Machine Learning

Anders Jonsson & **Vicenç Gómez**

Master in Intelligence Interactive Systems
2021-22

Lecture 8
Deep Learning and Applications II

1 Recap

2 Recurrent Neural Networks

3 Attention and Transformers

4 Application: Language Models

Content

1 Recap

2 Recurrent Neural Networks

3 Attention and Transformers

4 Application: Language Models

History of deep learning

- 1943 Computer model of artificial neuron [McCulloch and Pitts]
- 1960-70 Backpropagation algorithm for training neural networks [Kelley; Bryson; Dreyfus; Ivakhnenko; Lapa; Werbos]
- 1970-80 First AI winter
- 1980-86 Modern version of backpropagation [Linnainmaa; Fukushima; Parker; Rumelhart et al.]
- 1986-95 Second AI winter
- 1995- Novel network models, faster processors, GPUs
- 2010- Breakthrough of deep learning

Computational power

Content

1 Recap

2 Recurrent Neural Networks

3 Attention and Transformers

4 Application: Language Models

Recurrent Neural Networks

- Example of a **feedforward neural network** (here: MLP with 1 hidden layer)
- In the following, we will drop all parameter nodes and highlight outputs in blue

Feedforward vs Recurrent Neural Network

Recurrent Neural Networks (RNNs):

- Core idea: update **hidden state** $\mathbf{h}$ based on input and previous hidden state using same update rule (same/shared parameters) at each **time step**
- Allows for processing **sequences of variable length**, not only fixed-sized vectors
- **Infinite memory:** $\mathbf{h}$ is function of all previous inputs (long-term dependencies)

Basic Recurrent Neural Network

General Form:

$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t)$$

- We use t as the **time index** (in feedforward networks we used i as layer index)
- General form does not specify the form of the hidden and output mappings
- Important: f_h and f_y **do not change over time**, unlike in layers of feedforward net

Basic Recurrent Neural Network

Single-Layer RNN:

$$\mathbf{h}_t = \tanh(\mathbf{A}_h \mathbf{h}_{t-1} + \mathbf{A}_x \mathbf{x}_t + \mathbf{b})$$

$$\hat{\mathbf{y}}_t = \mathbf{A}_y \mathbf{h}_t$$

- Hidden state $\mathbf{h}_t$ = linear combination of input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_{t-1}$
- Output $\hat{\mathbf{y}}_t$ = linear prediction based on current hidden state $\mathbf{h}_t$
- $\tanh(\cdot)$ is commonly used as activation function (data is in the range $[-1, 1]$)
- Parameters $\mathbf{A}_h$, $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{b}$ are constant over time (sequences may vary in length)

Mapping Types

RNNs allow for processing **variable length** inputs and outputs:

- **One to Many:** Image captioning (image to sentence)
- **Many to One:** Action recognition (video to action)
- **Many to Many:** Machine translation (sentence to sentence)
- **Many to Many:** Object tracking (video to object location per frame)
- To determine the length of the output sequence, a **stop symbol** can be predicted

Backpropagation Through Time

- To train RNNs, we **backpropagate gradients through time**
- As all hidden RNN cells share their parameters, gradients get accumulated
- However, very quickly intractable (memory) for larger sequences (e.g., Wikipedia)

(Truncated) Backpropagation Through Time

- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

(Truncated) Backpropagation Through Time

- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

(Truncated) Backpropagation Through Time

- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

Multi-Layer RNNs

- Deeper **multi-layer RNNs** can be constructed by stacking RNN layers
- An alternative is to make each individual computation (=RNN cell) deeper
- Today, often combined with residual connections in vertical direction

Limitations of Vanilla RNNs

What is the problem with vanilla RNNs?

- The state update H_t is modeled using a zero-centered $\tanh(\cdot)$
- $\tanh(\cdot)$ assumes that the processed data is in the range $[-1, 1]$
- Remark: we omit the affine transformations and the output layer for clarity

Vanishing-Exploding Gradients

$$h_t = \tanh\left(a_h\, h_{t-1} \,+\, a_x\, x_t \,+\, b\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh' a_h \;\text{ with }\; \tanh' = \frac{\partial \tanh(x)}{\partial x}$$

What is the problem with vanilla RNNs?

