ODSL Machine Learning Block Course Week 2

Intro MLP

Aufgabe 1: KL Divergence (Tutorial)

- Derive a closed form expression for the KL divergence between two 1-dimensional Gaussians $KL(p||q)$ with two distributions $p(x) = N_1(\mu_1, \sigma_1)$ and $q(x) = N_1(\mu_2, \sigma_2)$
- Take two Gaussians with $N_1(\mu = 1, \sigma = 1)$ and $N_2(\mu = 1.5, \sigma = 0.2)$ and compute the divergences $KL(N_1||N_2)$ and $KL(N_2|N_1)$. Discuss the Result.
- Research the concept of "support of a distribution". How does the KL-divergence KL(p||q) behave in the case where there is no support of q(x)

Aufgabe 2: Likelihood Ratio (Tutorial)

- For binary classification and using Bayes' Theorem, write out the posterior distribution of p(class = 1 | x)
- Assuming a "balanced" dataset (prior p(class = 1) = p(class = 0)) simplify the above expression
- Write the posterior as a function of the Log Likelihood Ratio p(x| class = 1)/p(x class = 0)
- Discuss the relationship between the posterior and the log likelihood ratio. Does the functional form have a name?

Aufgabe 3: Optimization (Tutorial)

- Using PyTorch write the following function $f(\phi) = (x - x_0)^2 + 0.2(y - y_0)^2$ as a functionof the vector-like parameter $\phi = (x, y)$ with $x_0 = 1, y_0 = 2$
- Evaluate the function on the space -2 < x < 4 and -2 < y < 4 and draw the iso-lines for constant f
- Using PyTorch's `.backward()` API, implement a manual version of gradient descent of the function from the last section.

Tip: the parameter update is best done using the following

```
with torch.no_grad():
    pars = .... new parameters
    pars.requires_grad = True
```

Write the function such that you track the intermediate positions (x, y) and the function value

- Run Gradient Decent from a few initializations points and plot the trajectories
- Perform an Optimization on the above problem using the torch.nn.Adam optimizer and compare it to the manual gradient descent

Aufgabe 4: Binary Classification on Infinite Data (Tutorial)

- Write a function `get_data(N)` that produces samples and labels from two multivariate Gaussians
  - 1 Gaussian with `mean = [0,0]` and covariance matrix `cov = [[1,0],[0,1]]`, these samples should be labeled `0`
  - 1 Gaussian with `mean = [1,1]` and covariance matrix `cov = [[1,0.8],[0.8,1.0]]`. These samples should be labeled `1`

  The output of the function should have the following shapes:
  - samples = `(2*N,2)`
  - labels = `(2*N,1)`

  where `N` is the number of samples per Gaussian
- Draw a sample of `N=1000` and visualize it in the `(x1,x2)` and color the markers according to the label
- Write a logistic regression $p(z|f_{W,b}(x))$ with

$$f_{W,b}(x) = \sigma(Wx + b)$$

  as a PyTorch module
- Write a function `train_model_infinite_data(model,nsteps)` using the PyTorch's Adam optimizer and learning rate 1e-3 that trains the logistic model as a classification model (i.e. the cross entropy loss). The optimization should proceed for `nsteps` steps and return the trained model as well as the progression of loss values.

  For each step of the optimization draw a new independent sample using `get_data` (i.e. this is the case where you would have infinite data)
- Train the linear regression model using `train_model_infinite_data` and plot the loss as a function of optimization steps.
- Write a function `plot_decision_boundary` that plots a contour plot for a given f_ϕ. Visualize the trained model from above.
- Using the `torch.nn.Sequential` API, build a deep network and repeat the above analysis.
- Compare the Performance in the infinite data limit between the shallow and deep model. Discuss the results. Which one is better? Why does there seem to be a lower bound for the deep model?
- Write a function `def log_lhood_ratio` that computes the likelihood ratio for a given point x. Create a scatter plot of the likelihood ratio vs the two model output probabilities

Aufgabe 5: Binary Classification Finite Data (Homework)

- Take again the setup from Excercise 4. Adjust the training function to take a tuple (X,y) as training data. The model should only be trained in each iteration on this one sample of training data. Track the performance for a large independent "test sample" during the training loop (use `torch.no_grad` to avoid interfering with the training.
- Train the logistic regression model using a finite training sample of 500 samples. Plot the train and test losses.
- ... as well as for the deep network
- Plot the decision boundary of the deep model. Discuss the result.
- Plot a comparison of both the train and test losses of the shallow and deep model. Discuss the results.
- Try adjusting the Training Sample size such that the Deep Model exceeds the shallow one in performance

Aufgabe 6: Depth vs Width (Homework)

