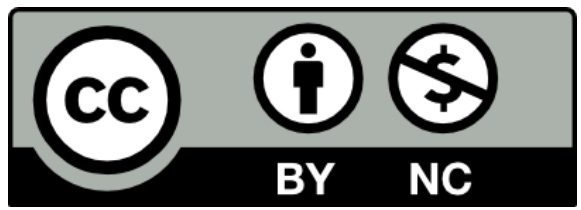
Business 4720 - Class 18

Advanced AI Techniques

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
jevermann@mun.ca



MEMORIAL UNIVERSITY

This Class

What You Will Learn:

- Data Augmentation
- Transfer Learning & Finetuning
- Attention Mechanisms
- Transformer Architecture
  - Multi-Headed Attention
  - Positional Encoding
  - BERT and GPT models
- Variational Autoencoders
- General Adversarial Networks

MEMORIAL UNIVERSITY

Based On

Kevin P. Murphy: *Probabilistic Machine Learning – An Introduction*. MIT Press 2022.

`https://probml.github.io/pml-book/book1.html`

Chapters 15, 19, 20

Foster, D: *Generative Deep Learning*, 2nd edition. O'Reilly Media 2022.

Chapters 3, 4, 9, 10

Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J.: *Dive into Deep Learning (D2L)*. Cambridge University Press. 2023.
`https://d2l.ai/`
`https://github.com/d2l-ai/d2l-en`

Data Augmentation

Purpose:

- Increase sample size
- Regularization
- Ensure realistic data
- Class balancing

Methods:

- Synthesize additional data
- Synthesize additional variations
- Introduce noise
- Synthetic oversampling

MEMORIAL UNIVERSITY

Data Augmentation [cont'd]

Source: Murphy Fig. 19.1

Examples in Vision

- Geometric transformations
  - Rotate, Flip, Crop, Translate
- Color space transformations
  - Brightness, contrast, saturation
- Noise injection
  - Gaussian noise, random pixels, black pixels
- Kernal filters
  - Blurring or smoothing

Data Augmentation [cont'd]

Examples in Text

- Character, word or sentence shuffling
- Word replacement with synonyms or similar words
- Random word insertion and deletion
- Noise insertion at embedding level
- Forward-backward translation

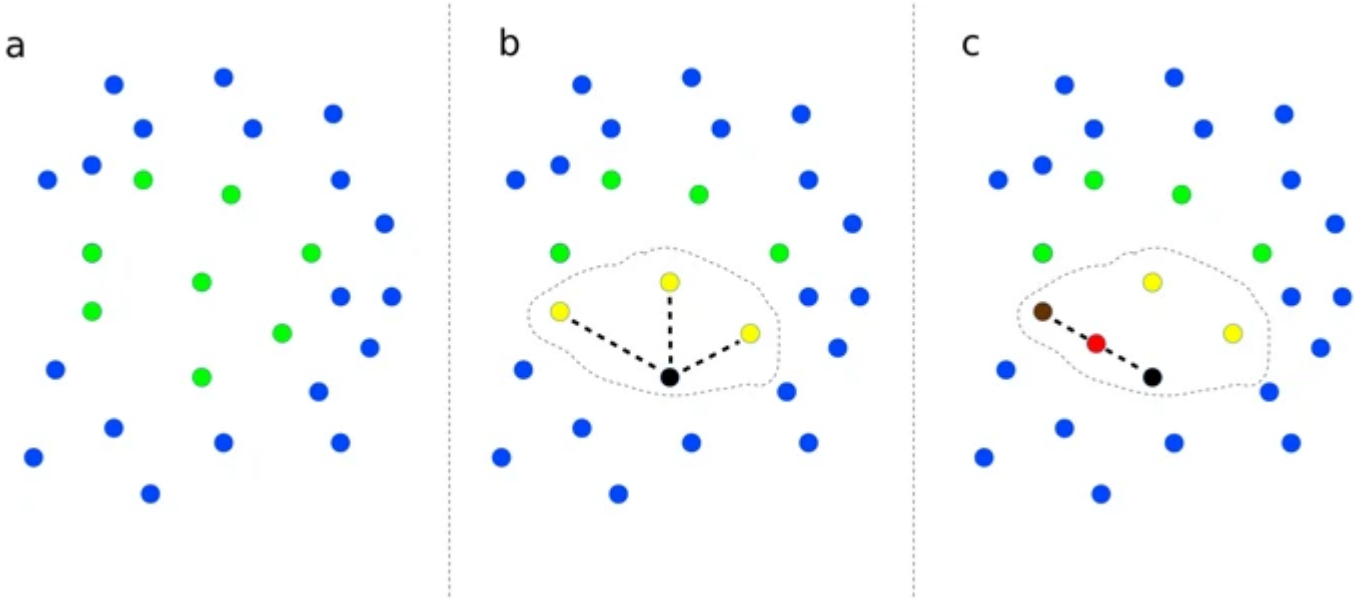
MEMORIAL UNIVERSITY

Data Augmentation [cont'd]

Examples in Tabular Data

- SMOTE, SMOTE-NC ("Synthetic Minority Oversampling Technique")
- AdaSyn ("Adaptive synthetic sampling")
- Variational autoencoders

SMOTE principles



Source: Schubach, M., Re, M., Robinson, P.N. et al. Imbalance-Aware Machine Learning for Predicting Rare and Common Disease-Associated Non-Coding Variants. Sci Rep 7, 2959 (2017). https://doi.org/10.1038/s41598-017-03011-5 (CC-BY-4.0)

MEMORIAL UNIVERSITY

Transfer Learning

Ideas

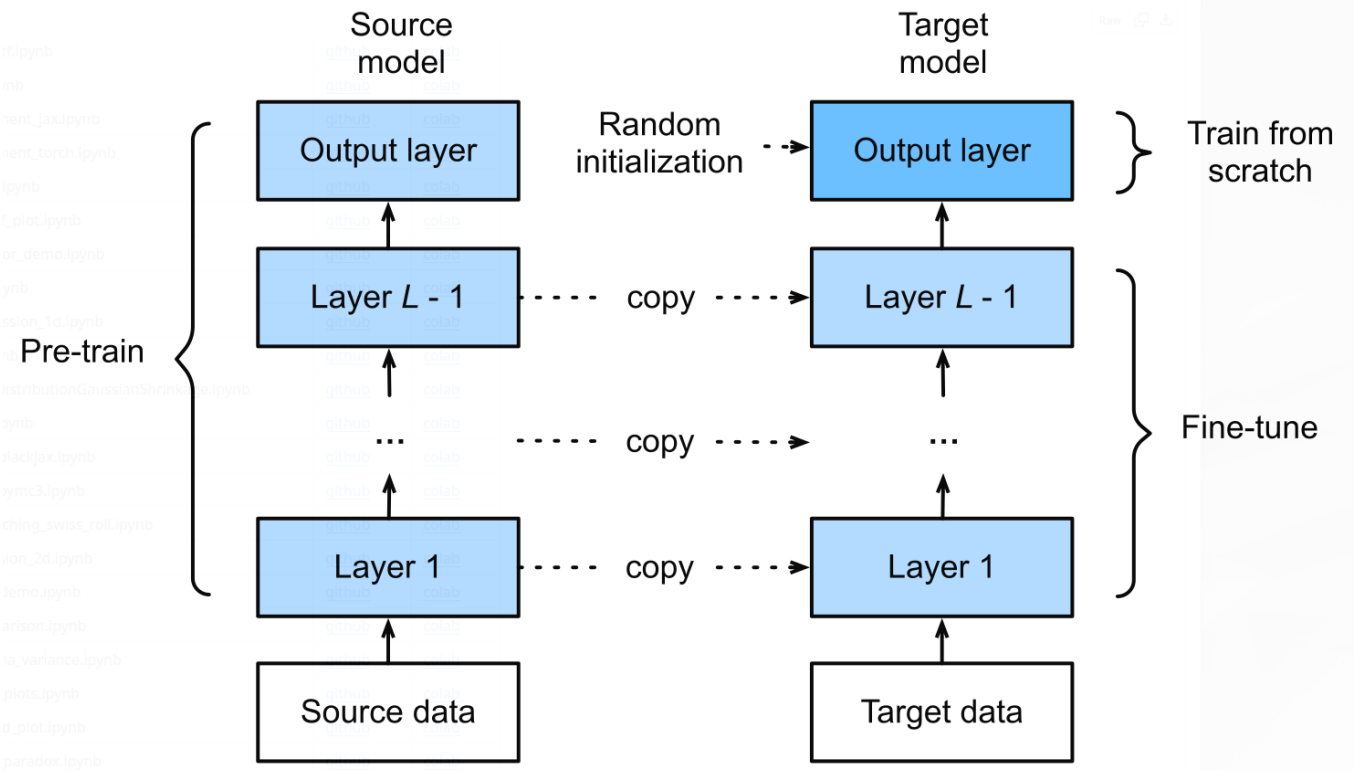
- Transfer learned models to related domains
  - Example: Use CNN for classifying cars to classify trucks
- For data-poor tasks similar to other tasks
  - Example: Classify endangered bird species with CNN trained on common birds
- To reduce training effort/cost
  - Example: Adapt pre-trained CNN to recognize organization-specific images
- Related to multi-objective learning/optimization

Phases

1. *Pretraining* on large *source data set*
2. *Fine-tuning* on smaller *target data set*

MEMORIAL UNIVERSITY

Transfer Learning [cont'd]



Source: Murphy Fig 19.2 (CC-BY-NC-ND) from

`https://github.com/d2l-ai/d2l-en` (CC-BY 4.0)

Options

- Do not fine-tune pre-trained layers (fix/freeze)
- Fine-tune pre-trained layers using small learning rate

Data Augmentation in Python

Load Tensorflow example data set:

```
import keras
from keras import layers
import tensorflow_datasets as tfds

# Load a Tensorflow example image data set
train_ds, test_ds = tfds.load(
    "cats_vs_dogs",
    split=["train[:75%]", "train[75%:100%]"],
    as_supervised=True)
```

Resize images to same size using Keras Resizing layer:

```
# Create Resizing layer
resize_fn = keras.layers.Resizing(150, 150)

# Apply resizing layer to data sets
train_ds = train_ds.map(lambda x, y: (resize_fn(x), y))
test_ds = test_ds.map(lambda x, y: (resize_fn(x), y))
```

MEMORIAL UNIVERSITY

Data Augmentation in Python [cont'd]