- Let us consider an RNN with one dimensional hidden state $h_t \in \mathbb{R}$
- We have: $$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = \left(\prod_{i=t-k+1}^{t} \tanh'_i \right) a_h^k$$
- Thus, the gradient vanishes if $\tanh(\cdot)$ saturates as in feedforward networks
- RNNs require careful initialization to avoid saturating activation functions

Vanishing-Exploding Gradients

$$h_t = \tanh\left(a_h\, h_{t-1} + a_x\, x_t + b\right)$$
$$\approx a_h\, h_{t-1} + a_x\, x_t + b$$
$$\frac{\partial h_t}{\partial h_{t-1}} \approx a_h$$

However, gradients might still misbehave:

- Let us now assume that the RNN has been initialized well such that the activation functions are not saturated $\Rightarrow a_h\, h_{t-1} + a_x\, x_t + b \in [-1, 1] \Rightarrow \tanh(x) \approx x$
- We now have:

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

Vanishing-Exploding Gradients

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

For $a_h > 1$ gradients will **explode** (become very large, cause divergence):

- Example: For $a_h = 1.1$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 13781$
- This problem is often addressed in practice using **gradient clipping**
- Forward values do not explode due to bounded $\tanh(\cdot)$ activation function

For $a_h < 1$ gradients will **vanish** (no learning in earlier time steps):

- Example: For $a_h = 0.9$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 0.0000266$
- Avoiding this problem requires an **architectural change**
- But residual connections do not work here as the parameters are shared across time and the input and desired output at each time step are different

Gradient Clipping

The effect of exploding gradients can be dampened by a simple heuristic which clips the gradients to $\|\texttt{A.grad}\|_2 \leq \tau$ before applying the gradient update during SGD:

$$\texttt{A.grad} = \begin{cases} \texttt{A.grad} & \text{if } \|\texttt{A.grad}\|_2 \leq \tau \\ \tau \frac{\texttt{A.grad}}{\|\texttt{A.grad}\|_2} & \text{otherwise} \end{cases}$$

The maximal gradient magnitude τ is a hyperparameter, often $\tau \in [1, 10]$.

What can we do to avoid vanishing gradients?

Gated Recurrent Networks

UGRNN

Collins, 2017

GRU

Cho, 2014

LSTM

Hochreiter, 1997

- **UGRNN:** Update Gate Recurrent Neural Network
- **GRU:** Gated Recurrent Unit
- **LSTM:** Long Short-Term Memory
- LSTM was the first and most transformative (revolutionized NLP in 2015, e.g. at Google), but also most complex model. UGRNN and GRU work similarly well.
- Common to all architectures: **gates** for filtering information

Gradient Flow

$$\mathbf{h}_t = \tanh\left(\mathbf{A}_h\mathbf{h}_{t-1} + \mathbf{A}_x\mathbf{x}_t + \mathbf{b}\right)$$

$$\mathbf{h}'_t = \tanh' \mathbf{A}_h \approx \mathbf{A}_h$$

with $\mathbf{h}'_t = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$

$$\mathbf{u}_t = \sigma\left(\mathbf{A}_{uh}\mathbf{h}_{t-1} + \mathbf{A}_{ux}\mathbf{x}_t + \mathbf{b}_u\right)$$

$$\mathbf{s}_t = \tanh\left(\mathbf{A}_{sh}\mathbf{h}_{t-1} + \mathbf{A}_{sx}\mathbf{x}_t + \mathbf{b}_s\right)$$

$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \mathbf{s}_t$$

$$\mathbf{h}'_t = \mathbf{u}'_t \odot \mathbf{h}_{t-1} + \mathbf{u}_t + (1 - \mathbf{u}'_t) \odot \mathbf{s}_t + ..$$

- UGRNN can maintain gradient flow despite small $\mathbf{A}_h$ by setting its gate to $u \approx 1$

Gradient Flow

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

- UGRNN is able to **keep the state** of a variable **over a long time horizon** ($u \approx 1$)
- For example, it can implement a logic to keep track of being inside a quote or not

Gated Recurrent Unit (GRU)

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rh}\,\mathbf{h}_{t-1} + \mathbf{W}_{rx}\,\mathbf{x}_t + \mathbf{b}_r)$$

$$\mathbf{u}_t = \sigma(\mathbf{W}_{uh}\,\mathbf{h}_{t-1} + \mathbf{W}_{ux}\,\mathbf{x}_t + \mathbf{b}_u)$$

$$\mathbf{s}_t = \tanh(\mathbf{W}_{sh}\,(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{sx}\,\mathbf{x}_t + \mathbf{b}_s)$$

$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \mathbf{s}_t$$

- **Reset gate** controls which parts of the state are used to compute next target stat
- **Update gate** controls how much information to pass from previous time step

