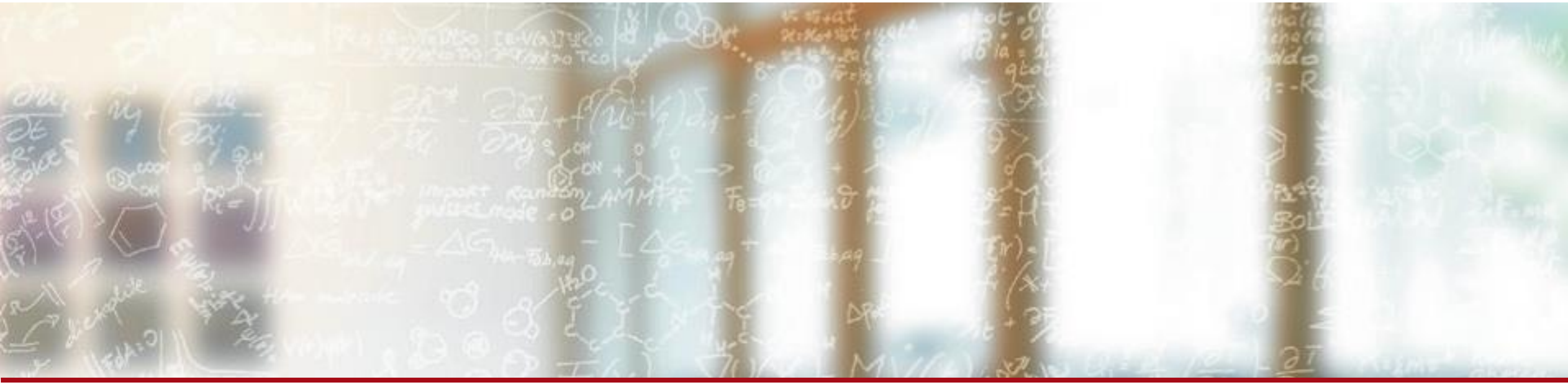


Multi-Node Distributed Training with tf.keras

CSCS TensorFlow 2020

Henrique Mendonça and Rafael Sarmiento, CSCS

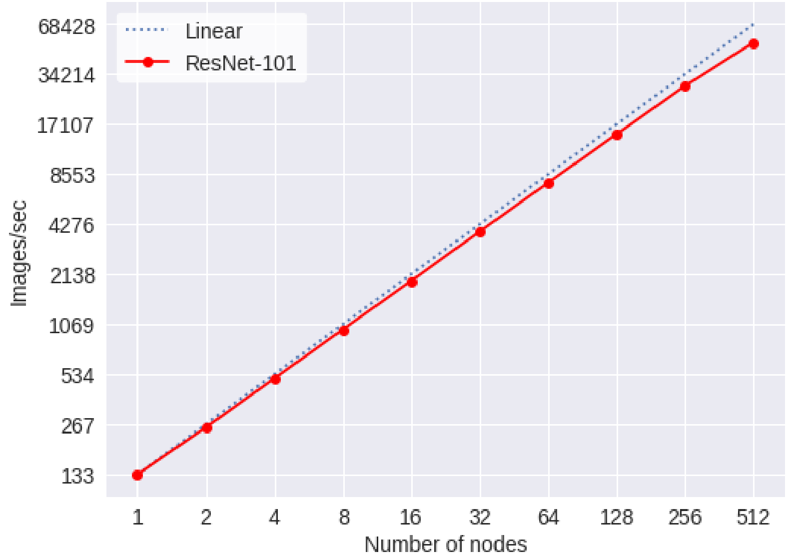
7-8 Sep 2020

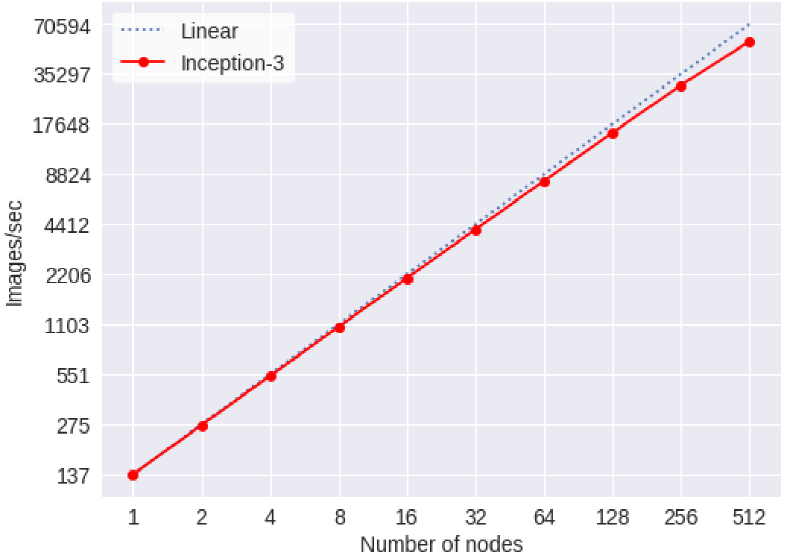
Motivation: Why distribute?

- Why should we use multiple GPU's to train a DL model?

Motivation: Why distribute?

