CEITEC

Central European Institute of Technology

BRNO | CZECH REPUBLIC

Data Science Practicum

(Lecture 6, 23.10.)

Denisa Šrámková

NLP

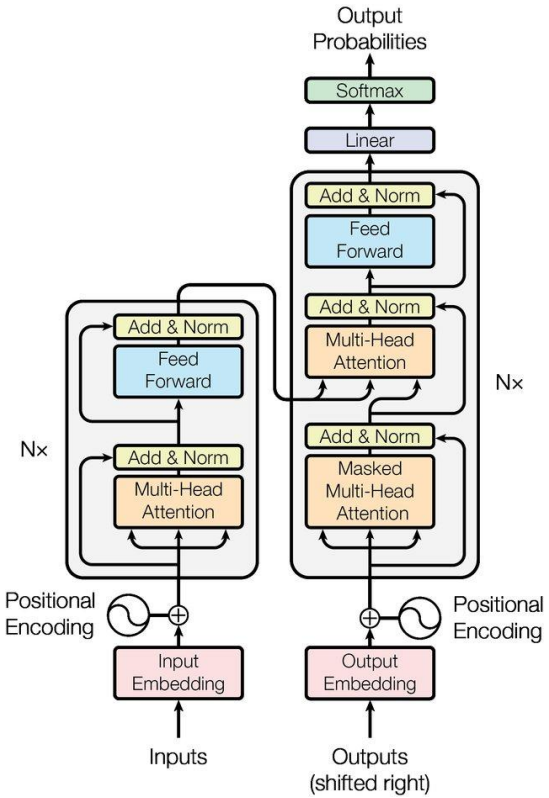
Natural Language Processing

- Transformer architecture - training details
- Text classification - *continuation*
- Hyperparameter optimization
- Text generation

Transformer architecture

Main categories:

1. Encoder only: BERT
2. Decoder-only: GPT-like
3. Encoder-Decoder: BART/T5-like

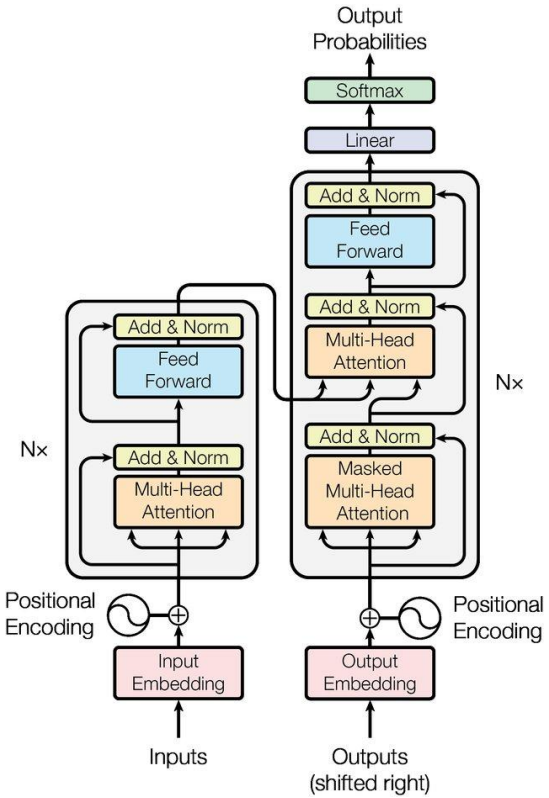


How are these models pre-trained?

1. Encoder only: BERT

2. Decoder-only: GPT-like

3. Encoder-Decoder: BART/T5-like

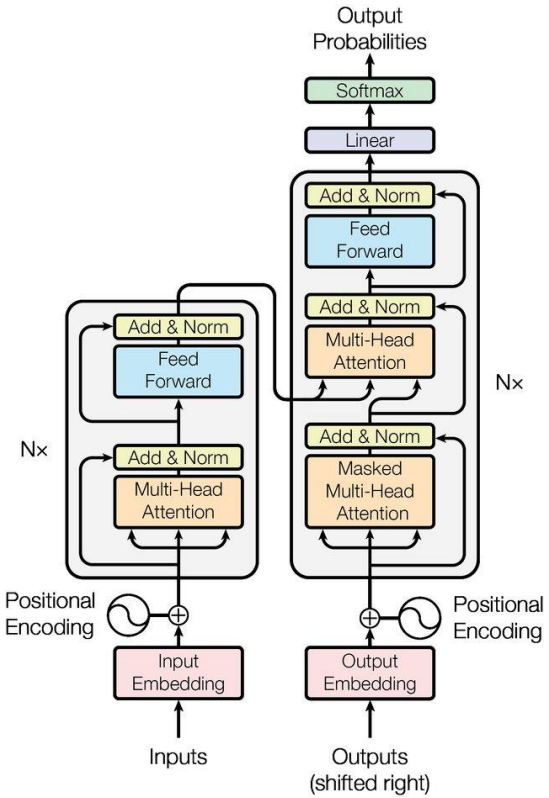


How are these models pre-trained?

