, Sorelle Friedler

Spring 2024

HAVERFORD
COLLEGE

Admin

- **Lab 8** due April 18 (one week from today
  - Sorelle office hours TODAY 4-5pm in H110
- **Midterm April 25** in class
- **Project presentations**: last week of classes
- **Writeup** due by the end of finals period
  - May 11 for seniors
  - May 17 for non-seniors

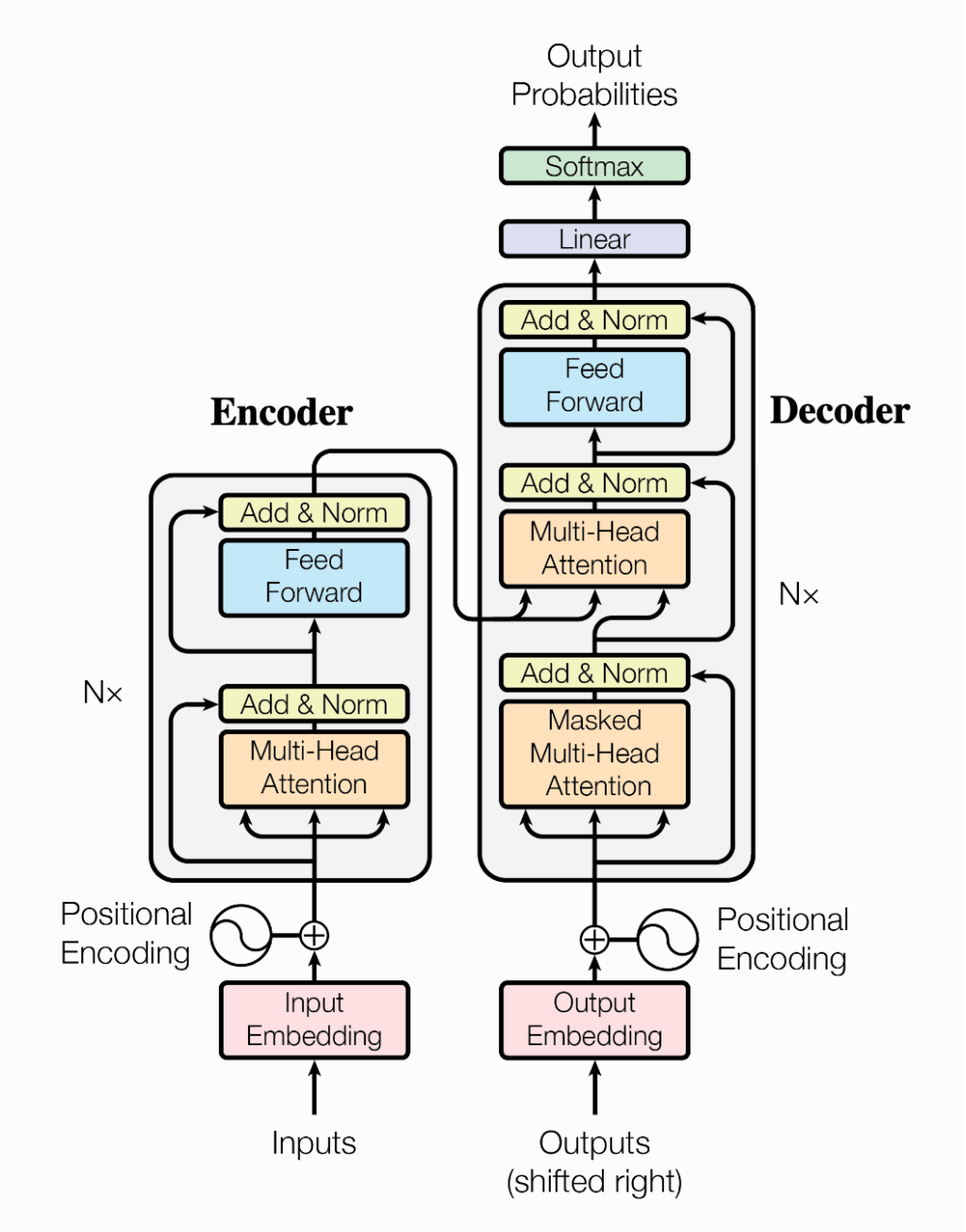
Outline for April 11

- Transfomers big picture + positional encodings
- CNNs for image segmentation
- Generative Adversarial Networks (GANs)

Outline for April 11

- Transfomers big picture + positional encodings
- CNNs for image segmentation
- Generative Adversarial Networks (GANs)

Machine Translation Transformer Architecture



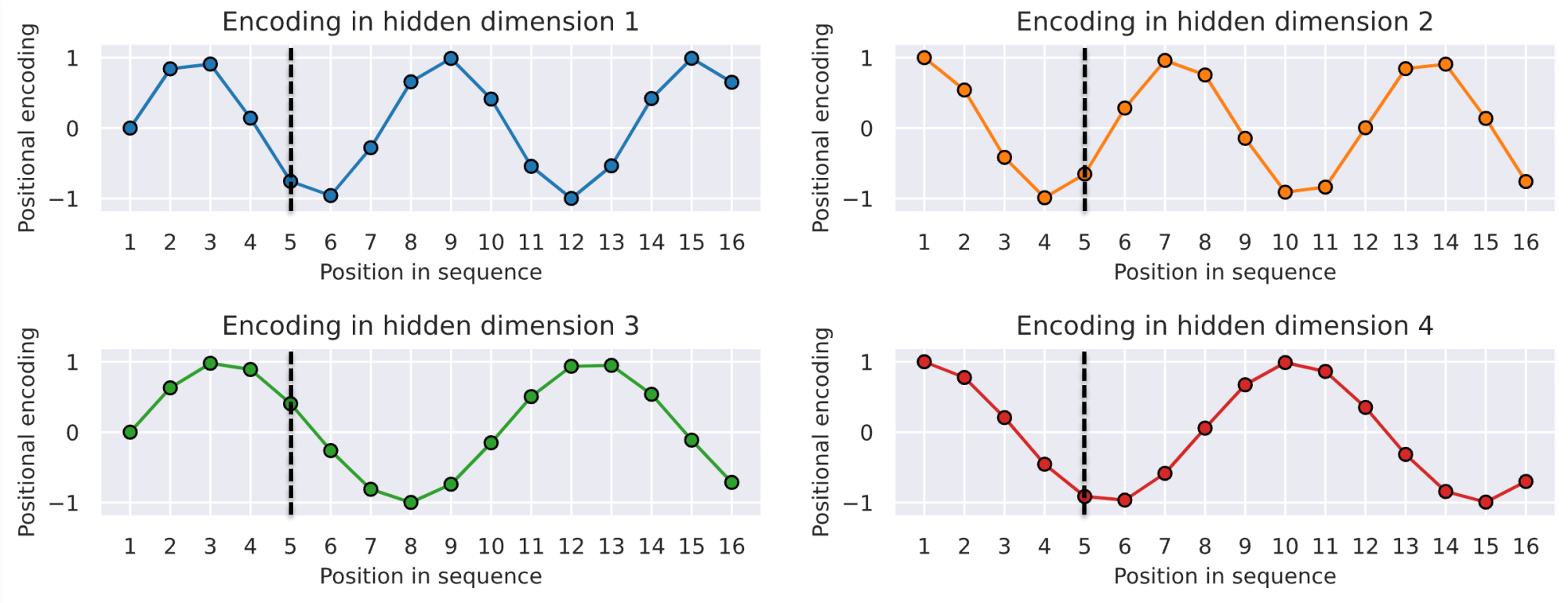
"Attention is all you need"

Positional Encodings

- Problem: multi-head attention is a weighted average
- Since word order is important, need to explicitly encode it
- Can learn the positional encodings with enough data
- However it is often easier to use fixed positional encodings
- Idea: sine/cosine functions! (relative and absolute positions)

Positional Encodings

$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{model}}}}\right) & \text{if } i \bmod 2 = 0 \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{model}}}}\right) & \text{otherwise} \end{cases}$$

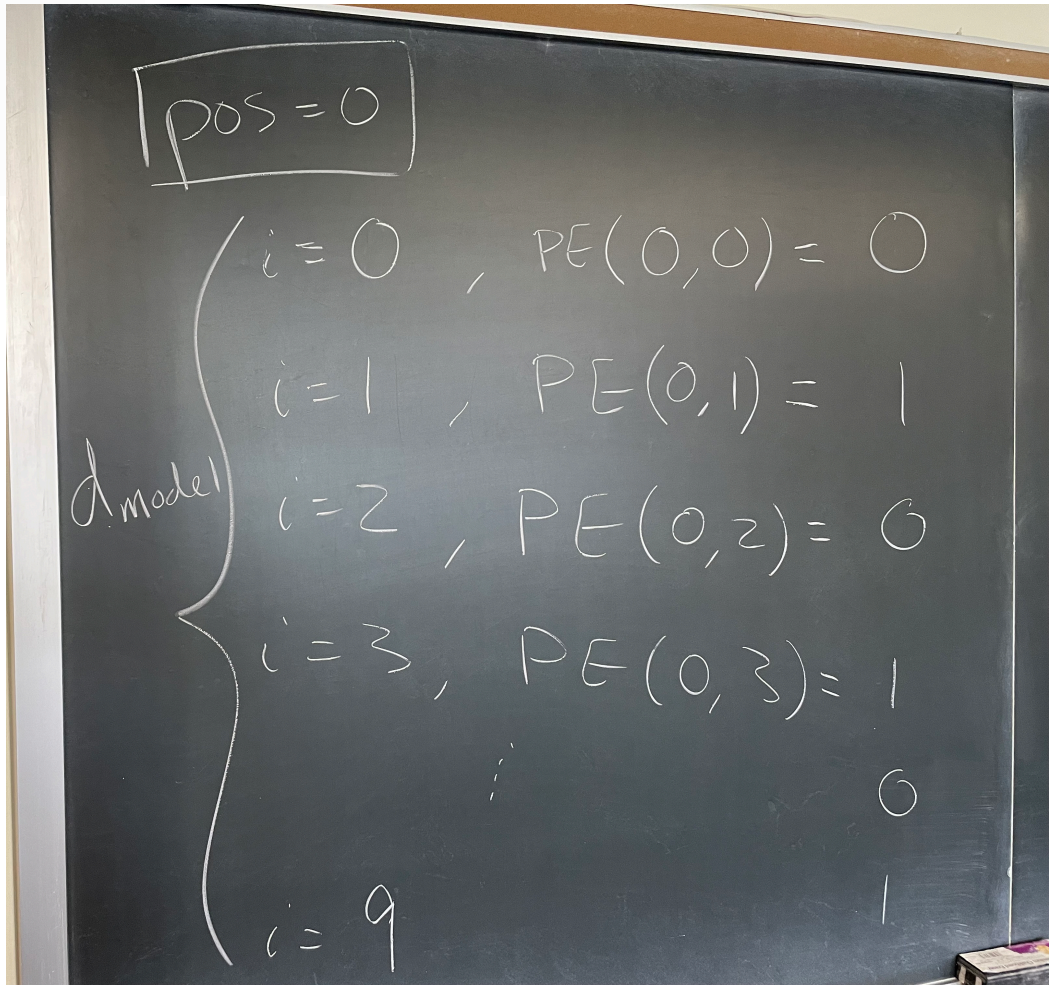


Phillipe Lippe

Positional Encodings

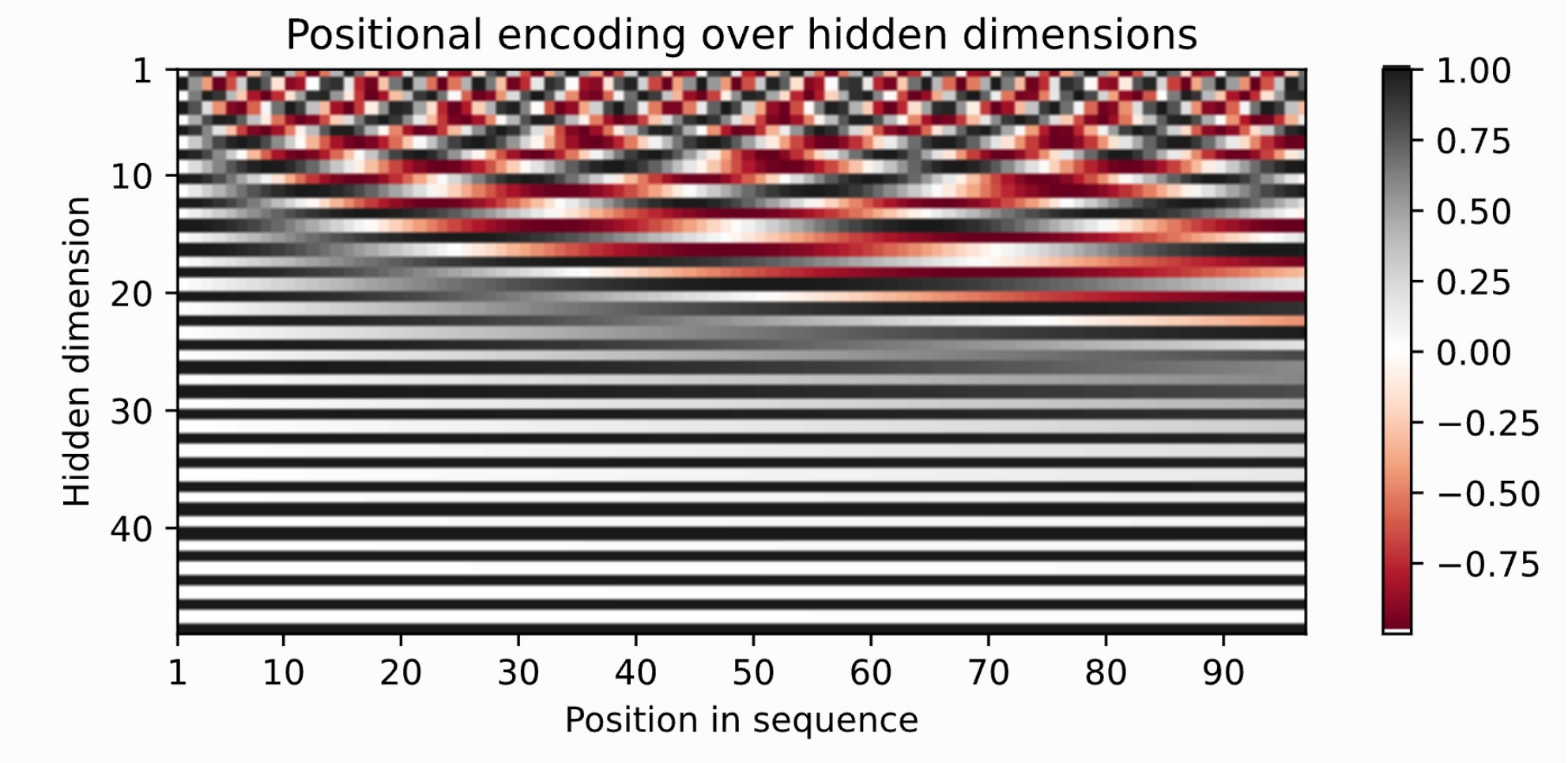
$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{model}}}}\right) & \text{if } i \bmod 2 = 0 \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{model}}}}\right) & \text{otherwise} \end{cases}$$

Pair exercise: if pos=0 and d_{model} = 10, what are the positional encodings?



Phillipe Lippe

Positional Encodings



- Positional encodings are *added* to the embedding
- This creates the input to the transformer

Transformer big-picture

Full tutorial:

https://highdimensionalgrace.com/posts/big_transformer/

Credit: Grace Proebsting

Transformer workflow for text generation

Notation used throughout

X

Matrix Multiply

+

Matrix Addition

÷

(scalar) division

If one divides a matrix by some real number x, then each element in matrix is divided by x.

$softmax$

Converts a vector of real numbers into a probability distribution (i.e. makes them sum to one.)

$LayerNorm$

The gist: scales a vector of real numbers such that their mean is 0 and variance is 1.

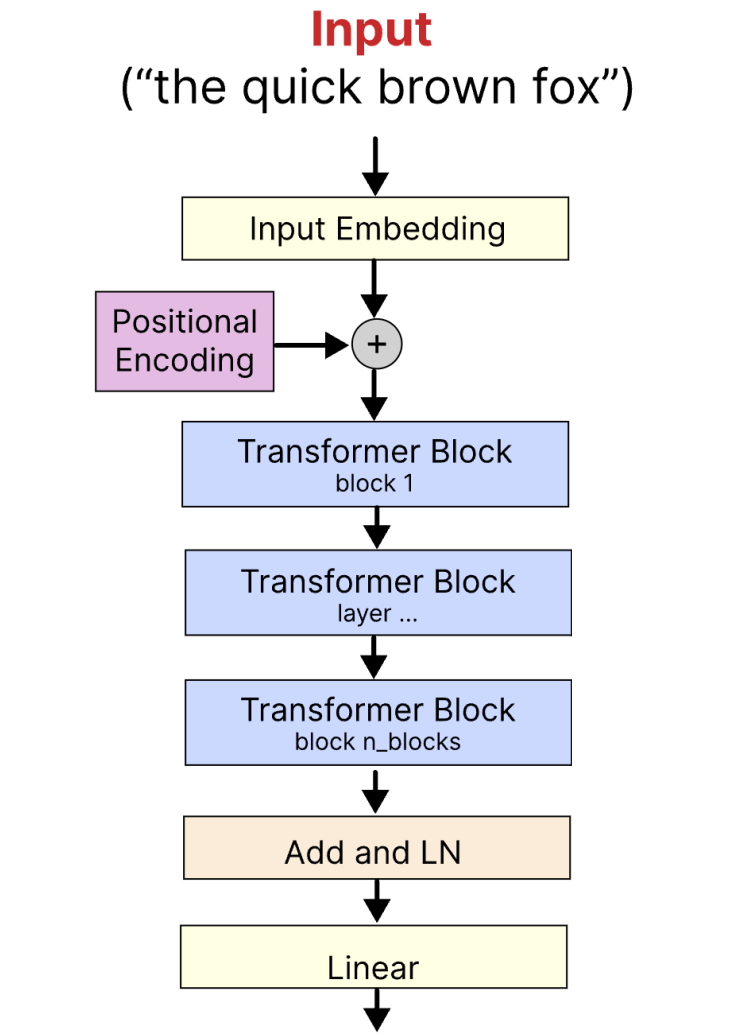
$*eLU$

(???) Linear Unit

probably a Gaussian Linear Unit (GeLU) or a Rectified Linear Unit (ReLU). The gist: “zeros out” negative numbers.