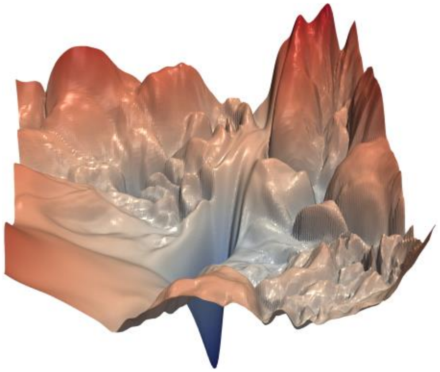
- Because it's faster
  - Allow quick experimentation of new ideas and models



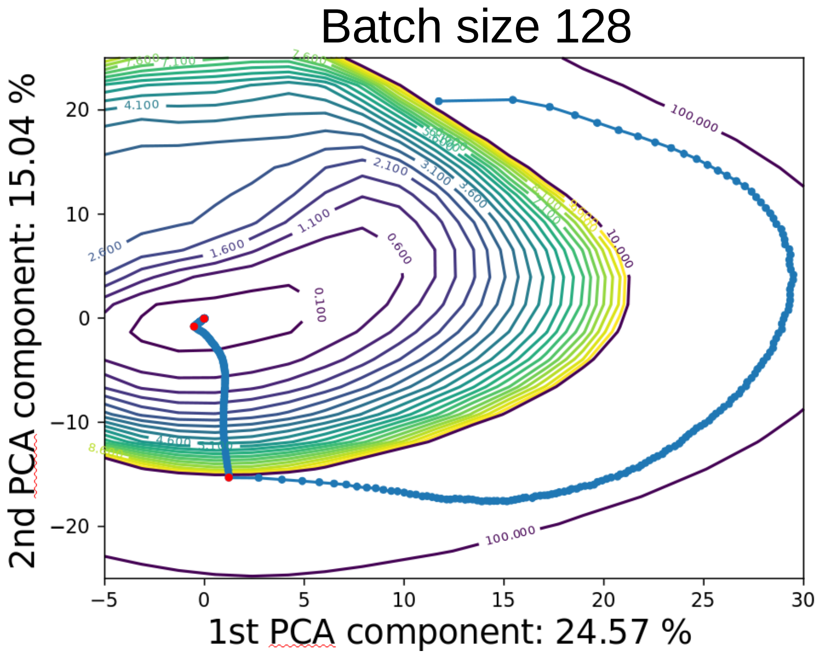


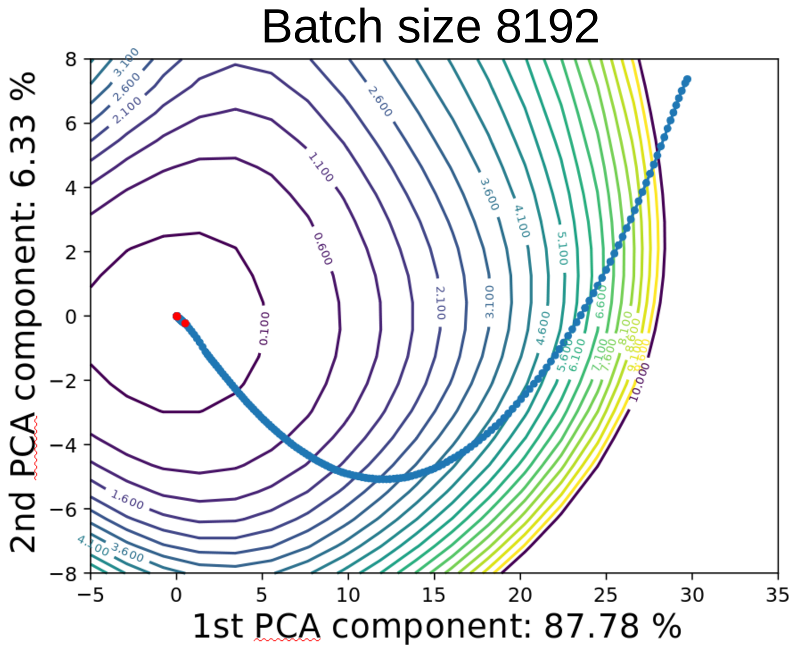
- Allows training models larger than memory
  - Out-of-Core learning: Not covered in this course
- May allow better accuracy, especially in larger models. Why?

Motivation: Sample noise



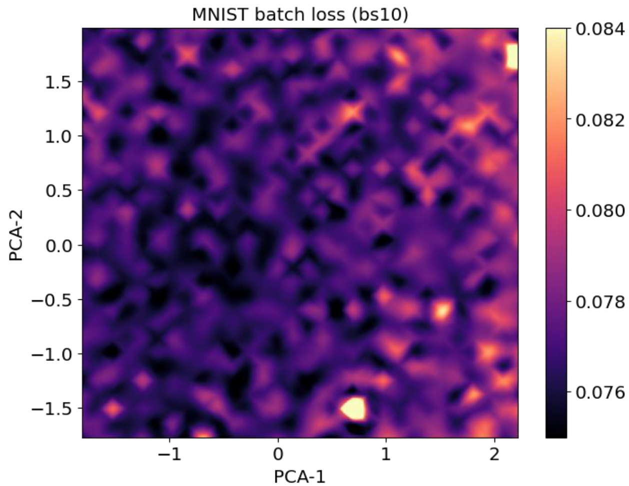
Visualizing the Loss Landscape of Neural Nets
https://arxiv.org/pdf/1712.09913.pdf

