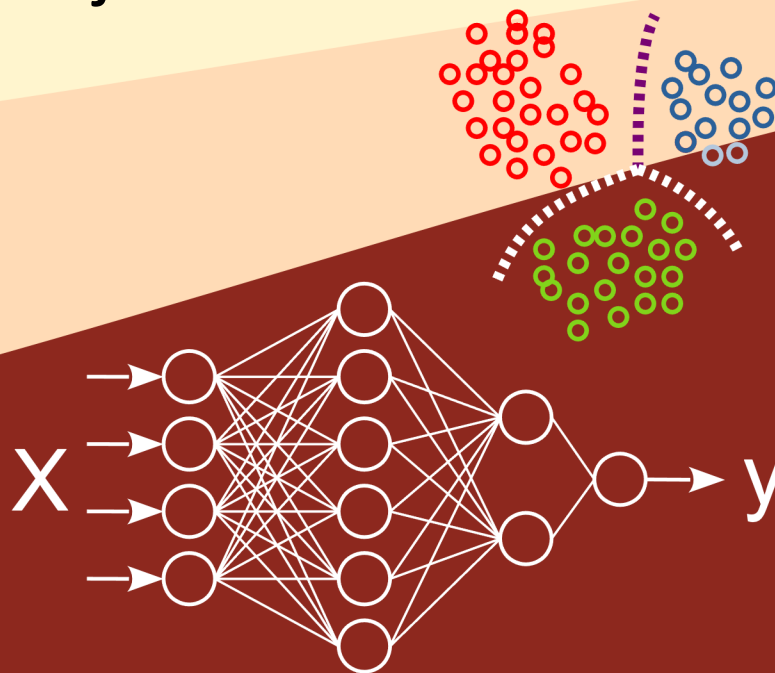
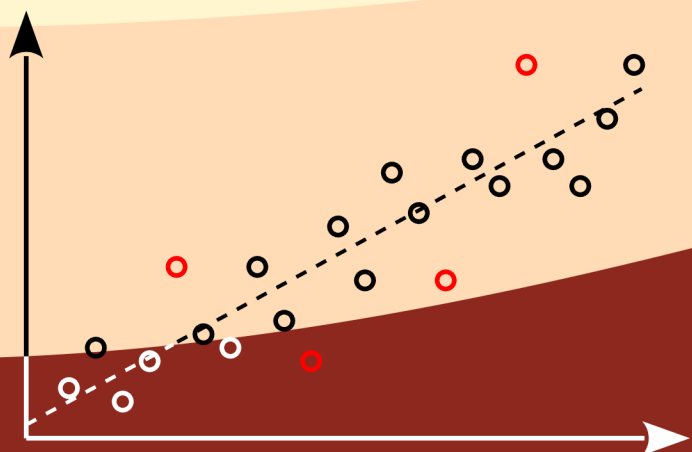


Arhitektura i Razvoj Inteligentnih Sustava

Tjedan 6: Učenje, testiranje, metrika,
repozitoriji modela



Creative Commons



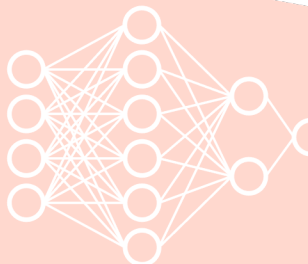
- slobodno smijete:

- dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo
- prerađivati djelo



- pod sljedećim uvjetima:

- imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.
- dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, prerađivanje možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

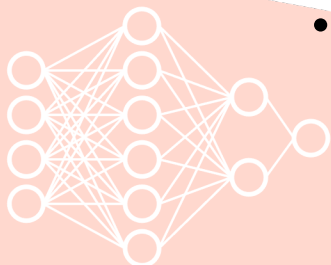
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

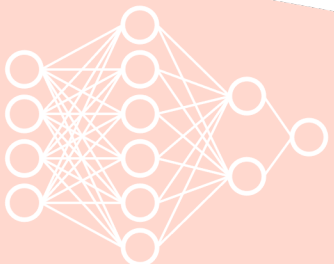
Programski jezici i okviri

- Nekoliko programskih jezika koji se ističu
 - python
 - interpreterski jezik – većina okvira i modula pisana je u drugim programskim jezicima koju su prevedeni na konkretnu arhitekturu procesora i dostupni python-u kao biblioteke (*library*)
 - programiranje na visokom nivou, bez (???) potrebe poznavanja konkretnih implementacijskih detalja
 - R
 - vrlo slično kao i python
 - drukčija namjena – više orijentirana na masovnu obradu okvira podataka
 - Java
 - općeniti programski jezik
 - JIT (*just in-time*) prevođeni jezik
 - i dalje se masovno koriste matematičke biblioteke za linearnu algebru pisane u drugim programskim jezicima



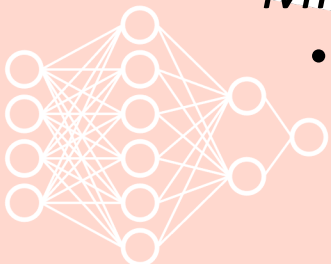
python moduli

- scikit-learn: <https://scikit-learn.org/stable/>
- PyTorch: <https://pytorch.org>
- Keras: <https://keras.io>
- TensorFlow: <https://www.tensorflow.org>