Data augmentation using Keras layers:

```
# Define a sequential model of random transformation layers
augmentation = keras.models.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomTranslation(.2, .2, fill_mode="reflect"),
    layers.RandomRotation(0.2, fill_mode="reflect"),
    layers.RandomZoom(.2, .2, fill_mode="reflect"),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2)])

# Apply augmentation model to training data
train_ds = train_ds.map(lambda x, y: (augmentation(x), y))
```

- This example applies all layers to all training images; consider applying single transformations to an image or random combinations of transformations
- This example replaces the data set with the transformed images; consider adding the transformed images to the data set

Load an image classification model and its pre-trained weights from Keras applications *without* the top classification layer:

```
base_model = keras.applications.Xception(
    weights="imagenet",
    input_shape=(150, 150, 3),
    include_top=False)
```

Freeze/fix the model's parameters, that is, make not trainable:

```
base_model.trainable = False
```

Transfer Learning in Python [cont'd]

Create a new layer with specific input image shape:

```
# Create new input model
inputs = keras.Input(shape=(150, 150, 3))
# Pre-trained Xception weights requires input scaling
# from [0, 255] to [-1., +1.]
scale_layer = keras.layers.Rescaling(scale=1/127.5, offset=-1)
x = scale_layer(inputs)
```

Add a new bottom layer for classification to the model. Make sure the base model is not trained:

```
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)
outputs = keras.layers.Dense(1)(x)

# Complete model has inputs and outputs
model = keras.Model(inputs, outputs)
# Not all parameters are trainable
model.summary(show_trainable=True)
```

Transfer Learning in Python [cont'd]

Compile and fit the new model (training only the classification layers) with default learning rate for optimizer:

```
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)
model.fit(train_ds.batch(64), epochs=2,
          validation_data=test_ds.batch(64))
```

Only a few epochs are needed

MEMORIAL UNIVERSITY

Transfer Learning in Python [cont'd]

Fine-tune the base model with a very small learning rate:

```
# Make all parameters trainable
base_model.trainable = True
# Now all parameters are trainable
model.summary(show_trainable=True)

# Very small learning rate
model.compile(
    optimizer=keras.optimizers.Adam(1e-5),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)
model.fit(train_ds.batch(64), epochs=2,
          validation_data=test_ds.batch(64))
```

Only a few epochs are needed

MEMORIAL UNIVERSITY

Data Augmentation and Transfer Learning in Python

Adapted from:

`https://keras.io/guides/transfer_learning/`

Implementation available on the following GitHub repo:

`https://github.com/jevermann/busi4720-ai`

The project can be cloned from this URL:

`https://github.com/jevermann/busi4720-ai.git`

MEMORIAL UNIVERSITY

Adapters

Fine-tuning problems:

- Fine-tuning may be slow for large model sizes
- Parameters may diverge too far from prior values
- New tasks require new fine-tuning of all parameters

Adapters:

- Fix/freeze pre-trained parameters
- Add additional model layers with their own parameters (”adapters”)

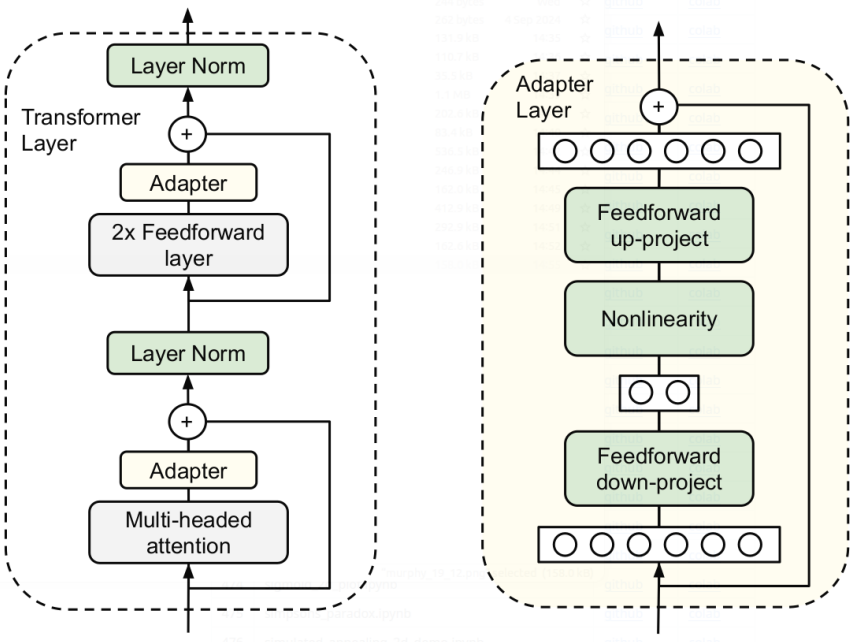
MEMORIAL UNIVERSITY

Adapters [cont'd]

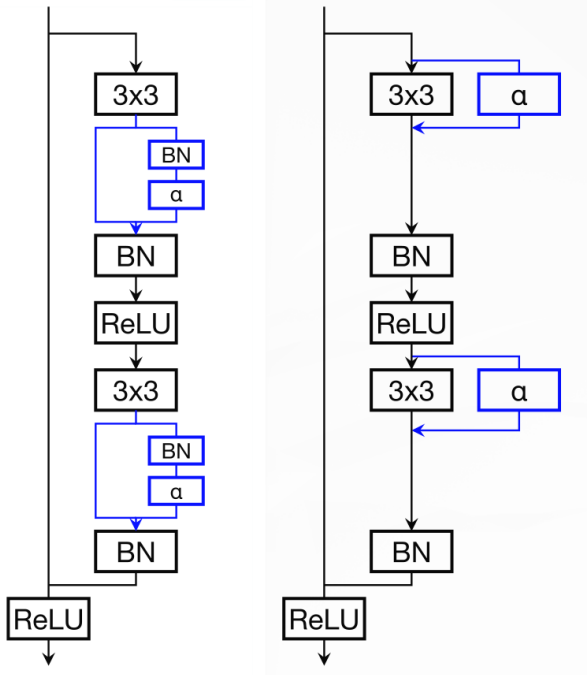
Examples

(a) Adapter in transformer layer

(b) Adapter in ResNet



(a)

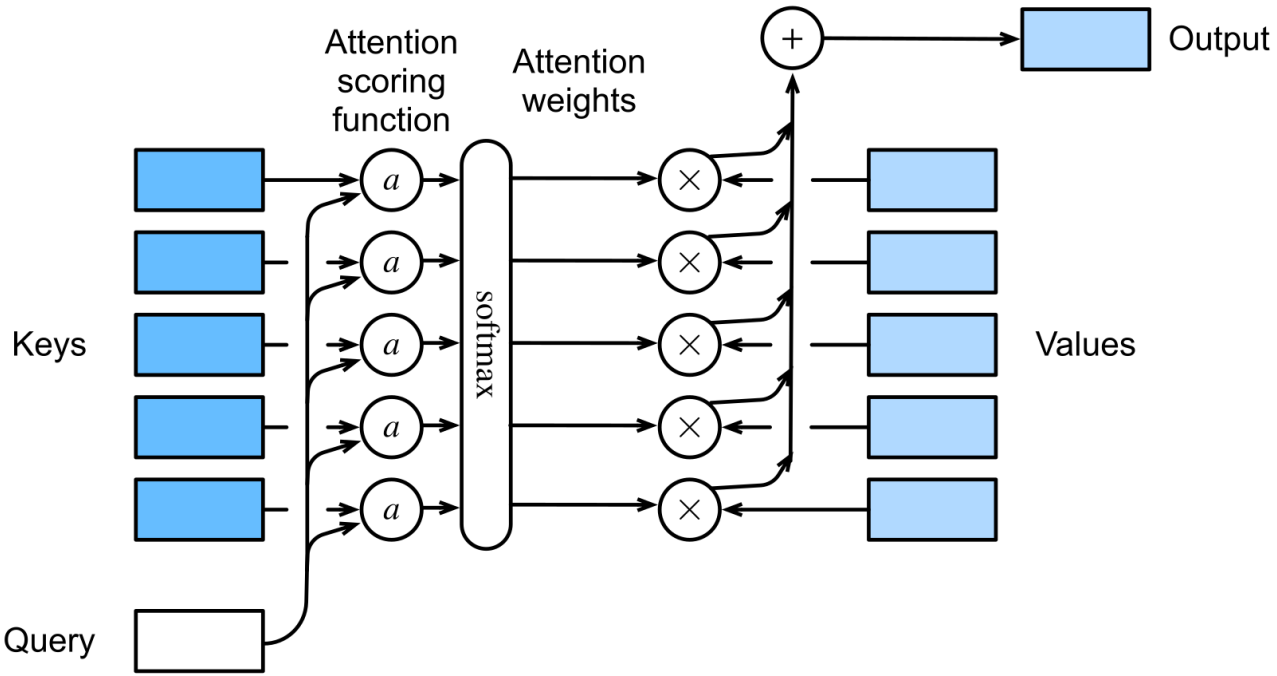


(b)

Source: Murphy Fig. 19.3

Attention

- A set of m key vectors K and value vectors V
- A query vector Q
- Attention score function a determines similarity of keys to query
- Determine weights for values based on attention scores



Source: Murphy, Fig. 15.16

Attention [cont'd]