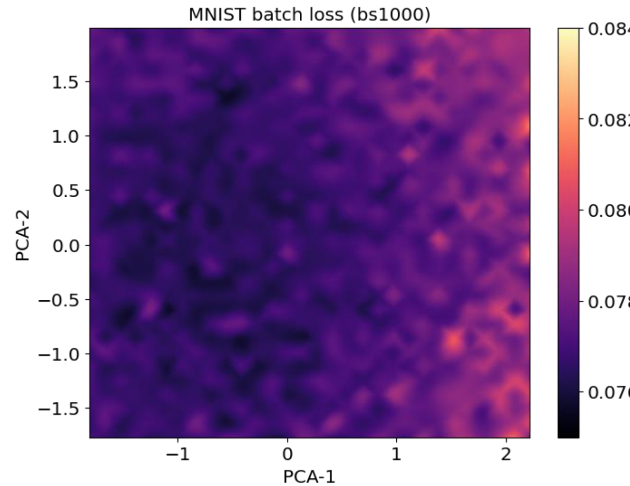




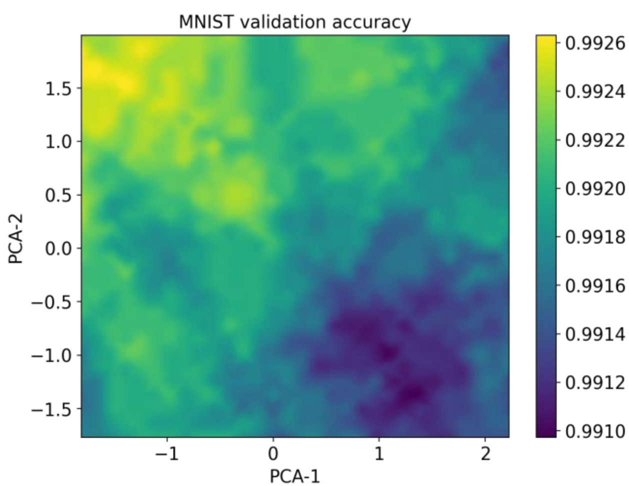
Motivation: Sample noise

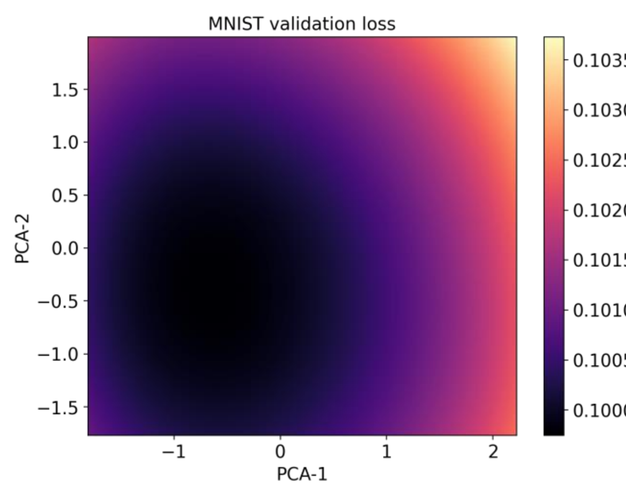
- Training:





- Validation:





Distributed training with TF Keras 2.3

- TensorFlow introduces [tf.distribute.Strategy]
  - Wraps model in a distributed scope

- Multi-GPU -> [tf.distribute.MirroredStrategy]
  - Single node/worker

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
   model = build_and_compile_model()
model.fit(dataset, epochs, steps_per_epoch)
```

  - [NCCL] AllReduce by default
  - Automatic data sharding across GPU's

Multi-worker Distribution

- Creates some additional complexity
  - external network communication
  - separated OS
  - separated processes
    - Facilitated by ipyparallel magic on JupyterLab

Multi-worker Distribution

- tf.distribute.experimental.MultiWorkerMirroredStrategy
- Communication:
  - NCCL AllReduce for all-reduce (if available)
  - Ring algorithm for all-gather
  - Includes fault tolerance when using BackupAndRestore
  - Datasets are automatically sharded across nodes
- Cluster Resolver:
  - defaults to TFConfig

```
os.environ['TF_CONFIG'] = '{
    "cluster": {"worker": ["nid01111:8888", "nid02222:8888"]},
    "task": {"type": "worker", "index": "0"}
}'
```

Multi-worker distribution with SLURM

- TensorFlow 2.2+

```
tf.distribute.cluster_resolver.SlurmClusterResolver(
    port_base=8888, auto_set_gpu=True, rpc_layer='grpc',
    jobs=None, gpus_per_node=None, gpus_per_task=None,
    tasks_per_node=None
)
```

- All parameters are automatically queried from SLURM

Multi-worker distribution with SLURM

```
%%px
strategy = tfd.experimental.MultiWorkerMirroredStrategy(
    cluster_resolver=tfd.cluster_resolver.SlurmClusterResolver(),
    communication=tfd.experimental.CollectiveCommunication.NCCL,
)
with strategy.scope():
    model = build_and_compile_model()
model.fit(dataset, epochs=N, steps_per_epoch=M)


