Text and Images together

TL;DR

- Text is represented as
    - a sequence of integers: an integer index into the list of tokens in the vocabulary

In this module, we show how to mix Text and Images

- **outside** the context of an LLM
    - no tokenization

This is useful for tasks that are different than the Language Model task ("predict the next token").

In particular, we show how to create an Image Classifier

- Image input, text output
    - output is a label or a caption of the input image

The standard "Computer Vision" task is to learn an association between images and a predetermined set of labels.

Highly successful image classifiers have thus been obtained.

But they are directly useful only for images from the same distribution as the Training dataset

Transfer Learning is necessary to adapt the Image Classifier to images that are different than training

- e.g.,different labels

The traditional Classifier architecture

- assigns disjoint roles for the two types of data
    - Images are Features
    - Text are Labels
- Can't learn to associate (parts of) Images with semantically related Text
- usually requires hand-labeled Images
    - expensive

We introduce a model called CLIP (Contrastive Image Language-Image Pre-training)

- that treats Image and Text in a common role

This, in turn, will facilitate the creation of an Image Classifier

- **without** Transfer Learning/Fine-Tuning to a new Target domain

The result is

- Zero-shot Image Classification of images *distinct from a training dataset*

And

- it is trained on an *abundant* source of image/text pairs
    - no expensive construction of a hand-labeled training dataset

In particular, we highlight the Loss Function

- Contrastive Loss

that illustrates how the Loss functions of the Advanced course are more complex than the simple ones of the Intro course.