Credit: Grace Proebsting

Transformer workflow for text generation



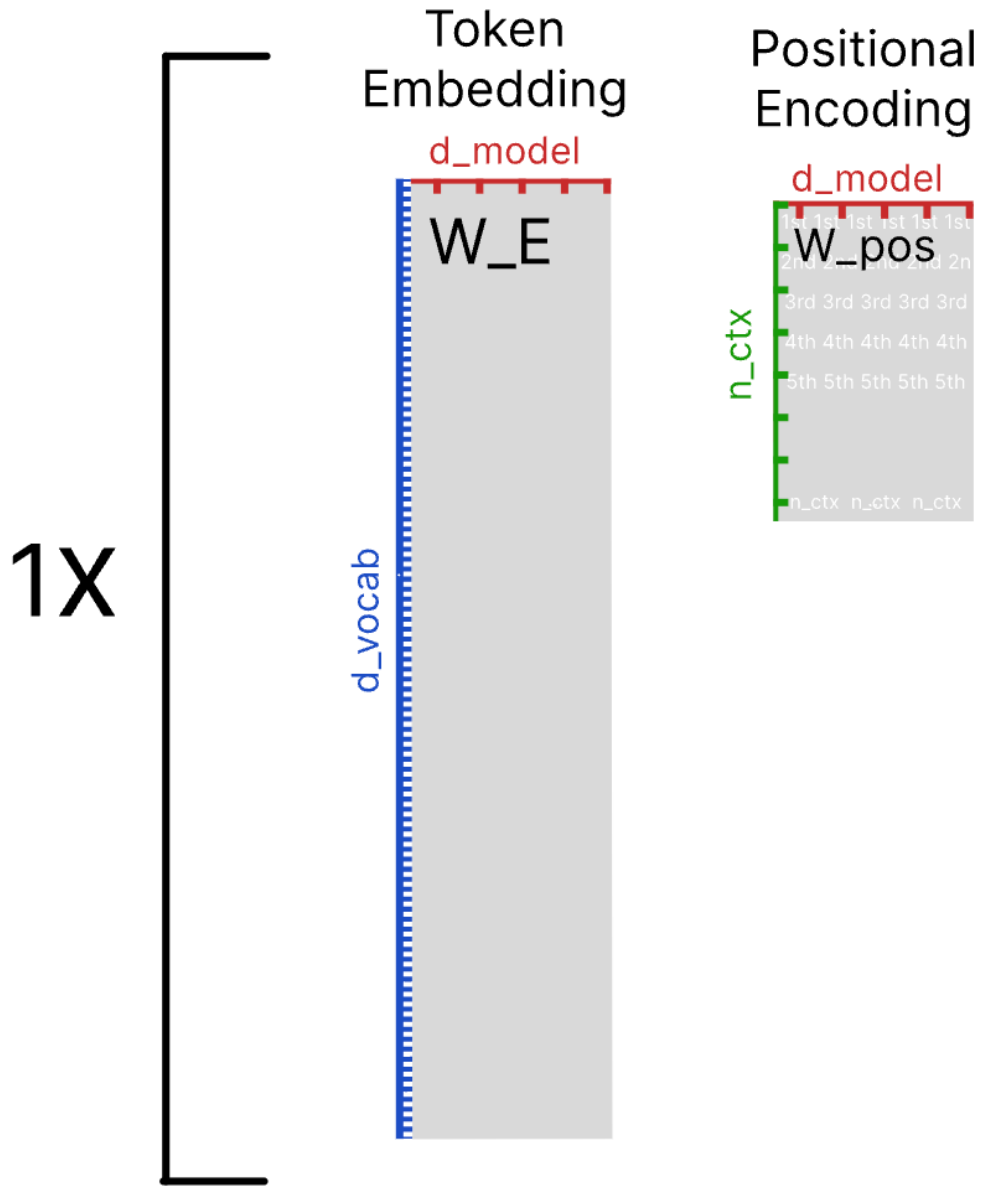
Each token in the model's vocabulary is given a score (a scalar value, called its 'logits') representing how likely the token is to follow the context.

In this case, 'jumped' would (hopefully) have the highest logits, while 'kumquat' might have low logits.

Credit: Grace Proebsting

Transformer workflow for text generation

Parameters

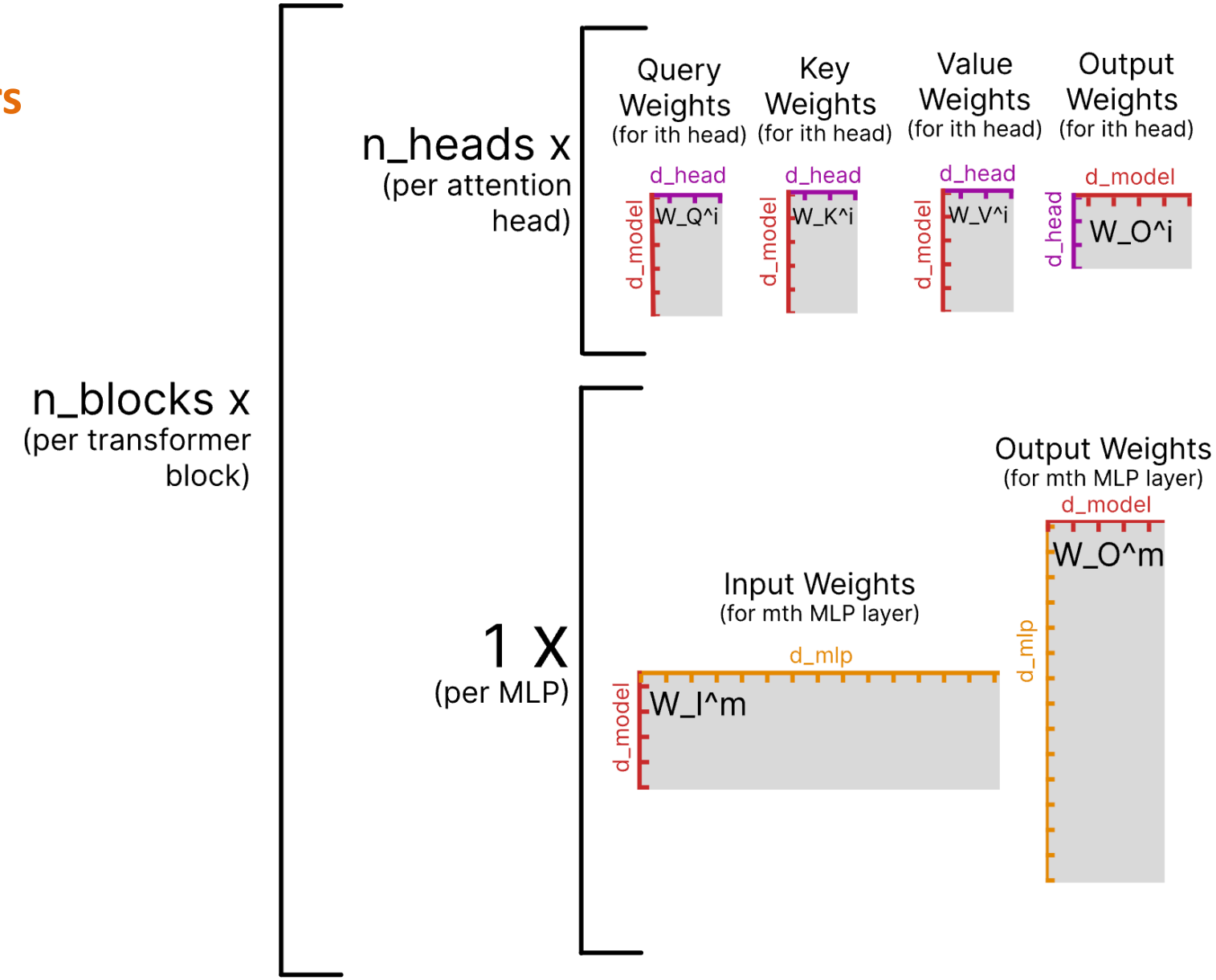


- **n_ctx** is the size of the model's context window (i.e. the max number of input tokens.)
- **d_model** is the 'embedding size.'
- **d_vocab** is the # of words in the model's vocabulary.

Credit: Grace Proebsting

Transformer workflow for text generation

Parameters



- **d_head** is the size of the queries/keys/values.
- **d_mlp** is the size of the MLP layer's "hidden layer".
- **n_blocks** is the # of 'transformer blocks' in the model, with 1 MLP layer and 1 Attn Head layer per block.
- **n_heads** is the # of attention heads per Attn Head layer.

Credit: Grace Proebsting

Transformer workflow for text generation

Parameters

For reference, **gpt2-small** has the following configuration:

- d_model = 768
- n_ctx = 1024
- d_vocab = 50257
- d_head = 64
- d_mlp = 3072
- n_blocks = 6
- n_heads = 12

This means that **gpt2-small has**:

- 6 multi-headed attn layers, with 12 heads per layer (so 72 attn heads in total.)
- 6 MLP layers.

Credit: Grace Proebsting

Transformer workflow for text generation

Input steps

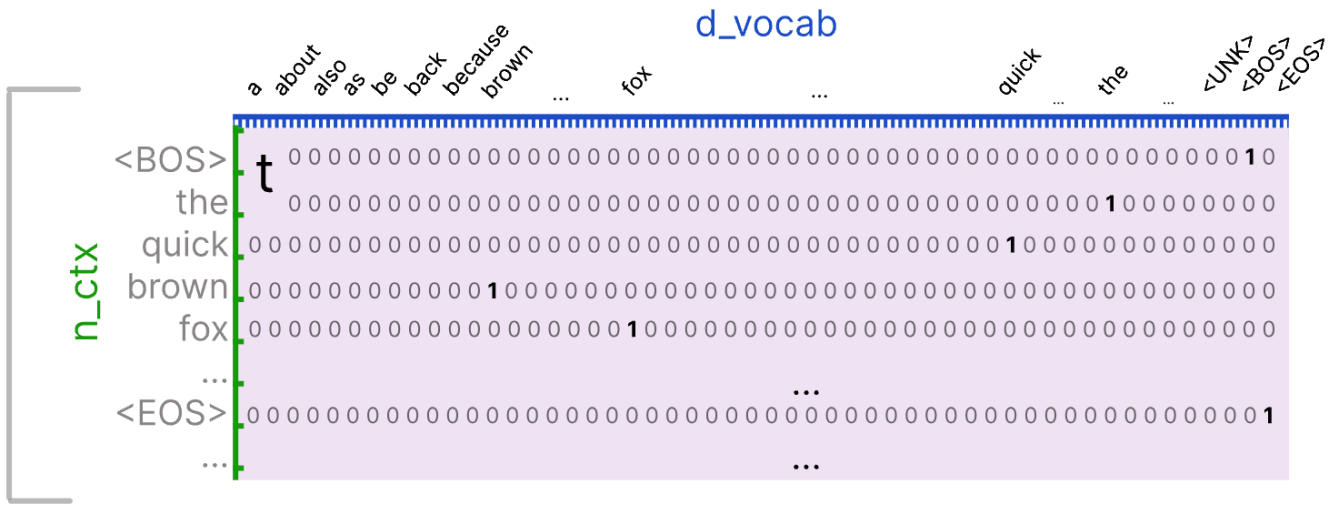
Step 0: First, convert our input string (the 'context') into tokens. Then, convert the context tokens into a matrix of 'one-hot encodings' (**t**)

Input string:
"the quick brown fox [...]"

Note: Normally, a model's vocab tokens are *sub-word pieces* rather than full words.

Tokenized input:
[<BOS>, the, quick, brown, fox, [...], <EOS>]

One-hot encodings of tokens, 't':



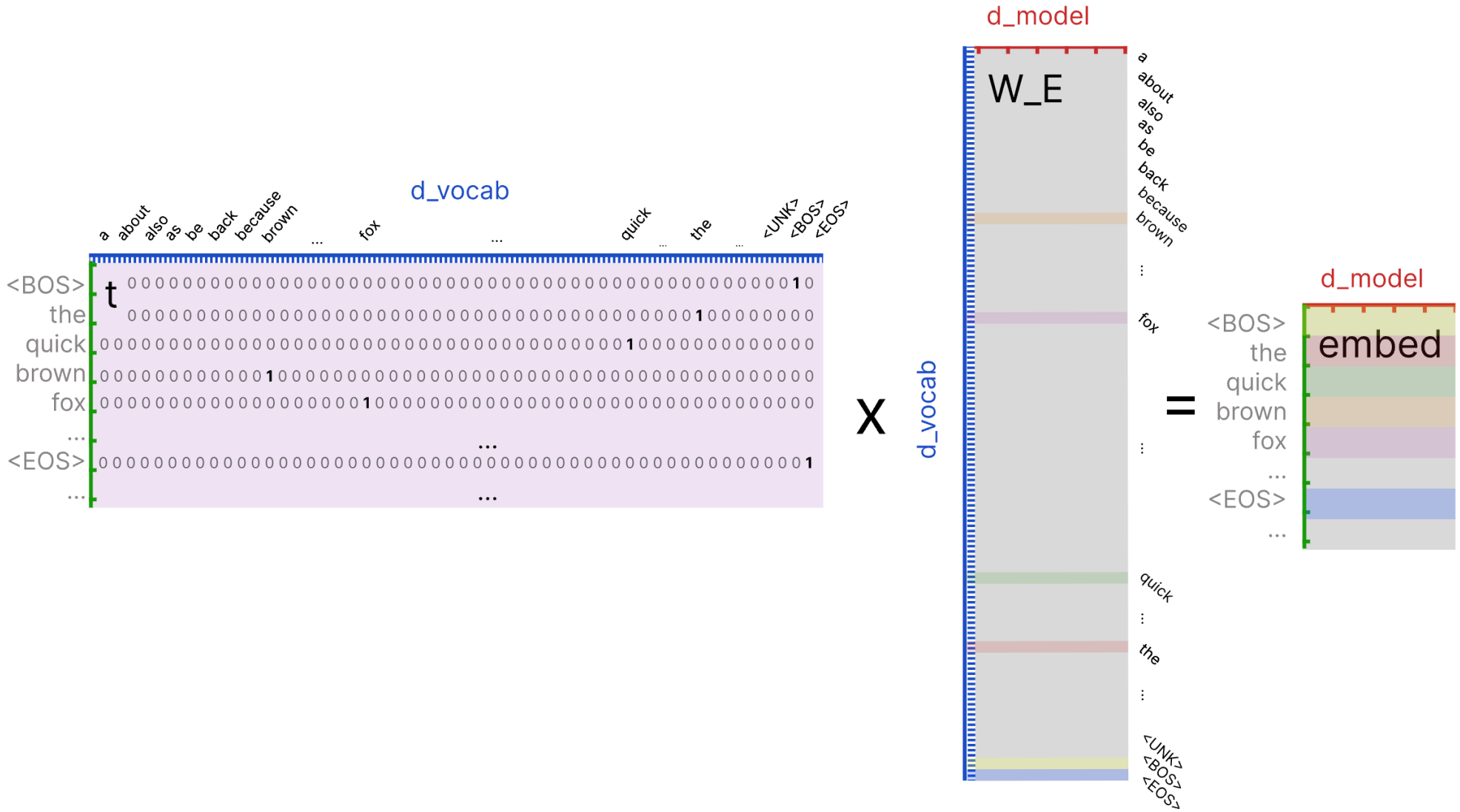
Each row represents a token in the context, and has a single nonzero entry: a 1 at the column corresponding to the row's token in the model's vocab.

Credit: Grace Proebsting

Transformer workflow for text generation

Input steps

Step 1: Multiply the one-hot encodings of our context (**t**) by the token embedding weights (**W_E**) to get the 'direct embedding' vectors for our context (**embed**).

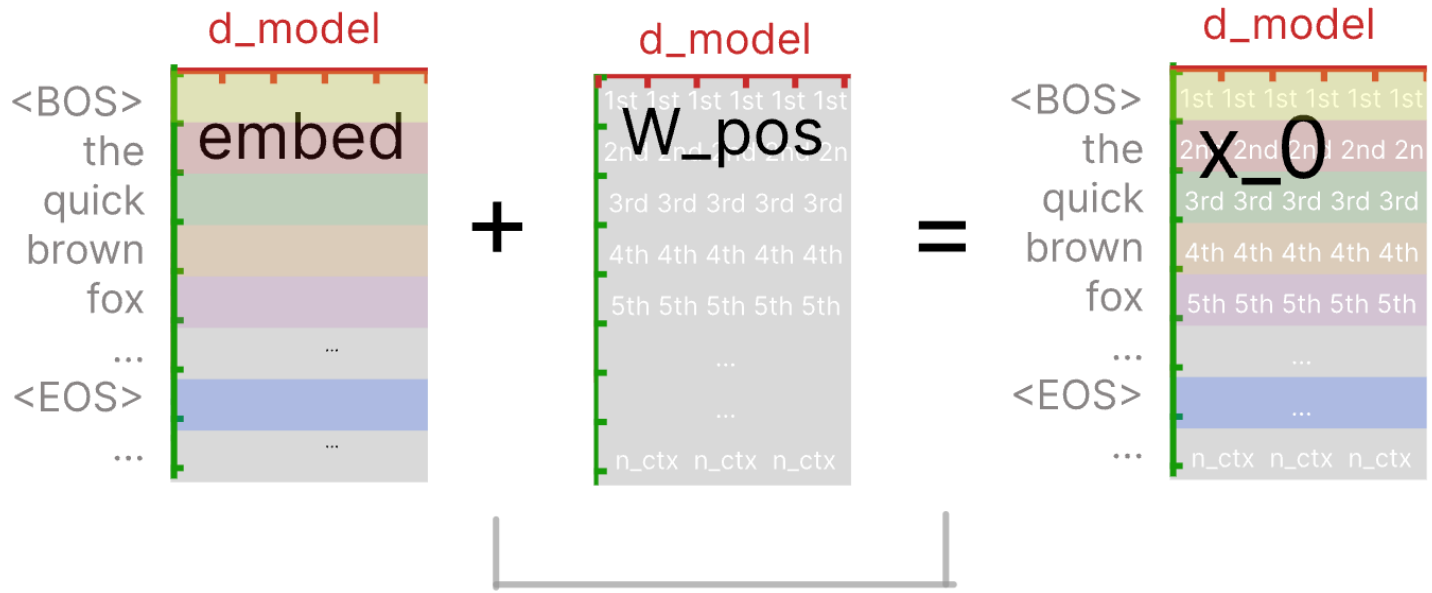


Credit: Grace Proebsting

Transformer workflow for text generation

Input steps

Step 2: Add the 'direct embeddings' for our context (**embed**) to the positional embedding (**W_pos**) to get the embeddings for the 0th layer, **x_0**.



