ATTENTIONAL PARSING NETWORKS

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Marcus Karr

November 2020

COMMITTEE MEMBERSHIP

TITLE:	Attentional Parsing Networks
AUTHOR:	Marcus Karr
ADVISOR:	Franz Kurfess, Ph.D. Professor Computer Science Department
DATE SUBMITTED:	November 2020
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D. Associate Professor Computer Science Department
COMMITTEE MEMBER:	Maria Pantoja, Ph.D. Associate Professor Computer Science Department
COMMITTEE MEMBER:	Jonathan Ventura, Ph.D. Associate Professor Computer Science Department

Abstract

Attentional Parsing Networks

Marcus Karr

Convolutional neural networks (CNNs) have dominated the computer vision field since the early 2010s, when deep learning largely replaced previous approaches like hand-crafted feature engineering and hierarchical image parsing. Meanwhile transformer architectures have attained preeminence in natural language processing, and have even begun to supplant CNNs as the state of the art for some computer vision tasks.

This study proposes a novel transformer-based architecture, the *attentional parsing network*, that reconciles the deep learning and hierarchical image parsing approaches to computer vision. We recast unsupervised image representation as a sequence-to-sequence translation problem where image patches are mapped to successive layers of latent variables; and we enforce symmetry and sparsity constraints to encourage these mappings take the form of a parse tree.

We measure the quality of learned representations by passing them to a classifier and find high accuracy ($> 90\%$) for even small models. We also demonstrate controllable image generation: first by "back translating" from latent variables to pixels, and then by selecting subsets of those variables with attention masks. Finally we discuss our design choices and compare them with alternatives, suggesting best practices and possible areas of improvement.

Acknowledgments

This paper would not be possible without my wife, Janice, and her unwavering support through setbacks and obstacles of every sort.

I would like to thank my advisor, Professor Franz Kurfess, for his enthusiasm and flexibility as this thesis has taken on new forms.

Professor Foaad Khosmood has also been instrumental to this work, although he may not know it. His NLP instruction introduced me to sequence translation, and later transformers, which I became intimately familiar with during class projects.

Finally I would like to thank Ben Trevett, whom I have never met, but whose custom transformer implementation was an inspiration and a help, and whose email correspondence encouraged me to explore transformers for vision.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of artificial intelligence is to create machines that think and perceive as people do. Computer vision, in particular, has seen tremendous progress in the past decade: we now have neural network models that classify images, recognize objects, and perform other visual tasks with high speed and accuracy.

However, despite their excellent benchmark performance and proven utility for real world applications, it is clear that the best vision models today do not process visual information as people do. For example, they are vulnerable to failure modes, like one pixel adversarial attacks [1], that do not affect human beings. Furthermore, it is difficult to diagnose and debug problems since neural network "decision making" is so opaque.

Ideally we would like to have vision models that are fast, accurate, and interpretable. What would it mean for a vision model to be interpretable? Perhaps its logic should resemble the way people decompose scenes into objects and objects into parts. Our goal, therefore, is the modular decomposition – or *parsing* – of images.

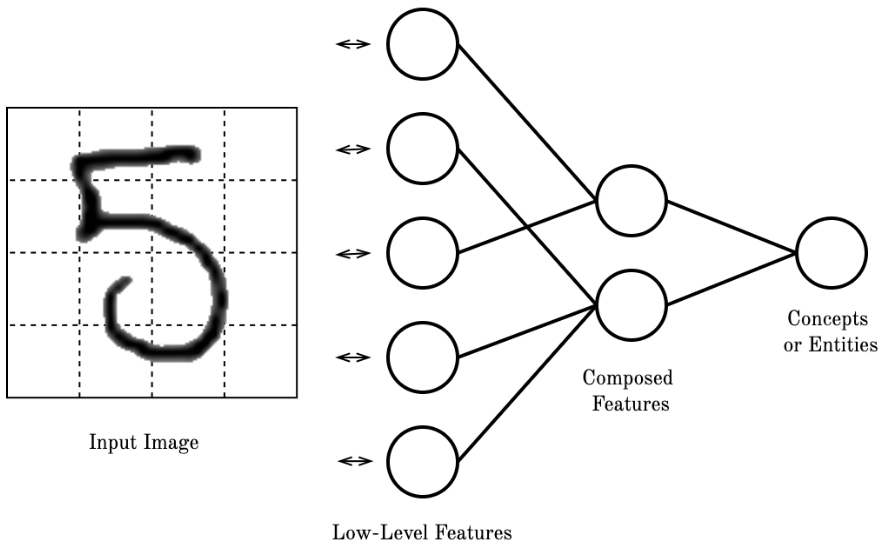

Figure 1.1: Attentional parsing network (overview)

To this end we present a novel transformer-based architecture, the attentional parsing network (APN), that takes inspiration from the hierarchical image parsing tradition. Unlike most image parsing models, ours is trainable end-to-end with off-the-shelf components widely available in popular deep learning software libraries.

The attentional parsing network maps sequences of image patches to sets of latent variables. After inputs have been passed through the network, the output of the final layer is passed backward to reconstruct the image. The pattern of activation representing the image is shaped through loss functions to be symmetrical with respect to forward and backward passes, and to be sparse, so that it takes on a tree-like structure.

To investigate whether and to what extent the model decomposes images into hierarchical and interpretable representations, we generate images from latent variables and visualize network activation patterns.

We measure the representational power of these patterns by using them to train a classifier, and find that even shallow networks can attain upwards of 90% accuracy. This suggests that even in unsupervised settings, APNs learn representations that correspond to human-assigned classes.

Chapter 2

Background

Attentional parsing networks reconcile two computer vision traditions — deep learning and hierarchical image parsing — with a novel application of the transformer. To clarify what this means, we review these concepts and give a brief history of their development.

2.1 Feature engineering

Until the early 2010s, most tasks in computer vision (CV), natural language processing (NLP), and other artificial intelligence (AI) or machine learning (ML) fields relied on feature engineering. Feature engineering refers broadly to the process of using domain knowledge to detect *features*, or salient subsets of the data, and developing algorithms to extract them.

Feature engineering typically borrows from signal processing and statistics. An example of a feature detector is the Sobel filter, which detects edges in images; and notable examples of feature extraction algorithms include the Hough transform and Scale-Invariant Feature Transform (SIFT) [2].

A workflow associated with feature engineering is as follows: extract features then pass them to a subsystem like a neural network, support vector machine, or other ML model for classification. Under this paradigm it is the engineer's job to identify the relevant features in the data and devise a way to extract them; and it is the model's job to map those features to desired outputs.

2.2 Deep learning

Deep learning refers to the use of multilayer neural networks to automatically learn features from data. Since this process obviates the need for feature engineering, it is sometimes called "end-to-end" deep learning: data goes in, results come out, and comparatively little human intervention is required.

Deep learning rose to prominence in the early 2010s due to a confluence of factors including the availability of large new datasets, GPU support, and new software libraries. The replacement of sigmoid activations with rectified linear units (ReLUs), new regularization techniques like dropout and batch normalization, and improved optimization algorithms are a few of the advances that enabled multilayer neural networks to train far more quickly and stably than previously possible.

