THE INTRODUCTION OF DEEP LEARNING

STAT499

Mentor: Dasha Petrov
Mentee: Jingyu Zhang

Content

- The Basic of Multilayer Perceptron
- A complete epoch / iteration
- Numerical Stability And Initialization
- Fix the overfitting problem

Predicting House Price

The Basic of Multilayer Perceptrons

A Multilayer Perceptron (MLP) is a basic neural network that consists of stacked layers of linear transformations followed by non-linear activation functions.

Activation Function

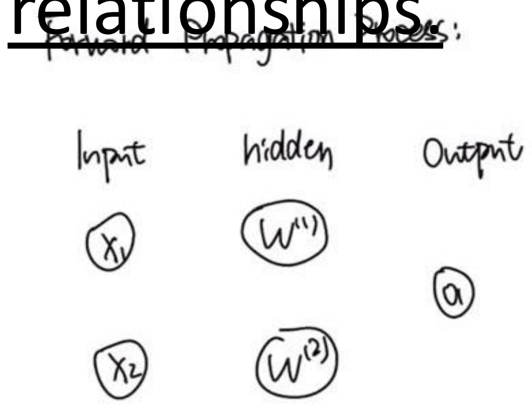
$$\hat{y} = W_2 \cdot \sigma(W_1 \cdot \mathbf{x} + b_1) + b_2$$

Where:

- $\mathbf{x}$: input vector
- W_1, W_2: weight matrices
- b_1, b_2: bias vectors
- σ: activation function (e.g. ReLU)

Why Multilayer Perceptrons?

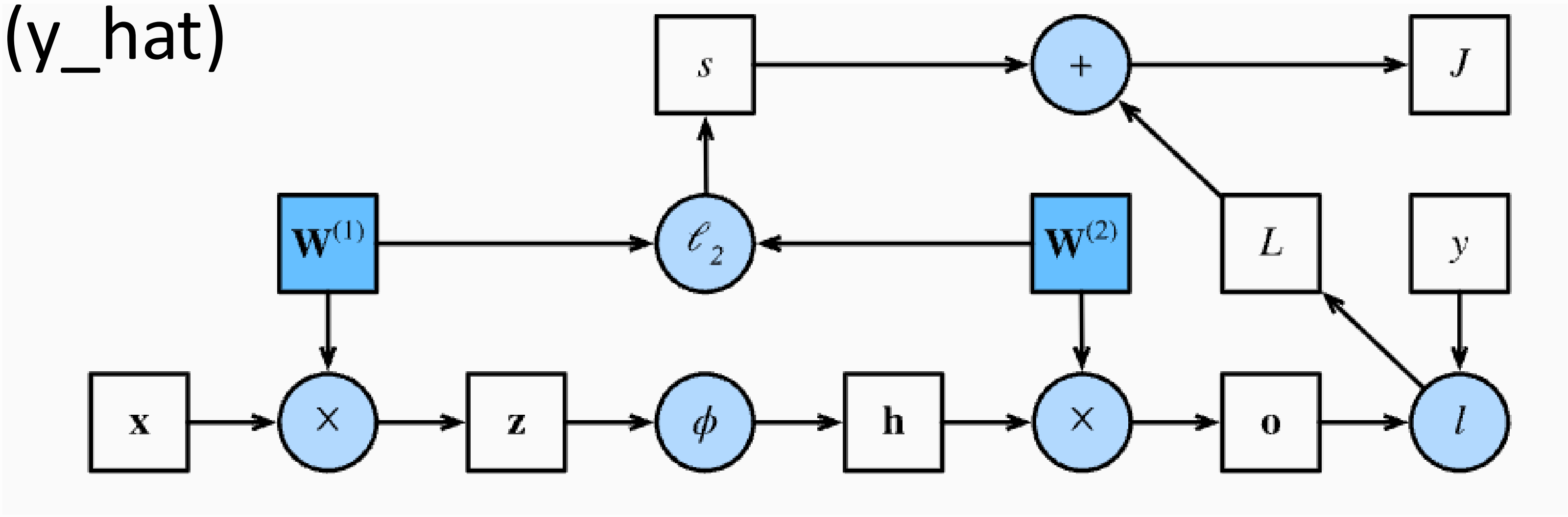
A network without hidden layers can only learn linear functions. Hidden layers allow the model <u>to capture more complex, non-linear relationships.</u>



