Intro to deep learning

Dr. Janoś Gabler, University of Bonn

Lecture 7: Classification via fine-tuning

Motivation

- After this lecture you know everything you need to start your final projects
- You will use a GPU to speed up computations
- You will fine-tune your first transformers model

Topics

- Calculating sklearn scores on huggingface models
- CPUs vs GPUs
- Fine-tuning vs. feature extraction
- Primer on (stochastic) gradient descent
- How to do fine-tuning with huggingface
- Topics for final projects

Sklearn scores + huggingface models

Refresher 1: Pipeline

```
>>> from transformers import pipeline
>>> classifier = pipeline(task="text-classification")
>>> sentiments = classifier(text)
>>> sentiments
[{'label': 'NEGATIVE', 'score': 0.9015460014343262}]
```

- Create a `pipeline` with the task `"text-classification"`
- Give it a text or a list of texts
- Results are lists of dicts that can be converted to DataFrames

Task 1

8 min

Refresher 2: Sklearn scores

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_test, y_pred, average=None)
```

```
array([1., 0.97142857, 0.95652174])
```

- Sklearn offers many scores
  - F1, accuracy, precision, recall, ...
  - Classification report
  - Confusion matrix
- Only take `y_test` and `y_pred` as arguments
- Can calculate sklearn scores on results calculated with other libraries

Task 2

10 min

CPUs vs GPUs

What is a CPU

- CPU = Central Processing Unit
- The thing that does computation and logic in your laptop
- There are different architectures
  - x86 (intel and AMD)
  - ARM (recent macs)
- Can do everything a modern computer needs
- Specific tasks can be done faster by other types of processors

CPU illustration

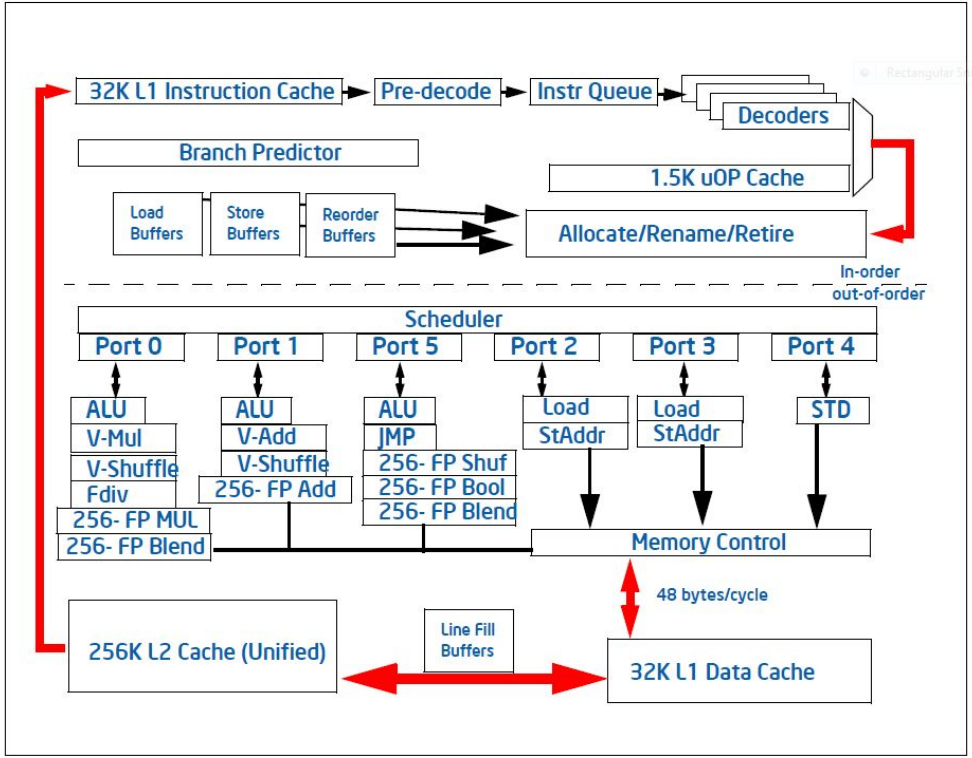
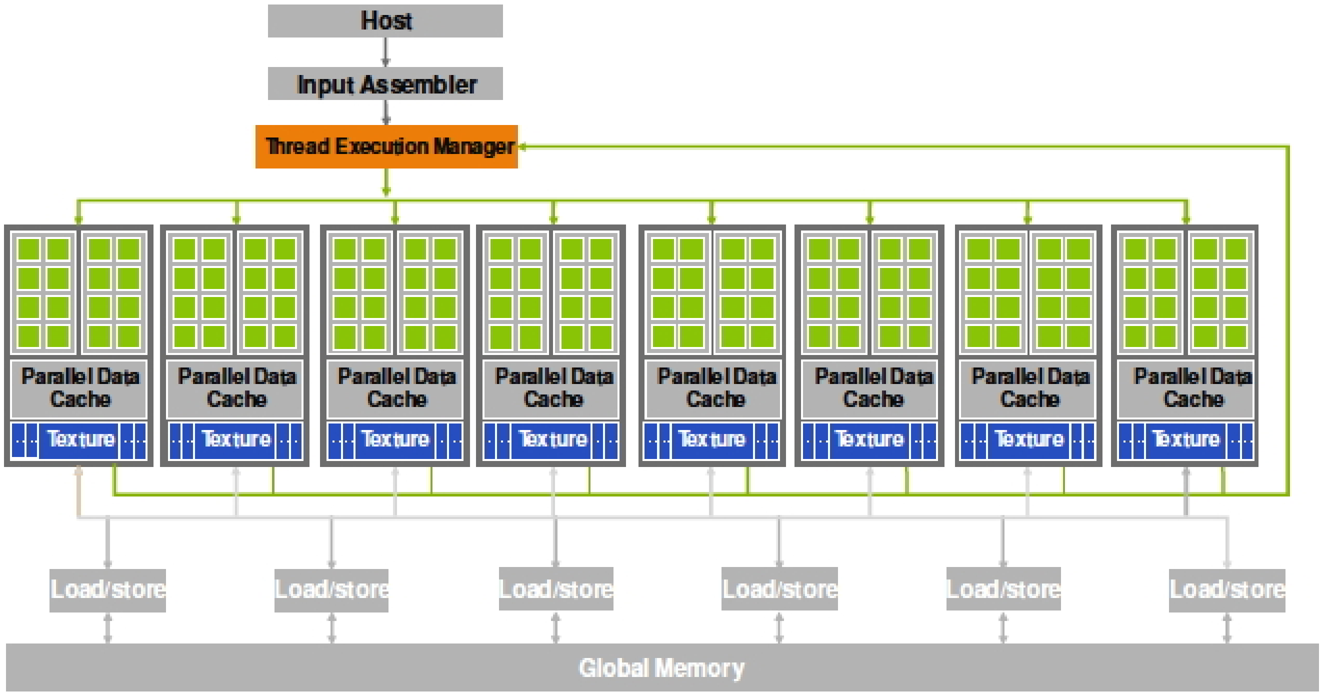


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

What is a GPU

- GPU = Graphics Processing Unit
- Specialized processor for floating point math
- You have some kind of GPU in your laptop