# Done!
```

Multi-worker distribution with SLURM

- Sharding
  - Both Training and Validating datasets are automatically distributed across nodes
    - File sharding is used when possible (TFrecords)

- Fault Tolerance
  - The BackupAndRestore callback saves a checkpoint every epoch and restore when needed

- model.predict() is not supported
  - Use a single node instead

Multi-worker distribution with SLURM

- Practise
  - Run MNIST training and inference on your GPU's
    - 03_mnist_tf.distribute.ipynb

Batch Norm Synchronisation

- BN: Exponential Moving Average per channel

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}}$$

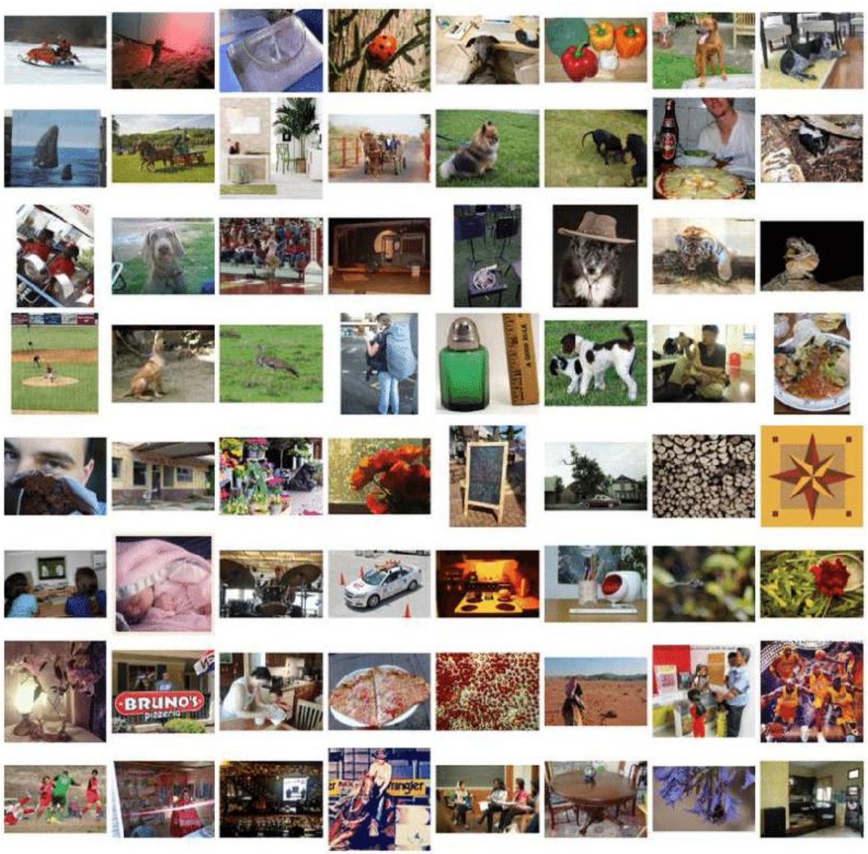
$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)}$$

- tf.keras.layers.experimental.SyncBatchNormalization
  - AllReduce across BN layers during forward pass
  - Synchronizes all statistics before autodiff

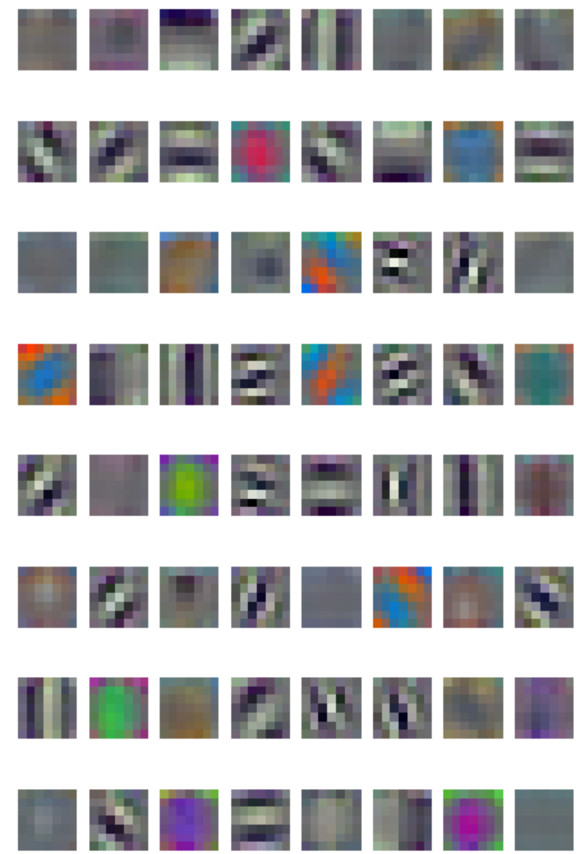
