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

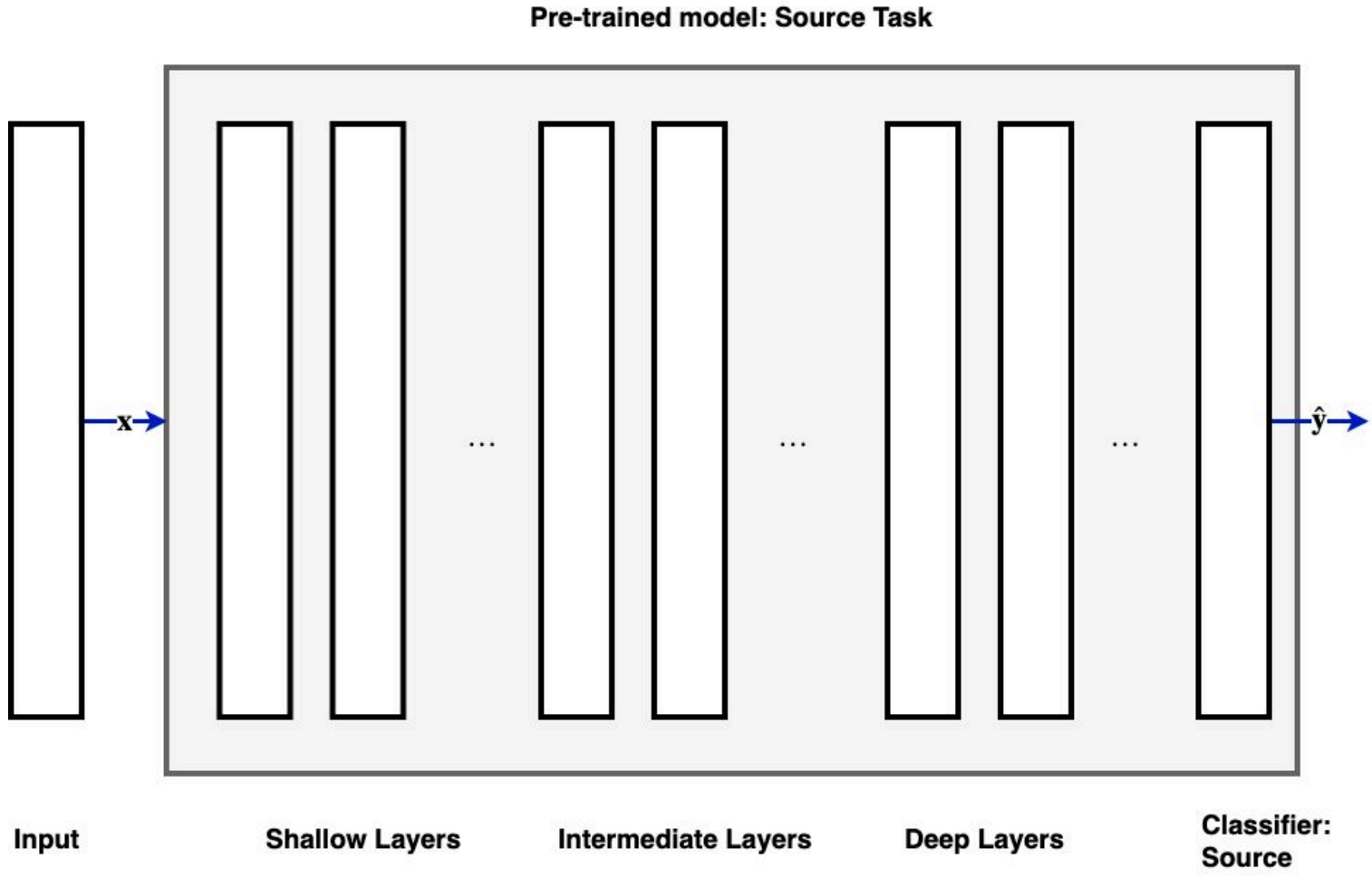
References

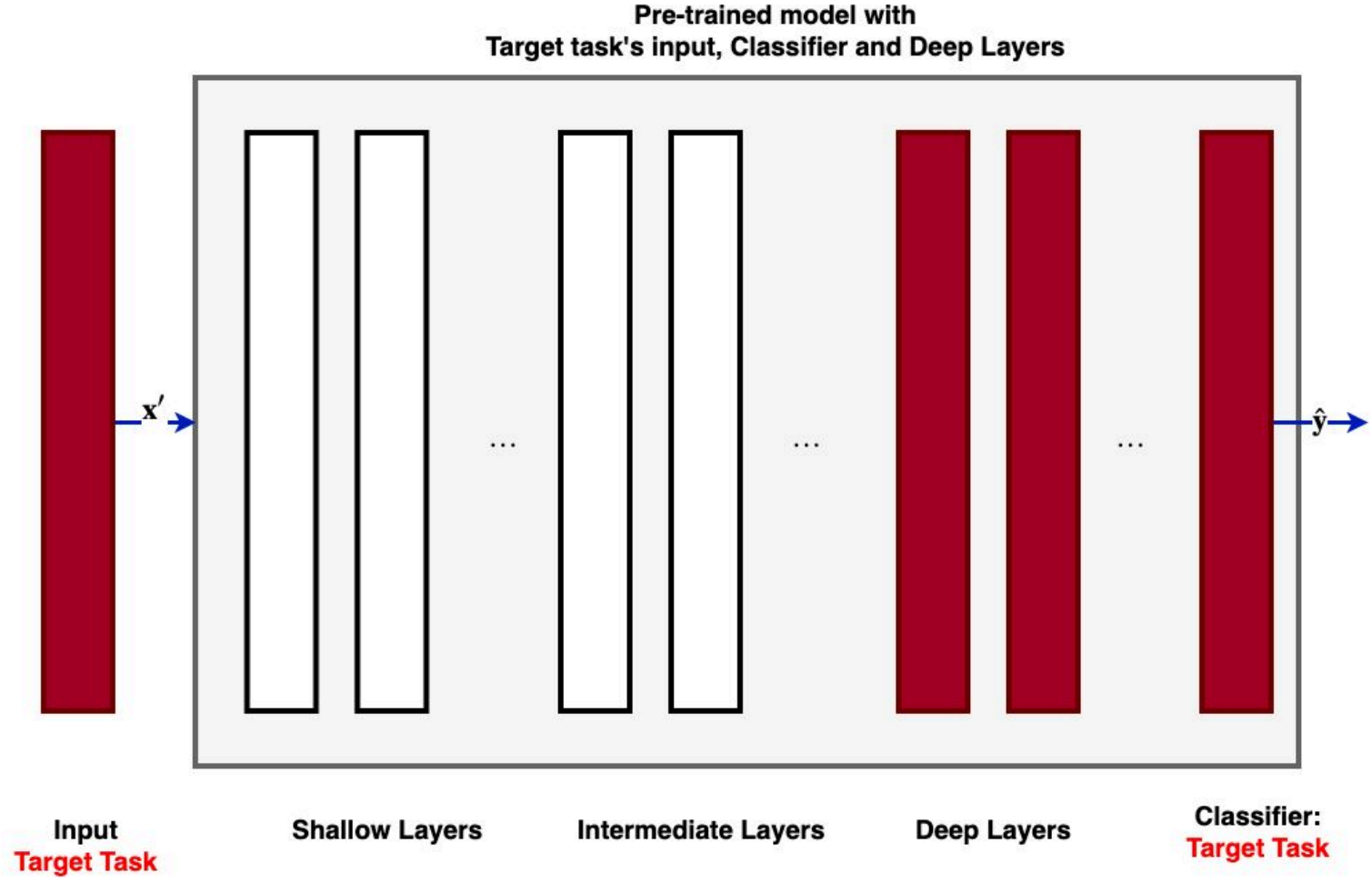
- CLIP paper (https://cdn.openai.com/papers/Learning_Transferable_Visual_Models_From_Natur
- LiT paper (https://arxiv.org/pdf/2111.07991.pdf)

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 do not generalize to successfully classify datasets distinct from the training set.