A shared embedding space for Image and Text

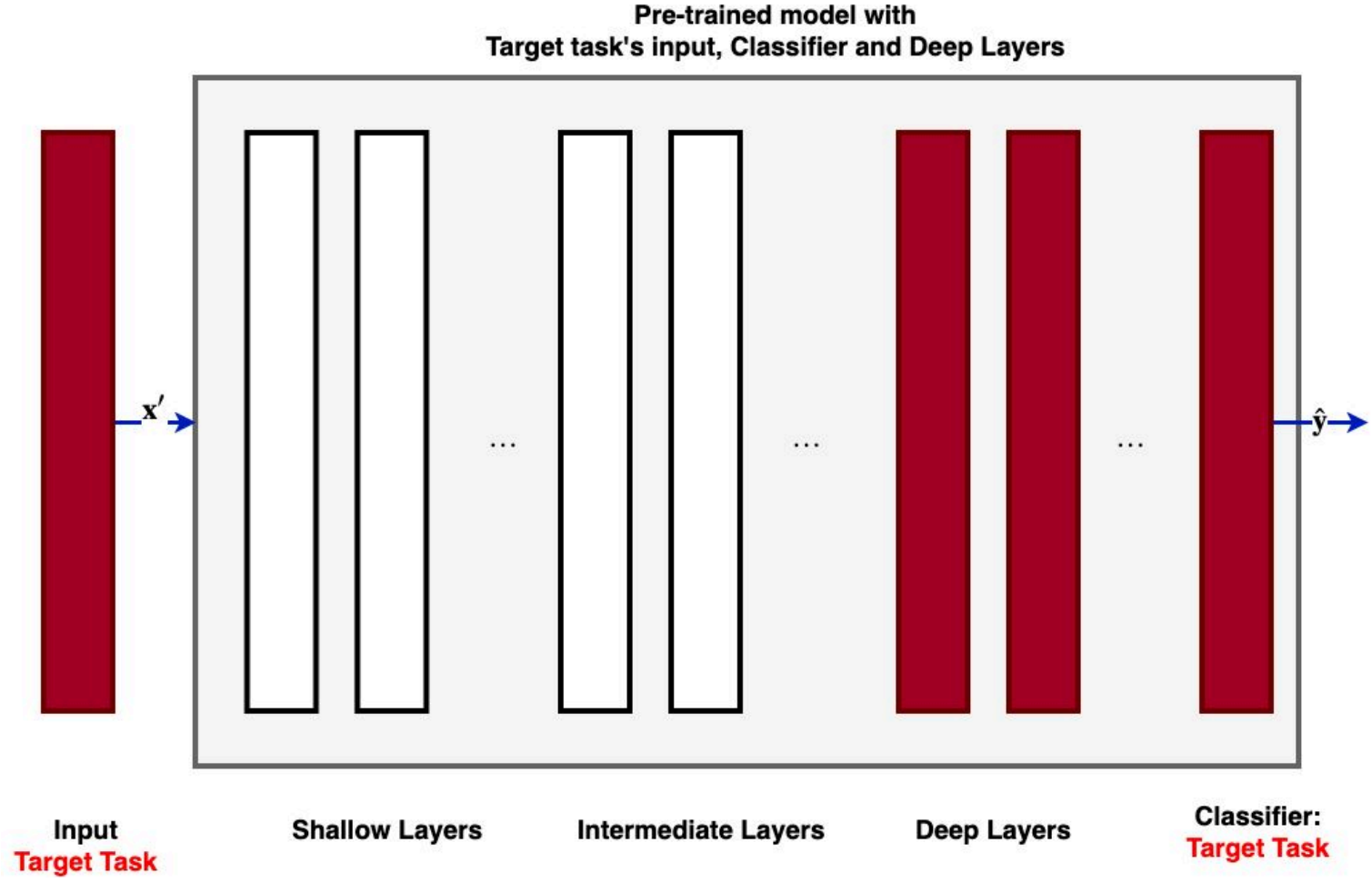
Image embedding space

A traditional Image Classification model consists of

- multiple layers prior to the "Classification head"
    - that creates alternate representations of the Input image
    - of increasing complexity as we go Deeper (closer to the Classification Head)

The purpose of the layers from the Input to the Classification Head

- is to create a representation
- from which the Classification Head can correctly predict the label of an Image


Pre-trained model with
Target task's input, Classifier and Deep Layers
x′
...
...
...
ŷ
Input
Target Task
Shallow Layers
Intermediate Layers
Deep Layers
Classifier:
Target Task

We call the alternative representation of an Image

- an *Image Embedding*

Transfer Learning (illustrated above)

- uses the Image Embedding of some Deep Layer of a Source Task Image Classifier
- to *transfer* properties that is common to
    - the Source Task Images
    - the Target Task Images
- in order to solve a new Target Task

