# ANFÄNGERPRAKTIKUM
# NEURAL NETWORKS FROM SCRATCH

# CROSS ENTROPY
# AND
# TRANSFORMERS FROM SCRATCH

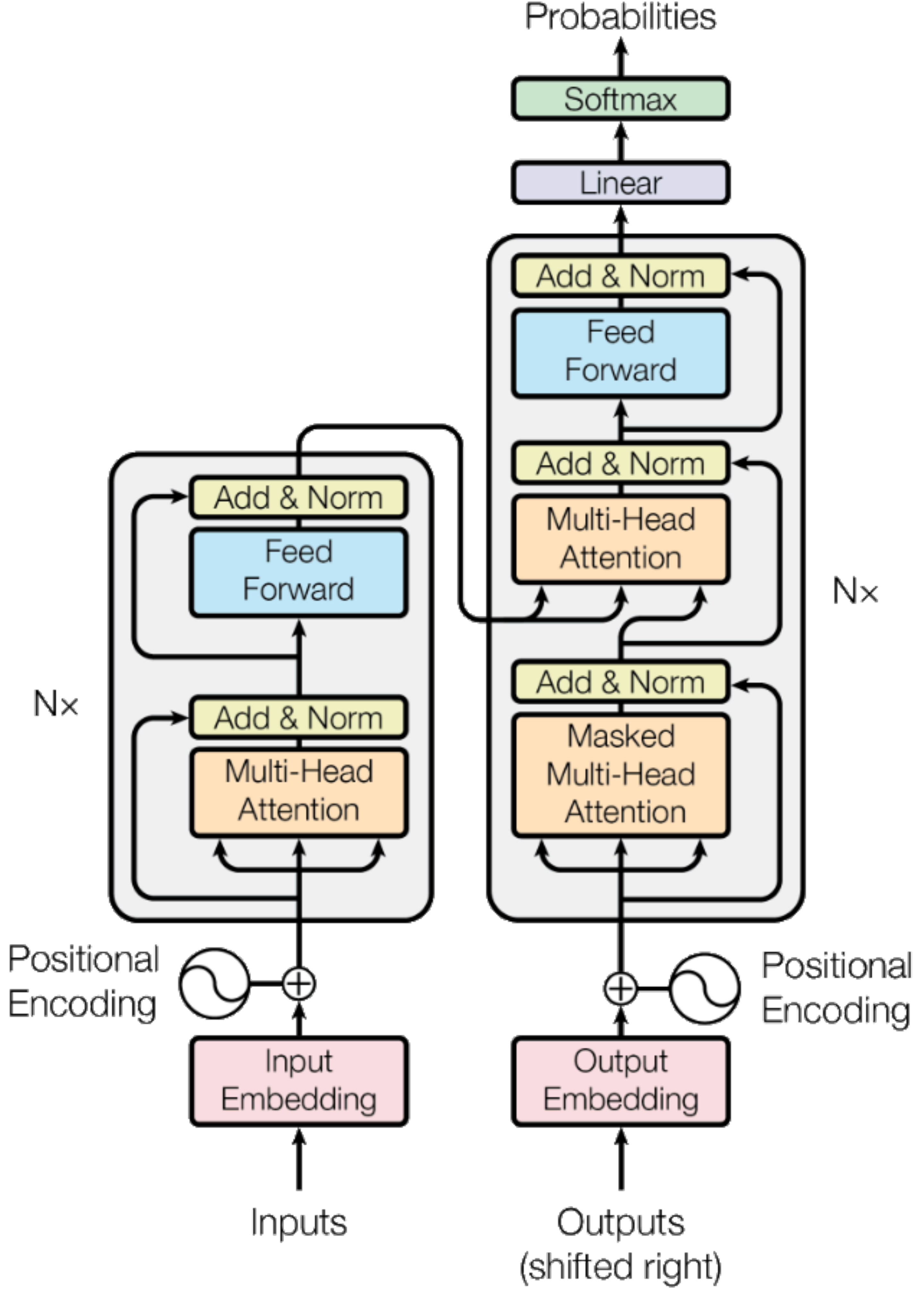
Hendrik Borras, Franz Kevin Stehle
hendrik.borras@ziti.uni-heidelberg.de, kevin.stehle@ziti.uni-heidelberg.de
HAWAII Group, Institute of Computer Engineering
Heidelberg University

# RECAP: TRANSFORMERS

Powerful token predictors

- Much more efficient than previous methods
- Based on self-attention and MLPs
- Are slowly also taking over other domains

Setup around transformers matters



# WHY CROSS ENTROPY?

How do we compare a Softmax vector to ground-truth?

- Intuitively one could just take the difference between prediction and truth
- However: From a stochastics point of view this is badly motivated

Instead: Take inspiration from information theory

- Utilize a function, which maximizes the data likelihood
- Or more intuitively: “Maximize, the probability, that what the Neural Network predicts is what we expect to see.”
- Likelihood is analytically difficult to compute -> Use a proxy

# CROSS ENTROPY ON A SLIDE

Cross entropy idea

How close are two probability distributions to each other?

And how much additional information do we need to transfer from one distribution to the other?

Cross entropy as a proxy

Minimizing cross-entropy is equivalent to maximizing the data likelihood

In the sense that when one is minimal the other is maximal

They are however not directly translatable otherwise

Formulation:

$$-\sum_{c=1}^{N} y_c \cdot \log(p_c)$$

With: $N$: Number of classes, $y_c$: True/False target label per class (1-hot vector), $p_c$: Softmax score per class