A Complete Epoch/Iteration

Forward Propagation:

Pass the input through the model to get predictions (y_hat)




```
output = activation(Wx + b)
```

Loss Function:

Compare the predicted values y_hat with the true labels y

```
loss = loss_function(y_hat, y)
```

Backward Propagation:

Compute the gradients of the loss with respect to each parameter using the chain rule.

```
loss.backward()
```

Update the weights and Zero gradients:

Use optimizer updates the weights and biases using the gradients, which must be zeroed afterward to prevent accumulation in the next iteration.

```
optimizer.step()
```

```
optimizer.zero_grad()
```

Numerical Stability And Initialization

where gradients become too small for learning to happen → Vanishing Gradients Problem!

where gradients blow up and make training unstable → Exploding Gradients Problem!

$\boxed{\text{Xavier Initialization}}$

Numerical Stability And Initialization

Xavier Initialization

to control the variance of weights such that the activations stay in a healthy range across layers.

$$o_i = \sum_{j=1}^{n_\mathrm{in}} w_{ij} x_j.$$

$$\frac{1}{2} (n_\mathrm{in} + n_\mathrm{out}) \sigma^2 = 1 \text{ or equivalently } \sigma = \sqrt{\frac{2}{n_\mathrm{in} + n_\mathrm{out}}}.$$

$$\begin{aligned}
\mathrm{Var}[o_i] & = E[o_i^2] - (E[o_i])^2 \\
& = \sum_{j=1}^{n_\mathrm{in}} E[w^2_{ij} x^2_j] - 0 \\
& = \sum_{j=1}^{n_\mathrm{in}} E[w^2_{ij}] E[x^2_j] \\
& = n_\mathrm{in} \sigma^2 \gamma^2.
\end{aligned}$$

$$\begin{aligned}
E[o_i] & = \sum_{j=1}^{n_\mathrm{in}} E[w_{ij} x_j] \\
& = \sum_{j=1}^{n_\mathrm{in}} E[w_{ij}] E[x_j] \\
& = 0,
\end{aligned}$$

$$U\left(-\sqrt{\frac{6}{n_\mathrm{in} + n_\mathrm{out}}}, \sqrt{\frac{6}{n_\mathrm{in} + n_\mathrm{out}}}\right).$$

How to fix the overfitting problem?

Dropout

A regularization method that randomly “drops out” a subset of neurons during training.

Let:

- $\mathbf{X} \in \mathbb{R}^n$: input vector
- $p \in [0, 1]$: dropout rate (probability of dropping a neuron)
- $\mathbf{m} \in \{0, 1\}^n$: dropout mask where

$$m_i \sim \mathrm{Bernoulli}(1 - p)$$

Then the output after applying dropout is:

$$\tilde{\mathbf{X}} = \frac{\mathbf{m} \odot \mathbf{X}}{1 - p}$$

Where:

- $\odot$: element-wise multiplication
- $\frac{1}{1-p}$: scaling factor to maintain the expected value