1. Encoder only: BERT
   - masked language modelling:

2. Decoder-only: GPT-like

3. Encoder-Decoder: BART/T5-like



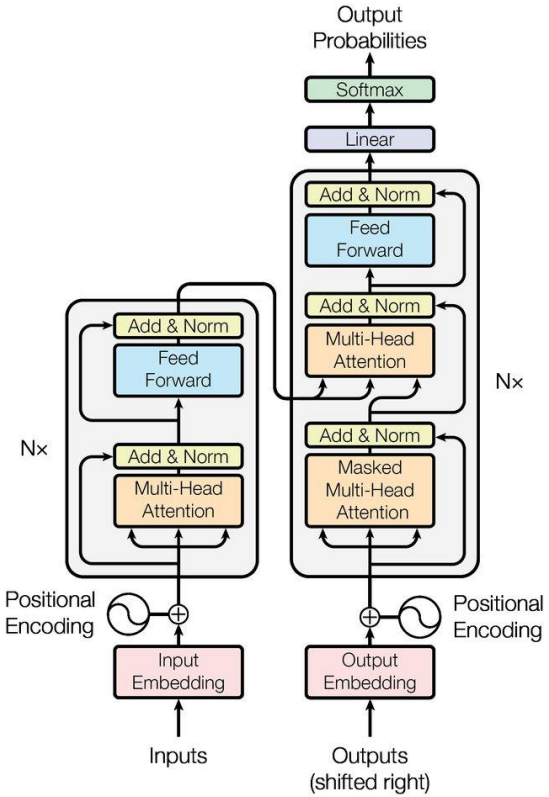
How are these models pre-trained?

1. Encoder only: BERT
   - masked language modelling:

"She [MASK] pizza." → model → "She ate pizza."

2. Decoder-only: GPT-like

3. Encoder-Decoder: BART/T5-like



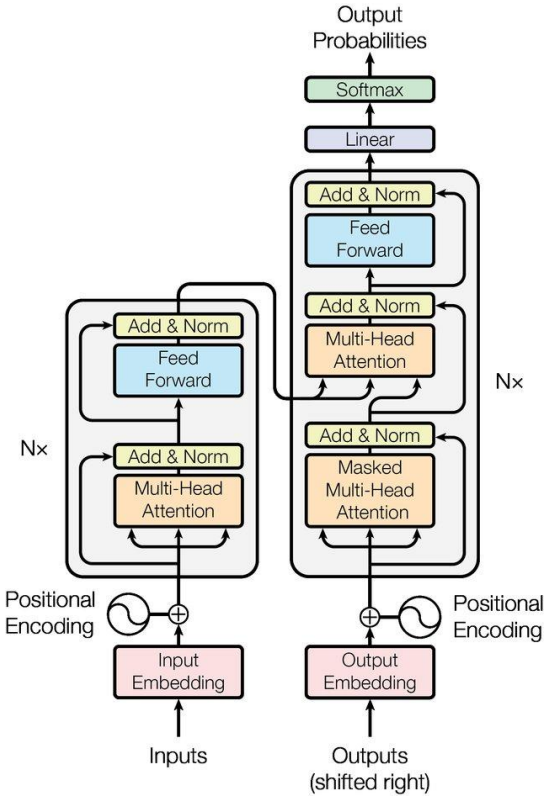
How are these models pre-trained?

1. Encoder only: BERT
   - masked language modelling:

"She [MASK] pizza." → model → "She ate pizza."

2. Decoder-only: GPT-like
   - predicting next word:

3. Encoder-Decoder: BART/T5-like



How are these models pre-trained?

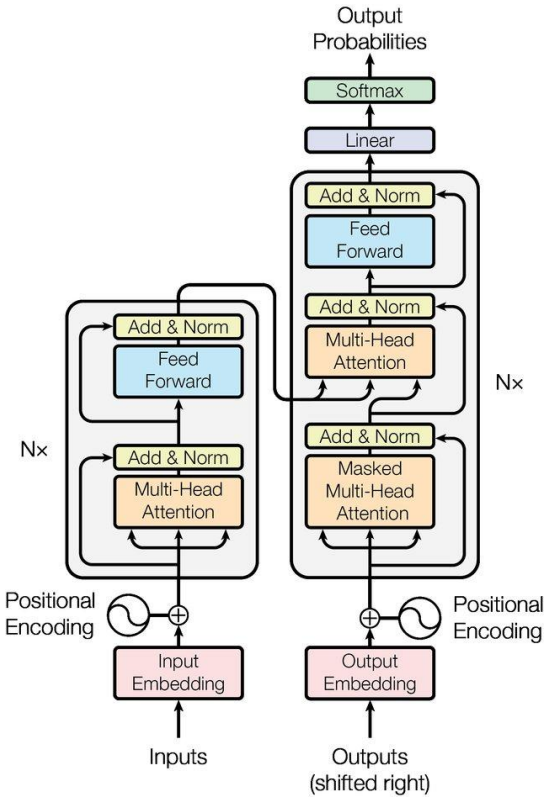
1. Encoder only: BERT
   - masked language modelling:

"She [MASK] pizza." → model → "She ate pizza."

2. Decoder-only: GPT-like
   - predicting next word:

"John didn't" → model → "study."

3. Encoder-Decoder: BART/T5-like



How are these models pre-trained?