So an Embedding captures properties of generalized Images

- independent of the Source and Target task

The traditional Image Classifier

- learns an association between an Image and Text

But this is association is very shallow

- no **meaning** is associated with the text labels

This is because the text labels

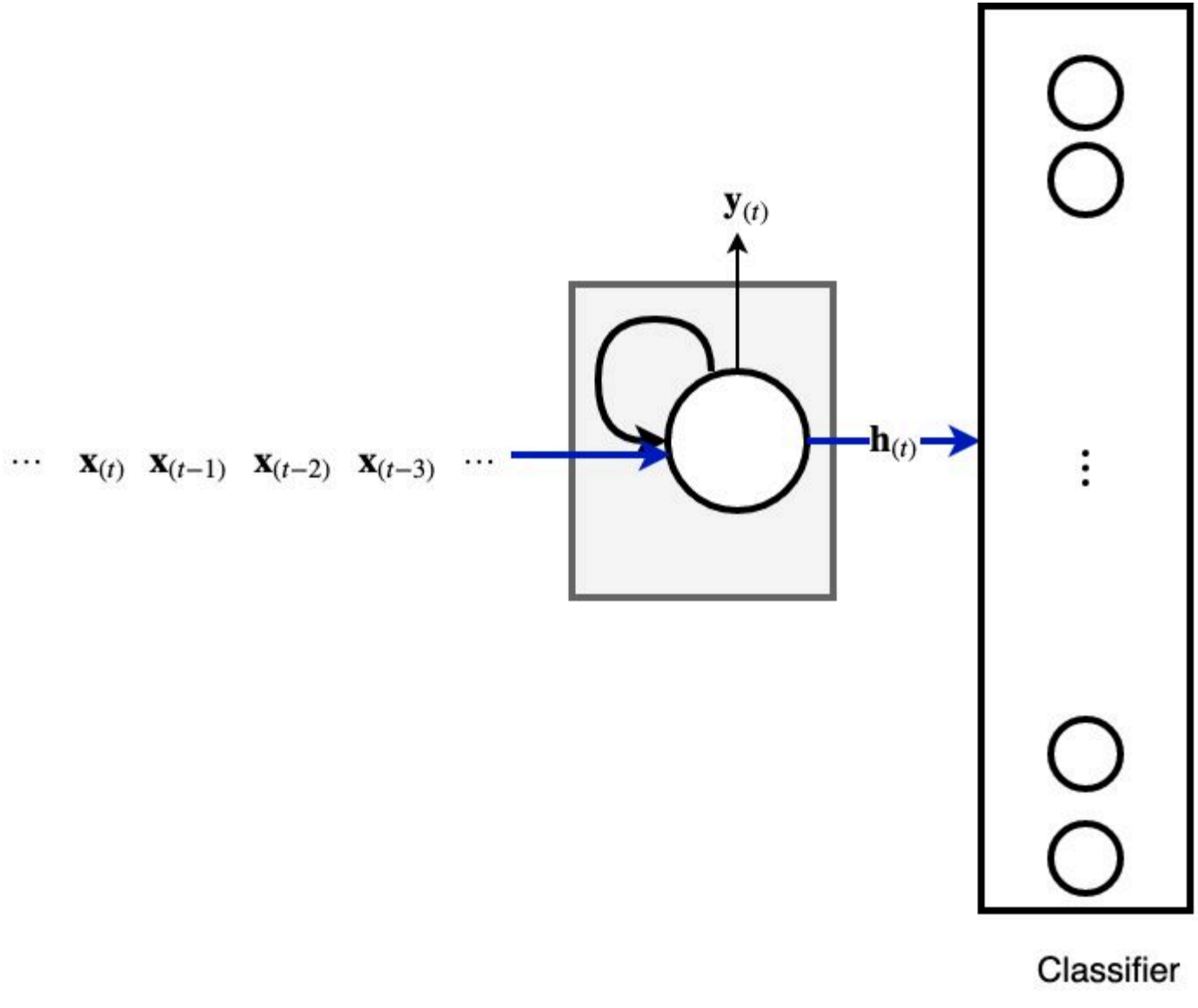
- are OHE
    - mutually orthogonal
- so no semantic association is possible between
    - different labels/different images

Text embedding space

"Language Models" ("predict the next token") have demonstrated a great ability for creating representations of text sequences

- Seem to capture semantics
- The fixed length "summary" of a text sequence is a *text embedding*
- Facilitate zero shot learning
    - Universal "text to text" API for all language tasks
    - Single model can "learn" to solve a new task without adjustment of weights

Latent state $\mathbf{h}_{\tp}$ is a fixed length "summary" of $\mathbf{x}_{(1)}, \ldots \mathbf{x}_{\tp}$



We call the fixed length vector representation of a Text sequence (alternative representation) -a *Text embedding*

These embeddings capture a lot of semantic concepts of Text

- in contrast to the Image Classifier's lack of "understanding" of the Labels

Creating a common embedding space

We can try to use

- Image embeddings
- Text embeddings

together in a single model

- to create an Image Classifier
- with a deeper association between Images and layer
    - compared to the traditional Image Classifier

One problem with this idea

- the Image and Text embedding spaces
- are disjoint
    - trained separately
    - maybe even the lengths of the embedding vectors differ

CLIP creates a *joint embedding space* for Image and Text.

