CS6120: Lecture 4

Kenneth Church

https://kwchurch.github.io

Agenda

- Homework
  - Assignment 2: HuggingFace Pipelines
- Background Material
- Old Business
  - Colab
  - Deep Nets: Inference
    - Classification & Regression
    - Anything → Vector
    - Machine Translation
    - Fill Mask
- New Business
  - bertviz
  - Deep Nets: Fine-Tuning
    - Easy: inference
    - Hard: pre-training
    - Not too hard: fine-tuning
  - Code: colab
    - HuggingFace Tutorial
    - HuggingFace Colab
    - run glue example

https://gluebenchmark.com/

GLUE

NYU ML2 UWNLP DeepMind

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems. GLUE consists of:

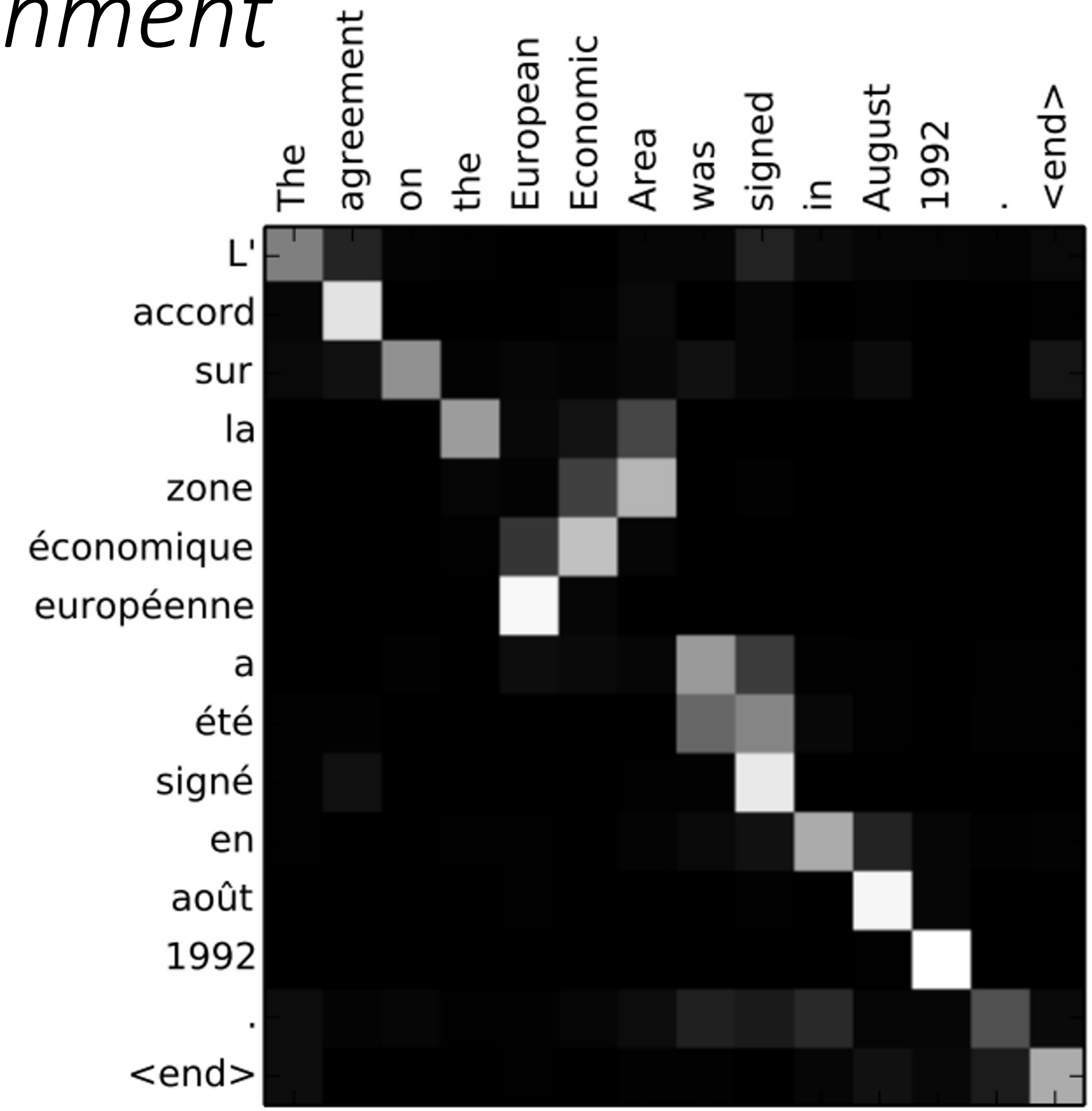
- A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty,
- A diagnostic dataset designed to evaluate and analyze model performance with respect to a wide range of linguistic phenomena found in natural language, and
- A public leaderboard for tracking performance on the benchmark and a dashboard for visualizing the performance of models on the diagnostic set.

The format of the GLUE benchmark is model-agnostic, so any system capable of processing sentence and sentence pairs and producing corresponding predictions is eligible to participate. The benchmark tasks are selected so as to favor models that share information across tasks using parameter sharing or other transfer learning techniques. The ultimate goal of GLUE is to drive research in the development of general and robust natural language understanding systems.

PAPER | STARTER CODE | GROUP | DIAGNOSTICS

Attention used to be called *Alignment*

- Attention is just another learned parameter, a number for each encoder hidden layer for each decoding step, softmaxed to be between 0 and 1
- At each decoding step, concatenate the current hidden layer with a combination of the attention-multiplied encoder hidden states, and pass this vector into a feedforward layer to get the output word at this step



From slides from last year

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Some BERT Models https://github.com/google-research/bert

- **BERT-Large, Uncased (Whole Word Masking)**:
  - 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Large, Cased (Whole Word Masking)**:
  - 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Base, Uncased**:
  - 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Large, Uncased**:
  - 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Base, Cased**:
  - 12-layer, 768-hidden, 12-heads , 110M parameters
- **BERT-Large, Cased**:
  - 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Base, Multilingual Cased (New, recommended)**:
  - 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters

Smaller Models
https://github.com/google-research/bert

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-Base)

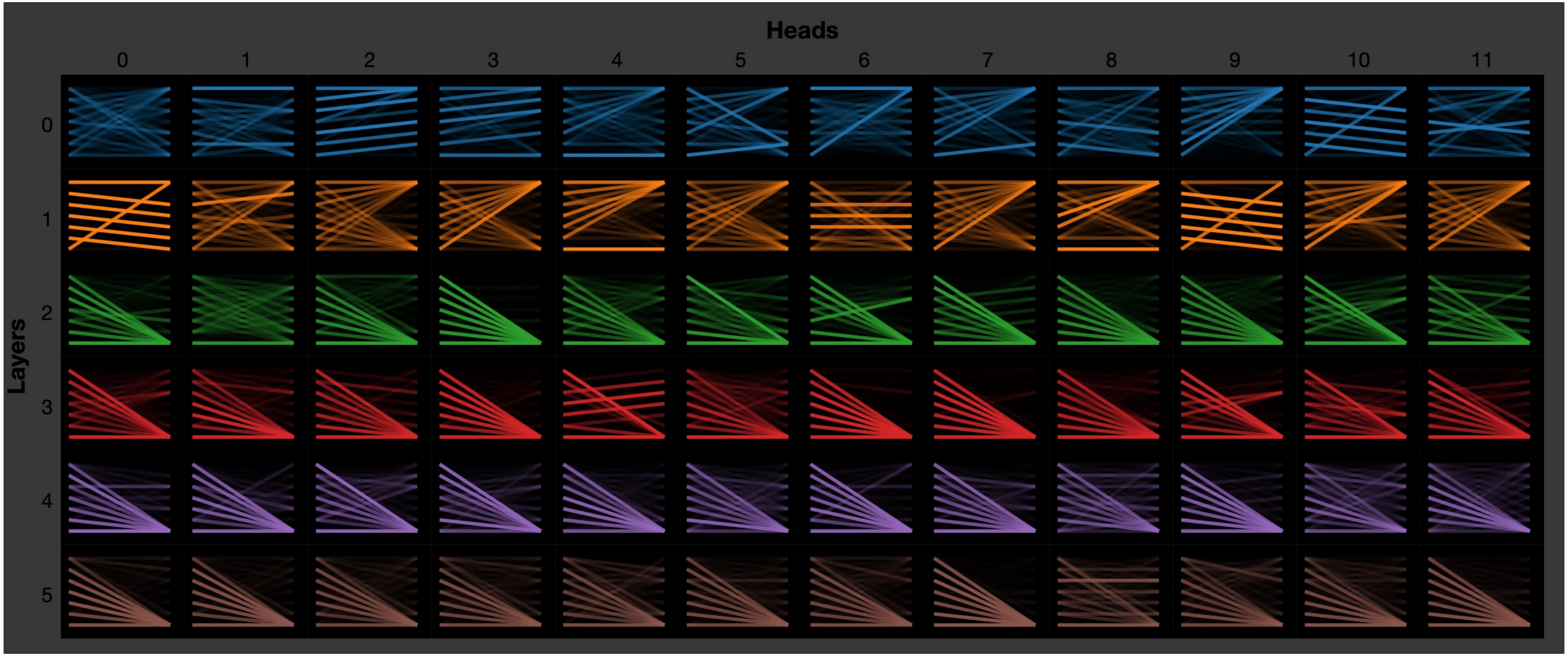
Note that the BERT-Base model in this release is included for completeness only; it was re-trained under the same regime as the original model.

Here are the corresponding GLUE scores on the test set:

Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI(v2)
BERT-Tiny	64.2	0.0	83.2	81.1/71.1	74.3/73.6	62.2/83.4	70.2	70.3	81.5
BERT-Mini	65.8	0.0	85.9	81.1/71.8	75.4/73.3	66.4/86.2	74.8	74.3	84.1
BERT-Small	71.2	27.8	89.7	83.4/76.2	78.8/77.0	68.1/87.0	77.6	77.0	86.4
BERT-Medium	73.5	38.0	89.6	86.6/81.6	80.4/78.4	69.6/87.9	80.0	79.1	87.7

- Trade-off: Costs v. Performance
- Costs:
  - Space
  - Time
  - Power
- Industry Practice
  - Train large models and then make them smaller (distillation)

What do we mean by layers, heads, etc.?
[colab]



A Multiscale Visualization of Attention in the Transformer Model (bertviz) https://aclanthology.org/P19-3007.pdf

