CPE 486/586: Deep Learning for Engineering Applications

04 Training Neural Networks and Computational Thinking

Spring 2026

Rahul Bhadani

Outline

1. **Recap**

2. **Training a Neural Network**

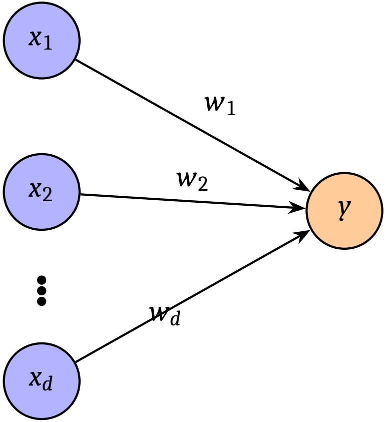
2.1 Data Preprocessing

2.2 Impact of Batch Size in the Training Process

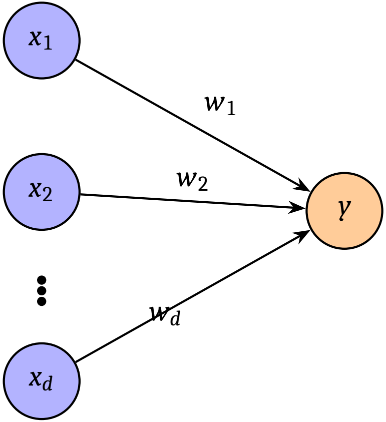
2.3 Preventing Overfitting and Regularization

Recap

Linear Layer

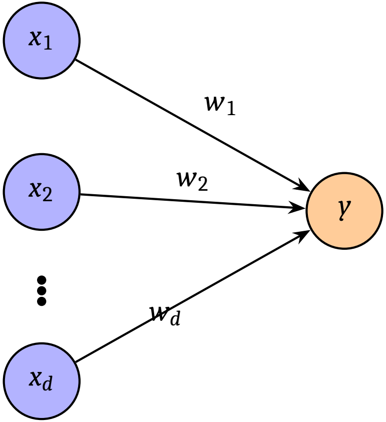


Linear Layer



$$y = \sum_{i=1}^{n} w_i x_i$$

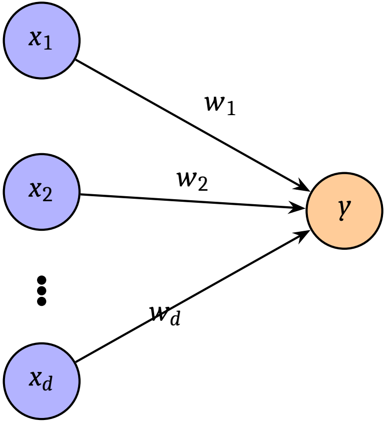
Linear Layer



$$y = \sum_{i=1}^{n} w_i x_i$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}$$

Linear Layer

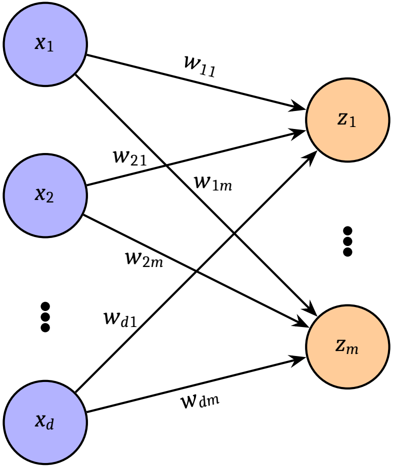


$$y = \sum_{i=1}^{n} w_i x_i$$

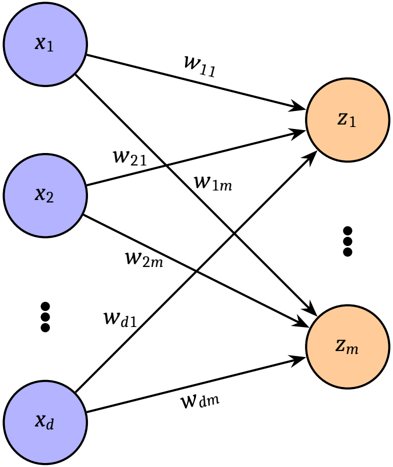
$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}$$