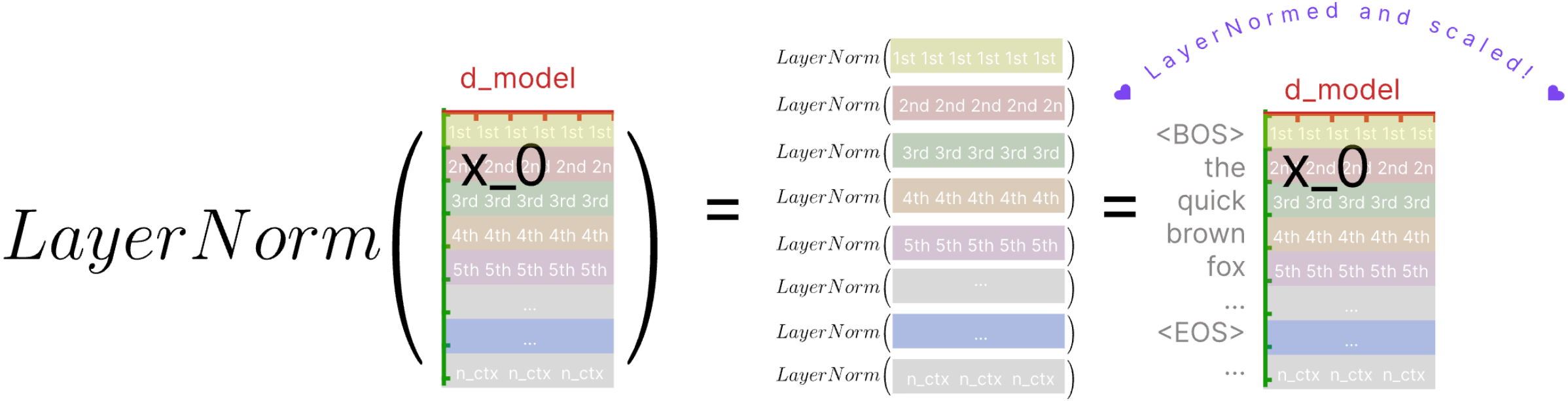
W_pos is either learned via gradient descent (just like all the other parameters) or 'hard-coded' as a mathematical function (i.e. using ~trig magic~!)

Credit: Grace Proebsting

Transformer workflow for text generation

Input steps

Step 3: LayerNorm the context embeddings for the 0th layer (**x_0**) so it can be inputted into the attention head layer. (In other words, normalize the embedding vector at each position in the context such that it has a mean of 0 and variance of 1.)

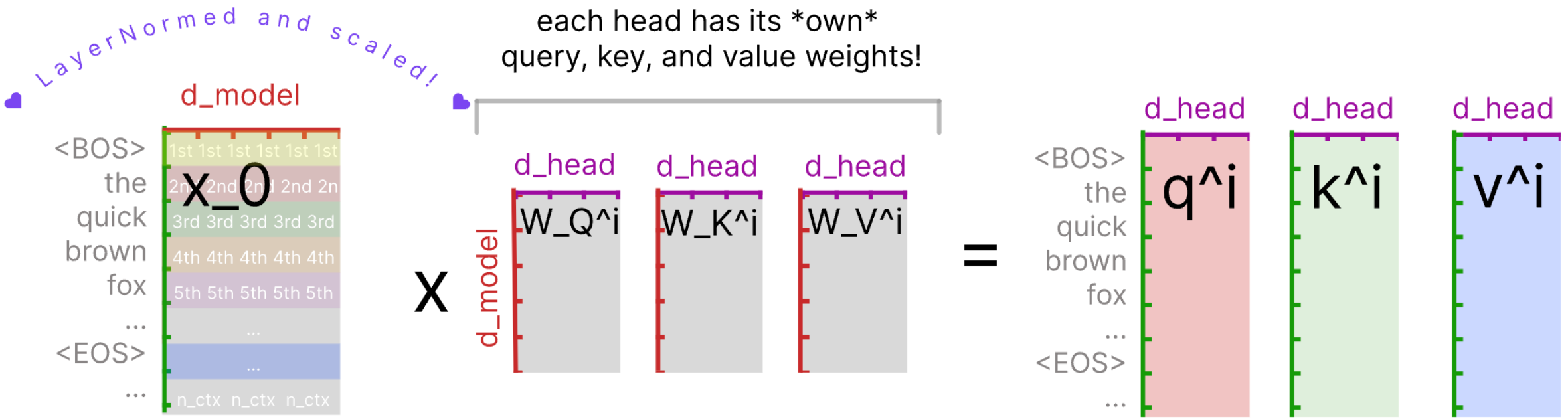


Credit: Grace Proebsting

Transformer workflow for text generation

Transformer block steps

1. Multiply the layernormed embeddings of our context for the 0th layer (**x_0**) by the key, query, and value weights for the ith head (**W_Q^i**, **W_K^i**, **W_V^i**) to get its queries, keys, and values (**q^i, k^i, v^i**).

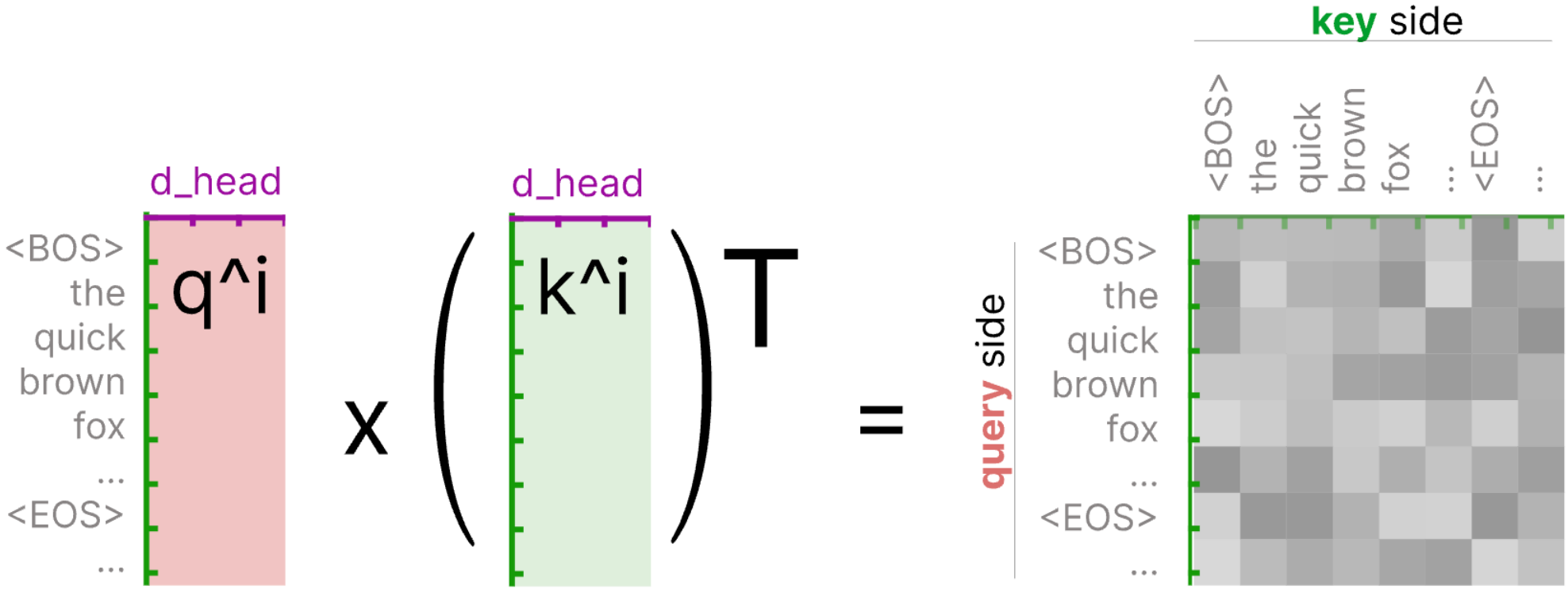


Credit: Grace Proebsting

Transformer workflow for text generation

Transformer block steps

2. Multiply the queries for the ith head (**q^i**) by its keys (**k^i**) to get its unnormalized/ unmasked attention scores.

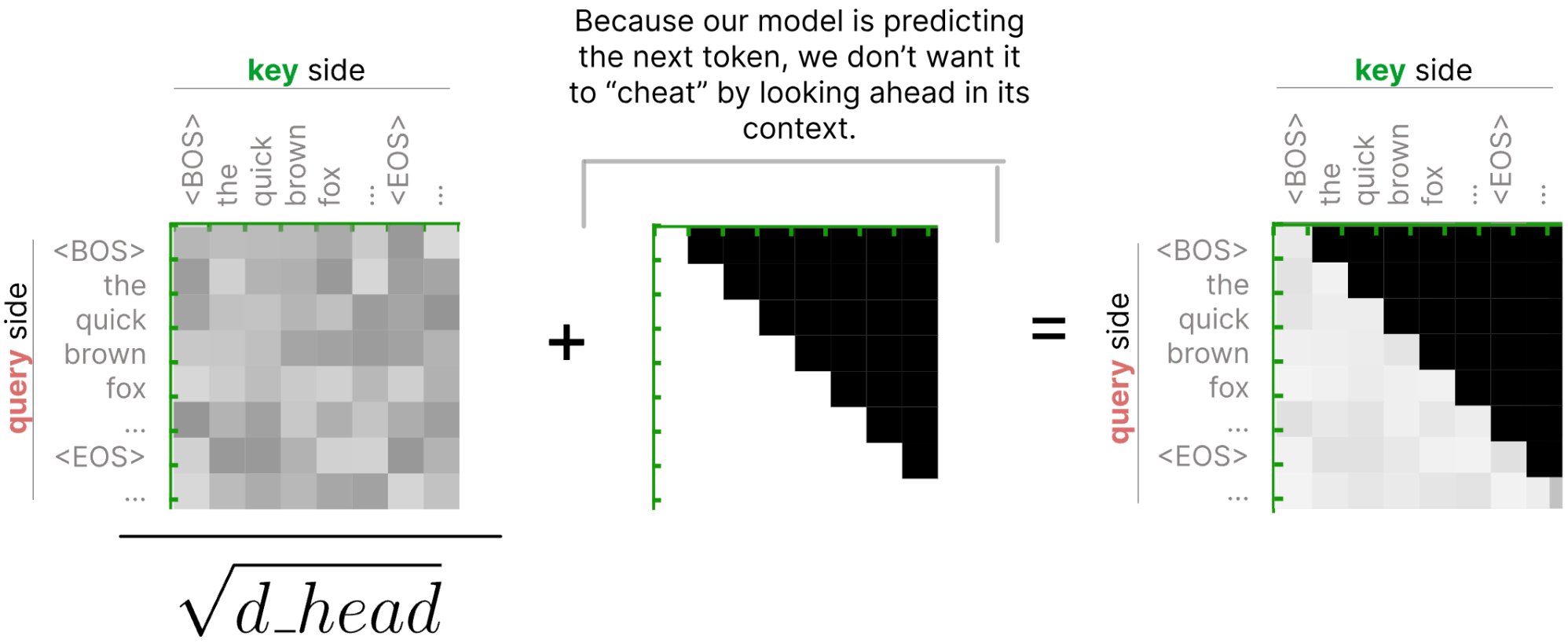


Credit: Grace Proebsting

Transformer workflow for text generation

Transformer block steps

3. Normalize the attention scores by dividing the scores by (the scalar) $\sqrt{d\_head}$.
   Mask the attention scores by adding 'negative infinity' (whatever that means) to all attention scores above the diagonal.

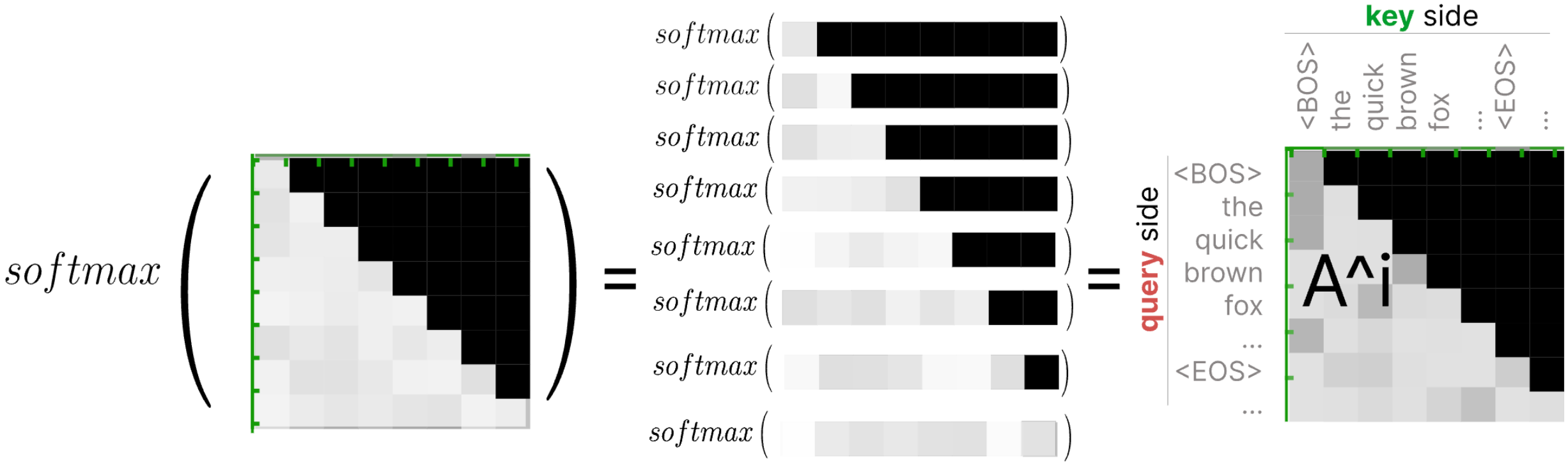


Credit: Grace Proebsting

Transformer workflow for text generation

Transformer block steps

4. Softmax the masked and normalized attention scores along the rows (i.e. along the ‘key-axis’) to get the attention pattern for the ith head, **A^i**.



Credit: Grace Proebsting

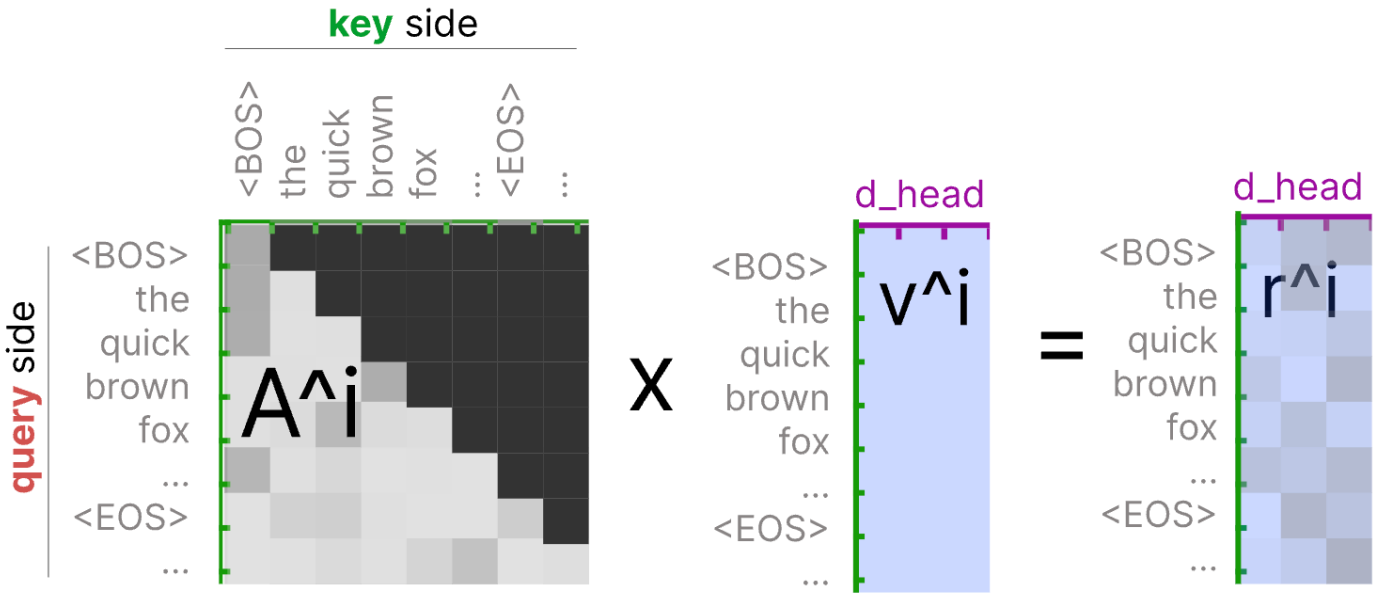
Transformer workflow for text generation

Transformer block steps

5. Multiply the attention pattern for the ith head (**A^i**) by its values (**v^i**) to get its results (**r^i**). (We call each row of r^i a 'result vector'.)

Each result vector is a weighted sum of the value vectors, where the weights are rows of the attention pattern.

For example, the "fox result vector" is a weighted sum of the value vectors, where the weights are the "fox row" of the attention pattern.

