CS 182 Lecture Notes, Spring 2022

Deep Learning

Professor: Marvin Zhang
Vishal Raman

Contents

1 02/28/2022 - Recurrent Neural Networks 5
- 1.1 Problem Setup 5
- 1.2 Dealing with Variable Length Inputs 5
- 1.3 Recurrent Neural Networks 6
- 1.4 Generating Outputs from RNNs 7
- 1.5 Vanishing/Exploding Gradients 7

2 02/28/2022 - Long Short-Term Memory(LSTM) 9
- 2.1 Bidirectional RNN Models 9

3 3/7/2022 - Transformers 11
- 3.1 Setup 11
- 3.2 Why Transformers 11
- 3.3 Attention 11
  - 3.3.1 What to attend over? 12
  - 3.3.2 Details of Attention 12
  - 3.3.3 Self-Attention 13
- 3.4 Transformers 13
  - 3.4.1 Transformer Encoder 13
  - 3.4.2 Positional Encoding 14
  - 3.4.3 Multi-Head Attention 14
  - 3.4.4 Transformer Decoder 15
  - 3.4.5 Masked Attention 15

4 03/09/2022 - Transformer Models 17
- 4.1 Encoder: BERT 17
  - 4.1.1 The GLUE Benchmark 17
- 4.2 Encoder: Vision Transformer 18
- 4.3 Swin Transformer 19
- 4.4 Unsupervised ViT 19
  - 4.4.1 Masked Autoencoders 19

5 03/14/2022 - Sequence-to-Sequence Models 21
- 5.1 Seq2seq Models 21
  - 5.1.1 RNN(LSTM) seq2seq, basic 21
  - 5.1.2 Multilayer RNNs 21
  - 5.1.3 Attention 22
- 5.2 Seq2Seq Transformers 23
  - 5.2.1 Cross Attention Layers 23
- 5.3 Recent seq2seq: T5 23

6 03/16/2022 - John DeNero: Neural Machine Translation 24
- 6.1 Conditional Sequence Generation 24
- 6.2 Search Strategies 24
- 6.3 Training Loss 25
- 6.4 Subwords 26
  - 6.4.1 Byte-Pair Encoding 26
- 6.5 Back Translations 26

7 03/28/2022 - Distribution Shift 27
7.1 Risk . . . 27
7.1.1 The ERM Assumption . . . 27
7.2 Distribution Shift in the Real World . . . 27
7.3 Distribution Shift Benchmarks . . . 28
7.3.1 ImageNet Challenge Test Sets . . . 28
7.3.2 WILDS Benchmark . . . 28
7.4 NLP: ANLI Dataset . . . 29
7.5 Robustification . . . 29
7.5.1 Train on Larger Datasets . . . 29
7.5.2 Data Augmentations . . . 29
7.5.3 Masked Autoencoders . . . 30
7.6 Anomaly Detection . . . 30
7.6.1 Implementation . . . 30
7.6.2 A Simple Baseline . . . 31
7.6.3 Benchmarks for Anomaly Detection . . . 31
7.6.4 Evaluting Binary Classifiers . . . 31

8 04/04/2022 - Adversarial Examples 32
8.1 Test time Adaptation . . . 32
8.1.1 Methods for Test-time Adaptation . . . 32
8.2 Adversarial Robustness . . . 32
8.2.1 Modern Adversarial Distortions . . . 33
8.2.2 Fooling Binary Logistic Regression . . . 33
8.3 Adversary Threat Models . . . 33
8.3.1 Fast Gradient Sign Method(FGSM) . . . 34
8.3.2 Projected Gradient Ascent(PGD) . . . 34
8.4 Adversarial Training . . . 34
8.4.1 Untargeted vs. Targeted Attacks . . . 34
8.4.2 Transferability of attacks . . . 35
8.4.3 Using Larger and More Diverse Data . . . 35
8.4.4 Data Augmentation . . . 35
8.4.5 Choice of Activation Functions . . . 35
8.5 Unforeseen Adversaries . . . 35

9 04/06/2022 - Generative Models 37
9.1 Density/Distribution Modeling . . . 37
9.2 Generating New Data . . . 37
9.3 Generative Adversarial Networks . . . 37
9.3.1 Conditional Generation with GANs . . . 38
9.3.2 Unsupervised Image-to-Image Translation . . . 38
9.3.3 Summary . . . 38
9.4 Density Models on Images . . . 38
9.5 Autoregressive Models . . . 38
9.5.1 Distribution over Pixels . . . 39
9.5.2 Summary . . . 39
9.6 Representation Learning . . . 39
9.6.1 Latent Variable Models . . . 39
9.6.2 Evidence Lower Bound(ELBO) . . . 40
9.6.3 Variational Autoencoders(VAEs) . . . 40

9.6.4 Summary . . . 40
9.7 Generating Images from Language . . . 40

10 04/11/2022 - Self-Supervised Learning 41
10.0.1 Why Self-Supervised . . . 41
10.0.2 Examples . . . 41
10.0.3 Masked Autoencoding(MAE) . . . 41
10.1 Self-supervised Learning in Vision . . . 41
10.1.1 Leveraging Spatial Context(2015) . . . 42
10.1.2 Predicting Rotations(2018) . . . 42
10.2 Contrastive Learning . . . 42
10.2.1 Contrastive Predictive Coding(CPC)(2018) . . . 42
10.3 Data Augmentation . . . 43
10.3.1 Momentum Contrast(MoCo)(2019) . . . 43
10.3.2 SimCLR . . . 43
10.4 Bootstrap your own latent(BYOL)(2020) . . . 44
10.5 Self-distillation with no labels (DINO)(2021) . . . 44
10.6 Multimodal Constrastive Learning: CLIP(2021) . . . 44

11 04/13/2022 - Massive Models 45
11.1 Scaling Laws . . . 45
11.1.1 Scaling Laws for NLMs(2020) . . . 45
11.1.2 Scaling Laws for autoregressive generative modeling(2020) . . . 45
11.2 Massive Models . . . 46
11.2.1 GPT-3(2020) . . . 46
11.2.2 Gopher(2021) . . . 46
11.2.3 Chinchilla(2022) . . . 46
11.2.4 Megatron-Turing NLG(2022) . . . 46
11.2.5 PaLM(2022) . . . 46
11.3 Applications of Massive Models . . . 47
11.3.1 Few-shot learning via Prompting . . . 47
11.3.2 Medium-shot learning with fine tuning . . . 47
11.3.3 Specializing to Code: Codex and AlphaCode . . . 47
11.4 Limitations of Massive Models . . . 47
11.4.1 Challenge Tasks . . . 47
11.4.2 Potential Harms and Biases . . . 47

12 04/18/2022 - Chelsea Finn: Meta Learning 49
12.1 How does Meta-Learning Work? . . . 49
12.2 Mathematical Formulation of Meta Learning . . . 49
12.3 Meta Learning Problem . . . 49
12.3.1 Other Similar Problems . . . 50
12.3.2 Terminology . . . 50

13 04/20/2022 - Sergey Levine: Deep Reinforcement Learning 51

§1 02/28/2022 - Recurrent Neural Networks

§1.1 Problem Setup

We consider settings in which the features $\mathbf{x}$ represent *sequential data* which may be **variable length**. Some examples include

- Text of different lengths
- Audio recordings/waves
- Video recordings

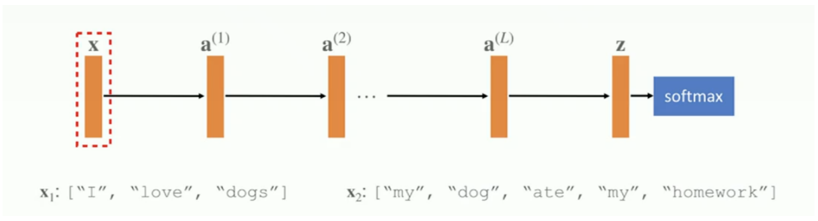
The labels could be scalars y(sentiment analysis, identification), or they could be sequences $\mathbf{y}$(translation, transcription, captioning). There could also be no label at all! This is true in unsupervised learning/generating modeling.

Some models we use include:

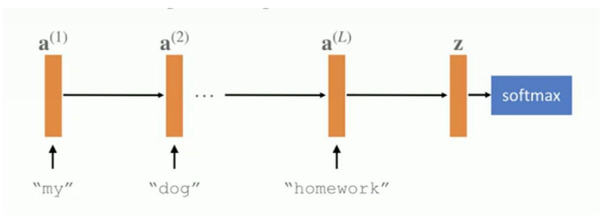
- Markov/n-gram models, hidden Markov models(HMMs)
- Embedding/clustering-based methods
- Convolutions("temporal" convolutions which could be a 1-d convolution of a sequence, or a 3-d convolution over a video).
- Recurrent Neural Networks(RNNs)
- Long short-term memory(LSTM)
- Gated Recurrent Units(GRUs)
- Transformers

§1.2 Dealing with Variable Length Inputs

Before, when dealing with images, we could reasonably assume fixed size inputs. However, with sequential data, it can be assumed that the input lengths will almost always vary.


x
a^(1)
a^(2)
a^(L)
z
softmax
x1: ["I", "love", "dogs"]
x2: ["my", "dog", "ate", "my", "homework"]

To deal with this, we feed one piece of input called a **token** per layer.



Now, the input to layer $\ell + 1$ is $[a^{(\ell)}; x[\ell]]$.

The issues with this approach are the following:

- We need as many layers as the max number of tokens. For very large inputs, we would have extremely deep networks which are difficult to train.
- We have a different $W^{(\ell)}$ and $b^{(\ell)}$ for each layer, which is a massive number of items to optimize.
- The later layers get trained very rarely, so it's hard to generalize to longer sequences at test time.
- Another issue is that $a^{(1)}$ is missing the previous layer output.

§1.3 Recurrent Neural Networks

To solve some of these issues, we use **weight sharing**: use the same parameters in every layer.

$$\text{Before: } a^{(l+1)} = \sigma(W^{(l+1)}[a^{(l)}; x[l]] + b^{(l+1)})$$
$$\text{Now: } a^{(l+1)} = \sigma(W[a^{(l)}; x[l]] + b)$$

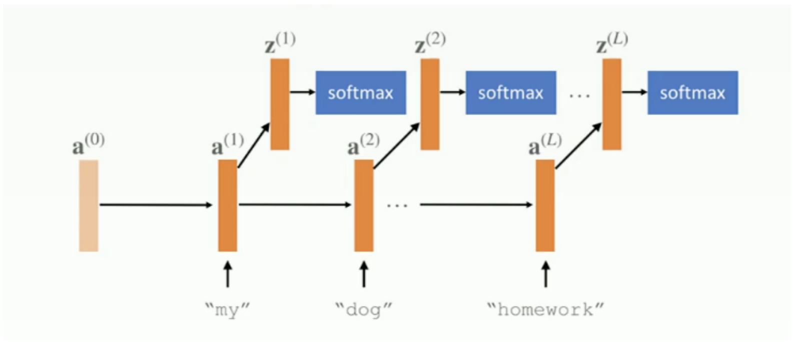
> **Remark 1.1.** In backpropagation, W and b get a gradient signal from every single time they are applied in the network, and we sum them to get the final gradient for W and b.

To address the problem with $a^{(1)}$, initialize some $a^{(0)}$ independently from the input x and feed it into $a^{(1)}$.

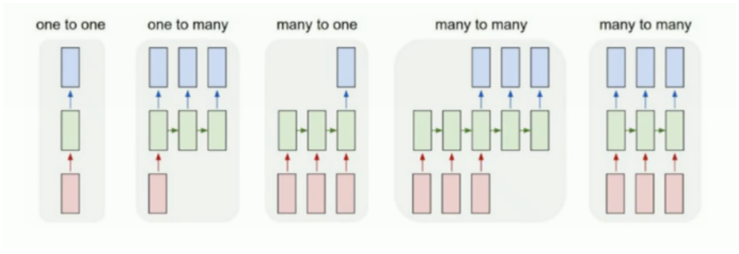
Putting these two ideas together gives us the Recurrent Neural Network(RNN).

> **Remark 1.2.** The values $x[l]$ are encoded from some word2vec algorithm and the end vector is the same length for each word.

The above applies for a sequence input, single output. If we wanted sequence input, sequence output, we could output a scalar at each layer to get a sequence output.



In general, different applications give rise to different ways we use RNNs.

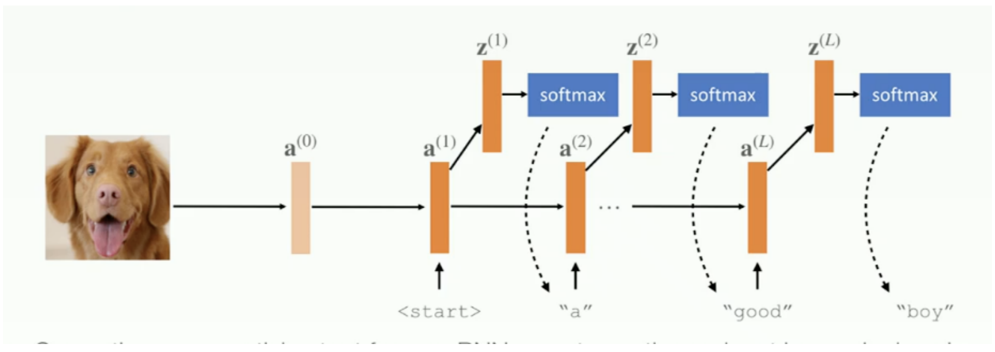


- One-to-one is not an RNN. It is like the ConvNet or Fully-Connected Net we have worked with.

- For translation, we might do something like the 4th diagram, since we need to see the whole sentence before we can translate.
- Image Captioning is single input, sequence output.
- Sentiment Analysis is many to one, sequence input single output.
- Text generation would be many to many like the last diagram.
- The many-to-many can have many different styles depending on the application.

§1.4 Generating Outputs from RNNs

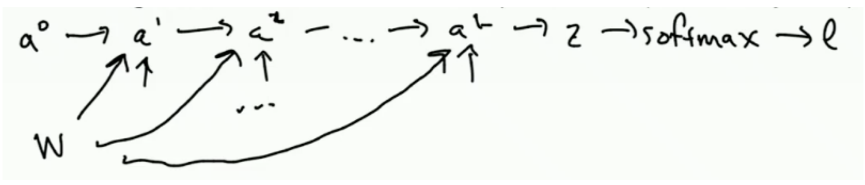
Generating a sequential output is done in an **autoregressive** manner: condition the model on what you have seen before to generate the next thing.



> **Remark 1.3.** In the above, we assume that the RNN is already trained, and then we use it to generate text using the above scheme.

§1.5 Vanishing/Exploding Gradients

What is the gradient of the final loss with respect to W/b?