- The embedding space of Images
- The embedding space of Text
- are combined into a *common* embedding space

Given

- an Image Encoder that embeds image I into an Image Embedding I_f

$$I_f = \text{image\_encoder}(I)$$

- a Text Encoder T_f embedding text sequence T into a Text Embedding T_f

$$T_f = \text{text\_encoder}(T)$$

CLIP defines two matrices that create a *common* embedding space

- matrix W_i to project from Image Embedding space to common space
- matrix W_t to project from Text Embedding space to common space

The Image and Text embeddings in the common space

- Image embedding I_f projected into common space

$$I_e = I_f \cdot W_i$$

- Text embedding I_t projected into common space

$$T_e = T_f \cdot W_t$$

CLIP trains a model using training data examples that map Images to Text

- just like any other Image Classifier

For a training example associating text sequence T (label, caption) with image i

$$\langle I, T \rangle$$

CLIP training definesmatrices W_i, W_t

- that *maximize the similarity* of the embeddings of I and T in the common space

$$I_e \cdot T_e$$

That is

- the embedding of the Image and its associated Text
- are close in common embedding space

CLIP loss: Contrastive Learning

The loss function and training for CLIP is of particular interest.

An important part of the Model is the *Contrastive Training* objective

- Minimize the distance between *correct* Image/Text Pairs
- Maximize the distance between *incorrect* Image/Text pairs

This type of objective is used in many places in Deep Learning

- so is worthy to introduce as an independent concept

Zero shot Image Classification

With a common Image/Text embedding space, Image Classification is easy

- does not require further training for a type of Image not seen in training

Simply

- embed all the Target labels into common embedding space
- given an Image I at inference time
    - embed the image in common embedding space: I_e
    - compute the dot product of I_e with the embedding of each Target label
    - select the Target label T with greatest similarity

$$I_e \cdot T_e$$

Details

Notation summary

term	dimension	meaning
n		number of examples in a training batch
N		number of examples in the training dataset
I	$(n \times h \times w \times c)$	Image batch
		image dimension $(h \times w \times c)$
T	$(n \times l)$	Text labels for batch
d_i		dimension of Image embedding
d_t		dimension of Text embedding
d_e		dimension of common embedding
W_i	$(d_i \times d_e)$	learned projection of image embedding to common embedding
W_t	$(d_t \times d_e)$	learned projection of image embedding to common embedding
t		learned temperature parameter (used in softmax)
I_f	$(n \times d_i)$	embedding of Image batch
T_f	$(n \times d_t)$	embedding of Text (label) batch
	$(n \times d_e)$	size of common embedding (for each of the n Image and Labels)
logits	$(n \times n)$	$\text{logits}_{i,j}$ is similarity of image i to label j
$\backslash\text{loss}_i$	1	Loss across images: reduce $(\text{logits} \cdot \text{labels})$ across text dimension
		For each image: compare image's logits to labels
$\backslash\text{loss}_t$	1	Loss across labels: reduce $(\text{logits} \cdot \text{labels})$ across image dimension
		For each label: compare labels's logits to images

Contrastive pre-training

CLIP is trained to solve an Image Classification task

- Associate the best text sequence description
- To an input Image

The "standard" approach is to jointly train

- an image feature extractor (e.g., a CNN)
- with a Classifier "head"
    - the labels are sparse OHE vectors representing "word" labels

In contrast, CLIP jointly trains

- an Image encoder
- a Text sequence encoder
- with the objective of
    - matching a Training image with its associated Text
    - the text is a semantically meaningful sequence of "words" rather than a OHE vector

The key is for the Image Encoder and Text Encoder to produce embeddings in a common space.