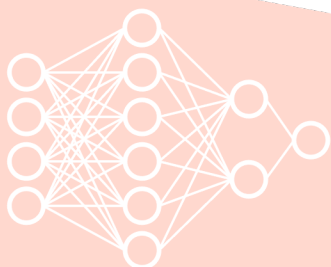
Učenje ANN - PyTorch

- Kriteriji ili funkcije gubitka
 - [PyTorch funkcije gubitka](#)
 - Početni korak u svakom koraku učenja
 - Računamo gradijent i spuštamo se u optimum
- Optimizacija - Više pristupa ([PyTorch optimizatori](#))
 - *Batch Gradient Descend* – Gradijent računamo na cijelom skupu podataka za učenje, ponavljamo toliko epoha dok nam ažuriranje težina bude prihvatljivo malo
 - *Stochastic Gradient Descend* (SGD) – Gradijent se računa za svaki uzorak u skupu podataka za učenje
 - Nezgodan jer se mogu dešavati fluktuacije
 - *Mini-batch* pristup – Kombinacija oba prethodna pristupa
 - *mini-batch-evi* se obrađuju slijedno



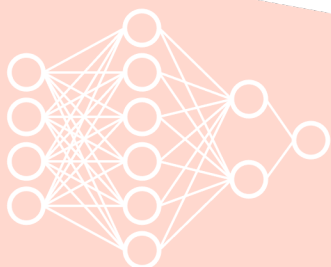
Paralelizacija?

- Što kada imamo veliki prostor značajki?
 - Svaki prolaz po uzorku iz skupa za učenje traje dovoljno dugo, što utječe i na SGD
 - Spas leži u optimizaciji i paralelizaciji matematičkih operacija
 - Linearna algebra – produkti nad matricama
 - Arhitektura procesora
 - ILP (*Instruction level parallelism*) – Paralelno izvođenje instrukcija u CPU-u. RISC arhitekture su tu u prednosti u odnosu na CISC
 - SIMD (*Single-instruction multiple-data parallelism*) – Jedna instrukcija koja barata s vektorom podataka
 - Kod standardnih CPU-ova postoje određene nadogradnje koje omogućavaju ovakve instrukcije. Recimo kod x86 to je MMX, SSE i slično
 - Ovdje primat ipak uzimaju grafički procesori (GPU)

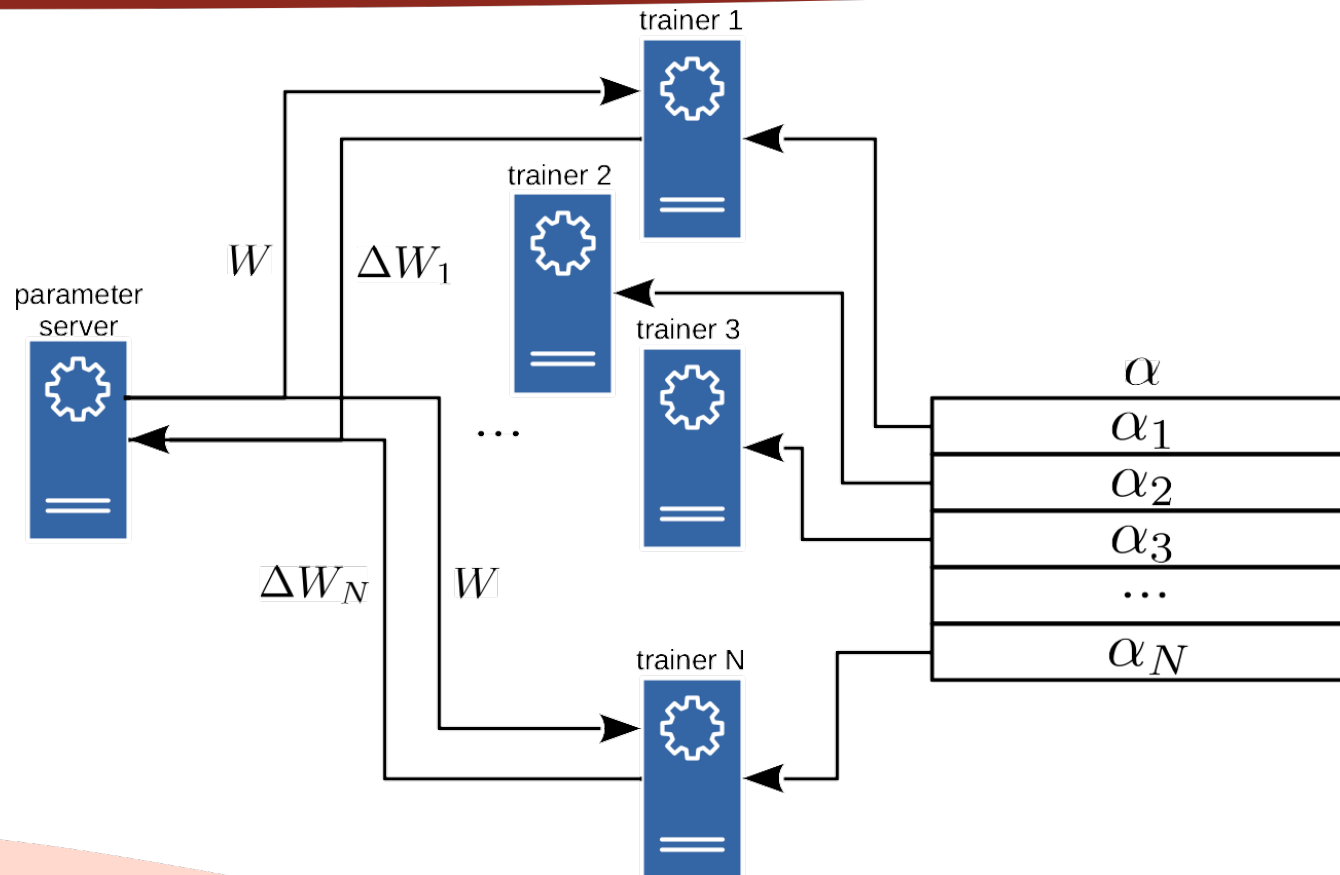


Paralelizacija?