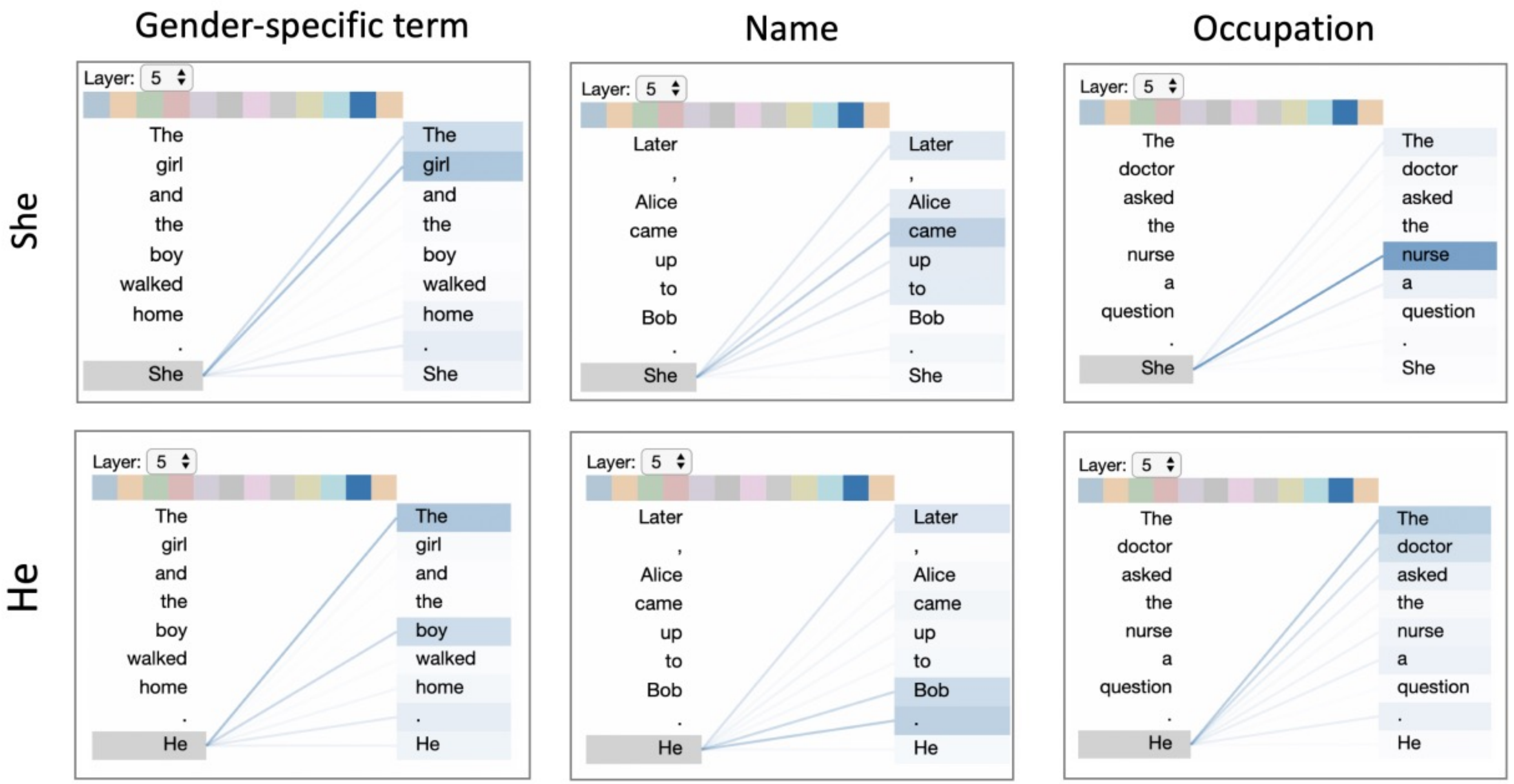
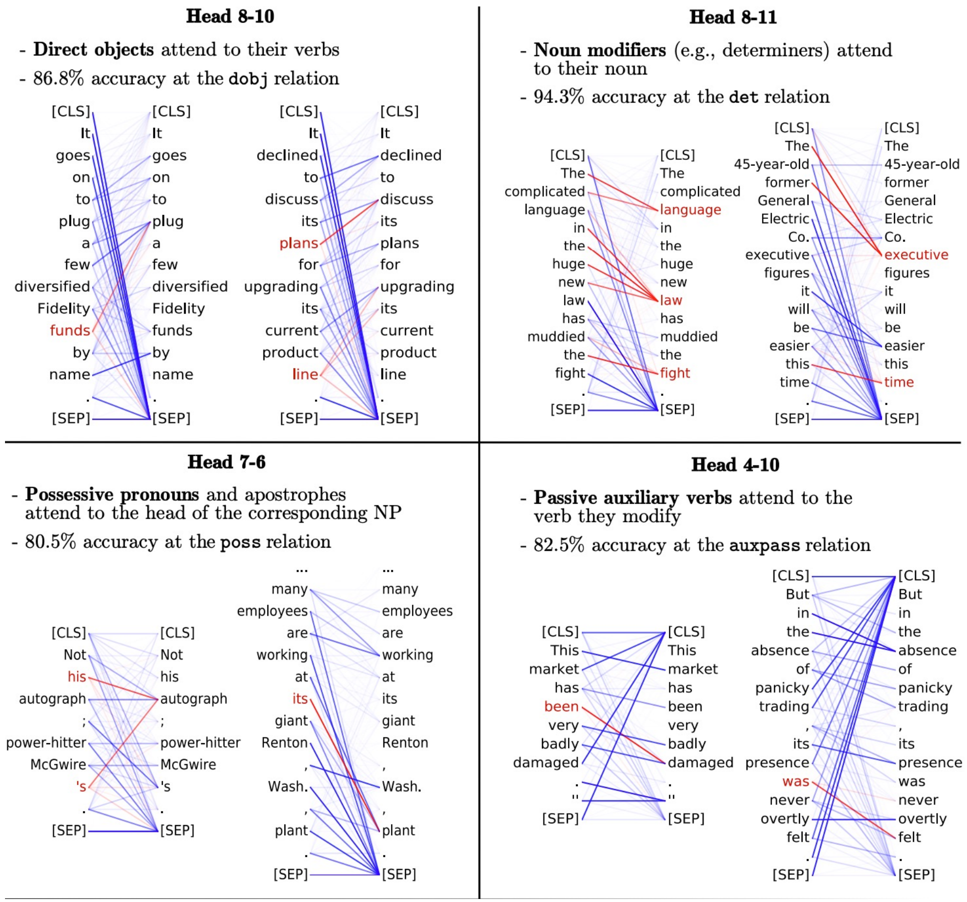


Figure 4: Attention pattern in GPT-2 related to coreference resolution suggests the model may encode gender bias.

https://arxiv.org/pdf/1906.04341.pdf



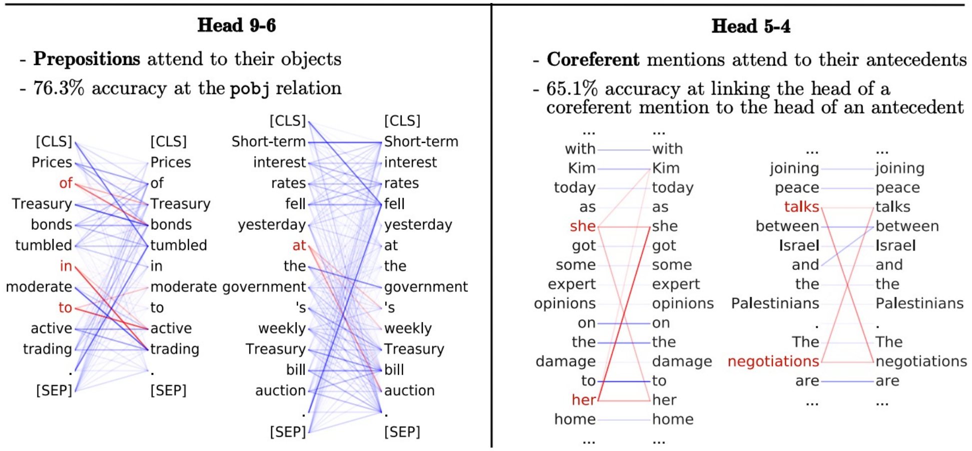


Figure 5: BERT attention heads that correspond to linguistic phenomena. In the example attention maps, the darkness of a line indicates the strength of the attention weight. All attention to/from red words is colored red; these colors are there to highlight certain parts of the attention heads' behaviors. For Head 9-6, we don't show attention to [SEP] for clarity. Despite not being explicitly trained on these tasks, BERT's attention heads perform remarkably well, illustrating how syntax-sensitive behavior can emerge from self-supervised training alone.

What Does BERT Look At?
An Analysis of BERT's Attention

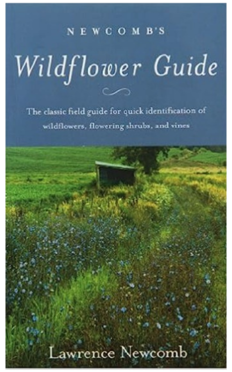
Kevin Clark[†] **Urvashi Khandelwal**[†] **Omer Levy**[‡] **Christopher D. Manning**[†]
[†]Computer Science Department, Stanford University
[‡]Facebook AI Research
{kevclark,urvashik,manning}@cs.stanford.edu
omerlevy@fb.com

Review

Standard 3-Step Recipe

Step	**Description**	**Time**	**Hardware**
1	Pre-Training	Days/Weeks	Large GPU cluster
2 Not Hard	Fine-Tuning (fit)	Hours/Days	1+ GPUs
3	Inference (predict)	Seconds/Minutes	0+ GPUs