- Some might have a dedicated GPU (laptop or desctop)
- Originated to accelerate computer graphics
- Used for deep learning since ~2008

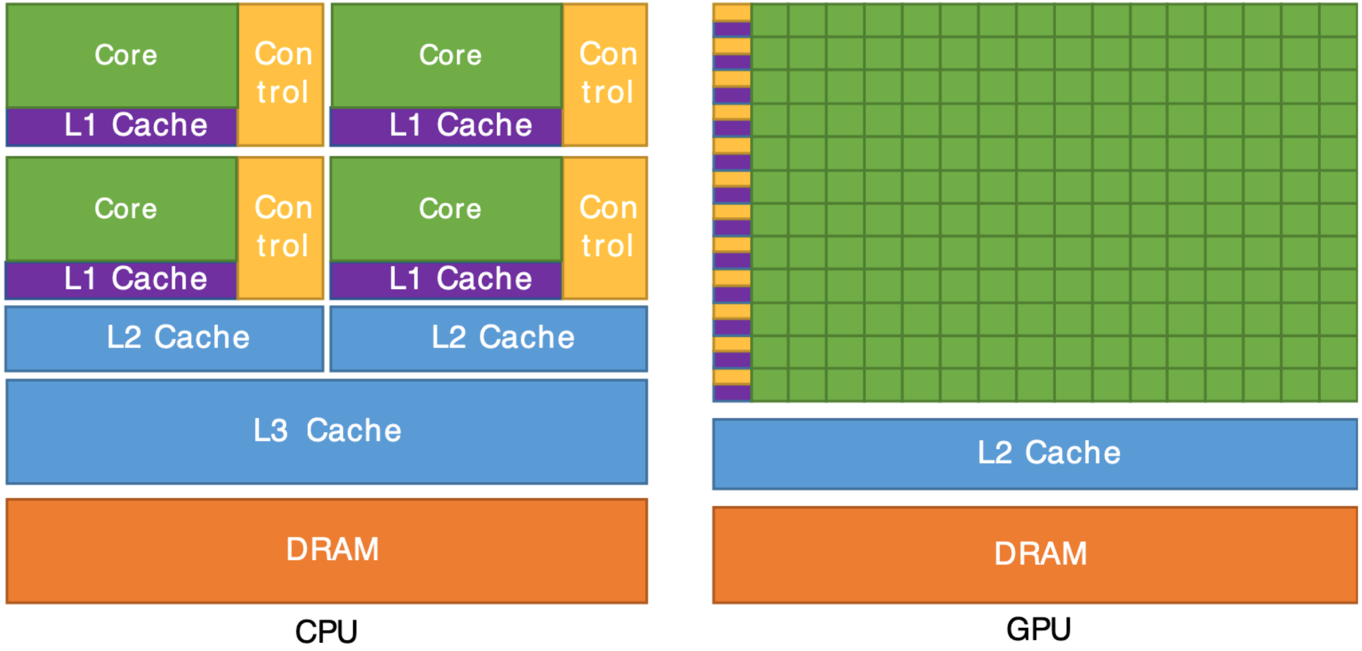
GPU illustration



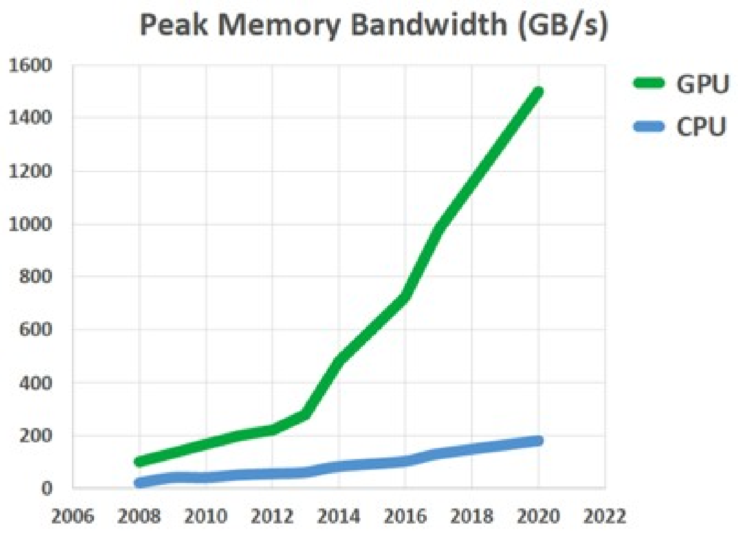
Why are GPUs so fast

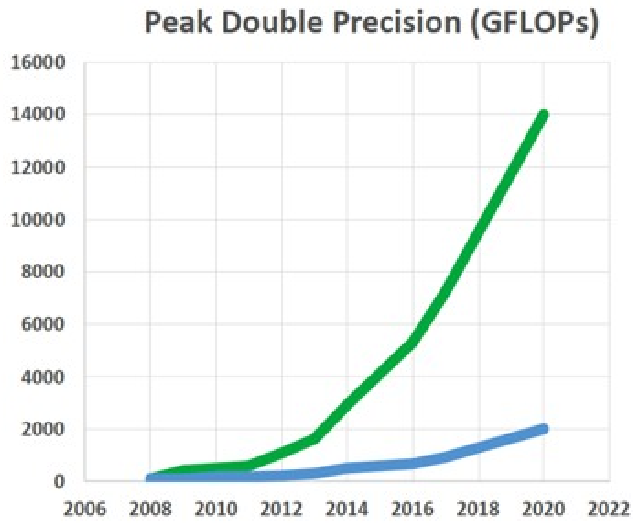
- Load data (e.g. elements of matrix in parallel)
- Do calculations in parallel
- Many more floating point units

Silicon allocation comparison



How much faster?





Power efficiency

- Large GPUs are power hungry (~200 to 500 Watts)
- CPUs typically use less (15 ~ 100 Watts)
- Power per flop is better on GPUs

Drawbacks of GPUs

- Do not work for all workloads
- High latency, i.e. slow for few calculations
  - Don't use the highway to go to the bakery
- Expensive to buy and rent
- Harder to program (but got much better!)

Can I use my laptop GPU?

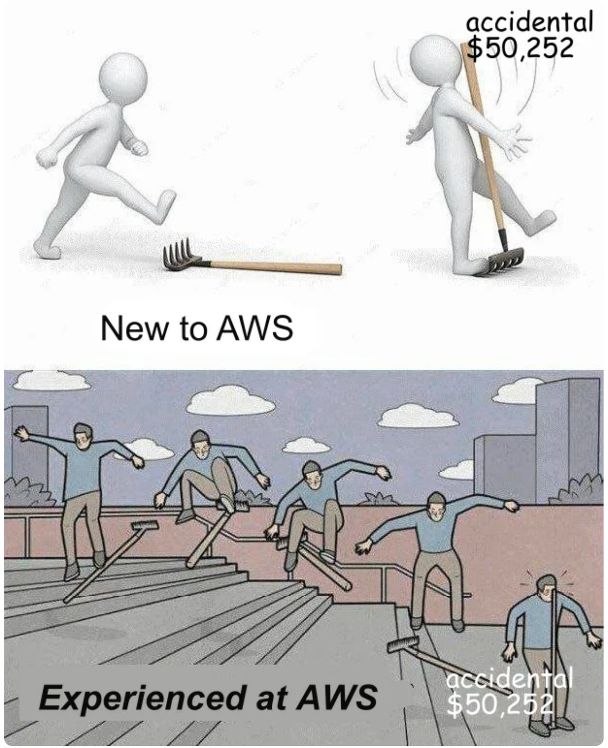
- Need a Cuda compatible GPU (from NVIDIA)
- Need to install correct cuda drivers