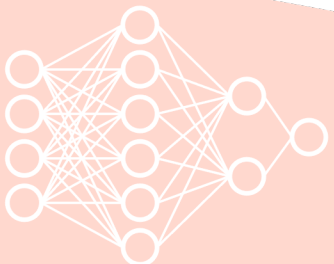
- A što kada imamo velike skupove podataka za učenje?
 - *Multi-threading* paralelizam
 - Više procesora i radnih dretvi (*thread*) koji koriste dijeljene memorijske segmente
 - Učenje kroz segmentaciju neuronske mreže – paralelizacija modela
 - Raspodjela slojeva kroz niz procesora – GPU cjevovod
 - Distribuirana računala
 - Više računala koja komuniciraju kroz mrežu
 - Učenje kroz horizontalnu segmentaciju skupa podataka za učenje (*parallel SGD*)
 - Skup podataka horizontalno segmentiramo – svaki segment predstavlja *mini-batch*
 - Svaki *mini-batch* predamo jednom čvoru (*trainer*) na učenje nad istim početnim modelom
 - Ažuriranja hiperparametara od svakog se čvora skupe u glavni čvor (*parameter server*) – radi se agregacija promjena hiperparametara
 - Novi model se dijeli prema svim čvorovima
 - Nova epoha
 - Sinkronizacija čvorova je usko grlo pristupa – Što ako jedan čvor konzistentno bude sporiji od 200 drugih?



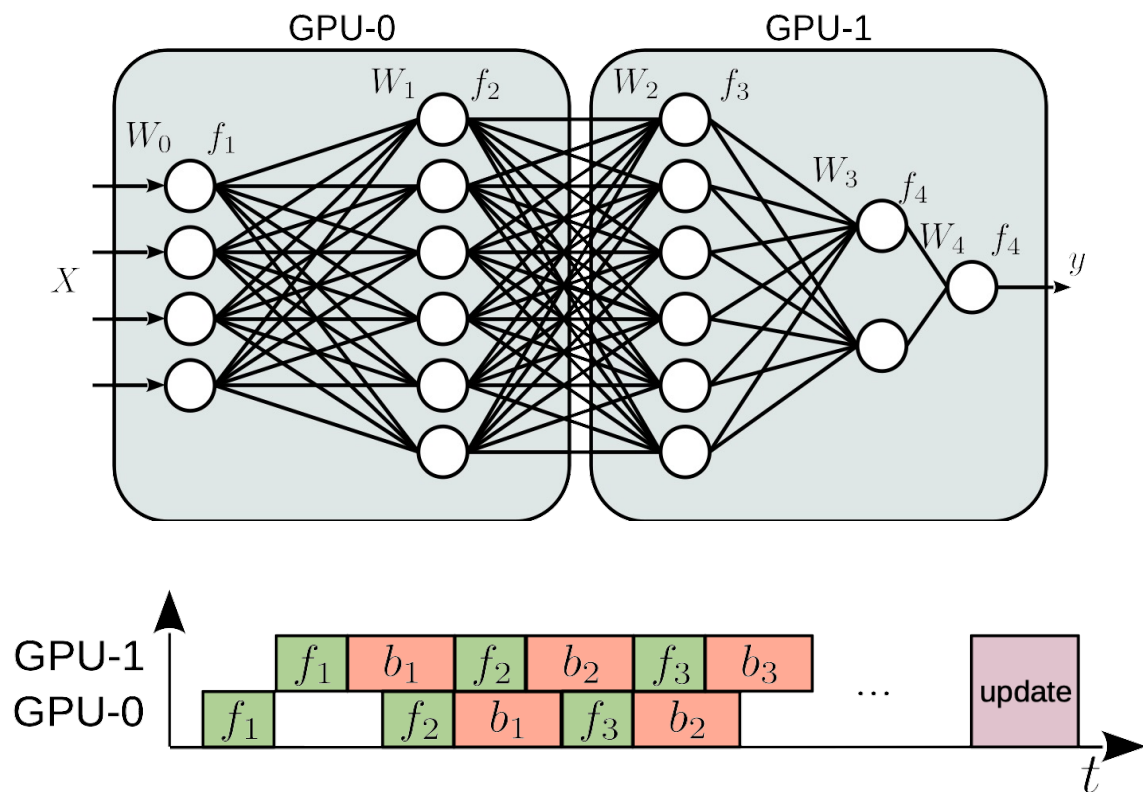
Podatkovna paralelizacija?



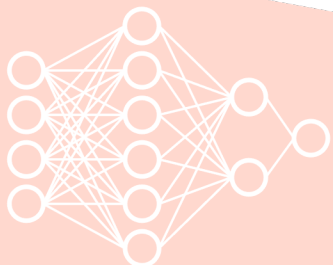
- Što se dešava ako imamo segment podataka koji u sebi sadrži samo jednu labelu?
 - Lokalno nebalansirani skup podataka za učenje
- Problem bizantskog ispada (*Byzantine trainer*)
 - Imamo čvorove koji smetaju i rade kontraproduktivno



Paralelizam modela

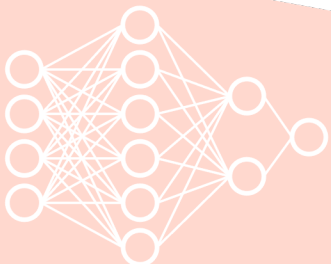


- Model se razbija na slojeve i jednako raspoređuje i GPU-ovima
- Svaki GPU slijedno radi jedan posao
- Ulančavamo *forward pass* na jednom GPU, pa na drugom
- Obrnuto ulančavamo *backward pass* na drugom, pa na prvom GPU
- Odradimo određeni broj *mini-batch-eva*, pa zatim ide sinkronizacija i ažuriranje hiperparametara
- Slijedi nova epoha