$a^0 \to a^1 \to a^2 \to \dots \to a^L \to z \to \text{softmax} \to \ell$
W

We have

$$\frac{d\ell}{dW} = \sum_{k=1}^{L} \frac{d\ell}{da^k}\frac{da^k}{dW},$$

$$\frac{d\ell}{da^1} = \frac{d\ell}{da^L}\frac{da^L}{da^{L-1}}\cdots\frac{da^2}{da^1},$$

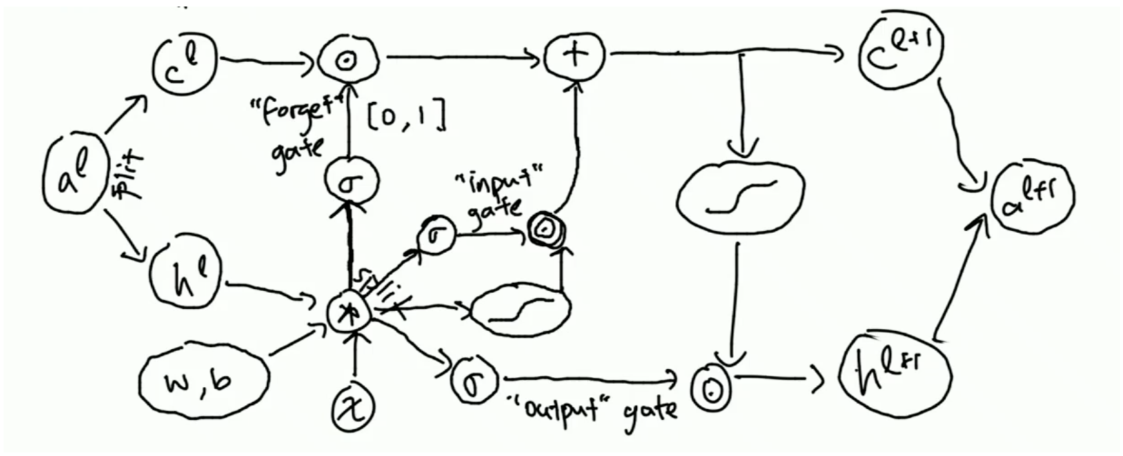
so it is very easy for gradients to vanish or explode.

We can deal with this as follows:

- For exploding gradients, we can just **clip** the gradients - divide by the magnitude of the large gradients to make them smaller.
- Vanishing gradients seem to require clever architecture choices: one idea we can use to address this is a **skip connection**!

- One architecture that does this is the **LSTM**: it is much older than skip-connections but employs the same basic principle that allow better gradient flow by making smarter connection choices.

§2 02/28/2022 - Long Short-Term Memory(LSTM)



1. Start with a state a^ℓ and split into two states, c^ℓ the cell state, and h^ℓ the hidden state. We have $d_c = d_h$, and $d_a = d_c + d_h$, since we exactly split a into two parts.
   - c^ℓ is the part that acts like a skip so we have better gradient flow with long sequences.
   - h^ℓ deals with the nonlinearities which we need for expressivity.
2. For the cell state c^ℓ, we element-wise multiply by scalars f^ℓ in $[0, 1]$(we can do this by passing some number into the sigmoid σ). This is the **forget gate** - when the scalar is close to 0, we forget these dimensions, and when the scalars are close to 1, we retain the information and gradient flow. Then, we add a bias i^ℓ, which is called the **input** to the cell state. This gives us $c^{\ell+1}$. Overall, we have

$$c^{\ell+1} = c^\ell \odot f^\ell + i^\ell.$$

3. For the hidden state h^ℓ, we use an affine transformation using parameters W, b and the input x. This is split into 4 equal parts each with the same dimension as h:
   - The first piece determines f^ℓ, the element sent into the **forget gate**.
   - The next piece is sent to a nonlinearity(doesn't matter which one) and a piece sent to the sigmoid which is elementwise-multiplied to the nonlinearity output. This is called the **input gate**.
   - The last piece is sent to a sigmoid to be used for the output gate.
4. We also take c^{l+1} and use a nonlinearity(doesn't matter which one) and element-wise multiplied by the component from the output gate to determine h^{l+1}.

Remark 2.1. Some elements are important but others are pretty arbitrary. For example, the GRU is very similar but it doesn't have an output gate. However, it still performs pretty well.

§2.1 Bidirectional RNN Models

Often, it can be useful to incorporate information from "the future". For example, when doing speech transcription, if we take an audio waveform and transcribe that into words, it might be useful to see the piece corresponding to the end of the word before the beginning of the word. Similarly, when doing contextual word representations, a word might have different interpretations based on the words around it(”my back hurts”, ”my back door is open”).

To do this, one option is to learn two RNNs! One processes the sequence in the forward direction and one does the reverse. But they are learned jointly to produce a single prediction/representation.

For a while, bidirectional LSTMs were the best model for learning language representations that can be fine tuned for a variety of downstream tasks(ELMo-embeddings from language models). But now we have transformers(BERT).

§3 3/7/2022 - Transformers

§3.1 Setup

We start with features x, which could be sequential data of variable length:

- Text
- Audio Waves
- Videos

We have labels y given by

- Sentiment analysis or translation
- audio transcription or speaker identification
- activity identification or video captioning
- No label! Unsupervised learning or generative modeling

We will use transformers to deal with these.

§3.2 Why Transformers

These have been massively influential in the past 4-5 years. They have been outcompeting other deep architectures, such as RNNs in language modeling and ConvNets in vision tasks.

Remark 3.1. This is pretty insane since we have such strong inductive biases for ConvNets.

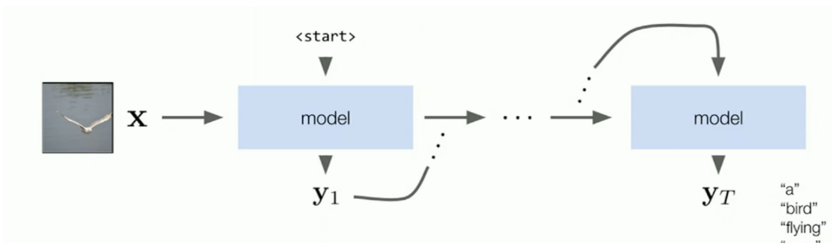
There are other modern models using *attention*, while they are not really transformers.

They are the backbone of BERT and GPT and they perform language tasks at a level which we didn't think was possible a few years ago. For this reason, people dub these as "foundation models".

§3.3 Attention

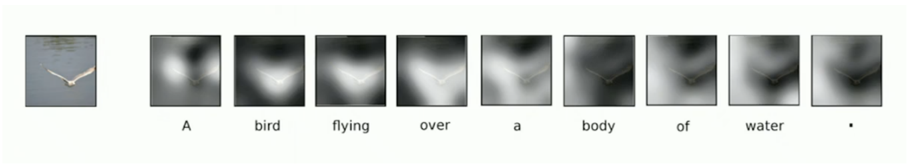
Originally formulated for tasks with sequential outputs.

- Translating between languages, captioning an image
- The features may not be sequential: for example, we could have an image, pass it through a model, and output a sequential caption vector.



We will generate the output one step at a time. However, we need to know what we have generated so far. We could handle this via *autoregressive generation* from an RNN, where we feed model output into the next input until we have generated an entire sequence of tokens.

Now, we motivate attention. For example with the caption, "A bird flying over a body of water .", we maybe want the model to focus on the parts of the image focusing on the recently generated token:



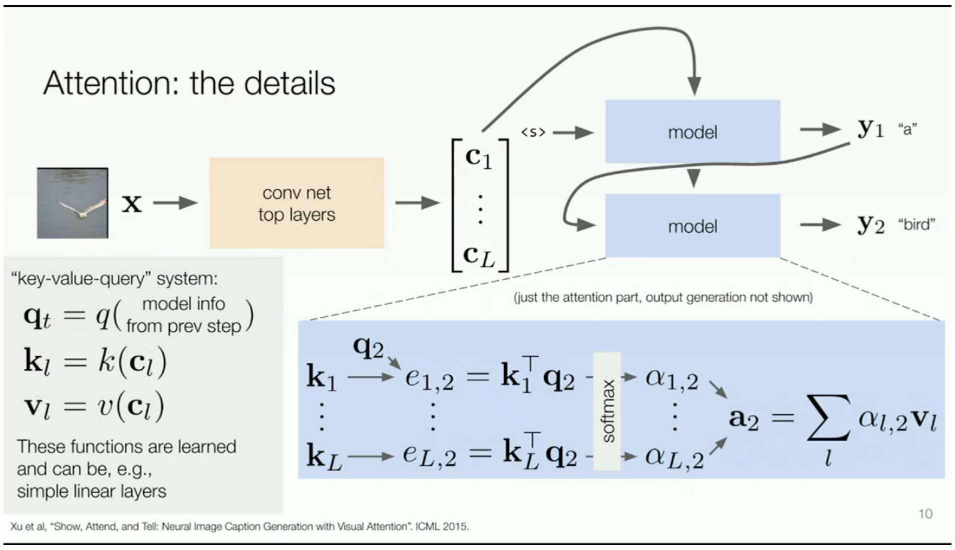
§3.3.1 What to attend over?

- If the features are words, we can attend over them directly.
- If the features are pixels, we can pass the image through the top layers of a ConvNet(usually pretrained on image net). This gives us an intermediate representation of an image, and these low dimensional representations give some understanding of high level features.

> **Remark 3.2.** Why did they not train the layers end-to-end? Possibly for optimization reasons, compute difficulties? But if this was done today, would probably be end-to-end.

§3.3.2 Details of Attention

After passing the image through the ConvNet, we have L context vectors from hopefully corresponding to different semantic elements of the image. Then, we pass these into the model to generate an output token. Then, we can pass this output into the next model in an auto-regressive fashion.

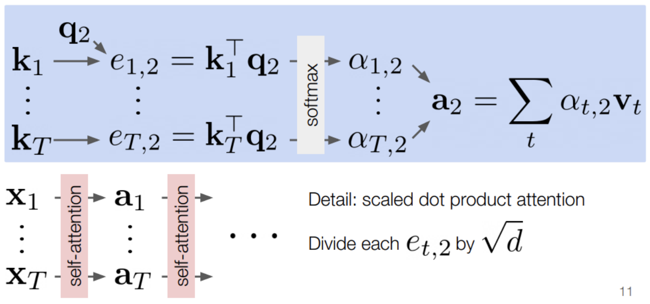

Attention: the details
$\mathbf{x}$
conv net top layers
$\mathbf{c}_1$
$\mathbf{c}_L$
<s>
model
model
$\mathbf{y}_1$ "a"
$\mathbf{y}_2$ "bird"
"key-value-query" system:
$\mathbf{q}_t = q(\text{model info from prev step})$
$\mathbf{k}_l = k(\mathbf{c}_l)$
$\mathbf{v}_l = v(\mathbf{c}_l)$
These functions are learned and can be, e.g., simple linear layers
(just the attention part, output generation not shown)
$\mathbf{q}_2$
$\mathbf{k}_1 \to e_{1,2} = \mathbf{k}_1^\top \mathbf{q}_2$
$\mathbf{k}_L \to e_{L,2} = \mathbf{k}_L^\top \mathbf{q}_2$
softmax
$\alpha_{1,2}$
$\alpha_{L,2}$
$\mathbf{a}_2 = \sum_l \alpha_{l,2} \mathbf{v}_l$
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention". ICML 2015.
10

> **Remark 3.3.** Note that everything here is differentiable, so we can backprop and learn k, q, v through auto-diff.

§3.3.3 Self-Attention

The goal of self-attention is to handle sequential features as the input. We can think of it as a layer that processes the entire sequence which produces intermediate activations $a_1 \to a_t$. Now, we have

$$q_t = q(x_t), k_t = k(x_t), v_t = v(x_t).$$


$\mathbf{q}_2$
$\mathbf{k}_1 \to e_{1,2} = \mathbf{k}_1^\top \mathbf{q}_2$
$\mathbf{k}_T \to e_{T,2} = \mathbf{k}_T^\top \mathbf{q}_2$
softmax
$\alpha_{1,2}$
$\alpha_{T,2}$
$\mathbf{a}_2 = \sum_t \alpha_{t,2} \mathbf{v}_t$
$\mathbf{x}_1$
$\mathbf{x}_T$
self-attention
$\mathbf{a}_1$
$\mathbf{a}_T$
self-attention
Detail: scaled dot product attention
Divide each $e_{t,2}$ by $\sqrt{d}$
11

Remark 3.4. One problem that we have is that self-attention is completely linear function of the inputs. So we need to interweave these with nonlinearities.

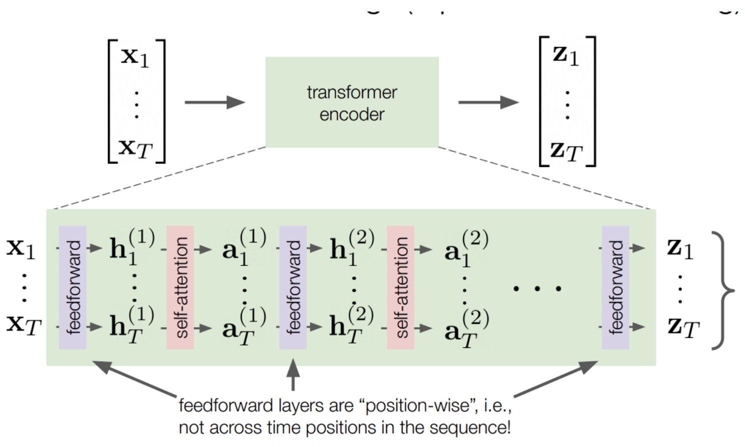
Remark 3.5. We also have scaled dot product attention, where we divide each $e_{t,2}$ by $\sqrt{d}$. This is similar to weight-initialization.

Remark 3.6. It seems like the ordering of the elements do not matter in the case of self-attention. This is true and is handled by transformers.

§3.4 Transformers

§3.4.1 Transformer Encoder

We start with a vector $X \to Z$, through a transformer encoder.

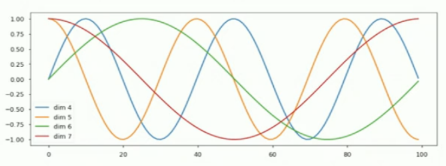


Remark 3.7. Note that for the feedforward layers, $h_i^{(1)}$ depends only on x_i and the same is true for the later layers.