$$\Rightarrow y = \mathbf{x}\mathbf{w}$$

Linear Layer: Case of Hidden Layer

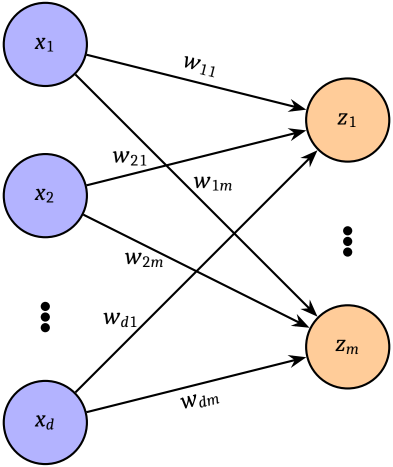


Linear Layer: Case of Hidden Layer



$$z_m = \sum_{i=1}^{n} x_i w_{im}$$

Linear Layer: Case of Hidden Layer

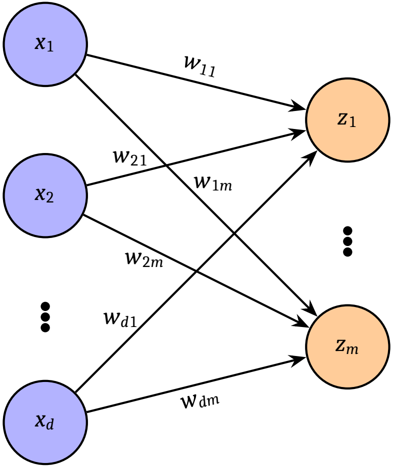


$$z_m = \sum_{i=1}^{n} x_i w_{im}$$

$$\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \vdots & z_m \end{bmatrix}_{1\times m}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{d1} & w_{m2} & \cdots & w_{dm} \end{bmatrix}_{d\times m} \qquad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}_{1\times d}$$

Linear Layer: Case of Hidden Layer



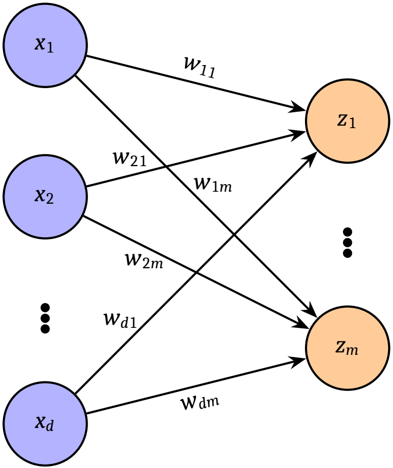
$$z_m = \sum_{i=1}^{n} x_i w_{im}$$

$$\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \vdots & z_m \end{bmatrix}_{1\times m}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{d1} & w_{m2} & \cdots & w_{dm} \end{bmatrix}_{d\times m} \qquad \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix}_{1\times d}$$

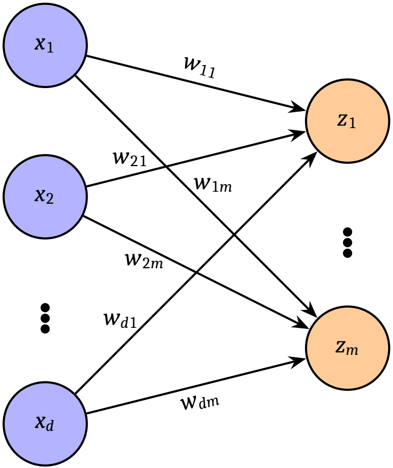
$$\Rightarrow \mathbf{z} = \mathbf{x}W$$

Linear Layer: Case of Hidden Layer



That was the case of one sample, if we consider n sample, then?