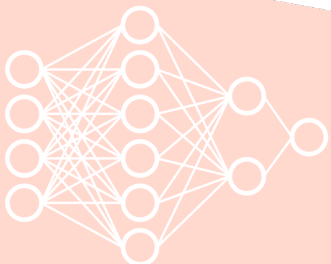
Praktični ANN paralelizam

- [PyTorch podatkovni paralelizam](#)
- [PyTorch paralelizam modela](#)
- [PyTorch RPC distribuirano učenje](#)



Paralelizam ostalih algoritama

- Regresija
 - Ima smisla jedino podatkovni paralelizam
- Algoritmi za grupiranje (*clustering*)
 - Teški za paralelizaciju
 - Slijed uzoraka definira pojavnost mikro i makro grupa
 - Kako agregirati dvije grupa s normalnom distribucijom?
 - Postoje mogućnosti kod metričkih algoritama za grupiranje
 - Podatkovni paralelizam
 - Zatim usklađivanje grupa između čvorova
 - Dosta otvorenog potencijala za znanstveno-istraživački rad



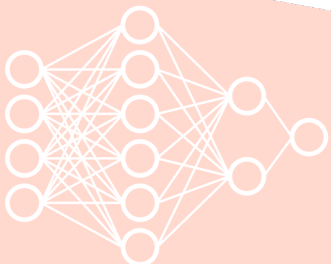
Repozitorij za eksperimente i učenje modela

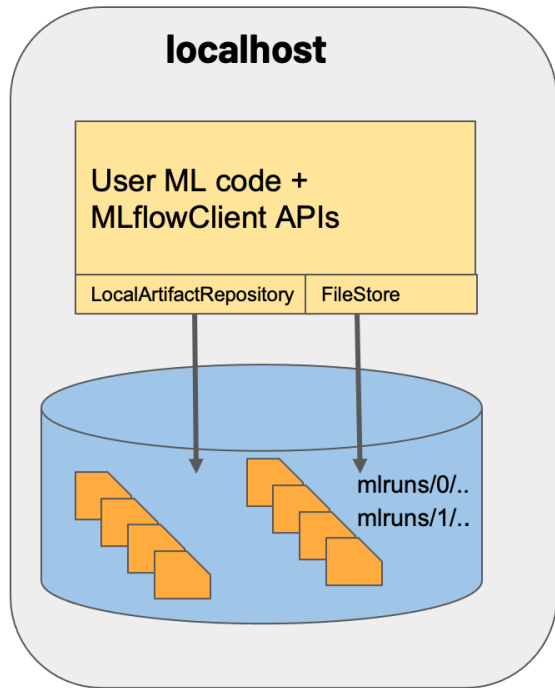
- Faza eksperimentiranja

- Razmatranje koji algoritam i arhitektura najbolje odgovaraju datom skupu podataka za učenje
- Verzioniranje i formalni proces učenja i „peglanja” modela nije problem
- Bitno je da ML inženjeri mogu komunicirati međusobno i dijeliti modele, kao i metriku

- Faza učenja

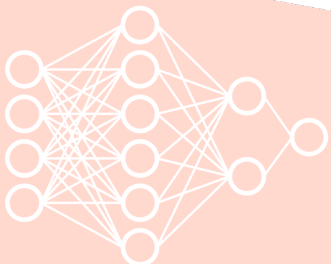
- U trenutku kada je cjevovod za učenje aktivan, i kada je učenje modela završeno imamo više zadataka
 - Provjeriti metriku – to se može automatski, zadacima u cjevovodu
 - Odraditi verzioniranje modela
 - Staviti verziju s dobrom metrikom na odobravanje
 - Ažurira se status modela
 - Cjevovod za učenje inicira spuštanje modela u repozitorij modela, u kojem ga može pokupiti poslužitelj modela