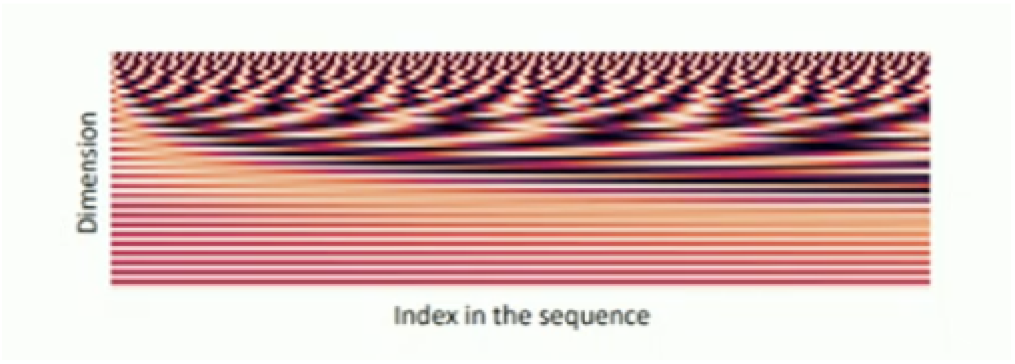
§3.4.2 Positional Encoding

After the feedforward layer, a positional encoding is added to each $h_t^{(1)}$. This is some information about what time-step each input corresponds to. Without this, the model cannot distinguish between different permutations of the input sequence. One choice is to add the following vector to h_t:

$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \cdots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$



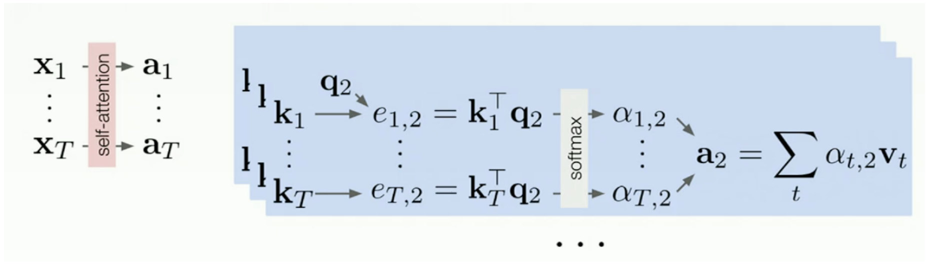
Are there alternatives to this? What if we just concatenated the time step after the first feed-forward layer? This appears worse because we care more about the relative positioning:



What if we learned the positional encodings? This is used sometimes and sometimes better due to expressivity. However, this has downsides: can't generalize to longer sequences, etc.

§3.4.3 Multi-Head Attention

Starting with self attention, we have a mechinism going from $\curvearrowleft \to \partial$, where $\partial_2 = \sum_t \alpha_{t,2} \gtrsim_t$. This works well if we have one particular key that we should pay attention to. But what if we have multiple features that we want to pay attention to? We can do this by having "multiple heads", which are independent attention blocks that pay attention to different values.

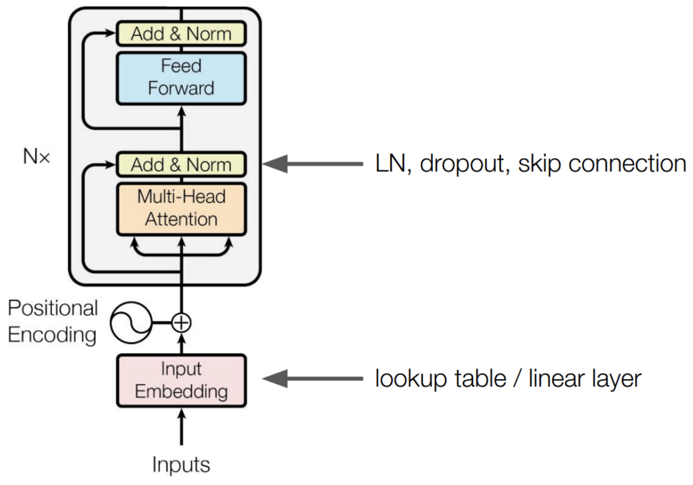

$\mathbf{x}_1$ → $\mathbf{a}_1$
$\mathbf{x}_T$ → $\mathbf{a}_T$
self-attention
$\mathbf{q}_2$
$\mathbf{k}_1 \to e_{1,2} = \mathbf{k}_1^\top \mathbf{q}_2$
$\mathbf{k}_T \to e_{T,2} = \mathbf{k}_T^\top \mathbf{q}_2$
softmax
$\alpha_{1,2}$
$\alpha_{T,2}$
$\mathbf{a}_2 = \sum_t \alpha_{t,2} \mathbf{v}_t$
$\cdots$

> **Remark 3.8.** Note that each of the key, query, and value functions for each head have their own learned parameters.

Remark 3.9. The final a_2 and every a_t is given by concatenating the outputs from each head and potentially feeding this through another linear layer.

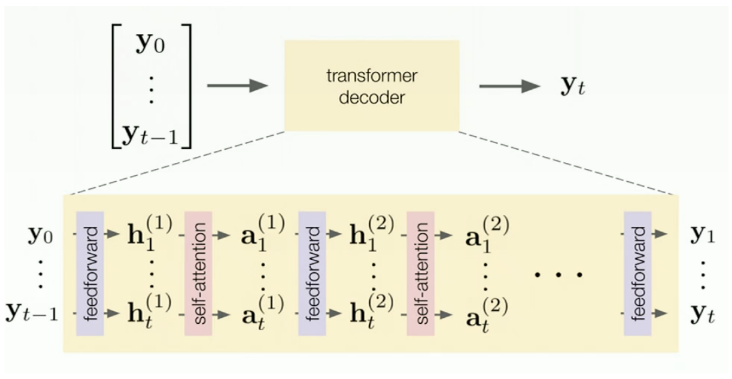
Remark 3.10. In order to not make this not scale linearly in complexity with the number of heads, we scale the dimension down proportionally to the number of heads, e.g., if 512 dimensions for 1 head, then 64 dimensions for 8 heads.

The final picture for the encoder is given by



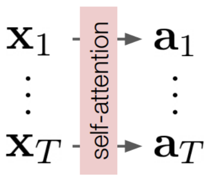
§3.4.4 Transformer Decoder

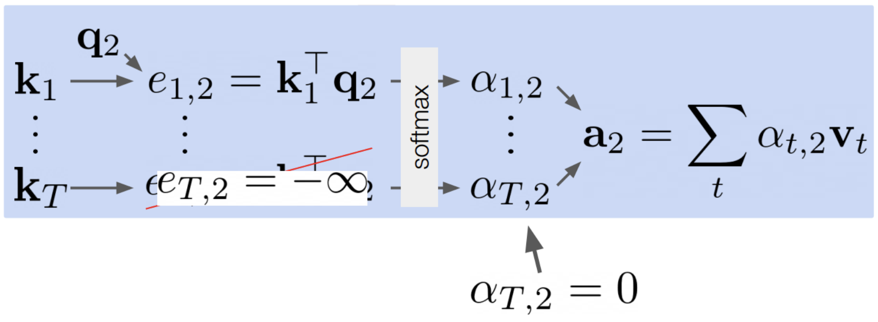
As before, we feed the outputs of the decoder into the Transformer decoder as input. However, at training time, we feed ground truth sentences in the decoder.



§3.4.5 Masked Attention

During training, we are passing entire sequences instead of passing in partial sequences since this is more computationally efficient. However, the model cannot be looking at the future to generate earlier elements from later ones. To do this, we ”mask” the keys and values corresponding to future timesteps.


$\mathbf{x}_1$
$\mathbf{x}_T$
self-attention
$\mathbf{a}_1$
$\mathbf{a}_T$

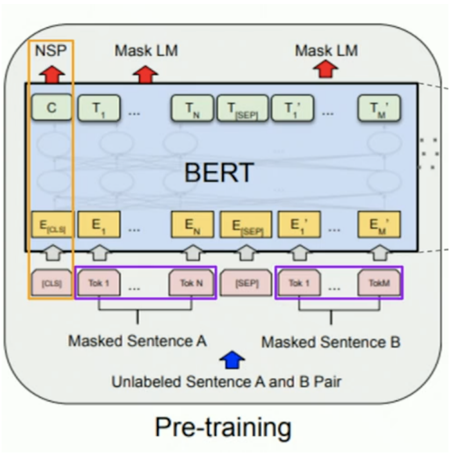

$\mathbf{q}_2$
$\mathbf{k}_1$
$e_{1,2} = \mathbf{k}_1^\top \mathbf{q}_2$
$\mathbf{k}_T$
$e_{T,2} = -\infty$
softmax
$\alpha_{1,2}$
$\alpha_{T,2}$
$\mathbf{a}_2 = \sum_t \alpha_{t,2}\mathbf{v}_t$
$\alpha_{T,2} = 0$

§4 03/09/2022 - Transformer Models

The original transformer is a sequence-to-sequence model for translation. These typically follow an encoder-decoder architecture and the transformer does well at handling these tasks. We will go into sequence-to-sequence models more in the future.

§4.1 Encoder: BERT

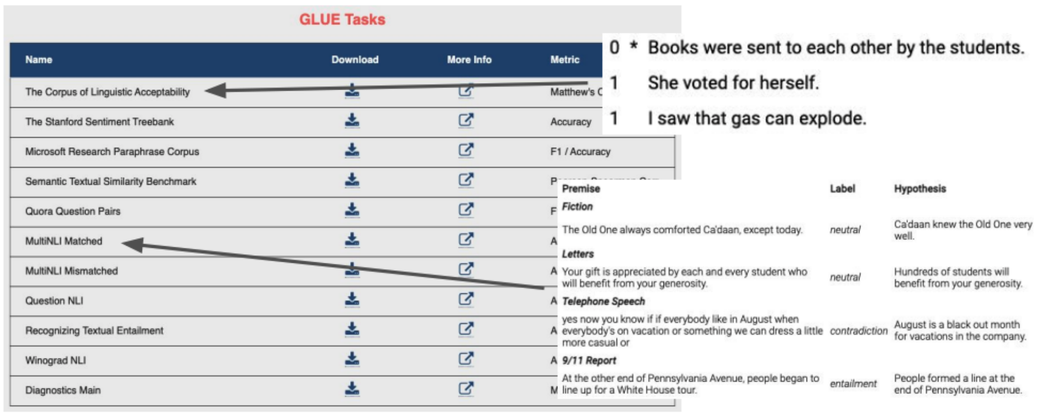
During training, input tokens(words) are randomly masked with $p = 0.15$. The model must predict these tokens before they were masked. The hypothesis is that it allows BERT to learn word level representations.



Then, pairs of sentences are passed in and the model must predict which sentence follows the other(done with the [CLS] token which is mapped to NSP), called the entailment prediction. The goal of this is for BERT to learn sentence level representations of the data.

§4.1.1 The GLUE Benchmark

It is comprised of several different language tasks.



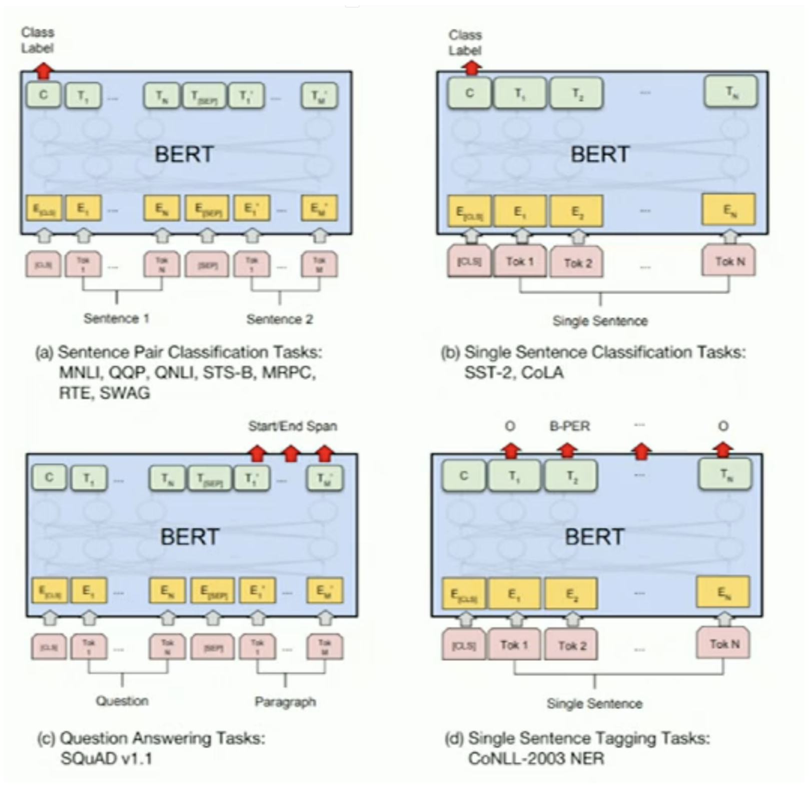
GLUE Tasks

Name	Download	More Info	Metric
The Corpus of Linguistic Acceptability	⤓	↗	Matthew's C
The Stanford Sentiment Treebank	⤓	↗	Accuracy
Microsoft Research Paraphrase Corpus	⤓	↗	F1 / Accuracy
Semantic Textual Similarity Benchmark	⤓	↗	P
Quora Question Pairs	⤓	↗	F
MultiNLI Matched	⤓	↗	A
MultiNLI Mismatched	⤓	↗	A
Question NLI	⤓	↗	A
Recognizing Textual Entailment	⤓	↗	A
Winograd NLI	⤓	↗	A
Diagnostics Main	⤓	↗	M

0	* Books were sent to each other by the students.
1	She voted for herself.
1	I saw that gas can explode.

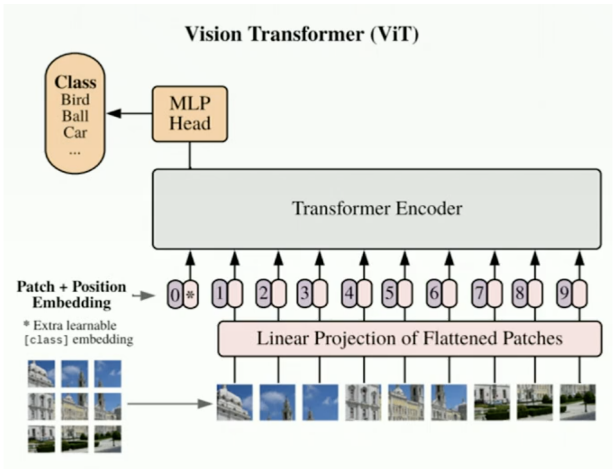
Premise	Label	Hypothesis
Fiction		
The Old One always comforted Ca'daan, except today.	*neutral*	Ca'daan knew the Old One very well.
Letters		
Your gift is appreciated by each and every student who will benefit from your generosity.	*neutral*	Hundreds of students will benefit from your generosity.
Telephone Speech		
yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or	*contradiction*	August is a black out month for vacations in the company.
9/11 Report		
At the other end of Pennsylvania Avenue, people began to line up for a White House tour.	*entailment*	People formed a line at the end of Pennsylvania Avenue.

BERT does very well in these tasks upon Fine-Tuning. After we use the pre-trained BERT model, we can fine tune in whatever way is dictated by the task to solve that particular task. Since BERT architecture relies on tokenized inputs, all the different tasks can be solved using the fine-tuned BERT model.


Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

§4.2 Encoder: Vision Transformer

The input image is divided up into 16 × 16 patches and are treated as a sequence and pass it into a transformer to see what it could do.

