NLP Reading Group — Meeting 5

Matt Smith

UW Data Science Club

4 April, 2019

Coverage

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
2. Deep Visual-Semantic Alignments for Generating Image Descriptions
3. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
4. The Evolved Transformer
5. Universal Transformers

I was wrong!

There's an extra element to the BERT paper that I didn't mention.

So let's cover it! :)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Combines:

- Bidirectional LMs
- ULMFiT finetuning
- OpenAI-style transformers
- ELMO

Using OpenAI-style transformers and finetuning is simple enough

Bidirectionality poses a challenge...

Bidirectionality

There are two ways of transferring context from pre-trained networks: feature-based and fine-tuning

ELMO is the SOTA feature-based method, and is combines the left and right contexts, conditioned independently, to create its embeddings

GPT is conditioned on just the left-to-right context

Can we combine them?

Bidirectionality

There are two ways of transferring context from pre-trained networks: feature-based and fine-tuning

ELMO is the SOTA feature-based method, and is combines the left and right contexts, conditioned independently, to create its embeddings

GPT is conditioned on just the left-to-right context

Can we combine them?

Yes! BERT's output at every step is conditioned on both the left and right context (this is superior to just concatenating the left and right context after every layer)

Bidirectional LMs

The language model we used previously to pre-train our transformers was to predict the next word given a sentence fragment

In the bidirectional case, there is no 'next' word. So the authors created a new LM objective

For 15% of the tokens (word pieces in this case) in the input sentence:

- replace them with a blank token 80% of the time
- replace them with a random token 10% of the time
- do nothing 10% of the time

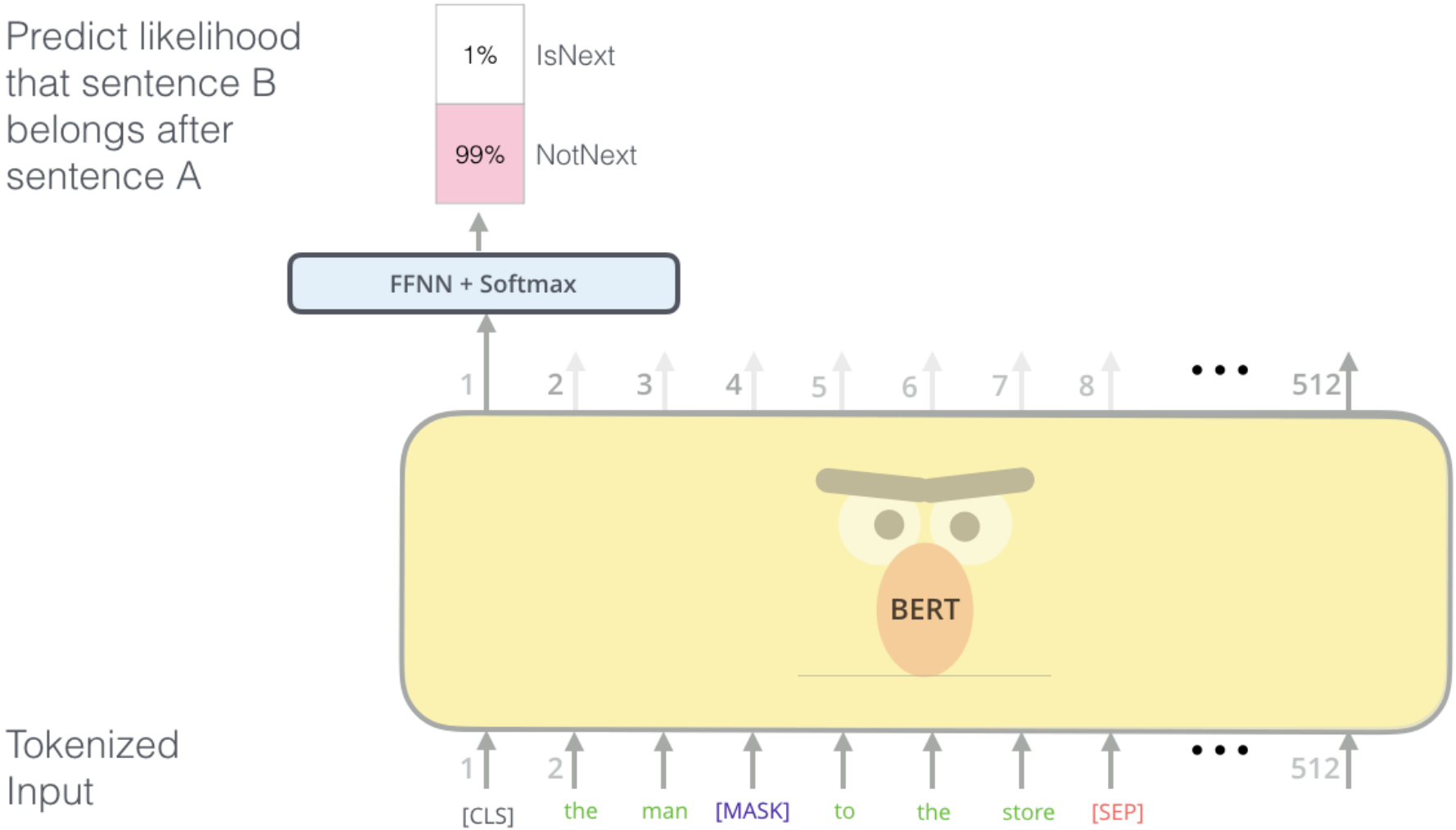
BERT must now predict which tokens were changed

Next Sentence Prediction Task

Because this LM task doesn't capture sentence-to-sentence relationships well, there is another pre-training task