ImageNet

- Transfer Learning made Deep Learning accessible

ImageNet-2012: 1,281,167 samples



1st convolutional layer of
SeResNeXT101.32x4d (ImageNet)

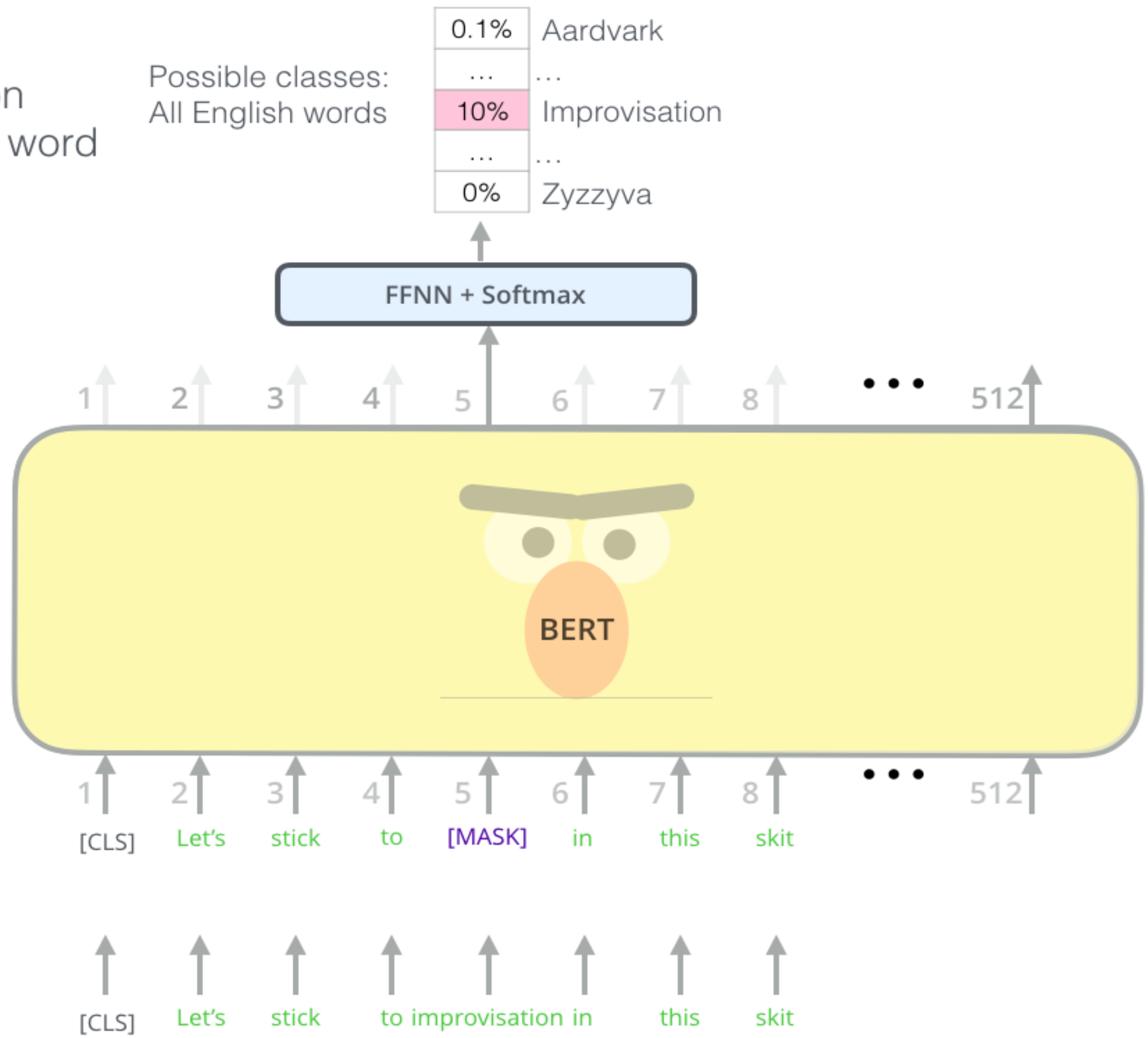


BERT: LM Encoder-Decoder

Use the output of the masked word's position to predict the masked word

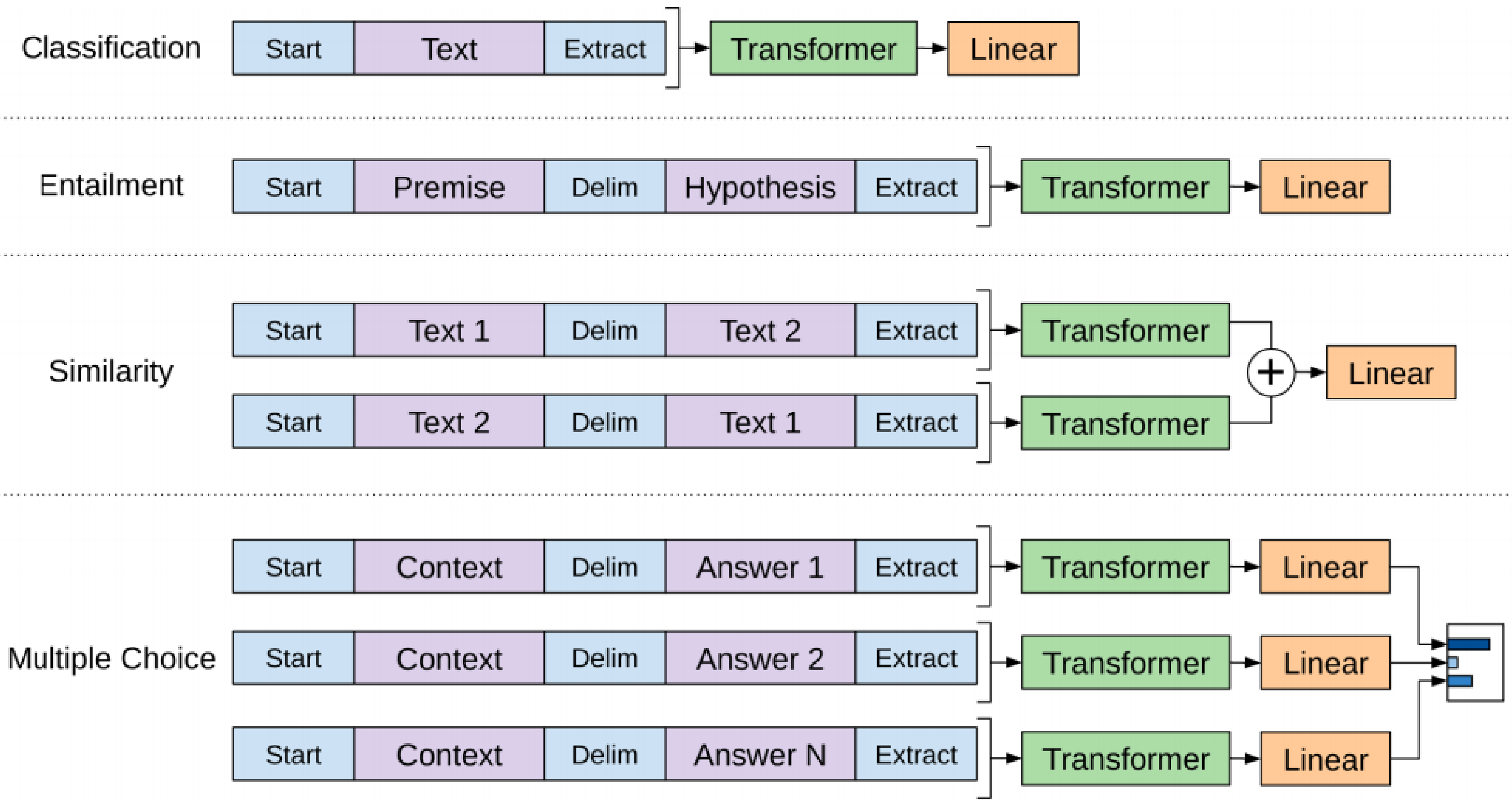
Randomly mask 15% of tokens

Input



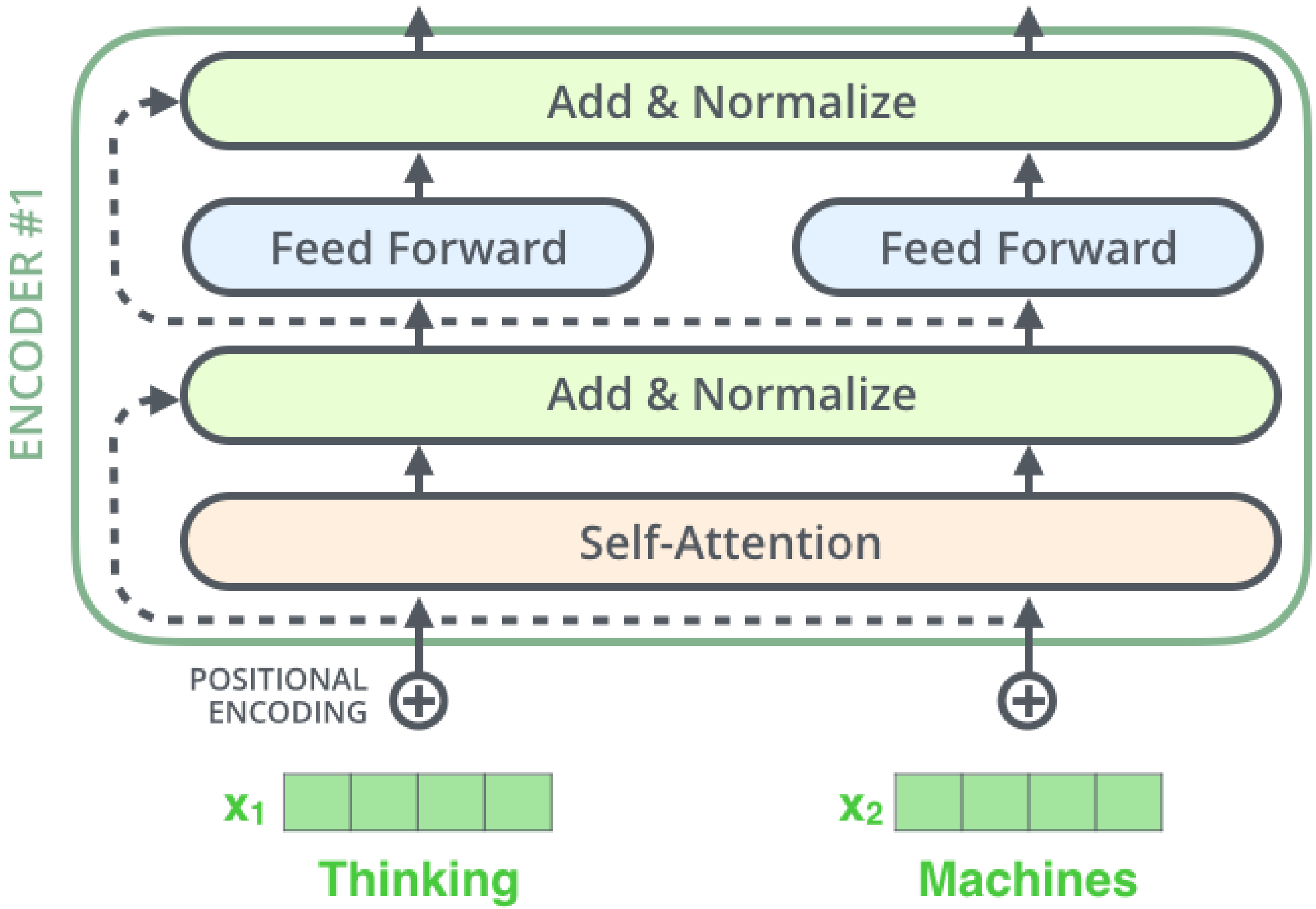