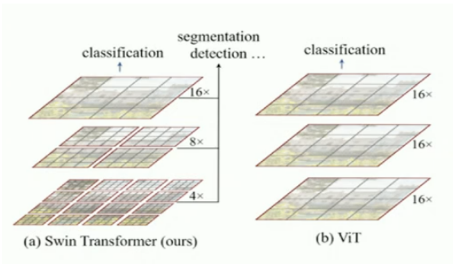

Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
Linear Projection of Flattened Patches

- During training, the image is split into patches and projected down with a learned network(the first feed forward layer).
- A ConvNet instead of a Linear Projection for the first layer also works well.
- These are combined with the positional encodings and fed to the encoder(these are usually learned for Vision Transformers)
- Then, a net on the first special position token is passed through a fully connected network which outputs logits, and we use cross-entropy loss and backpropagate the whole thing so that we can perform image classification.

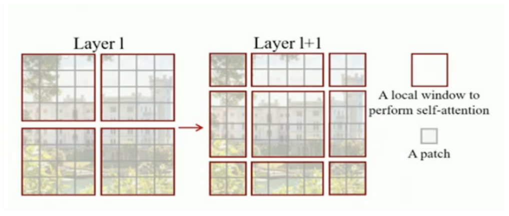
Remark 4.1. Why do we need positional encodings? Even though we want to mainly focus on local information, as we stack them deeper and deeper we are able to see non-local information, since the receptive field grows to the point where we can see things that are far apart. We wouldn't be able to understand the ordering of the patches if we didn't have positional embeddings.

§4.3 Swin Transformer

Swin stands for Shifted Window. The original vision transformer considered 16×16 patches of the image and did self-attention on each of the 16×16 patches of the image. But 16×16 is a fairly low resolution and we sometimes want more fine grained patches for certain tasks. Note that the sequence-length is quadratic in the size of the patch, so if we divide 16×16 into 8×8, we halfed the height and width but quadrupled the number of patches.



Swin does smaller patches, but to fix the quadratic issue, they perform self-attention over a local region of the image that we have. But this might not be able to address global dependencies. This is handled through the shifted window:



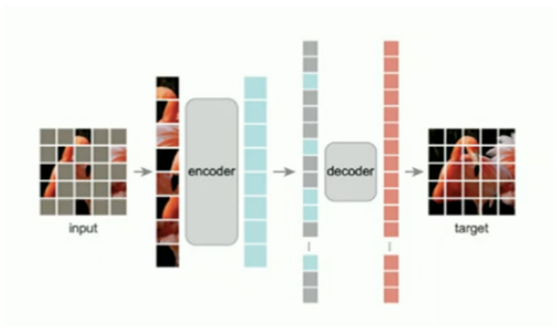
In the next layer, we shift over the window so that we can aggregate information over the whole image by stacking layers on top of each other.

§4.4 Unsupervised ViT

The main idea behind this is **distillation**, where you have two ViTs and they both have no supervision, but if you get clever and train one to match the other(with certain inductive biases), you can get good an unsupervised training of a model that can be used later.

§4.4.1 Masked Autoencoders

This is the similar to what BERT is doing. We have an input image and we mask some amount of the image(usually around 75% of the image). Using the unmasked patches, we pass them through a transformer encoder, map it back to the patches that weren't masked and mask tokens, and pass this to a decoder(not actually a transformer decoder).


input
encoder
decoder
target

§5 03/14/2022 - Sequence-to-Sequence Models

We are converting input sequences into output sequences. The canonical example is **machine translation**, where we translate sentence A into sentence B. This is the main example that we focus on today.

§5.1 Seq2seq Models

We already discused several models for handling sequential data(RNNs, LSTMs, Transformers). We also discussed cases where the input is not a sequence but the output is. Now, we fill in the blanks for sequence input and sequence output tasks.

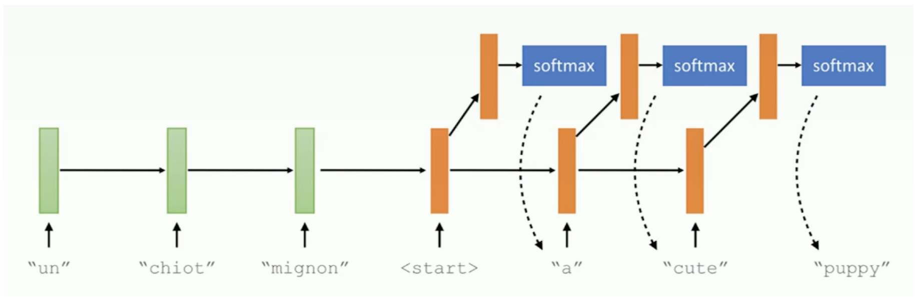
- Generally, we read in the entire input sequence before trying to generate the output sequence. This is not always necessary, for example, we could have transformer-type architectures that translate sequences into new sequences with masking.
- We focus on many-to-many where we read in the entire input first.

This is the canonical way of solving this problem and all seq2seq models follow the encoder-decoder architecture.

- The encoder reads in the input sequence and encodes it into a representation.
- The decoder conditions on this representation in order to decode the output.
- Historically, both of these were LSTMs with separate parameters.
- Nowadays, they are both usually transformers.

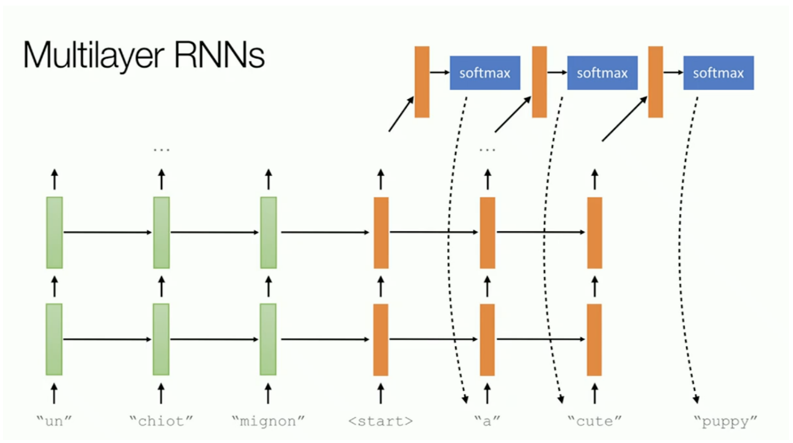
§5.1.1 RNN(LSTM) seq2seq, basic

We have a sentence ”un chiot mignon”. We feed a token for each word and move to each word in the sentence. Then, we pass it into a new decoder RNN by first giving it a ¡start¿ token and conditioned on the representation, we output probabilities of what the first word might be. Then, we pass this back into the decoder to keep generating words in an auto-regressive fashion. We keep going until we see an ¡end¿ token.



§5.1.2 Multilayer RNNs

A simple modification is to add more layers. All of our layers pass into hidden states which correspond to better and better representations.

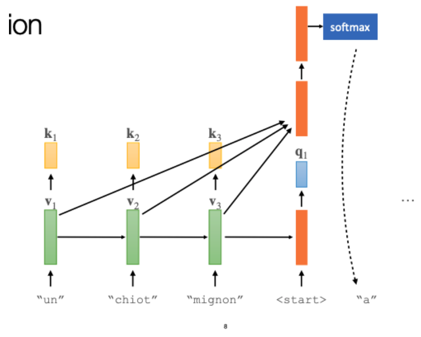


What are some issues with the above models?

- We have the *bottleneck problem.* All information about the source sequence has to pass through a direct connection between the encoder and decoder, so the connection between the encoder and decoder is very crucial.
- This makes it difficult - for example, if the last word we want to decode corresponds to the first word. The issue is that we saw a token whose information flow has to survive through all the different hidden states.

§5.1.3 Attention

Now, we use attention to connect the encoder and decoder is a more direct way without relying on a single connection. We make keys from each hidden state, and we treat the hidden states as values directly. Then, we generate queries from the hidden states of the decoder and comparing these with the keys and values with a dot product giving us a similarity score. Then, taking a softmax and taking a weighted sum, with all the values with the weights computed using the dot products, we obtain the representation that we use to generate the output at the particular timestep.



- With attention, the direct connection between encoder and decoder becomes less important.
- We can make the encoder a bidirectional RNN and attend over the outputs. Attention allows the decoder to look at all the hidden states representations that the encoder is producing. So we might get a better understanding of what a token means at a timestep by looking at the past and the future(one LSTM running forwards and backwards and concatenating representations). The decoder still needs to be unidirectional since we generate in an auto-regressive manner.

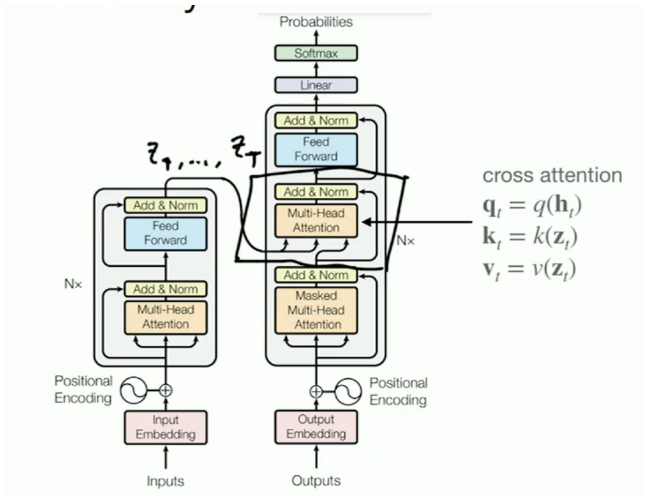
- The value function is the identity function in this case. This is slightly different from before, where we learned the value function. We also could have used a learned value function.
- We could also make the key and query identity functions. In that case, we are comparing the hidden states directly and based on those similarities, we generate the element to be used for the next timestep.

§5.2 Seq2Seq Transformers

Given an input sequence X, we learn a representation Z from the transformer encoder. Then, we have a separate decoder learning the output sequence which generates a sequence in an auto-regressive manner.

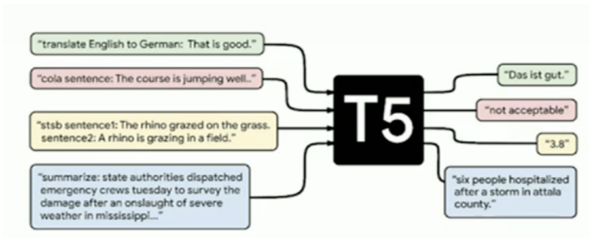
§5.2.1 Cross Attention Layers

In the cross attention layer, we use the key and value from the transformer encoder and the queries from the decoder,masked self-attention.



§5.3 Recent seq2seq: T5

- The text-to-text transfer transformer(T5) solves many different NLP tasks in a unified text input, text output framework.
- We pass in an input sequence prepended with the actual task and these are uniformly handled and solved using T5.
- This is trained on C4 and achieves a number of competitive and state-of-the-art results.


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

§6 03/16/2022 - John DeNero: Neural Machine Translation

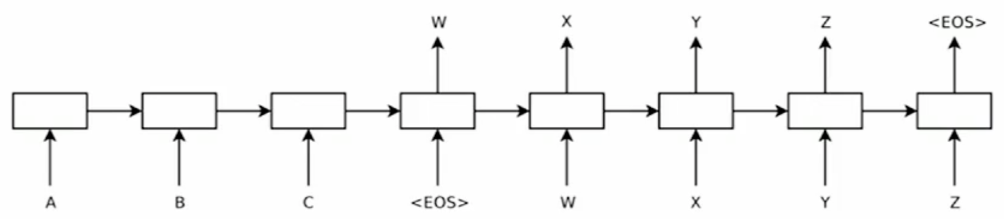
- Text as input and text as output
- Input and output have roughly the same information content
- The output space is very large, but the output is more predictable than a language modeling task.

One issue is that there is variety in Human-Generated Translations. This could be because of grammatical errors, different orderings of words, etc. So we have issues even with the labels of our dataset.

§6.1 Conditional Sequence Generation

We estimate $P(e|f)$ from a sequence model $P(f, e)$. To do this, we run an RNN over the whole sentence, which first computes $P(f)$, then computes $P(e, f)$.

"Sequence to sequence" learning (Sutskever et al., 2014)

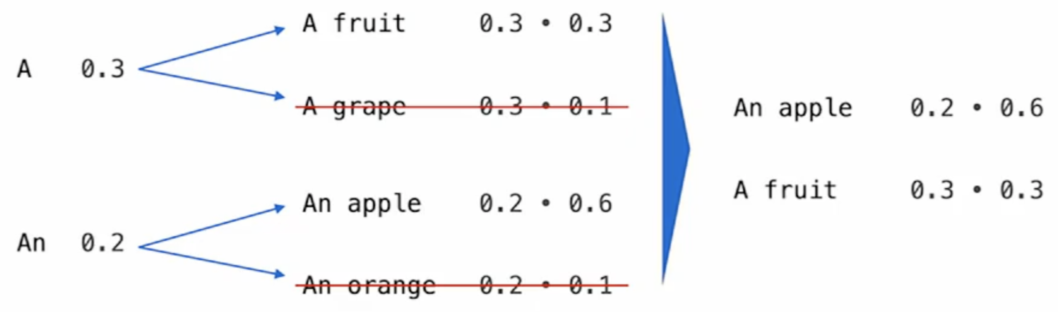


(Sutskever et al., 2014) Sequence to sequence learning with neural networks.

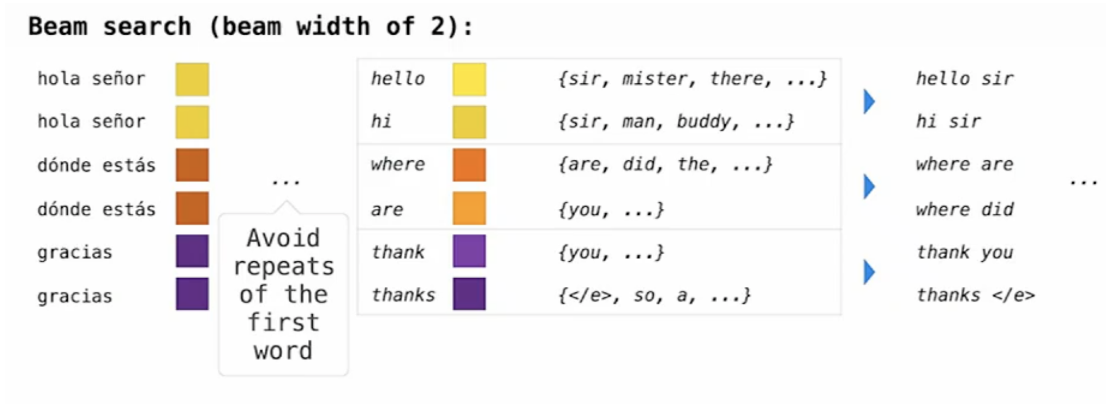
We can do better by using an encoder-decoder model: different parameters or architectures for encoding f and predicting e.