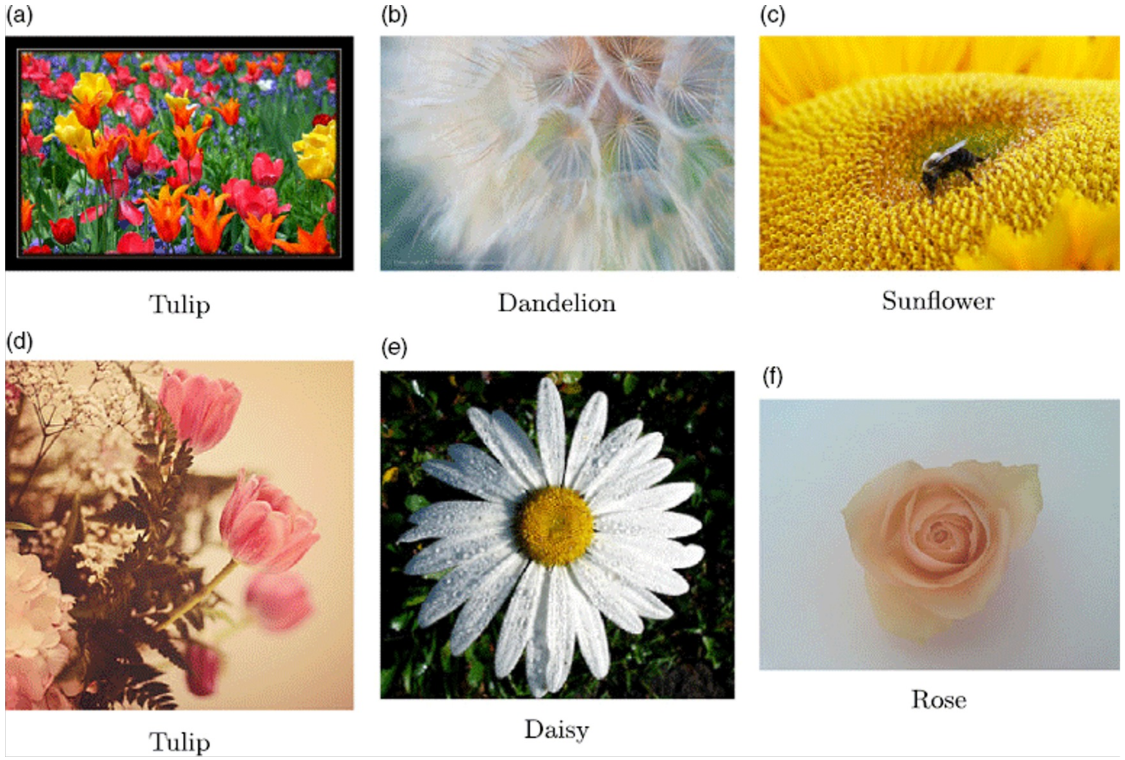
Guide Book App



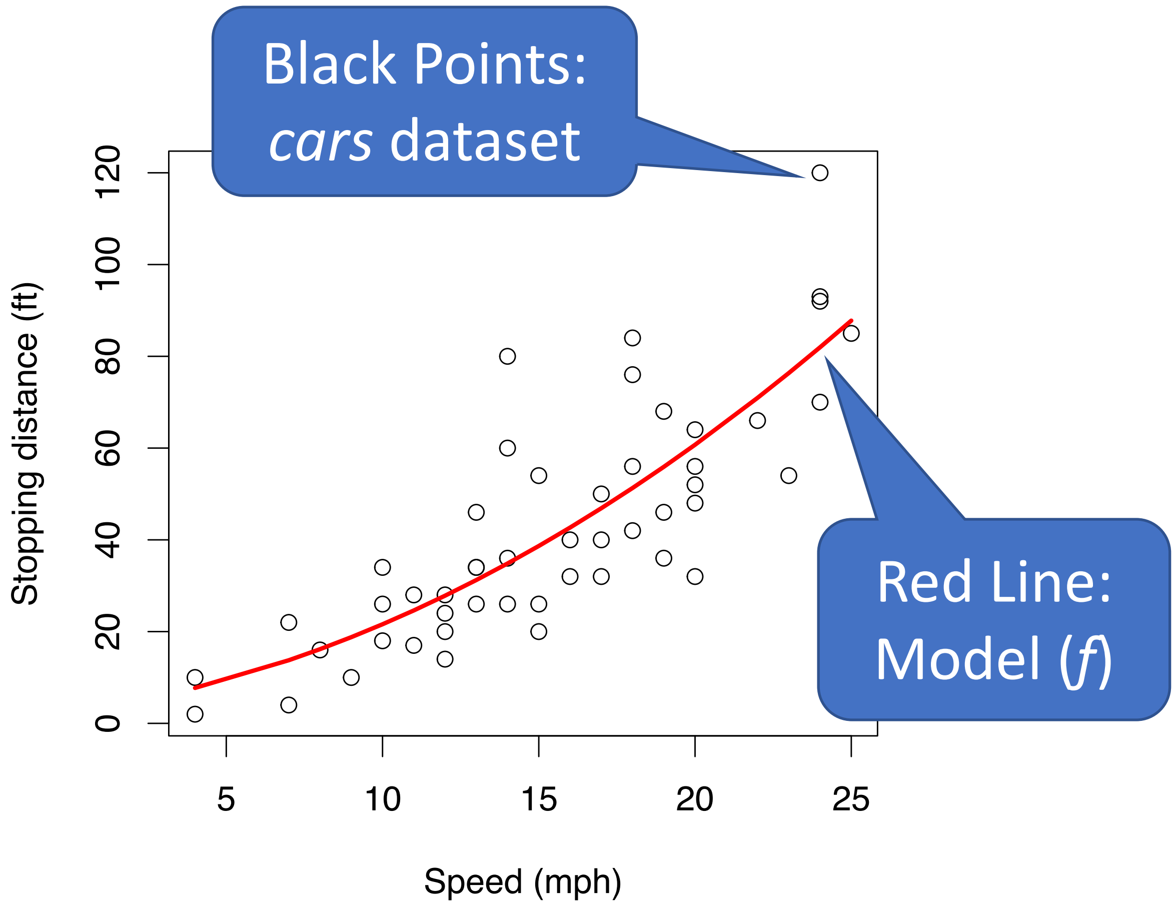
Example of Fine-Tuning (aka, *fit*)

fit: $f_{pre} + data \rightarrow f_{post}$

- f_{pre}: Resnet
  - Maps images (jpg files) → classes (strings)
  - Trained on ImageNet
    - input (x): 14M images (of many things)
    - output (y): 1000 classes (strings)
- data : flowers
  - input (x): 2195 pictures of flowers
  - outputs (y): 5 classes of flowers
    - *Tulip, Dandelion, Sunflower, Daisy, Rose*
- f_{post}:
  - Maps images (jpg files) → flowers (strings)
  - Reject modeling is hard



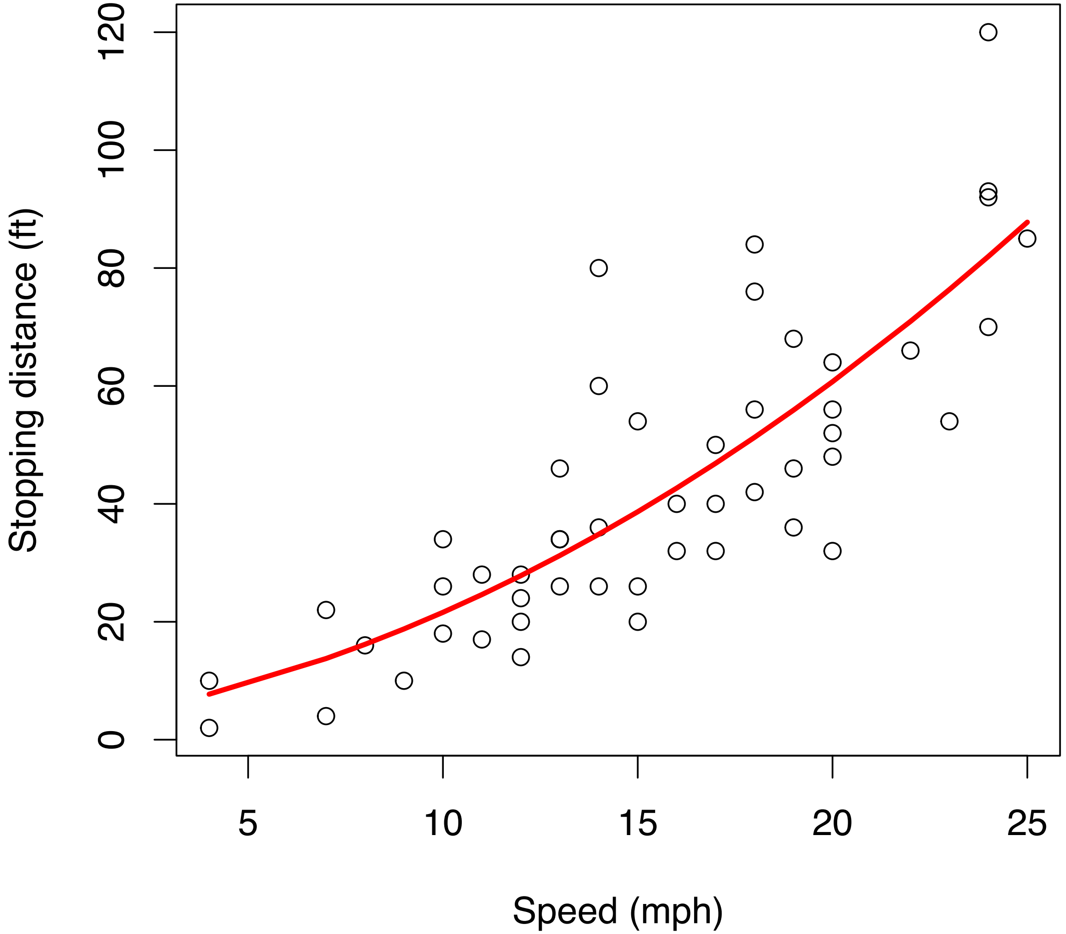
Fine-Tuning (Fit) in R (Statistics Package)



Make deep nets look like regression
(Not much programming)

- Notational Conventions:
  - Observations: Circles
  - Models (f): Red Lines
- Prediction: $f(x)$
  - Use model (f) to map
    - input x (speed) to
    - output y (stopping distance)
  - For linear regression,
    - f is a polynomial
  - For gft,
    - f is typically a model from a hub