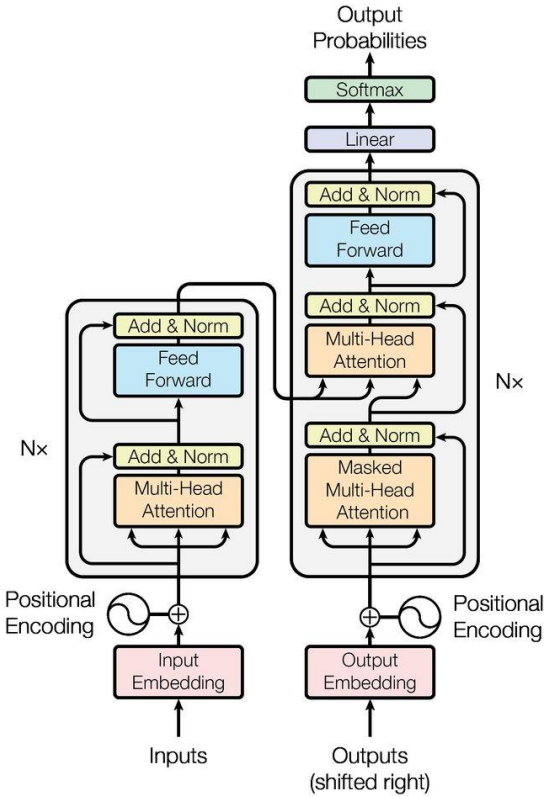
1. Encoder only: BERT
   - masked language modelling:

"She [MASK] pizza." → model → "She ate pizza."

2. Decoder-only: GPT-like
   - predicting next word:

"John didn't" → model → "study."

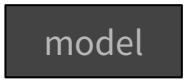
3. Encoder-Decoder: BART/T5-like
   - uses objectives of encoder or decoder models (e.g. T5: replacing random spans of text with single [MASK] token)



Text classification

“New bird species discovered in Philippines Cambridge, England, Aug. 17 (UPI) -- British and Filipino researchers have found a new bird species on a remote island in the northern Philippines, which is a relative to the familiar Moorhen.”

['new',' bird', 'species',' discovered', 'in', 'philippines', 'cambridge', ',' , 'england', ',', 'aug', '.', '17', '(', 'up', '##i', ')', '-', '-', 'british', 'and', 'filipino', 'researchers', 'have', 'found', 'a', 'new', 'bird', 'species', 'on', 'a', 'remote', 'island', 'in', 'the', 'northern', 'philippines', ',', 'which', 'is', 'a', 'relative', 'to', 'the', 'familiar', 'moor', '##hen', '.']



Label: 95.2% Science, 3.2% Business, 1.6% Sport

Text classification

Text classification

Raw input

"New bird species discovered in Philippines Cambridge, England, Aug. 17 (UPI) -- British and Filipino researchers have found a new bird species on a remote island in the northern Philippines, which is a relative to the familiar Moorhen."

Tokenization

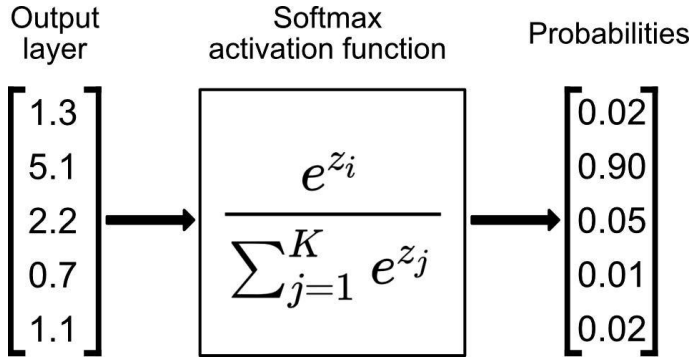
['new', 'bird', 'species', 'discovered', 'in', 'philippines', 'cambridge', ',', 'england', ',', 'aug', '.', '17', '(', 'up', '##i', ')', '-', '-', 'british', 'and', 'filipino', 'researchers', 'have', 'found', 'a', 'new', 'bird', 'species', 'on', 'a', 'remote', 'island', 'in', 'the', 'northern', 'philippines', ',', 'which', 'is', 'a', 'relative', 'to', 'the', 'familiar', 'moor', '##hen', '.']

model

Prediction

Label: 95.2% Science, 3.2% Business, 1.6% Sport

Softmax:



Exercise 1: Text classification

https://github.com/simecek/dspracticum2023/blob/main/lesson06/ds_practicum_ex1_text_classification.ipynb

Hyperparameters

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=16,
    evaluation_strategy='epoch',
    learning_rate=5e-5,
    weight_decay=0.0
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

Hyperparameters

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=16,
    evaluation_strategy='epoch',
    learning_rate=5e-5,
    weight_decay=0.0
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

TrainingArguments

class transformers.TrainingArguments <source>

(output_dir: str, overwrite_output_dir: bool = False, do_train: bool = False, do_eval: bool = False, do_predict: bool = False, evaluation_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'no', prediction_loss_only: bool = False, per_device_train_batch_size: int = 8, per_device_eval_batch_size: int = 8, per_gpu_train_batch_size: typing.Optional[int] = None, per_gpu_eval_batch_size: typing.Optional[int] = None, gradient_accumulation_steps: int = 1, eval_accumulation_steps: typing.Optional[int] = None, eval_delay: typing.Optional[float] = 0, learning_rate: float = 5e-05, weight_decay: float = 0.0, adam_beta1: float = 0.9, adam_beta2: float = 0.999, adam_epsilon: float = 1e-08, max_grad_norm: float = 1.0, num_train_epochs: float = 3.0, max_steps: int = -1, lr_scheduler_type: typing.Union[transformers.trainer_utils.SchedulerType, str] = 'linear', warmup_ratio: float = 0.0, warmup_steps: int = 0, log_level: typing.Optional[str] = 'passive', log_level_replica: typing.Optional[str] = 'warning', log_on_each_node: bool = True, logging_dir: typing.Optional[str] = None, logging_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'steps', logging_first_step: bool = False, logging_steps: float = 500, logging_nan_inf_filter: bool = True, save_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'steps', save_steps: float = 500, save_total_limit: typing.Optional[int] = None, save_safetensors: typing.Optional[bool] = False, save_on_each_node: bool = False, no_cuda: bool = False, use_cpu: bool = False, use_mps_device: bool = False, seed: int = 42, data_seed: typing.Optional[int] = None, jit_mode_eval: bool = False, use_ipex: bool = False, bf16: bool = False, fp16: bool = False, fp16_opt_level: str = 'O1', half_precision_backend: str = 'auto', bf16_full_eval: bool = False, fp16_full_eval: bool = False, tf32: typing.Optional[bool] = None, local_rank: int = -1, ddp_backend: typing.Optional[str] = None, tpu_num_cores: typing.Optional[int] = None, tpu_metrics_debug: bool = False, debug: typing.Union[str, typing.List[transformers.debug_utils.DebugOption]] = '', dataloader_drop_last: bool = False, eval_steps: typing.Optional[float] = None, dataloader_num_workers:

https://huggingface.co/docs/transformers/v4.34.1/en/main_classes/trainer#transformers.TrainingArguments

Hyperparameters

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=16,
    evaluation_strategy='epoch',
    learning_rate=5e-5,
    weight_decay=0.0
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

- learning rate
- dropout
- weight decay
- warmup

TrainingArguments

class transformers.TrainingArguments <source>

(output_dir: str, overwrite_output_dir: bool = False, do_train: bool = False, do_eval: bool = False, do_predict: bool = False, evaluation_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'no', prediction_loss_only: bool = False, per_device_train_batch_size: int = 8, per_device_eval_batch_size: int = 8, per_gpu_train_batch_size: typing.Optional[int] = None, per_gpu_eval_batch_size: typing.Optional[int] = None, gradient_accumulation_steps: int = 1, eval_accumulation_steps: typing.Optional[int] = None, eval_delay: typing.Optional[float] = 0, learning_rate: float = 5e-05, weight_decay: float = 0.0, adam_beta1: float = 0.9, adam_beta2: float = 0.999, adam_epsilon: float = 1e-08, max_grad_norm: float = 1.0, num_train_epochs: float = 3.0, max_steps: int = -1, lr_scheduler_type: typing.Union[transformers.trainer_utils.SchedulerType, str] = 'linear', warmup_ratio: float = 0.0, warmup_steps: int = 0, log_level: typing.Optional[str] = 'passive', log_level_replica: typing.Optional[str] = 'warning', log_on_each_node: bool = True, logging_dir: typing.Optional[str] = None, logging_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'steps', logging_first_step: bool = False, logging_steps: float = 500, logging_nan_inf_filter: bool = True, save_strategy: typing.Union[transformers.trainer_utils.IntervalStrategy, str] = 'steps', save_steps: float = 500, save_total_limit: typing.Optional[int] = None, save_safetensors: typing.Optional[bool] = False, save_on_each_node: bool = False, no_cuda: bool = False, use_cpu: bool = False, use_mps_device: bool = False, seed: int = 42, data_seed: typing.Optional[int] = None, jit_mode_eval: bool = False, use_ipex: bool = False, bf16: bool = False, fp16: bool = False, fp16_opt_level: str = 'O1', half_precision_backend: str = 'auto', bf16_full_eval: bool = False, fp16_full_eval: bool = False, tf32: typing.Optional[bool] = None, local_rank: int = -1, ddp_backend: typing.Optional[str] = None, tpu_num_cores: typing.Optional[int] = None, tpu_metrics_debug: bool = False, debug: typing.Union[str, typing.List[transformers.debug_utils.DebugOption]] = '', dataloader_drop_last: bool = False, eval_steps: typing.Optional[float] = None, dataloader_num_workers:

https://huggingface.co/docs/transformers/v4.34.1/en/main_classes/trainer#transformers.TrainingArguments

Learning rate

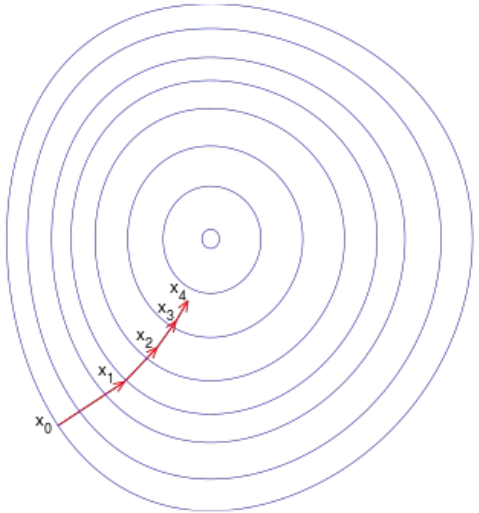
Derivative of Error with respect to weight

Old weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate



Learning rate

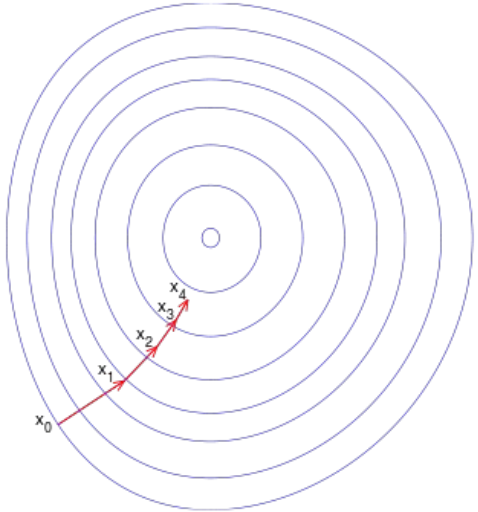
Old weight

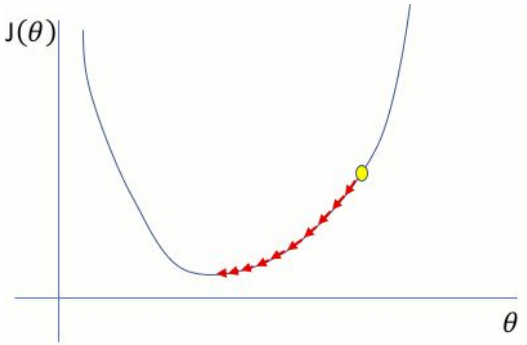
Derivative of Error with respect to weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate





Learning rate

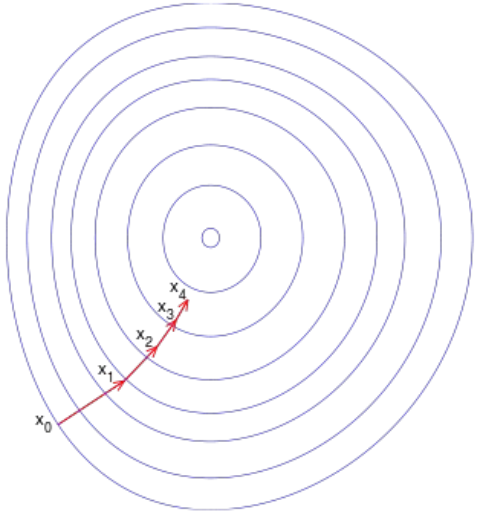
$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

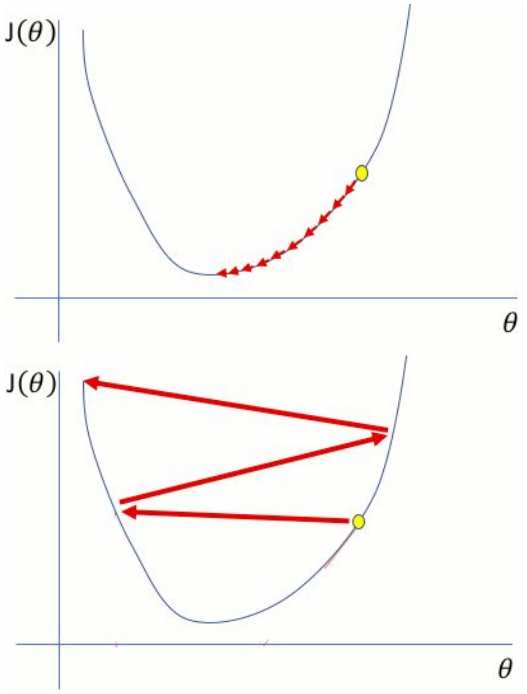
New weight

Old weight

Learning rate

Derivative of Error with respect to weight





Learning rate

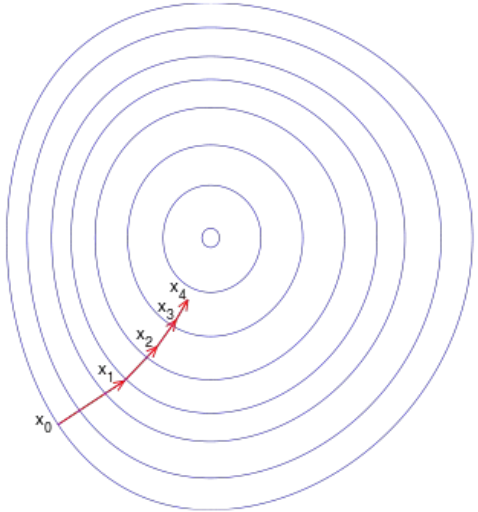
Old weight

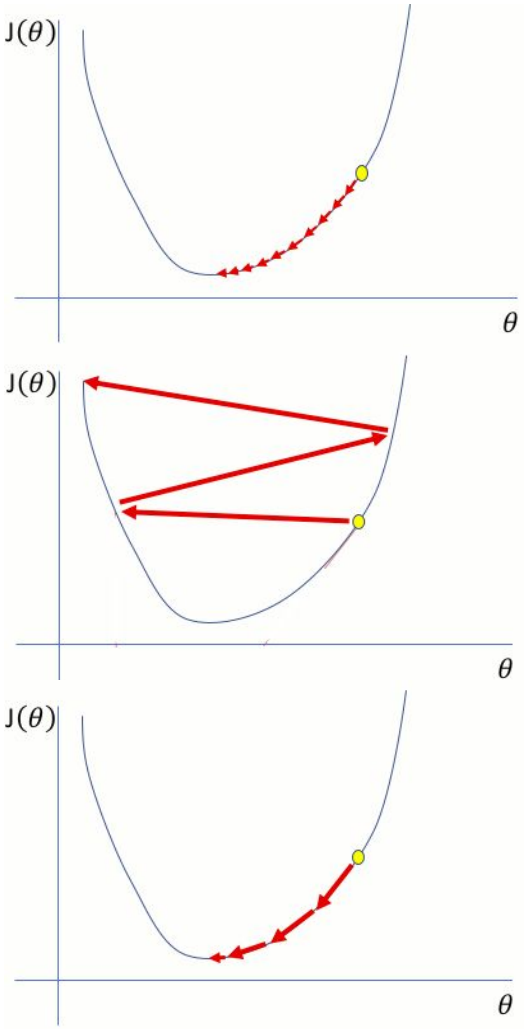
Derivative of Error with respect to weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