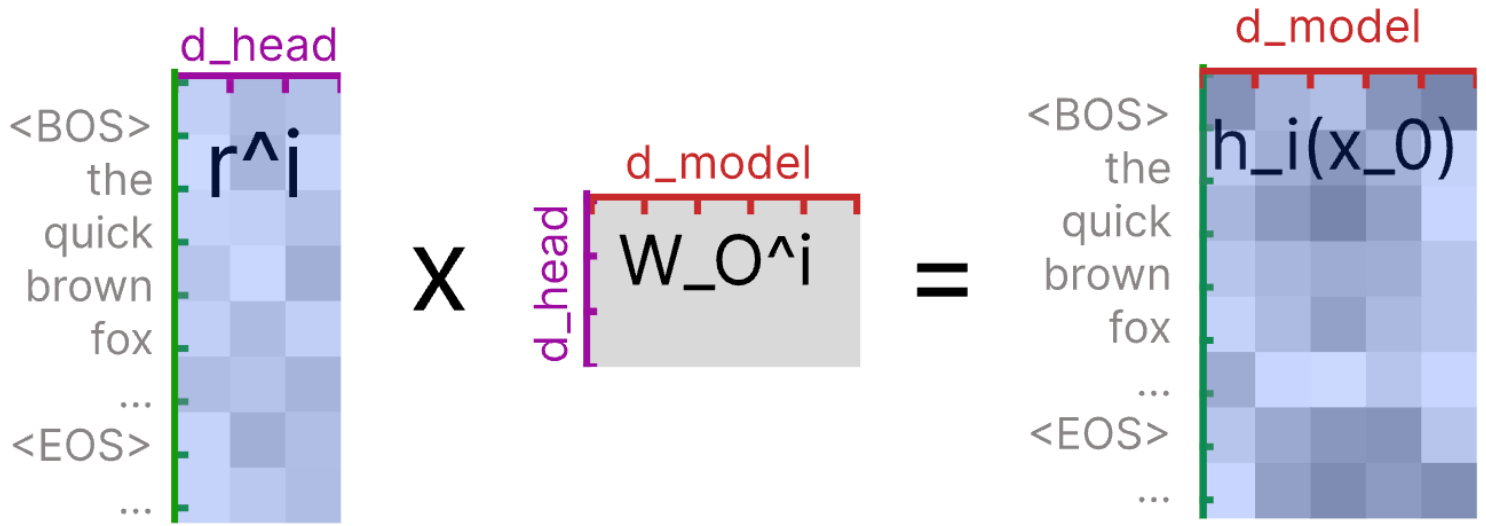


Credit: Grace Proebsting

Transformer workflow for text generation

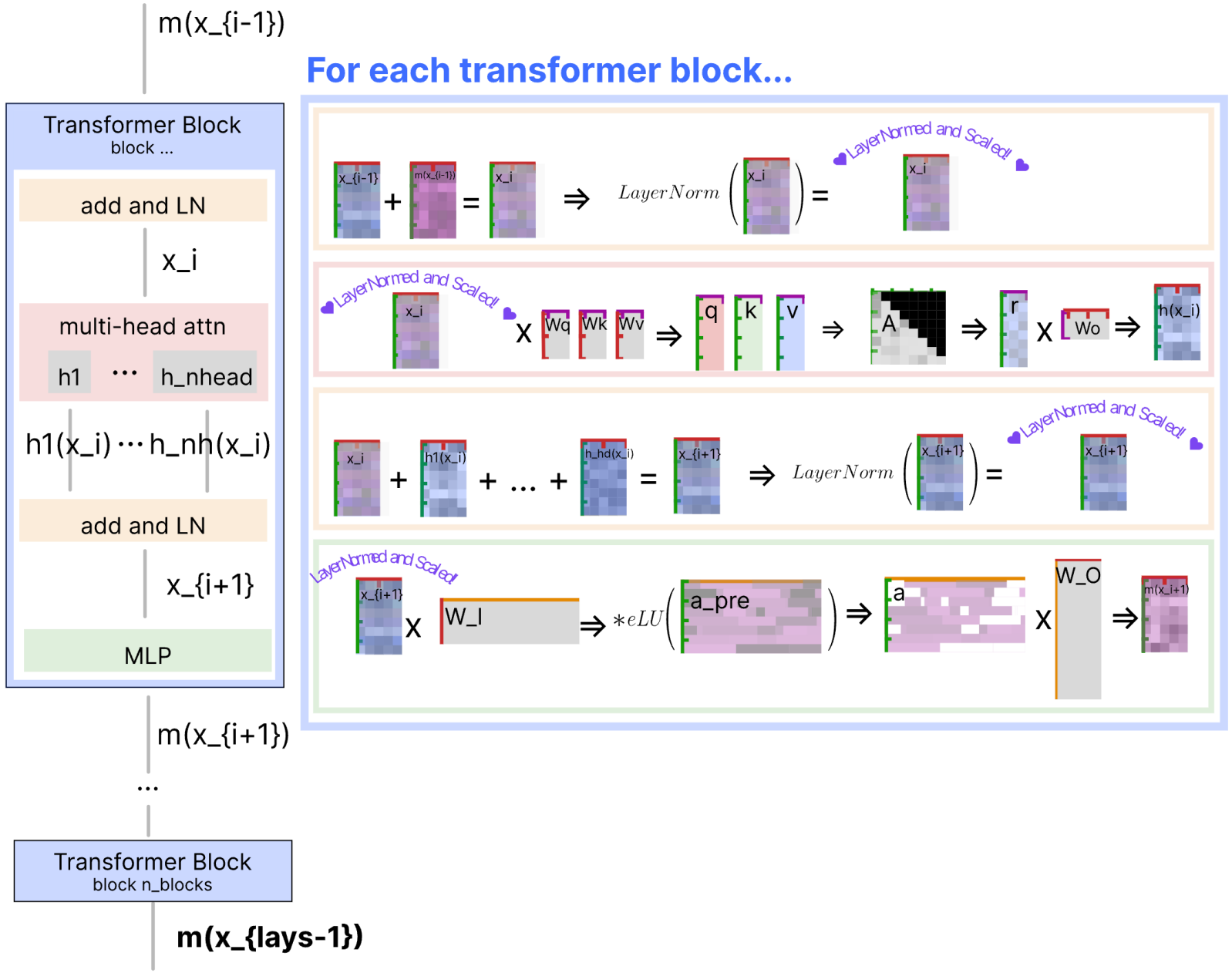
Transformer block steps

6. Multiply the ith head's results (**r^i**) by its output matrix (**W_O^i**) to get the output of the ith attention head in the 1st multi-headed attention layer, **h_i(x_0)**.



Credit: Grace Proebsting

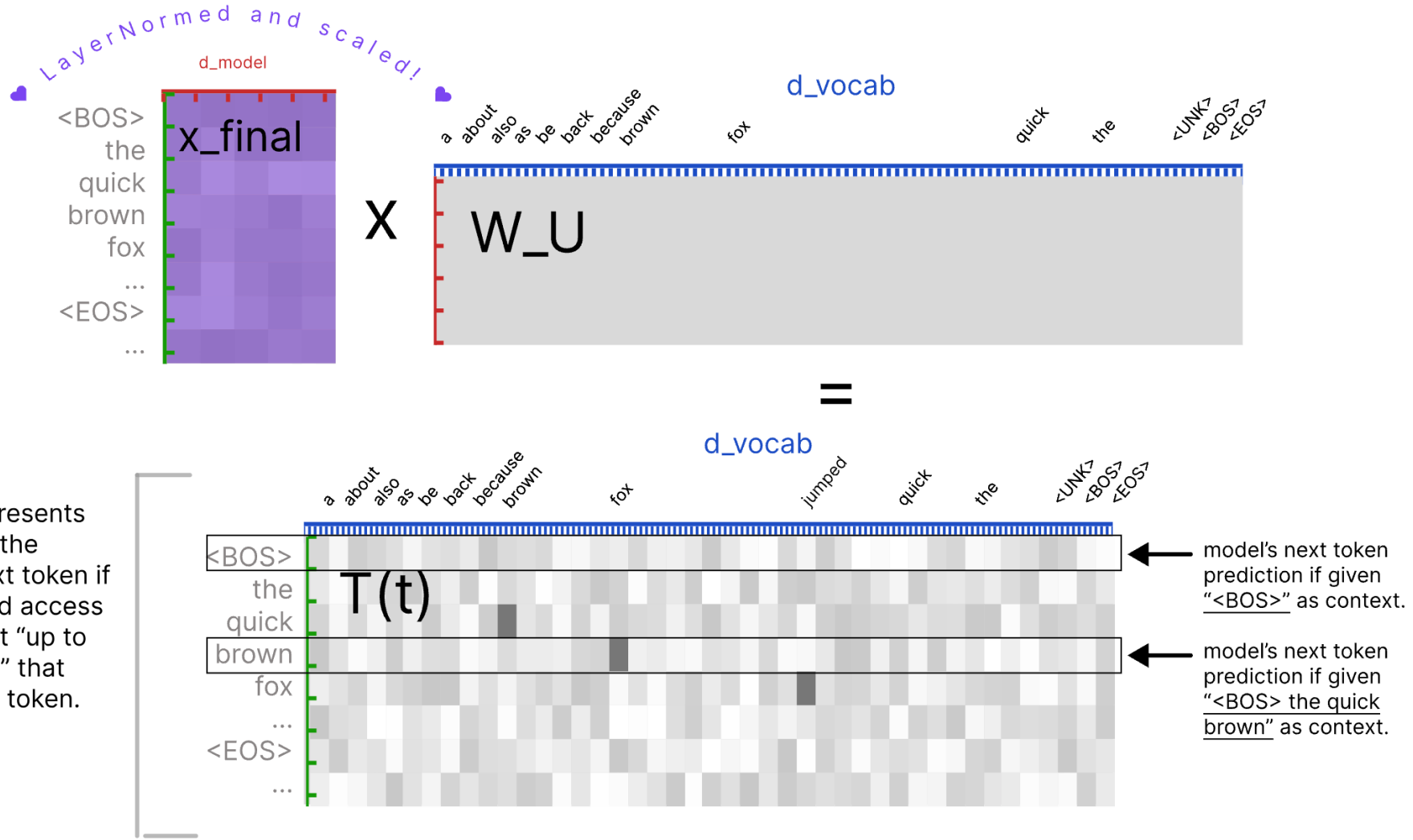
Transformer workflow for text generation



Credit: Grace Proebsting

Transformer workflow for text generation

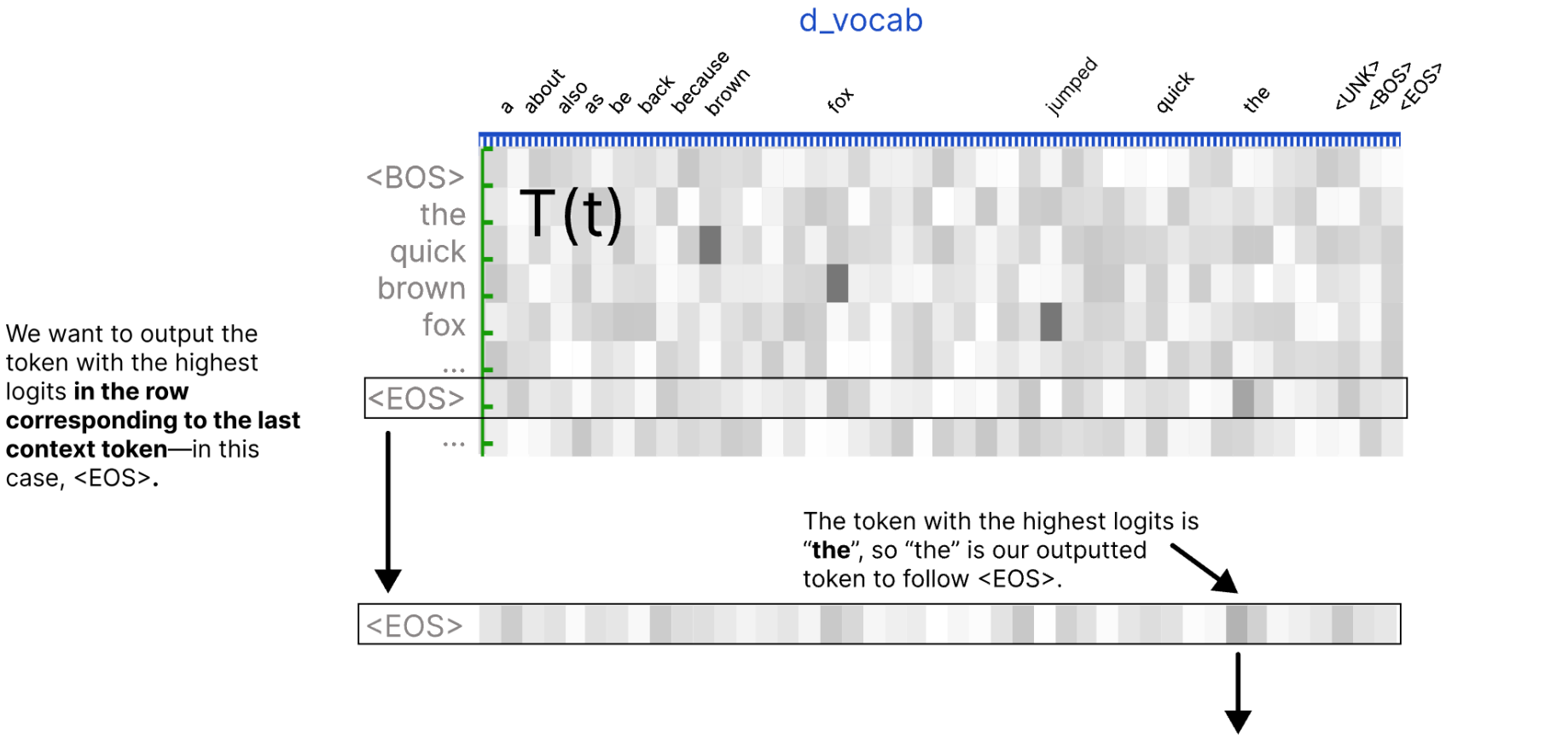
Step 14: Multiply the layernormed final embedding for our context (**x_final**) by the unembedding weights, **W_U,** to get the transformer logits, **T(t)**.



Credit: Grace Proebsting

Transformer workflow for text generation

Optional final step: if we want to use our GPT model to **generate novel text** to follow an inputted context, we create our output by 1) extracting the row of T(t) corresponding to the last token in the input, and 2) outputting the vocab token from this row with the highest logits.



Credit: Grace Proebsting

Outline for April 11

- Transfomers big picture + positional encodings
- CNNs for image segmentation

Next time!

- Generative Adversarial Networks (GANs)

Outline for April 11

- Transfomers big picture + positional encodings
- CNNs for image segmentation
- Generative Adversarial Networks (GANs)

GAN progress over time

Ian Goodfellow
@goodfellow_ian

Follow

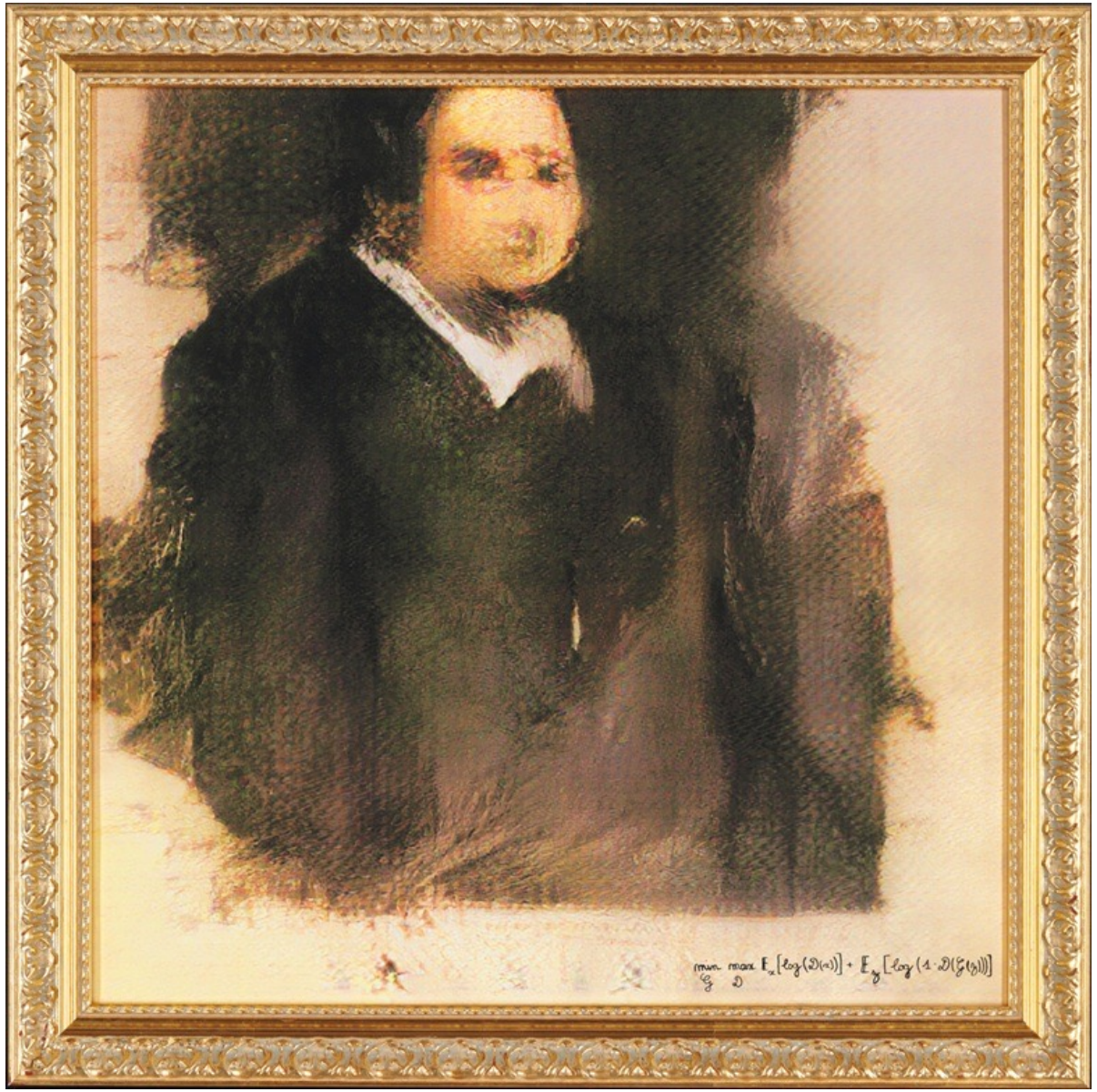
4.5 years of GAN progress on face generation. arxiv.org/abs/1406.2661
arxiv.org/abs/1511.06434
arxiv.org/abs/1606.07536
arxiv.org/abs/1710.10196
arxiv.org/abs/1812.04948

4:40 PM - 14 Jan 2019

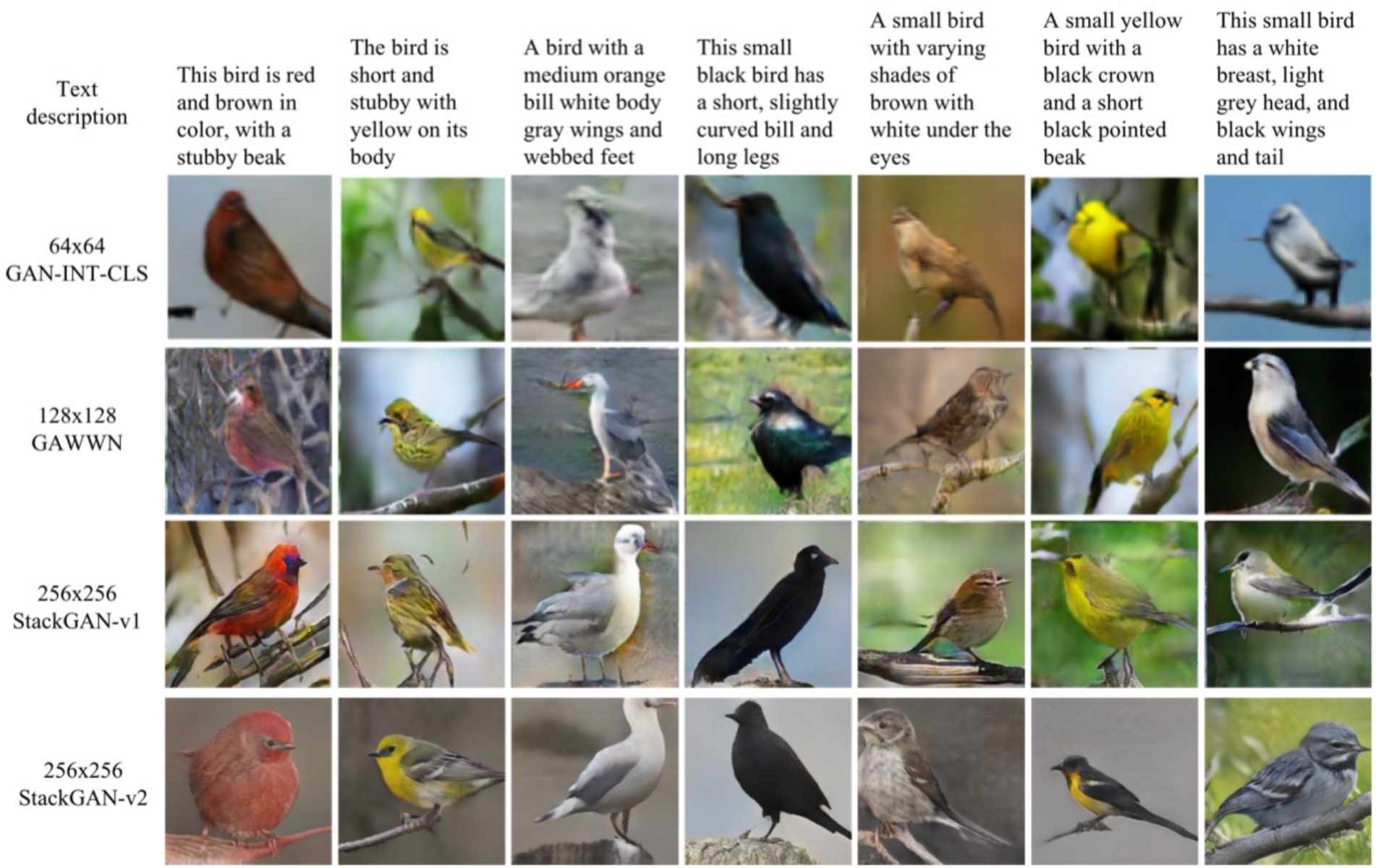
StyleGAN. Karras et al. (2019)

GAN painting

Sold for almost half a million dollars



GANs have been very successful in other domains



https://arxiv.org/pdf/1710.10916v3.pdf

Idea behind GANs (Generative Adversarial Networks)

Which is “real” and which is “fake”?