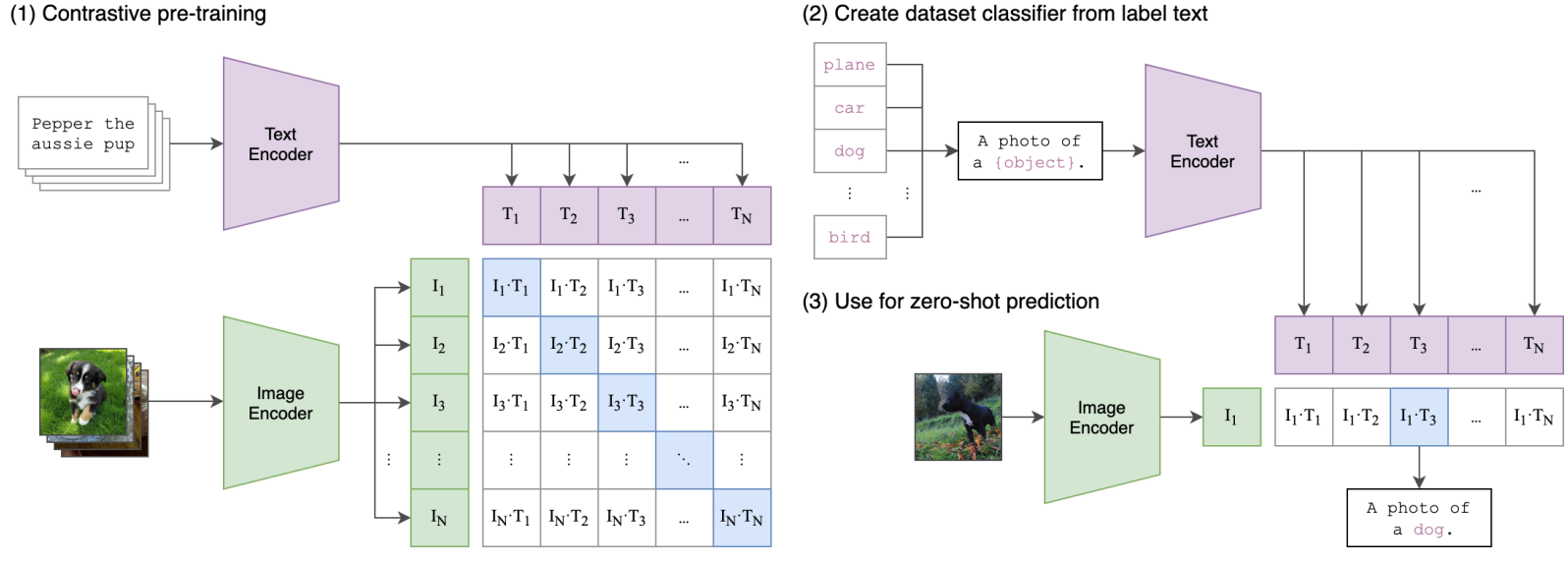
The method of *Contrastive Learning*

- is to learn features (Image/Text embeddings in the common space)
- that make related concepts (an Image and correct Text) "similar"
    - high similarity, low distance

Hopefully, the picture (labeled "(1)") describing the *training* will help:

- A training example is an (Image, Text) pair: (I_i, T_i)
- The Image I_i is encoded into the joint embedding space by the Image Encoder
- It's Text T_i is encoded into the joint embedding space by the Text Encoder
- The matrix is a "similarity" (inverse of distance) between all Images and all Text labels
    - the similarity is defined by the dot product of the Image and Text embeddings

CLIP architecture



Contrastive training details

Examples

Given a set of labeled training examples

$$\langle I_i, T_i \rangle \text{ for } 1 \leq i \leq m$$

Choose

- an Image Encoder:
    - d_i is the length of image embeddings
- a Text Encoder:
    - d_t is the length of text embeddings

each creating separate embedding spaces.

Model parameters

Define learnable model parameters

- matrix W_i, W_t

$$W_i : \quad (d_i \times d_e)$$
$$W_t : \quad (d_t \times d_e)$$

These matrices enable project of each example image I_i and label T_i into common space.

Similarity matrix

Create a similarity matrix S

$$S : (m \times m)$$

that compares each image in the training data set with each label in the training data set.

$$\begin{array}{lll} S^{\backslash \text{ip}} & \text{Row } i & \text{comparison of image } I_i \\ S_j & \text{Column } j & \text{comparison of Text label } T_j \end{array}$$

$$S_j^{\backslash \text{ip}} = \mathcal{I}_i \cdot \mathcal{T}_j$$

where $\mathcal{I}_i$, $\mathcal{T}_j$ are the embeddings *in common space* of

- Image I_i
- Text label T_j

This is the matrix on the left side of the diagram.