Datasets
Example: Cars



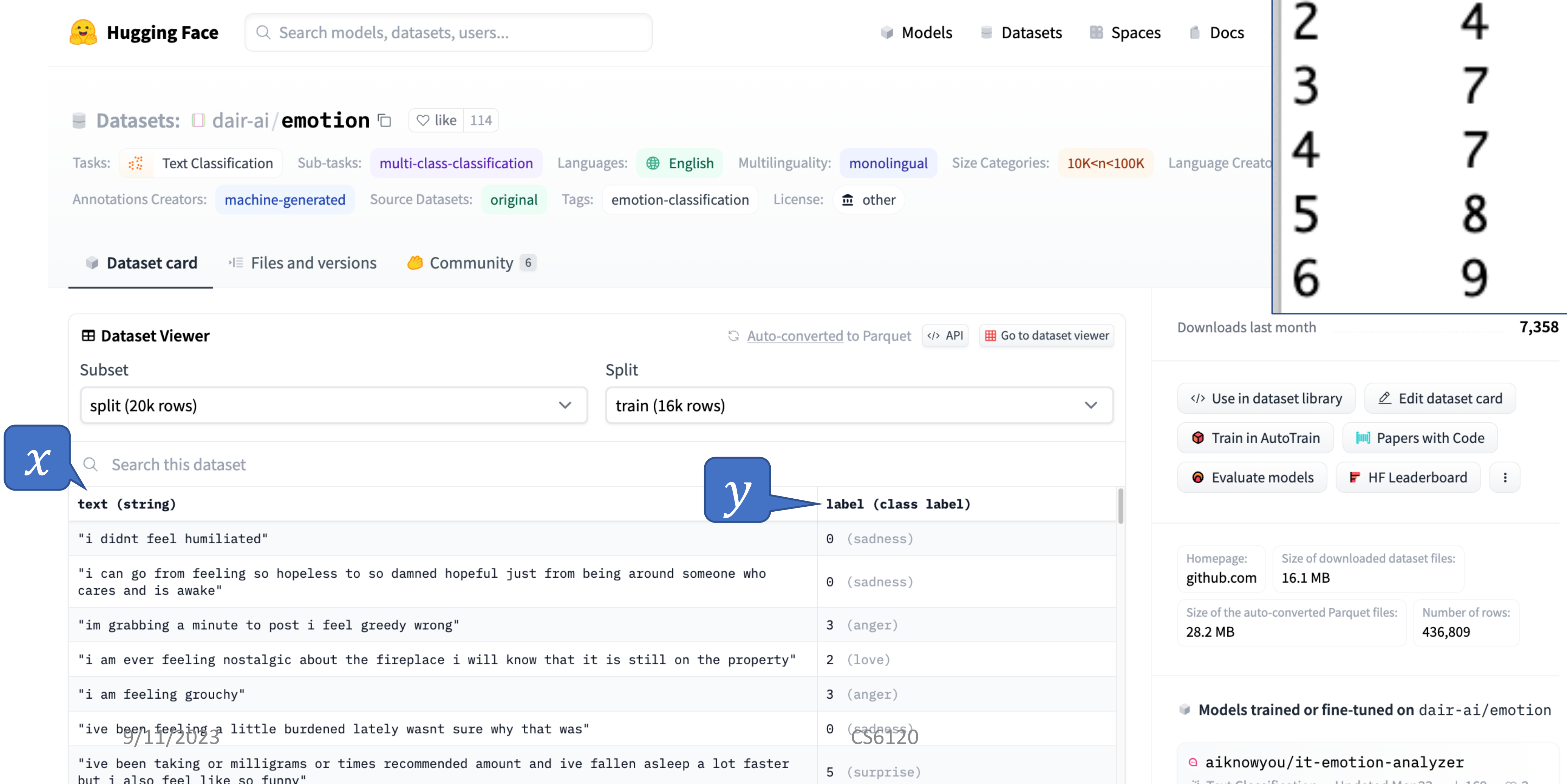

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

Two Columns:
1. cars$speed
2. cars$dist

Example of Datasets in HuggingFace

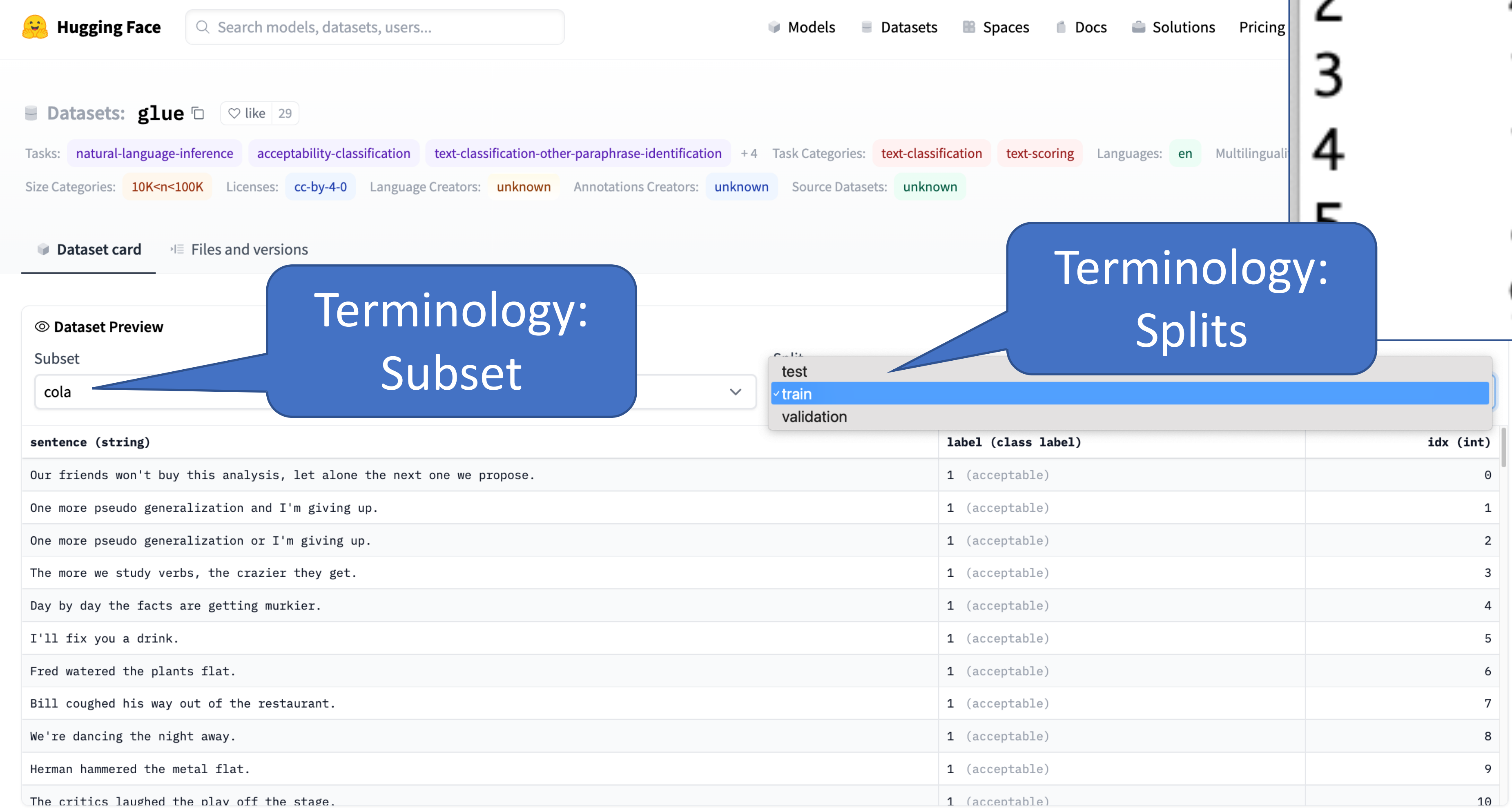
https://huggingface.co/datasets/dair-ai/emotion

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

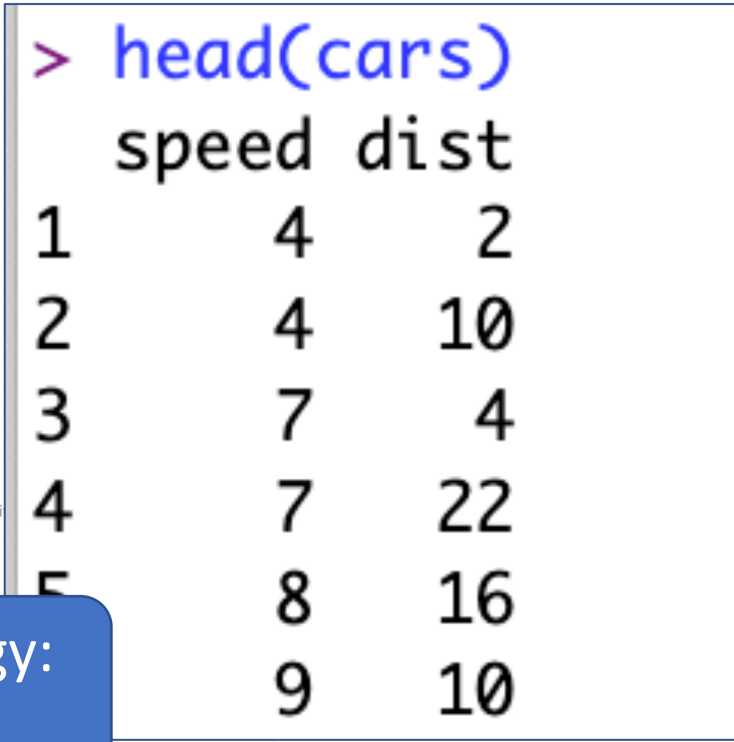


Emotion → GLUE (Cola)

https://huggingface.co/datasets/glue/viewer/cola/train



sentence (string)	label (class label)	idx (int)
Our friends won't buy this analysis, let alone the next one we propose.	1 (acceptable)	0
One more pseudo generalization and I'm giving up.	1 (acceptable)	1
One more pseudo generalization or I'm giving up.	1 (acceptable)	2
The more we study verbs, the crazier they get.	1 (acceptable)	3
Day by day the facts are getting murkier.	1 (acceptable)	4
I'll fix you a drink.	1 (acceptable)	5
Fred watered the plants flat.	1 (acceptable)	6
Bill coughed his way out of the restaurant.	1 (acceptable)	7
We're dancing the night away.	1 (acceptable)	8
Herman hammered the metal flat.	1 (acceptable)	9
The critics laughed the play off the stage.	1 (acceptable)	10



```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

Datasets in HuggingFace colab

Datasets in HuggingFace.ipynb

File Edit View Insert Runtime Tools Help Last edited on September 30

Comment Share

+ Code + Text Connect

Compare with https://huggingface.co/datasets/imdb The IMDB dataset has three splits. Each split has two columns (features).

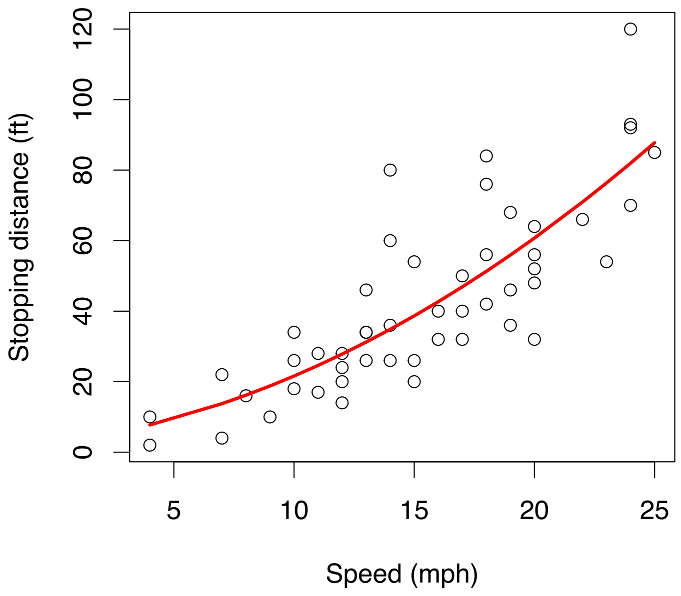
```
for split in imdb:
    print(split + '\t' + str(imdb[split]))
```

```
train   Dataset({
    features: ['text', 'label'],
    num_rows: 25000
})
test    Dataset({
    features: ['text', 'label'],
    num_rows: 25000
})
unsupervised    Dataset({
    features: ['text', 'label'],
    num_rows: 50000
})
```

```
imdb['train']['text'][0]
```

```
'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in
1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of f
ilms considered "controversial" I really had to see this for myself.<br /><br />The plot is centered around a young Swedish drama
student named Lena who wants to learn everything she can about life. In particular she wants to focus her attentions to making so
me sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues i
n the United States. In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has s
```

glm (General Linear Models) in R (and Sklearn)




```
# Create the black points
plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
g = glm(dist ~ poly(speed, 2), data=cars)
o = order(cars$speed)
# Show predictions as a red line
lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```

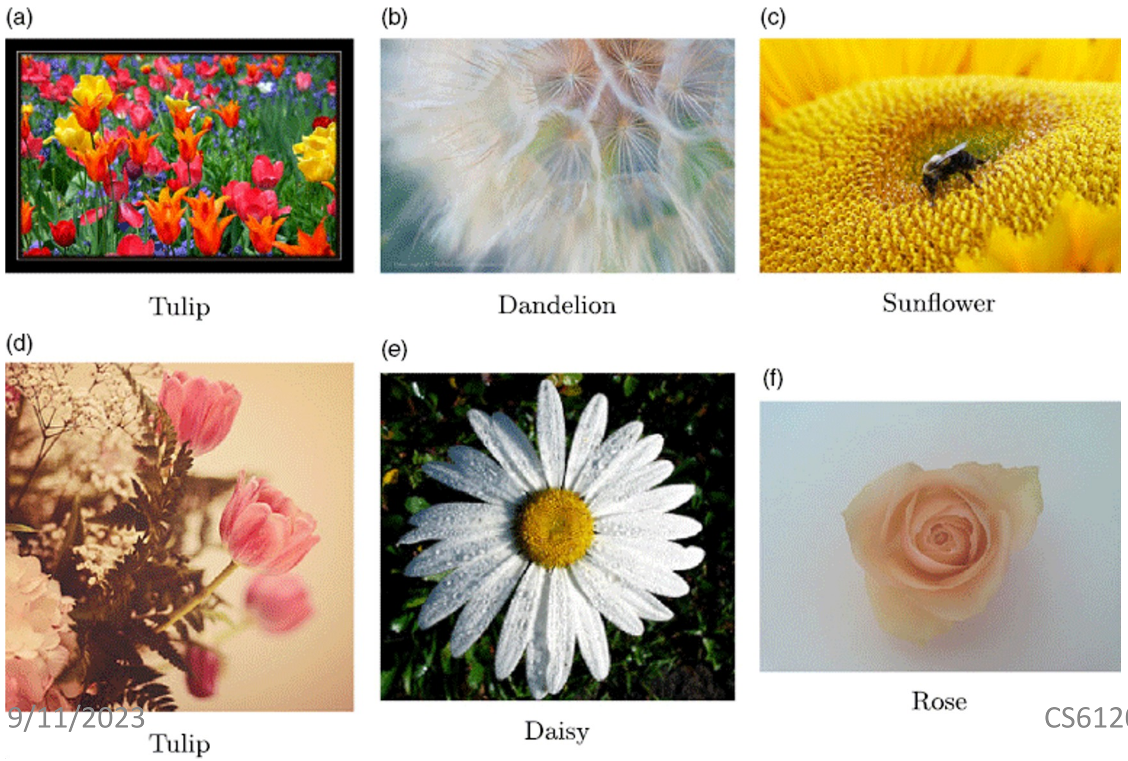
glm: fit poly model with data

predict: $dist \approx g(cars\$speed)$

Fine-Tuning (fit) in GFT

$$f_{pre} + data \rightarrow f_{post}$$

- Flowers
  - f_{pre}: Resnet
  - $data$: flowers




```
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

f_{pre}: Pre-trained Model

f_{post}: Post-trained Model