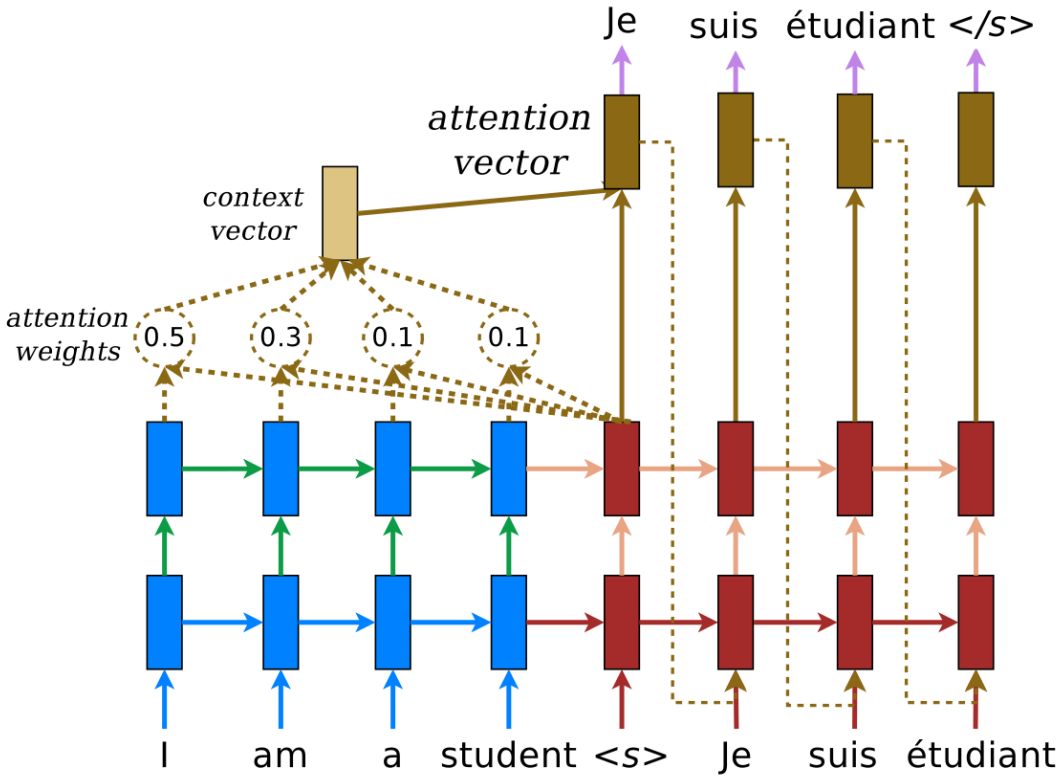
Scaled Dot-Product Attention

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

MEMORIAL UNIVERSITY

Attention in Seq2Seq Models

- Encoder – Decoder network
- Allow decoder access to input sequence, but word (token) order need not be preserved, must infer *alignment*



Source: Murphy, Fig. 15.18

Attention in Seq2Seq Models

Instead of fixed context vector for decoder:

$$c = h_T^e$$

now:

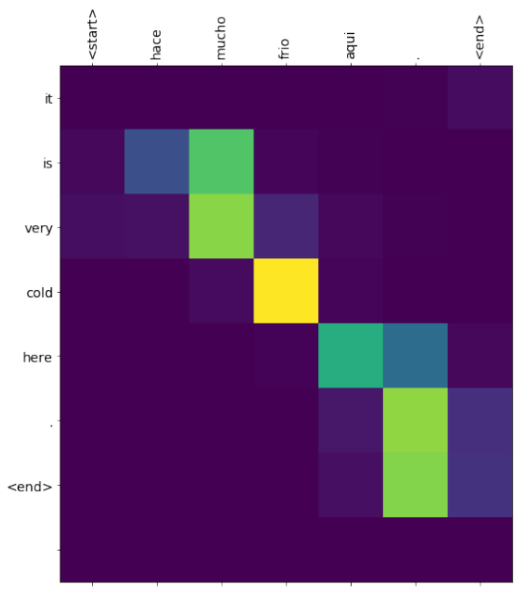
$$c_t = \text{Attn}(Q = h_{t-1}^d, K = h^e, V = h^e)$$

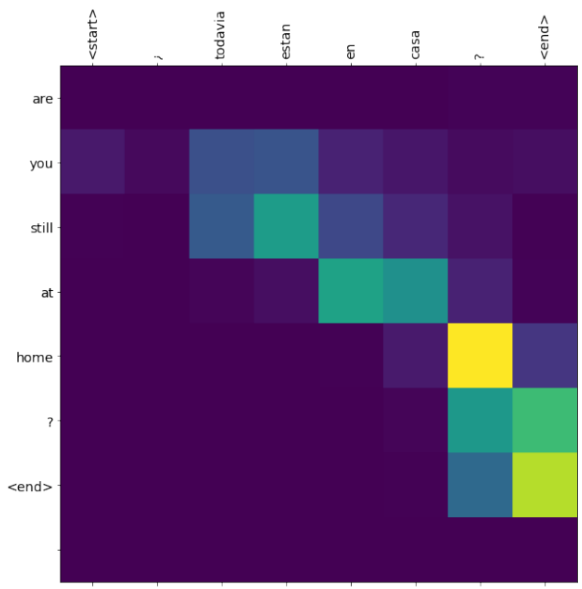
Query Q is the hidden state of decoder, keys K and values V are hidden states from encoder

MEMORIAL UNIVERSITY

Attention in Seq2Seq Models

Example: Attention in English – Spanish translation





Source: Murphy, Fig. 15.19

MEMORIAL UNIVERSITY

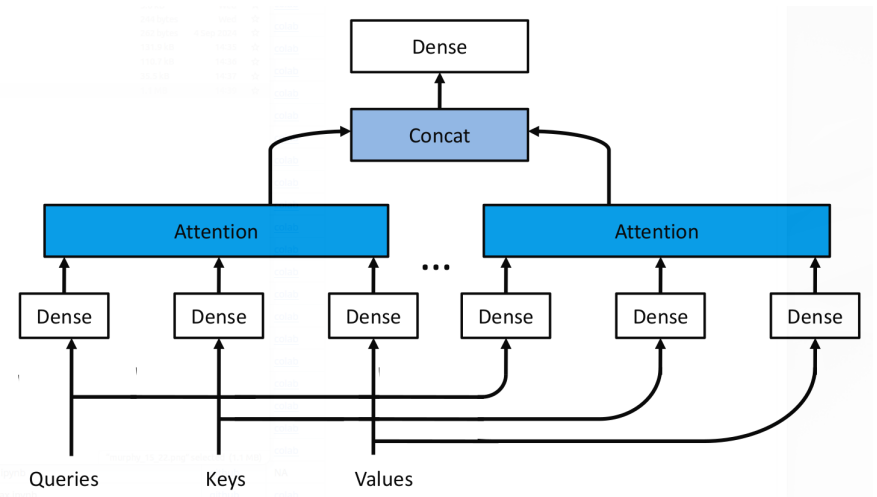
Multi-Headed Attention

$$\text{MultiHead}(Q, K, V) = W_o \,\text{Concat}(h_1, \ldots h_h) = W_o \begin{pmatrix} h_1 \\ \vdots \\ h_n \end{pmatrix}$$

where each head h_i is defined as:

$$h_i = \text{Attn}(QW_i^{(Q)}, KW_i^{(K)}, VW_i^{(V)})$$

and $W_i^{(Q)}$, $W_i^{(K)}$, $W_i^{(V)}$, and $W^{(O)}$ are trainable weight matrices.



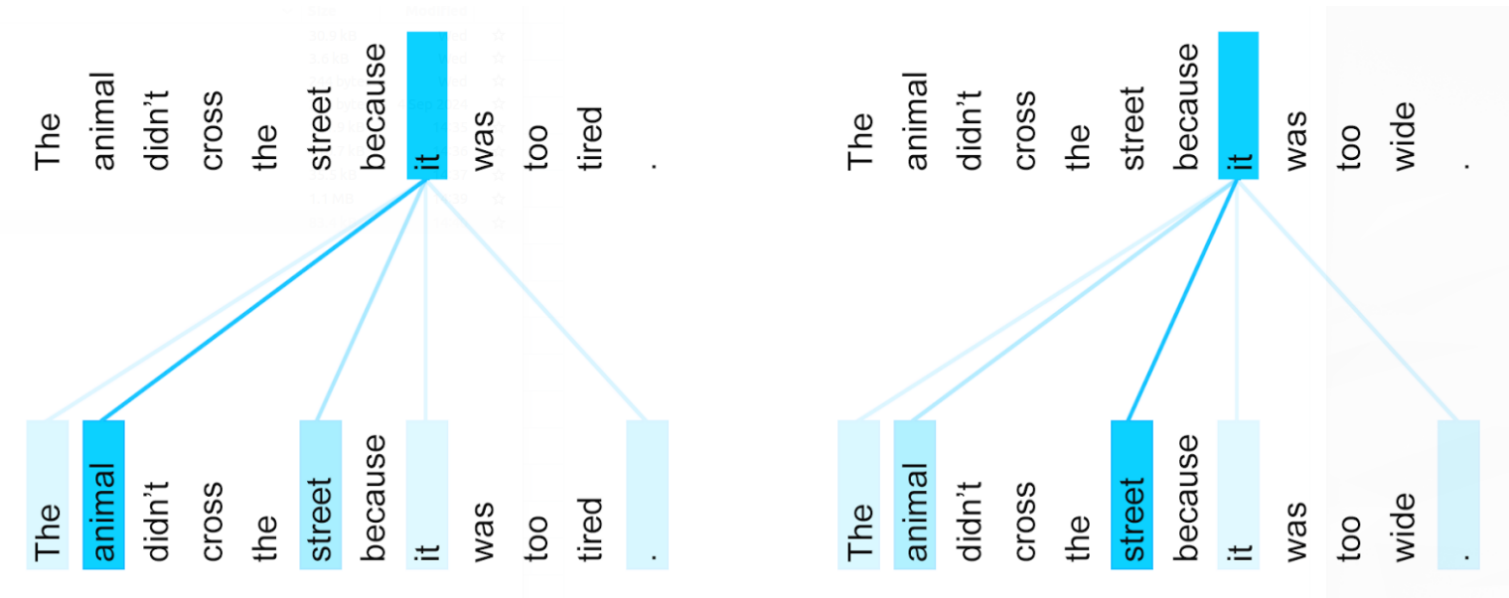
Source: Murphy, Fig. 15.24

Self-Attention

Query, keys, and values come from the same model, i.e. from the encoder or decoder.

$$y_i = \text{Attn}(x_i, X, X)$$

Where y_1 is output i, x_i is input i, and the matrix X is the matrix of all n inputs in the sequence.



Source: Murphy, Fig. 15.23

Positional Encoding

Model must know the order words (tokens) occur in. Positional encodings provide this.

$$p_{i,2j} = \sin\left(\frac{i}{C^{2j/d}}\right), \; p_{i,2j+1} = \cos\left(\frac{i}{C^{2j/d}}\right) \; \forall j \in [1 \ldots d/2]$$

- d is the chosen dimensionality of the positional encoding.
- C is the maximum vocabulary size.