Rather than using the intensity of the dot product

- we convert to probabilities
- using the Softmax on each row

when defining each of the components of the Loss.

Doing so enables us to create

- a probability distribution $p^{\backslash \mathrm{ip}}$
- using row $S^{\backslash \mathrm{ip}}$
- over Text labels $\{T_j \mid 1 \leq j \leq m\}$

representing the probability that image I_i is each of the possible labels.

and to create

- a probability distribution p_j
- using column S_j
- over Images $\{I_i \mid 1 \leq i \leq m\}$

representing the probability that Text T_j corresponds to each possible Image.

Contrastive Loss function

Suppose we have a single (Image, Text) pair (I_i, T_i) from the set of m training examples.

Our objective is to ensure that the similarity in common space is such that

- $\mathcal{I}_i \cdot \mathcal{T}_i$ is as large as possible
- $\mathcal{I}_i \cdot \mathcal{T}_j$ is as small as possible for $i \neq j$

That is: we maximize the *contrast* between

- an image and its correct label
- and the image with any incorrect label

We define two Losses over the entries of Similarity Matrix (probability form) S,

We create an *Image Loss* for example i : $\loss_i^{\ip}$.

Recall that $p^{\ip}$ is the probability distribution over Text labels.

- for Image I_i

Entry i in this distribution vector is the entry for the *correct* label T_i

We can define

- target probability vector $t^{\backslash \text{ip}}$ for example i
- as the OHE vector
    - where $t_i^{\backslash \text{ip}} = 1$ is the only non-zero entry

Minimizing the Cross Entropy loss of $p^{\backslash \text{ip}}$ and $t^{\backslash \text{ip}}$

- states the goal of having
    - $p_i^{\backslash \text{ip}}$ as close to 1 as possible
    - $p_j^{\backslash \text{ip}}$ as close to 0 as possible for $j \neq i$

We can similarly define a *Text Loss* for example i : $\loss_t^{\ip}$.

Recall that p_i is the probability distribution over Images

- for Text label T_i

Entry i in this distribution vector is the entry for the *correct* image I_i

As above

- we create Text Loss for example i using Cross Entropy.

Minimizing Cross Entropy loss of p_i and t_i

- states the goal of having
  - p_i^{ip} as close to 1 as possible
  - $p^{(j)}$ as close to 0 as possible for $j \neq i$

Total loss

The per example loss is the sum of the per example Image and Text losses

$$\loss^{\ip} = \loss_i^{\ip} + \loss_t^{\ip}$$

and the Total Loss is the sum over the m per example losses.

Creating Contrastive examples

With a large number of examples m

- the similarity matrix becomes large

There is a clever trick to avoid computing the large matrix.

Essentially

- the per-example loss $\loss^{\ip}$ for example i
- involves the comparison over all m examples

Recall that mini-batch Gradient Descent is used in training Neural Networks.

Rather than comparing example i against

- all m training examples
- we compare it against
    - only the examples in the *same mini-batch* as example i

To simplify the computation further

- the target $t^{\setminus \text{ip}}$ (used in minimizing the Cross Entropy of $p^{\setminus \text{ip}}$)
- is constructed under the (imperfect) assumption
- that all examples $j \neq i$ in the mini-batch have labels that are different than true label T_i

This is called the *in-batch negatives* trick

The in-batch negatives trick is simple but crude.

- a "second best" label
- is considered equally bad
- as a "completely incorrect" label

Pseudo-code for Pre-Training

Pseudo code for training batch:

```
I_f = image_encoder(I)
T_f = text_encoder(T)

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)

loss = (loss_i + loss_t)/2
```

Pre-training dataset

In order to train CLIP successfully

- we need a large number of Image/Text pairs.

Other than manual labeling: where do these come from ?