Given two masked sentences, predict whether the second one follows the first one in the training corpus or not

Next Sentence Prediction Task



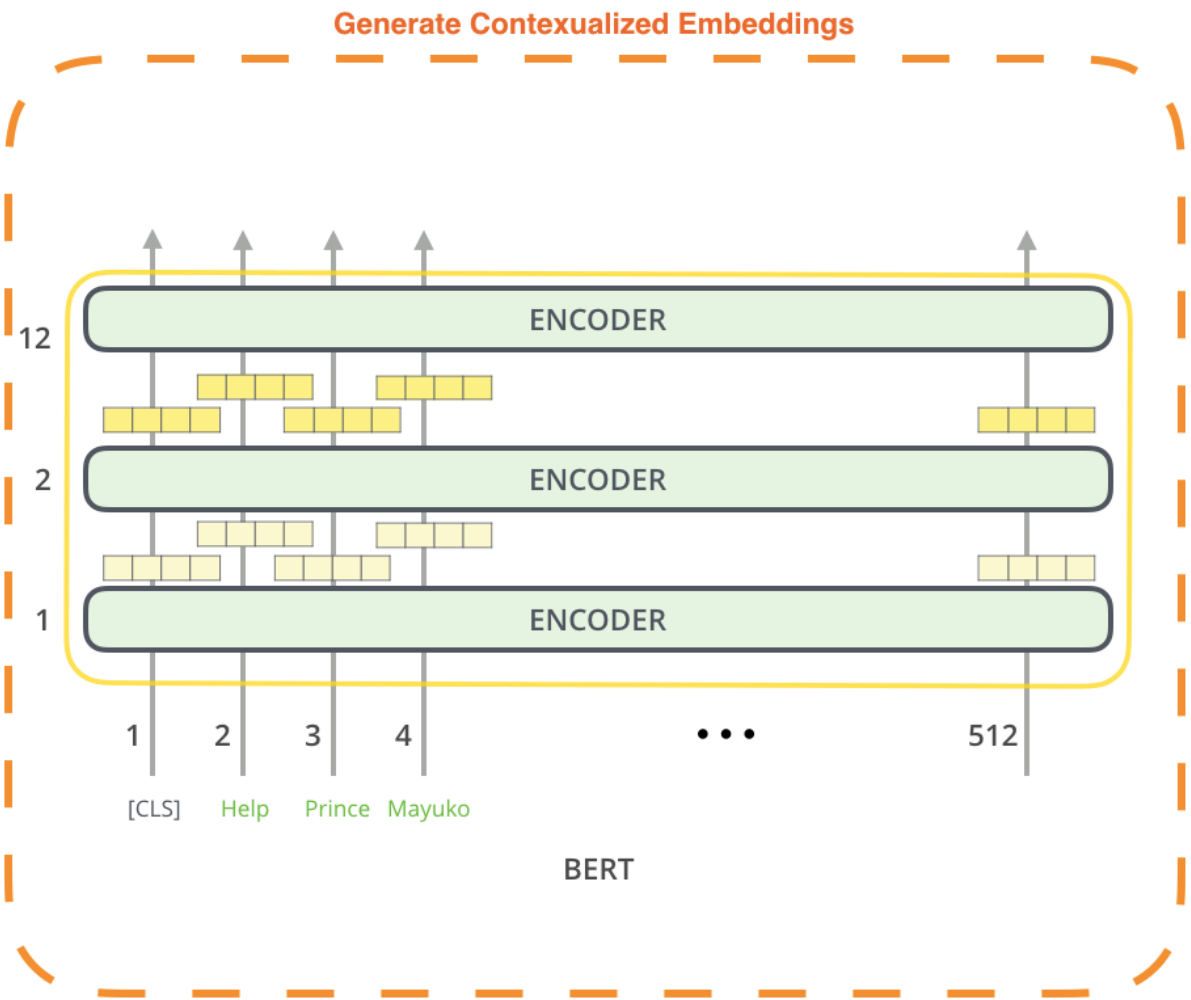
What do Onions and Ogres Have In Common?

So now we have a bidirectional LM that we can finetune in a general way using task-specific input transformations and it's great

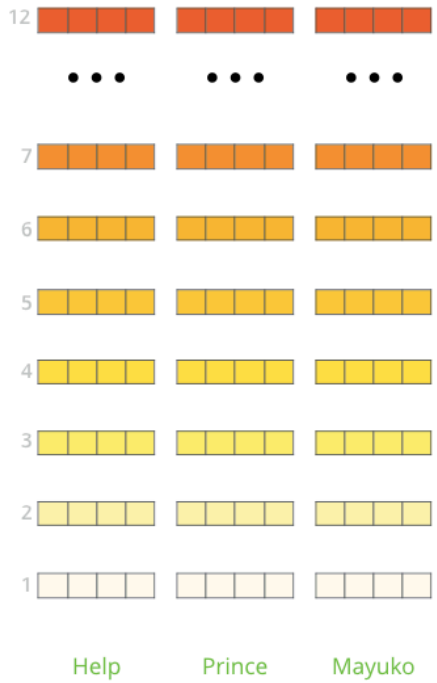
Remember in ELMO, when we used a weighted sum of the internal layers of a biLM to create better embeddings?