§6.2 Search Strategies

- For each target position, each word is scored(alternatively, you could use a restricted list, but the quality degrades), and you choose the highest score.
- Greedy decoding: extend a single hypothesis with the next word that has the highest probability. This will tend to have repetitions because it is not uncommon for a word to appear more than once causing a loop.
- We want sequences with the highest probability, not just words. To do this, we use Beam Search: Fixed number of partial translations(usually two), then prune.



In actually implementing Beam Search for Batch Decoding, we have the following:



- We do something to make sure we don't pick the same word for both branches.
- Then, we consider the scores for the top vocabulary options and choose the top two.

Generally, NMT models prefer translations that are two short.

$$s(e) = \sum_{i=1}^{m} \log P(e_i | e_{1:i}, f).$$

- The model is made up of a product of things less than 1, so the longer the sequence, the smaller it tends to be.
- Another explanation has to do with how they are trained, we don't just maximize the probability of unseen data, but we hold out the probability mass with unseen things like dropout, etc. It forces the probability terms to be smaller than they should be, and multiplying things that are shorter than they should be results in smaller sentences.
- "For more than 50% of sentences, the model in fact assigns its global best score to the empty translation" (Stahlberg and Byrne, 2019)

To deal with these issues, we use alternative scorings on the beam:

Length normalization: $s(e)/m$

Google's correction (2016): $\dfrac{s(e)}{\frac{(5+m)^\alpha}{(5+1)^\alpha}}$

Word reward: $s(e) + \gamma m$

§6.3 Training Loss

We do something called **teacher forcing**: during training, only use the predictions of the model for the loss, not for the input.

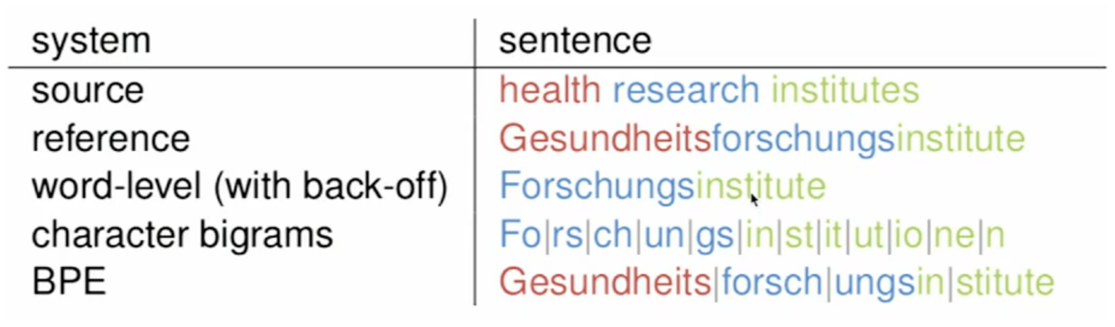
One hack is label smoothing: update toward in a distribution in which 0.9 is assigned to an observed word and 0.1 probability is decided uniformly among the other words.

§6.4 Subwords

The sequence of symbols that are embedded should be common enough that an embedding can be estimated robustly for each, and all symbols have been observed during training:

- Solution 1: Symbols are words with rare words replaced by UNK.
  - Replacing UNK in the output is a new problem(like alignment).
  - UNK in the input loses all information that might have been relevant from the rare input word(tense, length, POS)
- Solution 2: Symbols are subwords
  - Byte-Pair encoding: splits a word into subwords
  - Other techniques that find common subwords aren't reliably much better.
  - Training on many sampled subword decompositions can improve out-of-domain translations.
  - If there is only one way of splitting up a word, we might miss other ways of splitting a word that is better for translation.

§6.4.1 Byte-Pair Encoding



system	sentence
source	health research institutes
reference	Gesundheitsforschungsinstitute
word-level (with back-off)	Forschungsinstitute
character bigrams	Fo|rs|ch|un|gs|in|st|it|ut|io|ne|n
BPE	Gesundheits|forsch|ungsin|stitute

You start by splitting each word into characters. Then you find the most frequent pairs of characters and turn that into a new character. Repeat.

§6.5 Back Translations

One trick is to synthesize an en-de parallel corpus by using a de-en system to translation monolingual de sentences. This can help if the de sentences are already in a en-de parallel corpus!

§7 03/28/2022 - Distribution Shift

Today, we will be talking about **distribution shift**: when the test data comes from a different distribution than the training data. The real world is full of distribution shift - the world changes all the time and it is common that the training data becomes outdated or is not representative of the full picture. Usually, it is benign, but it can be harmful sometimes.

This can cause accuracy to go down, but it can cause other degradations:

- worse calibration
- worse fairness: worse for particular users

Most of these examples come from computer vision(image classification in particular).

§7.1 Risk

Recall, we defined **risk** $R(\theta) = \mathbb{E}[\ell(\theta; X, Y)]$. We also defined empiral risk $\hat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i)$, a monte carlo estimate of the true risk.

- Supervised learning is refered to as empirical risk minimization(ERM).
- Why and when does it make sense to use this as an objective?

§7.1.1 The ERM Assumption

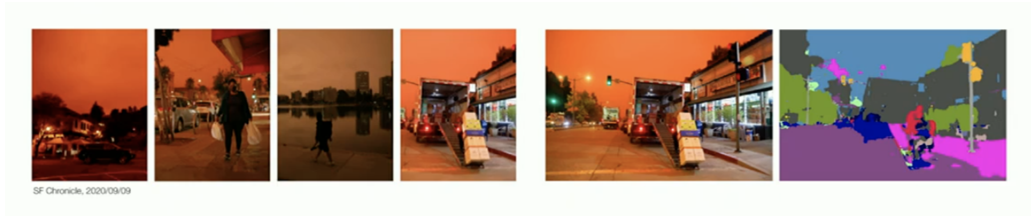
ERM is based on the assumption that the test data distribution is the same as the training data distribution. Under this assumption(and good validation, regularization), we can derive *generalization bounds* of how well we expect models to generalize to new data, and even for deep neural networks!

This assumption is used by almost all supervised learning methods, and it is referred to as "the big lie of machine learning" by Prof. Zoubin Ghahramani. This is because it is very rarely the case that training and test data match perfectly. There are many cases where it "almost" holds, but there are scenerios that do not generalize at all.

§7.2 Distribution Shift in the Real World

Distribution shift in the real world is the norm, not the exception. But we need to consider whether or not it will be a problem in our application.

- If it is in a continuous deployment setting, the model will likely encounter future scenarios not represented in training data.



- If the model interacts with end users, some users will be atypical and challenge the model in strange ways.

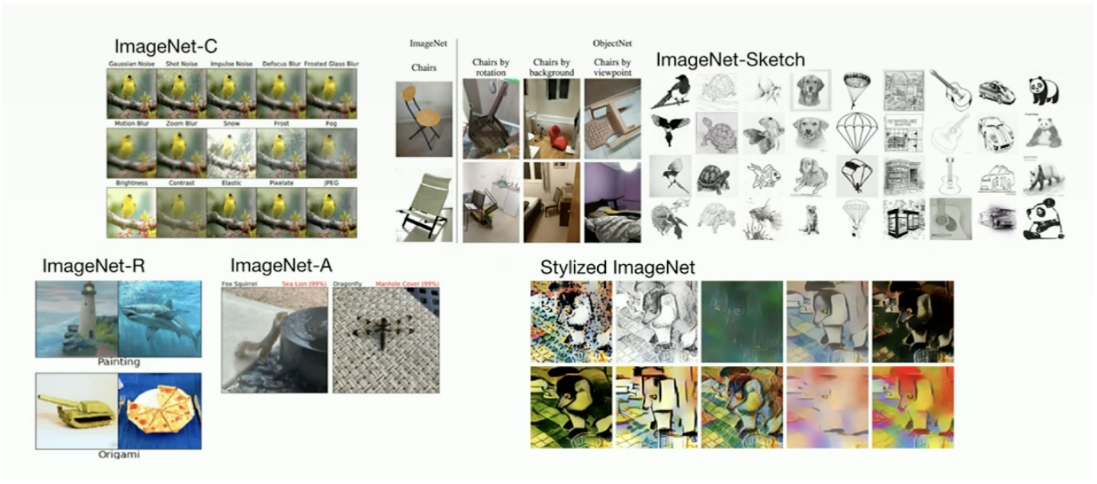
§7.3 Distribution Shift Benchmarks

We have multiple objectives:

- Benchmarks that are diverse and representitive or real applications. If we had synthetic benchmarks then we run into the issue where we design methods that overfit to the synthetic benchmarks but don't perform well in the real world.
- Benchmarks that are easy to use and evaluate on.

§7.3.1 ImageNet Challenge Test Sets

This is a popular way to test model **robustness** to different distribution shifts. The test sets are designed to stress test models by simulating extreme or highly unusual events(that are still plausible).

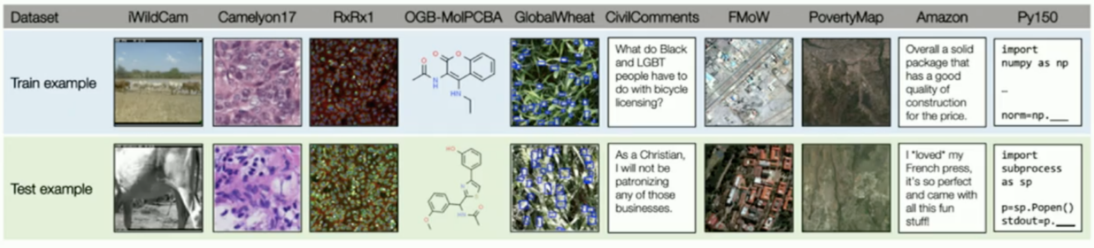


They contain the same classes as ImageNet so any model trained on ImageNet can be evaluated on these test sets.

§7.3.2 WILDS Benchmark

- Having easy to use and standardized challenge test sets are important but they might not be the full picture.
- We are limiting ourselves to just image net, which does not cover the entire space.

WILDS aims to curate a suite of problems that faithfully represent how distribution shift manifests itself in real world applications.


Dataset
iWildCam
Camelyon17
RxRx1
OGB-MolPCBA
GlobalWheat
CivilComments
FMoW
PovertyMap
Amazon
Py150
Train example
Test example

What do Black and LGBT people have to do with bicycle licensing?

As a Christian, I will not be patronizing any of those businesses.

Overall a solid package that has a good quality of construction for the price.

I "loved" my French press, it's so perfect and came with all this fun stuff!

```
import
numpy as np

_

norm=np.___
```

```
import
subprocess
as sp

p=sp.Popen()
stdout=p.___
```

§7.4 NLP: ANLI Dataset

- Natural language inference is the task of determining if a premise sentence and hypothesis sentence are related through contradiction, neutrality, or entailment. This is part of the GLUE benchmark.
- The **adversarial natural language inference (ANLI)** dataset consists of crowd-sourced hypothesis written to fool state-of-the-art pre-trained models.
- To construct the dataset, an annotator is asked to write a hypotehsis given a premise and a condition.
  - If the model corrects predicts the condition, the annotator tries again.
  - If incorrect, hypothesis is verified by other annotators.

§7.5 Robustification

How do we make models more robust? For WILDS, the training dataset comes with more info that we can leverage. But for image set, we don't get additional information for training. Some general techniques that have proven useful are:

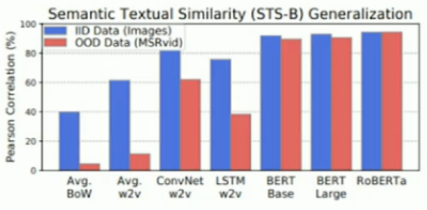
- Train larger models on larger, more diverse datasets
- Using heavy data augmentation and alternate/additional training objectives.

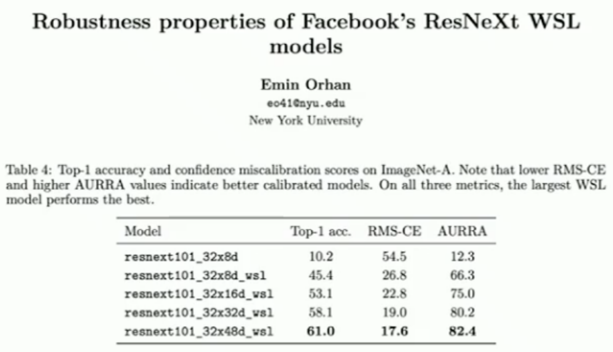
§7.5.1 Train on Larger Datasets

Does this improve "robustness"? Now, the test dataset is less out of distribution(OOD), so is the model more robust or have we just captured a training sample that better covers the test data.

Pretrained Transformers Improve Out-of-Distribution Robustness

Dan Hendrycks[1*] **Xiaoyuan Liu**[1,2*] **Eric Wallace**[1]
Adam Dziedzic[3] **Rishabh Krishnan**[1] **Dawn Song**[1]
[1]UC Berkeley [2]Shanghai Jiao Tong University [3]University of Chicago
{hendrycks,ericwallace,dawnsong}@berkeley.edu





Robustness properties of Facebook's ResNeXt WSL models

Emin Orhan
eo41@nyu.edu
New York University

Table 4: Top-1 accuracy and confidence miscalibration scores on ImageNet-A. Note that lower RMS-CE and higher AURRA values indicate better calibrated models. On all three metrics, the largest WSL model performs the best.

Model	Top-1 acc.	RMS-CE	AURRA
`resnext101_32x8d`	10.2	54.5	12.3
`resnext101_32x8d_wsl`	45.4	26.8	66.3
`resnext101_32x16d_wsl`	53.1	22.8	75.0
`resnext101_32x32d_wsl`	58.1	19.0	80.2
`resnext101_32x48d_wsl`	**61.0**	**17.6**	**82.4**

§7.5.2 Data Augmentations

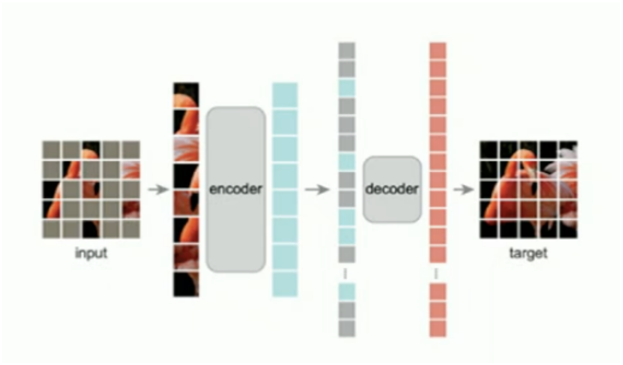
Some techniques are:

- Mixup: produce element-wise convex combinations of datapoints and improve corruption robustness.
- AutoAugment: learns complex augmentation strategies from basic ones by training tens of thousands of deep neural networks. This is a very expensive strategy but not a bad strategy.