```
g = glm(dist ~ poly(speed, 2), data=cars)
```

Fine-Tuning (fit) in R

Emotion Dataset

$$classify: label \sim text$$

Task/Loss

Columns

Row

Hugging Face — Search models, datasets, users...

Datasets: emotion like 18

Tasks: multi-class-classification text-classification-other-emotion-classification Task Categories: text-classification

Language Creators: machine-generated Annotations Creators: machine-generated Source Datasets: original

Dataset card Files and versions

Dataset Preview

Subset: default

Split: train

text (string)	label (class label)
i didnt feel humiliated	0 (sadness)
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake	0 (sadness)
im grabbing a minute to post i feel greedy wrong	3 (anger)
i am ever feeling nostalgic about the fireplace i will know that it is still on the property	2 (love)
i am feeling grouchy	3 (anger)
ive been feeling a little burdened lately wasnt sure why that was	0 (sadness)
ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny	5 (surprise)
i feel as confused about life as a teenager or as jaded as a year old man	4 (fear)
i have been with petronas for years i feel that petronas has performed well and made a huge profit	1 (joy)
i feel romantic too	2 (love)
i feel like i have to make the suffering i m seeing mean something	0 (sadness)

Fine-Tuning (fit): Numerous Use Cases

$$f_{pre} + data \rightarrow f_{post}$$

- Flowers
  - f_{pre}: Resnet
  - $data$: flowers
- Emotion Classification
  - f_{pre}: https://huggingface.co/bert-base-uncase
  - $data$: https://huggingface.co/datasets/dair – ai/emotion
- GLUE
- SQuAD
- Machine Translation
- Speech Recognition
- Vision
- and much more

```
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

f_{pre}: Pre-trained Model

f_{post}: Post-trained Model

```
7 g = glm(dist ~ poly(speed, 2), data=cars)
```

Fine-Tuning (fit) in R

GLUE Subsets

Subset	Dataset
COLA	H:glue,cola
SST2	H:glue,sst2
WNLI	H:glue,wnli
MRPC	H:glue,mrpc
QNLI	H:glue,qnli
QQP	H:glue,qqp
SSTB	H:glue,sstb
MNLI	H:glue,mnli

GLUE COLA SUBSET	
Sentence	**Label**
Bill sang himself to sleep.	1 (acceptable)
Bill squeezed the puppet through the hole.	1 (acceptable)
Bill sang Sue to sleep.	1 (acceptable)
The elevator rumbled itself to the ground.	0 (unacceptable)
If the telephone rang, it could ring itself silly.	1 (acceptable)
She yelled hoarse.	0 (unacceptable)
Ted cried to sleep.	0 (unacceptable)
The tiger bled to death.	1 (acceptable)

GFT Program for COLA

https://github.com/kwchurch/gft/blob/master/examples/fit_examples/model.HuggingFace/language/data.HuggingFace/glue/cola.sh

```
#!/bin/sh

echo hostname = `hostname`

gft_fit --model H:bert-base-cased \
    --data H:glue,cola \
    --metric H:glue,cola \
    --figure_of_merit matthews_correlation \
    --output_dir $1 \
    --eqn 'classify: label ~ sentence' \
    --num_train_epochs 3
```

GLUE COLA SUBSET

Sentence	Label
Bill sang himself to sleep.	1 (acceptable)
Bill squeezed the puppet through the hole.	1 (acceptable)
Bill sang Sue to sleep.	1 (acceptable)
The elevator rumbled itself to the ground.	0 (unacceptable)
If the telephone rang, it could ring itself silly.	1 (acceptable)
She yelled hoarse.	0 (unacceptable)
Ted cried to sleep.	0 (unacceptable)
The tiger bled to death.	1 (acceptable)

Simple Equations Cover Many Cases of Interest
GLUE: A Popular Benchmark

Subset	Dataset	Equation
COLA	H:glue,cola	$classify:\ label \sim sentence$
SST2	H:glue,sst2	$classify:\ label \sim sentence$
WNLI	H:glue,wnli	$classify:\ label \sim sentence$
MRPC	H:glue,mrpc	$classify:\ label \sim sentence1\ +\ sentence2$
QNLI	H:glue,qnli	$classify:\ label \sim sentence1\ +\ sentence2$
QQP	H:glue,qqp	$classify:\ label \sim question + sentence$
SSTB	H:glue,sstb	$\boxed{regress}:\ label \sim question1 + question2$
MNLI	H:glue,mnli	$classify:\ label \sim premise + hypothesis$

Equation Keywords ≈ Pipeline Tasks

Benchmark	Subset	Dataset	Equation
GLUE	COLA	H:glue,cola	$classify:\ label \sim sentence$
SQuAD 1.0		H:squad	$\boxed{classify\_spans}: answers{\sim}question + context$
SQuAD 2.0		H:squad_v2	$classify\_spans: answers{\sim}question + context$
CONLL2003	POS	H:conll2003	$\boxed{classify\_tokens}: pos\_tags{\sim}tokens$
	NER	H:conll2003	$classify\_tokens: ner\_tags{\sim}tokens$
TIMIT		H:timit_asr	$\boxed{ctc}: text{\sim}audio$
Amazon Reviews		H:amazon_reviews_multi	$classify:\ label \sim question + sentence$
VAD		C:$gft/datasets/VAD/VAD	$regress: Valence\ +\ Arousal\ +\ Dominance \sim Word$

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data

✓ --model

➢ **--eqn** $\boldsymbol{task}: \boldsymbol{y} \sim \boldsymbol{x_1} + \boldsymbol{x_2}$

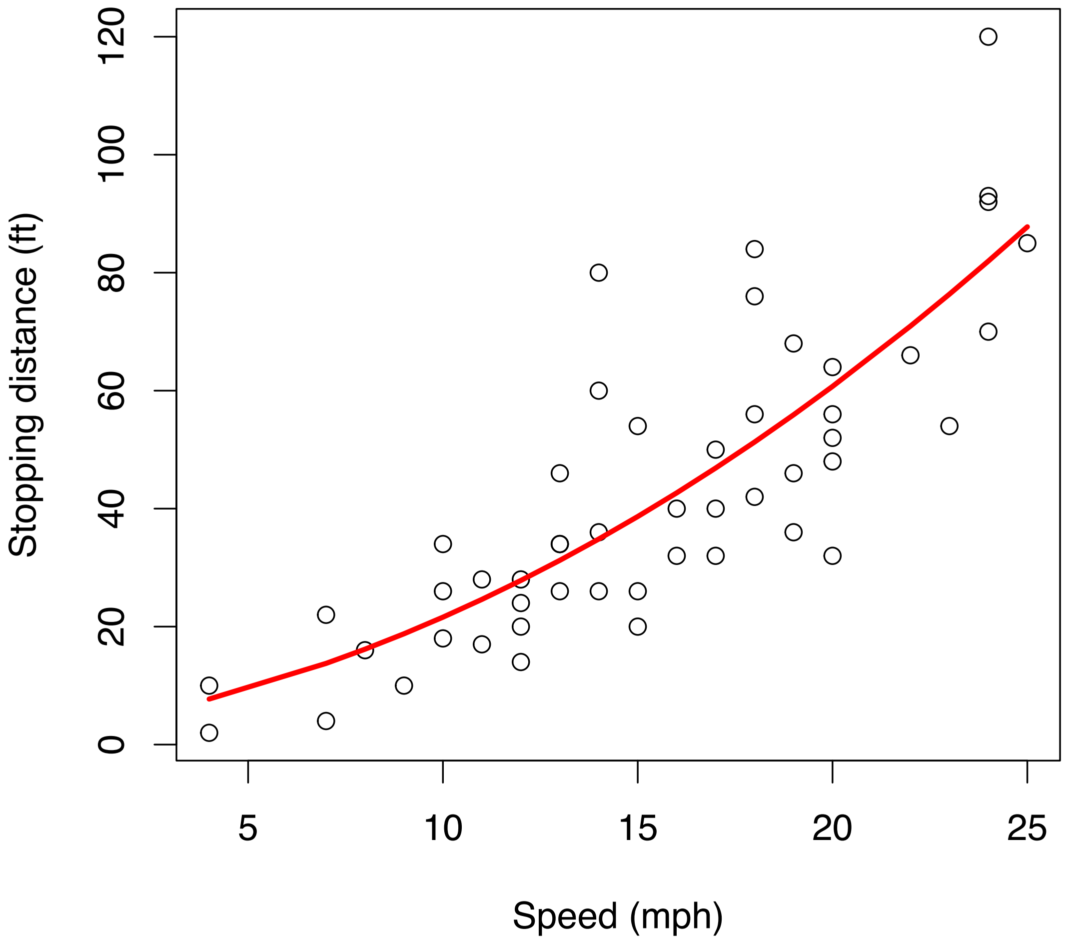
- **--task**
  1. classify (text-classification)
  2. classify_tokens (token-classification)
  3. classify_spans (QA, question-answering)
  4. classify_audio (audio-classification)
  5. classify_images (image-classification)
  6. regress
  7. text-generation
  8. MT (translation)
  9. ASR (ctc, automatic-speech-recognition)
  10. fill-mask

Fit a model (*g*) to data (*cars*)

Regression in *R*:

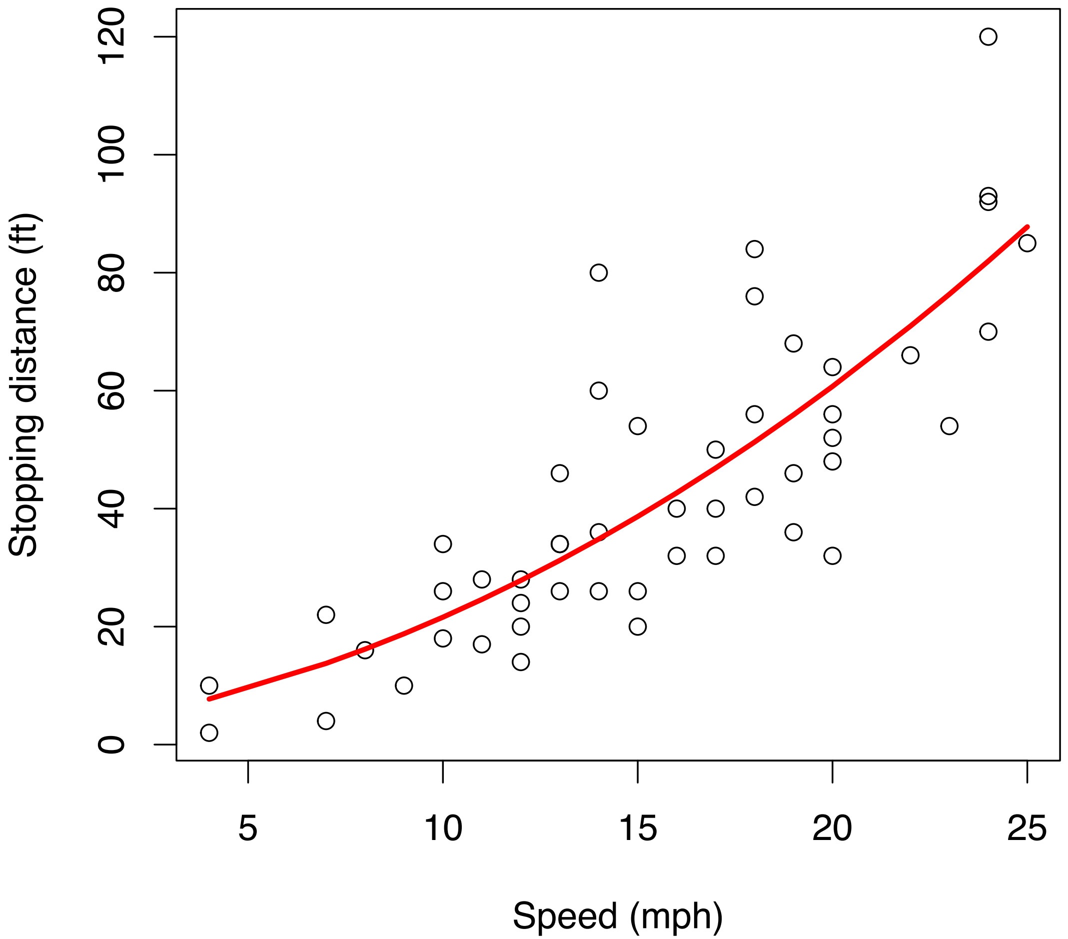
Summarize (almost) anything

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```



Regression in *R*:

Summarize (almost) anything

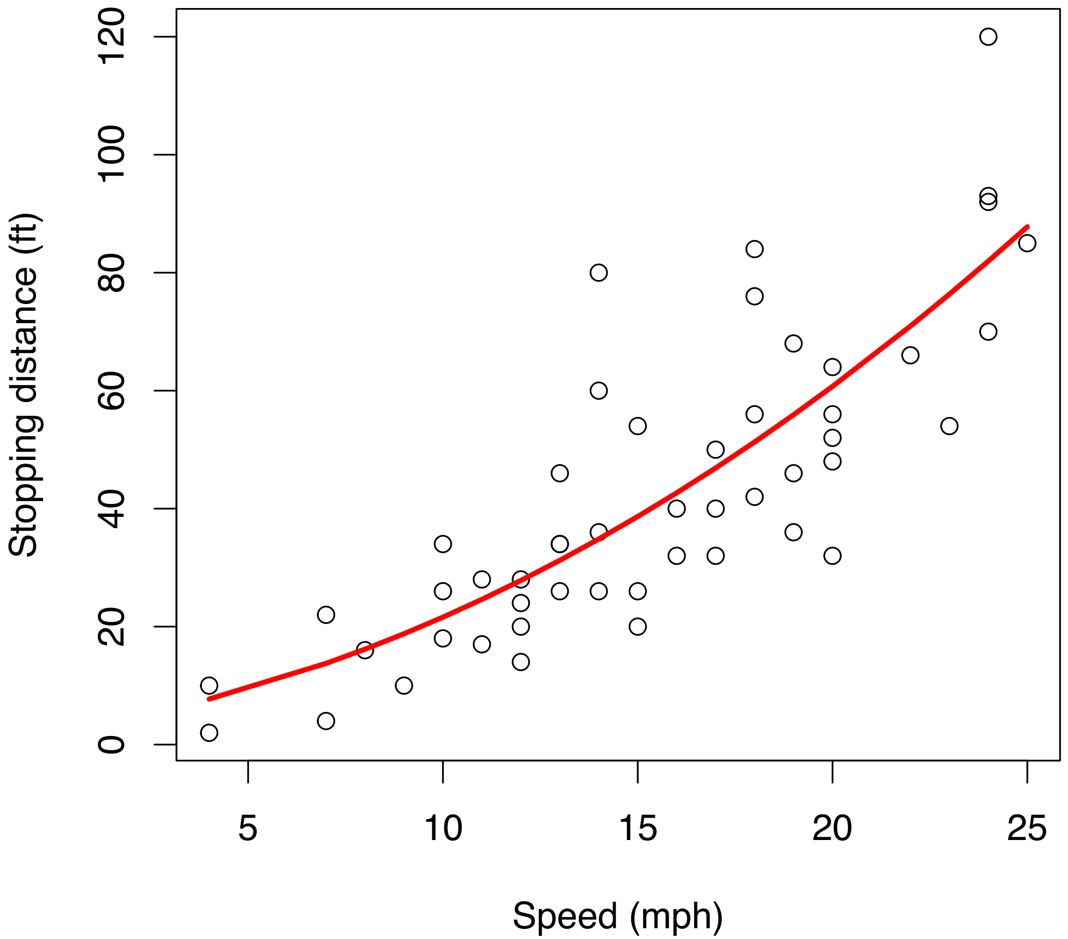



```
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
```

Regression in *R*:

Summarize (almost) anything

Fit a model (*g*) to data (*cars*)




```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
> summary(g)

Call:
glm(formula = dist ~ poly(speed, 2), data = cars)

Deviance Residuals:
    Min        1Q    Median        3Q       Max
-28.720    -9.184    -3.188     4.628    45.152

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       42.980      2.146  20.026  < 2e-16 ***
poly(speed, 2)1  145.552     15.176   9.591 1.21e-12 ***
poly(speed, 2)2   22.996     15.176   1.515    0.136
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 230.3131)

    Null deviance: 32539  on 49  degrees of freedom
Residual deviance: 10825  on 47  degrees of freedom
AIC: 418.77

Number of Fisher Scoring iterations: 2
```

Predict

Summarize a model (*g*)

Opportunity to Improve *g*

gft_summary

- Summarize almost anything
  - Models
  - Datasets

Tasks

- classify, text-classification — $y \in \{0,1,2,\ldots\}$
- regress — $y \in \mathbb{R}$ or $y \in \mathbb{R}^N$
- QA, Question Answering, classify spans — y for each start/end of span
- token classification — y for each token
  - NER (Named Entity Recognition)
  - POS (Part of Speech Tagging)
- translation, MT
- ASR, Automatic Speech Recognition, ctc — y for each phoneme

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
    - 4 Arguments, --output_dir, --metric, --splits
    - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
    - Input: 4 Arguments (x from data or stdin)
    - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
    - Input: 4 Arguments, --split, --metric, …
    - Output: Score
4. gft_summary: Find good stuff
    - Input: 4 Arguments
    - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
    - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data
✓ --model
✓ --eqn $task: y \sim x_1 + x_2$
✓ **--task**

1. classify (text-classification)
2. classify_tokens (token-classification)
3. classify_spans (QA, question-answering)
4. classify_audio (audio-classification)
5. classify_images (image-classification)
6. regress
7. text-generation
8. MT (translation)
9. ASR (ctc, automatic-speech-recognition)
10. fill-mask

HuggingFace run_glue.py colab

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/README.md

GFT

Subset	Dataset	Equation
COLA	H:glue,cola	$classify: label \sim sentence$
SST2	H:glue,sst2	$classify: label \sim sentence$
WNLI	H:glue,wnli	$classify: label \sim sentence$
MRPC	H:glue,mrpc	$classify: label \sim sentence1 + sentence2$
QNLI	H:glue,qnli	$classify: label \sim sentence1 + sentence2$
QQP	H:glue,qqp	$classify: label \sim question + sentence$
SSTB	H:glue,sstb	$regress: label \sim question1 + question2$
MNLI	H:glue,mnli	$classify: label \sim premise + hypothesis$

Text classification examples

GLUE tasks

Based on the script `run_glue.py`.

Fine-tuning the library models for sequence classification on the GLUE benchmark: General Language Understanding Evaluation. This script can fine-tune any of the models on the hub and can also be used for a dataset hosted on our hub or your own data in a csv or a JSON file (the script might need some tweaks in that case, refer to the comments inside for help).

GLUE is made up of a total of 9 different tasks. Here is how to run the script on one of them:

```
export TASK_NAME=mrpc

python run_glue.py \
  --model_name_or_path bert-base-cased \
  --task_name $TASK_NAME \
  --do_train \
  --do_eval \
  --max_seq_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/$TASK_NAME/
```

where task name can be one of cola, sst2, mrpc, stsb, qqp, mnli, qnli, rte, wnli.

HuggingFace run_glue.py colab

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/README.md

GFT

Subset	Dataset	Equation
COLA	H:glue,cola	$classify:\ label \sim sentence$
SST2	H:glue,sst2	$classify:\ label \sim sentence$
WNLI	H:glue,wnli	$classify:\ label \sim sentence$
MRPC	H:glue,mrpc	$classify:\ label \sim sentence1\ +\ sentence2$
QNLI	H:glue,qnli	$classify:\ label \sim sentence1\ +\ sentence2$
QQP	H:glue,qqp	$classify:\ label \sim question + sentence$
SSTB	H:glue,sstb	$regress:\ label \sim question1 + question2$
MNLI	H:glue,mnli	$classify:\ label \sim premise + hypothesis$

```python
task_to_keys = {
    "cola": ("sentence", None),
    "mnli": ("premise", "hypothesis"),
    "mrpc": ("sentence1", "sentence2"),
    "qnli": ("question", "sentence"),
    "qqp": ("question1", "question2"),
    "rte": ("sentence1", "sentence2"),
    "sst2": ("sentence", None),
    "stsb": ("sentence1", "sentence2"),
    "wnli": ("sentence1", "sentence2"),
}
```

Trainer

https://huggingface.co/learn/nlp-course/chapter3/3?fw=pt

```python
from datasets import load_dataset
from transformers import AutoTokenizer, DataCollatorWithPadding

raw_datasets = load_dataset("glue", "mrpc")
checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)


def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)


tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```python
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

```python
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```python
trainer.train()
```

Steps

- Datasets
  - colab
  - contains splits: train, validation, test
- Tokenizing
  - colab
  - maps input text to a sequence of tokens
    - where token is an offset into vocabulary
- Model: f
  - Includes tokenizer
  - Typically based on BERT
- Trainer
  - colab
  - For each epoch (pass over training split)
    - Train (update f, using stochastic gradient descent)
    - Evaluate (score f on validation split)

```python
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

```python
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```python
trainer.train()
```

https://en.wikipedia.org/wiki/Stochastic_gradient_descent
https://realpython.com/gradient-descent-algorithm-python

Stochastic gradient descent

Article Talk

From Wikipedia, the free encyclopedia

Stochastic gradient descent (often abbreviated **SGD**) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the very high computational burden, achieving faster iterations in exchange for a lower convergence rate.[1]

While the basic idea behind stochastic approximation can be traced back to the Robbins–Monro algorithm of the 1950s, stochastic gradient descent has become an important optimization method in machine learning.[2]

Background [edit]

Main article: M-estimation

See also: Estimating equation

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where the parameter w that minimizes $Q(w)$ is to be estimated. Each summand function Q_i is typically associated with the i-th observation in the data set (used for training).

https://en.wikipedia.org/wiki/Newton%27s_method

Newton's method

40 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about Newton's method for finding roots. For Newton's method for finding minima, see Newton's method in optimization.