Centre de Estudios Borjanos/AFP/Getty Images

Idea behind GANs (Generative Adversarial Networks)

Which is “real” and which is “fake”?

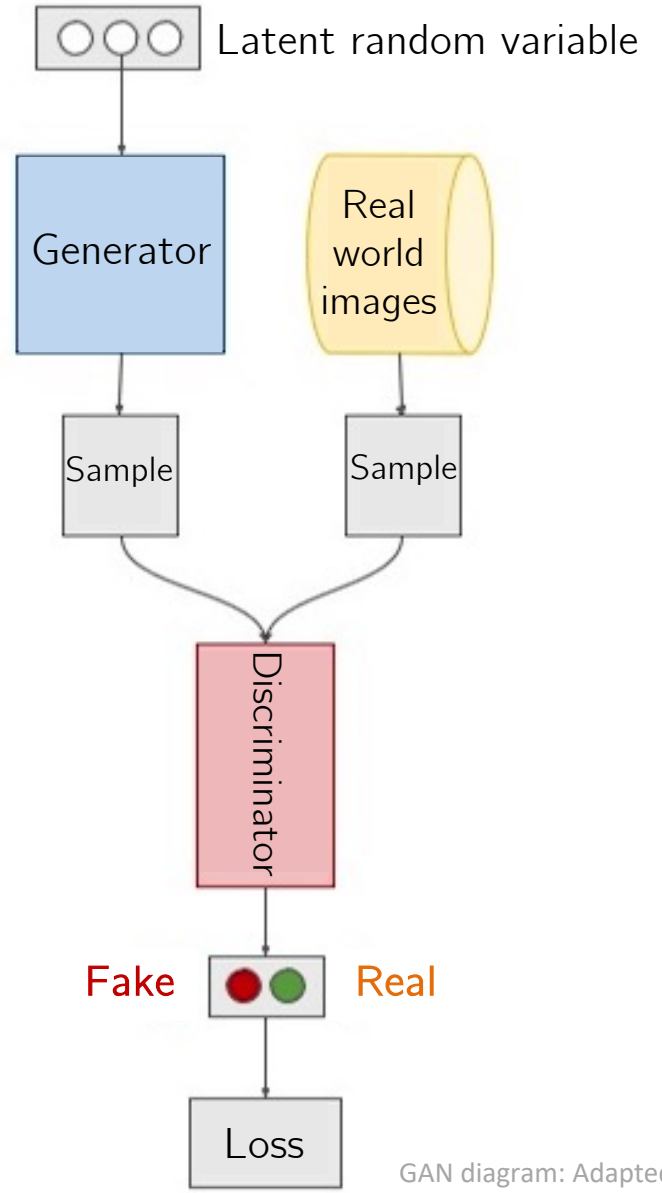
https://webartacademy.com/fake-picasso

Idea behind GANs (Generative Adversarial Networks)

Which is “real” and which is “fake”?

Photo: Courtesy International Foundation for Art Research (IFAR).

Typical architecture of an image GAN



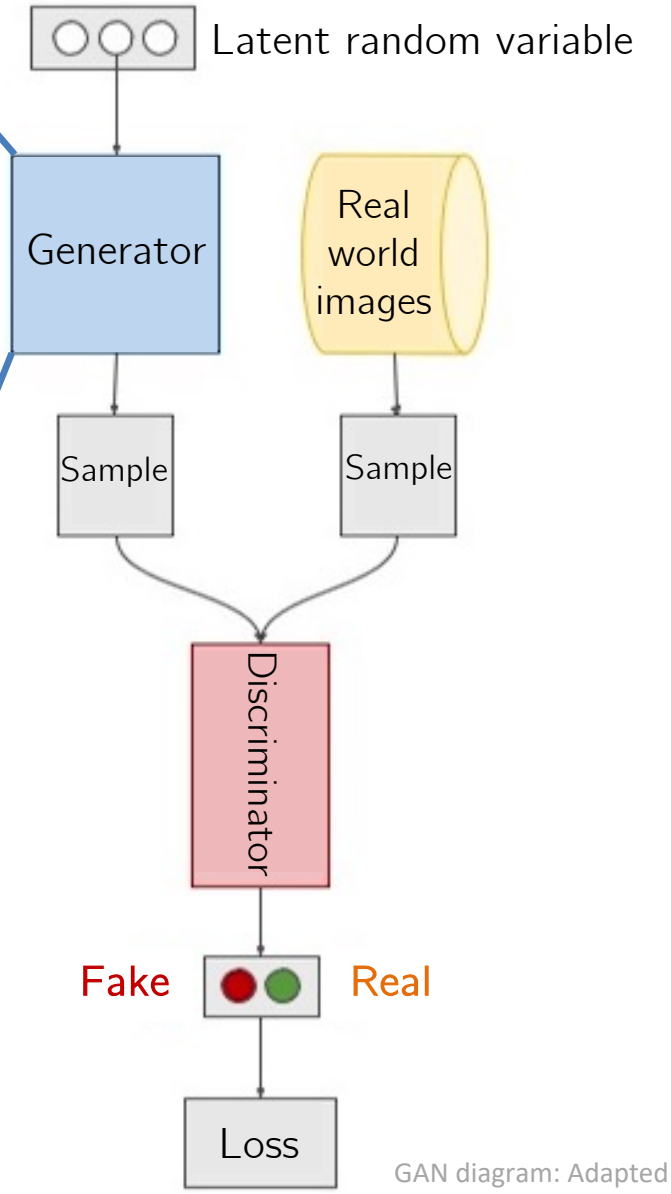
People Images / Getty Images

GAN diagram: Adapted from Kevin McGuinness

(Source: Lost in the Louvre)

Typical architecture of an image GAN

Generator ("forger") tries to create realistic artwork



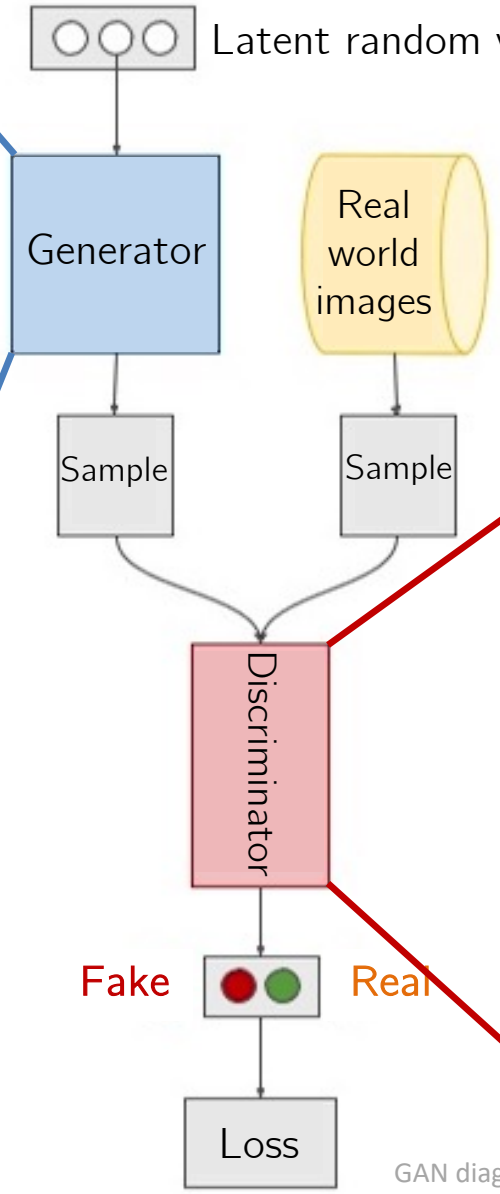
People Images / Getty Images

GAN diagram: Adapted from Kevin McGuinness

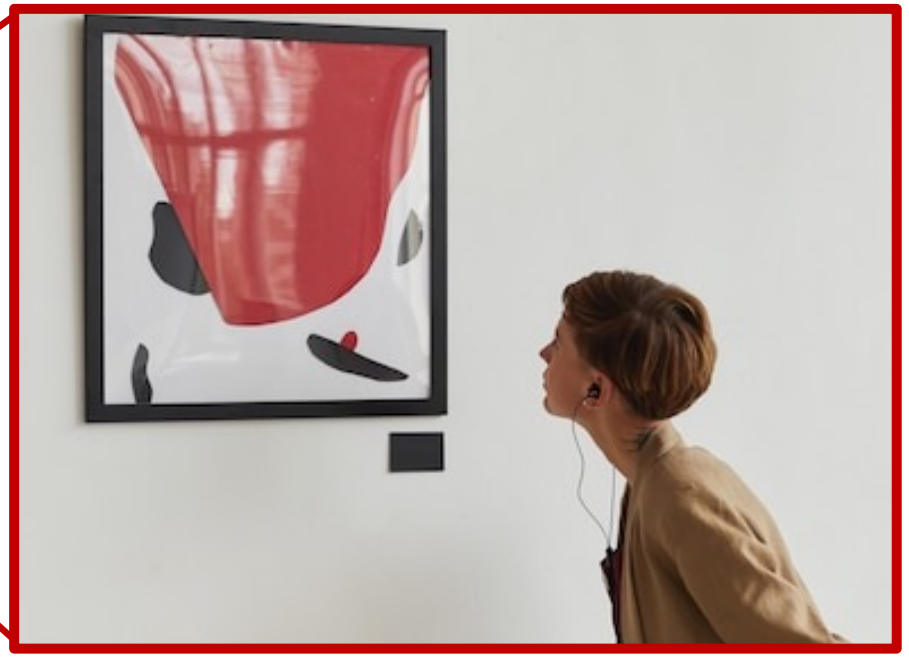
(Source: Lost in the Louvre)

Typical architecture of an image GAN

Generator ("forger") tries to create realistic artwork



Discriminator ("art critic") tries to identify real vs. fake



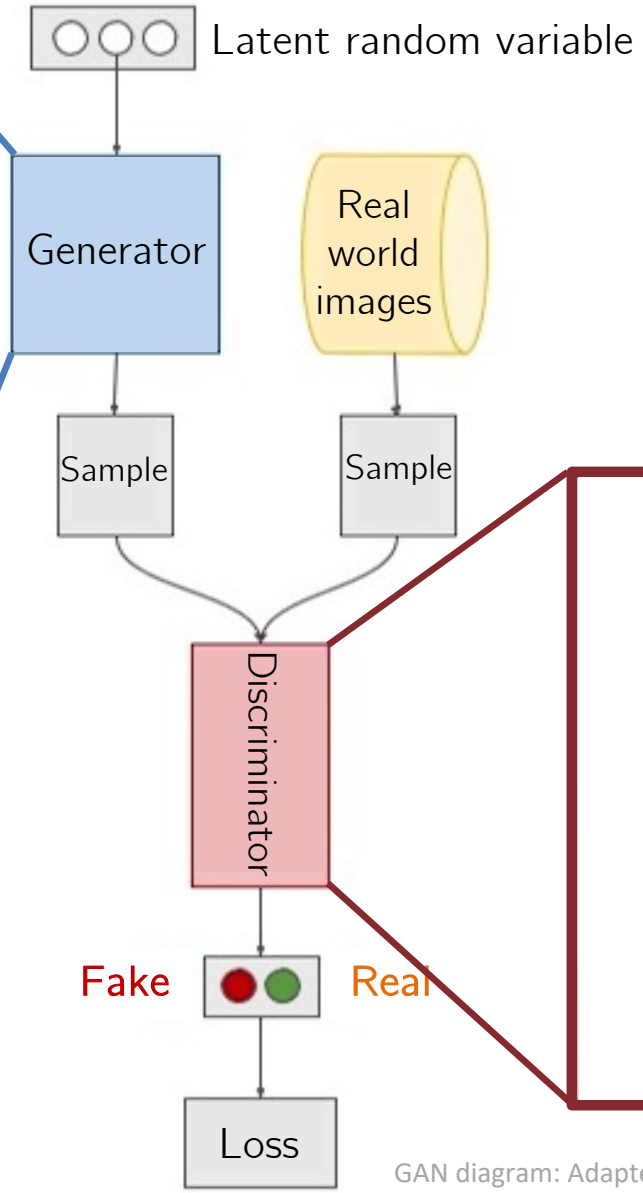
People Images / Getty Images

GAN diagram: Adapted from Kevin McGuinness

(Source: Lost in the Louvre)

Typical architecture of an image GAN

For images: generator is often a CNN



Discriminator is also often a CNN
(Note: goal is 50% accuracy)

GAN diagram: Adapted from Kevin McGuinness

Another GAN architecture for Fashion MNIST

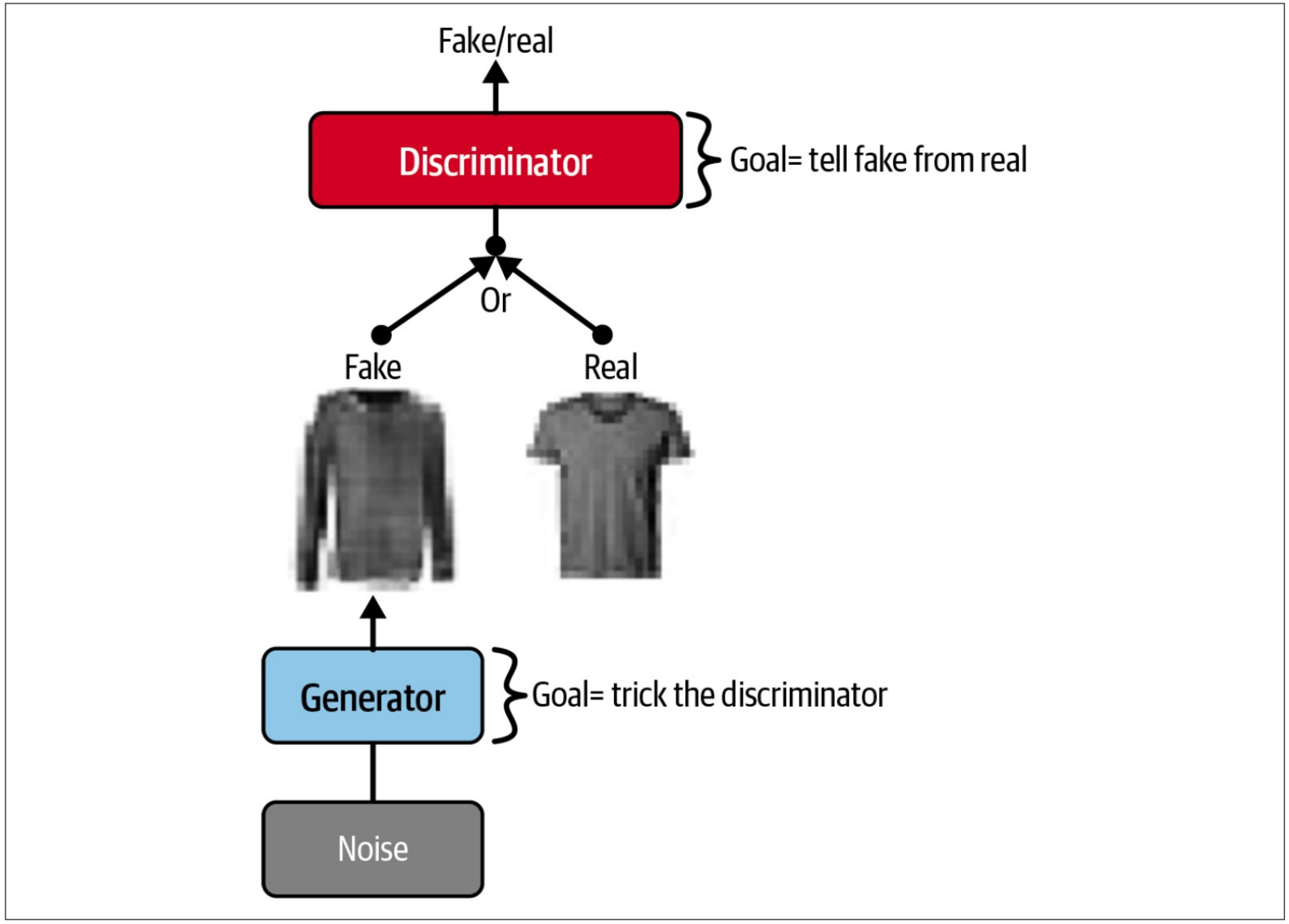


Figure 17-14. A generative adversarial network

Geron, Chap 17

Another GAN architecture for Fashion MNIST

Figure 17-15. Images generated by the GAN after one epoch of training

Geron, Chap 17

Another GAN architecture for Fashion MNIST

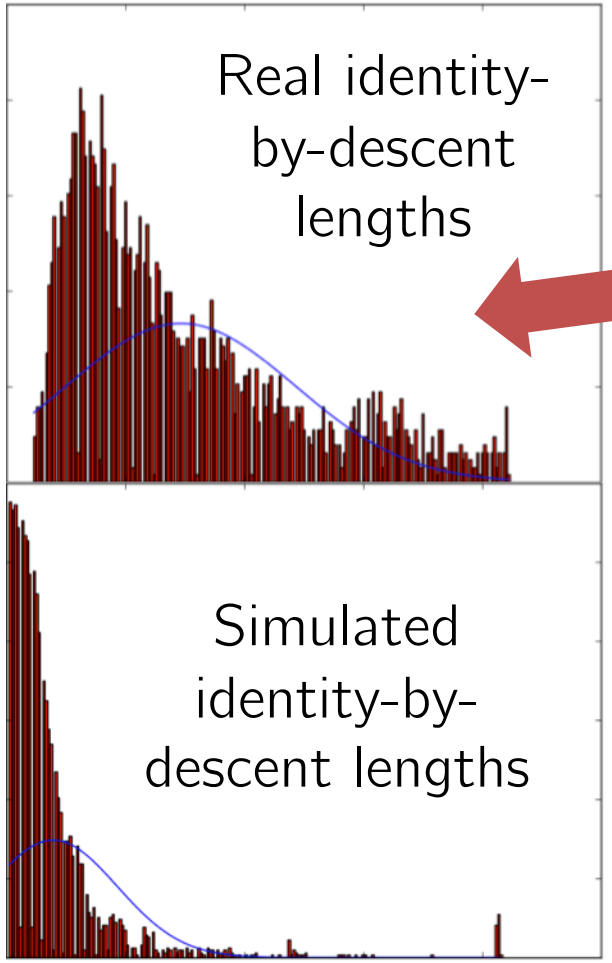
Figure 17-16. Images generated by the DCGAN after 50 epochs of training