BERT is a biLM with layers too! So we can take an internal sum of *it's* internal layers to create (hopefully better) embeddings

BERT Embeddings



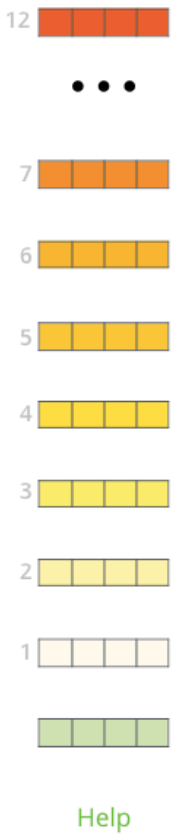
The output of each encoder layer along each token's path can be used as a feature representing that token.

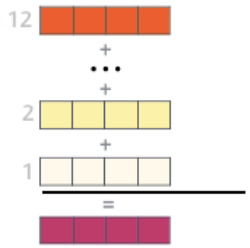
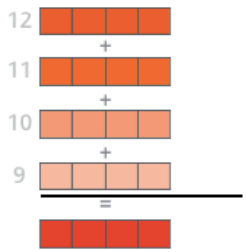
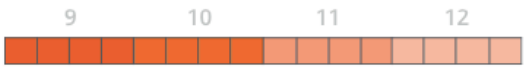


But which one should we use?

BERT Embeddings Performance

What is the best contextualized embedding for "Help" in that context?
For named-entity recognition task CoNLL-2003 NER



		Dev F1 Score
First Layer		91.0
Last Hidden Layer		94.9
Sum All 12 Layers		95.5
Second-to-Last Hidden Layer		95.6
Sum Last Four Hidden		95.9
Concat Last Four Hidden		96.1

Coverage

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
2. Deep Visual-Semantic Alignments for Generating Image Descriptions
3. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
4. The Evolved Transformer
5. Universal Transformers

Last Time

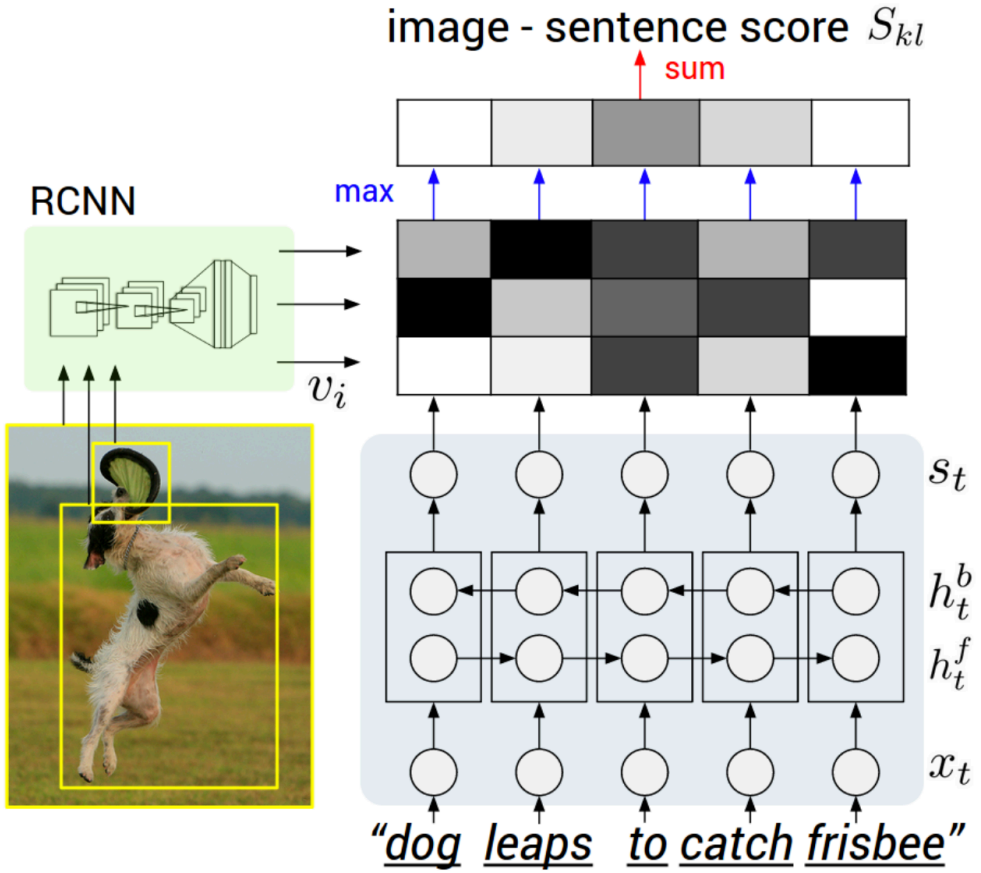


Figure 3. Diagram for evaluating the image-sentence score S_{kl}. Object regions are embedded with a CNN (left). Words (enriched by their context) are embedded in the same multimodal space with a BRNN (right). Pairwise similarities are computed with inner products (magnitudes shown in grayscale) and finally reduced to image-sentence score with Equation 8.

Last Time

Find a set of alignments that prefer to align adjacent words to the same bounding box. a_j is an alignment for word j, and it takes of values of $1, 2, \ldots M$ where there are M bounding boxes.

$$\psi_j^U(a_j = t) = v_j^T s_t$$

$$\psi_j^B(a_j, a_{j+1}) = \beta \mathbf{1}[a_j = a_{j+1}]$$

$$E(a) = \sum_{j=1..N} \psi_j^U(a_j) + \sum_{j=1..N-1} \psi_j^B(a_j, a_{j+1})$$

What the heck is β?

β is treated as a hyperparameter here. Higher values of β gives a larger reward for assigning adjacent words to the same bounding box.

Caption Generation

We can now take a set of regions and a caption and find the regions that best describe snippets of the caption. Now we would like to generate the caption ourselves.