In numerical analysis, **Newton's method**, also known as the **Newton–Raphson method**, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a real-valued function f, its derivative f', and an initial guess x_0 for a root of f. If f satisfies certain assumptions and the initial guess is close, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is a better approximation of the root than x_0. Geometrically, $(x_1, 0)$ is the x-intercept of the tangent of the graph of f at $(x_0, f(x_0))$: that is, the improved guess, x_1, is the unique root of the linear approximation of f at the initial guess, x_0. The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently precise value is reached. The number of correct digits roughly doubles with each step. This algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

Description [edit]

The idea is to start with an initial guess, then to approximate the function by its tangent line, and finally to compute the x-intercept of this tangent line. This x-intercept will typically be a better approximation to the original function's root than the first guess, and the method can be iterated.

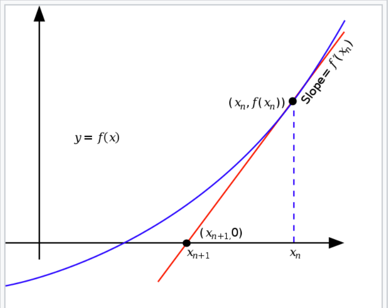
If the tangent line to the curve $f(x)$ at $x = x_n$ intercepts the x-axis at x_{n+1} then the slope is

$$f'(x_n) = \frac{f(x_n) - 0}{x_n - x_{n+1}}.$$

Solving for x_{n+1} gives

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We start the process with some arbitrary initial value x_0. (The closer to the zero, the better. But, in the absence of any intuition about where the zero might lie, a "guess and check" method might narrow the possibilities to a reasonably small interval by appealing to the intermediate value theorem.) The method will usually converge, provided this initial guess is close enough to the unknown zero, and



https://en.wikipedia.org/wiki/Hill_climbing

Hill climbing

文A 16 languages

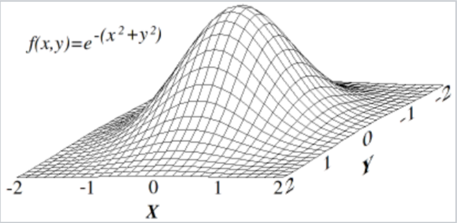
Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about the mathematical algorithm. For other meanings such as the branch of motorsport, see Hillclimbing (disambiguation).

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Hill climbing" – news · newspapers · books · scholar · JSTOR *(April 2017) (Learn how and when to remove this template message)*

In numerical analysis, **hill climbing** is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.



A surface with only one maximum. Hill-climbing techniques are well-suited for optimizing over such surfaces, and will converge to the global maximum.

For example, hill climbing can be applied to the travelling salesman problem. It is easy to find an initial solution that visits all the cities but will likely be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much shorter route is likely to be obtained.

Hill climbing finds optimal solutions for convex problems – for other problems it will find only local optima (solutions that cannot be improved upon by any neighboring configurations), which are not necessarily the best possible solution (the global optimum) out of all possible solutions (the search space). Examples of algorithms that solve convex problems by hill-climbing include the simplex algorithm for linear programming and binary search.[1]:253 To attempt to avoid getting stuck in local optima, one could use restarts (i.e. repeated local search), or more complex schemes based on iterations (like iterated local search), or on memory (like reactive search optimization and tabu search), or on memory-less stochastic modifications (like simulated annealing).

Trainer

https://huggingface.co/learn/nlp-course/chapter3/3?fw=pt

```python
from datasets import load_dataset
from transformers import AutoTokenizer, DataCollatorWithPadding

raw_datasets = load_dataset("glue", "mrpc")
checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)


def tokenize_function(example):
    return tokenizer(example["sentence1"], example["sentence2"], truncation=True)


tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```python
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

```python
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```python
trainer.train()
```

Steps (Review)

- Datasets
  - colab
  - contains splits: train, validation, test
- Tokenizing
  - colab
  - maps input text to a sequence of tokens
    - where token is an offset into vocabulary
- Model: f
  - Includes tokenizer
  - Typically based on BERT
- Trainer
  - colab
  - For each epoch (pass over training split)
    - Train (update f, using stochastic gradient descent)
    - Evaluate (score f on validation split)

```
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

```
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```
trainer.train()
```

Metric v. Loss

- Training Step
  - Optimize loss in SGD
  - Requirement:
    - loss is differentiable
  - Users no longer need to know how to differentiate loss
    - with modern frameworks:
    - pytorch, tensorflow
- Evaluation Step
  - Use metric to score f
- Ideally, loss ≈ metric

Subset	Dataset	Equation
COLA	H:glue,cola	$classify: label \sim sentence$
SST2	H:glue,sst2	$classify: label \sim sentence$
WNLI	H:glue,wnli	$classify: label \sim sentence$
MRPC	H:glue,mrpc	$classify: label \sim sentence1 + sentence2$
QNLI	H:glue,qnli	$classify: label \sim sentence1 + sentence2$
QQP	H:glue,qqp	$classify: label \sim question + sentence$
SSTB	H:glue,sstb	$regress: label \sim question1 + question2$
MNLI	H:glue,mnli	$classify: label \sim premise + hypothesis$

Task	Metric	Result	Training time
CoLA	Matthews corr	56.53	3:17
SST-2	Accuracy	92.32	26:06
MRPC	F1/Accuracy	88.85/84.07	2:21
STS-B	Pearson/Spearman corr.	88.64/88.48	2:13
QQP	Accuracy/F1	90.71/87.49	2:22:26
MNLI	Matched acc./Mismatched acc.	83.91/84.10	2:35:23
QNLI	Accuracy	90.66	40:57
RTE	Accuracy	65.70	57
WNLI	Accuracy	56.34	24

Not much better than chance

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/README.md

Loss Functions in Deep Learning

https://insideaiml.com/blog/LossFunctions-in-Deep-Learning-1025

Loss Functions in Deep Learning

Sanober Ibrahim
2 years ago

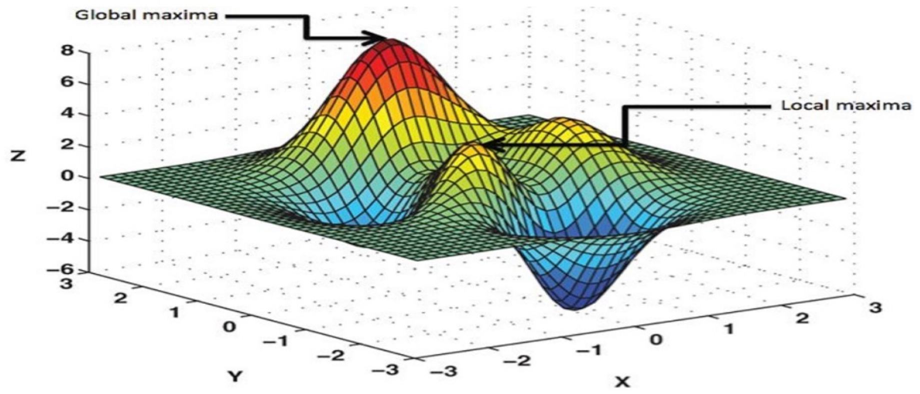


Table of Contents

Winograd Schema (GLUE WNLI)

- The trophy doesn't fit in the brown suitcase
  - because it is too large.
- What is too large?
  - A. The trophy
  - B. The suitcase

Task	Metric	Result	Training time
CoLA	Matthews corr	56.53	3:17
SST-2	Accuracy	92.32	26:06
MRPC	F1/Accuracy	88.85/84.07	2:21
STS-B	Pearson/Spearman corr.	88.64/88.48	2:13
QQP	Accuracy/F1	90.71/87.49	2:22:26
MNLI	Matched acc./Mismatched acc.	83.91/84.10	2:35:23
QNLI	Accuracy	90.66	40:57
RTE	Accuracy	65.70	57
WNLI	Accuracy	56.34	24

Not much better than chance

Table 1. Time line of the Winograd Schema Challenge.

1972:	Winograd's (1972) thesis introduces the original example.
2010:	Levesque [47] proposes the Winograd Schema Challenge.
2010–2011:	The initial corpus of Winograd schemas is created [50].
2014:	Levesque's Research Excellence talk "On our best behavior" [48].
2016:	The Winograd Schema Challenge is run at IJCAI-16. No systems do much better than chance [16].
2018:	WNLI is incorporated in the GLUE set of benchmarks. BERT-based systems do no better than most-frequent-class guessing [91].
2019, May:	Kocijan et al. [43] achieve 72.5% accuracy on WSC273 using pretraining.
2019, June:	Liu et al. [56] achieve 89.0% on WNLI.
2019, November:	Sakaguchi et al. [77] achieve 90.1% on WSC273.

from: https://doi.org/10.1016/j.artint.2023.103971

Winograd Schema (GLUE WNLI)

A Surprisingly Robust Trick for the Winograd Schema Challenge

Vid Kocijan[1], Ana-Maria Creţu[2], Oana-Maria Camburu[1,3], Yordan Yordanov[1], Thomas Lukasiewicz[1,3]
[1]University of Oxford
[2]Imperial College London
[3]Alan Turing Institute, London
firstname.lastname@cs.ox.ac.uk, a.cretu@imperial.ac.uk

Abstract

The Winograd Schema Challenge (WSC) dataset WSC273 and its inference counterpart WNLI are popular benchmarks for natural language understanding and commonsense reasoning. In this paper, we show that the performance of three language models on WSC273 consistently and robustly improves when fine-tuned on a similar pronoun disambiguation problem dataset (denoted WSCR). We additionally generate a large unsupervised WSC-like dataset. By fine-tuning the BERT language model both on the introduced and on the WSCR dataset, we achieve overall accuracies of 72.5% and 74.7% on WSC273 and WNLI, improving the previous state-of-the-art solutions by 8.8% and 9.6%, respectively. Furthermore, our fine-tuned models are also consistently more accurate on the "complex" subsets of WSC273, introduced by Trichelair et al. (2018).

to the small existing datasets making it difficult to train neural networks directly on the task.

Neural networks have proven highly effective in natural language processing (NLP) tasks, outperforming other machine learning methods and even matching human performance (Hassan et al., 2018; Nangia and Bowman, 2018). However, supervised models require many per-task annotated training examples for a good performance. For tasks with scarce data, transfer learning is often applied (Howard and Ruder, 2018; Johnson and Zhang, 2017), i.e., a model that is already trained on one NLP task is used as a starting point for other NLP tasks.

A common approach to transfer learning in NLP is to train a language model (LM) on large amounts of unsupervised text (Howard and Ruder, 2018) and use it, with or without further fine-tuning, to solve other downstream tasks. Building on top of a LM has proven to be very suc-

View PDF

ELSEVIER

Artificial Intelligence
Available online 11 July 2023, 103971
In Press, Corrected Proof ⓘ What's this?