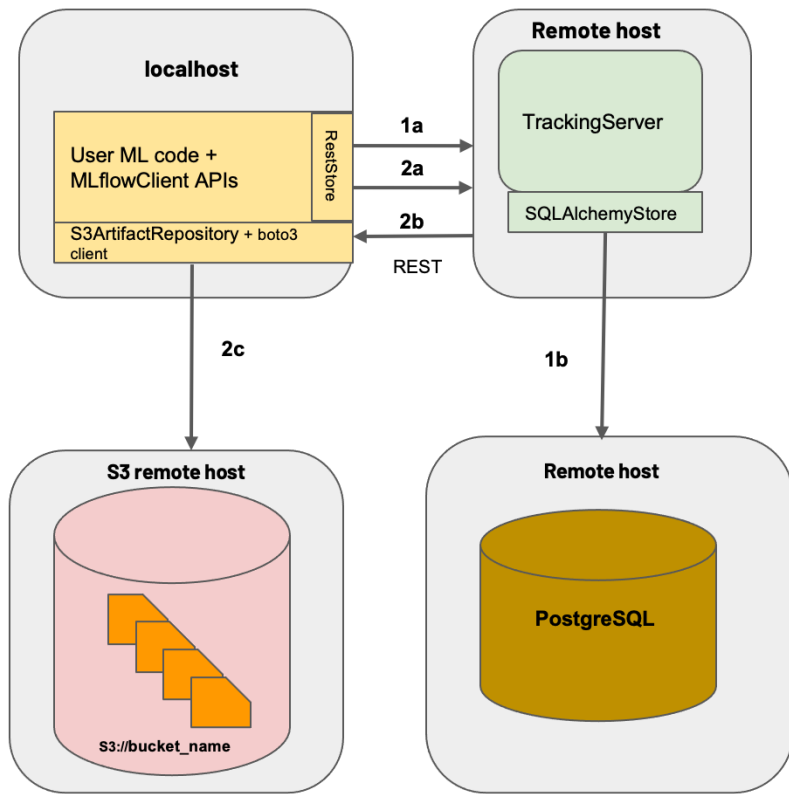


- python modul koji instaliramo s

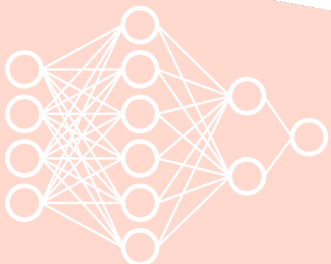
```
pip install mlflow
```
- Arhitektura 1 – rad na datotekama vezanim uz projekt na disku
 - Koristimo python API, ne definiramo REST API URL kod rada s mlflow-om
 - Tracking API pozivi završavaju u datotekama *mlruns* direktorija u projektu
 - U direktoriju se može startati server s `mlflow server`, nakon čega ga možemo otvoriti s browserom na portu 5000



MLflow arhitekture

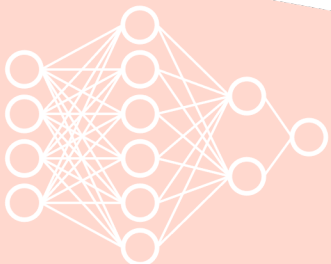
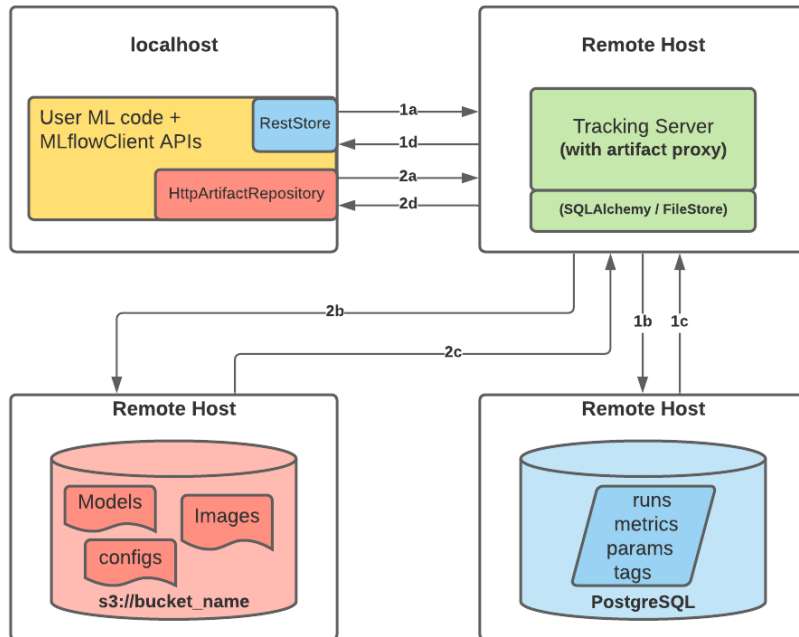


- MLflow server je povezan sa svojim repozitorijem koji se nalazi u bazi podataka (tipično MySQL ili PostgreSQL)
- Na MLflow server se povezujemo kroz mrežu korištenjem REST API-a
- U kodu definiramo URL MLflow servera
- Modele možemo spremiti u eksperimente (projekte) na MLflow serveru
- Za deployment model pohranjujemo direktno u *cloud storage-u* (AWS, GS, minIO)



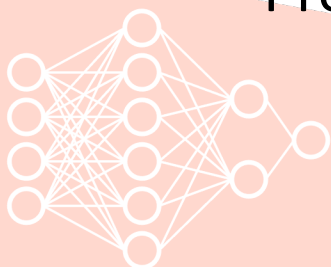
MLflow arhitektura

- Arhitektura slična prethodnoj
- Model se iz MLflow-a prebaci direktno u repozitorij modela (AWS, GS, minIO)
- Verzioniranje modela
- Status modela

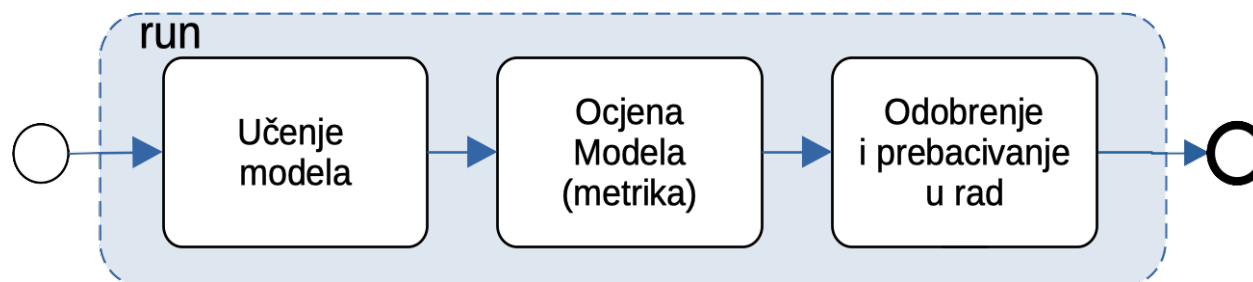


MLflow osnove

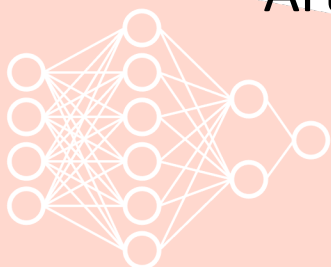
- MLflow server se koristi za
 - Definiranje projekta / eksperimenta
 - Definiranje jednog "pokretanja" (*run*) unutar tog projekta
 - Bilježenje parametara i metrika za modele
 - Bilježenje rezultata rada nad modelima
 - Spremanje raznih artefakata
 - Slike, grafovi, datoteke, konfiguracije
 - Struktura foldera
 - Modeli
 - Spremanje komentara, verzija i statusa modela
 - Prebacivanje modela u repozitorij modela (AWS, GS, minIO)