# NANO GPT

# NANOGPT MOTIVATION

Small but not outdated Transformer architecture

- Based on GPT-2
- Not SOTA, but understandable

Original implementation: https://github.com/karpathy/nanoGPT

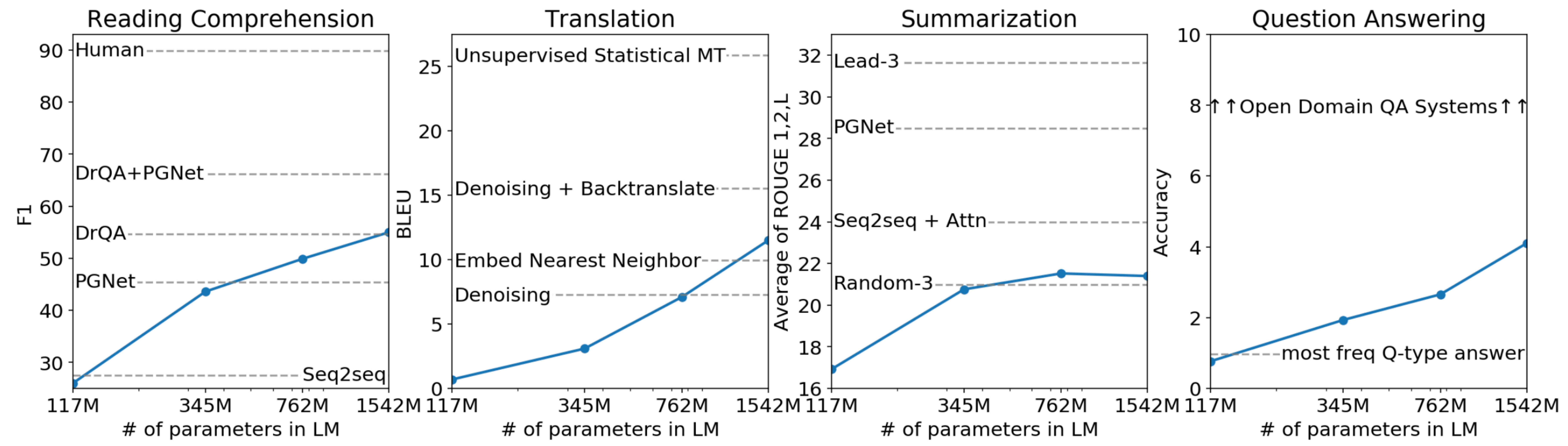
# GPT-2

Showed, that:

- Transformers can learn in a unsupervised setting
- Scales extremely well with the number of parameters

Originally published by OpenAI

- Before they closed-sourced all their models…
- Published in: Radford et al., Language Models are Unsupervised Multitask Learners (2019), https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf



# TRANSFORMER FROM SCRATCH

# WHAT INGREDIENTS DO WE NEED?

Dataset

- Pre-tokenized by Kevin

Cross entropy loss

Additional layers

- Can re-use previous implementation
- New generic: GELU, Dropout, LayerNorm
- New Transformer specific: MultiHead Attention, Embedding
- Composed layers: MLP, Block

Transformer architecture

- Linear
- Sigmoid
- Softmax
- Dropout
- one_hot
- LayerNorm
- GELU
- MLP
- MultiHeadAttention
- Embedding
- Block

Layers used in nanoGPT

# NEW ACTIVATION FUNCTION: GELU

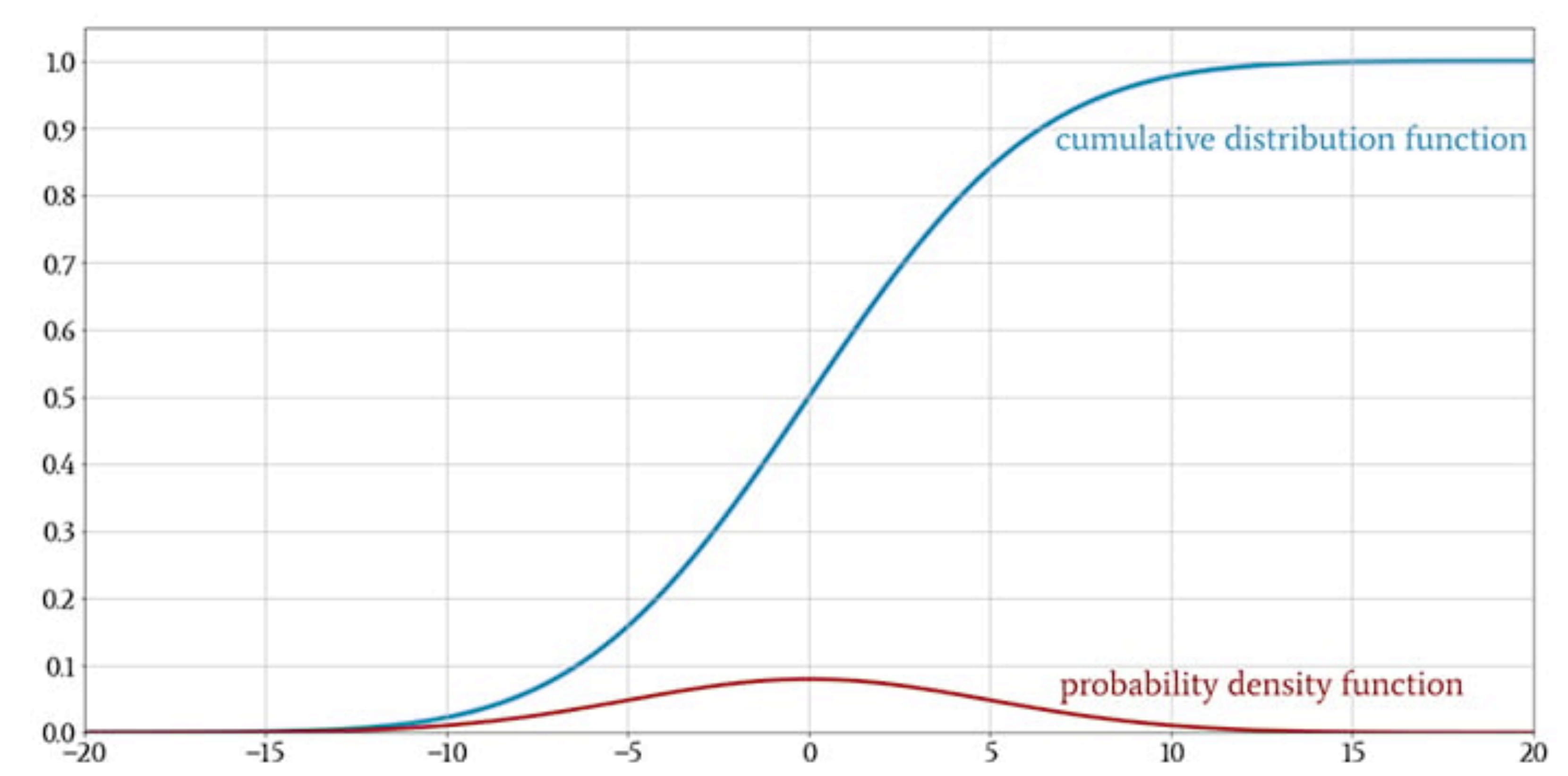
Inspired by very simple Rectified Linear Unit (ReLU:

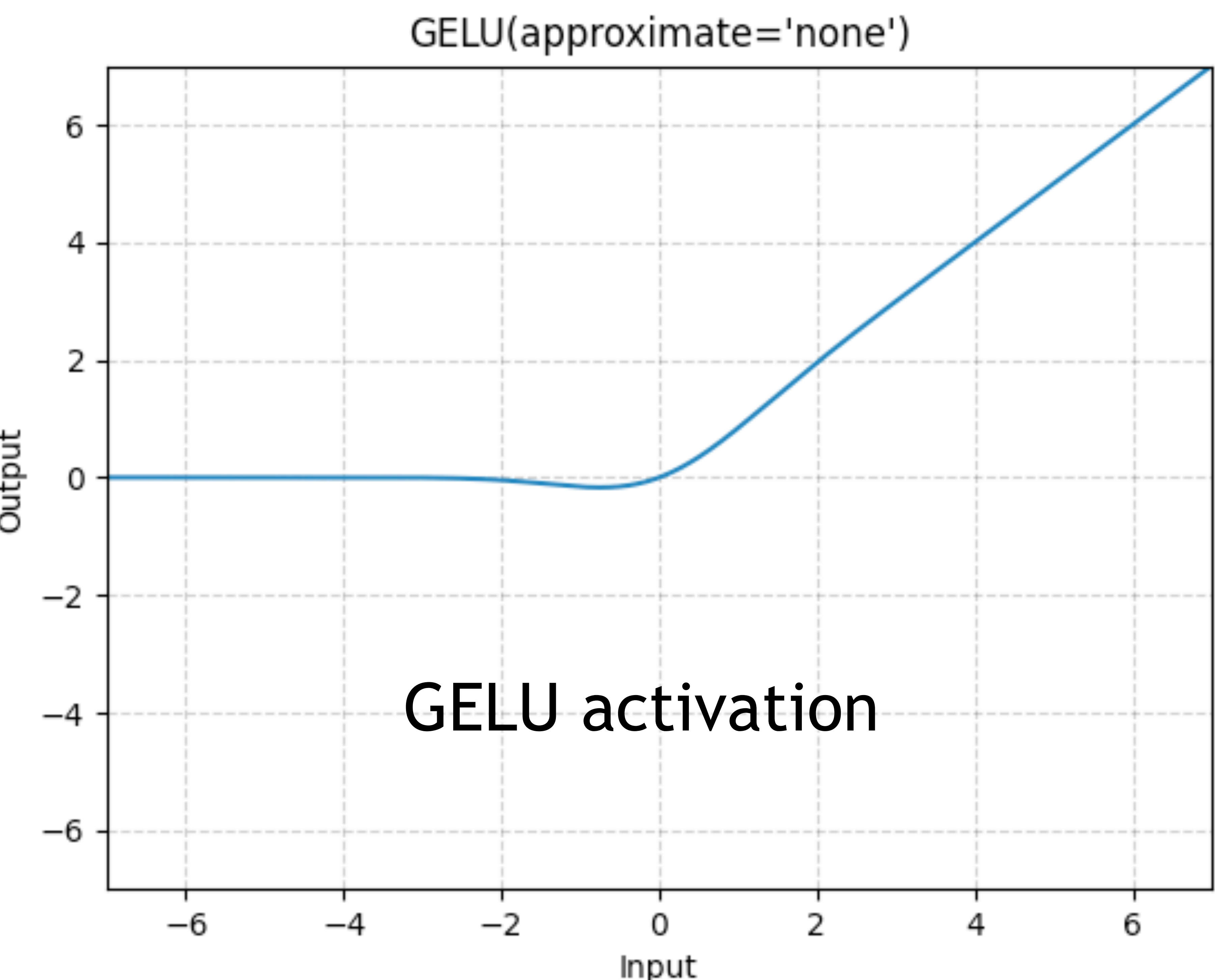
- Simply an if statement:
    - x>0 -> x
    - x<0 -> 0

GELU: Gaussian Error Linear Unit

- Applies the Gaussian Cumulative Distribution multiplied with the input
- Leads to overall better convergence

Disclaimer: A lot of activation function research is extremely emperical



# NEW REGULARIZERS: DROPOUT, LAYERNORM

LayerNorm:

- Normalizes activations in one layer
- Leads to more well-behaved training behavior

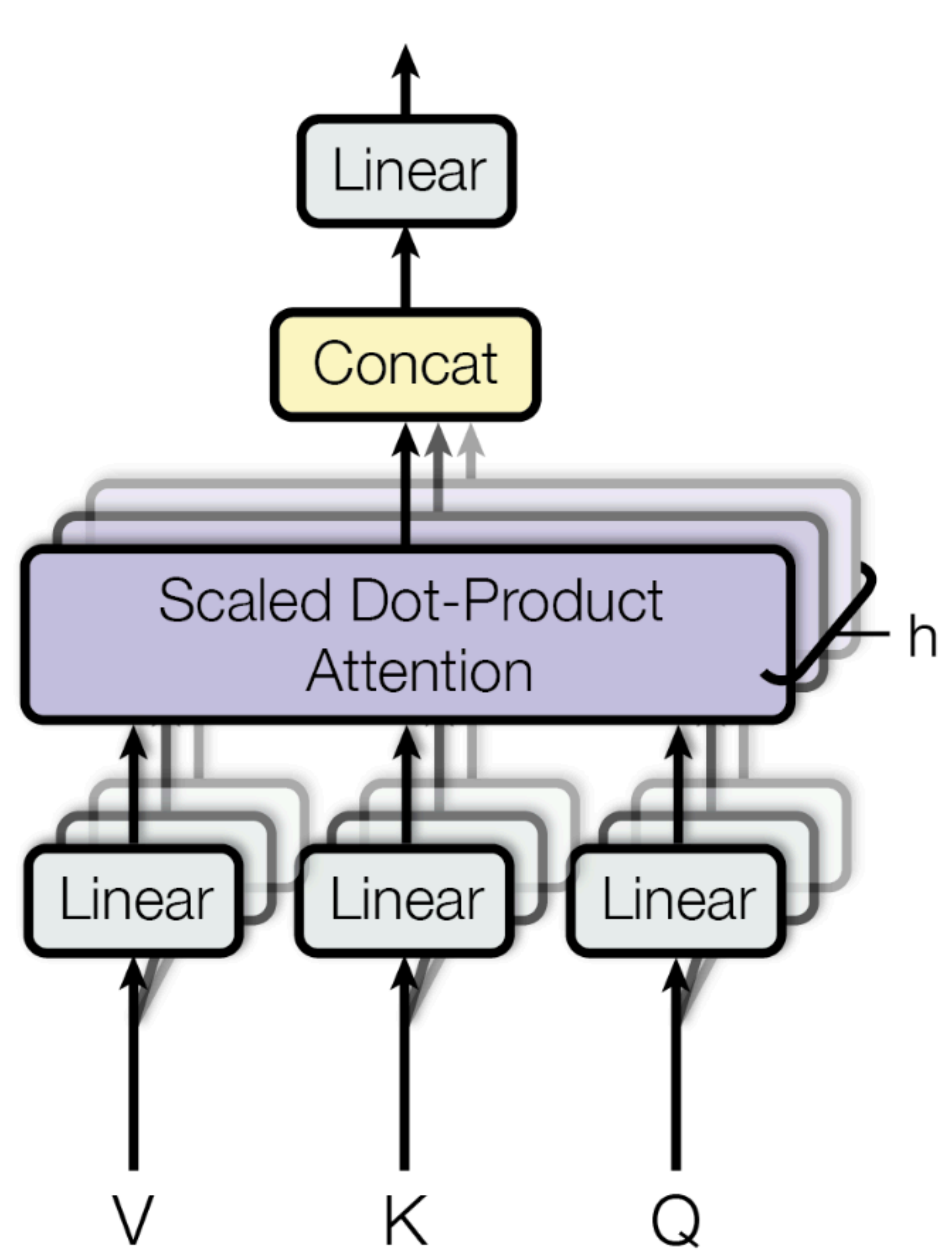
Dropout:

- Randomly deactivate neurons during training
- Network needs to learn to distribute functionality between neurons
- Leads to better generalization

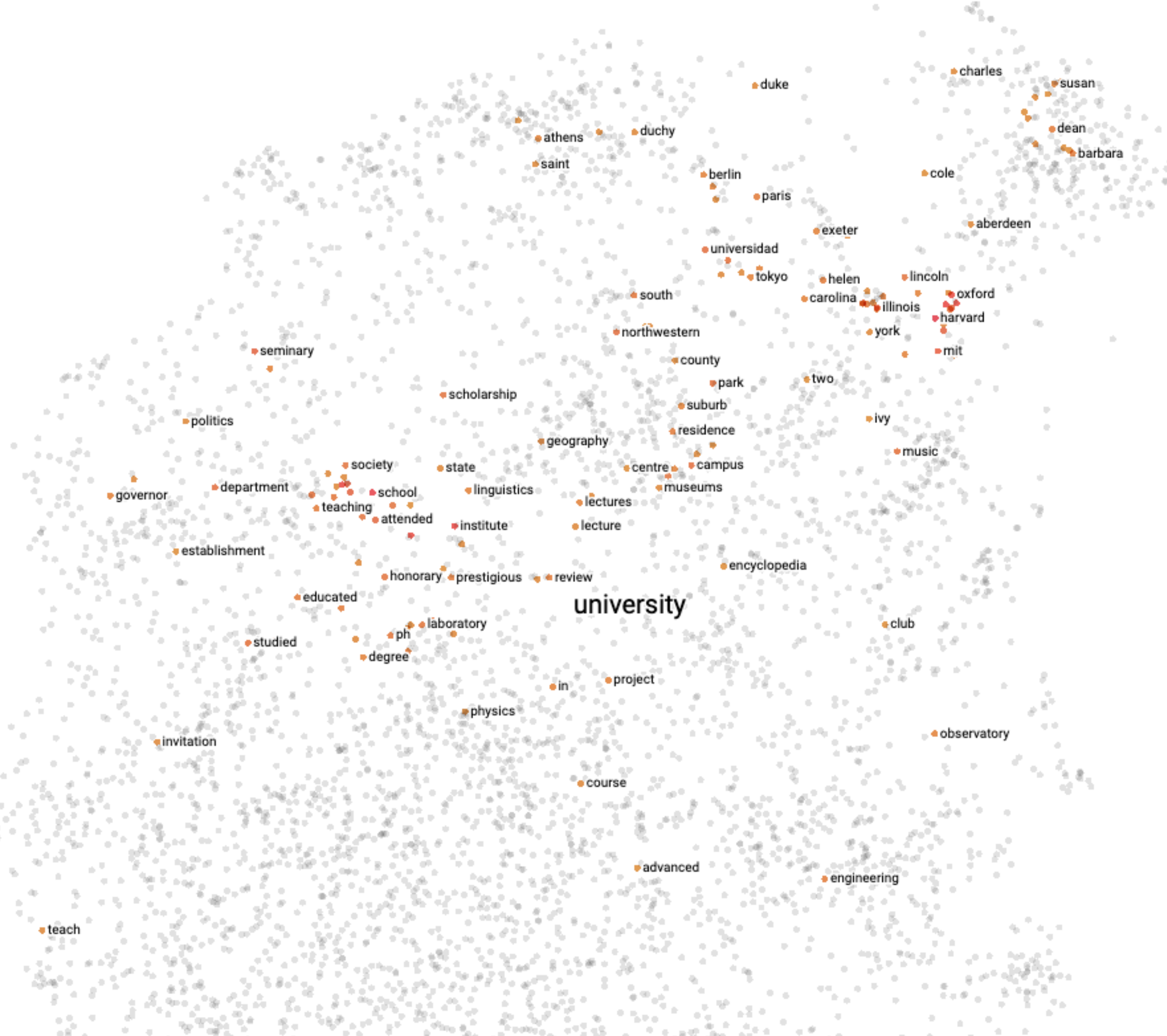
- Linear
- Sigmoid
- Softmax
- Dropout
- one_hot
- LayerNorm
- GELU
- MLP
- MultiHeadAttention
- Embedding
- Block

Layers used in nanoGPT

# NEW TRANSFORMER LAYERS: MULTIHEADATTENTION, EMBEDDING




*Ashish Vaswani et al.: Attention is All You Need, 2017. https://arxiv.org/abs/1706.03762*




Nearest points in the original space:

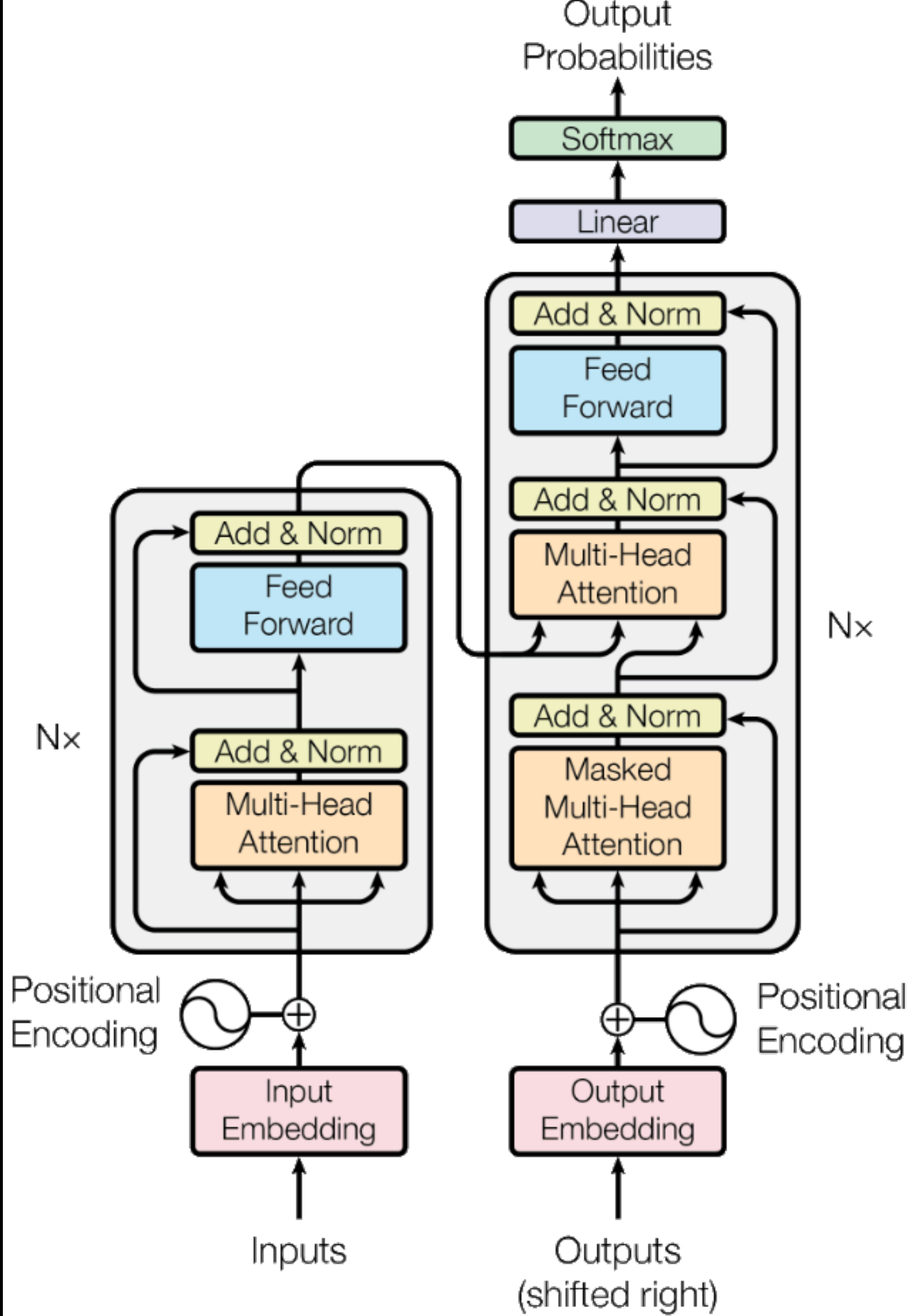
| | |
|---|---|
| college | 0.235 |
| harvard | 0.278 |
| school | 0.294 |
| universities | 0.301 |
| institute | 0.304 |
| cambridge | 0.311 |
| graduate | 0.315 |
| oxford | 0.326 |
| yale | 0.332 |
| stanford | 0.332 |
| professor | 0.335 |
| columbia | 0.365 |
| students | 0.373 |
| berkeley | 0.374 |
| colleges | 0.375 |
| princeton | 0.376 |
| mit | 0.380 |
| faculty | 0.395 |
| undergraduate | 0.395 |
| seminary | 0.395 |
| illinois | 0.398 |
| education | 0.399 |
| academic | 0.401 |
| chicago | 0.402 |
| academy | 0.404 |
| press | 0.408 |
| california | 0.409 |
| attended | 0.412 |
| cornell | 0.414 |
| student | 0.415 |
| arts | 0.418 |

*https://projector.tensorflow.org/*

# GOING THROUGH THE CODE

# TRANSFORMER ARCHITECTURE: LAYERS

```
def __init__(self,
                vocab_size: int=8192,
                context_length: int=256,
                batch_size: int=64,
                n_layer: int=6,
                n_embd: int=384,
                dropout: float=0.2,
                lr: float=1e-3) -> None:
    self.vocab_size = vocab_size
    …

    # Define lm_head first so we can pass its
    # weights_transposed property to the wte
    # embedding to implement weight tying

    self.lm_head = scr.Linear(self.n_embd,
                               self.vocab_size,
                               self.batch_size,
                               bias=False,
                               lr=self.lr,
                               weight_init_func=weight_init,
                               bias_init_func=bias_init)

    self.transformer = {
        "wte": scr.Embedding(self.vocab_size, self.n_embd, self.batch_size, self.lr,
weight_external=self.lm_head.weight_transposed),

        "wpe": scr.Embedding(self.context_length, self.n_embd, self.batch_size, self.lr,
init_func=weight_init),
        "drop": scr.Dropout(self.dropout),
        "h": [scr.Block(self.n_embd, self.context_length, 6, self.batch_size, self.lr, self.dropout,
weight_init, c_proj_weight_init, bias_init) for _ in range(self.n_layer)],

        "ln_f": scr.LayerNorm(self.n_embd, weight_init_func=weight_init),
        }
```



# TRANSFORMER ARCHITECTURE: FORWARD

```
def forward(self, idx, targets=None):
    b, t = idx.shape
    assert t <= self.context_length, f"Cannot forward sequence of length {t}, block size is only {self.context_length}"
    pos = np.arange(0, t, dtype=np.int64) # shape (t)

    # Forward the GPT model itself
    # Token embeddings of shape (b, t, n_embd)
    tok_emb = self.transformer['wte'].forward(idx)

    # Position embeddings of shape (t, n_embd)
    pos_emb = self.transformer['wpe'].forward(pos)

    # Main transformer
    x = self.transformer['drop'].forward(tok_emb + pos_emb)
    for block in self.transformer['h']:
        x = block.forward(x)
    x = self.transformer['ln_f'].forward(x)

    logits = self.lm_head.forward(x)
    # Compute loss and return
    return logits, loss
```

# BLOCK: LAYERS & FORWARD

```python
def __init__(self,
                d_model: int,
                context_size: int,
                n_heads: int,
                batch_size: int,
                lr: float,
                dropout: float,
                weight_init_func: Union[Callable, None],
                c_proj_init_func: Union[Callable, None],
                bias_init_func: Union[Callable, None]) -> None:

    self.d_model = d_model
    …

    self.ln_1 = LayerNorm(d_model,
                            weight_init_func=weight_init_func)

    self.attn = MultiHeadAttention(d_model,
                                        context_size,
                                        n_heads,
                                        batch_size,
                                        lr,
                                        dropout,
                                        c_attn_weight_init_func=weight_init_func,
                                        c_proj_weight_init_func=c_proj_init_func,
                                        bias_init_func=bias_init_func)

    self.ln_2 = LayerNorm(d_model,
                            weight_init_func=weight_init_func)

    self.mlp = MLP(d_model,
                        batch_size,
                        lr,
                        dropout,
                        c_fc_init_func=weight_init_func,
                        c_proj_init_func=c_proj_init_func,
                        bias_init_func=bias_init_func)
```

```python
def forward(self, input: ArrayLike) -> np.ndarray:

    input = np.asanyarray(input)

    x = self.ln_1.forward(input)
    x = self.attn.forward(x)[0]

    x = input + x

    residual = copy.deepcopy(x)

    x = self.ln_2.forward(x)
    x = self.mlp.forward(x)
    x = residual + x

    return x
```