The defeat of the Winograd Schema Challenge

Vid Kocijan [a 1], Ernest Davis [b], Thomas Lukasiewicz [c d], Gary Marcus [e], Leora Morgenstern [f]

Show more

+ Add to Mendeley   Share   Cite

https://doi.org/10.1016/j.artint.2023.103971   Get rights and content

Abstract

The Winograd Schema Challenge—a set of twin sentences involving pronoun reference disambiguation that seem to require the use of commonsense knowledge—was proposed by Hector Levesque in 2011. By 2019, a number of AI systems, based on large pre-trained transformer-based language models and fine-tuned on these kinds of problems, achieved better than 90% accuracy. In this paper, we review the history of the Winograd Schema Challenge and discuss the lasting contributions of the flurry of research that has taken place on the WSC in the last decade. We discuss the significance of various datasets developed for WSC, and the research community's deeper understanding of the role of surrogate tasks in assessing the intelligence of an AI system.

Keywords

Commonsense reasoning; Winograd Schema Challenge

Steps (Review, again)

- Datasets
  - colab
  - contains splits: train, validation, test
- Tokenizing
  - colab
  - maps input text to a sequence of tokens
    - where token is an offset into vocabulary
- Model: f
  - Includes tokenizer
  - Typically based on BERT
- Trainer
  - colab
  - For each epoch (pass over training split)
    - Train (update f, using stochastic gradient descent)
    - Evaluate (score f on validation split)

```
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

```
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

```
trainer.train()
```

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue.py

- Trainer
  - colab
  - For each epoch
    - Train (update f, using SGD)
    - Evaluate (score f on validation split)

```python
    # Training
    if training_args.do_train:
        checkpoint = None
        if training_args.resume_from_checkpoint is not None:
            checkpoint = training_args.resume_from_checkpoint
        elif last_checkpoint is not None:
            checkpoint = last_checkpoint
        train_result = trainer.train(resume_from_checkpoint=checkpoint)
        metrics = train_result.metrics
        max_train_samples = (
            data_args.max_train_samples if data_args.max_train_samples is not None else len(train_dataset)
        )
        metrics["train_samples"] = min(max_train_samples, len(train_dataset))

        trainer.save_model()  # Saves the tokenizer too for easy upload

        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()
```

```python
    # Evaluation
    if training_args.do_eval:
        logger.info("*** Evaluate ***")

        # Loop to handle MNLI double evaluation (matched, mis-matched)
        tasks = [data_args.task_name]
        eval_datasets = [eval_dataset]
        if data_args.task_name == "mnli":
            tasks.append("mnli-mm")
            valid_mm_dataset = raw_datasets["validation_mismatched"]
            if data_args.max_eval_samples is not None:
                max_eval_samples = min(len(valid_mm_dataset), data_args.max_eval_samples)
                valid_mm_dataset = valid_mm_dataset.select(range(max_eval_samples))
            eval_datasets.append(valid_mm_dataset)
            combined = {}

        for eval_dataset, task in zip(eval_datasets, tasks):
            metrics = trainer.evaluate(eval_dataset=eval_dataset)

            max_eval_samples = (
                data_args.max_eval_samples if data_args.max_eval_samples is not None else len(eval_dataset)
            )
            metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))

            if task == "mnli-mm":
                metrics = {k + "_mm": v for k, v in metrics.items()}
            if task is not None and "mnli" in task:
                combined.update(metrics)

            trainer.log_metrics("eval", metrics)
            trainer.save_metrics("eval", combined if task is not None and "mnli" in task else metrics)
```

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue_no_trainer.py

- with trainer
  - 649 lines of code
- without trainer
  - 665 lines of code
  - SGD is more obvious

- in both cases,
  - the example is hardwired for GLUE
- no clear distinction between
  - foreground:
    - GLUE-specific functionality
      - Names of tasks, loss, metric
  - background:
    - General purposefunctionality that applies to many/most classification problems

```python
for epoch in range(starting_epoch, args.num_train_epochs):
    model.train()
    if args.with_tracking:
        total_loss = 0
    if args.resume_from_checkpoint and epoch == starting_epoch and resume_step is not None:
        # We skip the first `n` batches in the dataloader when resuming from a checkpoint
        active_dataloader = accelerator.skip_first_batches(train_dataloader, resume_step)
    else:
        active_dataloader = train_dataloader
    for step, batch in enumerate(active_dataloader):
        outputs = model(**batch)
        loss = outputs.loss
        # We keep track of the loss at each epoch
        if args.with_tracking:
            total_loss += loss.detach().float()
        loss = loss / args.gradient_accumulation_steps
        accelerator.backward(loss)
        if step % args.gradient_accumulation_steps == 0 or step == len(train_dataloader) - 1:
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
            progress_bar.update(1)
            completed_steps += 1

        if isinstance(checkpointing_steps, int):
            if completed_steps % checkpointing_steps == 0:
                output_dir = f"step_{completed_steps}"
                if args.output_dir is not None:
                    output_dir = os.path.join(args.output_dir, output_dir)
                accelerator.save_state(output_dir)

        if completed_steps >= args.max_train_steps:
            break

```

https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue_no_trainer.py

```python
    for epoch in range(starting_epoch, args.num_train_epochs):
        model.train()
        if args.with_tracking:
            total_loss = 0
        if args.resume_from_checkpoint and epoch == starting_epoch and resume_step is not None:
            # We skip the first `n` batches in the dataloader when resuming from a checkpoint
            active_dataloader = accelerator.skip_first_batches(train_dataloader, resume_step)
        else:
            active_dataloader = train_dataloader
        for step, batch in enumerate(active_dataloader):
            outputs = model(**batch)
            loss = outputs.loss
            # We keep track of the loss at each epoch
            if args.with_tracking:
                total_loss += loss.detach().float()
            loss = loss / args.gradient_accumulation_steps
            accelerator.backward(loss)
            if step % args.gradient_accumulation_steps == 0 or step == len(train_dataloader) - 1:
                optimizer.step()
                lr_scheduler.step()
                optimizer.zero_grad()
                progress_bar.update(1)
                completed_steps += 1

            if isinstance(checkpointing_steps, int):
                if completed_steps % checkpointing_steps == 0:
                    output_dir = f"step_{completed_steps}"
                    if args.output_dir is not None:
                        output_dir = os.path.join(args.output_dir, output_dir)
                    accelerator.save_state(output_dir)

            if completed_steps >= args.max_train_steps:
                break

```

```python
        model.eval()
        samples_seen = 0
        for step, batch in enumerate(eval_dataloader):
            with torch.no_grad():
                outputs = model(**batch)
            predictions = outputs.logits.argmax(dim=-1) if not is_regression else outputs.logits.squeeze()
            predictions, references = accelerator.gather((predictions, batch["labels"]))
            # If we are in a multiprocess environment, the last batch has duplicates
            if accelerator.num_processes > 1:
                if step == len(eval_dataloader) - 1:
                    predictions = predictions[: len(eval_dataloader.dataset) - samples_seen]
                    references = references[: len(eval_dataloader.dataset) - samples_seen]
                else:
                    samples_seen += references.shape[0]
            metric.add_batch(
                predictions=predictions,
                references=references,
            )

        eval_metric = metric.compute()
        logger.info(f"epoch {epoch}: {eval_metric}")

        if args.with_tracking:
            accelerator.log(
                {
                    "accuracy" if args.task_name is not None else "glue": eval_metric,
                    "train_loss": total_loss.item() / len(train_dataloader),
                    "epoch": epoch,
                    "step": completed_steps,
                },
                step=completed_steps,
            )

```

Running Cola on Discovery
Virtual Environment

```
module load python/3.8.1
python3 -m pip install --user --upgrade pip
python3 -m pip install --user virtualenv
python3 -m venv $HOME/venv/transformers
source $HOME/venv/transformers/bin/activate
pip install --upgrade pip
cd /work/k.church/githubs/transformers
pip install -e .

cd /work/k.church/githubs/transformers/examples/pytorch/text-classification/
pip install -r requirements.txt
pip install urllib3==1.26.6
```

Running Cola on Discovery

outdir=/**courses/CS6120.202410/**data/GLUE/model_outputs/cola
mkdir -p $outdir
dir=/work/k.church/githubs/transformers/examples/pytorch/text-classification
cd $outdir
sh ./cola.sh

https://github.com/google-research/bert

Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI(v2)
BERT-Tiny	64.2	0.0	83.2	81.1/71.1	74.3/73.6	62.2/83.4	70.2	70.3	81.5
BERT-Mini	65.8	0.0	85.9	81.1/71.8	75.4/73.3	66.4/86.2	74.8	74.3	84.1
BERT-Small	71.2	27.8	89.7	83.4/76.2	78.8/77.0	68.1/87.0	77.6	77.0	86.4
BERT-Medium	73.5	38.0	89.6	86.6/81.6	80.4/78.4	69.6/87.9	80.0	79.1	87.7

```
a_model/special_tokens_map.json
***** train metrics *****
  epoch                    =        1.0
  train_loss               =     0.5148
  train_runtime            = 2:18:37.50
  train_samples            =       8551
  train_samples_per_second =      1.028
  train_steps_per_second   =      0.032
10/02/2023 15:20:02 - INFO - __main__ - *** Evaluate ***
[INFO|trainer.py:761] 2023-10-02 15:20:02,237 >> The following columns in the evaluation set don't ha\
ve a corresponding argument in `BertForSequenceClassification.forward` and have been ignored: sentenc\
e, idx. If sentence, idx are not expected by `BertForSequenceClassification.forward`,  you can safely\
 ignore this message.
[INFO|trainer.py:3213] 2023-10-02 15:20:02,240 >> ***** Running Evaluation *****
[INFO|trainer.py:3215] 2023-10-02 15:20:02,240 >>   Num examples = 1043
[INFO|trainer.py:3218] 2023-10-02 15:20:02,241 >>   Batch size = 8
100% 131/131 [05:03<00:00,  2.32s/it]
***** eval metrics *****
  epoch                     =        1.0
  eval_loss                 =     0.4678
  eval_matthews_correlation =     0.4939
  eval_runtime              = 0:05:05.78
  eval_samples              =       1043
  eval_samples_per_second   =      3.411
  eval_steps_per_second     =      0.428
(transformers) [k.church@c0170 text-classification]$ ls -lt | head
total 120
drwxrwx---+ 2 k.church users  4096 Oct  2 15:25 my_cola_model
-rwxrwx---+ 1 k.church users 18621 Oct  2 12:01 run_xnli.py
-rw-rw----+ 1 k.church users 29229 Oct  2 12:01 run_glue_no_trainer.py
-rwxrwx---+ 1 k.church users 28270 Oct  2 12:01 run_glue.py
-rwxrwx---+ 1 k.church users 33590 Oct  2 12:01 run_classification.py
-rw-rw----+ 1 k.church users   112 Oct  2 12:01 requirements.txt
-rw-rw----+ 1 k.church users 10725 Oct  2 12:01 README.md
(transformers) [k.church@c0170 text-classification]$ find my_cola_model
my_cola_model
my_cola_model/README.md
my_cola_model/all_results.json
my_cola_model/config.json
my_cola_model/eval_results.json
my_cola_model/pytorch_model.bin
```

Agenda

- Homework
  - Assignment 2: HuggingFace Pipelines
- Background Material
- Old Business
  - Colab
  - Deep Nets: Inference
    - Classification & Regression
    - Anything → Vector
    - Machine Translation
    - Fill Mask

- New Business
  - bertviz
  - Deep Nets: Fine-Tuning
    - Easy: inference
    - Hard: pre-training
    - Not too hard: fine-tuning
  - Code: colab
    - HuggingFace Tutorial
    - HuggingFace Colab
    - run glue example