The authors adopt a similar approach to obtaining training data for Image Classification

- Search the Web for Images that have captions
- The caption become the labels for the training image
- **Unbounded** number of distinct labels
    - many Texts to describe the same (or similar image)
        - "Picture of a cat"
        - "Picture of my cat named 'Kitty'"
        - unbounded length of label

This is one reason Text labels need to be embedded

- Need a fixed length representation an unbounded number of labels, each of unbounded length

Inference: Zero shot

The fact that CLIP can deal with arbitrary labels (in the force of Text sequences) creates the possibility of classifying Images

- from an *unseen* Target dataset
- with no further training (other than the initial pre-training on the Source dataset)

Being able to solve a Target task without specifically being trained with examples of the task

- is called **Zero shot** learning

The pictures (labeled "(2)" and "(3)") in the CLIP diagram above describe the process for predicting the label of a single image

- Given the finite set of labels from the Target Task
- Convert the short labels (nouns or phrases) into longer sentences
    - i.e., "Cat" becomes "photo of a cat"
- Embed the sentences into joint embedding space, resulting in $T_1, \ldots T_N$
- Embed the Target Image into joint embedding space, resulting in I_1
- Create the similarity matrix of dimension $(1 \times N)$
    - computing $I_1 \cdot T_j$, for each $1 \leq j \leq N$
- Predict $j^* = \backslash \mathrm{argmax} j I_1 \cdot T_j$

Discussion

Prompt Engineering

In the description of zero shot inference, short labels of the Target task were expanded into longer sequences of words.

The authors call this *prompt engineering* (i.e., creating new "prompts" for input.

They suggest that careful prompt engineering for each Target task can improve Zero shot classification.

For example

- converting a label (denoted by placeholder `{label}`) for a pet to

    A photo of a {label}, a type of pet

- can improve classification
    - helps with polysemy (two words with identical spelling but different meaning)
        - "crane": a bird; a piece of construction equipment
    - the extra words "photo" and "type of pet" become attributes of the image
        - that can be related (by textual similarity) to other images/labels through the Text embeddings

-

Theory

Representation of CLIP is much more detailed than other Image Classification models

- Other models only need to find representation that separates examples
    - may be hyper-specific and not generalize well
- CLIP needs to understand other details of the image **through the text**
    - difference between image formats: "photo", "illustration", "drawing"
    - sub-images that are mixed with the target sub-image
        - a "cat" in a group of "cats"

Example: Zero shot classification with Prompt Engineering

Let's explore Prompt Engineering using this Colab notebook (https://github.com/openai/CLIP/blob/main/notebooks/Prompt_Engineering_for_ImageNe

Goal is zero-shot classification of CIFAR image dataset

- 1000 classes
- most class labels are single word

Feature engineering transformed each CIFAR class label into *multiple* prompts using "templates"

```
imagenet_templates = [
    'a bad photo of a {}.',
    'a photo of many {}.',
    'a sculpture of a {}.',
    'a photo of the hard to see {}.',
    'a low resolution photo of the {}.',
    'a rendering of a {}.',
    'graffiti of a {}.',
    'a bad photo of the {}.',
    'a cropped photo of the {}.',
    'a tattoo of a {}.',
    'the embroidered {}.',
    'a photo of a hard to see {}.',
    'a bright photo of a {}.',
    'a photo of a clean {}.',
    'a photo of a dirty {}.',
    'a dark photo of the {}.',
    'a drawing of a {}.',
    'a photo of my {}.',
    'the plastic {}.',
```

Using forward selection, the authors selected the 7 "best templates"

- Created 7 prompts (e.g., texts) from each CIFAR class label
- Tokenized each prompt
- text-encoded each to the common embedding
- normalized each embedding
- took the average (across the 7 prompts ?) embedding
    - and used it as the embedding for the CIFAR class

```
texts = [template.format(classname) for template in templates] #format with cla
ss
            texts = clip.tokenize(texts).cuda() #tokenize
            class_embeddings = model.encode_text(texts) #embed with text encode
r
            class_embeddings /= class_embeddings.norm(dim=-1, keepdim=True)
            class_embedding = class_embeddings.mean(dim=0)
            class_embedding /= class_embedding.norm()
            zeroshot_weights.append(class_embedding)
```