- Need to install correct version of Pytorch/JAX/...
- It won't be very fast!
  - Check how much power your charger can provide to your laptop
  - Compare that to how much a large GPU needs

Where can I get a GPU

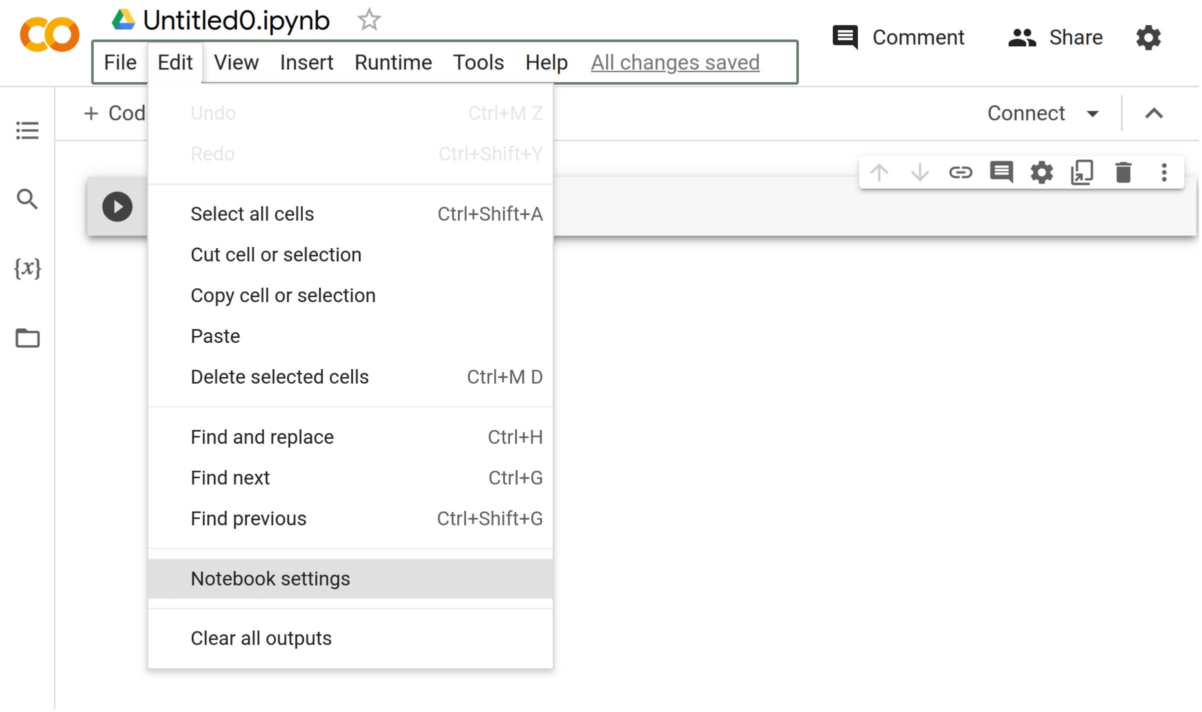
- Google colab: free
- Google Cloud, AWS, etc.: free trial
- Buy an (external) gaming GPU (~500+ Euros)
- Bender Cluster of the University (for PhD students)
- The important thing is that you learn how to use GPUs and colab is enough for that!

Be really careful

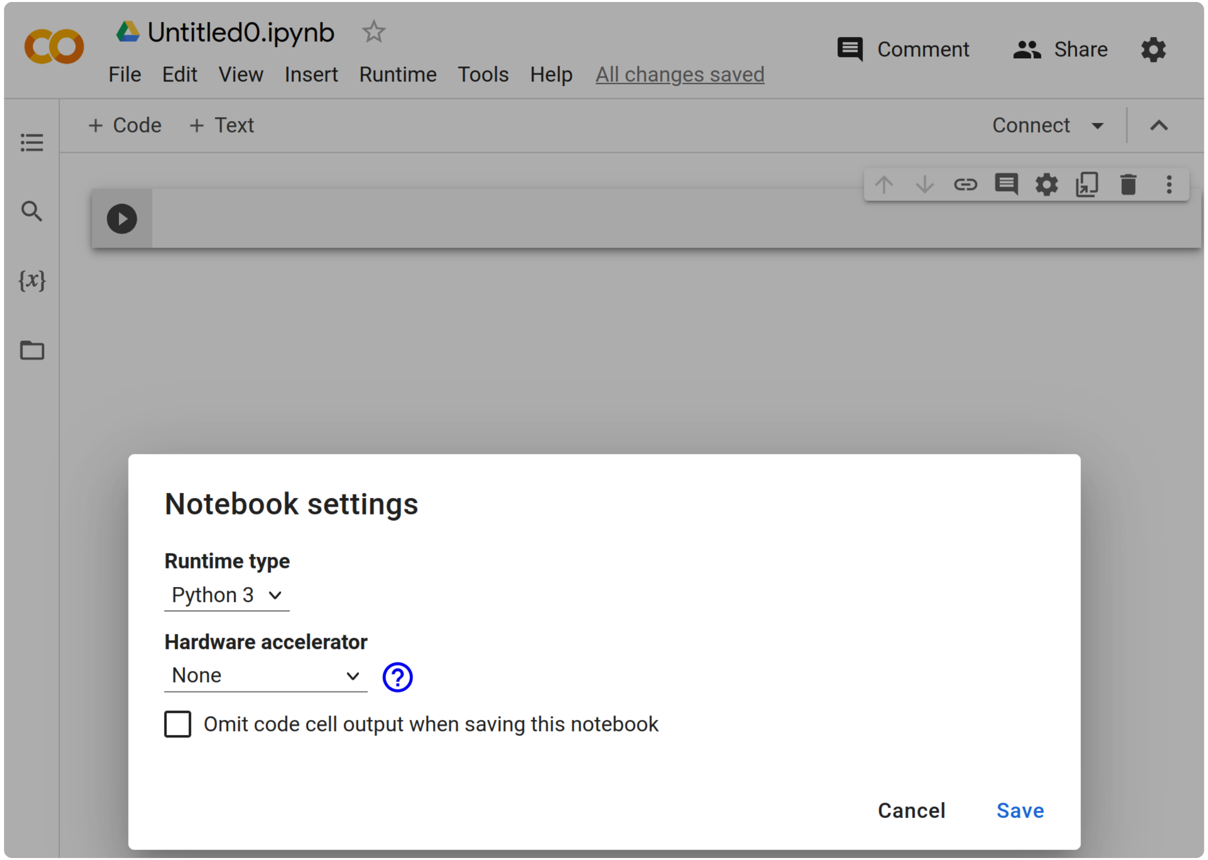
- I know people who created a 20k AWS bill for their company
- Whenever you registered your credit card, make sure you shut down instances when you don't need them
- Colab does not have that danger
- Having said that: You should know how to use AWS, Google Cloud, …



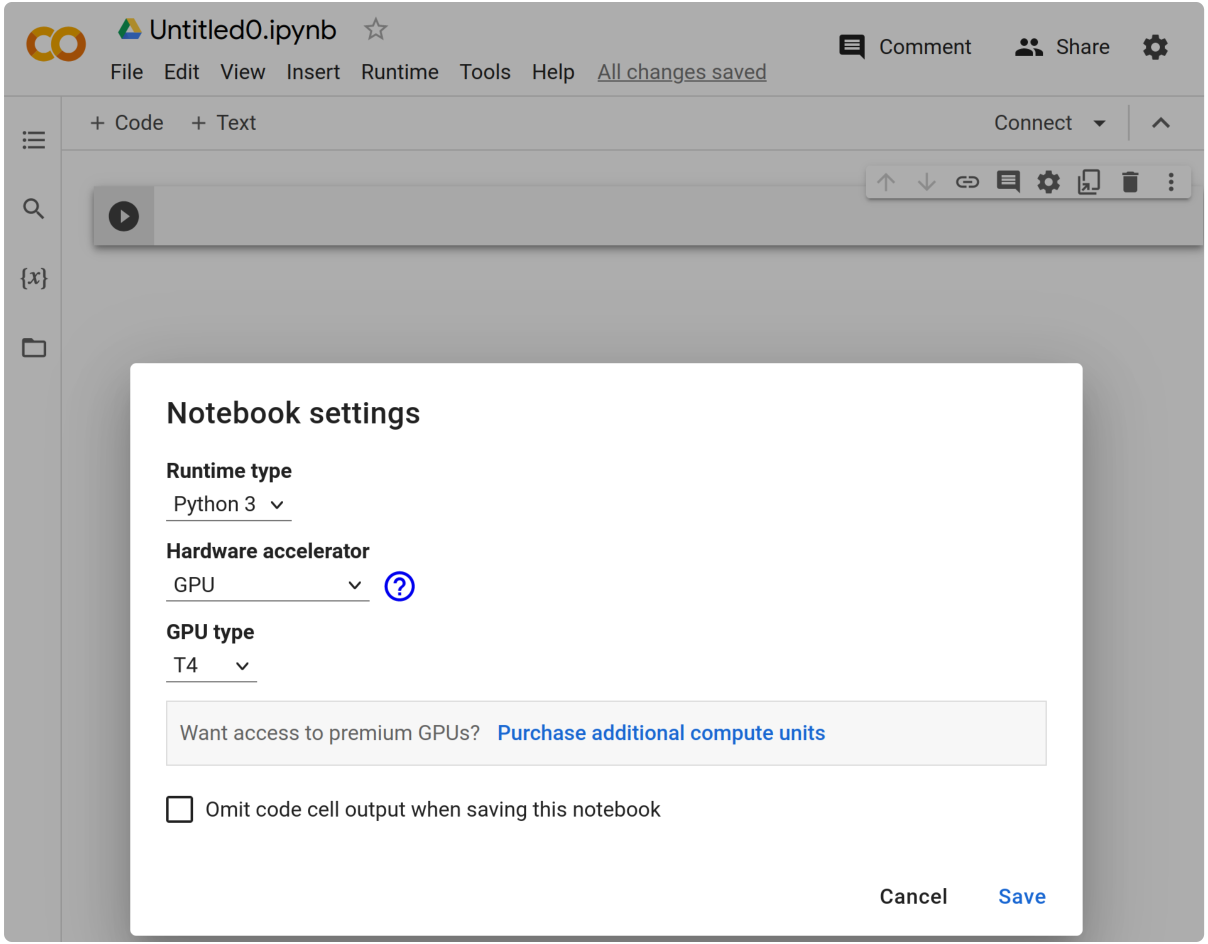
Enabling GPUs on colab



Enabling GPUs on colab



Enabling GPUs on colab



Using GPUs in Pytorch