Above all deep learning has exploded in popularity because it works so well. No matter the AI/ML field, most state of the art models today are multilayer neural networks. And thanks to a general-purpose training procedure (backpropagation), it is easy to get excellent results on many datasets without domain expertise or algorithmic ingenuity.

2.3 Convolutional neural networks

CNNs are the mostly widely used neural network architecture in computer vision [3]. They are named after the convolution operation, which in the case of images is a sliding window filter that acts as a feature detector. When multiple convolutional layers are stacked together, low-level features aggregate into higher-level features, and this process continues until at the final layer there is a single feature vector to represent the image.

CNNs are well suited to images because convolution is a local operation and images exhibit a great deal of spatial locality. Furthermore, convolution is implemented in the architecture as shared weights, reducing network parameter count. This makes CNNs much more efficient than fully connected networks. Finally, convolution gives CNNs the property of translation invariance, meaning features are detected independently of their position in the image.

2.4 Hierarchical image parsing

The aim of hierarchical parsing models is to represent objects as wholes made of parts, which themselves may be made of subparts [4]. Parsing models typically exhibit a tree-like structure where low-level features correspond to leaves, mid-level representations correspond to subtrees, and high-level concepts correspond to root nodes. If a particular feature is detected, then its corresponding node will be activated and the nodes of undetected features will remain inert. Thus only the active nodes constitute a parse and inactive nodes will be disregarded. As described in [5], "a parse tree is carved out of a fixed [structure] like a sculpture is carved from a rock."

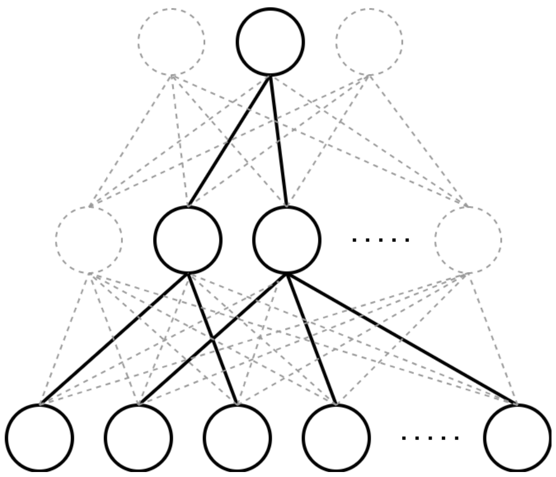

Figure 2.1: Image parse tree

In the figure above, active elements are solid and inactive elements are dashed. We see that inputs are represented as patterns of activation in the model as a whole, rather than being condensed into a single feature vector as in CNNs.

The hierarchical structure of learned representations imbues parsing models with the advantage that "long range spatial relationships can be represented by local potentials" [6]. Moreover their modularity makes interpretation straightforward — it is easy to see precisely how low-level features compose into more general representations.

Despite these advantages, image parsing models have fallen out of favor since the advent of deep learning and CNNs. With one notable exception to be covered in the next chapter, image parsing models rely heavily on feature engineering, their inference is slow, and training is multifaceted and complex.

2.5 Transformers

The architecture presented in this paper is based on transformers. Although we apply transformers to images, they were originally designed for sequence translation [7]. To make sense of transformers we will discuss their operation and historical context. We begin with a preliminary discussion on tokens.

2.5.1 Tokens and embeddings

Tokens are members of finite sets. In NLP contexts they can refer to words, word pieces, or phrases drawn from an enumerated vocabulary where each token has a unique integer ID. Tokens can then be represented as vectors whose length is the size of the vocabulary and whose values are all `0`, except for a single `1` at the index equal to the token ID. For example, if our vocabulary is

```
{
  ‘a’: 0,
  ‘am’: 1,
  ‘i’: 2,
  ‘student’: 3
}
```

then the `‘i’` token would be encoded as `[0, 0, 1, 0]`. Such representations, called *one-hot vectors*, allow us to format sequences as acceptable neural network inputs.

In the case of large vocabularies one-hot vectors are extremely sparse and their length may exceed the capacity of the rest of the network. The BERT language model, for instance, has vocabulary size 30522 and network dimension 768 [8]. To resolve mismatches and convert one-hot vectors to more efficient, information dense representations, we pass them through *embedding layers*, or weight matrices of shape `(vocab_size, network_dimension)`. Ultimately each token will be mapped to a unique vector. Therefore in later discussion "token" and "token vector" may be used interchangeably.

Finally we note that embedding layers can encode additional information about a token, such as its source vocabulary or position in a sequence. Usually these "metadata" vectors are added to token vectors, but it is also possible to concatenate them.

2.5.2 Sequence translation

Sequence translation is simply the mapping of one token sequence to another. Often sequence translation refers to the more specific case where sequences are drawn from two disjoint vocabularies, like English and French.

Before transformers, the long short-term memory (LSTM) variant of recursive neural networks led the field of sequence translation [9]. Under this approach, an encoder LSTM accepts tokens from a sequence and continually updates its internal state. After the final token of the sequence has been processed, the encoder's internal state represents, in vector form, the sequence as a whole. This representation is then used to initialize the internal state of a decoder LSTM, which generates a new token sequence [10]. See Figure 2.2.

The problem with this method is that the encoder's representation tends to be biased toward recent tokens and can fail to "remember" the early parts of long sequences. Therefore researchers began to incorporate not just the encoder's final internal state, but all its internal states, for better representation of long-term dependencies [11]. See Figure 2.3. However, this development prompted a new question: how to weight and aggregate a sequence of encoder states such that

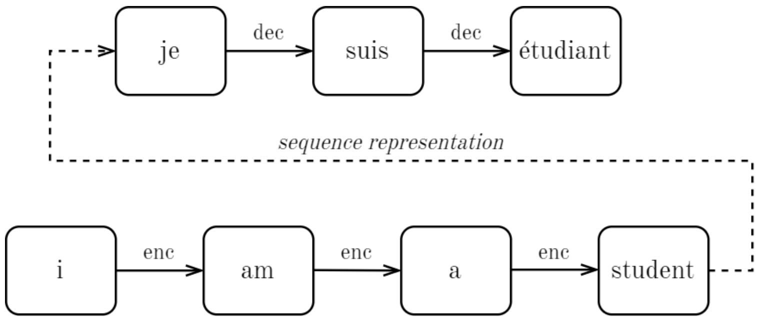

Figure 2.2: Translation without attention

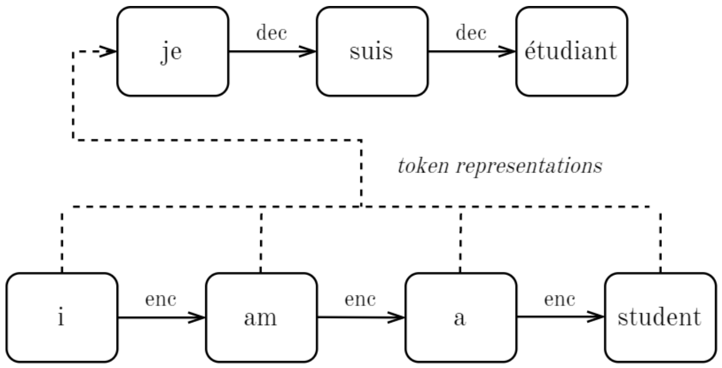

Figure 2.3: Translation with attention

the decoder can make best use of them?