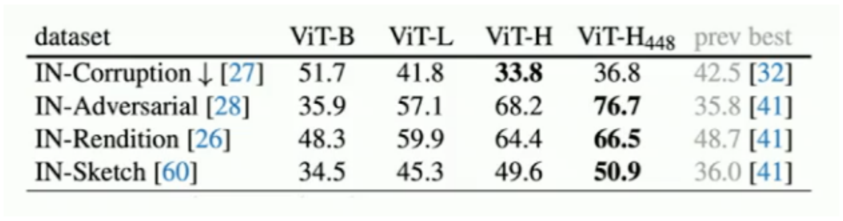
- AugMix: mixes together random augmentations using ones from AutoAugment
- PixMix: Mixes in images from a dataset(such as fractals) and mixes it together resulting in consistently good performance across several metrics.

§7.5.3 Masked Autoencoders

The current state of the art numbers are trained with ViT models with a pretrained masked autoencoding objective.



Then, we find tune the encoder on ImageNet. Supervised learning on the original ImageNet training set after this pretraining leads to the best results amongst models that do not get additional data.



dataset	ViT-B	ViT-L	ViT-H	ViT-H$_{448}$	prev best
IN-Corruption ↓ [27]	51.7	41.8	**33.8**	36.8	42.5 [32]
IN-Adversarial [28]	35.9	57.1	68.2	**76.7**	35.8 [41]
IN-Rendition [26]	48.3	59.9	64.4	**66.5**	48.7 [41]
IN-Sketch [60]	34.5	45.3	49.6	**50.9**	36.0 [41]

§7.6 Anomaly Detection

- If the system can detect an anomaly, then it might be able to enter some conservative mode or a fail-safe to avoid catastrophic errors.
- Detect malicious use of ML systems(hackers)
- Detect potential dangers, like dangerous novel microorganisms.

§7.6.1 Implementation

- Assign an anomaly score to every input x. We only do this for inputs because we want to be able to do this at test time as well. Higher scores are more anomalous.
- What if we learned a model of $p(x)$(a generative model) and treat x as anomalous if it has low $p(x)$?
  - This does not currently work well since learning models of $p(x)$ is very difficult.
  - It is also not sure if models using $p(x)$ or $\log p(x)$ is actually that well-framed of an idea to detect anomalies.
  - Modern deep generative models still do poorly at anomaly detection using the scheme for complex input spaces.

§7.6.2 A Simple Baseline

- Doesn't involve training a generative model
- Use the model's confidence $\max_k p_\theta(y = k|x)$ to detect anomalies.
- Specifically, we use the score $-\max_k p_\theta(y = k|x)$. We may also use $-\max_k z_k$, the negative of the max logit for some situations.
- The simple baselines work well across CV, NLP, speech recognition, classifications, but it cannot detect adversarial examples.
- Some more examples include likelihood ratios, outlier exposure, virtual logit matching.

§7.6.3 Benchmarks for Anomaly Detection

- There is a much larger search space for constructing anomaly detection benchmarks - we train a model on one dataset and treat any other data as anomalous.
- Example: train on CIFAR-10, evaluate on SVHN
- Example: train on CIFAR-10, evaluate on CIFAR-100
- Example: train on ImageNet-22k, evaluate on Species

§7.6.4 Evaluting Binary Classifiers

- We can think of anomaly detection as binary classification, since we have anomalous or not.
- What might be the issue of just considering accuracy? Class imbalance: What if we have 1 anomaly, 99 normal. The model that always predicts normal has 99% accuracy, but is not a good detector.
- So we need more detailed metrics, true positive, false positives, true negatives, false negatives.

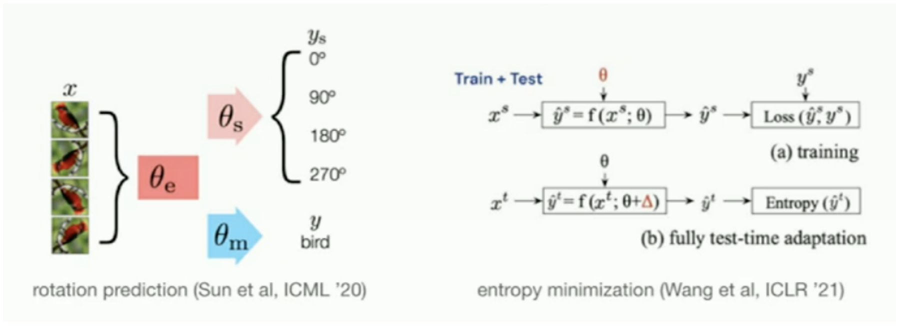
§8 04/04/2022 - Adversarial Examples

§8.1 Test time Adaptation

- We talked about heuristic techniques like scaling up datasets and using data augmentation techniques. But this all happens at training time.
- Are there training time techniques to adapt to shift at test time using available information? Assume we have access to and can change the model's parameters, or we have other means of augmenting the predictions.
- We can assume we have multiple test points, from which we can estimate statistics of the underlying test distribution, and adapt to that distribution.
- if there is a label shift($p(x|y)$ is the same but $p(y)$ changes), we can change the threshold for predicting various classes according to how the label should change. We can estimate this using the confusion matrix.

§8.1.1 Methods for Test-time Adaptation

- Batch Norm adaptation: leave the batch norm layers in training mode at test time. This way we recompute statistics that we use for normalization at the test time.
- Self-supervised learning: have a loss function that only depends on inputs and construct fake labels based on auxiliary tasks that we define. For example: rotation prediction, entropy minimization.

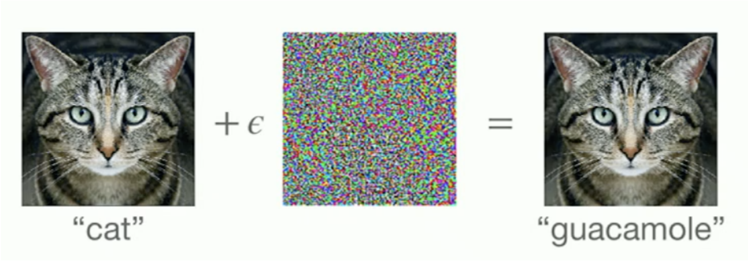


x
θ_e
θ_s
θ_m
y_s
0°
90°
180°
270°
y
bird
rotation prediction (Sun et al, ICML '20)
Train + Test
θ
y^s
x^s
$\hat{y}^s = f(x^s; \theta)$
$\hat{y}^s$
Loss $(\hat{y}^s, y^s)$
(a) training
θ
x^t
$\hat{y}^t = f(x^t; \theta + \Delta)$
$\hat{y}^t$
Entropy $(\hat{y}^t)$
(b) fully test-time adaptation
entropy minimization (Wang et al, ICLR '21)

If we have a standard model $g : \mathcal{X} \to \mathcal{Y}$, we can think of the adapted model as $f : \mathcal{X} \times \mathcal{P}_x \to \mathcal{Y}$, where $\mathcal{P}_x$ is a distribution that the input comes from. In practice, we can estimate $\mathcal{P}_x$ with $(x_1, \ldots, x_k)$.

§8.2 Adversarial Robustness

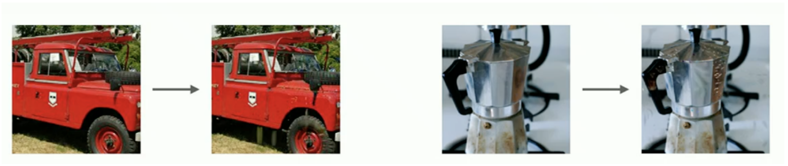
We have an adversarial distortion that is carefully crafted to be imperceivable to the human eye but cause the model to predict a different layer. These are models that have not been trained to handle adversarial examples.



Now, we can mostly train against imperceptible distortions.

§8.2.1 Modern Adversarial Distortions

In the modern examples, we have changes that are perceivable to the human eye, but the underlying class in unchanged. This is not good either because we don't see a difference in the class but a large degradation in the image, but it will fool robustly trained classifiers.



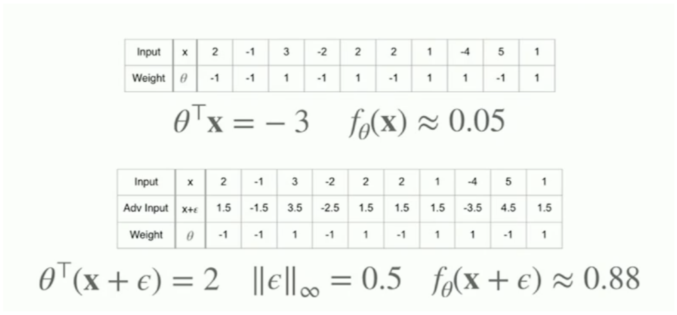
Modern networks can be make robust to imperceptible distortions, but they are not robust to perceptible distortions.

§8.2.2 Fooling Binary Logistic Regression

Suppose we have the model

$$f_\theta(x) = \frac{\exp \theta^T x}{1 + \exp \theta^T x}.$$

If we say that we can change the inputs to $x + \epsilon$ with $\|\epsilon\|_\infty \le 0.5$, we can completely change the prediction of the model:



Input	x	2	-1	3	-2	2	2	1	-4	5	1
Weight	θ	-1	-1	1	-1	1	-1	1	1	-1	1

$$\theta^\top \mathbf{x} = -3 \quad f_\theta(\mathbf{x}) \approx 0.05$$

Input	x	2	-1	3	-2	2	2	1	-4	5	1
Adv Input	x+ϵ	1.5	-1.5	3.5	-2.5	1.5	1.5	1.5	-3.5	4.5	1.5
Weight	θ	-1	-1	1	-1	1	-1	1	1	-1	1

$$\theta^\top(\mathbf{x} + \epsilon) = 2 \quad \|\epsilon\|_\infty = 0.5 \quad f_\theta(\mathbf{x} + \epsilon) \approx 0.88$$

- The comulative effect of many small changes made the adversary powerful enough to change the classification decision.
- Adversarial examples exist for non deep learning models.

§8.3 Adversary Threat Models

It hopes to make assumptions about what the adversary looks like and what they are capable of.

- A simple model is to assume the adversary has an ℓ^p attack *distortion budget* ϵ: for some p, ϵ, $\|x_{adv} - x\|_p \le \epsilon$.
- Not all distortions have small ℓ^p norm, for example rotations. But this is usually tractable to study.
- The goal is to find a distortion δ to maximize the loss subject to the budget

$$x_{adv} = x + \underset{\delta : \|\delta\|_p \le \epsilon}{\operatorname{argmax}} \ell(\theta; x + \delta, y).$$

§8.3.1 Fast Gradient Sign Method(FGSM)

One simple attack:

$$x_{FGSM} = x + \epsilon \operatorname{sign}(\nabla_x \ell(\theta; x, y)).$$

In this case, $\|x_{FGSM} - x\|_\infty = \epsilon$. If the gradient is accurate, the moving in the positive direction increases the loss and vice versa, so we go in the final direction that maximizes the overall loss.

The attack is called fast because it only uses a single gradient ascent step. Nowadays, this is easy to defend against.

§8.3.2 Projected Gradient Ascent(PGD)

PGD uses multiple gradient ascent steps so is more powerful than FGSM. How it works with T steps and ℓ^∞ budget ϵ

1. Randomly initialize a perturbed image $\tilde{x} = x + n$ where $n_i \sim \mathcal{U}[-\epsilon, \epsilon]$ and initialize $\delta = 0$.
2. for $t = 1, \ldots, T$, set $\delta = \operatorname{clip}(\delta + \alpha \operatorname{sign}(\nabla_\delta \ell(\theta; \tilde{x} + \delta, y)), -\epsilon, \epsilon)$.
3. Set $x_{PGD} = \tilde{x} + \delta$.

§8.4 Adversarial Training

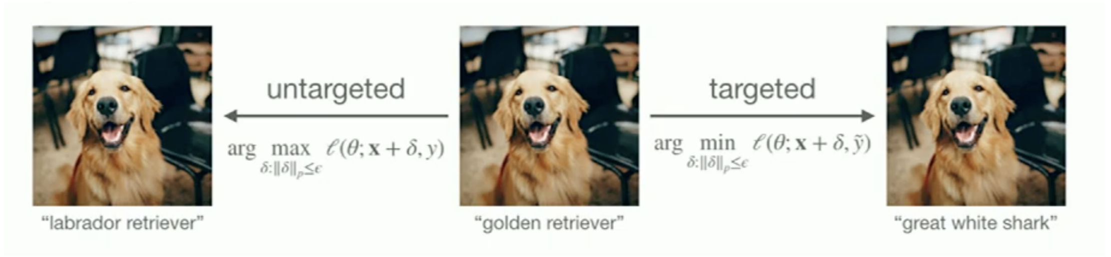
A common procedure is as follows:

1. Sample a minibatch $(x^{(1)}, y^{(1)}), \ldots, (x^{(B)}, y^{(B)})$.
2. Create $x_{adv}^{(i)}$ from $x^{(i)}$ for all i.
3. Optimize the average training loss on these adversarial training examples.

The downside of this is that AT can reduce accuracy on nonadversarial examples by 10%.

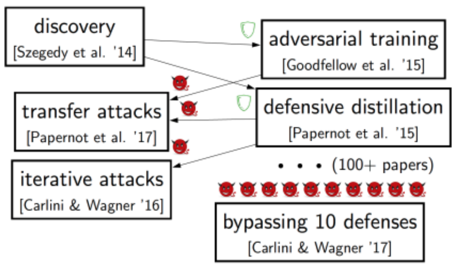
§8.4.1 Untargeted vs. Targeted Attacks

So far, we have assumed untargeted attacks which try to maximize the loss. By contrast, a targeted attack optimizes examples to be misclassified as a predetermined target $\tilde{y}$. Targeted attack evalution is standard for ImageNet because there are many similar classes:



"labrador retriever"
untargeted
$\arg\max_{\delta:\|\delta\|_p \le \epsilon} \ell(\theta; \mathbf{x}+\delta, y)$
"golden retriever"
targeted
$\arg\min_{\delta:\|\delta\|_p \le \epsilon} \ell(\theta; \mathbf{x}+\delta, \tilde{y})$
"great white shark"

§8.4.2 Transferability of attacks



The adversarial arms race

An example crafted for one model can potentially be used to attack different models. For example, given M_1, M_2 neural network models, x_{adv} designed for M_1 can sometimes result in a high loss for $M_2(x_{adv})$ even if M_2 is a different architecture.

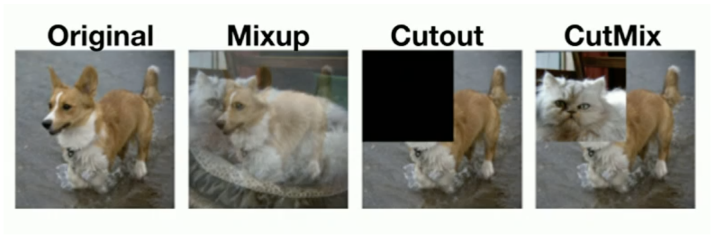
The transfer rates can vary greatly, but even moderate amounts of transferability demonstrate that adversarial failure modes are somewhat shared across models. Consequently, an attacker does not need access to a model's parameters or architectural information to try to attack it.