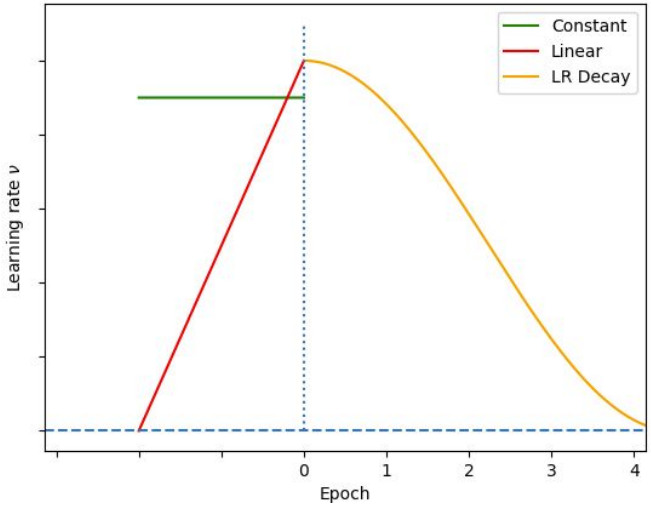
Learning rate





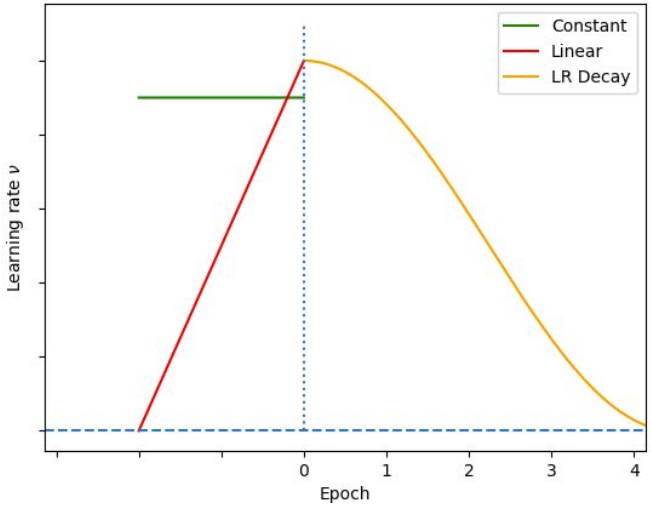
Warmup

So far our learning rate has been constant, but it doesn't have to be.



Warmup

So far our learning rate has been constant, but it doesn't have to be.



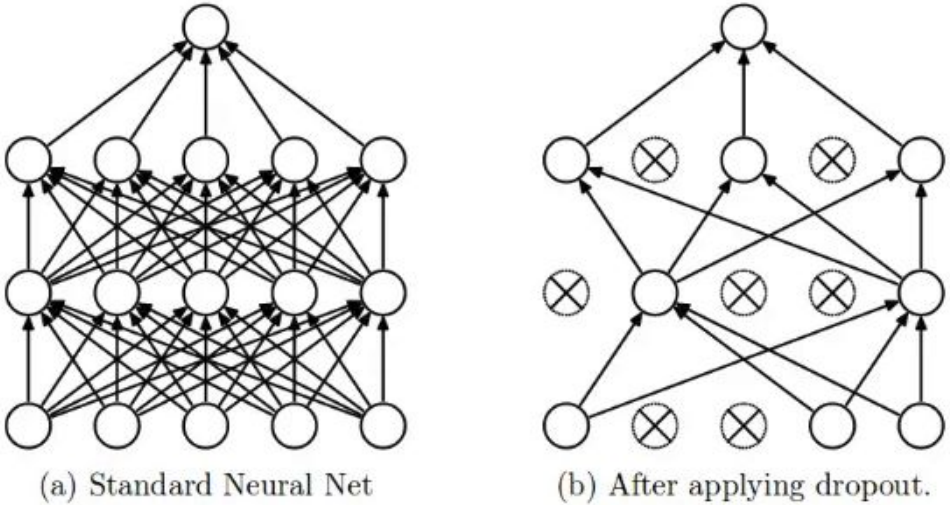
Regularization techniques

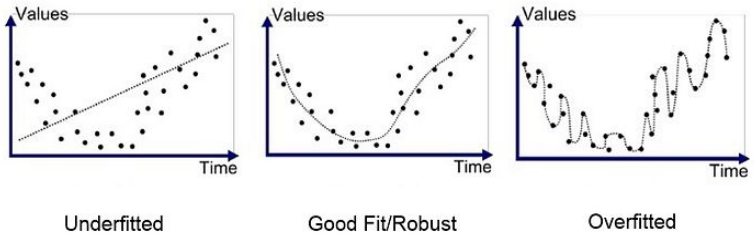
Generalization = ability to cope with new unseen instances

Regularization = methods improving generalization

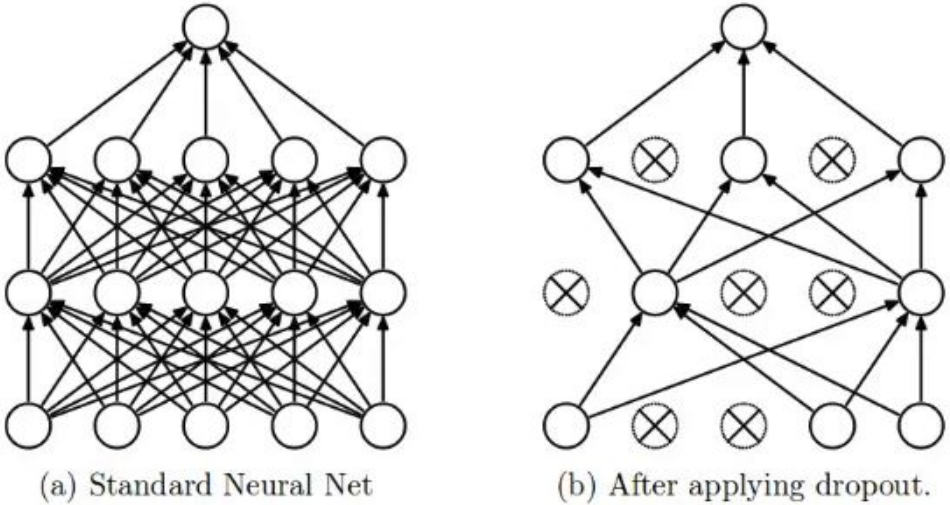
- **Dropout**
- **Weight decay**
- Early stopping
- Ensemble methods
- …