```
>>> import torch
>>> device = torch.device(
...     "cuda" if torch.cuda.is_available() else "cpu")
>>> device
```

```
device(type='cuda')
```

```
>>> a = torch.ones(200, 200)
>>> a.sum()
```

```
tensor(40000.)
```

```
>>> a_gpu = torch.ones(200, 200).to(device)
>>> a_gpu.sum()
```

```
tensor(40000., device='cuda:0')
```

- To make your code portable, define device with an if condition
- Using GPU = doing calculations with tensors that live on the GPU

Be careful

```
a + a_gpu
```

```
---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call last)

<ipython-input-29-d90d383cf244> in <cell line: 1>()
----> 1 a + a_gpu

RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu!
```

Measuring runtime

```
>>> from time import time
>>> start = time()
>>> (a_gpu ** 2).sum()
>>> gpu_time = time() - start
>>> gpu_time
```

```
0.000629425048828125
```

- Simple approach using `time.time`
- Inaccurate for very fast functions
- Alternative in notebooks: `%timeit` -> runs functions multiple times -> might cause memory problems on GPU if you do not actively delete variables

Task 3

10 minutes

Using GPUs with Huggingface Pipeline

```
>>> from transformers import pipeline
>>> classifier = pipeline(
...     task="text-classification",
...     device="cuda:0" if torch.cuda.is_available() else None,
...)
>>> sentiments = classifier(text)
>>> sentiments
[{'label': 'NEGATIVE', 'score': 0.9015460014343262}]
```

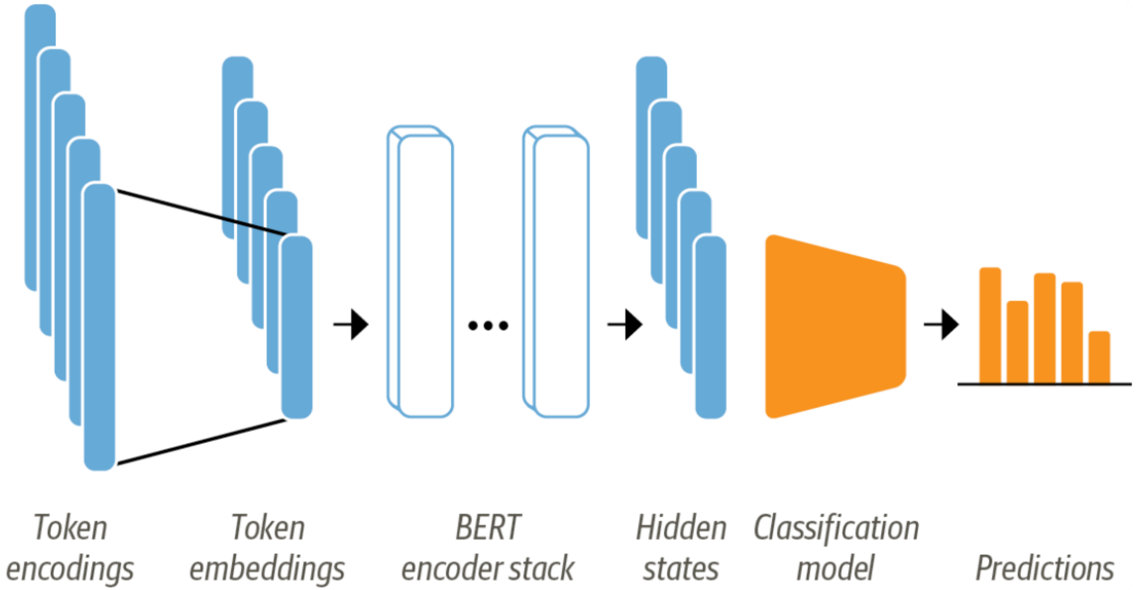
- Pass a device to the pipeline
- For our purpose: Always "cuda:0"
- In the future, pytorch devices will work
- Check the documentation

Task 4

8 min

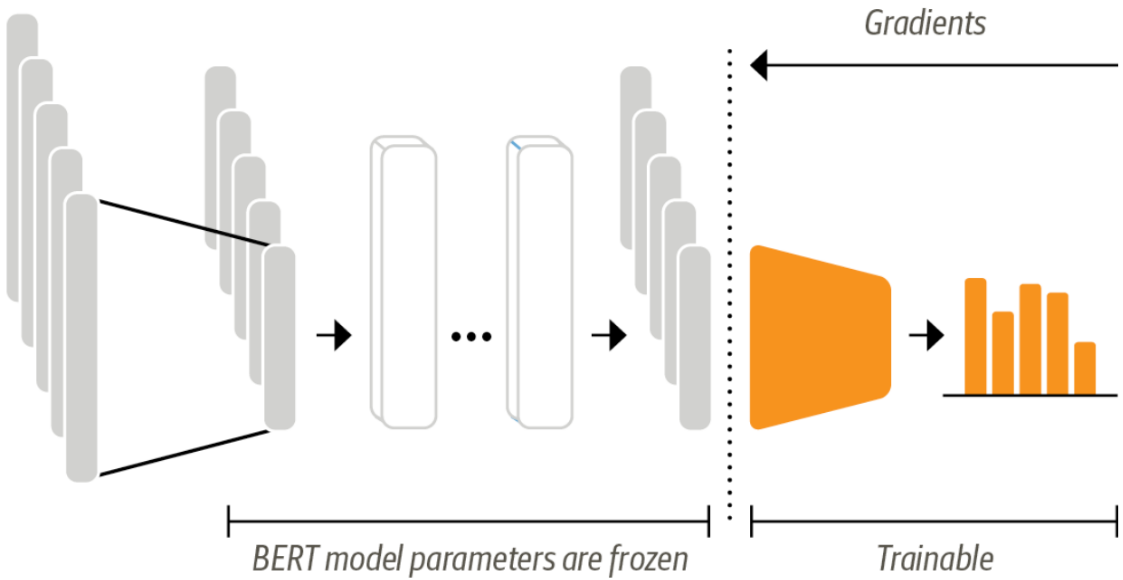
Fine-tuning vs. feature extraction

Illustration of the Bert Model



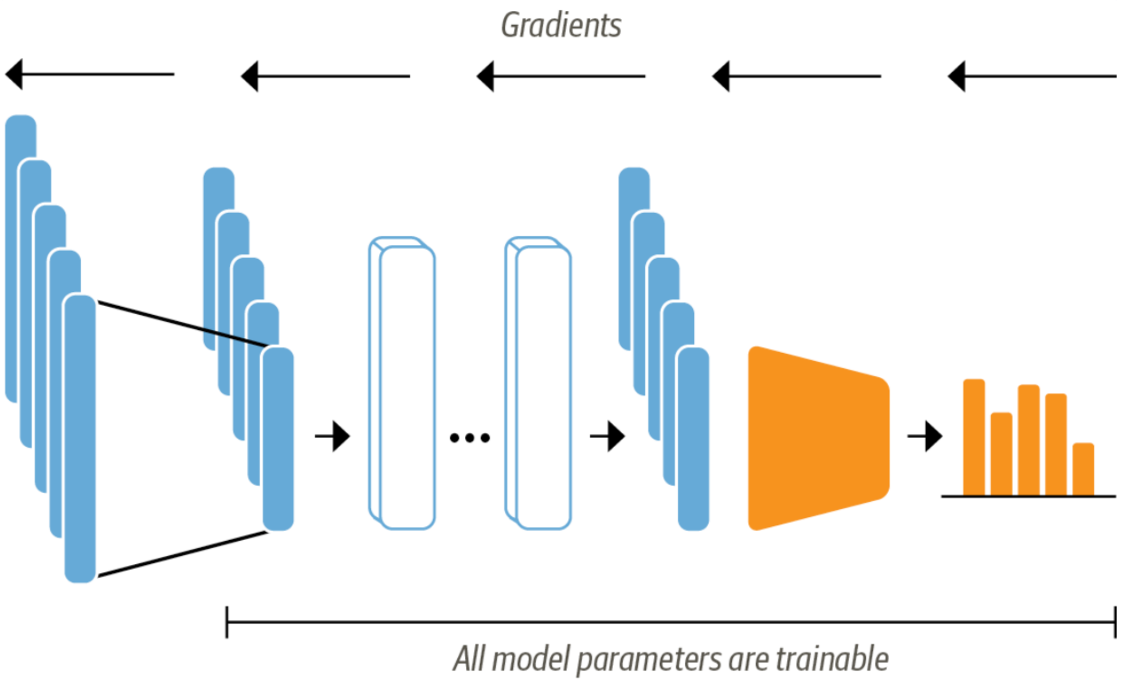
Source: Natural Language Processing with transformers, Fig 2-3

Illustration of Feature extraction