- Transfer Learning would be necessary to train a new Classifier Head (to accommodate the target set's labels)
- 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
- Training datasets are "small-ish": depends on human-intensive hand-labeling of images


Pre-trained model: Source Task
x
...
...
...
ŷ
Input
Shallow Layers
Intermediate Layers
Deep Layers
Classifier: Source


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

CLIP (Contrastive Image Language-Image Pre-training) is a model

- That creates representations of Image and Text *in a joint space*
- Leverages recent advances in Natural Language Processing to enable better Computer Vision models
- Facilitates **zero shot** learning of new Image Classification datasets
- Uses naturally occurring (and abundant) source of Training data

A key objective is use Natural Language supervision to learn about images.

Image embedding space

The prefix of an Image Classification model creates representations of Images that make it possible for a Classifier Head to predict the correct label.

- A chosen Deep layer of an Image Classification model creates *image embeddings*
- Transfer Learning: graft a new Classifier Head to learn the class labels of a new dataset

But Transfer Learning typically treats the labels as OHE vectors

- mutually orthogonal
- no relationship between semantically related class labels

Thus, the representation of Images potentially encodes "common" vision concepts

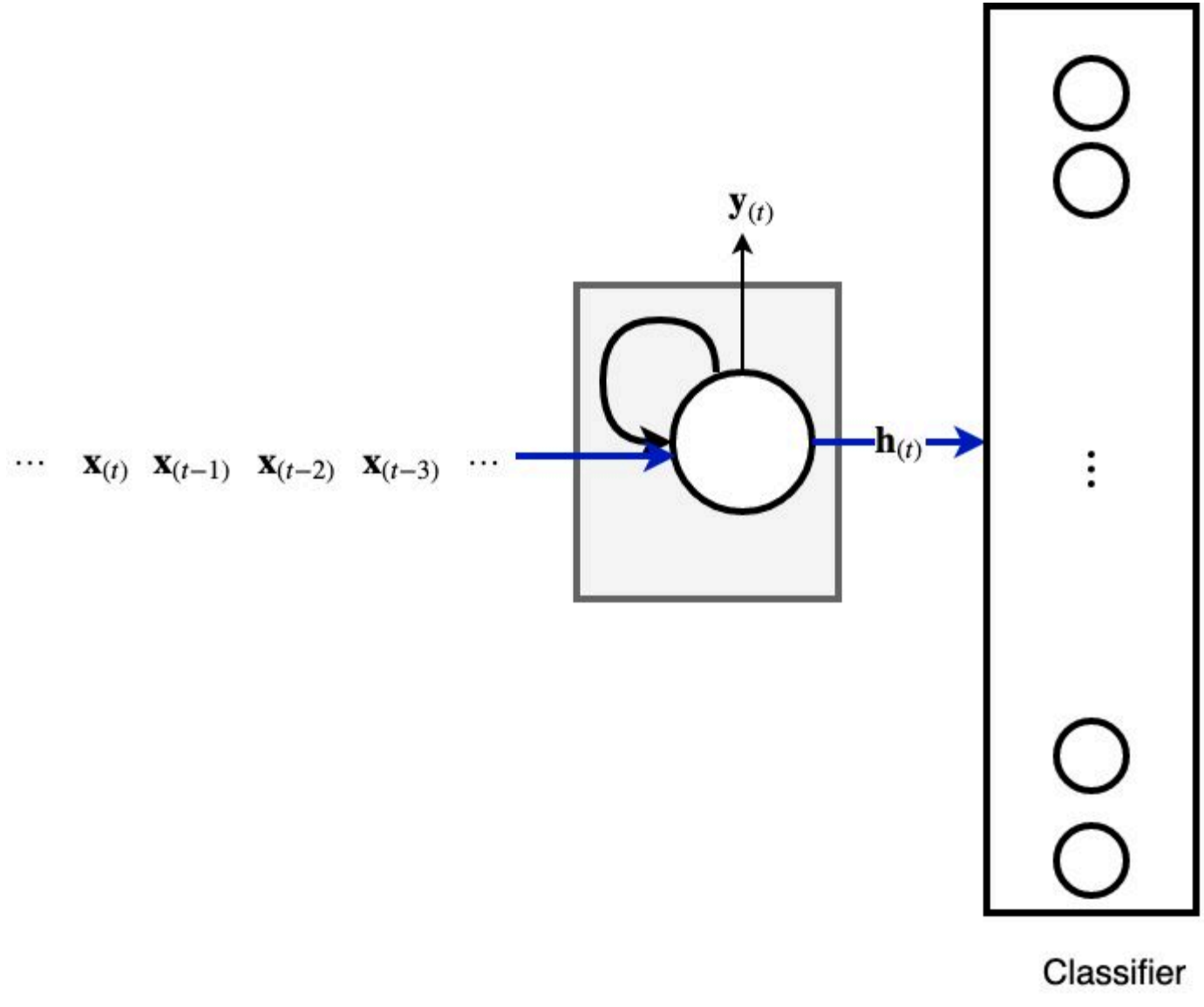
- But no such encoding of the Text

Text embedding space

Similarly: "Language Models" (e.g., GPT) 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 $\mathbf{h}_{(t)}$ is a fixed length "summary" of $\mathbf{x}_{(1)}, \ldots \mathbf{x}_{(t)}$



Creating a common embedding space