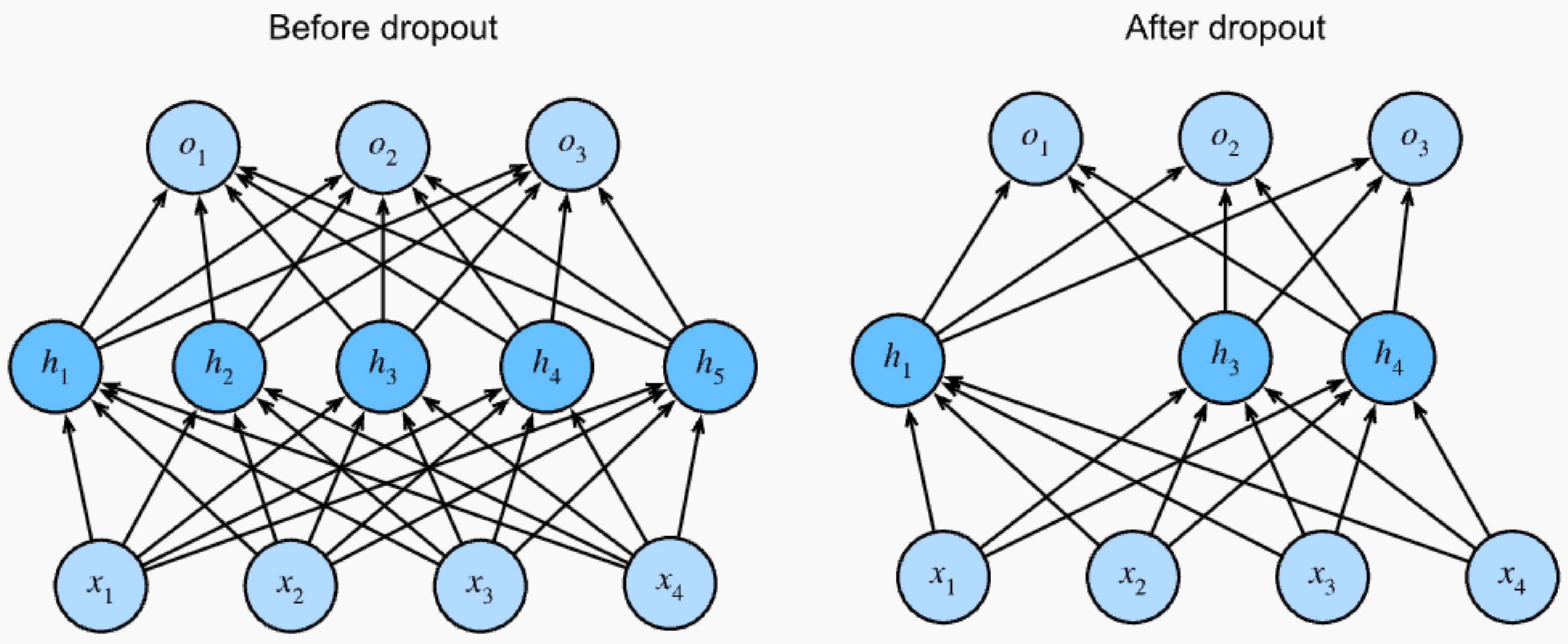


Fig. 5.6.1 MLP before and after dropout.

PREDICTING HOUSE PRICES

Project Overview:

Goal: predict house prices using the Kaggle House Prices dataset, and explore how model complexity impacts performance.

Dataset: 2006–2010 home sales data, includes 79 explanatory variables.

Method:

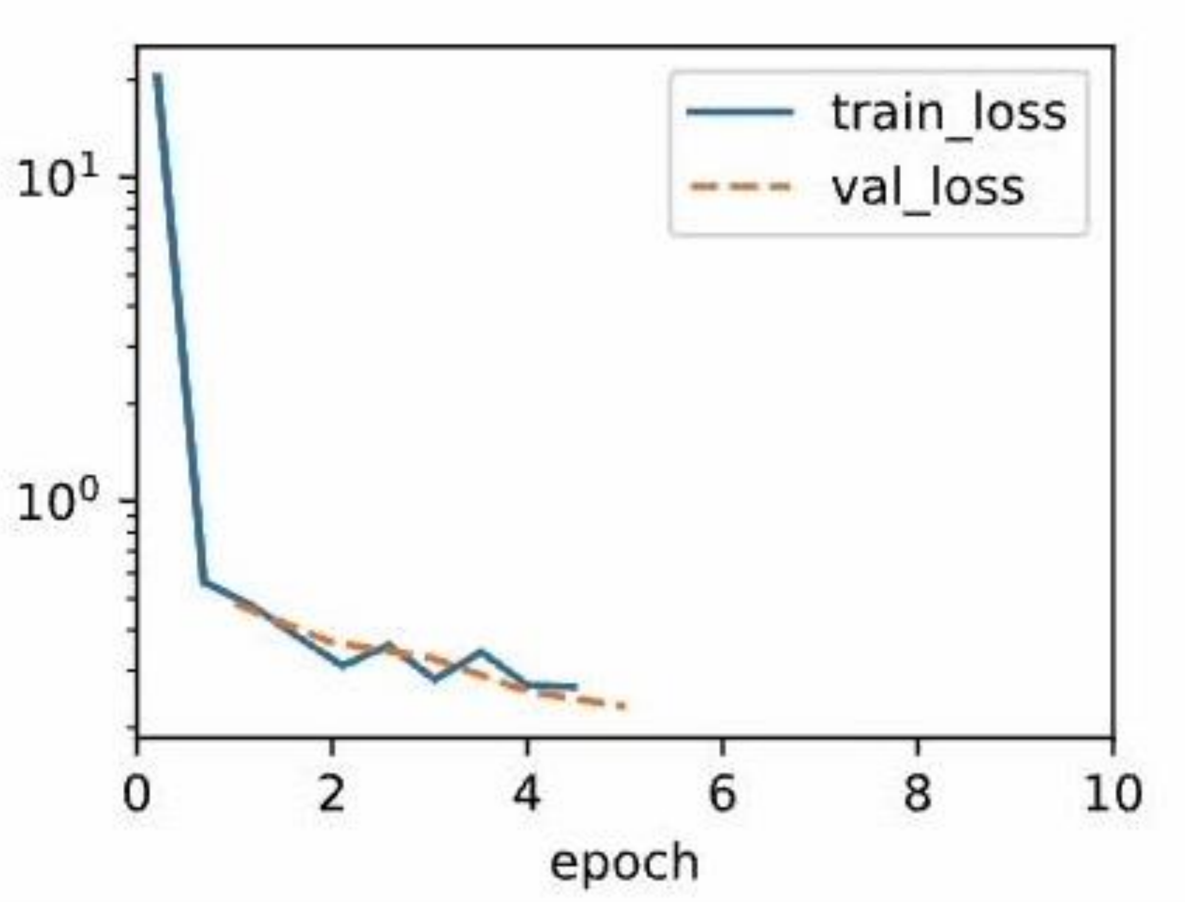
- K-Fold Cross-Validation (n=5)
- ReLu, L2 regularization and dropout