One early solution does

> not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. [11]

The process of "adaptively choosing a subset of vectors", or paying attention to some vectors but not others, came to be known as *attention*. Over several years researchers proposed various mechanisms for aligning input vectors with output vectors, as well as integrating these mechanisms with LSTMs [12], but by 2017 the transformer architecture emerged as a clear standout in the field.

2.5.3 Attention in transformers

The landmark paper, *Attention Is All You Need* [7], showed that LSTMs are unnecessary for sequence translation. Its key innovation is the query, key, value formulation for token alignment. First, each token vector is passed through three separate matrices to obtain query, key, and value vectors. Then the softmax function is applied to the dot-product of queries and keys to produce a normalized attention matrix. Finally we multiply this attention matrix by the value vectors and pass them through a final linear layer.

The formula is

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

where Q, K, V are queries, keys, and values, respectively, and d_k is the network dimension. (Scaling by its inverse square root is not strictly necessary but improves stability.)

Intuitively, the attention matrix assigns weights to token pairs based on the similarity of their query and key representations. When the attention matrix is multiplied by value vectors, we end up with a transformed sequence that might be regarded as a kind of weighted average.

This formulation is flexible. The queries, keys, and values can all be taken from the same sequence, which gives us self-attention. Or the queries can be taken from a target sequence while the keys and values come from a source sequence, which gives us cross-attention (Figure 2.4). In either case we create a mapping between two sequences and then transform one of the sequences according to this mapping.

There are many other features relevant to the effective functioning of trans-

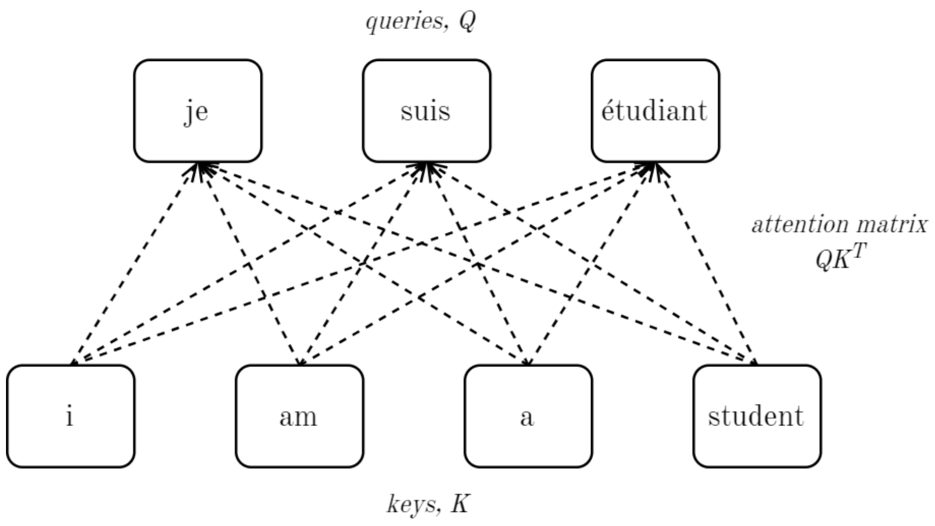

Figure 2.4: Transformer cross-attention

formers, including multiple attention heads, expansion layers, layer normalization, and residual connections. We gloss over these in order to highlight the salient feature of transformers and the one most relevant to our work — namely, that attention assigns weights between sets of vectors. However, there are still a couple secondary features worth mentioning.

2.5.4 Position embeddings

Previously we stated that attention constitutes a mapping between sets and sequences. To clarify, transformer attention works on sets by default, but can be made to work on sequences by encoding positional information. To accomplish this, positions are passed through an embedding layer and added to corresponding token vectors. While positional embeddings are typical in most transformer applications (which involve sequences), much of our architecture (which involves sets) will forgo them.

2.5.5 Attention masks

Attention masks are optional transformer inputs that cause some tokens to be zeroed out of the attention matrix. In effect, they instruct the system not to pay attention to the specified tokens. Masking is useful for transformer applications like generation and in-filling. We will be using masks for controllable generation.

Chapter 3

Related work

We are unaware of any prior literature on hierarchical image parsing with transformers, and believe our model is unique. However, there are a number of similarities with other work, both in implementation details and architecture, that we review in the sections below.

3.1 Transformers for vision

While transformers found quick acceptance and wide adoption in NLP [13], they have been much less prominent in CV. This is because transformers were designed specifically for natural language sequences, where they led to swift advancement in a number of tasks that had been progressing slowly. Also by 2017 CNNs had already contributed to great strides in CV and were becoming a mature technology. The ResNet architecture [14], for instance, debuted in 2015 and remains a preferred backbone for object recognition and image segmentation models to this day [15]. It is not surprising that transformers have only slowly made their way into CV — they have not really been needed.

Transformers also face a serious technical drawback: they are computationally expensive. Attention matrices are $O(N^2)$, which places sharp limits on input size. This is a major issue for NLP tasks, where even GPT-3, one of the largest ML models to date, does not accept sequences longer than 2048 tokens [16]. Consequently a great deal of research has gone into reducing transformer complexity, for example by approximating quadratic attention with linear substitutes [17], incorporating recurrence [18], or replacing dot-product attention with locality-sensitive hashing [19]. While these modifications show promise, they are not as intuitive or easy to implement as "vanilla" transformers, so for simplicity we have decided to stick with the original.

If transformer complexity is problematic for NLP, then it is downright prohibitive for CV, since pixel-wise attention is $O(h^2w^2)$ in the height and width of the image. One way to deal with the high cost is to apply pixel-wise attention to small windows and use these as drop-in replacements for convolutional blocks in a ResNet-like architecture [20]. Another approach is to attend to convolved features rather than pixels [21]. Yet another clever workaround is to indirectly attend to every pixel by attending first to rows and then to columns [22].

Our approach is to break images into contiguous patches and feed them to the transformer as if each is a word or token. An image, therefore, is treated as a sequence of patches. This is the same basic method used in [23], which outperformed ResNet-152 while requiring a fraction of the training resources. [23] differs from ours in that it breaks images into square tiles, while we also experiment with row and column patches; and it is limited to classification, while we further integrate the image-as-sequence method into a hierarchical model.

3.2 Recursive cortical networks

Recursive cortical networks (RCNs) represent the pinnacle of hierarchical image parsing outside the deep learning framework [24]. Using hand-engineered filters, RCNs extract foreground/background, surface and contour features from images, which are then passed to a tree-like probabilistic graphical model (PGM). (It is not exactly a tree because of edges between nodes of the same layer.) The RCN settles on the parse with maximum a posteriori likelihood after multiple rounds of message-passing. Since probabilities propagate both up and down the tree, inference is not possible with a single pass and an iterative procedure is required.

RCNs are remarkable for their similarities to human vision, including top-down feedback and the "hallucination" of optical illusions. RCNs are also extremely sample efficient, requiring 50,000 times fewer training examples than neural networks trained on the same tasks [24].

Unfortunately RCNs share a major disadvantage with other hierarchical image parsing models, namely lack of speed and parallelizability. Nevertheless RCNs are a major inspiration for our goal of implementing image parsing in a differentiable, end-to-end fashion.