By contrast: 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

To be precise:

- Image I is encoded by an Image Encoder into I_f in "image embedding space"

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

  - I_f is projected into I_e in "common embedding space" via matrix W_i

$$I_e = I_f \cdot W_i$$

- Text T is encoded by a Text Encoder into T_f in "text embedding space"

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

  - T_f is projected into T_e in "common embedding space" via matrix W_t

$$T_e = T_f \cdot W_t$$

A primary use of the common embedding space

- Generative models that mix text and images

But we remark in passing that the paper had a different objective

- better Image Classification

Since I_e and T_e are both in "common embedding space"

- their similarity may be measured via the dot product

$$I_e \cdot T_e$$

This creates the potential for a different (and perhaps better) kind of Image Classifier

- match an Image
- to textual descriptions of the label

```
An image of a {label}
```

- rather than a numeric encoding of the label

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

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
$\mathcal{L}_i$	1	Loss across images: reduce $(\text{logits} \cdot \text{labels})$ across text dimension
		For each image: compare image's logits to labels
$\mathcal{L}_t$	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.

To be precise, CLIP is trained (enforced by the Loss function) to create a similarity metric such that

- the similarity between
    - the embedding of a Training image (in the joint embedding space)
    - and the embedding of the Text label of the training image (in the joint embedding space)
    - is *maximal* across the embeddings of all text labels in the Training set