# MLP: LAYERS & FORWARD

```
def __init__(self,
                d_model,
                batch_size,
                lr,
                dropout,
                c_fc_init_func,
                c_proj_init_func,
                bias_init_func):

    self.d_model = d_model
    …

    self.c_fc = Linear(d_model,
                            4*d_model,
                            batch_size,
                            lr,
                            weight_init_func=c_fc_init_func,
                            bias_init_func=bias_init_func)
    self.gelu = GELU()

    self.c_proj = Linear(4*d_model,
                            d_model,
                            batch_size,
                            lr,
                            weight_init_func=c_proj_init_func,
                            bias_init_func=bias_init_func)

    self.dropout = Dropout(dropout)
```

```
def forward(self, x: np.ndarray) -> np.ndarray:
    x = self.c_fc.forward(x)
    x = self.gelu.forward(x)
    x = self.c_proj.forward(x)
    x = self.dropout.forward(x)
    return x
```

# MULTIHEAD ATTENTION: LAYERS

```
def __init__(self, d_model: int,
                   context_size: int,
                   n_heads: int,
                   batch_size: int,
                   lr: float=0.1,
                   dropout: float=0.1,
                   c_attn_weight_init_func: Union[Callable, None] = None,
                   c_proj_weight_init_func: Union[Callable, None] = None,
                   bias_init_func: Union[Callable, None] = None) -> None:

    self.d_model = d_model
    …

    self.attn_dropout = Dropout(dropout)
    self.resid_dropout = Dropout(dropout)
    self.softmax_attn = Softmax(axis=-1)

    self.depth = d_model//n_heads

    self.c_attn = Linear(d_model,
                         3*d_model,
                         batch_size,
                         lr,
                         weight_init_func=c_attn_weight_init_func,
                         bias_init_func=bias_init_func)

    self.c_proj = Linear(d_model,
                         d_model,
                         batch_size,
                         lr,
                         weight_init_func=c_proj_weight_init_func,
                         bias_init_func=bias_init_func)

    self.mask = np.tril(np.ones((context_size, context_size), dtype=np.float32)).reshape(1, 1, context_size, context_size)
```

# MULTIHEAD ATTENTION: FORWARD

```
def forward(self, input: ArrayLike) -> tuple:

    self.input = np.asanyarray(input)

    B, T, C = self.input.shape

    q, k, v  = np.split(self.c_attn.forward(self.input), 3, axis=2)

    k = k.reshape((B, T, self.n_heads, C//self.n_heads)).transpose(0, 2, 1, 3) # (B, nh, T, hs)
    q = q.reshape((B, T, self.n_heads, C//self.n_heads)).transpose(0, 2, 1, 3) # (B, nh, T, hs)
    v = v.reshape((B, T, self.n_heads, C//self.n_heads)).transpose(0, 2, 1, 3) # (B, nh, T, hs)

    self.k = k
    self.q = q
    self.v = v

    attn = (q @ k.transpose(0, 1, 3, 2))*(1.0/math.sqrt(k.shape[-1]))

    attn = np.where(self.mask == 0, -1e9, attn)
    attn = self.softmax_attn.forward(attn)
    attn = self.attn_dropout.forward(attn)

    self.attn = attn

    x = attn @ v

    x = np.ascontiguousarray(x).transpose(0, 2, 1, 3).reshape(self.batch_size, -1, self.n_heads*self.depth)
    x = self.c_proj.forward(x)
    x = self.resid_dropout.forward(x)

    return x, attn
```

# EMBEDDING: LAYERS & FORWARD

```
def __init__(self, num_embeddings: int,
                    embedding_dim: int,
                    batch_size: int,
                    lr: float,
                    init_func: Union[Callable, None] = None,
                    weight_external = None):

    self.rng = np.random.default_rng()
    …

    # If we get external weights passed, use them
    # instead of allocating ones on our own.
    # This is used for implementing weight tying.
    #
    # https://paperswithcode.com/method/weight-tying

    if not isinstance(type(weight_external), NoneType):
        if self.init_func:
            self.weight = np.asanyarray(self.init_func((num_embeddings, embedding_dim)))
        else:
            self.weight = self.rng.standard_normal((num_embeddings, embedding_dim), dtype=np.float32)

    else:
        self.weight = weight_external

    self.gradient_projection_mask = np.eye(num_embeddings, dtype=np.uint8)

    self.input = None
    self.grad_weight = None
```

```
def forward(self, input: ArrayLike) -> np.ndarray:
    self.input = np.asanyarray(input)
    return self.weight[self.input.astype(np.int32), :]
```

# GELU

```
class GELU():
    def __init__(self) -> None:
        self._sqrt_of_2_by_pi = np.sqrt(2/np.pi)
        self.input = None

    def forward(self, input: ArrayLike) -> np.ndarray:
        self.input = np.asanyarray(input)
        return (0.5*input*(1 + np.tanh(self._sqrt_of_2_by_pi*(input + 0.044715*np.power(input, 3)))))
```

# DROPOUT

```
class Dropout():
    def __init__(self, p: float=0.2):
        self.p = p
        self.scale = 1/(1 - p)
        self.rng = np.random.default_rng()

        self.mask = None
        self.input = None


    def forward(self, input: ArrayLike, train: bool = False) -> np.ndarray:
        input = np.asanyarray(input)
        self.input = input

        if train:
            self.mask = self.rng.binomial(1, 1 - self.p, size=input.shape).astype(input.dtype)*self.scale
        else:
            self.mask = 1

        return input*self.mask
```

# LAYER NORM

```
class LayerNorm():
    def __init__(self,
                    normalized_shape: Union[int, tuple[int]],
                    eps: float=1e-05,
                    lr: float=1e-3,
                    weight_init_func: Union[Callable, None] = None,
                    bias_init_func: Union[Callable, None] = None) -> None:

       self.eps = eps
       …
       if self.weight_init_func:
            self.weight = np.asanyarray(self.weight_init_func((normalized_shape)))
        else:
            self.weight = np.ones((normalized_shape), dtype=np.float32)
        if self.bias_init_func:
            self.bias = np.asanyarray(self.bias_init_func((normalized_shape)))
        else:
            self.bias = np.zeros((normalized_shape), dtype=np.float32)

        self.axis = None
        …

    def forward(self, input: ArrayLike) -> np.ndarray:
        self.input = input
        self.axis = tuple(range(-len(self.normalized_shape), 0))

        mean = np.mean(input, axis=self.axis, keepdims=True)
        var = np.var(input, axis=self.axis, keepdims=True)

        self.x_centered = input - mean
        self.stddev_inv = 1/np.sqrt(var + self.eps)

        output = self.x_centered*self.stddev_inv
        return self.weight*output + self.bias
```