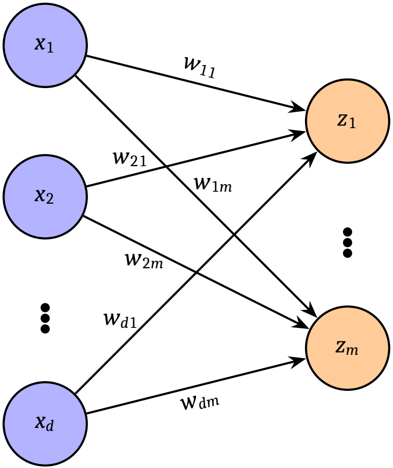
Linear Layer: Case of Hidden Layer



That was the case of one sample, if we consider n sample, then?

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}$$

Linear Layer: Case of Hidden Layer



That was the case of one sample, if we consider n sample, then?

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}$$

$$\Rightarrow Z = XW$$

Nonlinear Operation

$$A = \sigma(Z)$$

which will use elementwise operation.

Training a Neural Network

How to Train your Neural Network

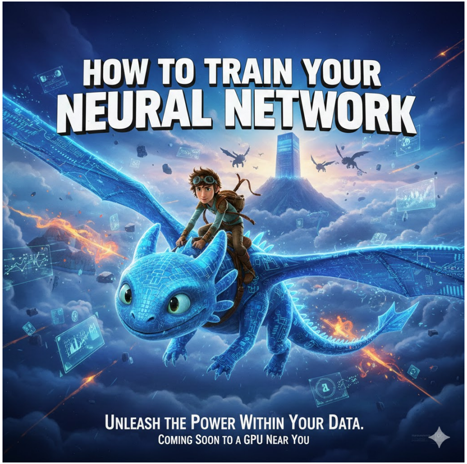

HOW TO TRAIN YOUR
NEURAL NETWORK
UNLEASH THE POWER WITHIN YOUR DATA.
COMING SOON TO A GPU NEAR YOU

Image generation from Google Gemini.

Training a Neural Network

Some criteria to consider:

1. Data Preprocessing: Normalization/Standardization
2. Batch Size
3. Batch Normalization
4. Dropout
5. Early-Stopping
6. Learning-Rate Scheduling
7. Optimizer Selection
8. Weight Initialization
9. Activation Functions
10. Network Depth & Width
11. Skip/Residual Connections
12. Regularization
13. Loss Functions
14. Gradient Clipping
15. Warm-Up Scheduling

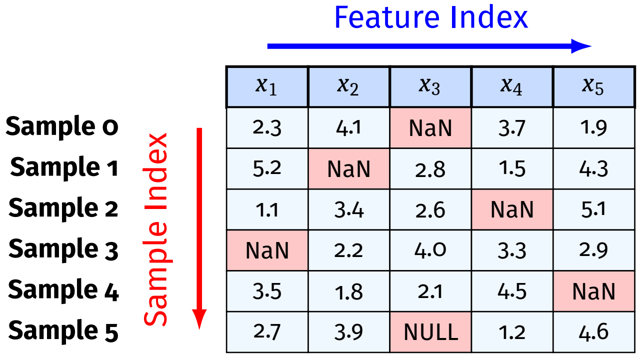
Training a Neural Network

Data Preprocessing

Imputing Missing Values

Definition

The process of filling miss values or correct known errors is called **imputation**.



Feature Index

Sample Index

	x_1	x_2	x_3	x_4	x_5
Sample 0	2.3	4.1	NaN	3.7	1.9
Sample 1	5.2	NaN	2.8	1.5	4.3
Sample 2	1.1	3.4	2.6	NaN	5.1
Sample 3	NaN	2.2	4.0	3.3	2.9
Sample 4	3.5	1.8	2.1	4.5	NaN
Sample 5	2.7	3.9	NULL	1.2	4.6

Legend:
🟦 Feature header
⬜ Valid data
🟥 Missing (NaN)

Figure: Data Table Structure: Features as Columns, Samples as Rows. Red cells indicate missing values (NaN) that require imputation before model training.