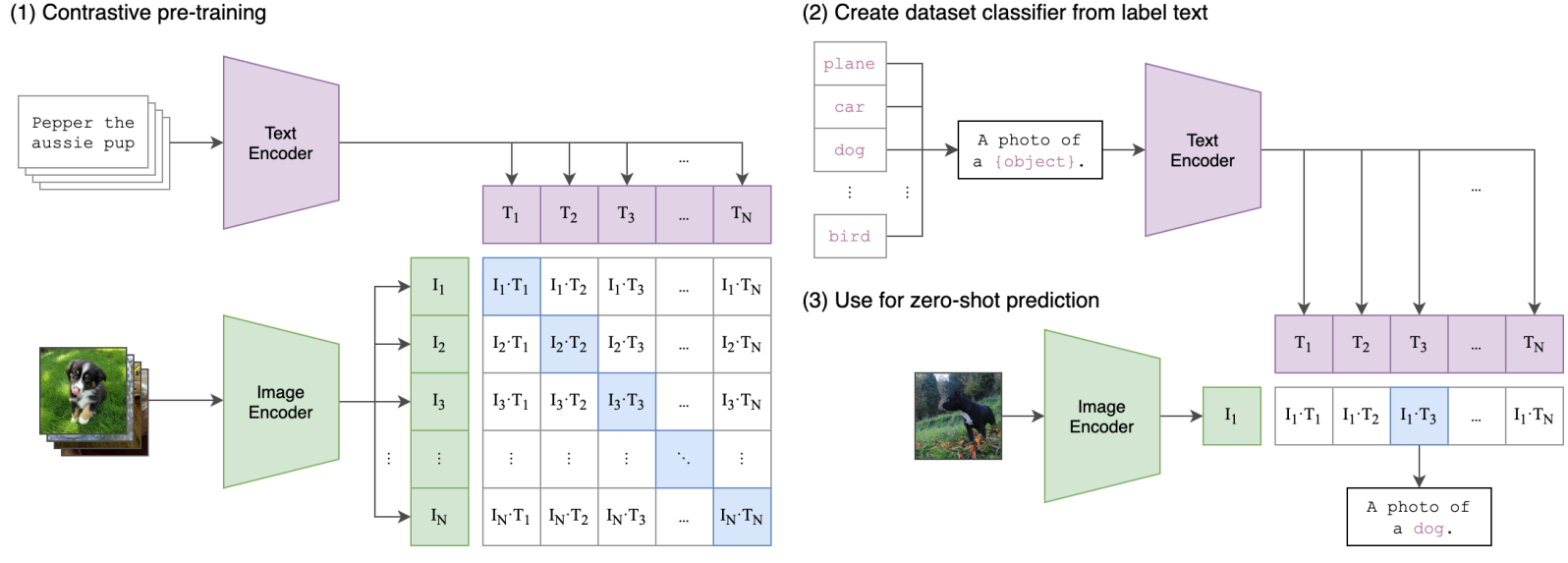
The method of *Contrastive Learning*

- 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



Pre-training details

Choosing Encoders for Image and Text

There are many existing "encoders" for both Image and Text.

- Image
    - A deep layer of any Image Classifier
- Text
    - Simplest: Continuous Bag of Words (CBOW)
    - Final latent state of an RNN (Encoder)
    - Transformer
        - use latent representation of the special <CLS> token typically prepended to the sequence

There is no need to re-invent one just for CLIP.

The authors report experiments with different Encoder choices.

Creating a common embedding space

However: there are some issues

- Technical
  - length of Image embedding is d_i
  - length of Text embedding is d_t
  - Not necessarily the case that $d_i = d_t$
- Semantic
  - The two embedding spaces were trained independently
  - Can't assume embedding of an Image in its own space is related to embedding of Text in its own space

The solution to both these problems is to project (via the dot product) each embedding into a shared space where embeddings are of length d_e.

- Matrix W_i of dimension $\left(d_i \times d_e\right)$ is a *learned projection* of image embedding to common embedding
- Matrix W_t of dimension $\left(d_t \times d_e\right)$ is a *learned projection* of text embedding to common embedding

Once projected into the common embedding space

- The similarity matrix (each element a dot product of a single Image embedding and single Text embedding) can be computed

The key: projection matrices W_i, W_t are *learned* as part of training

- not pre-specified
- they adapt to the objective (as defined by minimization of Loss)

Pre-training Loss function

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

Our objective is to ensure that the similarity (dot product of vectors within common embedding space) is such that

- $I_i \cdot T_i$ is as large as possible
- $I_i \cdot T_j$ is as small as possible for $i \neq j$

That is: the similarity (a scalar value) is maximized for the correct Text of the Image.

We can state this mathematically.