3.3 Capsule networks

Capsule networks are the primary, if not the only, deep learning approach to image parsing. They have undergone a series of revisions since their introduction in 2011 but all versions share a commitment to learned feature detectors and groups of neurons, called "capsules",

> that perform some quite complicated internal computations on their inputs and then encapsulate the results of these computations into a small vector of highly informative outputs. [25]

In standard neural networks, the elementary unit is the neuron and its activation is a scalar. In capsule networks, by contrast, the elementary unit is the capsule and its activation is a vector. The values in the activation vector represent explicit graphics parameters like position, scale, orientation, or lighting, as well as a special probability value that indicates whether the capsule is active (or, alternatively, whether the corresponding visual entity is present).

Beginning with [5] capsules were built into parse trees. Child nodes would "vote" for the parent nodes best able to accept their outputs in an iterative procedure called "dynamic routing", which, the authors argued, was "far more effective than the very primitive form of routing implemented by max-pooling" in CNNs. In order to keep the model differentiable, the iterative procedure was unrolled into three layers.

The latest capsule network, the stacked capsule autoencoder (SCAE) [26], boasts a sophisticated architecture with multiple components and two types of capsules: parts and objects. Part capsules take CNN feature vectors and output graphics parameters and presence probabilities, as before, but the routing between parts (child nodes) and objects (parent nodes) is handled by a transformer instead of unrolled iterations. Meanwhile object capsule activations contain not only presence probabilities but object-viewer perspective information as well.

The wealth of explicit geometric representations built into the SCAE give it robustness to affine transformations and align with the goal of "inverse computer graphics". The results are impressive, with a reported unsupervised classification accuracy of 98.7% on the MNIST dataset [26].

Our model is much simpler and more general. We avoid domain-specific representations (like image geometry) and our counterparts to capsules have uniform activations (not different kinds). However, we both use transformers for routing between layers and both our node activations are vectors. One last point of difference is the choice of feature detector: CNN (theirs) versus transformer (ours).

3.4 Summary

We use transformers for feature detection and routing (as in [23] and SCAEs, respectively) and take architectural inspiration from RCNs. Both SCAEs and RCNs route signal through tree-like structures, but SCAEs have only two layers (parts and objects) while RCNs have multiple layers of intermediate representation.

Model	Feature Detector	Routing Mechanism	Representations	Differentiable
RCN	Hand-crafted	PGM	Explicit	No
SCAE	CNN	Transformer	Explicit	Yes
APN	Transformer	Transformer	Implicit	Yes

Table 3.1: Comparison of image parsing models

We summarize other similarities and differences in the table above. The feature detector, again, is the “front end” that extracts relevant data from images. The routing mechanism determines activity within the hierarchical structure, “carving out” the parse tree. Representations are explicit if they stand for predefined parameters, and implicit if they are learned. Finally, “differentiable” is shorthand for whether the model can be trained with backpropagation and run on a GPU.

Chapter 4

System design

Attentional parsing networks represent images as patterns of activation in a multilayer structure. Layers are collections of vectors, or “tokens”, that we “translate” between with transformers. In an attempt to force interlayer connections to assume a tree-like pattern, we impose two constraints on the system: 1) interlayer attention matrices must be symmetrical with respect to forward and backward passes, and 2) interlayer attention must be sparse, so that different tokens account for different kinds of input.

Figure 4.1 presents a high level overview of the system. We first break the image into patches and pass them through an embedding layer that formats them as a sequence of tokens. Next we map this sequence to a set of trainable vectors (“learned queries”), which we then map to another set like it. The activations of the top layer are passed back down until they go through the embedding layer, which finally outputs a sequence of patches that reconstructs the image.

In this chapter we discuss the function, structure, and motivation of each stage in greater detail.

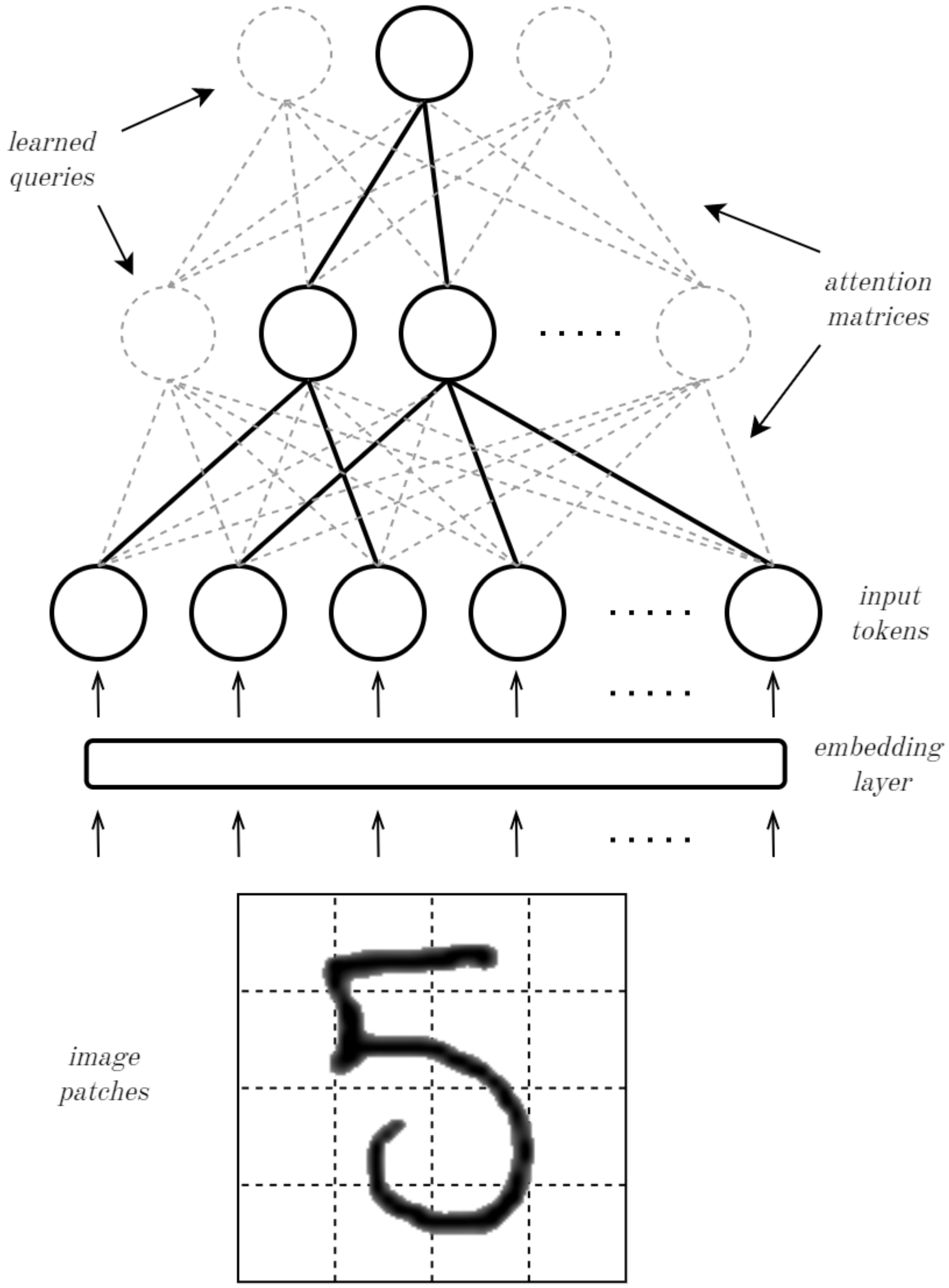

Figure 4.1: Attentional parsing network (detail)

4.1 Image patches

There are many ways to break images into contiguous pieces but perhaps the most straightforward is to use square tiles with no overlap. This means that for 28x28 MNIST images we can use 1x1, 2x2, 4x4, 7x7, or 14x14 squares with correspondingly fewer tiles per size. There is a tradeoff between the information per tile and the number of tiles, and we found after informal experimentation that 49 4x4 tiles works reasonably well. Of course, for larger images, such small tiles would be infeasible.