Dropout


(a) Standard Neural Net
(b) After applying dropout.


Values
Time
Values
Time
Values
Time
Underfitted
Good Fit/Robust
Overfitted

Dropout


(a) Standard Neural Net
(b) After applying dropout.
Present with probability p
w
(a) At training time
Always present
pw
(b) At test time

Weight decay

Gradient descent: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

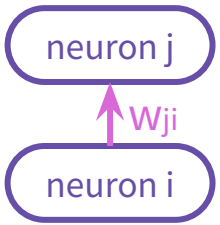
Weight decay: $w_{ji}^{(t+1)} = (1 - \zeta)(w_{ji}^{(t)} + \Delta w_{ji}^{(t)})$

Weight decay

Gradient descent: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

Weight decay: $w_{ji}^{(t+1)} = (1 - \zeta)(w_{ji}^{(t)} + \Delta w_{ji}^{(t)})$



Weight decay

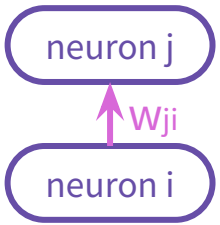
Gradient descent: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$ update of weight w_{ji} at time step t+1

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

learning rate

computed from backpropagation

Weight decay: $w_{ji}^{(t+1)} = (1 - \zeta)(w_{ji}^{(t)} + \Delta w_{ji}^{(t)})$



Weight decay

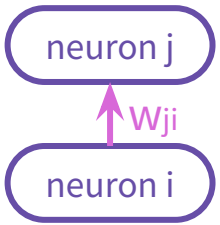
Gradient descent: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$ update of weight w_{ji} at time step t+1

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

learning rate

computed from backpropagation

Weight decay: $w_{ji}^{(t+1)} = (1-\zeta)(w_{ji}^{(t)} + \Delta w_{ji}^{(t)})$

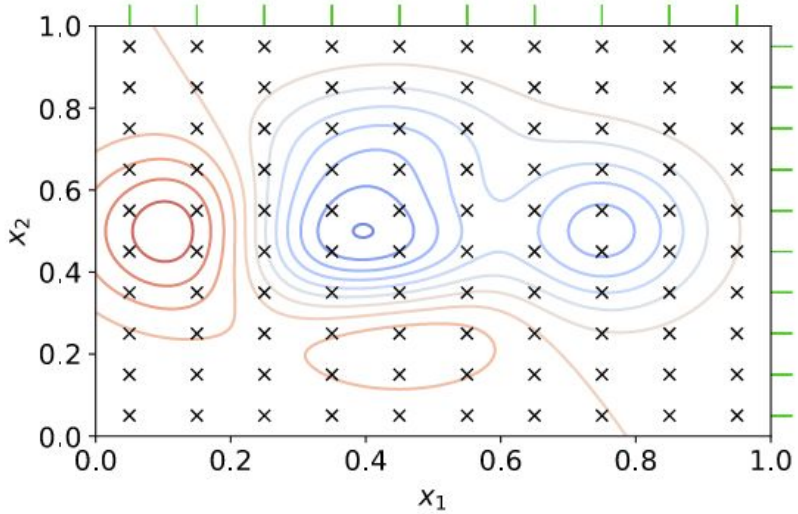


Hyperparameter optimization: Approaches

How can we choose the best values for our hyperparameters?

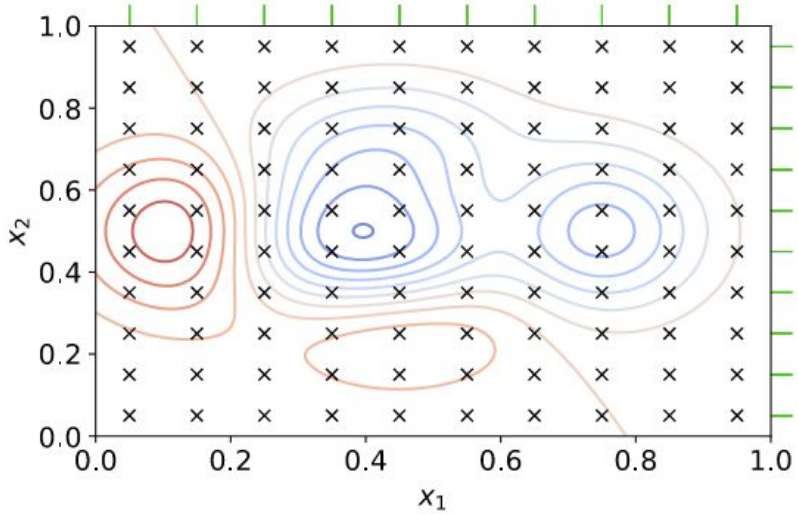
Hyperparameter optimization: Approaches

Grid search/ parameter sweep(/ exhaustive brute force search):