For example, if $d = 4$ the i-th row is:

$$p_i = [\sin\left(\frac{i}{C^{0/4}}\right), \cos\left(\frac{i}{C^{0/4}}\right), \sin\left(\frac{i}{C^{2/4}}\right), \cos\left(\frac{i}{C^{2/4}}\right)]$$

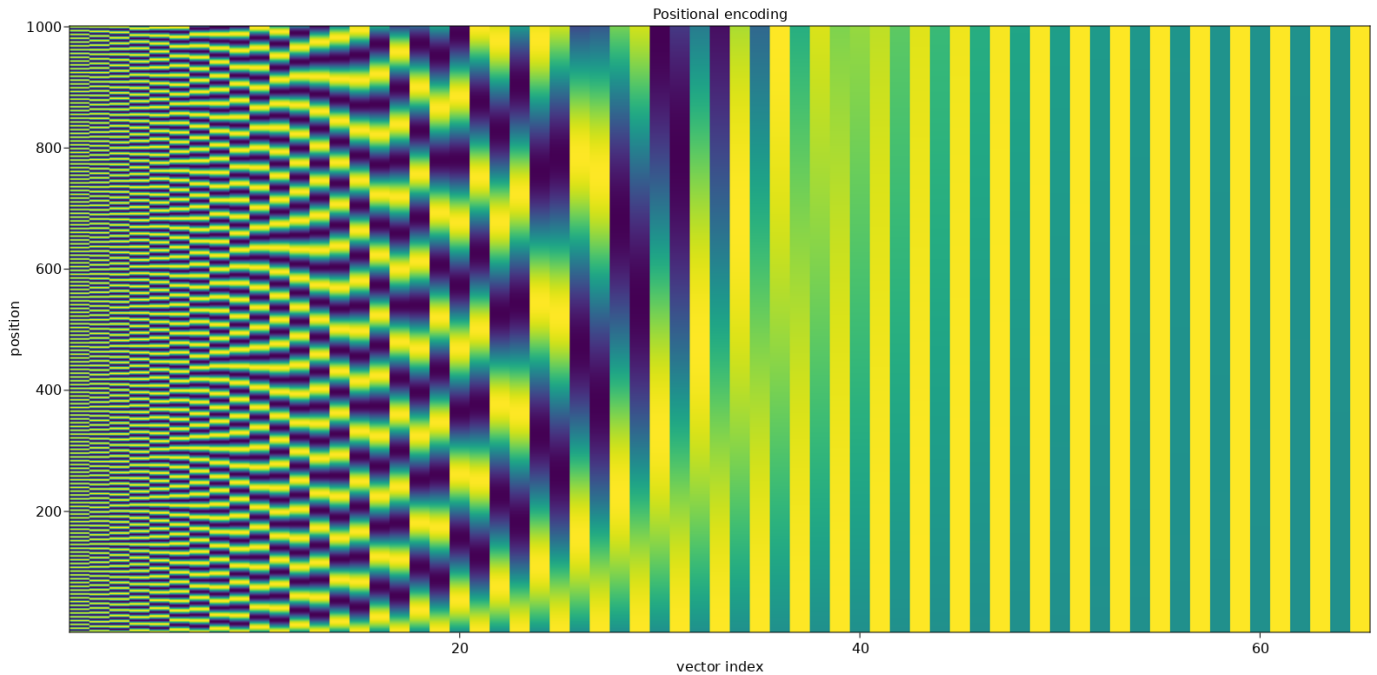
Combine with word embeddings X:

$$\text{POS}(\text{Embed}(X)) = X + P$$

MEMORIAL UNIVERSITY

Positional Encoding [cont'd]

- Example for 1000 positions (rows) and embedding dimension of 64.
- Each row is a real-valued vector representing its location in the sequence:



Source: `https://commons.wikimedia.org/wiki/File:Positional_encoding.png`

MEMORIAL UNIVERSITY

Positional Encoding in Python

Define weights:

```
import numpy as np

def position_encoding_matrix(seq_len, d, C=10000):
    P = np.zeros((seq_len, d))
    for i in range(seq_len):
        for j in np.arange(int(d / 2)):
            denominator = np.power(C, 2 * i / d)
            P[i, 2 * j] = np.sin(i / denominator)
            P[i, 2 * j + 1] = np.cos(i / denominator)
    return P
```

MEMORIAL UNIVERSITY

Positional Encoding in Python [cont'd]

Define Keras layer:

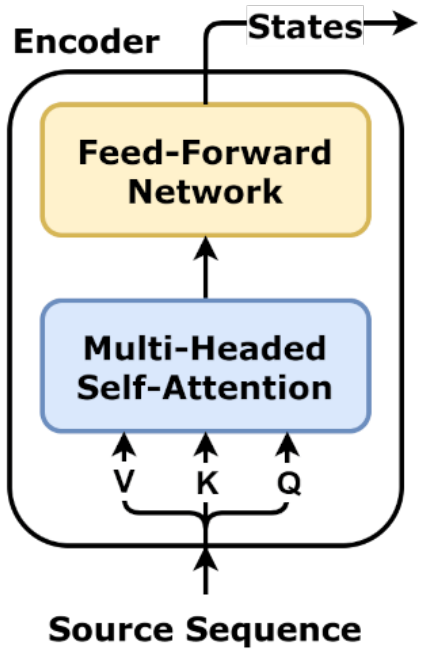
```
class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super().__init__()
        # Generate the positional encoding weights
        position_embedding_matrix = \
            position_encoding_matrix(maxlen, embed_dim)
        # Trainable token embedding layer
        self.token_emb = layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim)
        # Positional embedding with positional
        # encoding weights is not trainable
        self.pos_emb = layers.Embedding(
            input_dim=maxlen, output_dim=embed_dim,
            weights=[position_embedding_matrix],
            trainable=False
        )

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(0, maxlen, 1)
        # Return token plus position embedding
        return self.token_emb(x) + self.pos_emb(positions)
```

Transformer

Encoder uses multi-headed self-attention:

```
def EncoderBlock(X):
  Z = LayerNorm(MultiHeadAttn(Q=X,K=X,V=X)+X)
  E = LayerNorm(FeedForward(Z) + Z)
  return E
```

```
def Encoder(X, N):
  E = POS(Embed(X))
  for n in range(N):
    E = EncoderBlock(E)
  return E
```



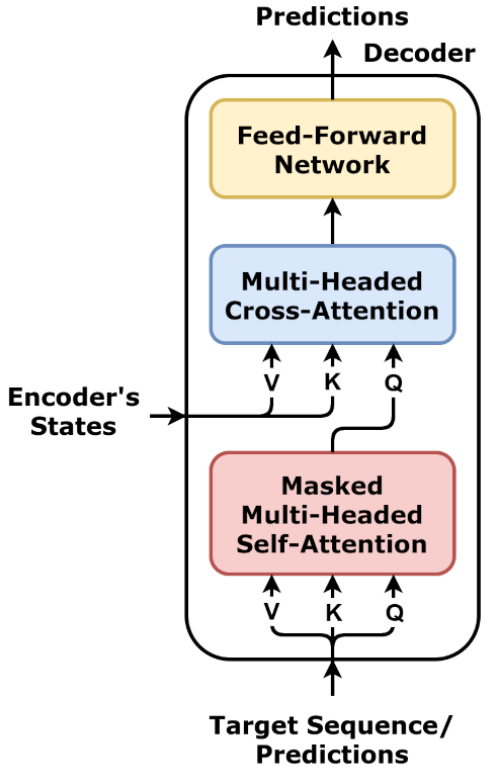
https://commons.wikimedia.org/wiki/File:Transformer,_one_encoder_block.png

Transformer

Decoder uses multi-headed self-attention:

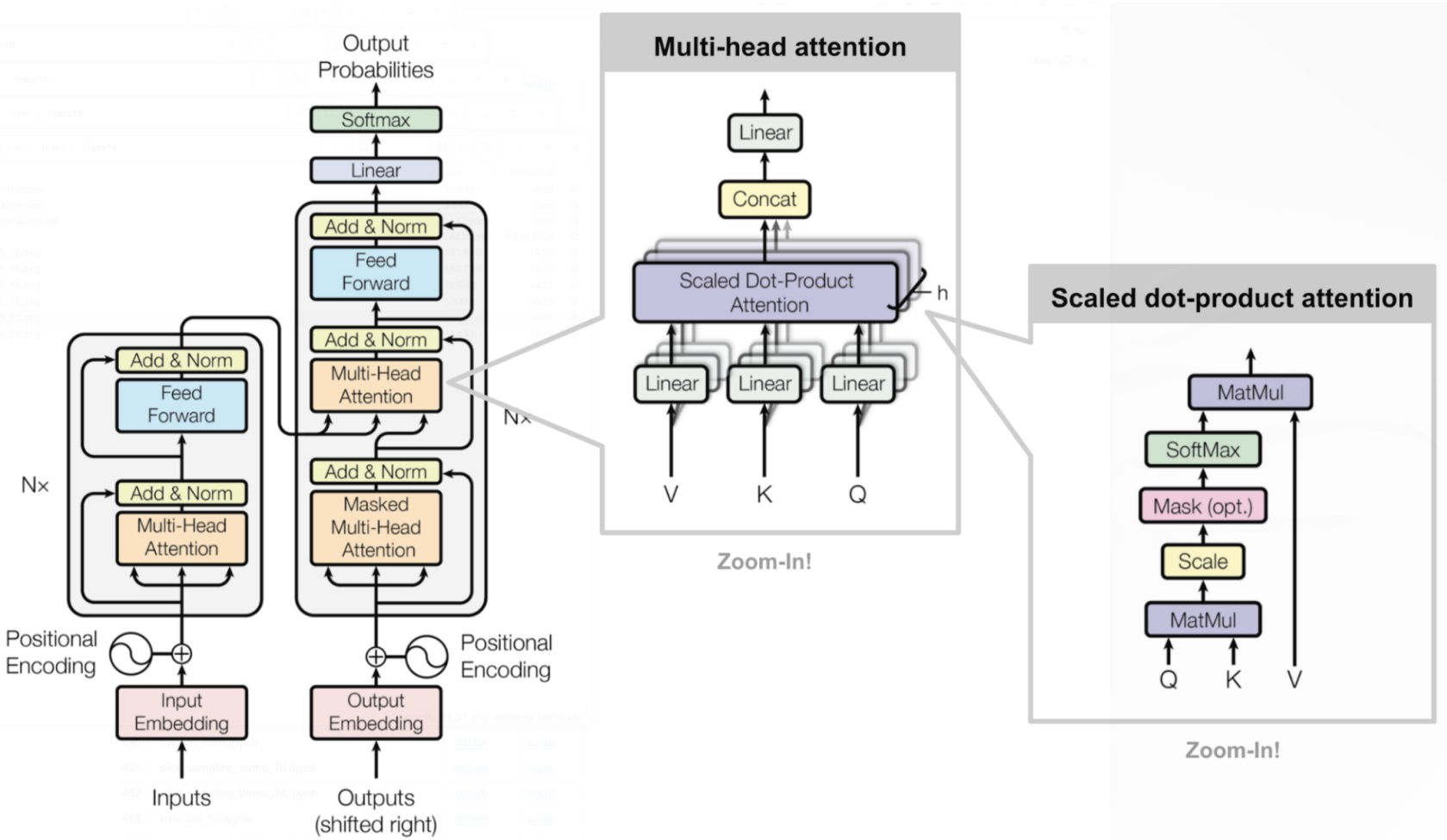
```
def DecoderBlock(Y, E):
  Z = LayerNorm(MultiHeadAttn(Q=Y,K=Y,V=Y)+Y)
  Z' = LayerNorm(MultiHeadAttn(Q=Z,K=E,V=E)+Z)
  D = LayerNorm(FeedForward(Z') + Z')
  return D
```

```
def Decoder(Y, E, N):
  D = POS(Embed(Y))
  for n in range(N):
    D = DecoderBlock(D,E)
  return D
```



https://commons.wikimedia.org/wiki/File:Transformer,_one_decoder_block.png

Transformer [cont'd]