We also experimented with breaking images into rows and found that this works just as well and sometimes better. But since individual rows are not interpretable for visualization we decided not to pursue them further.

It is also possible to break images into overlapping patches, as is common with CNNs. This should lead to higher accuracy and reconstruction resolution at the expense of time and memory resources, but we leave these experiments to future work.

4.2 Embedding layer

In NLP contexts token embedding layers convert one-hot vectors into denser representations. In our case we are not dealing with one-hot vectors but 16-dim vectors (from 4x4 tiles). 16 dimensions is much less than the 512 or 768 dimensions most transformers use for natural language, so out of concern for sufficient representational power we "upsample" our 16-dim patches into higher-dimensional vectors. This has the added advantage of performing much the same function as convolutional filters [23]. We could simply use larger patches

We add position embeddings to input tokens before passing them to the transformer, as is customary in sequence translation. We experimented with row-column versus index positions and found that index positions work better and are simpler to implement.

4.3 Layer translation

4.3.1 Mechanics

Translation occurs between layers just as it occurs between sequences in the standard transformer: self-attention transforms both layers, then cross-attention is applied between the target and the source. Compare the depiction in Figure 4.2 (APN layers) with Figure 2.4 (language sequences) — they are structurally identical. The only difference is that Figure 4.2 shows the complete computation, where the normalized attention matrix multiplies the “value” vectors of the inputs, giving the layer activations.

4.3.2 Learned queries

Despite the structural similarities, there is a significant difference in what the vectors represent. When transformers translate a source sequence into a target sequence, the tokens in the target sequence belong to a vocabulary, like the set of French words. Only a subset of French is present in the target sequence. However, when we translate the input sequence of image patches into a target (namely the set of vectors that constitutes the next layer), the target is always the same — the entire vocabulary is present.

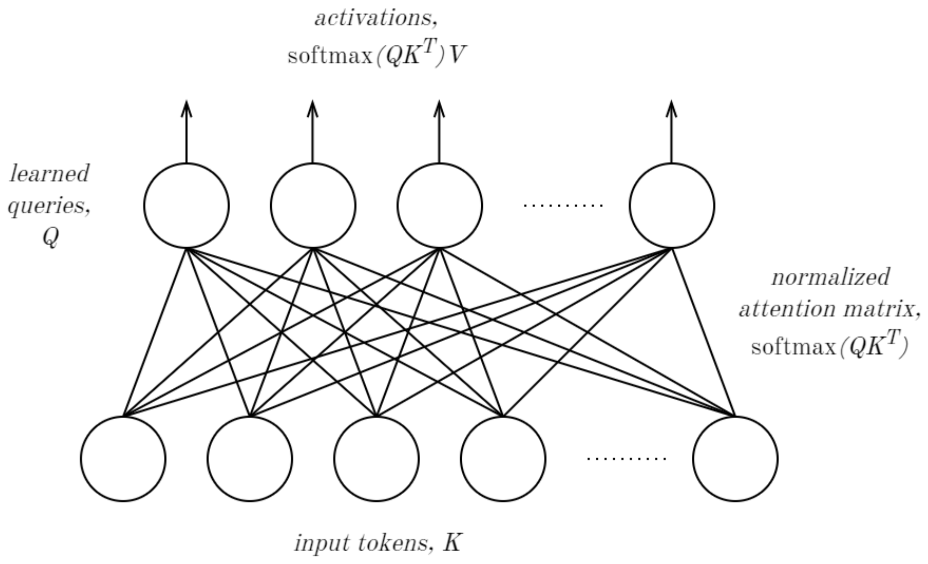

Figure 4.2: Cross-attention between layers

What differs is the cross-attention matrix between the input layer and the next layer: different tokens will attend more to some inputs than others, and this in turn affects activations propagating forward.

From another perspective these tokens are latent variables weighted dynamically according to input. Since transformer attention is based on dot-products, similar tokens produce higher attention scores, resulting in a sort of content-addressable routing.

These tokens are trainable parameters initialized randomly and learned with the rest of the model. We follow [21] in calling them "learned queries" after the analogous role queries play in sequence translation, but they can also be interpreted as latent variables.

4.4 “Carving out” a parse tree

The architecture as described so far can be used for arbitrary purposes: it is simply a kind of vector-based multilayer network. There are, as yet, no notions of hierarchy or anything resembling a parse tree. The following subsections explain our attempts to “carve out” a parse tree from the network.

4.4.1 Backward pass

We send the activations of the final layer backward through all layers, including the transpose of the embedding layer, in order to reconstruct the input. This differs from traditional autoencoders with separate encoder and decoder modules. We utilize a backward pass to build a single, unified activation pattern, which would not be possible with a separate decoding stage.

4.4.2 Symmetry constraint

We want the network activation pattern to be symmetrical across forward and backward passes, otherwise we would we have two parse trees for one image. Specifically we want the cross-attention matrix between layers on the forward pass to be equal to its transpose on the backward pass. We enforce this constraint with mean square error (MSE) loss.

4.4.3 Sparsity constraint

If left to its own devices the network will make use of all the bandwidth available to it, meaning that most inputs will be mapped to most learned queries (or latent variables). Instead we want different kinds of inputs to be mapped to different subsets of each layer, and we want to exaggerate these differences such that any given input will activate only a few nodes per layer.

Sparsity cuts a tree-like shape into the network activation pattern and forces certain nodes to be responsible for certain parts of the data. Therefore we apply an L1 loss to encourage the cross-attention matrices to contain mostly zeroes [27].

4.5 Generation

Generation is possible by translating from the final layer of learned queries back down through the network when no input image is present. Instead of a forward and backward pass, we simply conduct a backward pass. If we use all the tokens in each layer, the resulting image gives some indication of what the network has learned from all the data. But we can also use subsets of tokens to see what parts of the data, specifically, they are responsible for.

We apply attention masks to each layer to select the tokens used for generation. Ideally we would like to see compositional representations such that low-level features correspond to the bottom layer and high-level features correspond to the top layer. We illustrate the use of attention masks in Figure 4.3.

Note that during generation the backward pass begins with the top layer, while during image representation the backward pass begins with the top layer's activations. This means we can inspect how different tokens account for different parts of the data, but we cannot expect generated images to be comparable to reconstructed images.