Common Types of Missing Values

1. Missing Completely at Random (MCAR), i.e. the probability of being missing is not related to the data and is only dependent on some parameter ϕ.
2. Missing at Random (MAR), i.e., the missingness probability is only related to the observed variables in the data.
3. Missing not at random (MNAR), neither of the above, and the missingness probability is related to missing values of the incomplete variable, which are unknown to us, even after conditioning on observed information

Common Imputation Methods

1. Get rid of those samples where missing # features exceed certain number (or percentage)
2. Get rid of feaures where # missing values exceed certai numbers (or percentage)
3. Sample Mean (only for continuous data)
4. Sample Median (only for continuous data)
5. k-Nearest Neighbors (kNN) imputation
6. Expectation-Maximization (EM) imputation
7. Bayesian Approach

> Mean and median approach or anything similar would fail to quantify uncertainty in the estimated missing value.

Bayesian Estimation for Missing Data

1. Model your data first (probabilistic modeling): e.g. normal distribution.
2. Compute prior distribution over uknown parameters.
3. Calculate posterior: update the prior using observed data as (Bayes' Theorem in the play). This gives a posterior distribution over model parameters and missing values.
4. Draw samples using Markov Chain Monte Carlo simulation that will be plausible missing values.

Under a Bayesian framework, missing observations can be thought of as any other parameter in the model whose distribution needs to be determined.

Bayesian Framework

Posterior Distribution

$$p(\boldsymbol{\theta} \mid X) = \frac{p(\mathbf{X} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int_{-\infty}^{\infty} p(\mathbf{X} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}} \propto p(\mathbf{X} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$$

$p(\mathbf{X} \mid \boldsymbol{\theta})$ is likelihood, $p(\boldsymbol{\theta})$ is prior.

$$p(\mathbf{X}_{mis} \mid \mathbf{X}_{obs}) = \int p(\mathbf{X}_{mis} \mid \mathbf{X}_{obs}, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathbf{X}_{obs})\, d\boldsymbol{\theta}.$$

Calculating posterior is hard, so we use Monte Carlo methods (Markov Chain Monte Carlo (MCMC)).
An example code for MCMC imputation using PyMC is available at
https://www.pymc.io/projects/examples/en/latest/howto/Missing_Data_Imputation.html

Normalization/Standardization

Definition

In simpler terms: adjusting values measured on different scales to a common scale is **Normalization**.

Otherwise, aligning entire probability distribution of adjusted values is **Normalization**.

Standard Normalization

$$\hat{x} = \frac{x - \mu}{\sigma}$$

Minmax Normalization

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Training a Neural Network

Impact of Batch Size in the Training Process

Batch Size

Definition

The number of samples passed through the neural network simultaneously is called **batch size**.

Determining a suitable batch size is an important consideration to the training process, as it may influence the learning rate of the model.

$$\mathbf{w} = \mathbf{w} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}_i(\mathbf{w})$$

m is the batch size.

How does the batch size affect training?

Stochastic Gradient Descent and Batch Size

❓ What’s problem if we use entire training set for gradient descent?

Stochastic Gradient Descent and Batch Size

❓ What's problem if we use entire training set for gradient descent?

Stochastic gradient descent provides approximation of true gradients using one sample at a time after shuffling the training sample and pick one training sample to update the weights. It optionally adds a small noise to the gradient calculation.

Mini Batch

Single sample gradient calculation can be very slow, so we can choose more than one sample at a time (but not all of them). This is called **mini batch**.

Suggested Reading

1. Masters, Dominic, and Carlo Luschi. **”Revisiting small batch training for deep neural networks.”** arXiv preprint arXiv:1804.07612 (2018).
2. You, Yang, Yuhui Wang, Huan Zhang, Zhao Zhang, James Demmel, and Cho-Jui Hsieh. **”The limit of the batch size.”** arXiv preprint arXiv:2006.08517 (2020).
3. Smith, Samuel L., Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. **”Don’t decay the learning rate, increase the batch size.”** arXiv preprint arXiv:1711.00489 (2017).

Batch Normalization

Normalization applied per batch:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$\mu_{\mathcal{B}}$ batchwise mean, $\sigma_{\mathcal{B}}^2$ is batchwise variance, and ϵ is to avoid numerical instability.

Batch Normalization

But why Batch Normalization?