Source: Murphy, Fig. 15.26

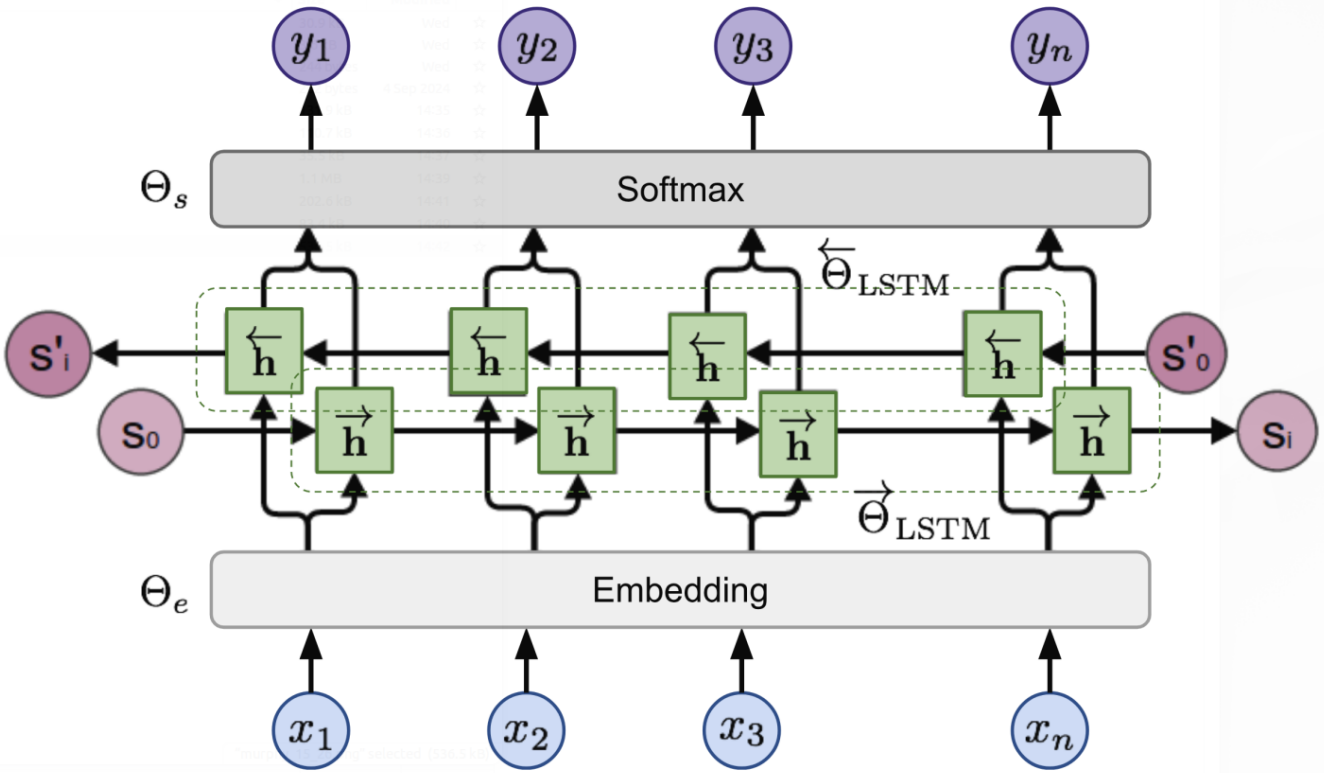
Transformer Encoder in Python

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super().__init__()
        self.att = layers.MultiHeadAttention(num_heads,
                                             embed_dim)
        self.ffn = keras.Sequential([
                layers.Dense(ff_dim, activation="relu"),
                layers.Dense(embed_dim)])
        self.norm1 = layers.LayerNormalization()
        self.norm2 = layers.LayerNormalization()
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, inputs):
        att_out = self.att(inputs, inputs)
        att_out = self.dropout1(att_out)
        out1 = self.norm1(inputs + att_out)
        ffn_out = self.ffn(out1)
        ffn_out = self.dropout2(ffn_out)
        return self.norm2(out1 + ffn_out)
```

MEMORIAL UNIVERSITY

ELMo –"Embeddings from Language Model"

- Words learned in context of past and future words
- Bi-directional (multi-layered) LSTM
- Word/token representation is concat of of embedding layer and outputs of both LSTM networks
- Self-supervised learning for pre-training and subsequent fine-tuning

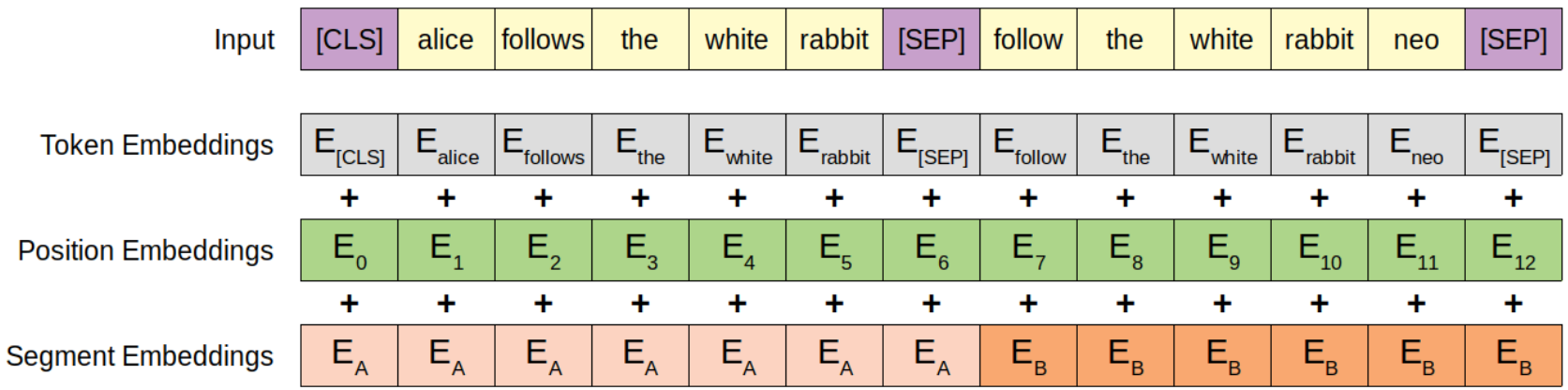


Source: Murphy, Fig. 15.32

BERT

- ”Bidirectional Encoder Representations from Transformers”
- Encoder-only
- Stacked set of transformers
- Learns representations of tokens in context of past and future words

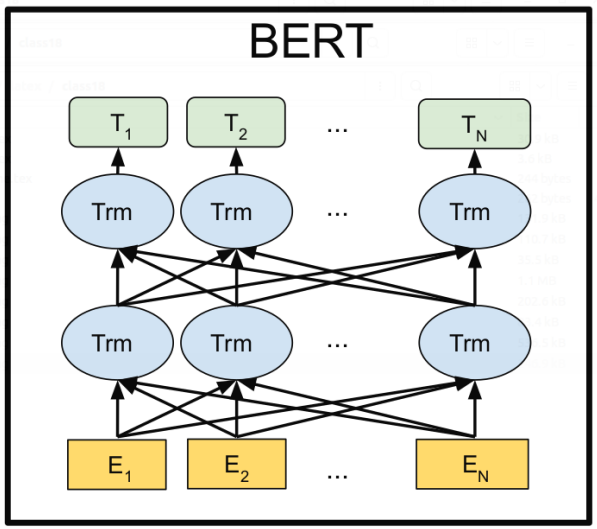
Three embeddings for teach token:

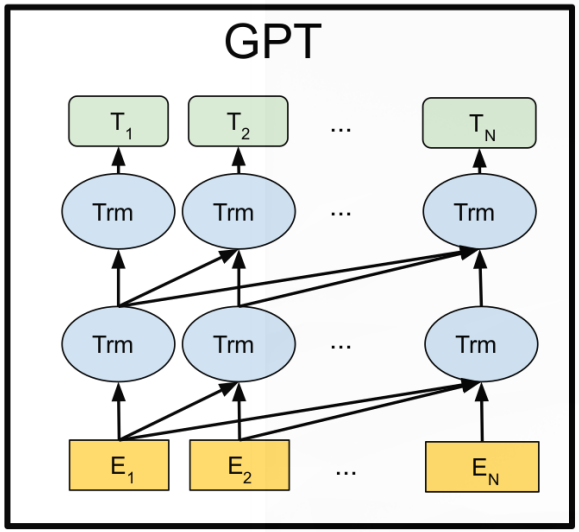


`https://commons.wikimedia.org/wiki/File:BERT_input_embeddings.png`

MEMORIAL UNIVERSITY

Comparison of BERT and GPT Architectures

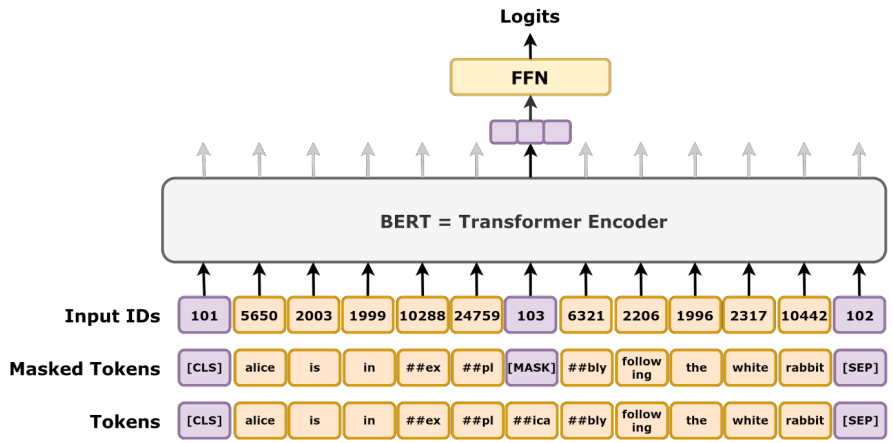




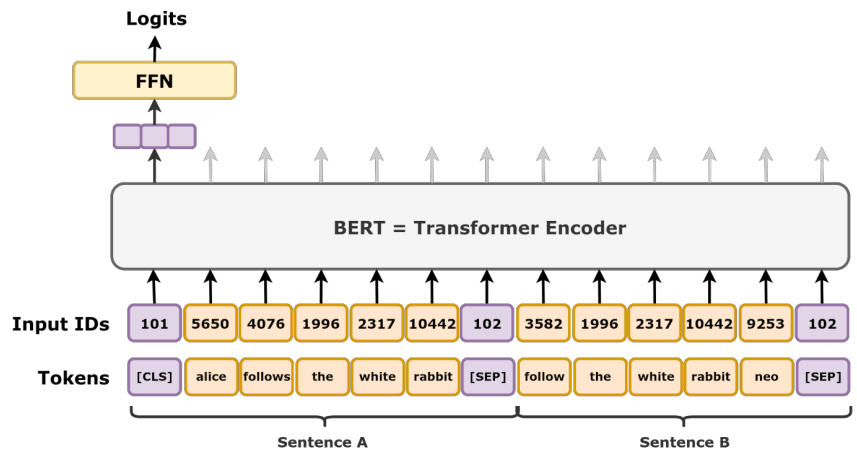
Source: Murphy, Fig. 15.33

BERT – Pre-Training Tasks [cont'd]

- Masked Language Modelling
- Next sentence prediction



https://commons.wikimedia.org/wiki/File:BERT_masked_language_modelling_task.png

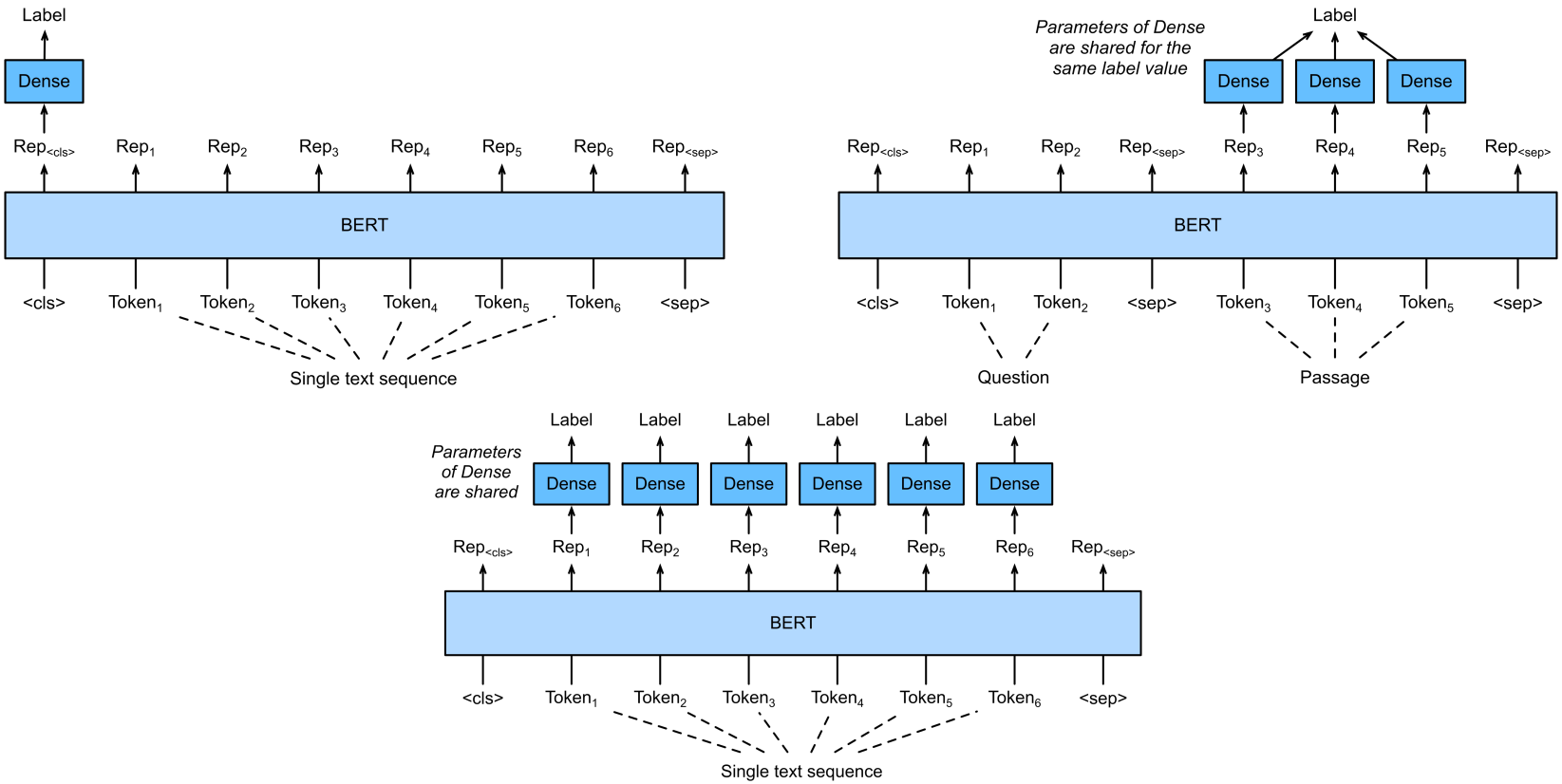


https://commons.wikimedia.org/wiki/File:BERT_next_sequence_prediction_task.png