Source: Natural Language Processing with transformers, Fig 2-4

Illustration of Fine tuning



Source: Natural Language Processing with transformers, Fig 2-6

Feature Extraction

- Train parameters of classification model
- Last hidden states contain relevant information "by accident"
- Classifier can be anything (e.g. random forrest)
- CPU is enough

Fine-Tuning

- Train all parameters
- Last hidden states are optimized to contain relevant information
- Classifier is a differentiable neural network
- Very slow without GPU

Steps for fine-tuning in Huggingface

1. Tokenize a dataset (as last time)
2. Instantiate a pre-trained model with classification head
3. (Put the model on the GPU)
4. Write a function to calculate metrics
5. Specify `` `TrainingArguments` ``
6. Instantiate a `` `Trainer` `` an train the model

How will we tackle this

- Steps 1 to 3 are easy
- For steps 4 and 6:
  - I give you some terminology about training neural networks
  - I walk you through in the slides
  - You do it in practice and it is ok if you just copy from the slides
- After lecture 9 you will understand what happened in the background and will be able to set tuning parameters if you have to

Tokenizing the dataset

```
>>> from datasets import load_dataset
>>> from transformers import AutoTokenizer
>>> ds = load_dataset("rotten_tomatoes")
>>> model_name = "distilbert-base-uncased"
>>> tokenizer = AutoTokenizer.from_pretrained(model_name)
>>> def tokenize(batch):
...     return tokenizer(batch["text"], padding=True, truncation=True)

>>> ds_encoded = ds.map(tokenize, batched=True, batch_size=None)
>>> ds_encoded.column_names
```

```
{'train': ['text', 'label', 'input_ids', 'attention_mask'],
 'validation': ['text', 'label', 'input_ids', 'attention_mask'],
 'test': ['text', 'label', 'input_ids', 'attention_mask']}
```

Instatiating a model