Implemented and compared Model:

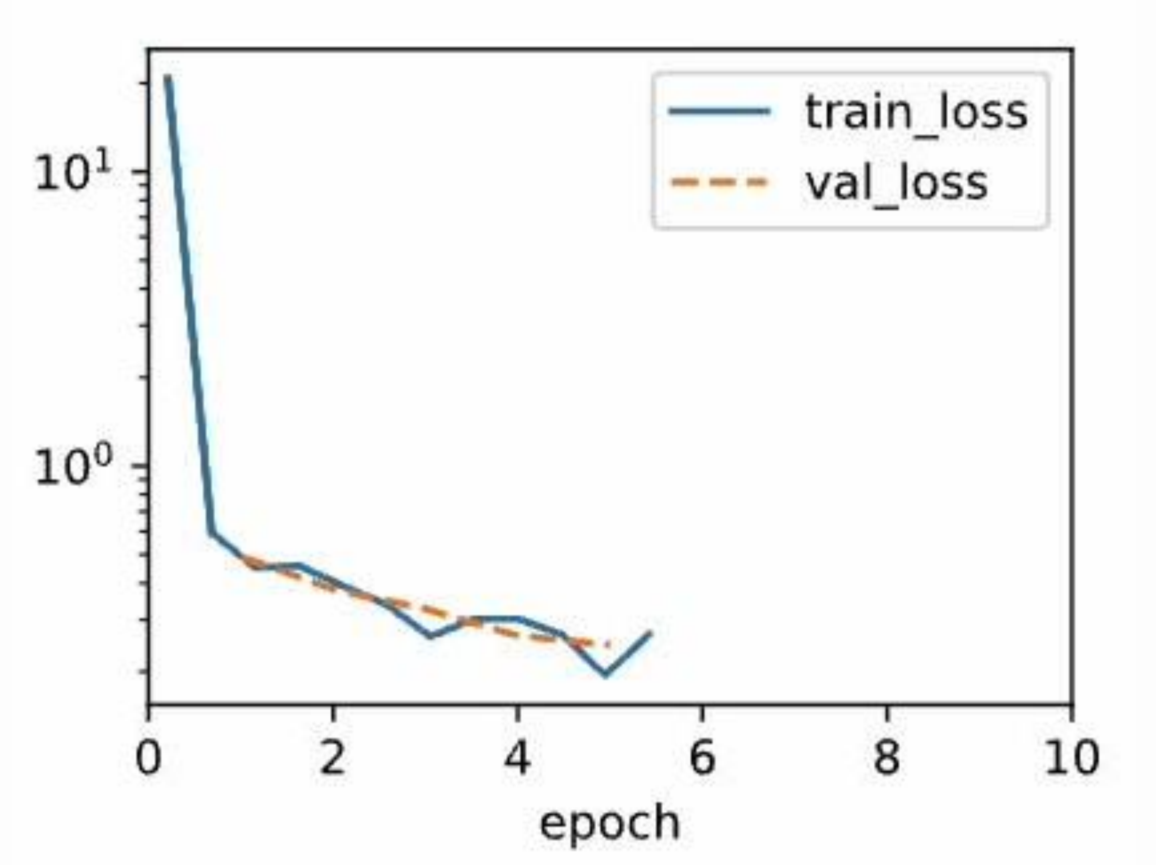
- Linear Regression(baseline)
- MLP
- MLP with ReLU, Dropout and L2