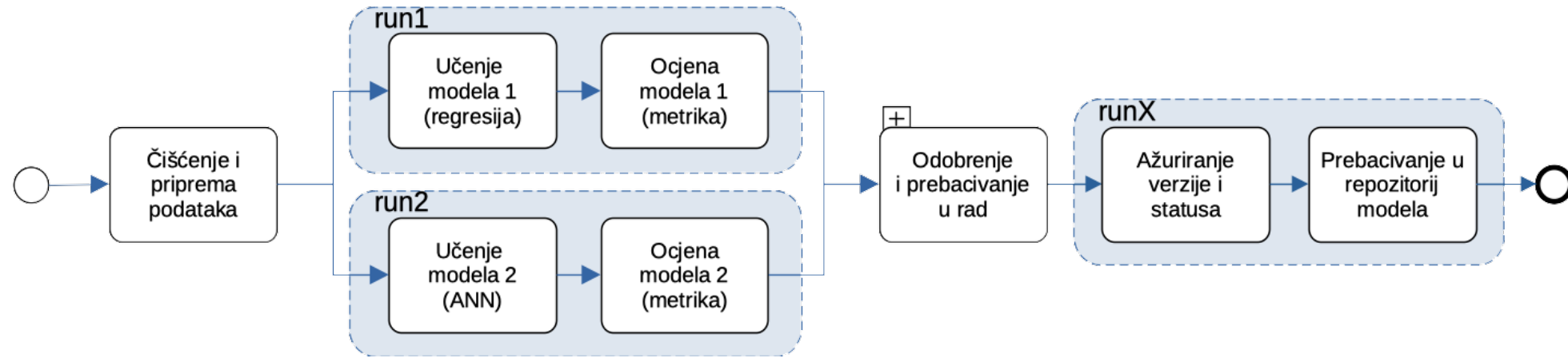
MLflow pokretanje



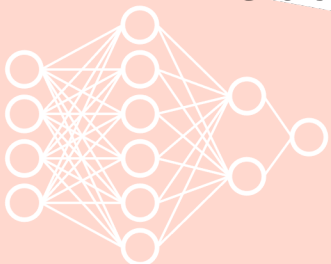
- Niz koraka koji se spoje u jednu cjelinu
 - U MLflow spremamo parametre, metriku, grafike, model, komentare
 - Mijenjamo verzije u ovisnosti u metrici
 - Odobravamo i prebacujemo u repozitorij modela (AWS, GS, minIO)
 - Workflow koristi MLflow da spremi produkte svojeg rada
 - Artefakti spremljeni u MLflow mogu utjecati na workflow



Cjev... za uče...



- U cjevovodu možemo kombinirati pokretanja kako nam je potrebno
- Primjer: odabir modela koji najbolje odgovara ulaznim podacima
- Odobrenje, ažuriranje verzija, statusa
- Prebacivanje u rad



MLflow API – učenje modela

- Definiramo tracking server URL
- Potražimo ili stvorimo naš projekt / eksperiment
- Definiramo shemu ulaz / izlaz
- Stvorimo *run* i u njega s *log_model* upišemo model koji smo naučili
- Na kraju definiramo verziju i stage

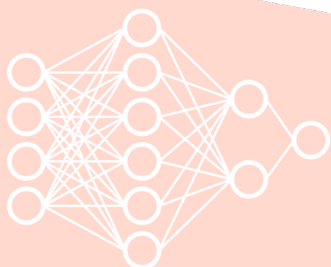
```
mlflow.set_tracking_uri("http://localhost:5000")
exps = mlflow.search_experiments(filter_string="name LIKE '%Diabetes%'")
if len(exps)==0:
    exp_id = mlflow.create_experiment("Linear regression - Diabetes dataset")
else:
    exp_id = exps[0].experiment_id
```

```
lr = LinearRegression()
lr.fit(train_x, train_y)

schema = infer_signature(train_x, train_y)
with mlflow.start_run(run_name="Training diabetes linear regression model",
                    experiment_id=exp_id) as run:
    mlflow.sklearn.log_model(lr, artifact_path="sklearn-model", signature=schema)

name="Diabetes linear regression model"
client = MlflowClient()
rms = client.search_registered_models("name='"+name+"'")
if len(rms)==0: client.create_registered_model(name)

model_uri = "runs:/{}/sklearn-model".format(run.info.run_id)
model_src = RunsArtifactRepository.get_underlying_uri(model_uri)
mv = client.create_model_version(name, model_src, run.info.run_id,
                                description="Linear regression model for diabetes dataset")
mlflow.end_run()
```



MLflow API rezultat učenja

[Linear regression - Diabetes dataset](#) >

Training diabetes linear regression model

Run ID: 578c10b24b684aa3a936e4891fc0d90e

Date: 2023-04-07 08:35:15

Status: FINISHED

Lifecycle Stage: [active](#)