The authors use a RNN language model to predict the caption for an image using a CNN encoding to initialize the RNN

RNN Choice

One notable thing is that both this and the previous RNN are vanilla RNNs and not LSTM/GRUs!

$$b_v = W_{hi}[\text{CNN}_{\theta_c}(I)]$$

$$h_t = relu(W_{hx}x_t + W_{hh}h_{t-1} + b_h + \mathbf{1}[t=1]b_v)$$

$$y_t = softmax(W_{oh}h_t + b_o)$$

Model

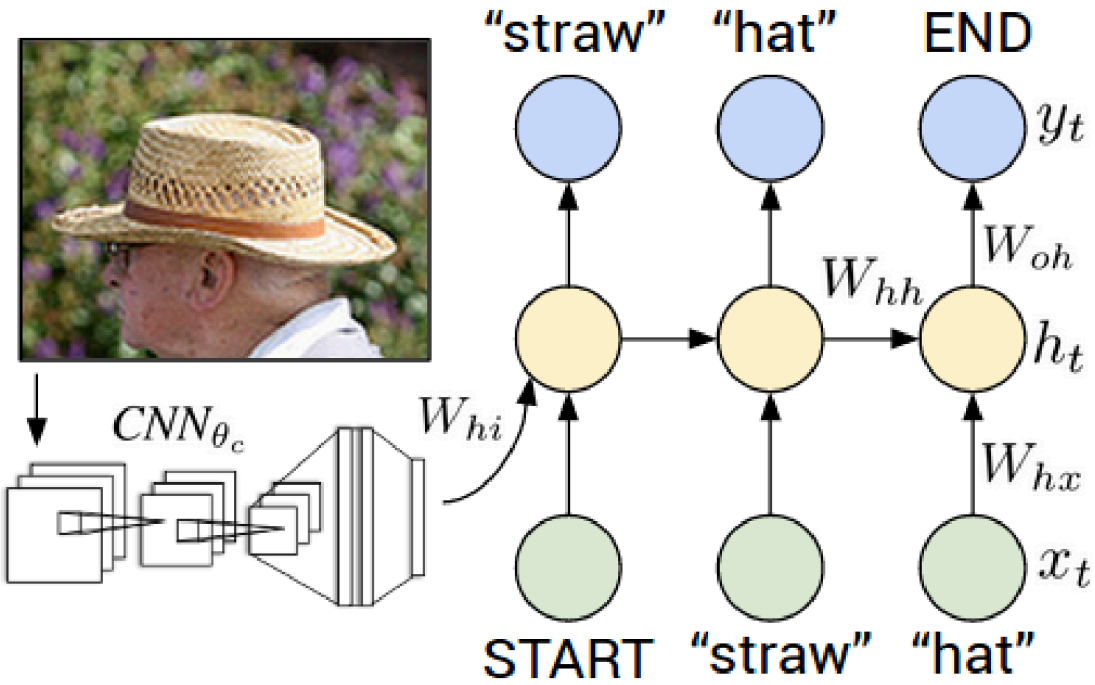


Figure 4. Diagram of our multimodal Recurrent Neural Network generative model. The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence. The RNN is conditioned on the image information at the first time step. START and END are special tokens.

Training

We use SGD with mini-batches of 100 image-sentence pairs and momentum of 0.9 to optimize the alignment model. We cross-validate the learning rate and the weight decay. We also use dropout regularization in all layers except in the recurrent layers and clip gradients elementwise at 5 (important). The generative RNN is more difficult to optimize, party due to the word frequency disparity between rare words and common words (e.g. ” a” or the END token). We achieved the best results using RMSprop, which is an adaptive step size method that scales the update of each weight by a running average of its gradient norm.

Momentum

When we perform gradient descent, we can often get stuck in small plateaus or local minima.

Instead of performing a standard gradient update, we take a weighted sum of our current gradient and our previous update, called our 'momentum'.

The idea is that this momentum will carry us over plateaus or small local minima to find a much better local minimum.

Gradient Clipping

Many RNNs have cliff-like behaviour in their loss functions.

Taking large steps with SGD near these cliffs can cause bad convergence behaviours.

It's also just a good idea in general to limit your step size.

Gradient clipping reduces the magnitude of the loss gradient if it is above a certain threshold.

Dropout

Randomly zero-out some of the nodes in our neural network, changing the nodes we zero-out on each training iteration.

Dropout at Evaluation Time

Remember not to apply dropout when you are evaluating your network. Most deep learning frameworks will do this for you

This gives a powerful regularizing effect.

The motivation for dropout is that it is like ensembling many smaller networks, and ensembling is Really Good ™.

Cross-Validation

Earlier we talked about having a training set and a dev set. This is one way of validating a choice for hyperparameter, and is called 'holdout cross-validation'.

k-fold CV

Another way of doing cross-validation is k-fold CV, where your dataset is partitioned into k folds. The model is trained k times, once on each subset of $k-1$ folds, and the remaining fold is used to validate the model. The results are averaged over the k runs.

Cross-validation is a way of partitioning your training data in order to validate your model. Here the authors are using cross-validation to pick the best learning weight and weight decay for their optimizer.

Section 4

Section 4 of the paper details experiments they did to evaluate their model as well as try to extend it.

I don't really like this part of the paper but you can read it if you wish.

Let's just look at some more pretty pictures and call it a day.

But first, they mention how they got their training data...

Datasets: We use the Flickr8K, Flickr30K and MSCOCO datasets in our experiments. These datasets contain 8,000, 31,000 and 123,000 images respectively and each is annotated with 5 sentences using Amazon Mechanical Turk. For Flickr8K and Flickr30K, we use 1,000 images for validation, 1,000 for testing and the rest for training. For MSCOCO we use 5,000 images for both validation and testing.

