Data Science | Lab 5: Artificial Neural Network

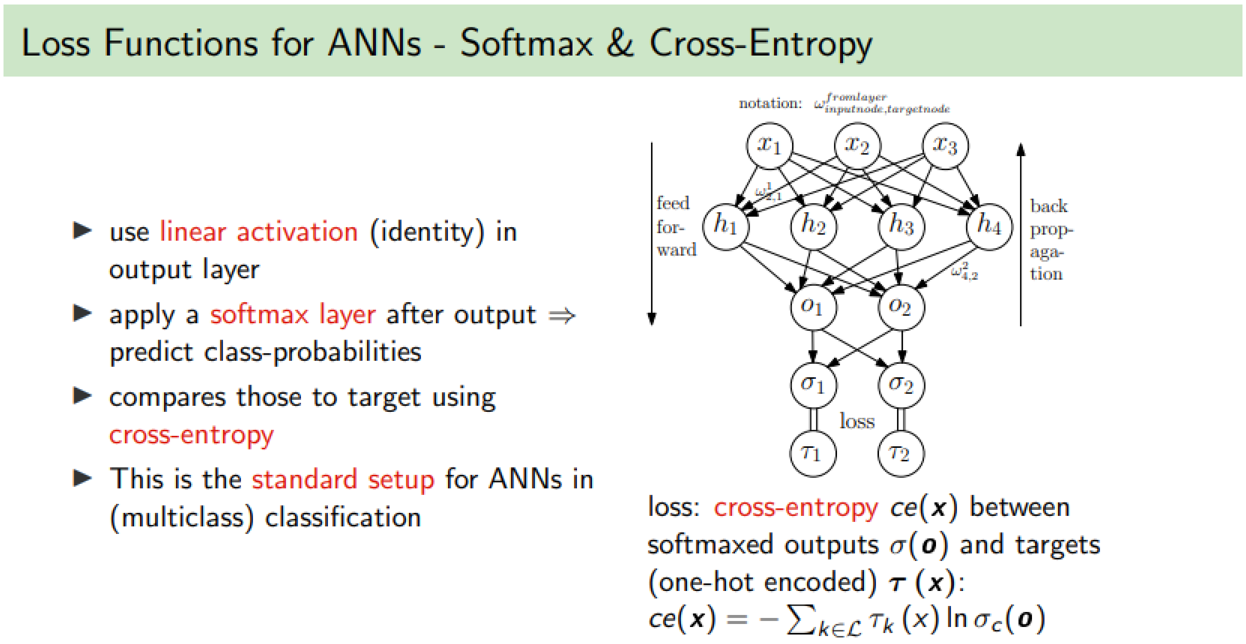
Learning Goals

- understand the parameters of an ANN
- configure and train an ANN
- understand effects of different optimization functions and learning rates
- interpret the (training) loss curve

Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is a fully-connected artificial neural network (ANN) consisting of at least one hidden layer.

Recall the principles of artificial neural networks from the lecture. Take a look at scikit-learn's `MLPClassifier` 📝 documentation and make yourself familiar with its usage.



Loss Functions for ANNs - Softmax & Cross-Entropy

- use linear activation (identity) in output layer
- apply a softmax layer after output ⇒ predict class-probabilities
- compares those to target using cross-entropy
- This is the standard setup for ANNs in (multiclass) classification

loss: cross-entropy $ce(\boldsymbol{x})$ between softmaxed outputs $\sigma(\boldsymbol{o})$ and targets (one-hot encoded) $\boldsymbol{\tau}(\boldsymbol{x})$:

$ce(\boldsymbol{x}) = -\sum_{k\in\mathcal{L}} \tau_k(x) \ln \sigma_c(\boldsymbol{o})$

Tasks

Design Decisions

👥 Group work: Discuss the parameter settings of the network

Discuss the following aspects of the network and refer to the network depicted above whenever possible:

- How many hidden layers are used?
- Which activation function should be used in the hidden layer(s)?
- Which activation function should be used in the output layer?
- Which loss function is used?

Refer to the 📝 documentation of the `MLPClassifier` and detect the parameters corresponding to:

- choice of optimizer
- L2 regularization
- initial learning rate
- early stopping and tolerance

- learning rate decay and momentum

First Experimentation

1. Make a simple train-test split of the breast cancer data (`test_size=0.1`) using `random_state=42`.
2. Implement a Multi-layer Perceptron (`MLPClassifier`) with one hidden layer consisting of `64` hidden nodes with a ReLU activation and a softmax output layer to compute the cross entropy loss. Use a batch size of `100`.