Source http://jalammar.github.io/illustrated-bert/

BERT: Encoder Fine-Tuning



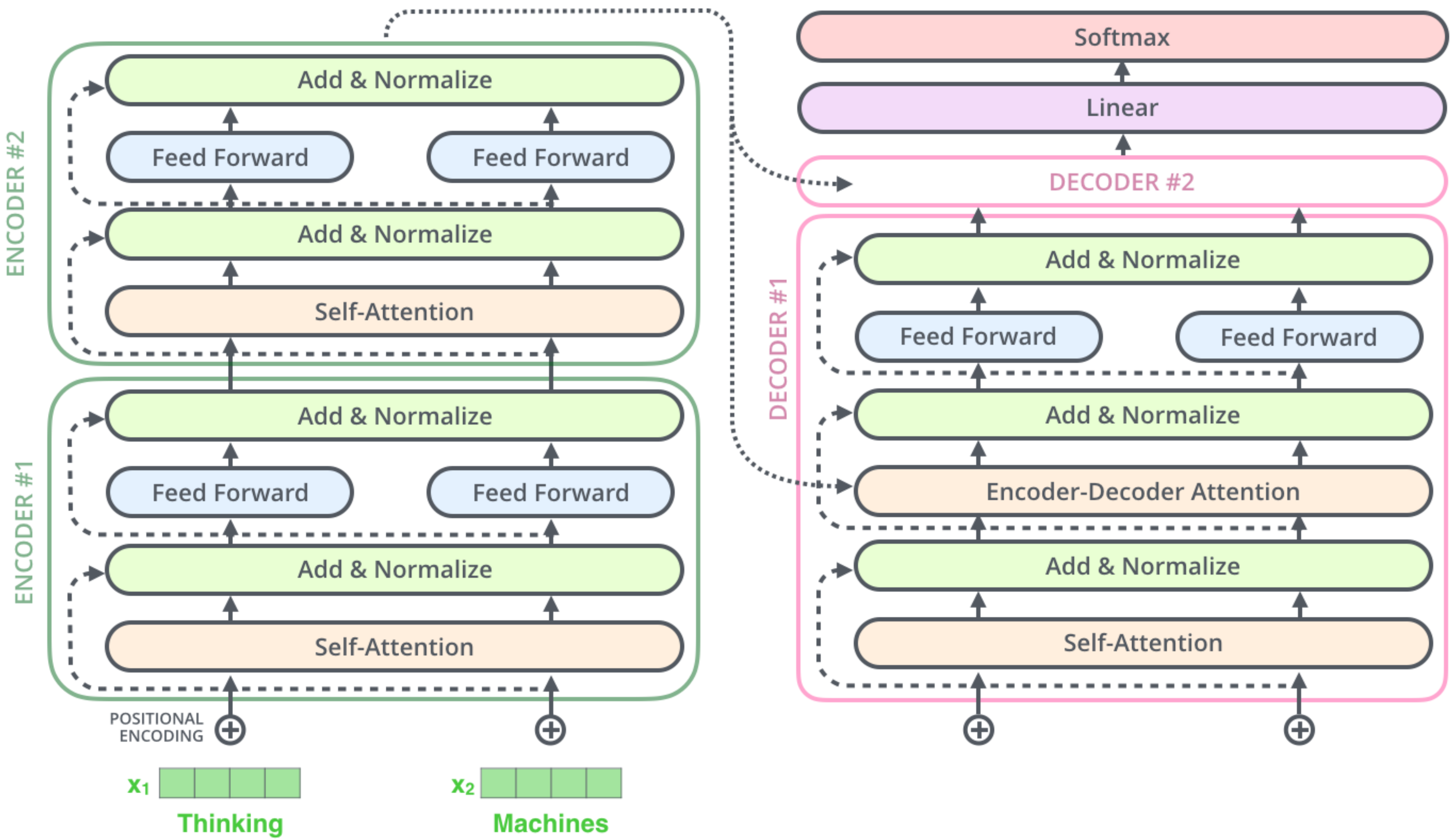
Source http://jalammar.github.io/illustrated-bert/
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding https://arxiv.org/abs/1810.04805

Transformer: Encoder Blocks



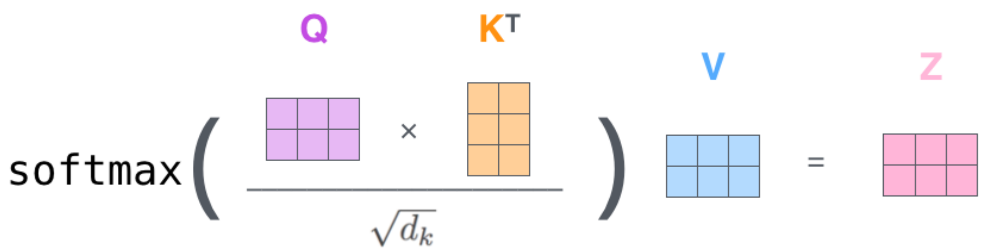
Source https://jalammar.github.io/illustrated-transformer
Attention Is All You Need https://arxiv.org/pdf/1706.03762.pdf

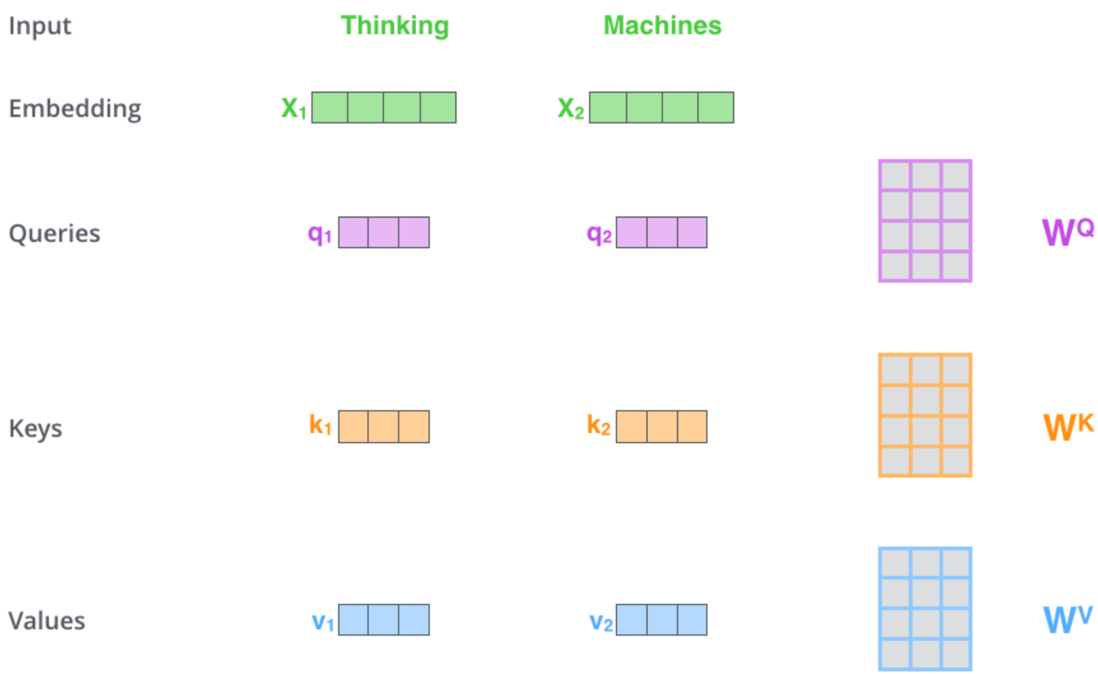