Eventually: the image embedding has its cosine similarity compared with the average embedding for each label

```
# predict
image_features = model.encode_image(images)
image_features /= image_features.norm(dim=-1, keepdim=True)
logits = 100. * image_features @ zeroshot_weights
```

Observation from notebook on why templates help

> *Speculating, we think it's interesting to see different scales (large and small), a difficult view (a bad photo), and "abstract" versions (origami, video game, art), were all selected for, but we haven't studied this in any detail. This subset performs a bit better than the full 80 ensemble reported in the paper, especially for the smaller models.*

Adding textual hints (via prompt engineering: "a bad photo") seemed to help

- perhaps implicitly creating an attribute of an image
- that creates a relationship with other Images having a similar attribute

Experiments

The authors report many experiments using CLIP, in an effort to discover

- its strengths, weaknesses
- how it works

One experiment compares

- Zero shot learning of a Target task, using CLIP
- Transfer learning
    - creating a Target task specific head on top of an existing Source task Image Classifier (e.g., ResNet)

They call the Transfer Learning method *linear probing*.

Zero shot CLIP outperforms Linear Probing with ResNet on many classification tasks

- does better on Target tasks in which Target task training set has few examples per class
    - i.e., too few examples per class to adequately train the new Classification head

Bias

Section 7 of the paper is an effort to probe implicit biases that we have learned are present in seemingly unbiased text (e.g., Wikipedia)

For example, there are datasets used to probe for biases about race

- add "egregious" categories: animals ("ape", "orangutan"), criminal ("thief") to true Text label
- Blacks misclassified as animals more often than Whites
- Young people misclassified more often as thief
    - but adding a "child" class reduces this misclassification

Understanding that a model learns unintended bias

- through biased natural language
- through photos in very limited contexts

is something that is becoming more prevalent in describing and evaluating models.

Similar approach from Keras site

There is a similar example (https://keras.io/examples/nlp/nl_image_search/) on the Keras website.

It's instructive to compare the two approaches.

Joint Embedding

CLIP: via a shared learned embedding matrices $\backslash \mathbf{W}_i$ (image) and $\backslash \mathbf{W}_t$ (text).

Keras: via two separate multi-layer "embedding" networks of blocks consisting of `Dense` layers to transform the dimension.

```
def project_embeddings(
    embeddings, num_projection_layers, projection_dims, dropout_rate
):
    projected_embeddings = layers.Dense(units=projection_dims)(embeddings)
    for _ in range(num_projection_layers):
        x = tf.nn.gelu(projected_embeddings)
        x = layers.Dense(projection_dims)(x)
        x = layers.Dropout(dropout_rate)(x)
        x = layers.Add()([projected_embeddings, x])
        projected_embeddings = layers.LayerNormalization()(x)
    return projected_embeddings
```

Note that the statement

```
x = layers.Add()([projected_embeddings, x])
```

is implementing a skip connection.

- `projected_embeddings` that is input to the body of the loop

Loss function

The "logits" are the dot-product of the Text and Image embeddings.

CLIP: Compare the logits to the text target labels and (separately) to the image target labels.

- recall: the target "labels" are just indices since the diagonal element is the correct "target"

  loss_i = cross_entropy_loss(logits, labels, axis=0) loss_t = cross_entropy_loss(logits, labels, axis=1)

  loss = (loss_i + loss_t)/2

Keras

This approach is a little more sophisticated.

It compares the similarity (using dot product) of

- pairs of labels
- pairs of images

It uses the average similarity as the "correct" label

- the best target (highest average) is the correct one: I_i, T_i
- but mis-classifying a text/image pair is mitigated if the incorrect class
    - have similar texts
    - have similar images

CLiP demo (HuggingFace)

You can use the Inference API to play with CLiP on HuggingFace (https://huggingface.co/openai/clip-vit-large-patch14)

In [2]:
```
print("Done")
```

```
Done
```