First compute the $(N \times N)$ similarity matrix S

- Normalize each embedding vector so the elements are positive and sum to 1
    - L2 normalization
- Compute the similarity of Image I_i with each Text T_j
    - as the dot product $I_i \cdot T_j$ of the normalized vectors
    - resulting in a vector of length N for each Image I_i
- Convert the similarity vector (row of length N) for Image I_i into a probability distribution via a Softmax

The last 2 steps implement a multinomial Logistic Regression Classifier

- where T_j are the weights ("pattern matched against and Image) for logit j of the Classifier output

The import of having Image and Text embeddings in a joint space is that we can perform this pattern matching.

In essence: the Image and Text are equivalent representations of the same concept.

We create an *Image Loss* for example i : $\mathcal{L}_i^{(\mathbf{i})}$

- subscript i denotes *image* (not an index)

$S^{(\mathbf{i})}$, row i of the similarity matrix

- Can be interpreted as the *probability distribution over the N possible* **labels** for Image I_i
- column $j = i$ is the correct label

Use Binary Cross Entropy for the Image loss

- want the probability of the correct label (column i) to be 1:

$$S_i^{(\mathbf{i})} = 1$$

- want the probability of any other label $j \neq i$ to be 0:

$$S_j^{(\mathbf{i})} = 0,\ j \neq i$$

We also create a *Text Loss* for example i : $\mathcal{L}_t^{(\mathbf{i})}$

- subscript t denotes text

Rather than using the *rows* of similarity matrix S, we use the *columns*

S_j, column j of the similarity matrix

- Can be interpreted as the *probability distribution over the N possible* **images** for Text T_j
- Row $i = j$ is the correct image

Use Binary Cross Entropy for the Text Loss

- want the probability of the correct image (row i) to be 1:
$$S_j^{(j)} = 1$$
- want the probability of any other image $i \neq j$ to be 0
$$S_j^{(\mathbf{i})} = 0,\ i \neq j$$

n.b., above we write the loss for example j, per our custom of using j to index columns

This is called a *Contrastive* objective

- maximizing the *contrast* between correct (value 1) and incorrect (value 0)

Cross Entropy (loss used in the multinomial Logistic Regression Classifier) *is* a contrastive loss.

- It is minimized when
    - the probability assigned to the correct logit is highest (i.e, 1)
    - the probabilities assigned to incorrect logits is lowest (i.e., 0)

But how do we create the *many possible* negative examples to contrast with the positive one ?

Rather than explicitly creating many negative examples

- we assume that all examples in *a single batch*
- other than the one with the correct label
- are negatives: completely incorrect

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

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

$$\mathcal{L}^{(\mathbf{i})} = \mathcal{L}_i^{(\mathbf{i})} + \mathcal{L}_t^{(\mathbf{i})}$$

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

Technical notes

In Mini-batch Gradient Descent, examples are processed in batches

- batch size denoted as n in this paper
- rather than over all N training examples
- to be precise: in the explanation above, replace N with n

The batches are arranged so that there is a single example for each of the n labels

- arranged so that example i of the batch has the i^{th} label
- doing so allows the code to specify the target vector for the mini-batch as $[0, n-1]$

The in-batch negatives trick is implemented by using matrix multiplication

- which achieves the dot product of each image encoding with each text encoding

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

Image Classification models need (Image, Text) pairs as training examples.

Traditionally, these were created manually (sometimes crowd-sourced) at considerable effort

- hence, datasets were limited in size

This is because labels were

- pre-defined
- from a small set

This problem is similar to what researchers in NLP encountered.

The recent rapid advances in NLP were achieved, in part, by cleverly using the entire Web as a source of text

- Raw text is unlabeled
- But can turn a sequence into a "predict the next element of sequence" Semi-Supervised example
- Hence: lots of training examples

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 $j^* = \underset{j}{\operatorname{argmax}}\, I_1 \cdot T_j$

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 $\mathbf{W}_i$ (image) and $\mathbf{W}_t$ (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.

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)](https://huggingface.co/openai/clip-vit-large-patch14)

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

```
Done
```