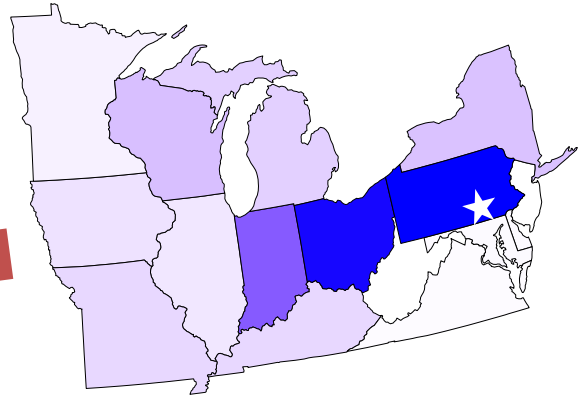
Geron, Chap 17

GAN for biology

High-quality synthetic data is crucial in population genetics

ABC simulations • Compare methods • Develop intuition • Training data for ML methods

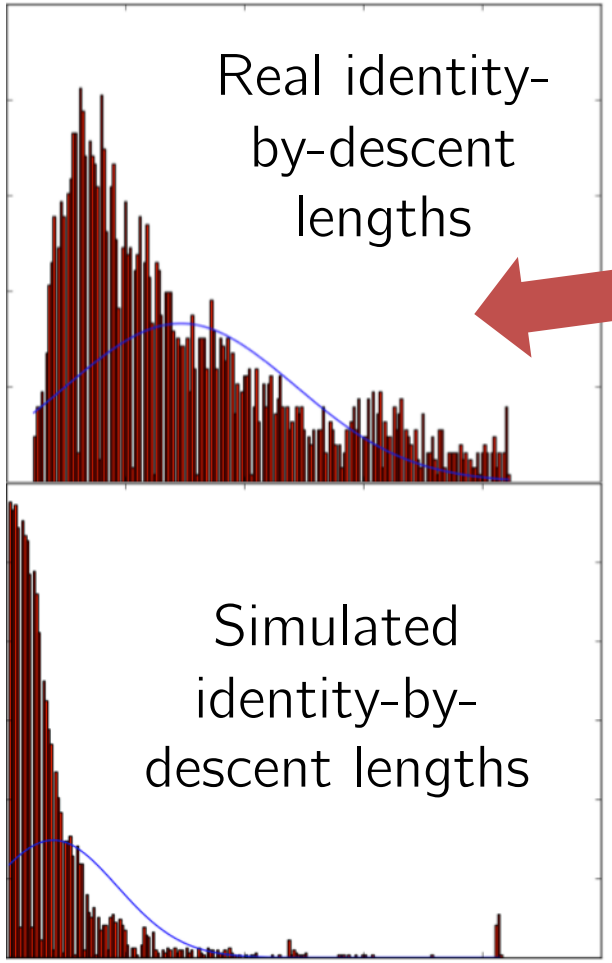




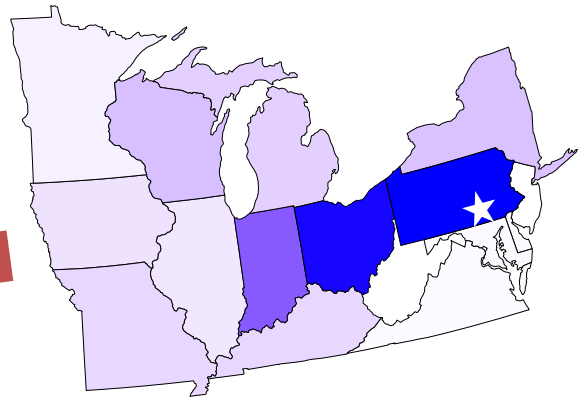
Amish from Lancaster, PA

High-quality synthetic data is crucial in population genetics

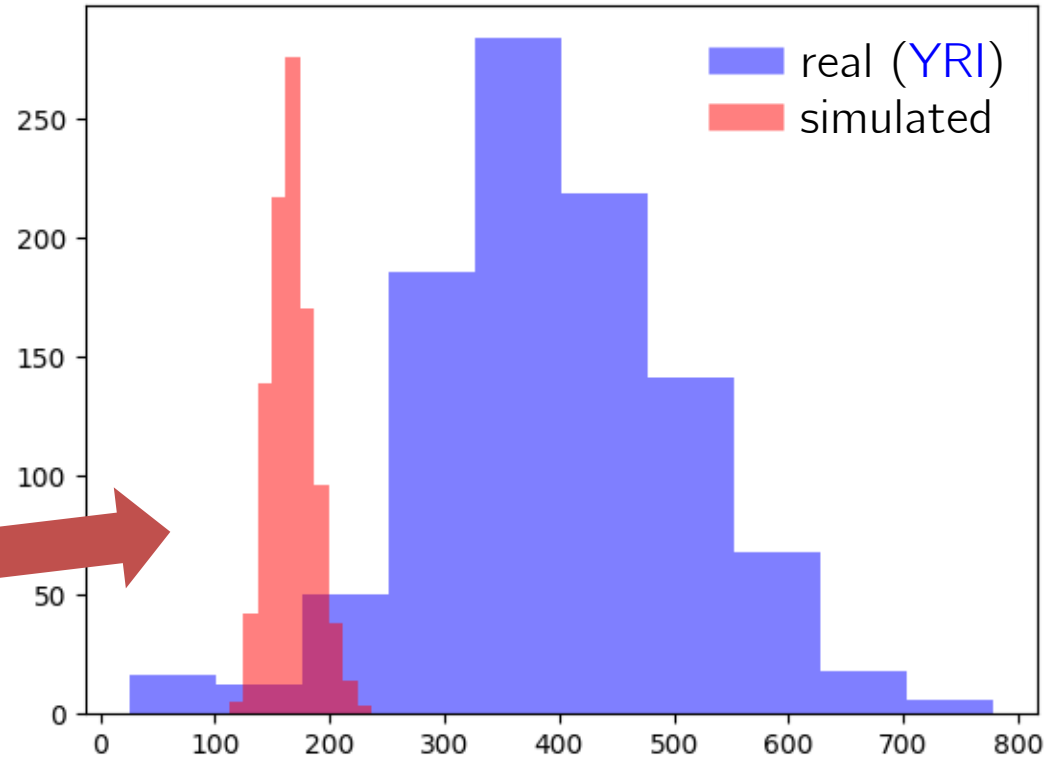
ABC simulations • Compare methods • Develop intuition • Training data for ML methods



Amish from Lancaster, PA



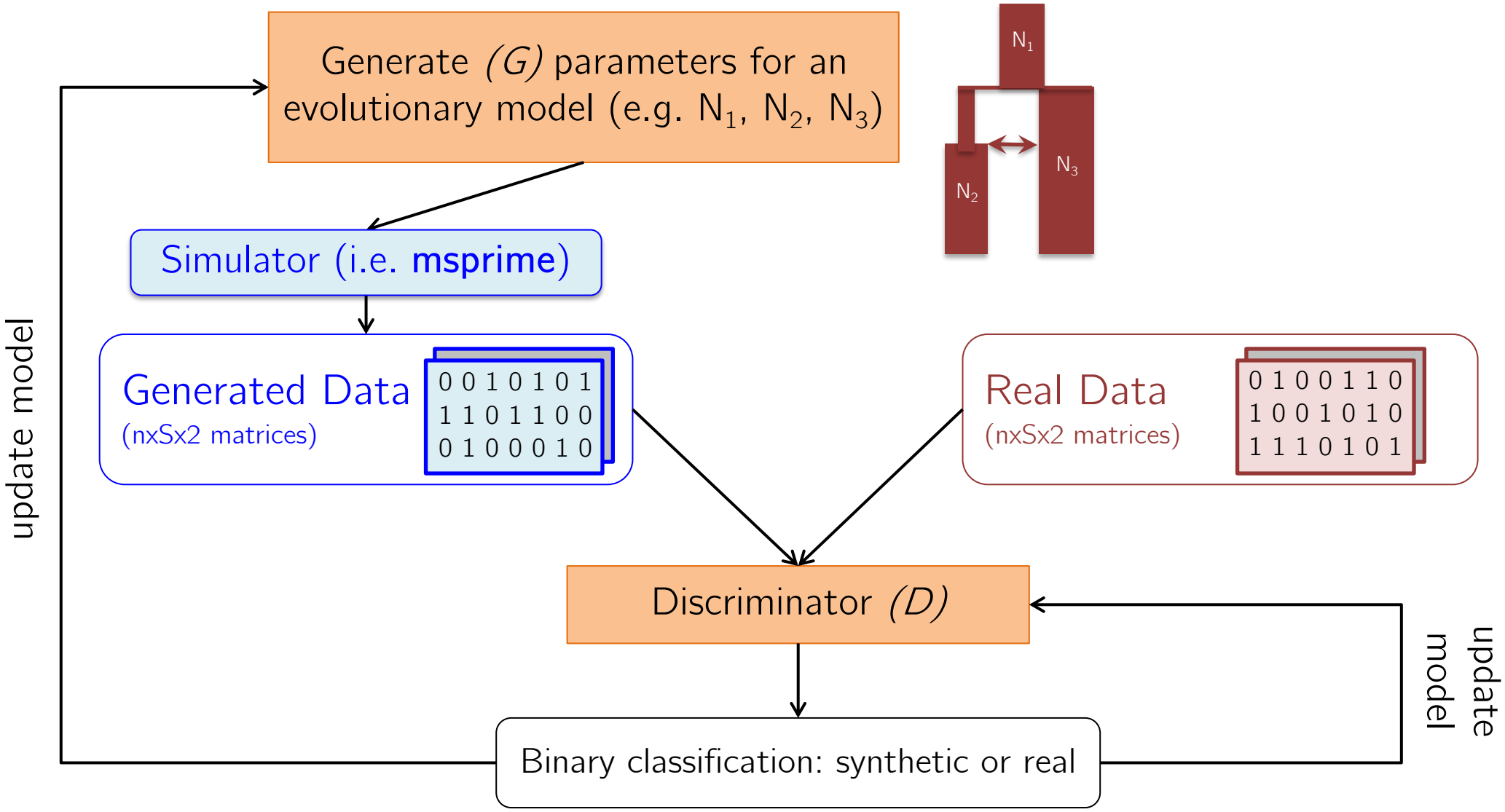
Number of SNPs for simulated/real datasets



YRI: Yoruba in Ibadan, Nigeria

pg-gan algorithm overview

Wang, Wang, Kourakos, Hoang, Lee, Mathieson, Mathieson. "Automatic Inference of Demographic Parameters Using GANs". *MER*, 2021 https://github.com/mathiesonlab/pg-gan



pg-gan loss functions (dual optimization problem)

Regions of simulated data

$$Z = \{z^{(1)}, \cdots, z^{(M)}\}$$

Regions of real data

$$X = \{x^{(1)}, \cdots, x^{(M)}\}$$

Generator Loss

$$\mathcal{L}_G(\Theta) = -\frac{1}{M}\sum_{m=1}^{M} \log \overbrace{D(z^{(m)})}^{\text{Predict fake data as real}}$$

Pair exercise: how does this relate to binary cross entropy loss?

Discriminator Loss

$$\mathcal{L}_D(\Theta, X) = -\frac{1}{M}\sum_{m=1}^{M} \left[\log \overbrace{D(x^{(m)})}^{\text{Predict real data as real}} + \log(1 - \overbrace{D(z^{(m)})}^{\text{Predict fake data as fake}})\right]$$

binary cross-entropy loss

$$H(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i)$$

$D(\cdot)$ = prob of real

if real $\Rightarrow$ $y_i = 1$

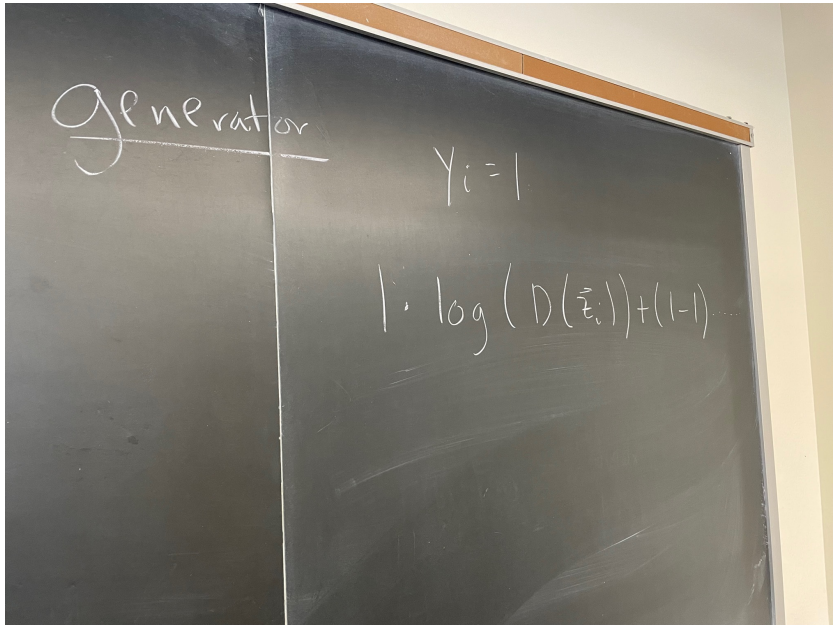
$1 \cdot \log(D(\vec{x}_i)) + (1-1)\log(1-D(\vec{x}_i))$

if fake $\Rightarrow$ $y_i = 0$

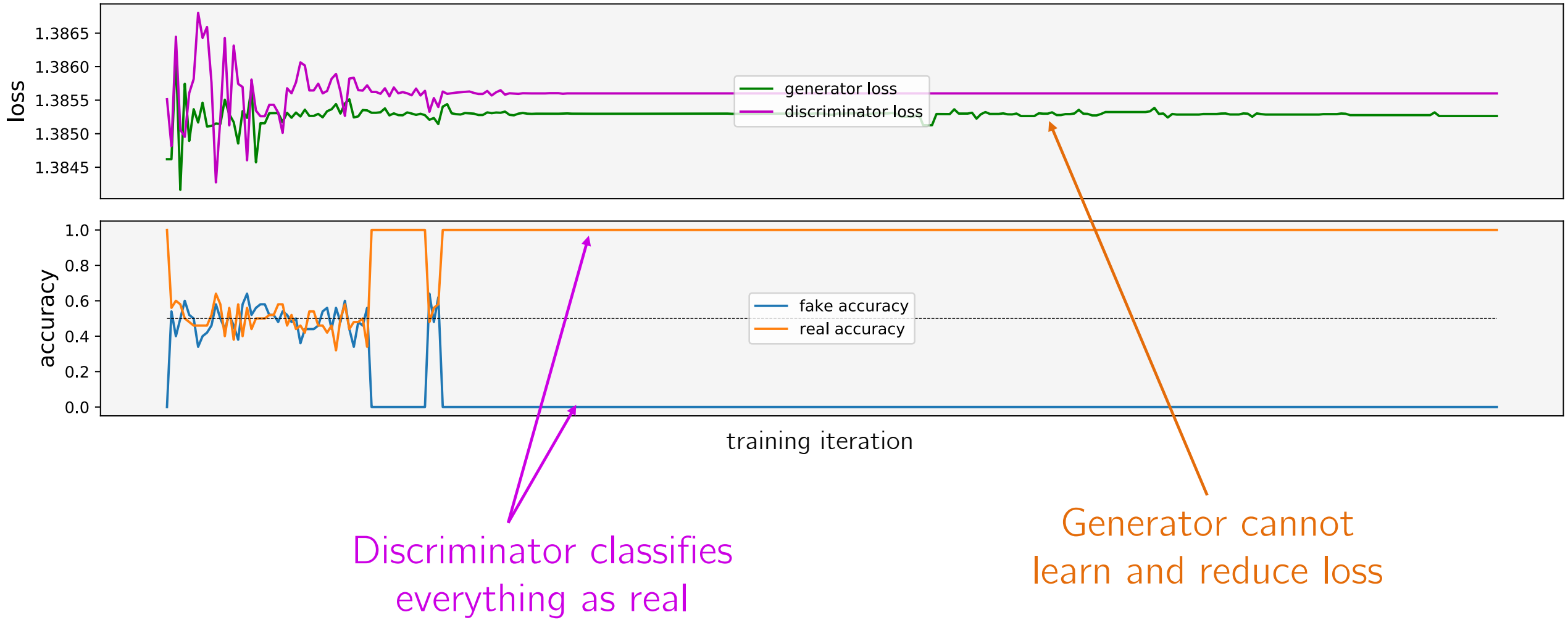
$0 \cdot \log(D(\vec{z}_i)) + (1-0)\log(1-D(\vec{z}_i))$

"true" $\Rightarrow$ 1-hot

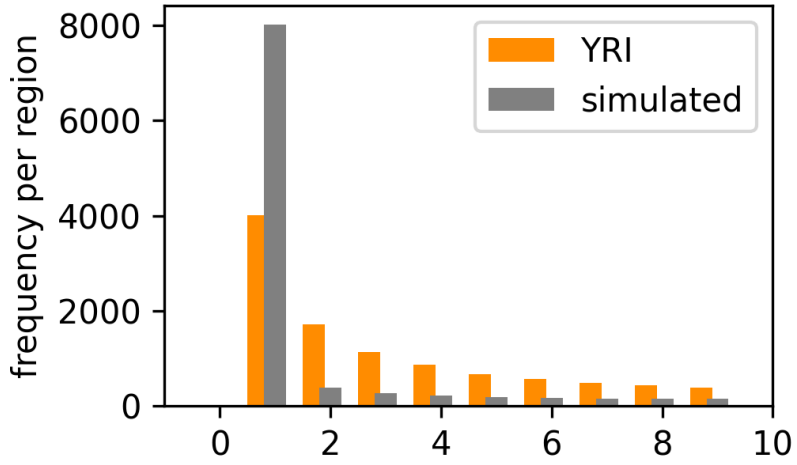
only one term will be non-zero in cross-entropy