```
>>> len(ds["train"].unique("label"))
```

```
2
```

```
>>> from transformers import AutoModelForSequenceClassific

>>> num_labels = 2

>>> model = (AutoModelForSequenceClassification
...     .from_pretrained(model_name, num_labels=num_labels
...     .to(device)
...)
```

- Works similar to `` `AutoModel` ``
- Adds a classification head
- Needs to know how many categories there are

Task 5

5 min

Primer on model training

- Training a neural network means minimizing a loss function
- Minimizing is done with some form of stochastic gradient descent
  - Needs fast computation of gradients
  - Needs data parallel objective functions
- Will have an entire lecture on this
- Today, just some intuition and terminology

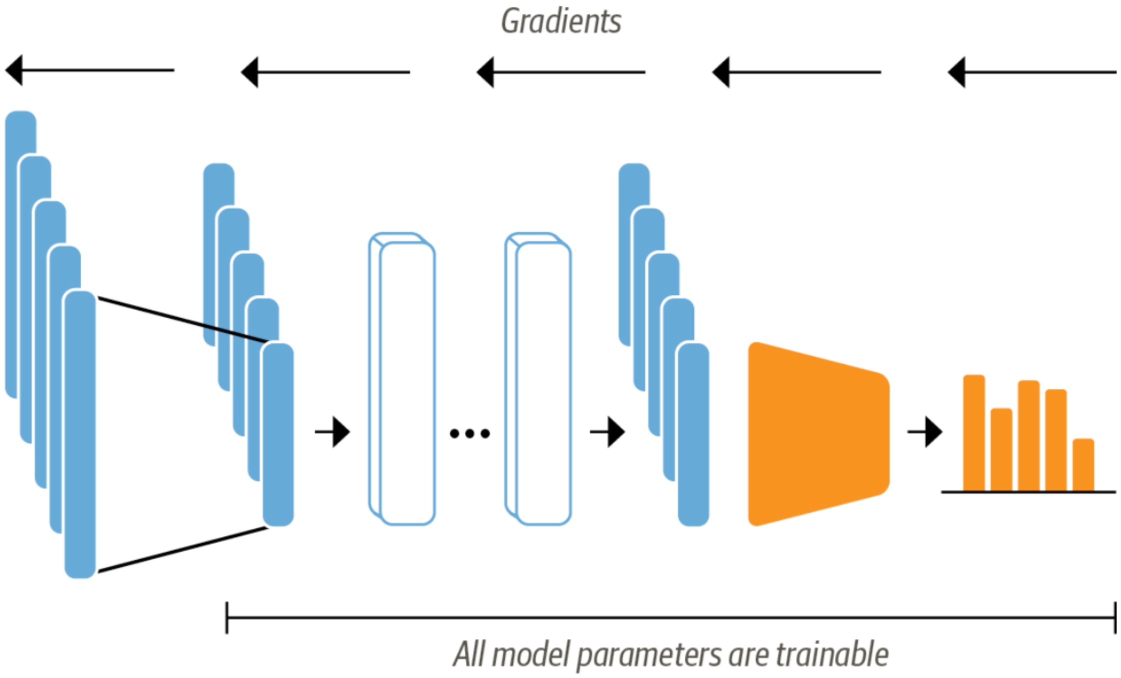
How are gradients calculated

- A few years ago: back propagation
  - User implements derivatives for model components
  - Intermediate values and Jacobians are stored in a forward pass
  - Gradients are accumulated in a backwards pass
- Nowadays: automatic differentiation
  - Same steps but pre-implemented and automated
- Why backwards accumulation?
  - Matrix vector product is cheaper than matrix matrix product

Should you still learn back-propagation

- You should not do manual back-propagation in practice
- Many gain intuition for training problems by thinking about gradients flowing backwards through a neural network
- Still a lot of references in deep learning terminology

Gradients "flowing backwards"



Source: Natural Language Processing with transformers, Fig 2-6

The optimization problem

- $\theta \in \mathcal{R}^d$ is a vector of parameters
- $Z \in \mathcal{R}^{n \times m}$ is a matrix of data (x and y)
- $\ell(\theta, Z)$ is a scalar loss function
- $j(\theta, Z)$ is the gradient of ℓ w.r.t. θ
- Goal: $min_\theta \ell(\theta)$
- In words: find parameters of the neural net that minimize the loss function

Gradient descent

- Given:
  - Parameter guess θ_k
  - Learning rate η
- Goal: Form new parameter guess θ_{k+1}
  - $\theta_{k+1} = \theta_k - \eta \cdot j(\theta_k, Z)$
- Intuition:
  - Gradient gives us a locally valid downhill direction
  - Learning rate limits how far we go in that direction

Pseudo code for GD

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params - learning_rate * params_grad
```

- Repeat the update for a fixed number of epochs
- Chosen according to a computational budget
- More is not always better (overfitting)

A few problems with that

- Calculating a gradient on the entire dataset is slow
  - $\rightarrow$ Stochastic gradient descent (today)
- Can get stuck in local optima / flat spots
  - $\rightarrow$ Momentum (in 2 weeks)
- Learning rate is a very crude way to determine step length
  - Might want to decay the learning rate over time
  - Might want to have different learning rates per parameter
  - $\rightarrow$ Adagrad, Adadelta, RMSprop, Adam (in 2 weeks)

Stochastic gradient descent

- Given:
  - Parameter guess θ_k and learning rate η as before
  - batch_size b
  - batch of the data $z_k \in \mathcal{R}^{b \times m}$
- Parameter update:
  - $\theta_{k+1} = \theta_k - \eta \cdot j(\theta_k, z_k)$
- Intuition:
  - Calculating gradient on a subset of data is faster
  - Less accurate but good enough to point downhill

Pseudo code for SGD

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

- Epoch = number of steps it takes to visit the entire dataset
- Batch size is chosen by the user, often between 8 and 64

Should we use SGD in economics?