# PROJECT WORK

# CHANGE IN SCHEDULE

Recent learnings

- Transformer implementation more complex, than initially expected
- Very easy to make mistakes

Thus: Transformer implementation is now part of the projects

- Will allow for more time for the implementation
- Reduce overall time pressure and stress

**Adjusted schedule**

- 1 Month: Transformer implementation
- Each Wednesday: Regular check-in with every group
- 1 Month: Individual advanced Projects

# FORMALITIES

Total “Arbeitsaufwand” for the practical: 180h

- We expect, that about 60h have been covered by now
  - About 10h lectures and 50h exercises
- 120h remaining for the project work

Advanced, individual group Projects

- Proposed by the groups, but we’ll give some examples to use
- General plan to be discussed with us before starting
- **Project proposals to be submitted with nanoGPT code**

# PROJECT PROPOSALS

General content of a project proposal

- Title & Abstract
- Estimated time plan

Projects should be built such that one can distinguish individual work

Some example ideas: Next Slide

# EXAMPLE 1: EXPLORING OPTIMIZERS

Currently using SGD, which is fine, but not optimal

SOTA optimizers should improve performance noticeably

- Examples: ADAM, RMSProp, AdaDelta

Potential extension to second order optimizers

- Usually very complex to implement
- In our case an almost logical extension

**input** : $\gamma$ (lr), $\beta_1, \beta_2$ (betas), $\theta_0$ (params), $f(\theta)$ (objective)
$\lambda$ (weight decay), *amsgrad*, *maximize*
**initialize** : $m_0 \leftarrow 0$ ( first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

**for** $t = 1$ **to** $\ldots$ **do**
  **if** *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  **else**
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  **if** $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2$
  $\widehat{m_t} \leftarrow m_t / \big(1-\beta_1^t\big)$
  $\widehat{v_t} \leftarrow v_t / \big(1-\beta_2^t\big)$
  **if** $amsgrad$
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}^{max}} + \epsilon\big)$
  **else**
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \big(\sqrt{\widehat{v_t}} + \epsilon\big)$

**return** $\theta_t$

ADAM algorithm

# EXAMPLE 2: TRANSFORMERS FOR MUSIC

Built using MIDI as input data

Prediction of next note in a song

See: MusicTransformer (Huang et al. 2018)

MIDI keyboard example

https://en.wikipedia.org/wiki/File:Bach-_Crab_Canon_from_the_Musical_Offering.webm

# EXAMPLE 2: TRANSFORMERS FOR MUSIC

Built using MIDI as input data

Prediction of next note in a song

See: MusicTransformer (Huang et al. 2018)

MIDI keyboard example

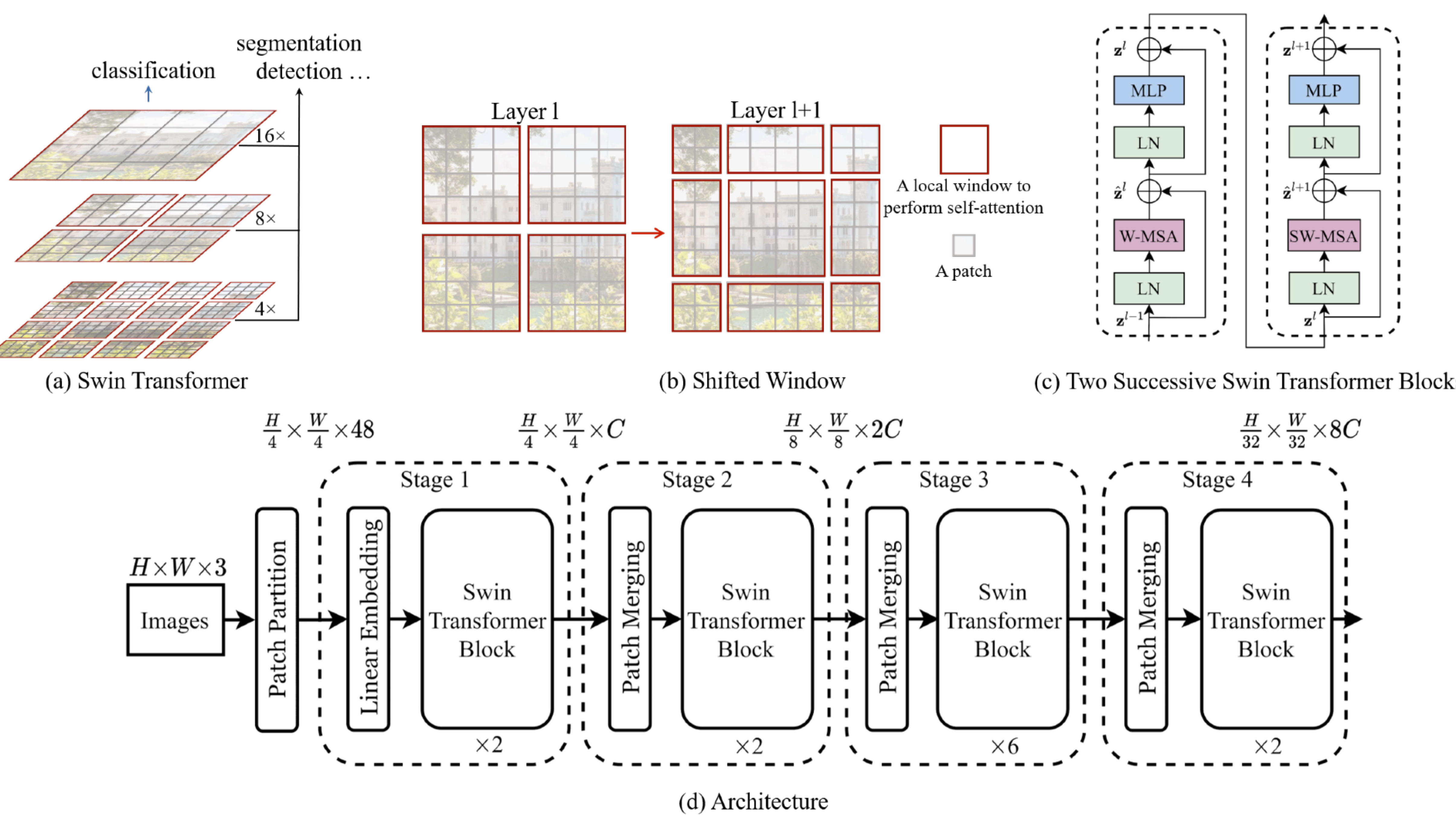
https://en.wikipedia.org/wiki/File:Bach-_Crab_Canon_from_the_Musical_Offering.webm