Data Preprocessing: We convert all sentences to lowercase, discard non-alphanumeric characters. We filter words to those that occur at least 5 times in the training set, which results in 2538, 7414, and 8791 words for Flickr8k, Flickr30K, and MSCOCO datasets respectively.

Amazon Mechanical Turk

Mechanical Turk: Humans that you pay to label data for you.

Often used to generate training data or evaluate the output of your model to make sure it's sane.

They can answer yes/no questions, draw bounding boxes, object detection, facial landmark detection, etc.

The other big one is FigureEight/Crowdflower but Google has their own offering as well for computer vision data.

More Examples

Figure 11. Additional examples of alignments. For each query test image above we retrieve the most compatible sentence from the test set and show the alignments.

More Examples

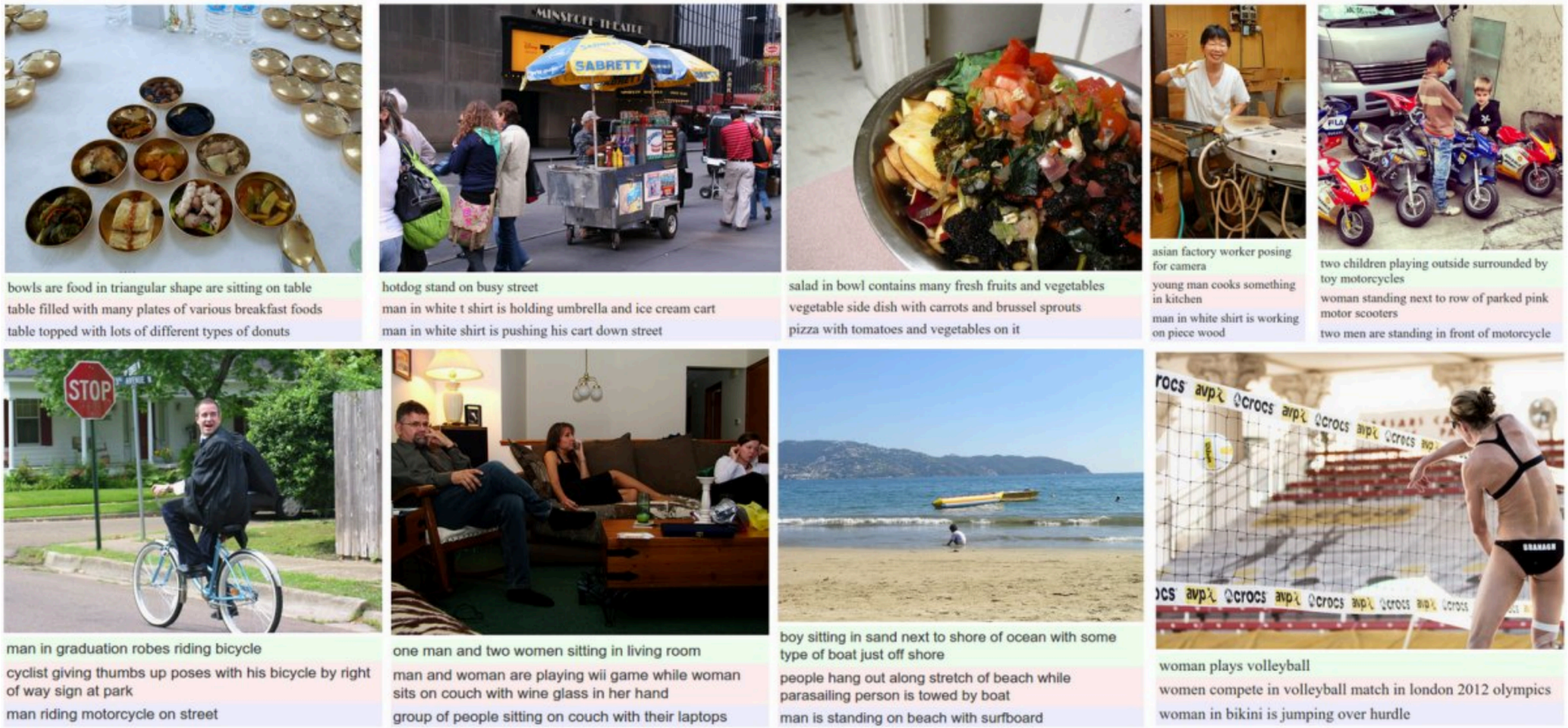


Figure 12. Additional examples of captions on the level of full images. Green: Human ground truth. Red: Top-scoring sentence from training set. Blue: Generated sentence.

Coverage

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
2. Deep Visual-Semantic Alignments for Generating Image Descriptions
3. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
4. The Evolved Transformer
5. Universal Transformers

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

An important note about transformers is because they don't have recurrent connections, they only consider a fixed-size context

While these contexts are sufficient to get amazing results on many sentence understanding tasks, they are insufficient for others

Transformer-XL adds a recurrent mechanism to transformer networks

Understanding Long Sequences With Transformers

Suppose the transformer takes a context of length k

How to predict word x_{n+1} given words $x_1 x_2 \dots x_n$?

Pass $x_{n-k} x_{n-k+1} \dots x_n$ into the transformer

Thus to predict a sentence of length n requires $n - k + 1$ calls to the transformer

This also means that our training on the first/last word of a sentence is limited

Understanding Long Sequences With Transformer-XLs

Suppose the transformer takes a context of length k

How to predict word x_{n+1} given words $x_1 x_2 \dots x_n$?