PREDICTING HOUSE PRICES-RESULT

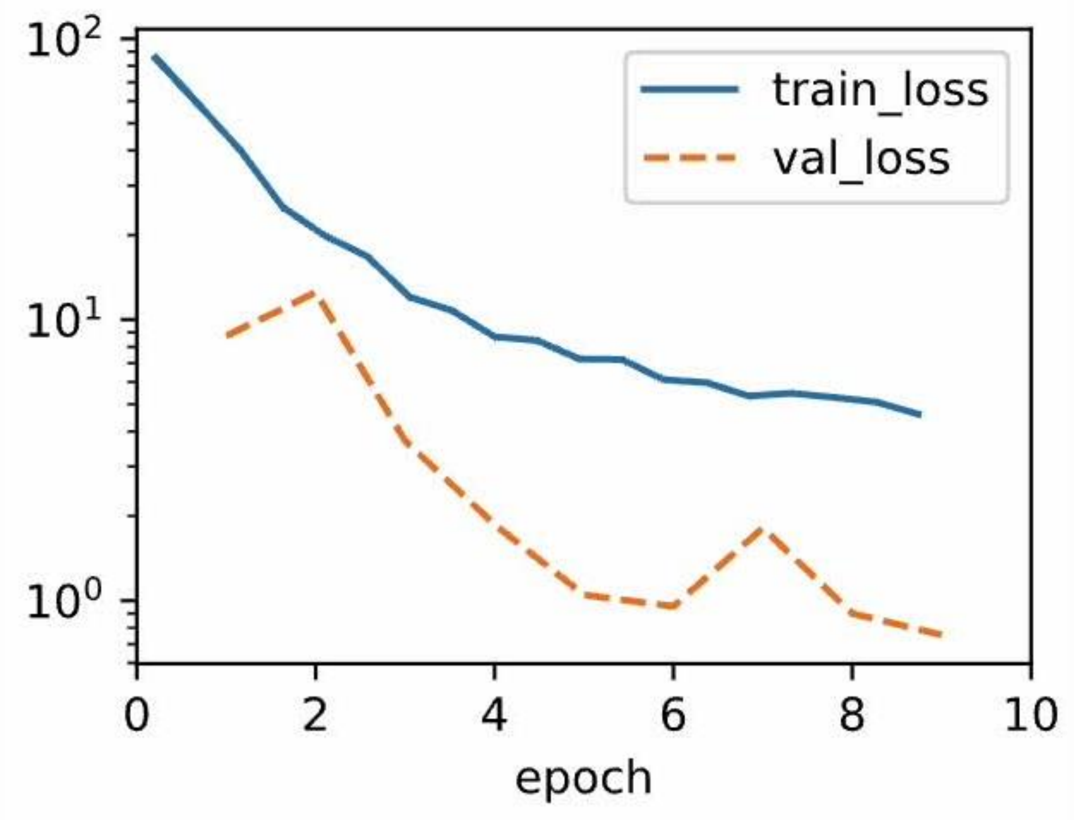
linear regression model



MLP



MLP with ReLU, Dropout and L2



- Linear Regression achieved the lowest validation MSE
- More complex models (NN with Dropout & L2) did not outperform the linear model

Thank You For Listening!

Thank You DRP Program

Thank You, My mentor 👉 DASHA!!!!!!