Batch Normalization

But why Batch Normalization?

1. Batch Normalization smoothened out the loss landscape.
2. Gradients become more reliable and predictive.

Reading:

1. Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. **”How does batch normalization help optimization?.”** Advances in neural information processing systems 31 (2018).
2. Section 3.3 of Huang, Lei. Normalization Techniques in Deep Learning. Cham: Springer, 2022.

Training a Neural Network

Preventing Overfitting and Regularization

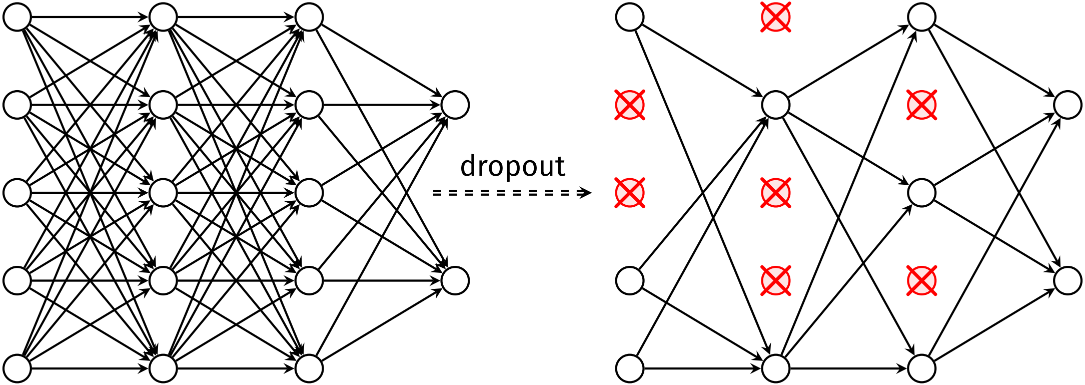
Dropout

Dropout: A new way to regularize a network

- The central idea behind *dropout* is to randomly dropout nodes in the neural network with a probability p. After the nodes are "dropped out" the remain network is a subnetwork of the original.
- Dropout has been shown to work quite well at preventing overfitting and allowing the network to find a good local minimum.
- **Recommendation**: Use dropout!

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Dropout in a Neural Network



Dropout Model

During training, Dropout stochastically ”thins” the network by sampling a mask from a Bernoulli distribution.

$$\begin{aligned}
r_i^{(l)} &\sim \text{Bernoulli}(p) \tag{Mask}\\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} \otimes \mathbf{y}^{(l)} \tag{Thinned Output}\\
\mathbf{z}^{(l+1)} &= W^{(l+1)}\tilde{\mathbf{y}}^{(l)} + \mathbf{b}^{(l+1)}\\
\mathbf{y}^{(l+1)} &= \sigma(\mathbf{z}^{(l+1)})
\end{aligned}$$

- $\mathbf{r}^{(l)}$ is a vector of independent Bernoulli random variables, each of which has probability p of being 1.
- The operator $\otimes$ denotes an element-wise product.
- σ is an activation function.
- The thinned outputs as input to the next layer.
- For learning, the derivatives of the loss function are backpropagated through the thinned network.
- At test time, the weights are scaled as $W_{test}^{(l)} = pW^{(l)}$. The resulting neural network is run without dropout.

Regularization in Neural Network

L2 Regularization

$$\hat{\mathcal{L}}(W) = \mathcal{L}(W) + \frac{\alpha}{2}||W||_2^2$$

L1 Regularization

$$\hat{\mathcal{L}}(W) = \mathcal{L}(W) + \frac{\beta}{2}||W||_1$$

1. L1 regularization promotes sparsity by forcing weights to zero.
2. L2 regularization shrinks weights, but some of the components of weights large than others that are contributing more to the model.

References and Additional Reading

1. He, Yulei, Guangyu Zhang, and Chiu-Hsieh Hsu. **Multiple imputation of missing data in practice: basic theory and analysis strategies**. Chapman and Hall/CRC, 2021.
2. Huang, Lei. Normalization Techniques in Deep Learning. Cham: Springer, 2022.

The End