Hyperparameter optimization: Approaches

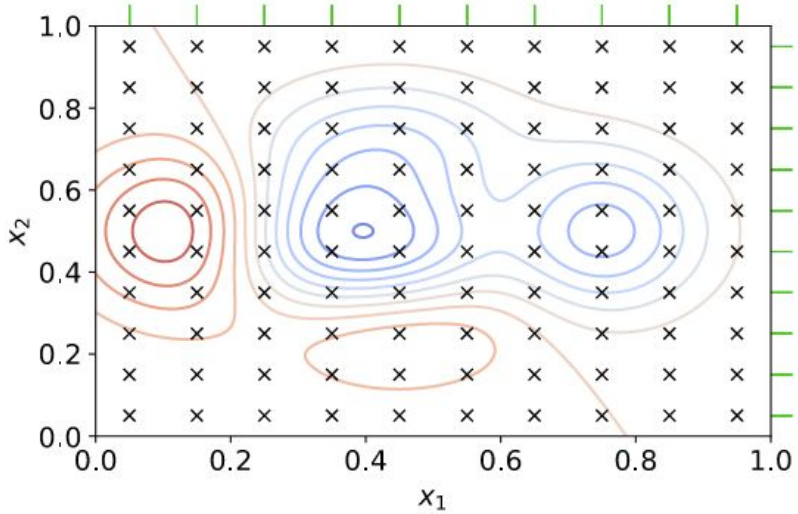
Grid search/ parameter sweep(/ exhaustive brute force search):



Implementation?

Hyperparameter optimization: Approaches

Grid search/ parameter sweep(/ exhaustive brute force search):



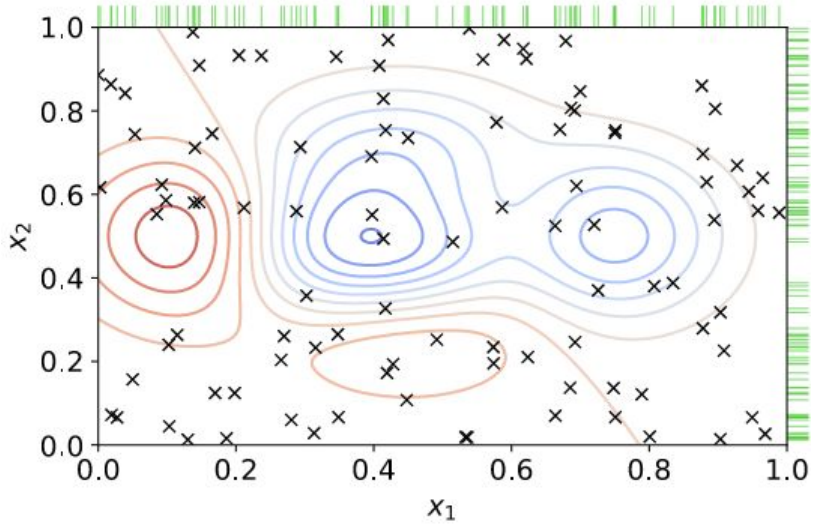
Implementation?

```
lr_values = [0.003, 0.0003, 0.0003]
epoch_values = [2, 4, 8]

for lr in lr_values:
  for epochs in epoch_values:
    train_model(lr_values, epoch_values)
```

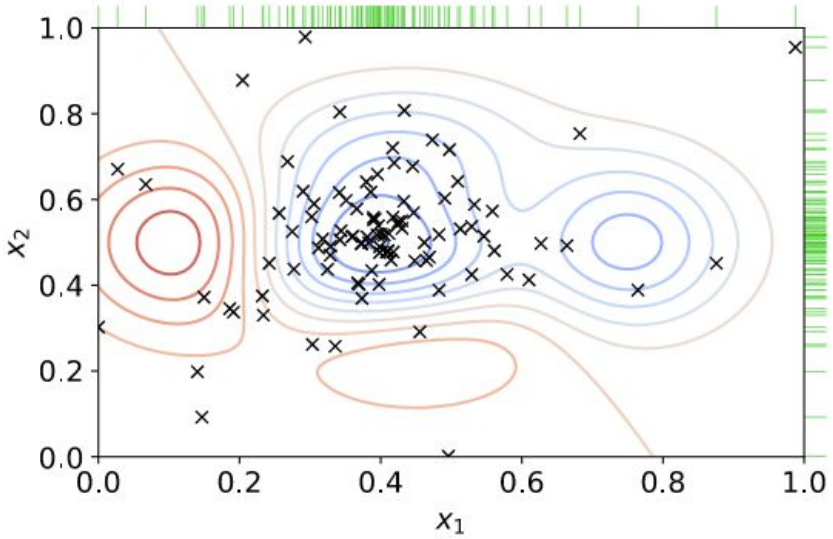
Hyperparameter optimization: Approaches

Random search:



Hyperparameter optimization: Approaches

Bayesian optimization:



https://github.com/bayesian-optimization/BayesianOptimization

Exercise 2: Genomic benchmarks

https://github.com/simecek/dspracticum2023/blob/main/lesson06/ds_practicum_exercise2_genomic_benchmarks.ipynb

Exercise 3: Text generation

https://github.com/simecek/dspracticum2023/blob/main/lesson06/ds_practicum_exercise3_text_generation.ipynb

Homework

1) Try to increase the performance of the model from Exercise 1 and report your best results (e.g. you can apply some hyperparameter optimization)
   - if you are not successful report what approaches you have tried and all of your results