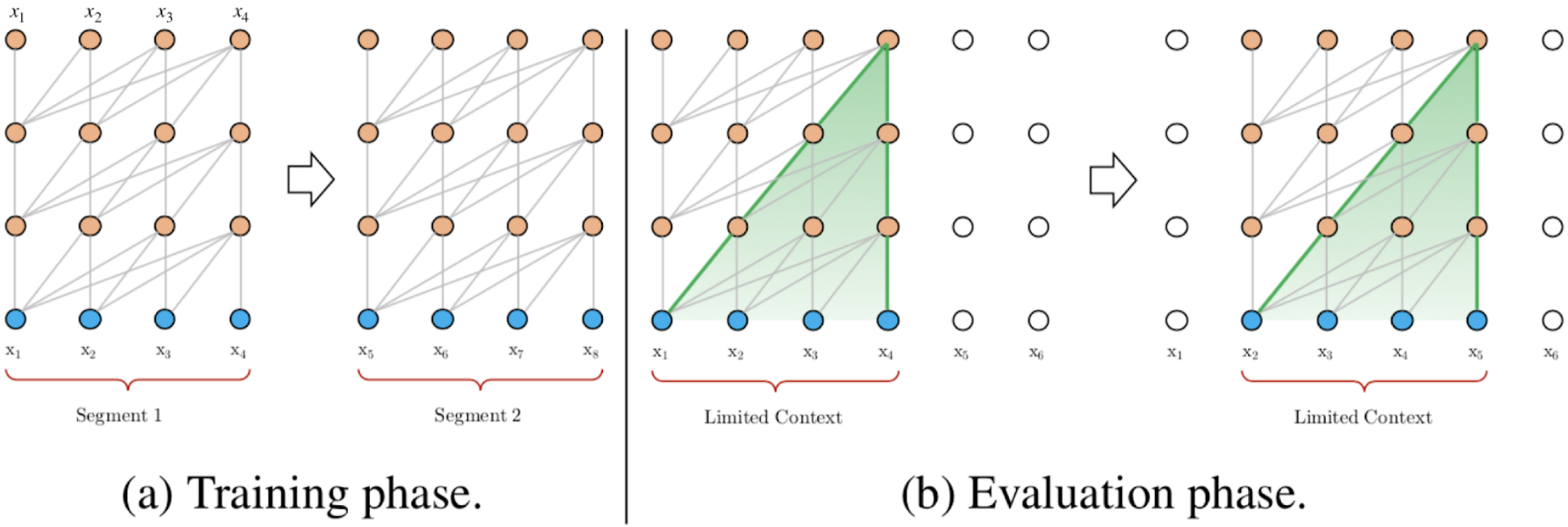
Break the sentence into blocks of length k

The prediction for the words after the start of the ith block is based on the $i-1$th block and ith block

Thus to predict a sentence of length n requires n/k calls to the transformer

So we also get a nice performance boost from this :)

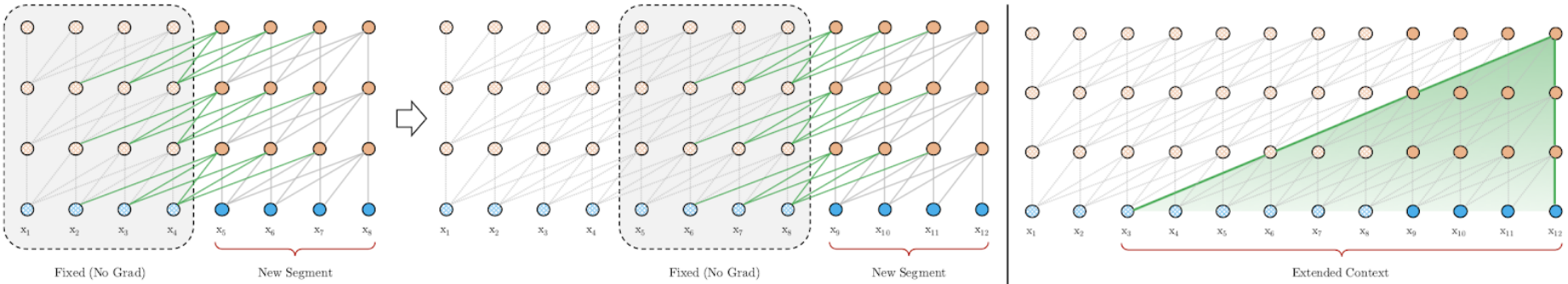
Vanilla Transformer Context



(a) Training phase.

(b) Evaluation phase.

Transformer-XL Context



(a) Training phase.

(b) Evaluation phase.

New Positional Embeddings

Vanilla transformers used fixed per-segment noise to encode absolute word order

However since Transformer-XLs have recurrent connections, they may see these encodings repeated multiple times, causing confusion

Instead of encoding absolute positional information, we only store relative positional information

Longer Recurrent Connections in Training

The authors only passed the intermediate values from the previous segment to the next segment during training

However they found that during evaluation they could pass intermediate values from several previous blocks

This was limited only by available GPU memory as there were no bad generalization effects observed