Transformer: LM Encoder-Decoder

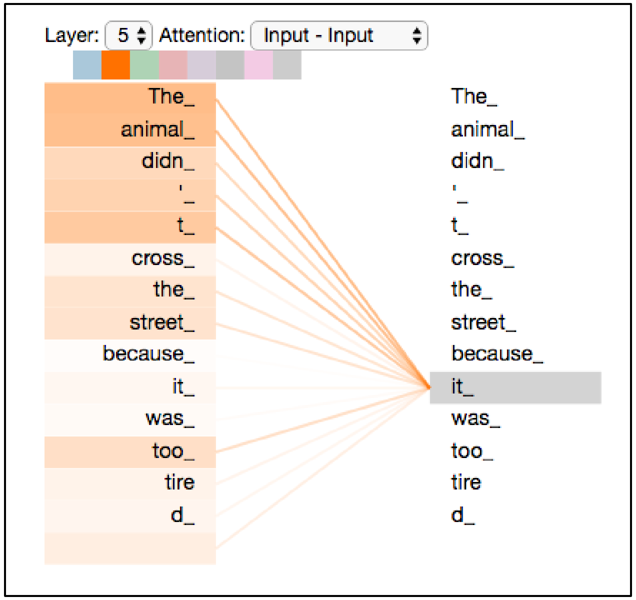


Source https://jalammar.github.io/illustrated-transformer
Attention Is All You Need https://arxiv.org/pdf/1706.03762.pdf

Transformer: Self-Attention at a High Level

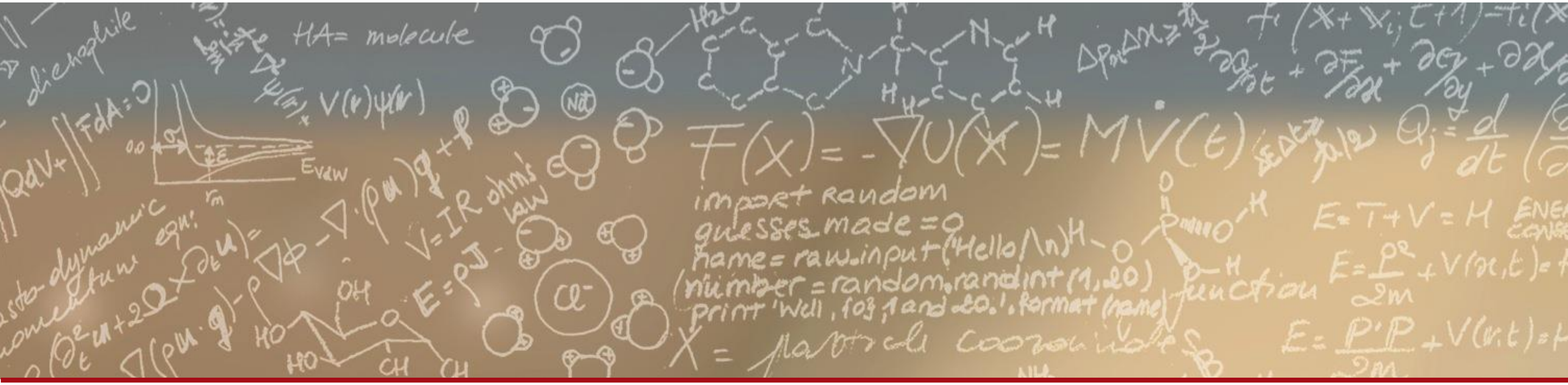






Source https://jalammar.github.io/illustrated-transformer

Attention Is All You Need https://arxiv.org/pdf/1706.03762.pdf



Thank you for your attention.