§8.4.3 Using Larger and More Diverse Data

- Adversarial robustness scales slowly with dataset size
- Adversarial pretraining on a larger training set has been shown to help: to increase CIFAR-100 robustness, adversarially pretrain on ImageNet.

§8.4.4 Data Augmentation

One particularly effective technique, combined with adversarial training with a parameter exponential moving average is **CutMix**.

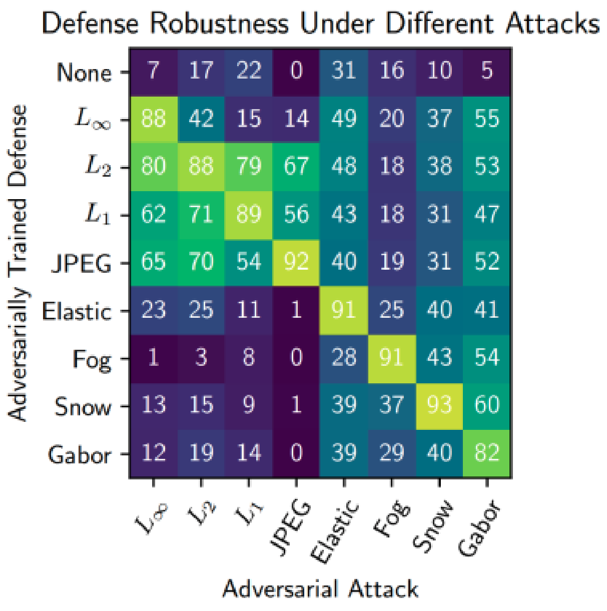


§8.4.5 Choice of Activation Functions

Sharp activations such as ReLUs make AT less effective since gradients are less conditioned, so by improving gradient quality for the attacker and network optimizer, smooth activiations like GELUs improve adversarial training.

§8.5 Unforeseen Adversaries

Models are far less robust to attacks they are not trained against. To estimate robustness to unforeseen attacks,we should measure robustness to multiple attacks not encountered during training



	L_∞	L_2	L_1	JPEG	Elastic	Fog	Snow	Gabor
None	7	17	22	0	31	16	10	5
L_∞	88	42	15	14	49	20	37	55
L_2	80	88	79	67	48	18	38	53
L_1	62	71	89	56	43	18	31	47
JPEG	65	70	54	92	40	19	31	52
Elastic	23	25	11	1	91	25	40	41
Fog	1	3	8	0	28	91	43	54
Snow	13	15	9	1	39	37	93	60
Gabor	12	19	14	0	39	29	40	82

§9 04/06/2022 - Generative Models

Some examples we have seen before are GPT-3 and Masked Autoencoders.

§9.1 Density/Distribution Modeling

Some potential motivations:

- Generation: Synthesize new data analysis that are useful/interesting/pretty.
  - Conditional Generation: Synthesize specific data points of interest
- Density modeling: understand/analyze the likelihood of various data points.
  - This doesn't work that well OOD yet.
- Representation learning(compression or dimensionality reduction): convert high dimensional data into lower dimensional representations.

§9.2 Generating New Data

- In NLP, autoregressive models like GPT reign when it comes to generation.
- Masked autoencoders are capable of generating images, but that is not what they are designed for. For CV, the leading approaches are generative adversarial networks(GANS) and diffusion/score-based models.
- For audio, the leading approach is autoregressive modeling.

§9.3 Generative Adversarial Networks

- We have two networks: a generator G that is synthesizing data from noisy inputs and a discriminator D which distinguishes real data from synthesized data.
- We make these compete against each other leading to the following objective:

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(x)))].$$

- The above objective is akin to the Binary Cross Entropy loss on all the data points where the real data points have label 1 and the synthetic ones are 0.
- If the generator wins, the generator will be generating images that are indistinguishable from real images. If the discriminator is too good, the generator might not be able to do anything to win.
- There is some theory: optimal G^* under Bayes optimal D^* recreates the data distribution.
- After training: throw away D, use G to generate new samples.

We have had many works contributing to GANs with newer tricks and scaling.

§9.3.1 Conditional Generation with GANs

There are many ways to condition the generator to guide the data it produces.

- If we have class information, we can pass the label to both G and D.
- Without class information, we can include a piece of the input that is "label-like"(uniform categorical) to do unsupervised discovery of different categories. Here, additonal objectives, such as maximizes the mutual information between the generator output and "label-like" input(InfoGAN)

We could also condition on other things, like another image.

§9.3.2 Unsupervised Image-to-Image Translation

Translate images from one domain to another domain. For example pictures of horses to zebras!

- CycleGAN, trains two image-to-image generators that turn images from one domain into images of the other domain(and correspondingly two discriminators). There is also cycle consistency in that if I generate a horse and use that to generate a zebra and back to the horse, I should get the same image.

§9.3.3 Summary

- GANs the go-to model for image generation. Other models are catching up in terms of quality but are much slower.
- GANs are not density models, so we can't get probabilty estimates of points from a GAN.
- This makes them difficult to evaluate, how do we determine whether a GAN is better than another GAN or a different model? Usually we have metrics based on inputting generated images into a pretrained classifier(Inception and FID scores). However, this is ad-hoc and doesn't have a theoretical foundation.

§9.4 Density Models on Images

Learning an approximation of $p_\theta(x)$ is difficult and may sometimes be useful(but not always) for detecting anomalous x's. This model precribes a clear approach for both training(MLE) and evaluation(better held out likelihood means better model).

There are three main classes of DGMs that allow for estimating $p_\theta(x)$: **autoregressive**, **latent variable**, and **flow-based models**.

§9.5 Autoregressive Models

Conceptually, we factorize the density according to the chain rule:

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i | x_1, \ldots, x_{i-1}).$$

- x may be sequential, or we can define an ordering.
- How do we define the model so that we get $p_\theta(x_i|x_{<i})$. In the transformer decoder, we used masking to ensure that we don't see future $x_{\geq i}$.

- We can do this similarly for images, masked convolutions(only pay attention to pixels that have come before, PixelCNN), masked self-attention(Image Transformer, Sparse Transformer).

§9.5.1 Distribution over Pixels

What distribution is $p_\theta(x_i|x_{<i})$ for image pixels and how do we parameterize it?

- One option is to make it a 256-dim Categorical (softmax) distribution. But this is expensive, and doesn't capture inductive biases(nearby values are closer than far away values).
- One effective approach is to use a mixture of (truncated) logistic distributions
- Other approaches also sometimes work, treating pixels as continuous values in [0, 1], Beta, Kumaraswamy, Logit-Normal distributions.

§9.5.2 Summary

- Offer the "best in class" modeling performance in terms of generation and likelihood metrics.
- This is true beyond images - there are very good autoregressive models for language and audio
- Similar to GANs, we can also do conditional generation on label information.
- GANs are superior though when it comes to image generation and they are very slow comparatively.
- They don't provide a notion of a latent space, so they aren't used for representation learning.

§9.6 Representation Learning

- The model should have a latent representation z that is lower dimensional. GANs had this.
- Ideally the model would have an encoder that maps $x \to z$.
- We might want to model z as probabilistic for some applications.

§9.6.1 Latent Variable Models

- Model $p(z)$ which is the prior(something simple, unit Gaussian) and $p_\theta(x|z)$ which is the observation model(producing images given a z, generator, generative model, decoder)
- The likelihood of a data point is given by
$$p_\theta(x) = \int p_\theta(x|z)p(z)\,dz.$$
This is intractable unless x, z are very simple.
- We are interested in $p_\theta(z|x)$, which is also intractable.
- Idea: what if we model $p_\theta(z|x)$ with another distribution $q_\phi(z; x)$?

§9.6.2 Evidence Lower Bound(ELBO)

$$\begin{aligned} D_{KL}(q_\phi \| p_\theta) &= \mathbb{E}_{q_\phi}[\log q_\phi(z;x) - \log p_\theta(z|x)] \\ &= \mathbb{E}_{q_\phi}[\log q_\phi(z;x) - \log p_\theta(z,x)] + \log p_\theta(x) \\ &= \mathbb{E}_{q_\phi}[\log q_\phi(z;x) - \log p_\theta(x|z) - \log p(z)] + \log p_\theta(x) \\ &= -(\mathbb{E}_{q_\phi}[\log p_\theta(x|z)] - D_{KL}(q_\phi \| p_z)) + \log p_\theta(x) \\ &\geq 0. \end{aligned}$$

The term $(\mathbb{E}_{q_\phi}[\log p_\theta(x|z)] - D_{KL}(q_\phi \| p_z))$ is called the evidence lower bound.

§9.6.3 Variational Autoencoders(VAEs)

- In VAEs, both the observation model $p_\theta(x|z)$ and recognition mode(or encoder) $q_\phi(z;x)$ are traininded to maximize the evidence lower bound. This is an example of variational inference.
- Intuitively, ELBO contains the reconstruction term and regularization term.
- After training, we may keep both models based on what we want. q_ϕ gives us a natural approach for generating representations of new data points, but p_θ combined with the prior allows us to synthesize new data points.

§9.6.4 Summary

- VAEs give a natural mechanism for representation learning and generating.
- Though estimating $p_\theta(x)$ is difficult, a lower bound can be easily obtained.
- We have a tradeoff between representation learning and generation quality.
- The VAEs which synthesize the best data points and result in the best likelihoods utilize complex priors and modeling choices(quantization and multiple levels of latent variables), making extracting useful representations difficult.

§9.7 Generating Images from Language

- OpenAI DALL.E 2 creates realistic images and art from a description in English.

It uses a Diffusion Model:

- A probabilistic model which defines a stochastic process where x is transformed via an additive Gaussion noise, into z which is just pure noise.
- In particular, we learn the reverse process $z \to x$ and the forward process $x \to z$ is fixed as incrementally adding small amounts of noise.
- Similar to VAEs, we train by maximizing the evidence lower bound.
- These generate impressive samples, but sampling is expensive.

§10 04/11/2022 - Self-Supervised Learning

Self-supervised Learning: Creating labels from unlabeled data. We hide some information from the model and ask it to predict this.

We can think of generative modeling as some form of self-supervised learning, but they are not the same in general.

Both of these have shown to be useful in representation learning in a number of domains.

§10.0.1 Why Self-Supervised

- Good supervision is not cheap, and we already have vast amounts of unlabeled data on the internet for a number of different modalities.
- These data may be able to teach models about the structure of the domain.
- Predict parts of the data given other parts
- Or, we may have additional domain knowledge we can pass on to the model via training/data augmentation.

§10.0.2 Examples

Before, we have seen examples such as

- Masked Autoencoder
- BERT
- Auto-regressive modeling(kind of) since we predict the future given the past.

§10.0.3 Masked Autoencoding(MAE)

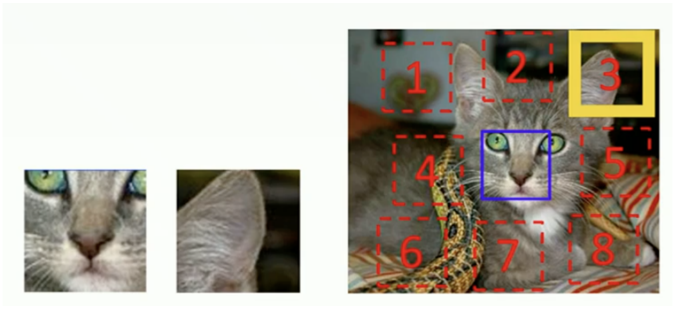
- Randomly masking out parts of the input(15% of tokens or 75% of image patches) and predicting these parts is an effective self-supervised approach.
- MAE vision transformers and BERT and both transformer encoders that turn masked inputs into representations that are useful for downstream tasks.
- During training, they are trained with simple decoder(token classifiers and a small transformer for MAE).
- These decoders attempt to recover the original input that was masked out and the encoder is thus trained to produce useful contextual representations.

§10.1 Self-supervised Learning in Vision

- For a long time, MAE was not performant or scalable approach to self-supervised learning in CV. We didn't have large enough datasets or compute power.
- As a result, we have a lot of other techniques.
- These leverage the known structure of images, **constrastive learning** and data augmentation.

§10.1.1 Leveraging Spatial Context(2015)

- Self-supervised task: Extract two patches from the image, and predict the relative position of the second patch with respect to the first.
- Proved to be useful in pre-training ConvNets for downstream object detection tasks, likely because the network learns about objects and their parts.



§10.1.2 Predicting Rotations(2018)

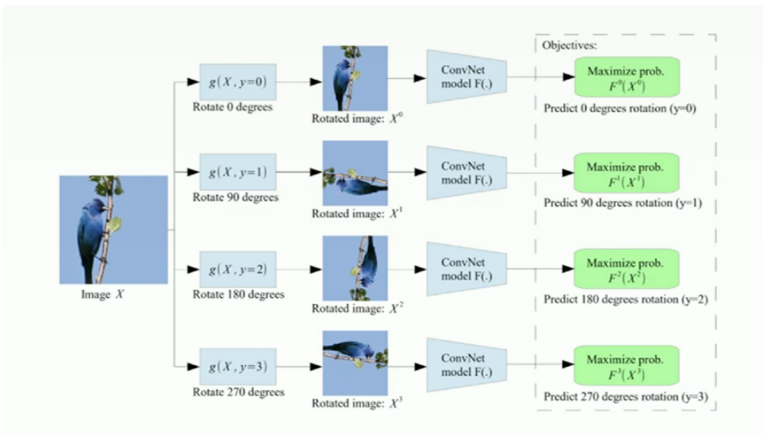


Image X
$g(X, y=0)$
Rotate 0 degrees
$g(X, y=1)$
Rotate 90 degrees
$g(X, y=2)$
Rotate 180 degrees
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^0
Rotated image: X^1
Rotated image: X^2
Rotated image: X^3
ConvNet model F(.)
Objectives:
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

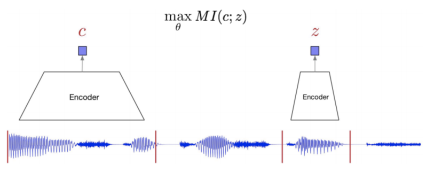
§10.2 Contrastive Learning

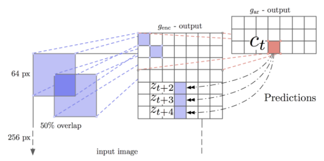
- Contrastive Learning: learned representations hsould be close together for similar inputs and far apart for dissimilar inputs.
- If we have representation z generated from some input z with representations $z_1, \ldots, z_k$ from other inputs.
  - If one of $z_1, \ldots, z_k$ is generated from a similar input z_+, we have the most common contrastive learning loss

$$-\log \frac{\exp(z^T z_+/\tau)}{\sum_{i=1}^K \exp(z^T z_i/\tau)}.$$

§10.2.1 Contrastive Predictive Coding(CPC)(2018)

- CPC defines similar as coming from the same input, which requires splitting up the input into multiple parts, and dissimilar as coming from different inputs.
- This has been applied to audio, images, text.
- Extracting a single representation from the whole input requires more work.