- If your function is not differentiable
  - no chance to use gradient descent
- If your function has "few" parameters (< 10 000)
  - storing a hessian or hessian approximation is feasible
  - second order methods are faster
- If your function is not data parallel
  - cannot use stochastic gradient descent
- Check out estimagic for optimizers that are relevant for econ

Fine-tuning with huggingface

Writing a function to compute metrics

- Used to select the best model after fine-tuning
- Used to monitor the optimization
- This function will be passed to the trainer and called internally
  - -> needs to have a specific interface
- You can copy the one we write here whenever you fine-tune a classification model

compute_metrics

```python
from sklearn.metrics import f1_score, accuracy_score

def compute_metrics(pred):
    logits, labels = pred
    preds = logits.argmax(axis=-1)
    f1 = f1_score(labels, preds, average="weighted")
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "f1": f1}
```

- `pred` is a prediction object that contains `logits` and `labels`
- `labels` are y_test, i.e. a vector of correct labels
- `logits` are an array of scores of shape (batch_size, n_choices)
- `argmax` converts `logits` to `y_pred`

Specifying arguments for training

```
from transformers import TrainingArguments

batch_size = 64
logging_steps = len(ds_encoded["train"]) // batch_size

training_args = TrainingArguments(
    # related to optimization
    optim="adamw_torch",
    per_device_train_batch_size=batch_size,
    num_train_epochs=1,
    # related to model selection
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    # related to monitoring
    output_dir="results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    disable_tqdm=False,
    logging_steps=logging_steps,
)
```

- Select the `` `adam` `` implementation from pytorch
- Train for 3 epochs with batch size 64
- Leave learning rates, etc. at default
- Select model with best f1 score at the end
- Copy paste the remaining stuff to get better monitoring than by default

Create a Trainer

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=ds_encoded["train"],
    eval_dataset=ds_encoded["validation"],
)
trainer.train()
```

```
trainer.train()
```

[381/402 26:19 < 01:27, 0.24 it/s, Epoch 2.84/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.434300	0.402095	0.833021	0.830949
2	0.230700	0.380281	0.852720	0.852596

- Collects all of our specifications
- Does the training when we call train
- Starts monitoring

Training result

```
trainer.train()
```

[402/402 27:43, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.434300	0.402095	0.833021	0.830949
2	0.230700	0.380281	0.852720	0.852596
3	0.127600	0.428453	0.848030	0.848025

```
TrainOutput(global_step=402, training_loss=0.26357928363244926, metrics=
{'train_runtime': 1667.6244, 'train_samples_per_second': 15.345, 'train_
steps_per_second': 0.241, 'total_flos': 516421048955760.0, 'train_loss':
0.26357928363244926, 'epoch': 3.0})
```

- Using 3 epochs already led to overfitting

Task 6

12 min

Usisg the fine tuned model

```
>>> custom_text = "Ex machina is a movie about AI taking o
>>> input_tensor = tokenizer.encode(
...     custom_text, return_tensors="pt").to(device)
>>> with torch.no_grad():
...     logits = model(input_tensor).logits.cpu()
>>> logits
```

```
tensor([[ 0.5853, -0.1150]])
```

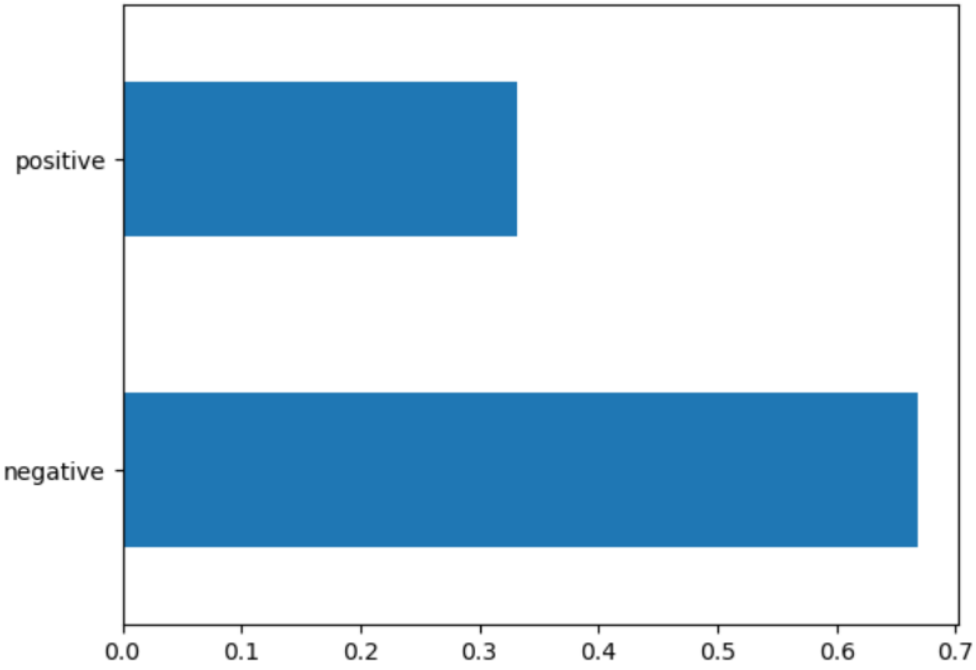
```
>>> import scipy
>>> probs = scipy.special.softmax(logits.flatten())
>>> probs
```

```
array([0.66823876, 0.3317612 ], dtype=float32)
```

- logits are the scores we took an argmax over to get a prediction
- Applying a softmax to them calculates probabilities according to a logit model
- Interpret them as a subjective degree of certainty, not a frequentist probability

Plotting probabilities

```
>>> labels = ["negative", "positive"]
>>> pd.Series(probs, index=labels).plot.barh()
```



Task 7

5 min

Final Projects

Goal

- The project should be helpful for your future career
- You show that you learned something in the class
- You go deeper than what we did in class in at least one area

Example 1: Focus on using NLP

- Use a real-word or example dataset
- Formulate a non-trivial problem you want to solve
- Try out several ways to solve the problem.
- Set up a rigorous benchmarking for the models
- Write two pages about which approach worked best and why you think this is the case

Example 2: Focus on understanding

- Choose a component of neural networks you want to understand better (e.g. different optimizers for training, transformers vs. LSTMs, encoders vs. decoders, ...)
- Do a literature review on that component (1 page)
- Implement (parts) of your component from scratch
- Compare the runtime and performance of your implementation with off the shelf versions
- Write down what you have learned (1 page)

Focus on economic applications

- Get access to an interesting dataset (e.g. via webscraping)
- Clean the data
- Formulate an interesting economic question and a strategy to use NLP to answer that question
- Try out one or two simple models to answer your question
- Write a 2-page research proposal and assess how feasible the project is (given the results of your first models)

More information

Check out the logistics page