- Consider the funcion

  $$f(x) = f(x_1, x_2) = \sin(5x_1) + \cos(8x_2)$$

  and the binomial probability model

  $$p(x, z) = p(z|x)p(x) = Bin(z|f(x))p(x)$$

  with a data distribution $p(x) = \text{Uniform}(-2, 2)$

  Write a function `get_data` that samples from the above joint distribution, returning features x and labels z

- Sample a dataset of 1000 points and visualize it.
- Train the following sequence of models on the binary classifcation task
  - A deep model with layers (2, 20, 20, 20, 20, 1)
  - A series of shallow model with layers (2, N, 1) with N = [300, 500, 1500, 7000]

  Collect the progression of loss function values for each of these models
- Plot the loss curves and discuss the results.

CNN

Aufgabe 7: Convolutions (Tutorial)

- Write a function that downloads an image at a given URL and converts it into greyscale and returns a 2D array
- Implement a function that performs a 3x3 convolution with zero padding and stride 1 by hand
- Pick a image URL and run both a vertical and a horizontal edge detector on your image
- Create a 2D convolution object (Conv2D) with 1 input channel and 2 output channels
- Research how to access the parameters of the convolution kernel
- Manually set the weights of the kernel such that the first output channel is a vertical edge detector and the second output channel is a horizontal edge detector
- Run your image through the PyTorch convolution and visualize the output channels

Aufgabe 8: Training ConvNets (Homework)

- Obtain the MNIST dataset (using `pip install mnist`) and plot a few examples from the training set
- Create Two PyTorch models that can take a batch of MNIST images in the form of a flattened vector (N, 768) and output logits for 10 classes
  - As a first model use a densely connected feed-forward model with 3 hidden layers of 100 units and ReLU activation
  - As a second model use a Convolutional Neural Network with 3 convolutional blocks (convolution, Relu, maxpool) with convolution kernel sizes of 5,3 and 2 and a maxpool kernel size of 2. Followed by a Linear projection layer to the 10 class labels

  Report the number of parameters for both models.
- Train a Convolutional Neural Network and a MLP on MNIST with 2D Convolutions and Max-Pooling with the Adam optimizer for 1000 steps. and learning rate 1e-3. Track the train and test loss for both models as you train.
- Plot the train and test loss as a function of number of training steps for both the CNN and MLP trainings. Discuss the results
- Try to increase the MLP size to match the CNN performance. Report the number of parameters of the model that comes closest to or matches the CNN performance

Aufgabe 9: Density Transformations (Tutorial)

- Consider a Standard Normal 2-D Gaussian $x \sim \text{Normal}(x, \Sigma_x)$

  Given a Matrix A and a vector b, we can look at the random number $y = f(x) = Ax + b$.

y will also be normally distributed. Derive the mean and covariance of $p(y) = N(\mu_y, \text{Sigma}_y)$

- To validate the above result, draw a sample of $p(x) = \text{Normal}(⊬, ⊮_2)$, transform it with $f(x)$, with `A = [[1,0.2],[0.2,1]]` and `b = [1,2]` and draw a sample from p(y).
- Go in the reverse direction. Sample $y \sim p(y)$, run them through the inverse transformation f^{-1} and compare the distribution of the transformed variables to the distribution of $p(x)$

Aufgabe 10: Normalizing Flows (Homework)

- Consider a N=1000 sample from mean = [1,2] and cov = [[0.5,0.3],[0.3,0.3]]. Write code that generates such a sample and visualize it on the 2-D plane
- Write two PyTorch functions that applies a affine transformation y = Ax + b and its inverse $x = f^{-1}(y)$ to a batch of 2-D points for a given matrix A and bias * Derive an expression for the Jacobian $\partial y / \partial x$ and write a function that computes the log abs determinant of the Jacobian for a given affine transform
- Using the Change of Variables Formula and the `torch.distributions` module write a function that evaluates the log-probability of a sample under the distribution $p(y = f_{A,b}(x))$ with $p(x) = \text{Normal}(0, 1)$
- Create a PyTorch module that has Affine Transformation parameters (A,b) and provides two methods: 1) "log_prob" for a batch of samples and "generate" to generate transformed samples starting from a base distribution
- Create a plotting function `plot(model)` that plots in three panes: a) a scatter plot of base distribution samples and transformed distribution samples (x1,x2), b) a histogram of the first coordinate of these samples and c) a histogram of the second coordinate of these samples. Initialize a Flow model and plot it using the function above
- Train a 2D Normalizing Flow q(x) using an affine transformation f(x) = Ax + b. Train it to maximize the likelihood. As you train, plot the current state of the model