§10.3 Data Augmentation

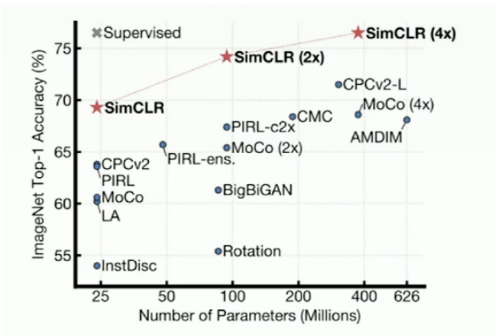
- Many other types of data augmentations, besides rotations, can be leveraged to produce useful self-supervised learning signals in CV.
- However, instead of trying to predict the augmentation, we can use the augmentations in conjunction with contrastive learning.
- Specifically, we consider different augmentations of the same image as similar and different images as dissimilar.
- Now we can learn representations of entire images directly, since we don't need to split up the image direction.

§10.3.1 Momentum Contrast(MoCo)(2019)

- A key challenge in constrastive learning is sampling a sufficiently large and diverse set of dissimilar examples for training
- If it is too large, then generating representations for all negatives will be expensive.
- MoCo maintains a queue of negatives and adds it to every training iterations with the representations from the latest minibatch, pushing out the oldest minipatch.
- The small problem is that the representations at the end of the queue are not from the same model anymore.
- Since the representation itself is changing, MoCo uses an exponential moving average(EMA) of the model parameters to generate representations for the queue, encouraging some similarity across mini batches.

§10.3.2 SimCLR

- We can remove the need for a queue if we incorporate larger batch sizes, longer training, larger models, stronger augmentations, and a few technical improvements.
- These are summarized by SimCLR.



§10.4 Bootstrap your own latent(BYOL)(2020)

- Contrastive learning is popular, but there are other approaches.
- One idea is to train two networks(the online and target or student and teacher).
- in BYOL, the online and target are given two augmentations of the same input, and the online is optimized to try to predict the target's network representation.
- The target network is updated as an EMA of the online network.

§10.5 Self-distillation with no labels (DINO)(2021)

- Have the student try to match the probability values of the teacher.
- They both get inputs and the teacher products some probability values and the student is trying to match these values.
- The idea is called **distillation** in the case where the teacher knows something; it has been pretrained on some dataset.
- Self-distillation is the self-supervised version of this, where it works well even when the teacher knows nothing.
- The student is optimized to try to match the probability values and the teacher is updated with an EMA on the student parameters.

§10.6 Multimodal Constrastive Learning: CLIP(2021)

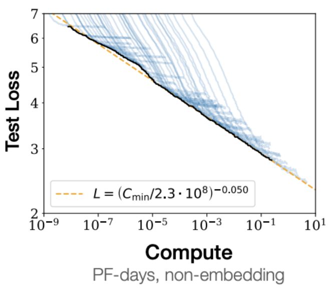
- We are increasingly seeing powerful multimodel models which combine information across modalities.
- Contrastive language-image pretraining(CLIP) is one model trained on a large dataset of (image, text) pairs collected from the internet. This is unlabeled data
- The CLIP model has a text encoder and an image encoder where we output representations which we embed into the same embedding space with the same dimension so we can compare them directtly. Then, we train with a contrastive objective where the (image, text) pairs are similar and all other non-pairs are dissimilar.
- This is an important piece of the DALL.E 2 system.

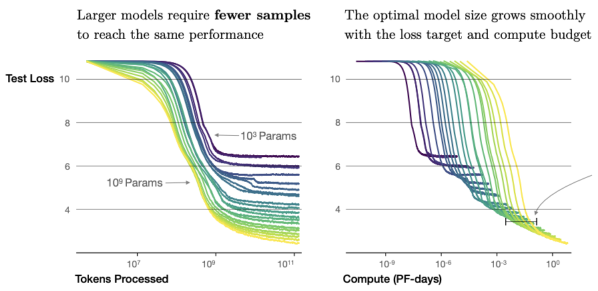
§11 04/13/2022 - Massive Models

We have models with over 100B parameters, and all of them are transformer decoder language models. A natural question is to consider the scaling laws theoretically.

§11.1 Scaling Laws

§11.1.1 Scaling Laws for NLMs(2020)

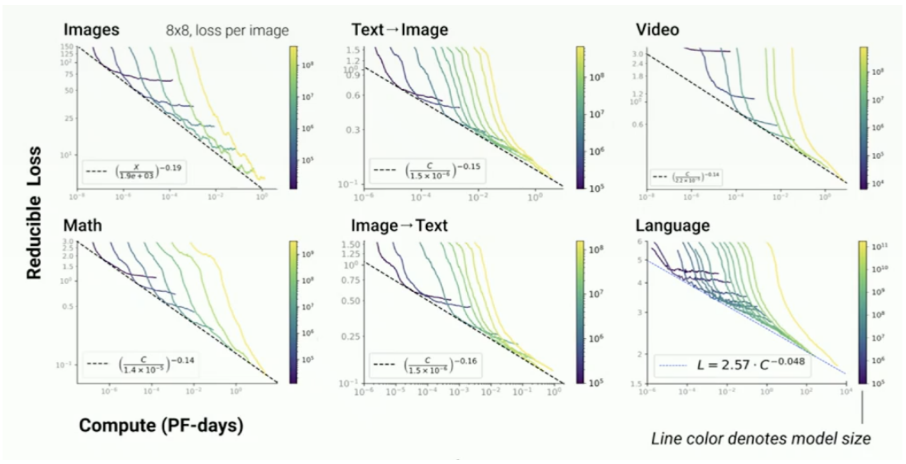




Some weird things from these plots on the right:

- The scaling laws hold for over six orders of magnitude for the amount of available compute and model size.
- Model size and dataset size need to be scaled together, but not equally : 8x model size requires only 5x dataset size. This is disputed today by other work saying that equal scaling is best.
- Larger models require fewer data points and optimization steps to reach the same performance as smaller models.
- For a fixed compute budget, the best performance comes from training large models and stopping well short of convergence.

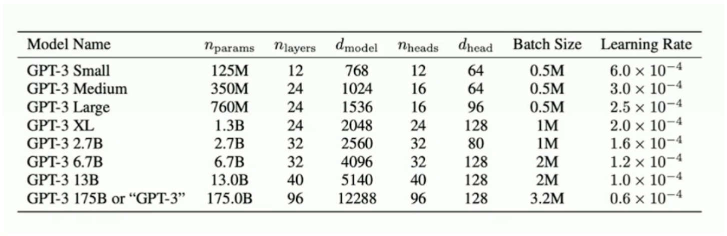
§11.1.2 Scaling Laws for autoregressive generative modeling(2020)



§11.2 Massive Models

§11.2.1 GPT-3(2020)

175B parameters, combined training set has 500B tokens. The model never sees the whole dataset(only 300B).



Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Some options we have are

- Zero shot: predict the answer only given a natural language description of the task . No gradient updates are performed
- One-shot: in addition to the task description, the model sees a sinlge example of the task. No gradient updates performed.
- Few shot: in addition to the task description, the model sees a few examples of the task. No gradient updates performed.

GPT-3 is very good at Few-shot learning, but there is significant room for improvement.

§11.2.2 Gopher(2021)

- Gopher is a 280B parameter transformer decoder
- 2T tokens but only trained for 300B tokens.
- many different sizes of the model, architecture considered and huge dataset collected.
- applicable to a diverse set of NLP tasks.

§11.2.3 Chinchilla(2022)

- Chinchilla considers varying hyperparameters (primarily, learning rate schedule) that Kaplan et al held fixed, which leads to different scaling conclusions
- They concluded that model and dataset size should scale **equally**
- Tested on MAssive Multitask Language Understand(MMLU) which consists of exam questions from 57 academic subjects from elementary to professional level.

§11.2.4 Megatron-Turing NLG(2022)

- MT-NLG is a 530B parameter transformer decoder
- Training set is (a puny) 339B tokens, training is done with 270B tokens

§11.2.5 PaLM(2022)

- PaLM is a 540B parameter (yes, you guessed it) transformer decoder
- Training set: 780B tokens; training: one full epoch!

§11.3 Applications of Massive Models

§11.3.1 Few-shot learning via Prompting

- Provide some examples of the task within the model input itself
- Prompt engineering is getting the model to perform the best by giving the best prompt
- PaLM using chain-of-thought prompting which not only prompts the correct answer but also the process at which the answer is reached.
- Self-consistency(sampling multiple answers and picking the most consistent answer).

§11.3.2 Medium-shot learning with fine tuning

- fine tuning refers to updating the model via gradient based optimization with a small data set.
- This is usually not feasible with the large size of the models.
- GPT-3 now offers fine tuning as part of the API.
- When possible, fine tuning significantly outperforms prompting.

§11.3.3 Specializing to Code: Codex and AlphaCode

- Codex is a 12B parameter model and fine tunes on 159GB from GitHub. This powers GitHub Copilot.
- AlphaCode scales up to competition level coding, comparable to the median competitor.
- 41B model size and 517B dataset size, model architecture is encoder-decoder, fine tuning on competition code, sampling/filtering many candidate solutions help in scaling to this level.

§11.4 Limitations of Massive Models

§11.4.1 Challenge Tasks

- Stress tests like ANLI significant room for improvement and scaling up doesn't solve this problem.
- Hard tasks like solutions for high school math contests have low accuracy.
- These models are trained/fine tuned with more text/code/math than a human will ever see, so it seems like something is missing.

§11.4.2 Potential Harms and Biases

- Papers come with a model card describing details, intended uses, and some analysis about potential harms.
- GPT-3 was analyzed for gender, race, and religion biases, and this shed light on its predispositions that seem in line with training.

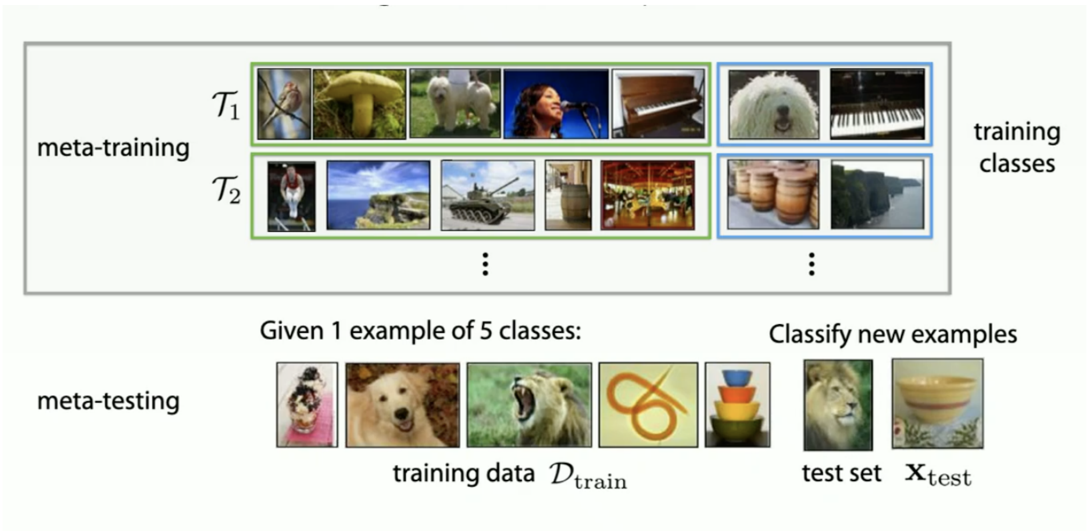
- Analysis on Gopher (and, to an extent, PaLM) demonstrates that large models, when given a toxic prompt, are more likely to generate toxic continuations
-

§12 04/18/2022 - Chelsea Finn: Meta Learning

- Adapting to new objects
- Adapting to new molecules
- Adapting from simulation to real
- Adapting to new regions of the world

§12.1 How does Meta-Learning Work?

- Given 1 example of 5 classes and some test sets, it would not be able to train on its own.
- Assume that we have other image data, split it into mini train sets and test sets.
- Hope that after doing the meta-training, we can meta-test with new classes.
- Image classification can be replaced with regression, language generatioon, skill learning, or really any ML problem.


meta-training
$\mathcal{T}_1$
$\mathcal{T}_2$
training classes
Given 1 example of 5 classes:
Classify new examples
meta-testing
training data $\mathcal{D}_{\text{train}}$
test set $\mathbf{x}_{\text{test}}$

§12.2 Mathematical Formulation of Meta Learning

In supervised learning, we have inputs x, outputs $y = g_\phi(x)$ and a dataset $\{(x, y)_i\}$.

In meta supervised learning, we have inputs $\mathcal{D}^{tr} = \{(x, y)_{1:K}\}$ and x^{ts}, and the goal is to predict the corresponding label output $y^{ts} = f_\theta(\mathcal{D}^{tr}, x^{ts})$. Now, we have a dataset of datasets $\{\mathcal{D}_i\}$, $\mathcal{D}_i = \{(x, y)_j\}$.

We reduce meta-learning to designing and optimizing f, which we can do using the dataset of datasets and end-to-end learning.

§12.3 Meta Learning Problem

Given data from $\mathcal{T}_1, \ldots, \mathcal{T}_n$, so a new task $\mathcal{T}_{test}$. We have a key assumption: meta-training and meta-test task drawn i.i.d. from same task distribution.

$$\mathcal{T}_1, \ldots, \mathcal{T}_n \sim p(\mathcal{T}), \mathcal{T}_j \sim p(\mathcal{J}).$$

The tasks can correspond to

- Recognizing handwritten digits from new languages
- Giving feedback to students on different exams
- classifying species in different regions of the world
- a robot performing different tasks

In general, the more tasks, the better.

§12.3.1 Other Similar Problems

- **Multi-Task Learning**: solve multiple tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ at once:
$$\min_\theta \sum_{i=1}^T \mathcal{L}_i(\theta, \mathcal{D}_i).$$
- **Transfer Learning**: Solve target task $\mathcal{T}_b$ after solving source task $\mathcal{T}_a$ by transferring knowledge from $\mathcal{T}_a$.
- **Meta-Learning**: Given data from $\mathcal{T}_1, \ldots, \mathcal{T}_n$ solve $\mathcal{T}_{test}$.

In Meta-Learning and Transfer Learning, we generally assume we don't have access to prior tasks, but in all of these settings the tasks all share some prior structure.

§12.3.2 Terminology

- D_i^{tr}: support set
- D_i^{test}: query set
- k-shot learning: learning with k exmaples per class
- N-way classification: choosing between N different classes.

§13 04/20/2022 - Sergey Levine: Deep Reinforcement Learning