MEMORIAL UNIVERSITY

BERT – Example Fine-Tuning Tasks

- Sentence classification
- Question answering
- Part-of-speech tagging



https://commons.wikimedia.org/wiki/File:BERT_on_sentence_classification.svg
https://commons.wikimedia.org/wiki/File:
BERT_on_multiple-choice_question-answering.svg
https://commons.wikimedia.org/wiki/File:BERT_on_tagging.svg

MEMORIAL UNIVERSITY

Tensorflow and Keras Tutorials

Keras

- Text generation with a miniature GPT
- Text classification with a transformer
- English-to-Spanish translation with seq2seq transformer
- Semantic similarity with BERT

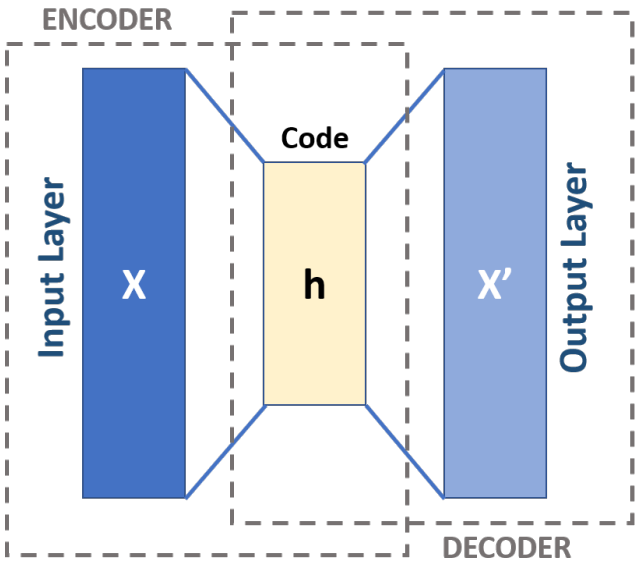
Tensorflow

- Neural machine translation with a Transformer and Keras
- Classify text with BERT
- Fine-tuning a BERT model

MEMORIAL UNIVERSITY

Auto-Encoder

Map input to *point* in lower-dimensional latent space:



https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png

Usage

- Face recognition
- Feature detection
- Anomaly detection
- Data synthesis (by adding noise to points in latent space)

Auto-Encoder Examples

Fashion MNIST – Original and Reconstruction

- *Left:* 3-layer densely connected network (decoder inverse)
- *Right:* 3-layer CNN (decoder inverse)

(a) (b)

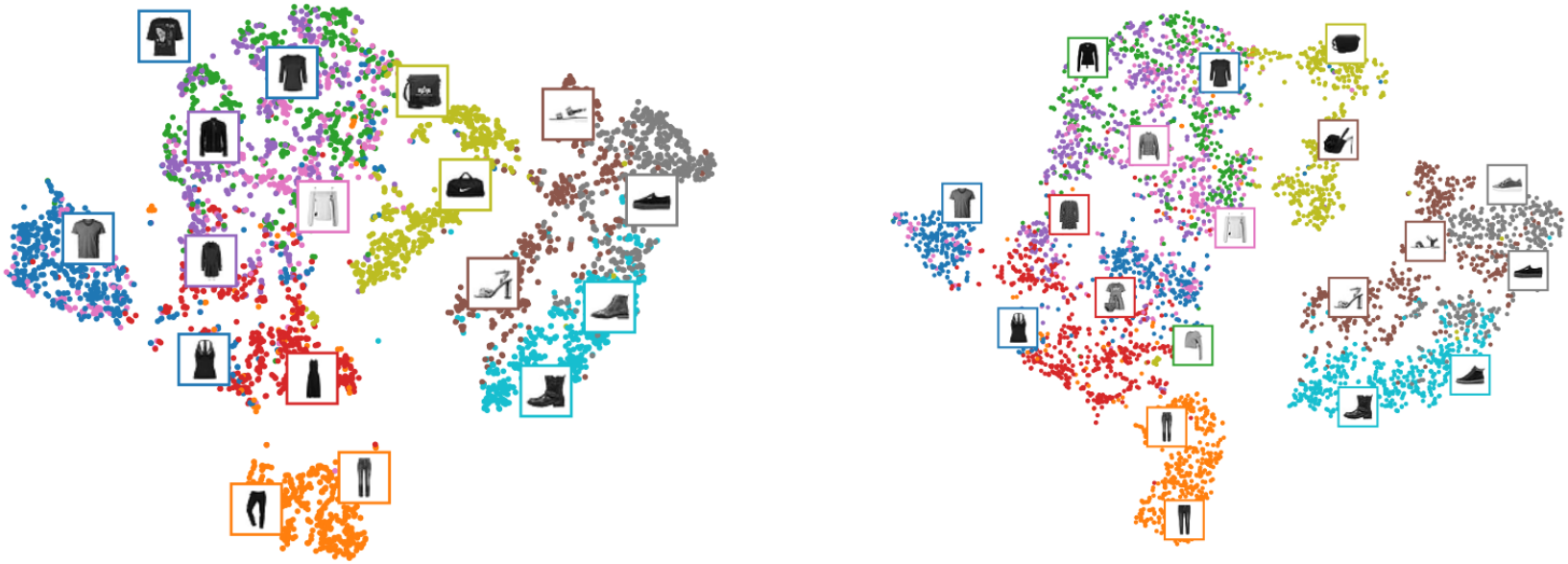
Source: Murphy, Fig. 20.17

MEMORIAL UNIVERSITY

Auto-Encoder Examples

Fashion MNIST – First 2 Latent Dimensions

- *Left:* 3-layer densely connected network (decoder inverse)
- *Right:* 3-layer CNN (decoder inverse)



Source: Murphy, Fig. 20.18

Auto-Encoder in Python

Implementations are available on the following GitHub repo:
`https://github.com/jevermann/busi4720-ai`

The project can be cloned from this URL:
`https://github.com/jevermann/busi4720-ai.git`

Important

While the encoder and decoder in the following examples have the same architecture, this need not be the case! One could encode with a CNN and decode with a dense network, or any other combination of architectures.

MEMORIAL UNIVERSITY

Auto-Encoder in Python [cont']

Encoder:

```
from tensorflow.keras import layers
from tensorflow.keras import models
import keras

encoder = models.Sequential([
    layers.Flatten(),
    layers.Dense(1000),
    layers.Dropout(0.25),
    layers.Dense(500),
    layers.Dropout(0.25),
    layers.Dense(30)
])
```

Decoder:

```
decoder = models.Sequential([
    layers.Dense(500),
    layers.Dense(1000),
    layers.Dense(28*28),
    layers.Reshape(target_shape=(28, 28))
])
```

Auto-Encoder in Python [cont']

Complete auto-encoder model:

```
auto_encoder = models.Sequential([encoder, decoder])
```

Load and transform data:

```
(x_train, y_train), (x_test, y_test) = \
                    keras.datasets.fashion_mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Compile and train the auto-encoder:

```
auto_encoder.compile(loss="mse")
auto_encoder.fit(x_train, x_train,
                 epochs=5, validation_data=(x_test, x_test))
```

MEMORIAL UNIVERSITY

CNN Auto-Encoder in Python [cont']

Encoder:

```
encoder = models.Sequential([
    layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    layers.Conv2D(16, (3, 3),
                  padding="same", activation="relu"),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(32, (3, 3),
                  padding="same", activation="relu"),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(64, (3, 3),
                  padding="same", activation="relu"),
    layers.MaxPool2D((2, 2))])
```

Decoder:

```
decoder = models.Sequential([
    layers.Conv2DTranspose(32, (3, 3), strides=2,
                activation="relu", input_shape=[3, 3, 64]),
    layers.Conv2DTranspose(16, (3, 3), strides=2,
                padding="same", activation="relu"),
    layers.Conv2DTranspose(1, (3, 3), strides=2,
                padding="same", activation="sigmoid"),
    layers.Reshape([28, 28])])
```

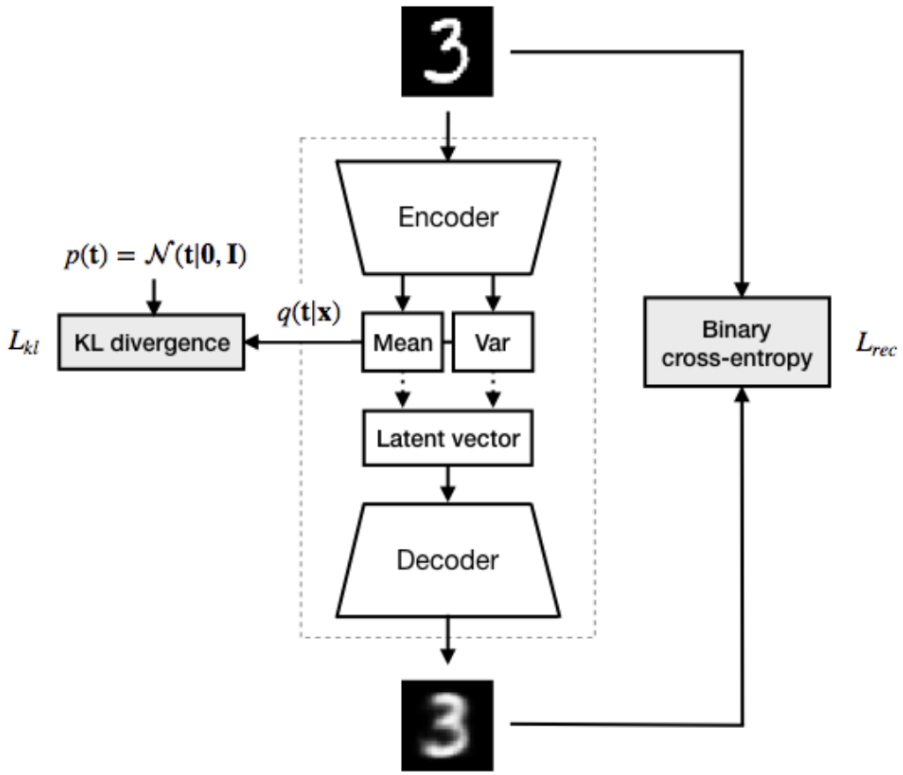
Auto-Encoder – Denoising

- Add noise to input
- Reconstruct clean output
- Reconstruction/recognition of ”corrupted” images

(a) (b)

Source: Murphy, Fig. 20.19

VAE – Variational Auto-Encoder



Source: Murphy, Fig. 20.22

Example: Map into Gaussian with mean vector μ and covariance matrix Σ

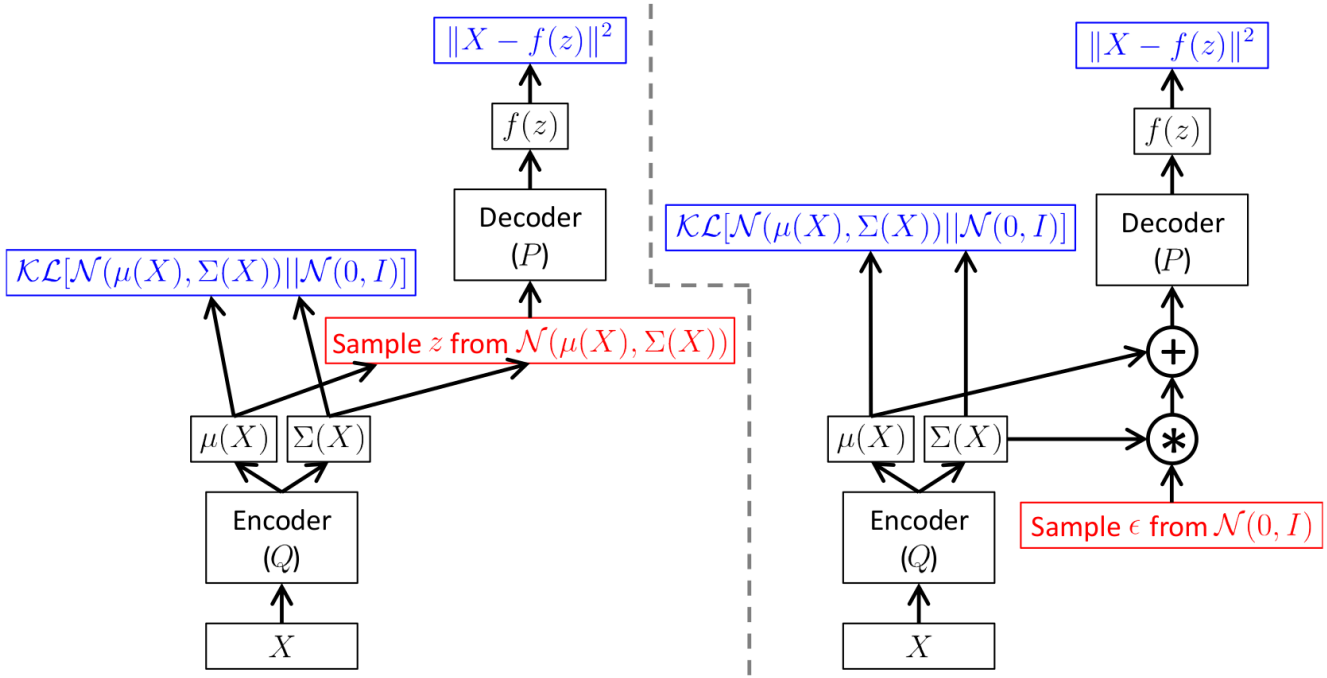
Principles

- Map input to *probability distribution* in lower - dimensional latent space.
- Sample a point in latent space from that distribution
- Loss is reconstruction loss plus (KL) distribution divergence loss, e.g. from normal

MEMORIAL UNIVERSITY

VAE – Variational Auto-Encoder

- Training and Reparameterization:



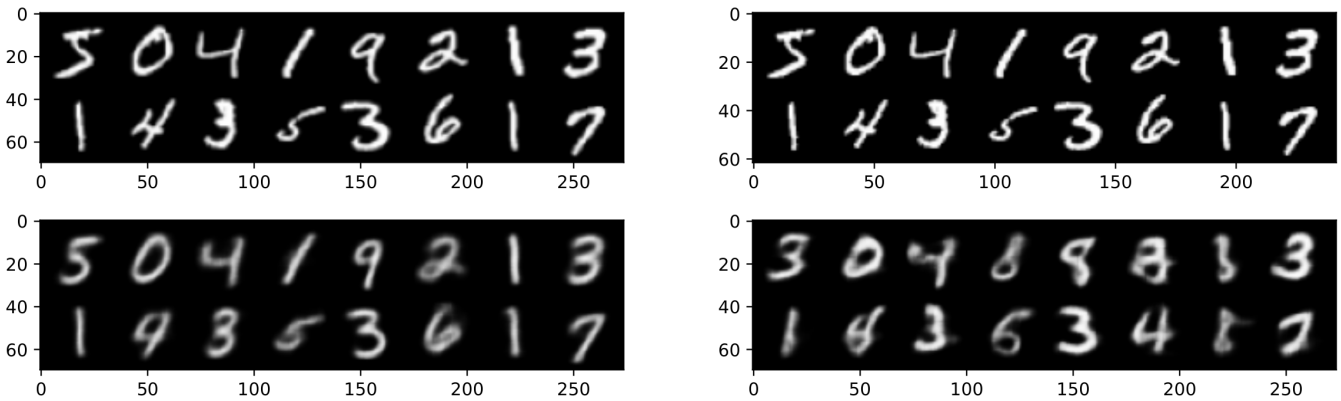
Source: Murphy, Fig. 20.23

- Ensures contiguous use of latent space
- Allows generation by sampling from a Gaussian and decoding (in contrast to deterministic AE)

MEMORIAL UNIVERSITY

VAE – Variational Auto-Encoder

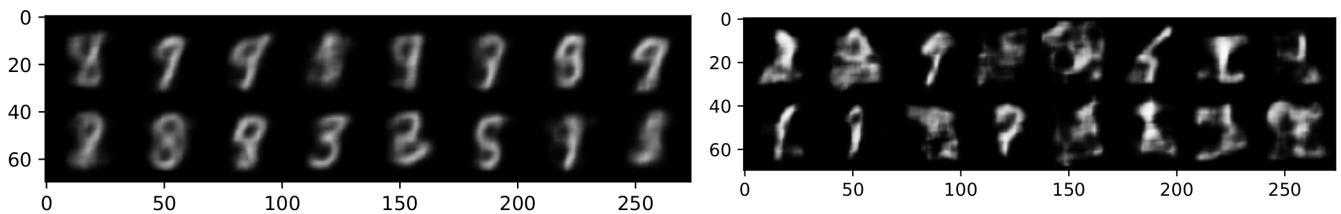
Reconstruction: VAE (a) — AE (b)



(a) *(b)*

Source: Murphy Fig. 20.24

Generation: VAE (a) — AE (b)



(a) *(b)*

Source: Murphy Fig. 20.25

VAE in Python

Adapted from:

`https://keras.io/examples/generative/vae/`

Implementation available on the following GitHub repo:

`https://github.com/jevermann/busi4720-ai`

The project can be cloned from this URL:

`https://github.com/jevermann/busi4720-ai.git`

MEMORIAL UNIVERSITY

VAE in Python [cont']

Sampling layer for sampling from a multi-variate Gaussian with given mean and log-variance vectors:

```
import tensorflow as tf
import keras
from tensorflow.keras import layers

class Sampling(layers.Layer):
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.random.normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

MEMORIAL UNIVERSITY

VAE in Python [cont'd]

Encoder:

```
# This is NOT a sequential model
encoder_inputs = tf.keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, (3,3), activation="relu",
                  strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, (3,3), activation="relu",
                  strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)

# Vector of means
z_mean = layers.Dense(10, name="z_mean")(x)

# Vector of log variances
z_log_var = layers.Dense(10, name="z_log_var")(x)

# Point in latent space
z = Sampling()([z_mean, z_log_var])

encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z])
```

MEMORIAL UNIVERSITY

VAE in Python [cont'd]

Loss function:

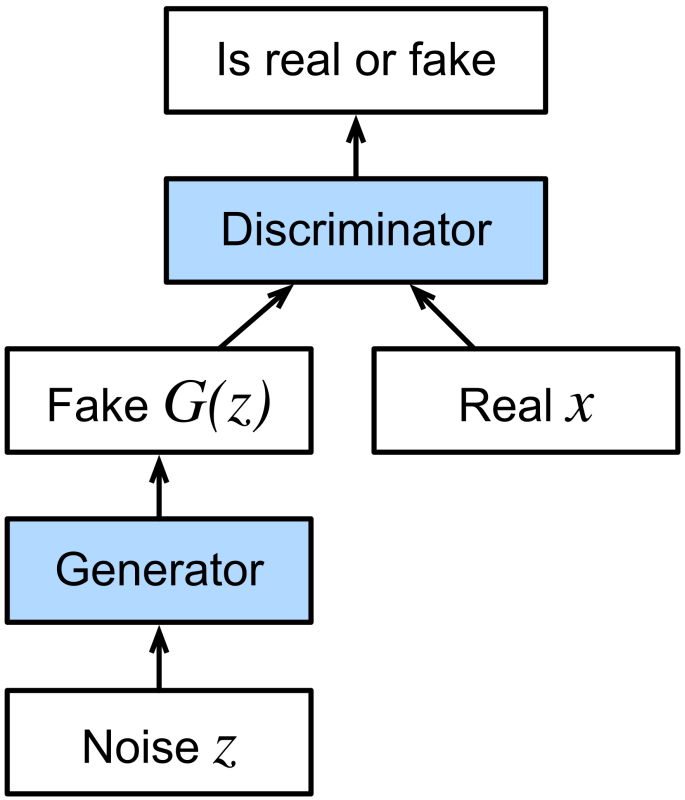
```
    def vae_loss(encoder, decoder, data):
        z_mean, z_log_var, z = encoder(data)
        reconstruction = decoder(z)
        mse_loss = keras.losses.MeanSquaredError()
        reconstruction_loss = mse_loss(data, reconstruction)
        kl_loss = -0.5 * (1 +
                          z_log_var -
                          tf.square(z_mean) -
                          tf.exp(z_log_var))
        kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss,axis=1))
        total_loss = reconstruction_loss + kl_loss
        return total_loss
```

- Combination of reconstruction loss and KL loss ("difference" to standard normal distribution)

MEMORIAL UNIVERSITY

GAN – Generative Adversarial Networks

- Generator produces data from a latent representation
- Discriminator attempts to distinguish generated output and true data
- Generator is trained to generate data that discriminator classifies as real



https://commons.wikimedia.org/wiki/File:Generative_adversarial_network.svg

MEMORIAL UNIVERSITY

GAN – Generative Adversarial Networks [cont'd]

Image generated by a GAN:

https://commons.wikimedia.org/wiki/File:GAN_Katze_StyleGAN2.png

MEMORIAL UNIVERSITY

GAN in Python

Adapted from:

`https://keras.io/examples/generative/dcgan_overriding_train_step/`

Implementation available on the following GitHub repo:

`https://github.com/jevermann/busi4720-ai`

The project can be cloned from this URL:

`https://github.com/jevermann/busi4720-ai.git`

MEMORIAL UNIVERSITY

GAN in Python

Discriminator receives image to classify:

```
discriminator = keras.Sequential([
    layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    layers.Conv2D(16, (3, 3),
        padding="same", activation="relu"),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(32, (3, 3),
        padding="same", activation="relu"),
    layers.MaxPool2D((2, 2)),
    layers.Conv2D(64, (3, 3),
        padding="same", activation="relu"),
    layers.MaxPool2D((2, 2)),
    layers.Flatten(),
    layers.Dropout(0.2),
    layers.Dense(1, activation="sigmoid")
])
discriminator.summary()
```

- In this example, the discriminator model is similar to the encoder of the AE example

GAN in Python [cont'd]

Generator receives latent space vector to generate image from:

```
latent_dim = 10

generator = keras.Sequential([
    keras.Input(shape=(latent_dim,)),
    layers.Dense(3 * 3 * 10),
    layers.Reshape((3, 3, 10)),
    layers.Conv2DTranspose(32, (3, 3), strides=2,
        padding="valid", activation="relu"),
    layers.Conv2DTranspose(16, (3, 3), strides=2,
        padding="same", activation="relu"),
    layers.Conv2DTranspose(1, (3, 3), strides=2,
        padding="same", activation="sigmoid"),
    layers.Reshape([28, 28])
])
generator.summary()
```

- In this example, the generator model is similar to the decoder of the AE example

MEMORIAL UNIVERSITY

GAN in Python [cont'd]

Training step for a single batch of real images:

```
class GAN(keras.Model):
    def train_step(self, real_images):
        # Sample random points in the latent space
        batch_size = tf.shape(real_images)[0]
        rnd_latent_vectors \
            = tf.random.normal((batch_size, self.latent_dim))

        # Decode them to fake images
        gen_images = self.generator(rnd_latent_vectors)

        # Combine them with real images
        images = tf.concat([gen_images, real_images], axis=0)

        # Assemble labels discriminating real from fake images
        labels = tf.concat(
            [tf.ones((batch_sz, 1)),
            tf.zeros((batch_sz, 1))], axis=0)
        # Add random noise to the labels - important trick!
        labels += 0.05 * tf.random.uniform(tf.shape(labels))
```

MEMORIAL UNIVERSITY

GAN in Python [cont'd]

Training the discriminator:

```
        # Train the discriminator
        with tf.GradientTape() as tape:
            predictions = self.discriminator(images)
            d_loss = self.loss_fn(labels, predictions)

        grads = tape.gradient(
            d_loss,
            self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights))
```

MEMORIAL UNIVERSITY

GAN in Python [cont'd]

Training the generator:

```
        # Sample random points in the latent space
        random_latent_vectors = \
            tf.random.normal((batch_size, self.latent_dim))

        # Assemble labels that say "all real images"
        misleading_labels = tf.zeros((batch_size, 1))

        # Train the generator
        with tf.GradientTape() as tape:
            predictions = self.discriminator(
                self.generator(random_latent_vectors))
            g_loss = self.loss_fn(
                misleading_labels, predictions)

        grads = tape.gradient(
            g_loss,
            self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))
```

MEMORIAL UNIVERSITY

GAN Training

Problems

- Loss not necessarily decreaseing as generator and discriminator alternatingly improve
- Discriminator may ”overpower” generator
  - Reduce discriminator learning rate
  - Remove CNN layers or reduce number of filters
  - Add dropout layers or increase dropout
- Generator may ”overpower” discriminator

MEMORIAL UNIVERSITY