   Hint: You might need to refer to the source code to find out what kind of activation is used in the output layer.
3. Train the `MLPClassifier` with SGD optimizer and the default L2 regularization. No momentum (needs to be turned off).
4. Print/plot the classification report and the confusion matrix for the test data.
5. Plot the training loss curve.

Use the following function to plot the loss curve:

```
def plot_learning_curve(df_loss_curve, title='Loss curve - Breast Cancer Data'):
    sns.lineplot(data=df_loss_curve, dashes=False)
    plt.title(title)
    plt.show()
```

You can pass a data frame to the function that can hold one or more "loss curves" from different training settings, e.g.:

```
loss_curve = clf.loss_curve_ + [np.NaN] * (max_iter - len(clf.loss_curve_))
df = pd.DataFrame(loss_curve, columns=['SGD Baseline'])
plot_learning_curve(df)
```

6. Reusing the same train-test split, experiment with different learning rates (by changing the value of `learning_rate_init` in the range between `0.0001` and `0.1`) and interpret the changes in the loss curve.
7. Inspect the shape of the classifier's attribute `coef_` (the weights of the network) and interpret it.

Different Optimizers

Next, experiment with different optimizers and settings. Use an initial learning rate of `0.0001` and `alpha=0.001` throughout. Make sure to implement the following:

- `'SGD Baseline'`: SGD optimizer with default L2 regularization.
- `'SGD with momentum'`: SGD optimizer with default L2 regularization and a momentum of 0.9.
- `'SGD with decreasing lr'`: SGD optimizer with automatically decreasing learning rate.
- `'Adam'`: Adam optimizer with default L2 regularization and decay rates.

Plot all training loss curves in a single plot and note the differences.

Early Stopping Strategy

Adapt the different classifiers/optimizer settings from above such that they stop early if the validation loss is increasing. This is used to prevent overfitting. Use 10% of the data for validation. Interpret the results.

Grid Search

Similar to the last lab, implement a grid search with 5-fold cross validation using the `GridSearchCV` to find the best hyperparameter settings for the `MLPClassifier`. Make sure to include both the **Adam optimizer and the SGD variant with momentum** in the grid.

Decide about other parameters (and their value ranges) to add to the grid, e.g.:

- learning rate
- number of hidden nodes
- activation function in the hidden layer
- batch size
- number of hidden layers

No need to use them all, but include at least two of them! Make sure to use early stopping to limit the runtime.

Homework

1. Take the quiz in Moodle. Make sure to have your Python notebook open and your code up and running.

   🕘 Deadline: first lab in January

2. Do the Reading Assignments and answer the related questions that are provided in an extra PDF. When finished, take the related quiz in Moodle. Make sure to have the Reading Assignment "NLP" completed before the first lab in January, the Reading Assignment "IP" before the second lab.

   🕘 Deadline: first or second lab in January

Optional: Notebook Feedback

If you want to get (ungraded) feedback on your notebook and the way you do the assignments, you can upload your Notebook as `firstname_lastname_dslab5.html` to Moodle.

The (voluntary) submission should consist of a notebook (exported to html) of the last lab (Lab 5: Artificial Neural Networks) **starting from the subsection "Different Optimizers"**. Cell outputs (whenever needed) should be well formatted. Make sure to include a **discussion section** at the end of the notebook where you comment your results and findings.

This is an individual assignment. Collaboration in discussing the approach or some technical details is highly welcome, the uploads need to be prepared and uploaded individually. Make sure that essential original content (anything else than exchanging comments or texts such as headlines or discussion) distinguishes your upload from that of your colleagues significantly. Anything else is considered plagriasim.

Dos

- Make sure all cell outputs (e.g. plots) are visible.
- Use markdown for comments/interpretations.
- Self-contained plots (e.g. add title).
- Use section headers to organize your notebook.

Don'ts

- Wrong file type (e.g. .ipynb instead of .html).
- Unnecessary/duplicate/inactive cells.
- Unnecessary imports or helper functions.
- Comments on outputs/results in code cells (instead of markdown).
- Overlong print statement outputs.
- Print outputs without formatting or context. Don't just print plain numbers.
- Plagiarism!

🕘 Deadline: first lab in January