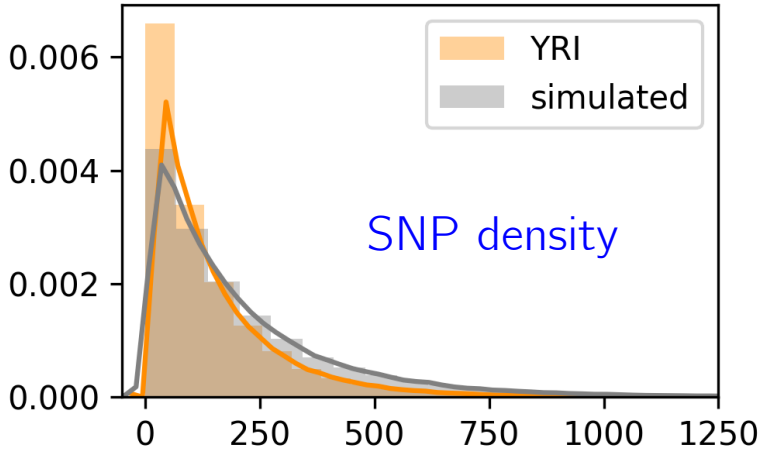
Example of failed GAN training



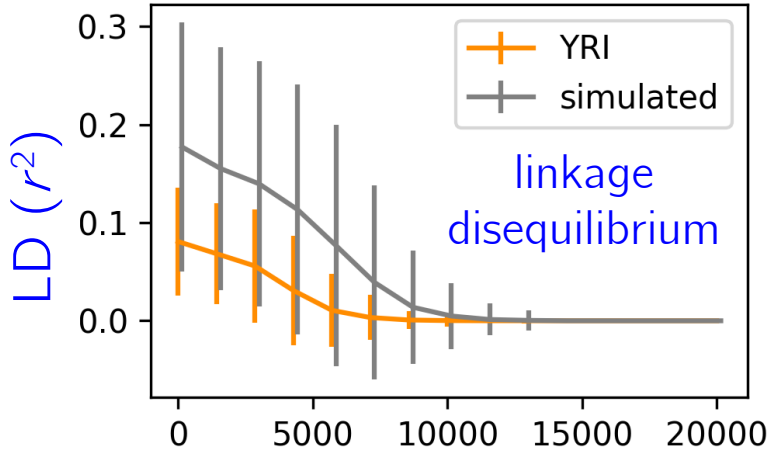
Example of failed GAN training



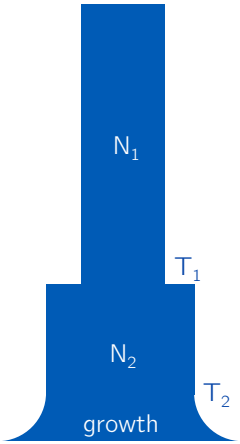
minor allele count (SFS)

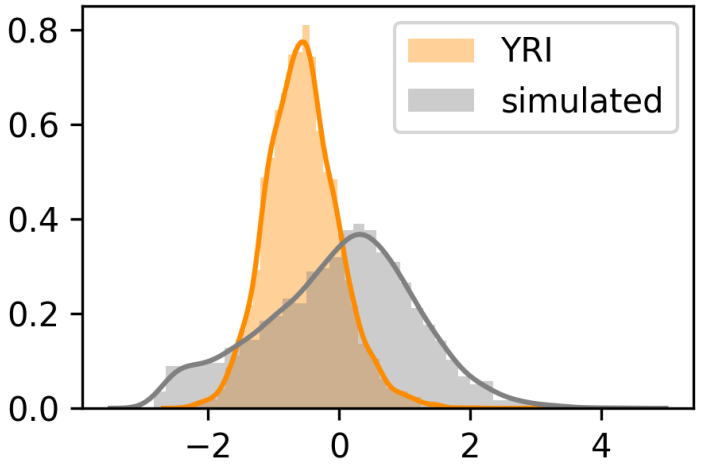


inter-SNP distances

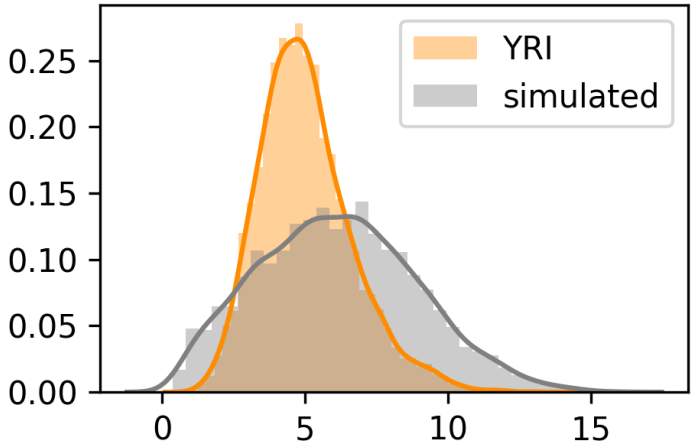


distance between SNPs

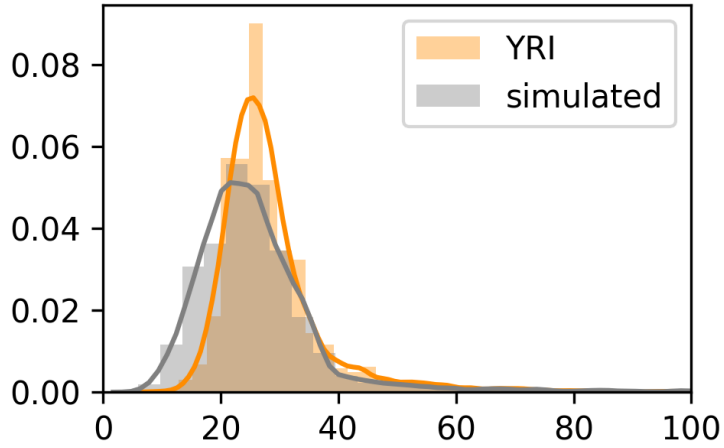




Tajima's D

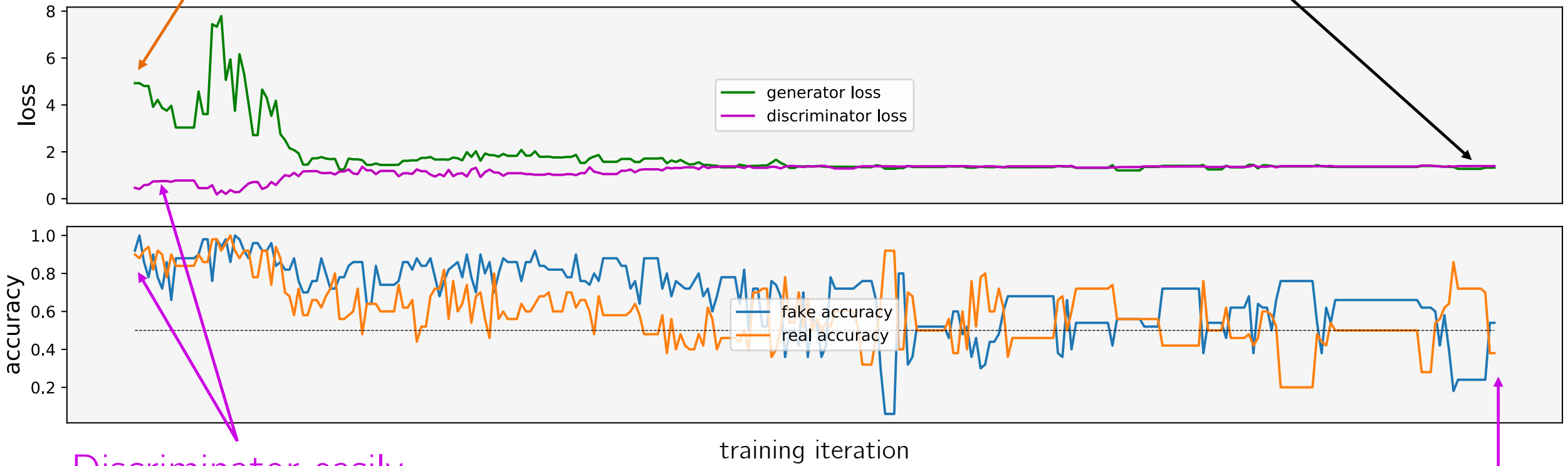


pairwise heterozygosity (π)



number of haplotypes

Example of successful GAN training



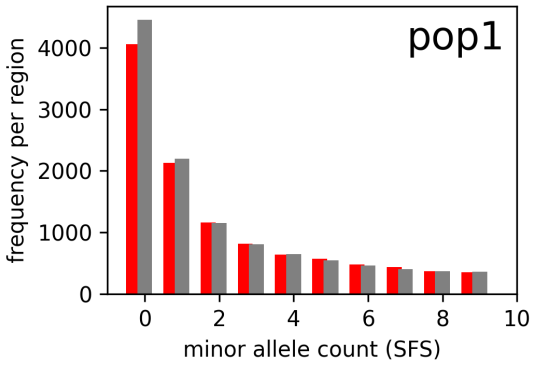
Generator not fooling discriminator

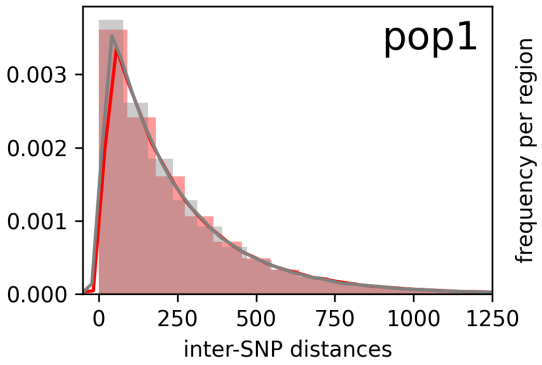
Generator and discriminator are balanced

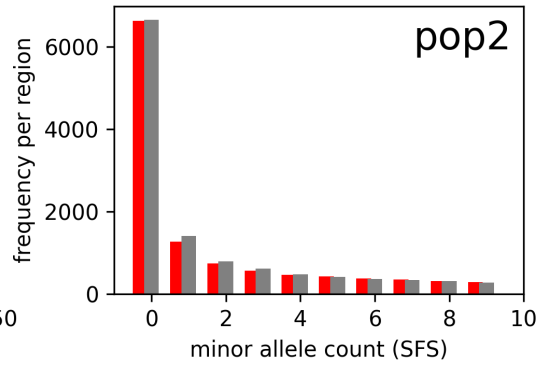
Discriminator easily able to tell training from simulated