Gated Recurrent Unit (GRU)

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fh}\,\mathbf{h}_{t-1} + \mathbf{W}_{fx}\,\mathbf{x}_t + \mathbf{b}_f)$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{ih}\,\mathbf{h}_{t-1} + \mathbf{W}_{ix}\,\mathbf{x}_t + \mathbf{b}_i)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{oh}\,\mathbf{h}_{t-1} + \mathbf{W}_{ox}\,\mathbf{x}_t + \mathbf{b}_o)$$
$$\mathbf{s}_t = \tanh(\mathbf{W}_{sh}\,\mathbf{h}_{t-1} + \mathbf{W}_{sx}\,\mathbf{x}_t + \mathbf{b}_s)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{s}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Passes along an additional cell state $\mathbf{c}$ in addition to the hidden state $\mathbf{h}$. Has 3 gates:

- **Forget gate** determines information to erase from cell state
- **Input gate** determines which values of cell state to update
- **Output gate** determines which elements of cell state to reveal at time t

Remark: Cell update $\tanh(\cdot)$ creates new target values $\mathbf{s}_t$ for cell state

Gated Recurrent Unit (GRU)

UGRNN:

- One gate
- Expose entire state
- Single update gate
- Few parameters

GRU:

- Two gates
- Expose entire state
- Single update gate
- Medium parameters

LSTM:

- Three gates
- Control exposure
- Input/forget gates
- Many parameters

A systematic study [Collins et al., 2017] states:

"Our results point to the GRU as being the most learnable of gated RNNs for shallow architectures, followed by the UGRNN."

Sequence to Sequence Learning

- **Two 4-Layer LSTMs** for encoding/decoding the source/target sentence
- Encoding operates in **reverse order** to introduce short-term dependencies
- Intermediate representation produced by the encoder is called **thought vector**
- Encoding using 1000 dim. word embeddings, decoding via **beam search**
- First end-to-end system that outperforms rule-based models ⇒ deployment

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014

Decoding

Let $\mathbf{w}_1, \ldots, \mathbf{w}_T$ denote the target sentence and let $\mathbf{v}$ denote the thought vector. **Sampling** a translation from the LSTM decoder is simple:

$$\mathbf{w}_t \sim p(\mathbf{w}_t | \mathbf{v}, \mathbf{w}_1, \ldots, \mathbf{w}_{t-1})$$

But often we like to compute the **most probable translation:**

$$\mathbf{w}_1, \ldots, \mathbf{w}_T = \underset{\mathbf{w}_1, \ldots, \mathbf{w}_T}{\operatorname{argmax}} \; p(\mathbf{w}_1, \ldots, \mathbf{w}_T | \mathbf{v})$$

This is costly, but a **greedy algorithm** often works well in practice:

$$\mathbf{w}_t = \underset{\mathbf{w}_t}{\operatorname{argmax}} \; p(\mathbf{w}_t | \mathbf{v}, \mathbf{w}_1, \ldots, \mathbf{w}_{t-1})$$

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014

Beam Search

Failure of Greedy Algorithm:

- p(Apples are good) $>$ p(Those apples are good)
- p(Those) $>$ p(Apples)

Idea of Beam Search:

- At each time step, **maintain a list** of the K best words and hidden vectors
- This can be used to produce a list of K best decodings for the following word, which can then be compared to select the most likely one

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014

Beam Search

https://d2l.ai/chapter_recurrent-modern/beam-search.html

Sutskever, Vinyals and Le: Sequence to Sequence Learning with Neural Networks. NIPS, 2014

Content

1 Recap

2 Recurrent Neural Networks

3 Attention and Transformers

4 Application: Language Models

Attention

- Attention in Computer Vision
  - 2014: Attention used to highlight important parts of an image that contribute to a desired output

- Attention in NLP
  - 2015: Aligned machine translation
  - 2017: Language modeling with **Transformer networks**

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

Attention

Challenges with RNNs

- Long range dependencies
- Gradient vanishing and explosion
- Large # of training steps
- Recurrence prevents parallel computation

Transformer Networks

- Facilitate long range dependencies
- No gradient vanishing and explosion
- Fewer training steps
- No recurrence that facilitate parallel computation

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i
- Example:

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i
- Example:
  - q : possible question (Q)

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i
- Example:
  - q : possible question (Q)
  - k_i : potentially useful information to answer (K)