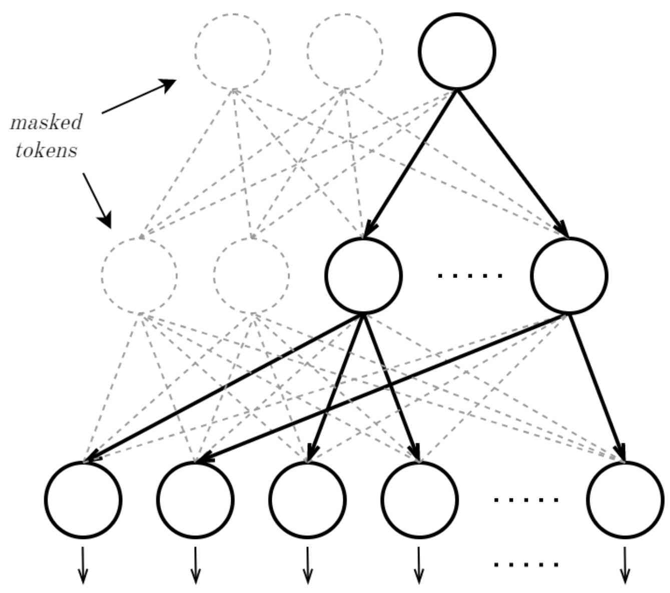

Figure 4.3: Masked generation

4.6 Training

Training is straightforward: we optimize network parameters with the sum of the MSE reconstruction loss, the symmetry loss, and the sparsity loss. We use the AdamW optimizer [28] with a `5e-4` learning rate and no learning rate scheduler.

4.7 Implementation

The model is implemented in PyTorch using custom versions of transformer modules like multi-head attention and encoder/decoder layers [7].

Image tiling is typically handled by convolutional layers bundled in deep learning libraries like PyTorch and TensorFlow, but since our model lacks convolution we use the PyTorch function `unfold` to tile the images instead.

We have prepared a Jupyter Notebook to demonstrate APN classification and generation here:

`https://colab.research.google.com/drive/1jzIi8oonKbvqXm3X7c1dlJNm8KsSvCAk`

Chapter 5

Testing and validation

5.1 Data

We use the MNIST hand-written digits dataset for evaluation [29]. MNIST is a familiar standard — the "Hello, world" of ML — and is easily accessible. Its simplicity enables rapid development and prototyping, and it continues to be used for cutting-edge models like stacked capsule autoencoders.

Across all experiments we use the same 30% validation set for consistent comparison. This gives us 42000 training images and 18000 validation images from the 60000 images available through the `torchvision.datasets` library.

5.2 Evaluation criteria

The primary evaluation criterion is unsupervised classification accuracy. Following [26] we freeze all network parameters after training, then pass the model's image representations to a classifier. We examine two different ways the model can represent an image: the outputs of the final layer and the activation pattern of the whole.

Our classifier is a sequential model consisting of a linear layer, batch normalization, dropout, ReLU activation, and a final linear layer. We compare the quality of APN representations by the classification accuracy it attains across a number of design permutations.

Throughout the course of development it was necessary to check whether the model behaved as expected, so we relied on visualizations of attention matrices and images generated from latent variables. These will be presented in the next chapter.

Chapter 6

Results

Tables 6.1 and 6.2 give the unsupervised classification accuracy achieved by attention matrices and network outputs. We did not bother to fill out Table 6.2 as exhaustively as Table 6.1 after seeing how poorly network outputs fare, but include it for comparison. "Vector dimension" refers to token vectors and the network dimension of transformer components. "APN dimensions" refers to the size of the middle and top layers. Input layers are length 49 in all cases, and the "constraints" and "accuracy" fields are self-explanatory.

Figure 6.1 gives the result of generating images from each of the 28 top-level nodes in a 40x28 network. To isolate these nodes we masked out all others in the same layer, but activated all nodes in the middle layer. Figure 6.2 shows what happens when we also mask out all but one middle layer token.

The attention matrices in Figures 6.3 and 6.4 show connection weights between nodes of different layers after being stimulated by an image from the validation set. Dark grid cells indicate weights close to zero and white grid cells indicate weights close to one.

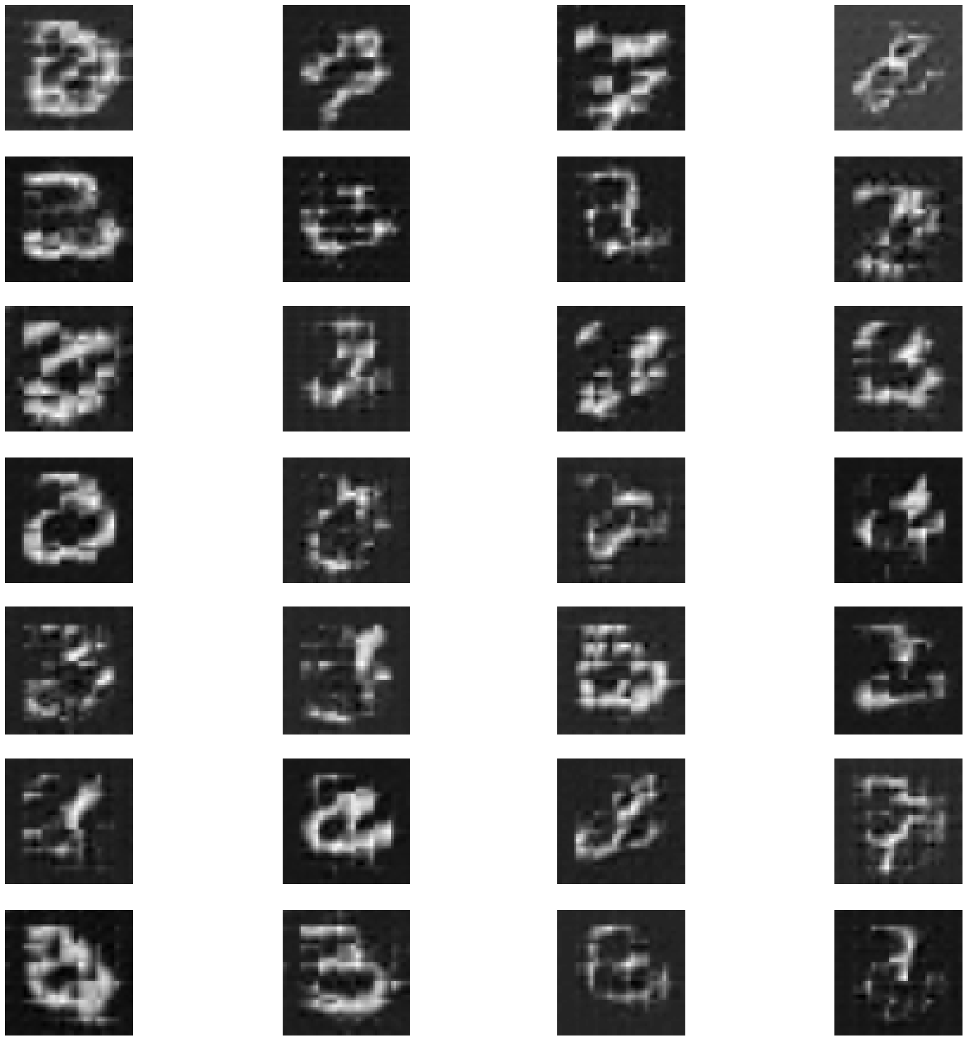

Figure 6.1: High-level representations learned by top-level nodes

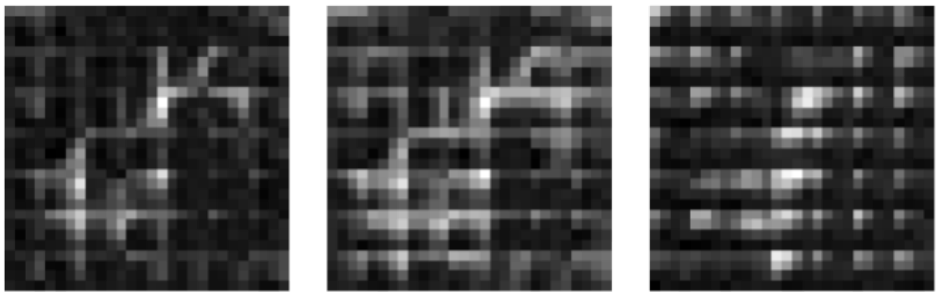

Figure 6.2: Low-level representations generated by 1 top-level node and 1 mid-level node. Left and center images generated from the same top-level node.