Artifacts

- sklearn-model
 - MLmodel
 - conda.yaml
 - model.pkl
 - python_env.yaml
 - requirements.txt

Full Path: mlflow-artifacts:/1/578c10b24b684aa3a936e4891fc0d90e/artifacts/sklearn-model

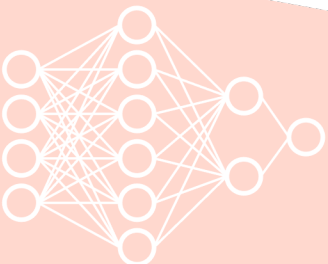
MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. [T](#)

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
Inputs (10)	
age	double
sex	double
bmi	double
bp	double
s1	double
Outputs (1)	
progression	double



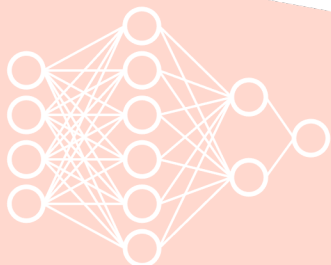
MLflow API – automatsko testiranje

- Učitamo model sa MLflow servera
- Stvorimo novi *run*
- Odradimo automatsku evaluaciju na MLflow serveru s *evaluate* – šaljemo testni skup podataka
- Završimo aktualni *run*

```
name="Diabetes linear regression model"
client = MLflowClient()
models = client.get_latest_versions(name, stages=["None"])

with mlflow.start_run(run_name="Testing diabetes linear regression model",
                      version:"+models[0].version, experiment_id=exp_id) as run:
    mlflow.evaluate(models[0].source, test, targets="progression",
                    model_type="regressor", evaluators=["default"])

mlflow.end_run()
```



MLflow API – automatsko testiranje

Linear regression - Diabetes dataset >

Testing diabetes linear regression model, version:1

Run ID: 4733b2089f9b4710bf59338b338d4d99

Date: 2023-04-07 08:35:20

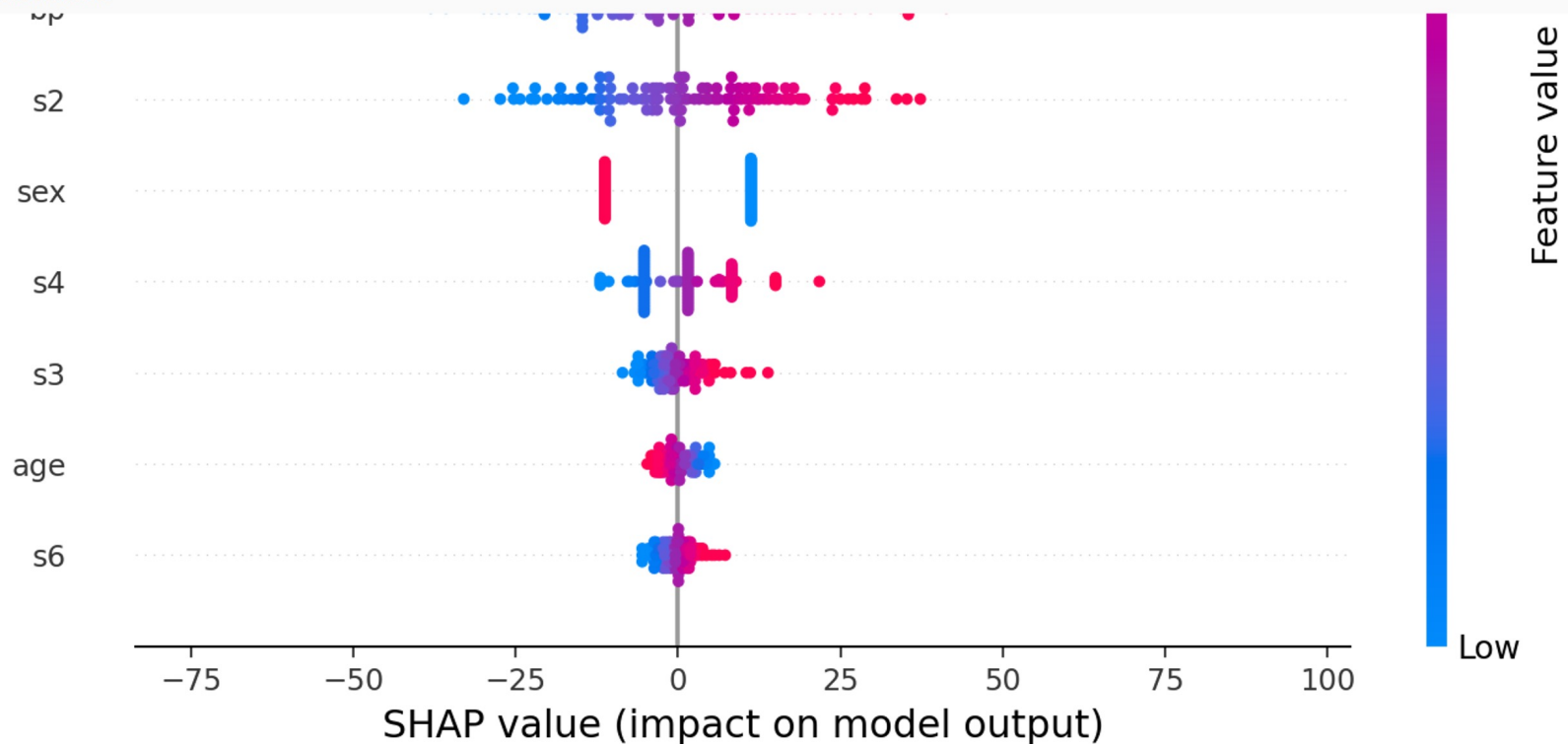
Status: FINISHED

Lifecycle Stage: active

Artifacts

- shap_beeswarm_plot.png
- shap_feature_importance_plot.png
- shap_summary_plot.png