# EXAMPLE 3: TRANSFORMERS FOR VISION

Explore a transformer architecture for image classification taks

Example architectures

- Vision Transformer
- Swin Transformer



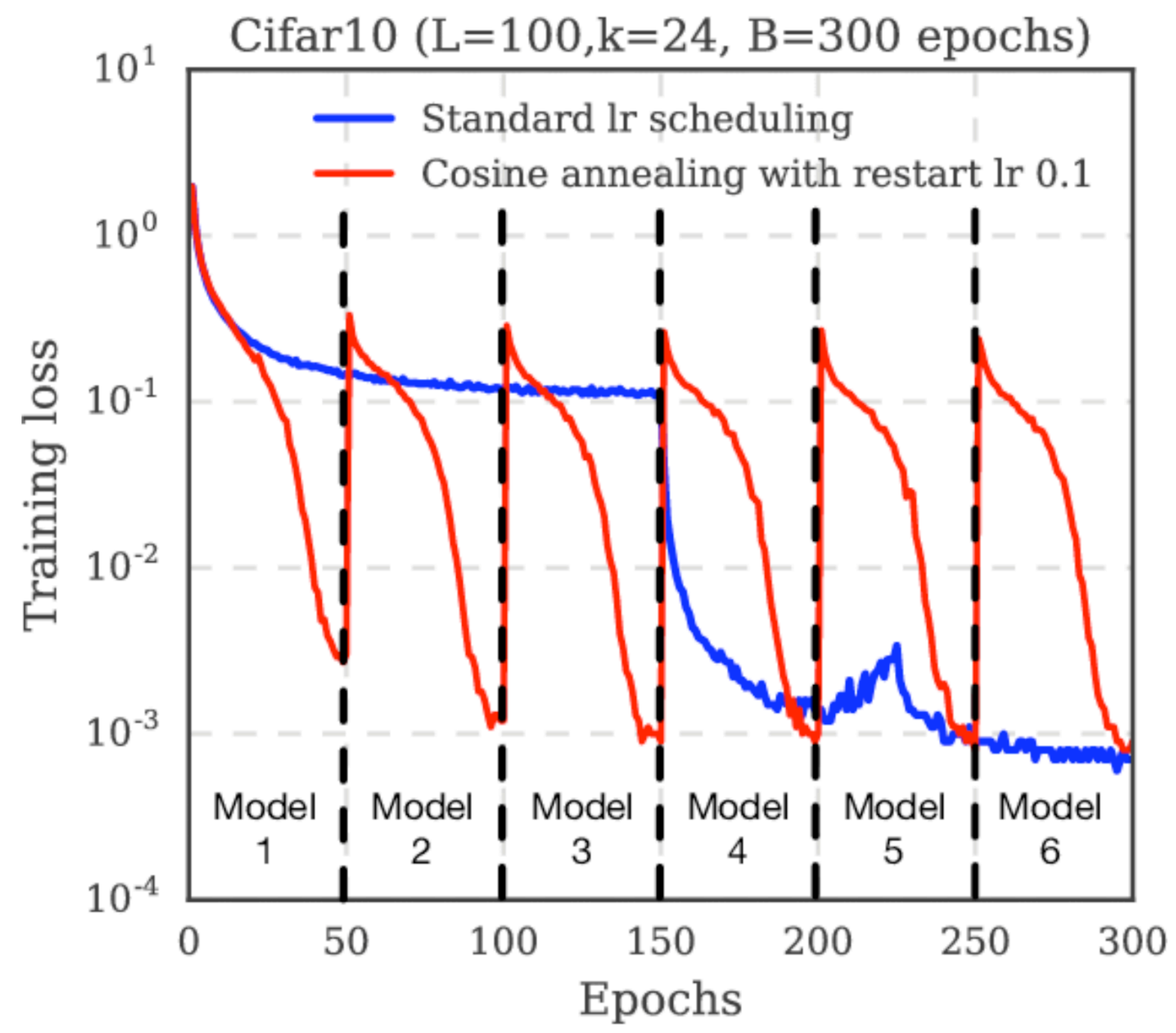

SWIN Transformer architecture

# EXAMPLE 4: TRAINING IMPROVEMENT

Improve Transformer training using tools external to the actual architecture

Example approaches:

- Learning rate schedulers
- Regularization methods
- Data augmentation




Cosine annealing LR scheduler with restarts

Loshchilov, et al; SGDR: Stochastic Gradient Descent with Warm Restarts (2017)

# EXAMPLE 5: OTHER TRANSFORMER ARCH

### Encoder-Only

*Auto-encoding* models:
Attention layers can access the whole sentence

**Example Tasks:**
Classification, Question Answering

**Example Models:**
BERT family

### Encoder-Decoder

*sequence-to-sequence* models:
Access patterns for encoder part as in encoder-only models, for decoder as in decoder-only models

**Example Tasks:**
Translation, Summarization

**Example Models:**
Original Transformer, BART, T5

### Decoder-Only

*Auto-regressive* models:
At each position, attention layers can only access elements positioned before it in the sequence

**Example Tasks:**
Text Generation

**Example Models:**
GPT family

*Shervin Minaee et al.: Large Language Models: A Survey, 2024.* https://arxiv.org/abs/2402.06196

# WRAPPING UP

# SUMMARY

Cross entropy loss for classification problems

- In general used as SOTA

# SUMMARY

Cross entropy loss for classification problems

- In general used as SOTA

Focus transformer in this practical nanoGPT

# SUMMARY

- Cross entropy loss for classification problems
  - In general used as SOTA
- Focus transformer in this practical nanoGPT
- Some general ideas for projects

# SUMMARY

- Cross entropy loss for classification problems
  - In general used as SOTA
- Focus transformer in this practical nanoGPT
- Some general ideas for projects

# ANY QUESTIONS?

# 5 MIN BREAK

Review Attention: Group 7 (Liam, Daniela, Marianna)
Review Vision-Transformer: Group 5 (Payam, et al.)
Review Tokenizer: Group 6 (Ole, et al.)

# THIS WEEKS EXERCISE

Review Attention: Group 7 (Liam, Daniela, Marianna)
Review Vision-Transformer: Group 5 (Payam, et al.)
Review Tokenizer: Group 6 (Ole, et al.)

# FROM THIS WEEK ON: PROJECT WORK

# PROJECT WORK: PART 1

Transformer from scratch

- Based on nanoGPT
- Backward path and update implementation

Project proposals

- General abstract of the project
- Estimated time plan

**Submission deadline:**

**Tuesday, December 17th 09:00 am**

https://csg.ziti.uni-heidelberg.de/teaching/ap_nn_from_scratch_materials/

# ANY QUESTIONS?