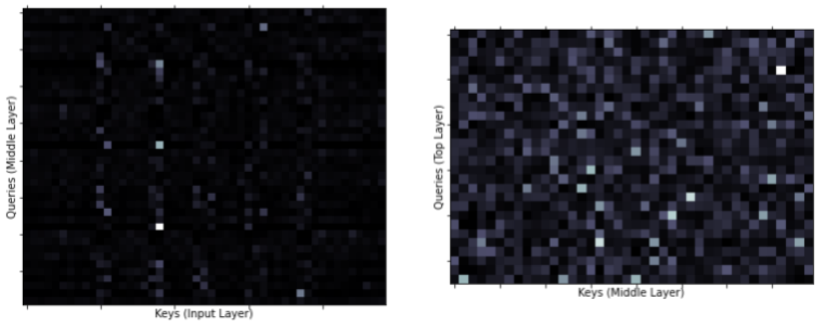

Figure 6.3: Attention matrices (no sparsity constraint). Left maps inputs to middle layer, right maps middle layer to final layer.

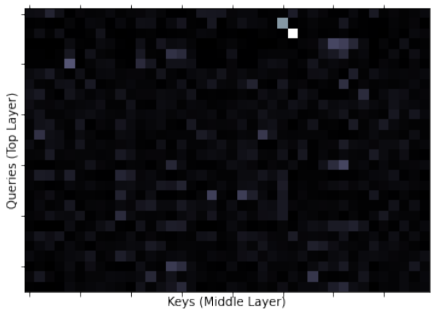

Figure 6.4: Attention matrix (with sparsity constraint). Maps middle layer to final layer.

Vector Dimension	APN Dimensions	Constraints	Accuracy (%)
32	28x10	None	90.1 ± 1.6
32	40x28	None	92.5 ± 0.3
64	28x10	None	91.5 ± 0.3
64	40x28	None	94.8 ± 0.3
128	28x10	None	93.8 ± 0.2
128	40x28	None	95.8 ± 0.3
32	28x10	Sparsity	87.6 ± 2.3
32	40x28	Sparsity	91.1 ± 0.8
64	28x10	Sparsity	89.2 ± 1.6
64	40x28	Sparsity	92.5 ± 0.8
128	28x10	Sparsity	89.5 ± 1.9
128	40x28	Sparsity	94.3 ± 0.2
32	28x10	Symmetry	62.9 ± 10.6
32	40x28	Symmetry	85.2 ± 8.8
64	28x10	Symmetry	59.4 ± 4.3
64	40x28	Symmetry	90.1 ± 3.8
128	28x10	Symmetry	51.3 ± 12.3
128	40x28	Symmetry	74.0 ± 15.7
32	28x10	All	61.9 ± 4.4
32	40x28	All	89.7 ± 1.7
64	28x10	All	59.3 ± 16.6
64	40x28	All	89.0 ± 1.3
128	28x10	All	49.6 ± 24.2
128	40x28	All	91.2 ± 0.6

Table 6.1: Unsupervised classification accuracy (attention matrix)

Vector Dimension	APN Dimensions	Constraints	Accuracy (%)
32	28x10	None	18.1 ± 3.3
32	40x28	None	36.0 ± 3.8
64	28x10	None	29.0 ± 1.8
64	40x28	None	46.1 ± 1.0
128	28x10	None	38.0 ± 1.1
128	40x28	None	46.1 ± 9.8

Table 6.2: Unsupervised classification accuracy (activations)

Chapter 7

Conclusion

7.1 Discussion of Results

7.1.1 Unsupervised classification accuracy

Table 6.1 shows that accuracy improves with network size, as one might expect. It is clear that additional constraints negatively impact performance, especially for smaller (28x10) networks. In some trial runs on small networks, more constraints introduce the risk of severely damaging the network's ability to represent images. The damage is visible in the low average score and high standard deviation of these design permutations.

The drastic performance difference between attention matrices (or activation patterns) and outputs (or activations) suggests that in networks of this type, attention matrices are far more representative of inputs than outputs from the final layer. The outperformance is clearer when we reflect that, since outputs scale linearly with vector dimension, flattened outputs yield larger inputs to classifiers than attention matrices. Thus attention matrices not only perform much better — they do so with fewer parameters.

Since attention matrices encode images independently of network dimension, it may be possible to use large, powerful transformers to compress inputs into compact, fixed-size representations. We find that the larger the network, the higher quality the representation, even if the representation size remains constant.

7.1.2 Generation

The generated images in Figures 6.1 and 6.2 show that the network exhibits hierarchical compositionality, as some top-level nodes are vaguely recognizable as MNIST digits while mid-level nodes seem to be responsible for local patterns.

Tiling artifacts show up in all images but are especially noticeable in Figure 6.2. We refer to the boundary distortions and "grid lines" that lie upon the image. It may be possible to eliminate these artifacts with overlapping tiles.

7.1.3 Constraints and the parse tree

The left hand attention matrix in Figure 6.3 shows that input attention is naturally sparse. The model needs only attend to a select few tiles in the input.

The right hand attention matrix shows that higher level abstractions are distributed much more evenly. It is possible to force the mid-top layer attention matrix to be sparse, as in Figure 6.4, but we found that the model "cheated" with a trivial solution: it activated the same one or two tokens regardless of input, then distributed the rest of its activity as needed.

Consequently we were unable to achieve one of our major goals: to carve a parse tree out of network attention matrices. The L1 sparsity loss was simply not up to the task. Similarly the disastrous performance hits from symmetry constraints reinforce the lesson that it is best not to interfere with the network.

A primary purpose of building a parse tree was to build interpretable representations that compose hierarchically. Specifically we wanted certain subsets of data to correspond to specific nodes in each layer. However, even without a parse tree, much of the desired behavior is already exhibited in the model. Through generation we see hierarchical compositionality, and through visualizing attention matrices we see that the input routing is naturally sparse.

Perhaps part of the inability to induce a parse tree is related to the size of the networks. We see that from the middle to the final layer attention is widely distributed, and perhaps this level of activity is necessary. Further experimentation with higher capacity networks — that is, with wider layers of learned queries more than two layers deep — may reveal that parse trees are possible, just not for the limited configurations explored here.