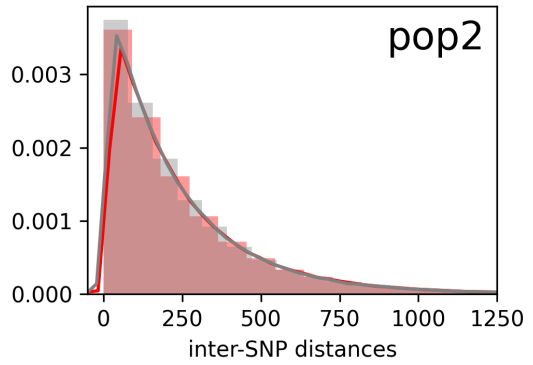
Discriminator is often confused

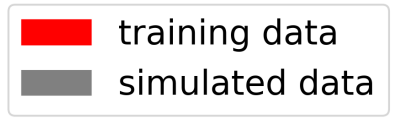
Example of successful GAN training

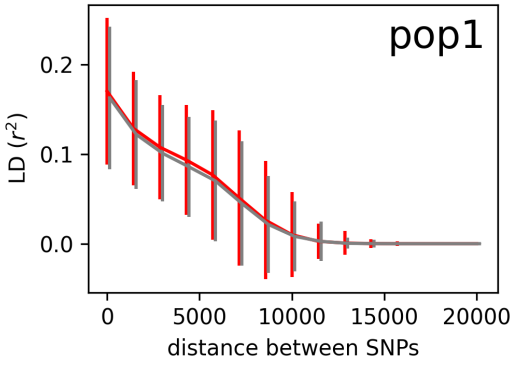


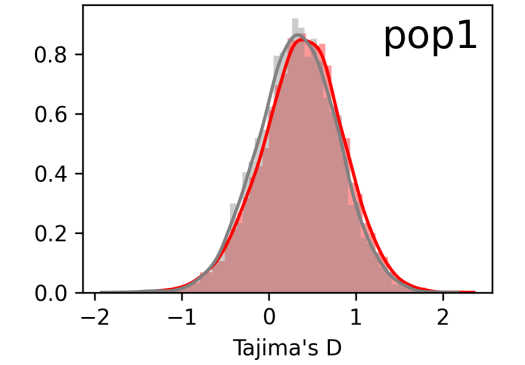


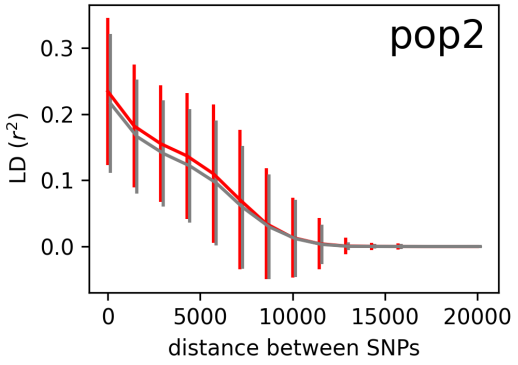


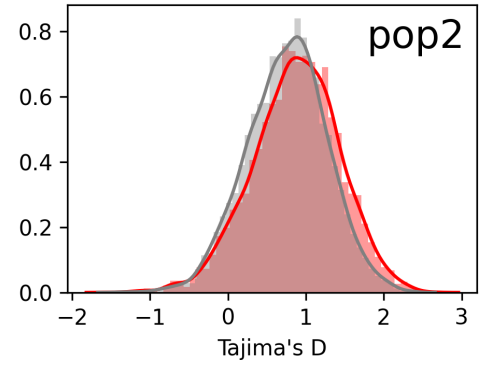


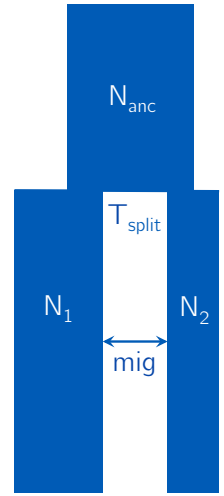


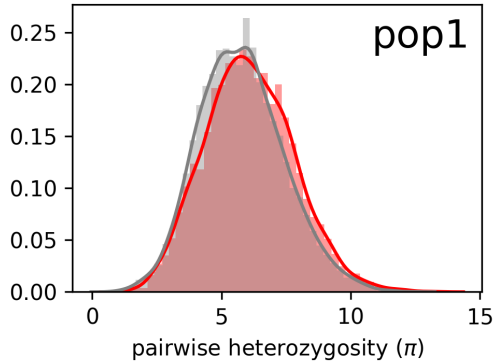


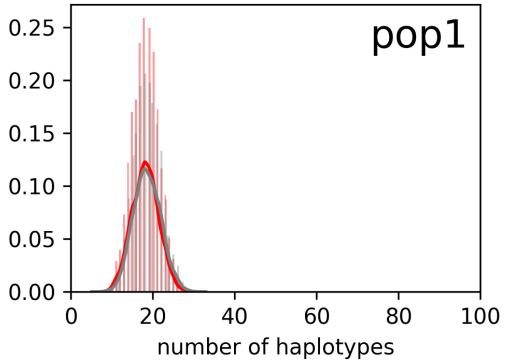


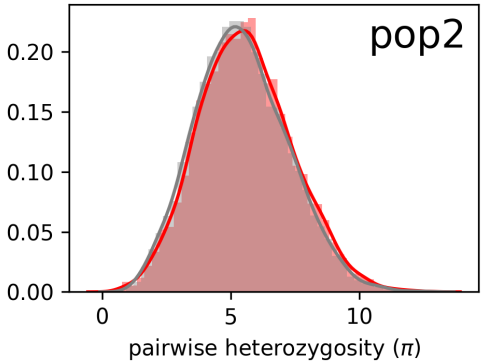


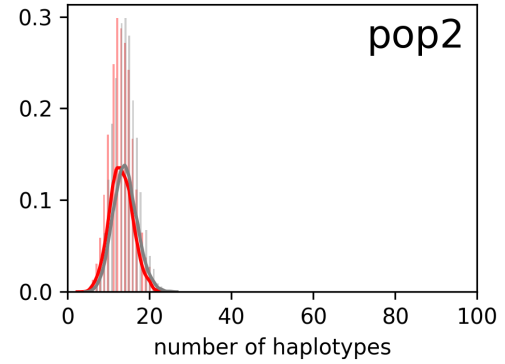


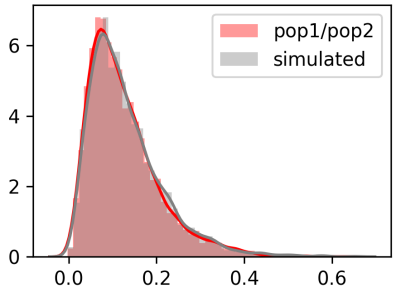






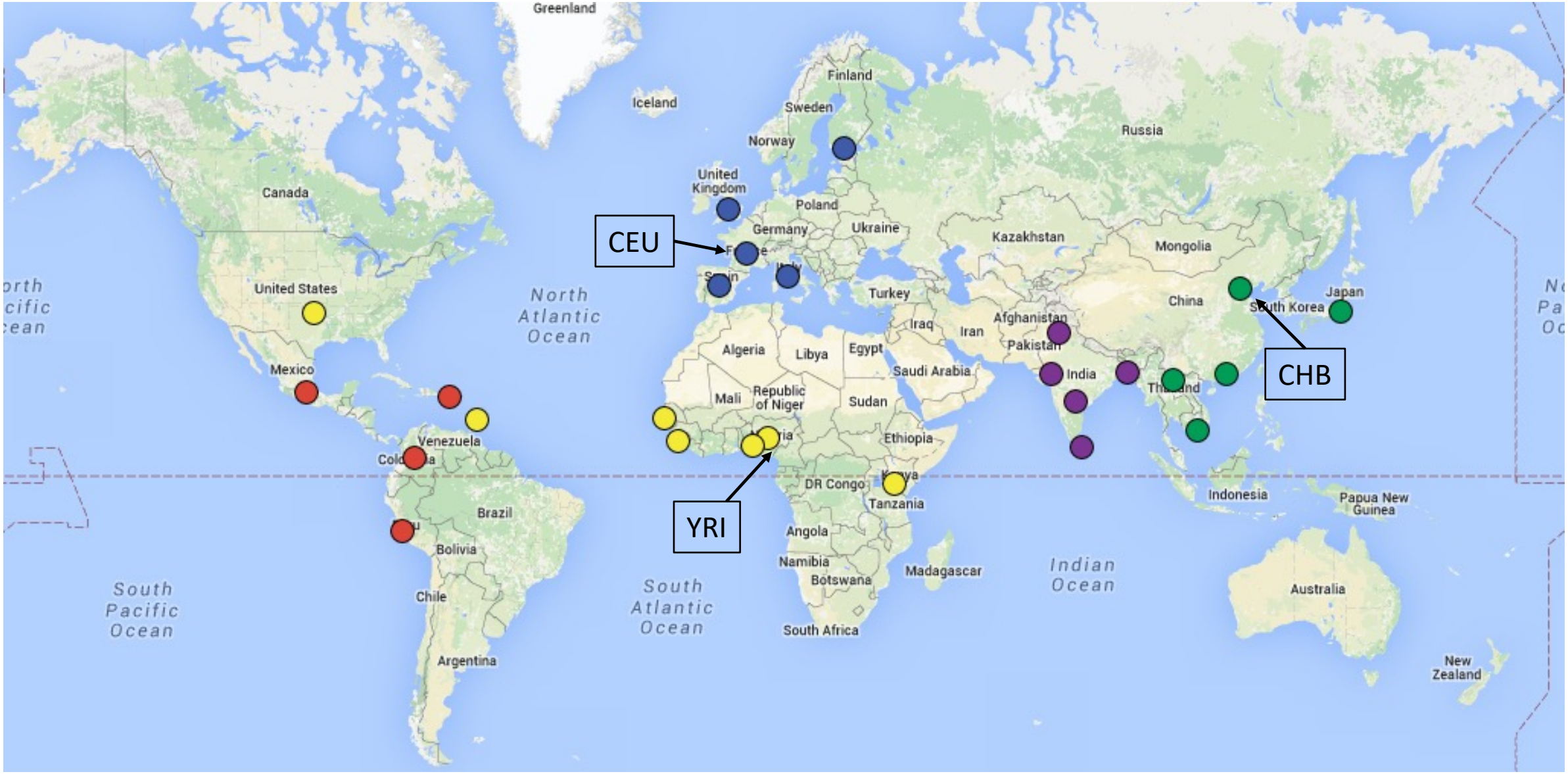




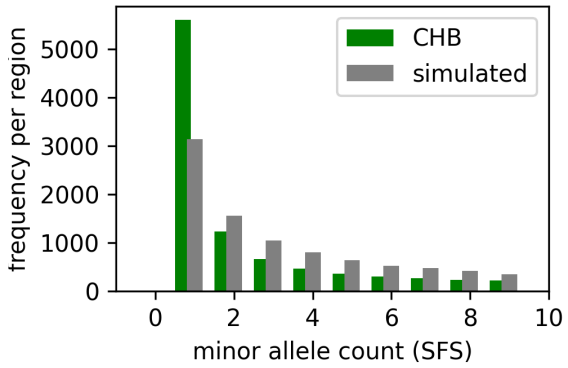


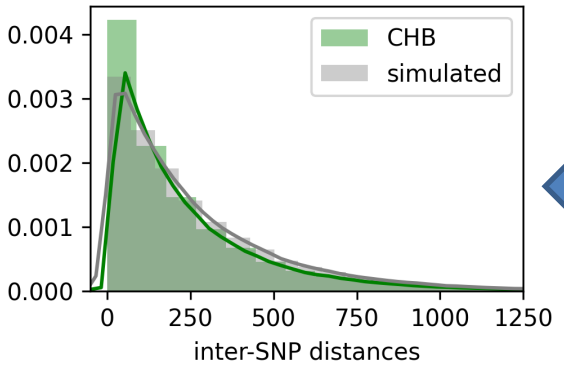
Hudson's F_{st}

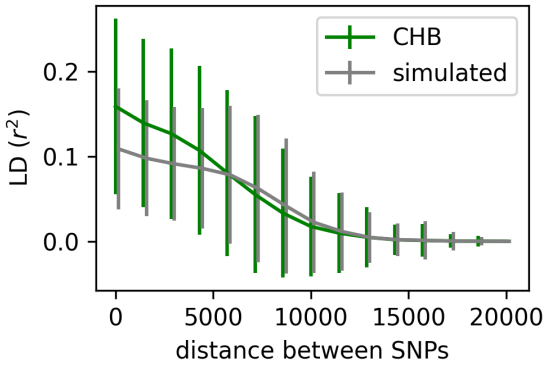
1000 Genomes populations from Africa, Europe, and East Asia

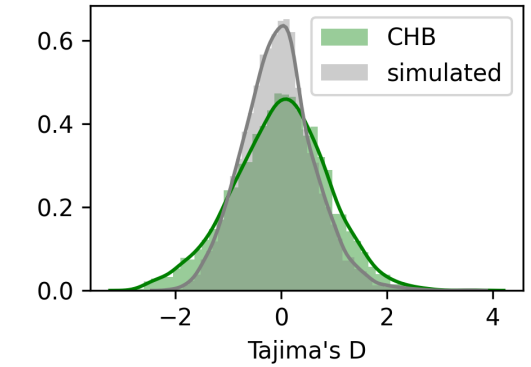


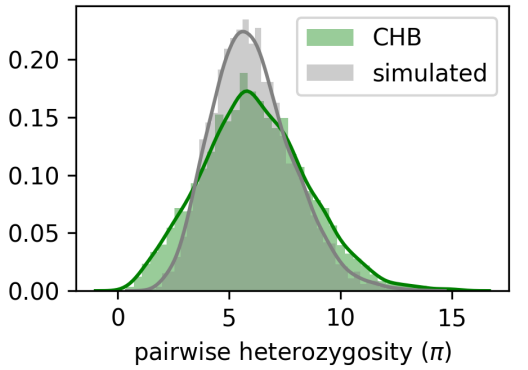
CHB: 1-param model

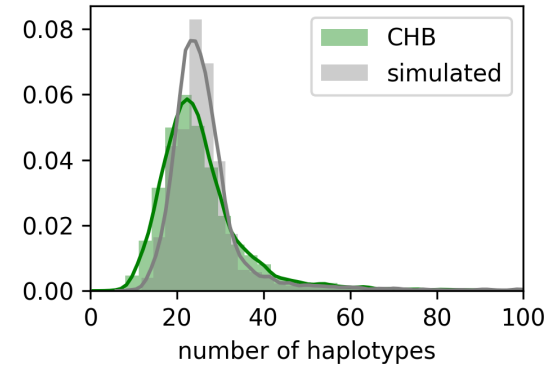


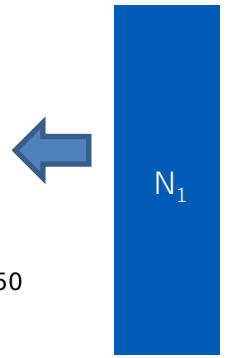


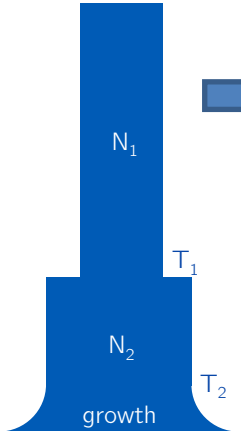




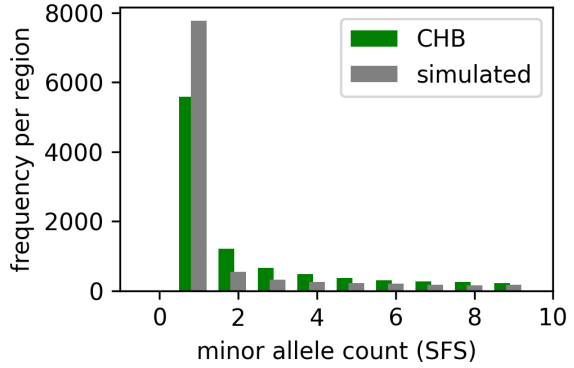


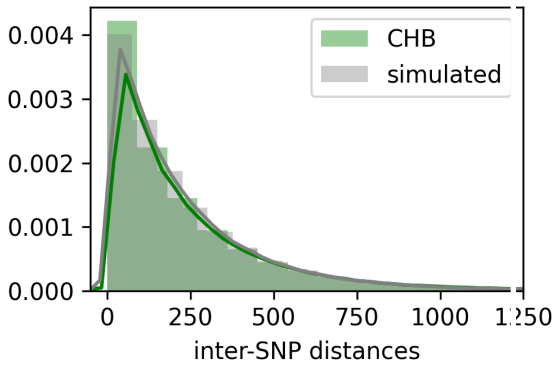


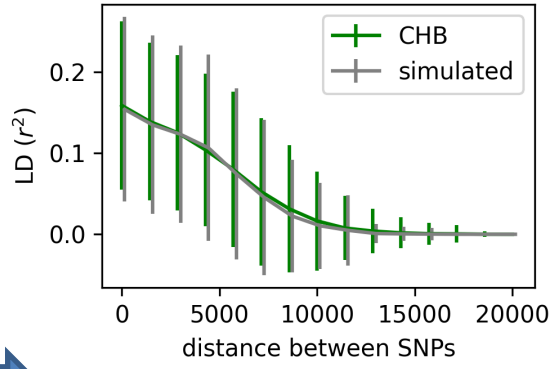


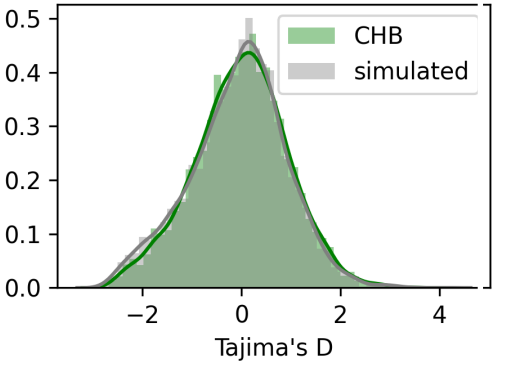


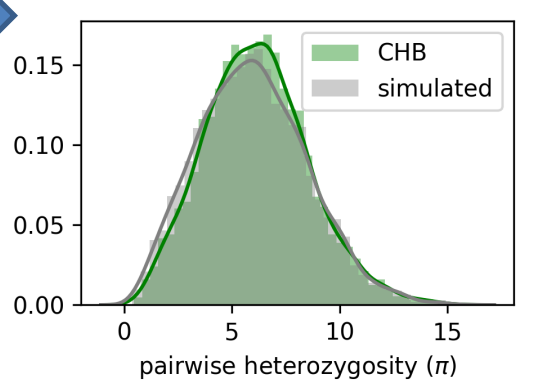
CHB: 5:param model

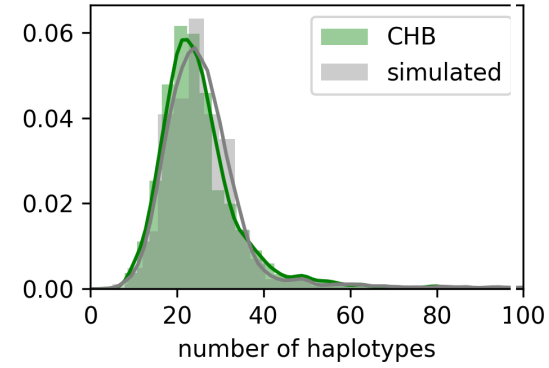




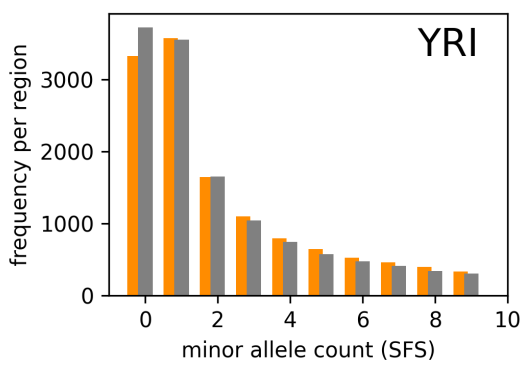


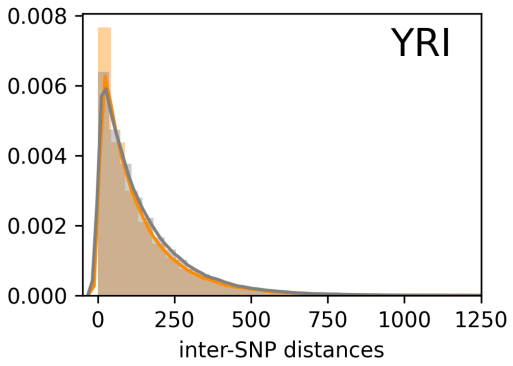


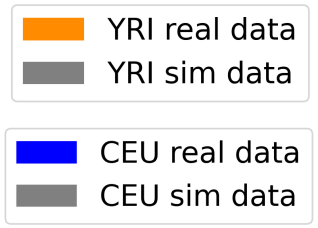


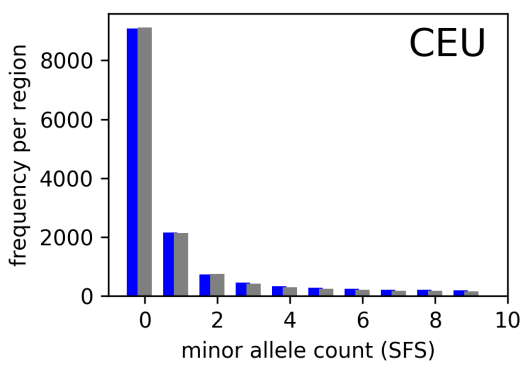


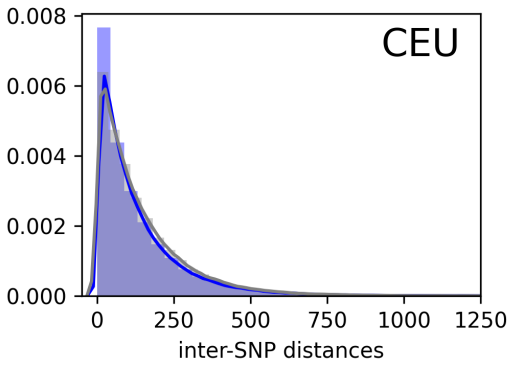
YRI/CEU summary statistics

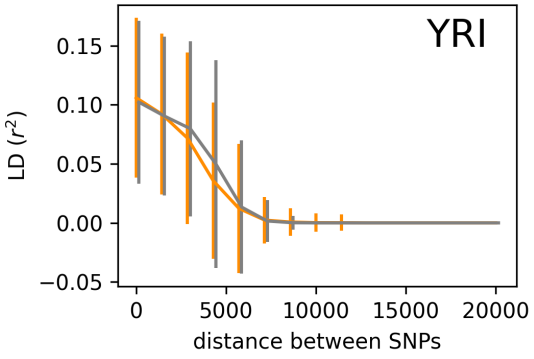


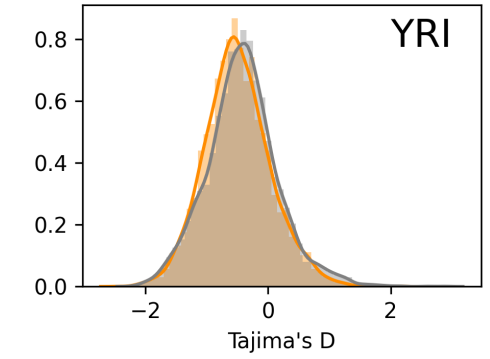


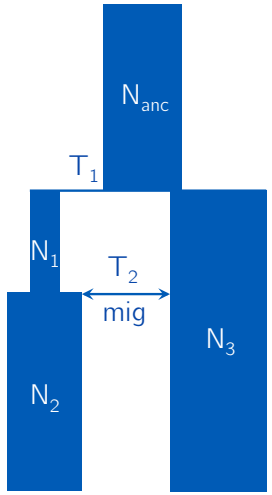


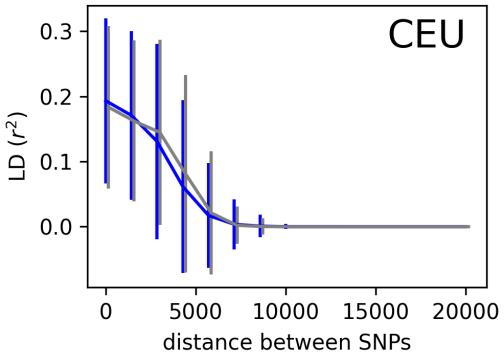


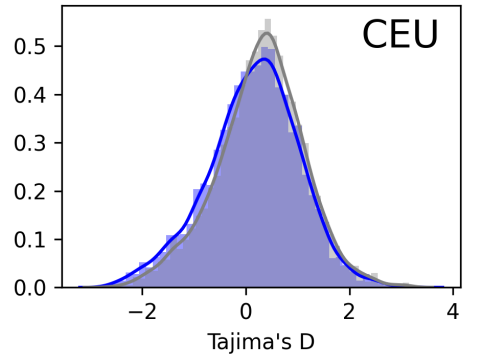


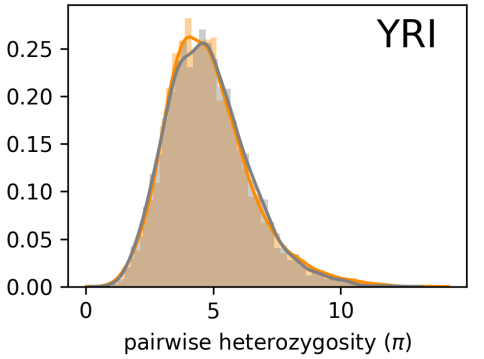


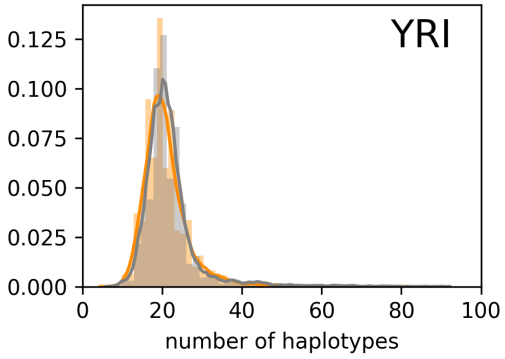


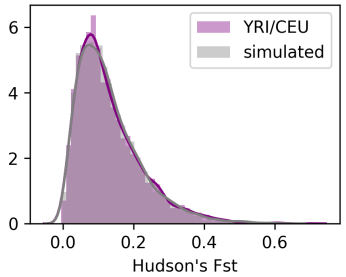


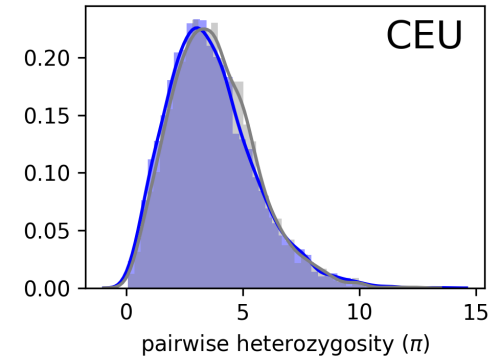


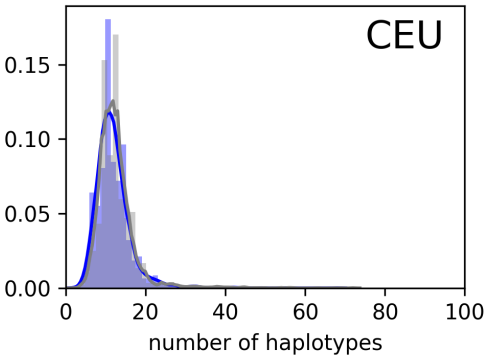












Out-of-Africa 3 (OOA3) summary statistics