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i
- Example:
  - q : possible question (Q)
  - k_i : potentially useful information to answer (K)
  - v_i : potential answer (V)

Attention Mechanism

- Mimics the retrieval of a **value** v_i for a **query** q based on a **key** k_i in a database
- In a probabilistic way

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \underbrace{\text{similarity}(q, k_i)}_{\alpha_i} \cdot v_i$$

- The **query** q and **key** k_i are embedding vectors
- Typically, similarity is chosen to be the dot product
- The produced output is a linear combination of values (also embedding vectors) v_i
- Example:
  - q : possible question (Q)
  - k_i : potentially useful information to answer (K)
  - v_i : potential answer (V)
- Everything is learned from data

The Transformer Network

- Initially developed for machine translation
- No recurrence
  - Input: entire sequence to translate
  - Output: entire translated sequence
- Encoder-decoder based on attention
- Key idea: learns abstract embeddings
  - 1st layer: pairs of embeddings
  - 2st layer: pairs of pairs of embeddings
  - Up to $N = 6$ layers

The Transformer Network: Encoder

- Input embedding: zero-order embedding (size n)
- Positional Encodings: concatenates positional information
- Multi-headed attention (see next slide)
- Normalization layer: reduces "covariance shift" and speeds up converge
- Feedforward layer: shared weights within layer
- Output: a higher order embedding of the sentence (size n)

Attention in the Transformer

- Three types of attention modules
  - Encoder: self-attention (key, value and query are input embeddings)
  - Encoder $\rightarrow$ Decoder : query (output embedding), key and value (input embeddings)
  - Decoder: self-attention (key, value and query are input embeddings) and masked attention (prevents attending future words)

Attention in the Transformer

- Multi-head attention
  - Computer multiple attentions per query with different weights
  - Lets network learn different semantic meanings of attention

Multi-headed Attention Visualization

The Transformer Network: Decoder

- Similar to encoder
- When decoding, an output value should only depend on previous outputs
- Masked multi-head attention: multi-head where some values are masked (i.e., probabilisties of masked values are nullified to prevent them from being selected)
- Teacher forcing (trained with correct output word)

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser and Polosukhin: Attention is All you Need. NIPS, 2017.

Transformer-based Models (BERT)

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

- Does not predict next words sequentially
- Uses a context composed of previous and future words
- Masked language modeling (MLM) like “fill the gaps”
- Text encoding, summarization, Question Answering
- Two training phases
  - First: unsupervised training (basically a Transformer **Encoder** stack)
  - Second: specific fine-tuning for a particular task

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Transformer-based Models (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Basically a Transformer **Decoder** stack
- Trained unsupervised on a large corpus to predict the next word
- "prompts" the language model with different inputs

Radford et al. Language models are unsupervised multitask learners (2019).
Brown et al. Language Models are Few-Shot Learners (2020)

Transformer-based Models (GPT)

Generative Pretrained Transformer (**GPT-x**)

- GPT-2 sizes

Radford et al. Language models are unsupervised multitask learners (2019).
Brown et al. Language Models are Few-Shot Learners (2020)

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **positive** sentiment: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **positive** sentiment: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `She looked tired. She'd been crying. Next to her was a man of medium build and unremarkable height, with brown hair just tousled enough to be fashionable. He wore a grey suit, its gravity somewhat offset by a black tie that featured Marvin the Martian. I recognized him.`

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **positive** sentiment: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `She looked tired. She'd been crying. Next to her was a man of medium build and unremarkable height, with brown hair just tousled enough to be fashionable. He wore a grey suit, its gravity somewhat offset by a black tie that featured Marvin the Martian. I recognized him.`
- **Zero-shot baseline:** `He was the guy who'd been chasing me the day I'd arrived in the city.`

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **positive** sentiment: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `She looked tired. She'd been crying. Next to her was a man of medium build and unremarkable height, with brown hair just tousled enough to be fashionable. He wore a grey suit, its gravity somewhat offset by a black tie that featured Marvin the Martian. I recognized him.`
- **Zero-shot baseline:** `He was the guy who'd been chasing me the day I'd arrived in the city.`
- **fine-tuning:** `He smiled at me and I smiled back. He was pleasant enough, for a man of his age.`

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **physically descriptive text**: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **physically descriptive text**: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `It's not her fault I was in a bad mood today. Not her fault I stayed up all night then took it out on her. In a way, she's like basketball for me. An escape. Maybe a much moodier escape, but one all the same.`