7.2 Summary of contributions

In one sense this study is a failure: we set out to build attentional *parsing* networks, but were unable to force them to look like parse trees. On the other hand, we have successfully implemented a vector-based multilayer network with content-addressable routing. Our network builds hierarchical, compositional representations, and these representations can be visualized through generation. Finally, we have found that the pattern of activity within the network effectively encodes the input.

With only slight modifications to the standard transformer, and with relatively few network parameters, it is possible to come within 3 percent of the state of the art for unsupervised image representation — 95.8% classification accuracy versus the 98.7% of SCAEs [26].

Chapter 8

Future work

Here are some questions that future work might aim to answer:

- Can we build a parse tree into the network with different loss functions? For example, the SCAE training regimen encourages sparsity by selecting a subset of the final layer that depends on the number of classes [26]. How might this affect performance?

- Can we eliminate tiling artifacts by using overlapping patches? This may not be necessary since tiling artifacts do not show up during reconstruction, but it may be desirable for higher quality latent variable visualizations.

- How might we extend our work to larger images? To different datasets? To different domains? Adapting to larger images should be relatively trivial: simply use larger patches and downsample rather than upsample. It may also be possible to apply APNs, with their general architecture, to language parsing datasets like Penn Treebank [30].

  If we can maintain high unsupervised classification accuracy for standard-sized images with a large number of classes, then it will be clear that the high quality of the model's learned representations is not a "fluke" or artifact of factors arising from the use of powerful models on a toy dataset.

- What kinds of compression rates can be achieved with the presented architecture? How does the quality of representations scale with transformer dimensions greater than the 128 tested here? Is there an optimal tradeoff point between the size of the transformer and the size of attention matrices?

- How effectively can unsupervised representations be transferred to different tasks for fine-tuning? Transfer learning and semi-supervised learning currently rely on data augmentation and contrastive methods [31], but APNs open the door to a regularized latent variable approach for unsupervised image learning.

- There is another class of network that encodes inputs as patterns of activation: autoassociative networks. How deep is this connection, and is it possible to implement classical examples of autoassociative networks with a variant of our so-called "attentional" networks? Recent work has drawn a parallel between transformers and autoassociative Hopfield networks [32], and our findings may shed further light on this nascent line of inquiry.

In many respects this paper is a preliminary work that can be significantly expanded upon. We believe that recontextualizing sequence-to-sequence transformers as arbitrary networks of latent variables can potentially lead to applications and insights well outside the limited scope of this work.

Bibliography

[1] J. Su, D. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019. [Online]. Available: https://arxiv.org/pdf/1710.08864

[2] J. Walsh and N. O'Mahoney, "Deep learning vs. traditional computer vision," *Computer Vision Conference (CVC)*, 2019. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1910/1910.13796.pdf

[3] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017, pMID: 28599112. [Online]. Available: https://doi.org/10.1162/neco_a_00990

[4] Z. Tu, X. Chen, A. Yuille, and S. Zhu, "Image parsing: Unifying segmentation, detection, and recognition," *International Journal of Computer Vision*, vol. 63, pp. 113–140, 2005. [Online]. Available: http://www.stat.ucla.edu/~sczhu/papers/IJCV_parsing.pdf

[5] S. Sabour, N. Frosst, and G. Hinton, "Dynamic routing between capsules," *Neural Information Processing System (NIPS)*, 2017. [Online]. Available: https://arxiv.org/pdf/1710.09829.pdf

[6] L. Zhu, Y. Chen, and A. Yuille, "Recursive compositional models for vision: Description and review of recent work," *Journal of Mathematical Imaging and Vision*, vol. 41, pp. 122–146, 2011. [Online]. Available: http://www.cs.jhu.edu/~ayuille/Pubs10_12/JMIVyuilleB.pdf

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistic (NAACL)*, 2019. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf

[9] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *ArXiv*, vol. abs/1506.00019, 2015. [Online]. Available: https://arxiv.org/pdf/1506.00019.pdf

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *Computing Research*

Repository (CoRR), vol. abs/1409.0473, 2015. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf

[12] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *ArXiv*, vol. abs/1508.04025, 2015. [Online]. Available: https://arxiv.org/pdf/1508.04025.pdf

[13] A. Gillioz, J. Casas, E. Mugellini, and O. A. Khaled, "Overview of the transformer-based models for nlp tasks," *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pp. 179–183, 2020. [Online]. Available: https://annals-csis.org/proceedings/2020/drp/pdf/20.pdf

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

[15] R. Zhang, L. Du, Q. Xiao, and J. Liu, "Comparison of backbones for semantic segmentation network," *International Conference on Signal Processing (ICSP)*, vol. 1544, March 2020. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1544/1/012196/pdf

[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krüger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *ArXiv*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/pdf/2005.14165.pdf

[17] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *ArXiv*, vol. abs/2004.05150, 2020. [Online]. Available: https://arxiv.org/pdf/2004.05150.pdf

[18] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," *ArXiv*, vol. abs/1901.02860, 2019. [Online]. Available: https://arxiv.org/pdf/1901.02860.pdf

[19] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *ArXiv*, vol. abs/2001.04451, 2020. [Online]. Available: https://arxiv.org/pdf/2001.04451.pdf

[20] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," in *Neural Information Processing Systems (NeurIPS)*, 2019. [Online]. Available: https://arxiv.org/pdf/1906.05909.pdf

[21] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *ArXiv*, vol. abs/2005.12872, 2020. [Online]. Available: https://arxiv.org/pdf/2005.12872.pdf

[22] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen, "Axial-deeplab: Stand-alone axial-attention for panoptic segmentation," in *European Conference on Computer Vision (ECCV)*, 2020. [Online]. Available: https://arxiv.org/pdf/2003.07853.pdf

[23] Preprint, "An image is worth 16x16 words: Transformers for image recognition at scale," *International Conference*

on Learning Representations (ICLR), 2020. [Online]. Available: https://openreview.net/pdf?id=YicbFdNTTy

[24] D. George, "A generative vision model that trains with high data efficiency and breaks text-based captchas," *Science*, December 2017.

[25] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ser. ICANN'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 44–51. [Online]. Available: http://nlp.stanford.edu/~sidaw/home/_media/papers:transauto3.pdf

[26] A. R. Kosiorek, S. Sabour, Y. Teh, and G. E. Hinton, "Stacked capsule autoencoders," *ArXiv*, vol. abs/1906.06818, 2019. [Online]. Available: https://arxiv.org/pdf/1906.06818.pdf

[27] E. Candès, M. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted 1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, pp. 877–905, 2007. [Online]. Available: https://authors.library.caltech.edu/12852/3/0711.1612.pdf

[28] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

[30] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a large annotated corpus of english: The penn treebank,” *Comput. Linguistics*, vol. 19, pp. 313–330, 1993. [Online]. Available: https://pdfs.semanticscholar.org/b6cb/2a910294a8c0f8dd2ed212ec36c1721ea8bc.pdf

[31] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, “A simple framework for contrastive learning of visual representations,” *ArXiv*, vol. abs/2002.05709, 2020. [Online]. Available: https://arxiv.org/pdf/2002.05709.pdf

[32] H. Ramsauer, B. Schafl, J. M. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. P. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, “Hopfield networks is all you need,” *ArXiv*, vol. abs/2008.02217, 2020. [Online]. Available: https://arxiv.org/pdf/2008.02217.pdf