Full Path: mlflow-artifacts:/1/4733b2089f9b4710bf59338b338d4d99/artifacts/shap_summary_plot.png
Size: 123.96KB



MLflow API – ručno testiranje

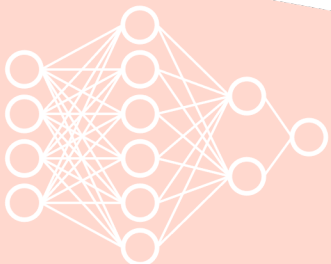
- Učitamo model sa MLflow servera
- Odradimo testiranje
- Sa *log_metric* upišemo određenu metriku

```
name="Diabetes linear regression model"
client = MlflowClient()
models = client.get_latest_versions(name, stages=["None"])

with mlflow.start_run(run_name="Manual test of diabetes linear regression model, version:" +
                      models[0].version, experiment_id=exp_id) as run:
    lr = mlflow.sklearn.load_model(models[0].source)
    predict = lr.predict(test_x)
    (rmse, mae, r2) = evalMetrics(test_y, predict)
    mlflow.log_metric("RMSE", rmse)
    mlflow.log_metric("MAE", mae)
    mlflow.log_metric("R2", r2)
mlflow.end_run()
```

▼ Metrics (3)

Name	Value
MAE 	44.15
R2 	0.557
RMSE 	55.02



MLflow API - PyTorch

```
with mlflow.start_run(run_name="Training CIFAR10 CNN",
experiment_id=exp_id) as run:
    model = ConvCifar10Net()

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr = 0.001, momentum = 0.5)
    mlflow.log_param("learning_rate", 0.001)
    mlflow.log_param("momentum", 0.5)

    epochs = 10
    mlflow.log_param("epochs", epochs)

    dataiter = iter(trainloader)
    images, labels = next(dataiter)
    plt = imshow(images)
    mlflow.log_figure(plt.gcf(), "examples.png")
    plt.close()

    schema = None

    for epoch in range(epochs):
        running_loss, met_loss = 0.0, 0.0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data

            optimizer.zero_grad()

            outputs = model(inputs)
            if schema is None:
                schema = infer_signature(inputs.detach().numpy(), outputs.detach().numpy())
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
        running_loss += loss.item()
        met_loss += loss.item()
        if i % 50 == 49:
            mlflow.log_metric("loss", met_loss/50, (epoch*len(trainloader))+i)
            met_loss = 0.0
        if i % 2000 == 1999:
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

    mlflow.pytorch.log_model(model, artifact_path="pytorch-model", signature=schema)
    name="CIFAR10 CNN"

    client = MlflowClient()
    rms = client.search_registered_models("name='"+name+"'")
    if len(rms)==0: client.create_registered_model(name)

    model_uri = "runs:/{}/pytorch-model".format(run.info.run_id)
    model_src = RunsArtifactRepository.get_underlying_uri(model_uri)
    v = client.create_model_version(name, model_src, run.info.run_id,
description="CNN model for CIFAR10")

    mlflow.end_run()
```

- Primijetite korištenje *log_param*, *log_metric* i *log_figure*

MLflow API – PyTorch učenje

Parameters (3)

Name	Value
epochs	10
learning_rate	0.001
momentum	0.5

Metrics (1)

Name	Value
loss 	1.067

Tags

Artifacts

pytorch-model

data


MLmodel

conda.yaml

python_env.yaml

requirements.txt

examples.png

Full Path: mlflow-artifacts:/2/4573ee33b87c41baa25b991601f95fa4/artifacts/pytorch-model 

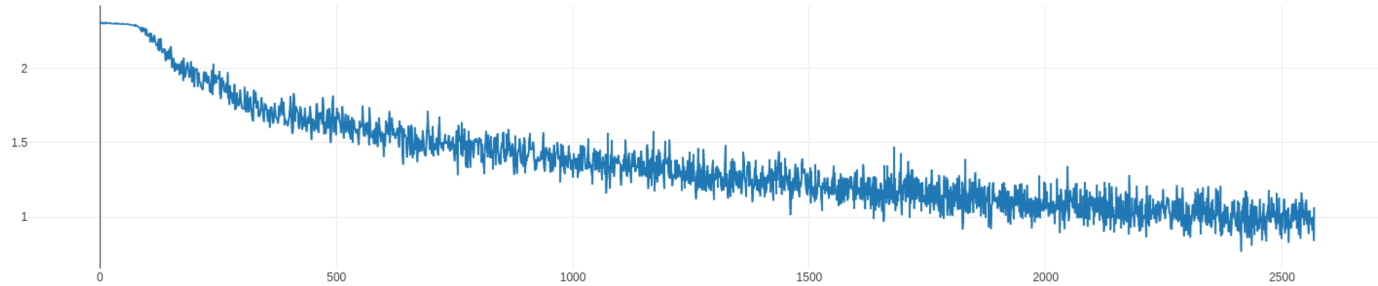
MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. [TI](#)

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
Inputs (1)	
-	Tensor (dtype: float32, shape: [-1,3,32,32])
Outputs (1)	
-	Tensor (dtype: float32, shape: [-1,10])



Metric	Latest	Min	Max
loss	1.188 (step=61249)	0.768 (step=115199)	2.31 (step=749)

Evaluacija

- Razne metrike
 - Ponovite gradivo iz Strojnog Učenja
 - [scikit-learn metrika](#)
 - [PyTorch metrika](#)