Coverage

The Question

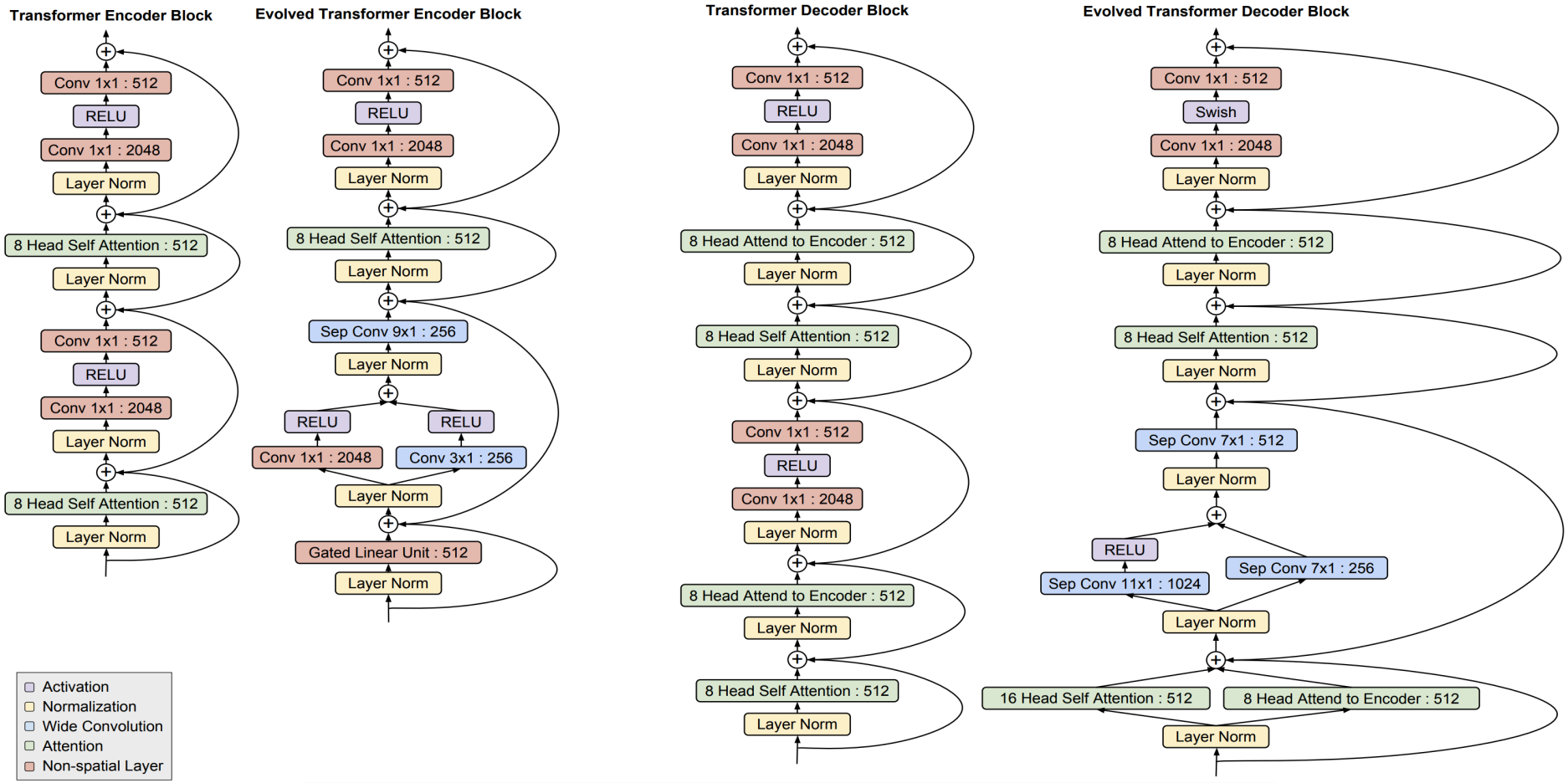
Can we use NAS (Neural Architecture Search) to find the optimal transformer network for English-to-German translation?

The Question

Can we use NAS (Neural Architecture Search) to find the optimal transformer network for English-to-German translation?

Yes, using 12 hours on 270 TPUv2s

Model


Transformer Encoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Self Attention : 512
Layer Norm
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Self Attention : 512
Layer Norm
Evolved Transformer Encoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Self Attention : 512
Layer Norm
Sep Conv 9x1 : 256
Layer Norm
RELU
RELU
Conv 1x1 : 2048
Conv 3x1 : 256
Layer Norm
Gated Linear Unit : 512
Layer Norm
Transformer Decoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Evolved Transformer Decoder Block
Conv 1x1 : 512
Swish
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
Sep Conv 7x1 : 512
Layer Norm
RELU
Sep Conv 7x1 : 256
Sep Conv 11x1 : 1024
Layer Norm
16 Head Self Attention : 512
8 Head Attend to Encoder : 512
Layer Norm
Activation
Normalization
Wide Convolution
Attention
Non-spatial Layer