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **physically descriptive text**: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `It's not her fault I was in a bad mood today. Not her fault I stayed up all night then took it out on her. In a way, she's like basketball for me. An escape. Maybe a much moodier escape, but one all the same.`
- **Zero-shot baseline:** `I'm standing in the kitchen, trying to decide if I should call her when she comes bustling in.`

Examples of fine-tuning (GPT)

Generative Pretrained Transformer (**GPT-x**)

- Task: Given some text, generate a natural continuation with **physically descriptive text**: (fine-tuning GPT2 with 5000 human samples suffice for strong performance according to humans)
- **Context (input):** `It's not her fault I was in a bad mood today. Not her fault I stayed up all night then took it out on her. In a way, she's like basketball for me. An escape. Maybe a much moodier escape, but one all the same.`
- **Zero-shot baseline:** `I'm standing in the kitchen, trying to decide if I should call her when she comes bustling in.`
- **fine-tuning:** `I pause on the steps, listening to the distant sound of music playing from the living room. looked tired. She'd been crying.`

Transformer-based Models (GPT)

Examples of few-shot learning

Task: Create opposite sentences (input in **bold**)

This is good.
This is bad.

The good horse came home.
The bad horse came home.

The good horse with a large head came home.
The bad horse with a small head came home.

The black cat had a disappointing time.
The white cat had a good time.

When you jump high, you fall hard.
When you fall hard, you get hurt.

He ran fast, but he came last.
He ran slowly, but he came first.

The book was huge, but the students finished it early.
The book was small, but the students finished it late.

Getting up early is a good habit.
Getting up late is a bad habit.

Task: Wherever boy, girl, man or woman comes, replace it with *

Q: you are a good boy.
A: you are a good *

Q: Where is the spoon?
A: Where is the spoon?

Q: Can he jump over the dog?
A: Can he jump over the dog?

Q: What is the time, the girl asked?
A: What is the time, the * asked?

Q: The woman is going to school with the man.
A: The * is going to school *

Q: The girl is taller than the woman.
A: The * is taller than the *

Q: The man, woman, girl and boy went on a trip.
A: The *,*,* and * went on a trip.

Problems and Controversy

Far from **understanding** languange

Opinion

GPT-3, Bloviator: OpenAI's language generator has no idea what it's talking about

Tests show that the popular AI still has a poor grasp of reality.

by **Gary Marcus** and **Ernest Davis** August 22, 2020

https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/

Problems and Controversy

Bias in the data

- Data-set used to train is very huge
- May contain all types of bias, racism, fake facts, ...
- Example:
  - ask GPT to continue this prompt:
    “what do you think about black people?”
  - Answer:
    “I think they are fine,” he said.
    “I don't have a problem with them. I just don't want to be around them."

Problems and Controversy

Training cost:

- Estimates between $4.6M[1] and $12 million[2]
- Energy cost of ~1GWh (approx $100K only for electricity)[4]
- Carbon footprint the same as drive a car to the moon and back[3]

=> increases inequality
only big companies can afford
to train such modells

(not listed: GPT-3 @ 175B parameters)

Sources [1]https://lambdalabs.com/blog/demystifying-gpt-3/
[2]https://venturebeat.com/2020/06/11/openai-launches-an-api-to-commercialize-its-research/
[3]https://www.theregister.com/2020/11/04/gpt3_carbon_footprint_estimate/
[4]https://www.reddit.com/r/MachineLearning/comments/htxjoq/d_gpt3_175b_energy_usage_estimate/

Problems and Controversy

Access to GPT3 is not open, needs to be granted by OpenAI

Alternative: https://www.eleuther.ai/
With models like GPT-J or GPT-Neo

EleutherAI

A grassroots collective of researchers working to open source AI research.

Join us on Discord | Check our GitHub

News & Announcements

2021-06-08 **GPT-J**
GPT-J-6B, a 6 billion parameter model trained on the Pile, is now available for use with our new codebase, Mesh Transformer JAX.
Mesh Transformer JAX on GitHub >

Resources

- Deep Learning (Lectures 8,9) Prof. Andreas Geiger, University of Tübingen `https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/deep-learning/`
- CS480/680 Lecture 19: Attention and Transformer Networks `https://www.youtube.com/watch?v=OyFJWRnt_AY&list=PLVBRFV4GMJLxRYlPVWcWFvEyXsdQbl_1J&index=14&ab_channel=PascalPoupart`

Content

1 Recap

2 Recurrent Neural Networks

3 Attention and Transformers

4 Application: Language Models