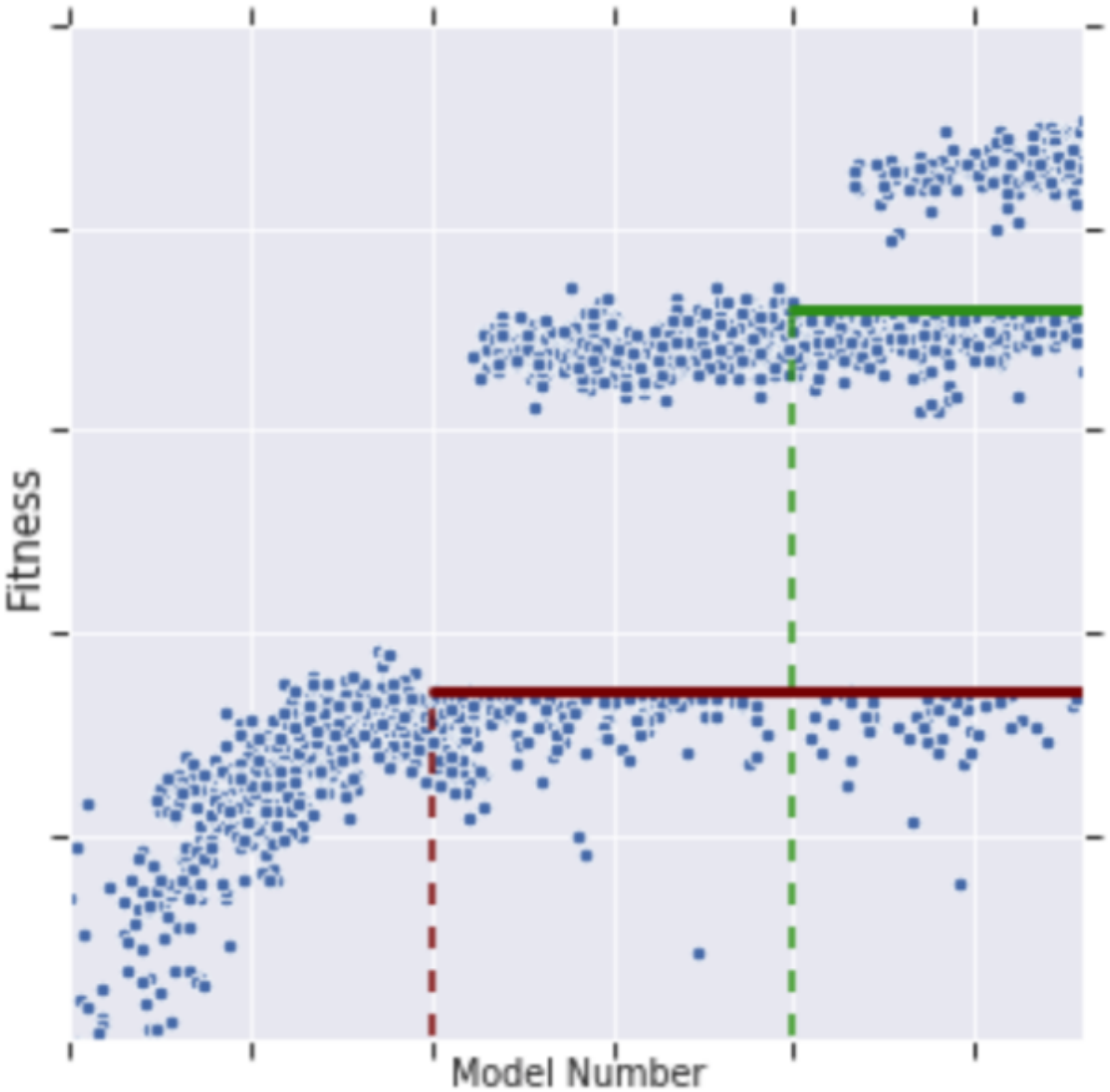
Efficient NAS

Our production datasets are often too large to efficiently train enough models to perform NAS.

Historically the solution is to use a smaller dataset with a similar domain (e.g. when optimizing for Imagenet, train on CIFAR-10).

Instead, the authors did something they call Progressive Dynamic Hurdles (PDH).

PDH



Initializing Evolution

Seed Model	Train Steps	Num Models	Top Model Perplexity
Transformer	PDH	6000	**4.50** $\pm$ 0.01
Random	PDH	6000	5.23 $\pm$ 0.19
Transformer	30K	14857	4.53 $\pm$ 0.07
Transformer	180K	2477	4.58 $\pm$ 0.05
Transformer	300K	1486	4.61 $\pm$ 0.02

Results

Model	Embedding Size	FLOPS	Parameters	Perplexity	BLEU	Δ BLEU
Transformer	128	0.15T	7.0M	8.62 ± 0.03	21.3 ± 0.1	-
ET	128	0.14T	7.2M	**7.62** ± 0.02	**22.0** ± 0.1	**+ 0.7**
Transformer	432	0.90T	45.8M	4.65 ± 0.01	27.3 ± 0.1	-
ET	432	0.85T	47.9M	**4.36** ± 0.01	**27.7** ± 0.1	+ 0.4
Transformer	512	1.19T	61.1M	4.46 ± 0.01	27.7 ± 0.1	-
ET	512	1.12T	64.1M	**4.22** ± 0.01	**28.2** ± 0.1	+ 0.5
Transformer	768	2.39T	124.8M	4.18 ± 0.01	28.5 ± 0.1	-
ET	768	2.23T	131.2M	**4.00** ± 0.01	**28.9** ± 0.1	+ 0.4
Transformer	1024	4.04T	210.4M	4.05 ± 0.01	28.8 ± 0.2	-
ET	1024	3.70T	221.7M	**3.94** ± 0.01	**29.0** ± 0.1	+ 0.2

Coverage

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
2. Deep Visual-Semantic Alignments for Generating Image Descriptions
3. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
4. The Evolved Transformer
5. Universal Transformers

Universality

It turns out that transformers aren't universal computation devices (can't simulate a TM)

They struggle to generally learn some simple tasks such as copying the input to the output

By modifying transformers to give them a recurrent step, universal transformers are universal computation devices that also outperform vanilla transformers on some tasks

Recurrence in Transformers

The original transformers has a fixed number of stacked self-attention layers

The authors of *Universal Transformers* say that this stops the transformer from having the bias towards inductive or recursive behaviour that RNNs have

Universal transformers are allowed to perform stacked self-attention any number of times they want per each input word

The model predicts the probability that computation on a word should halt

Animation?!