CEE 616: Probabilistic Machine Learning

M2 Deep Neural Networks: Neural Networks for Structured Data II

Jimi Oke

UMass Amherst

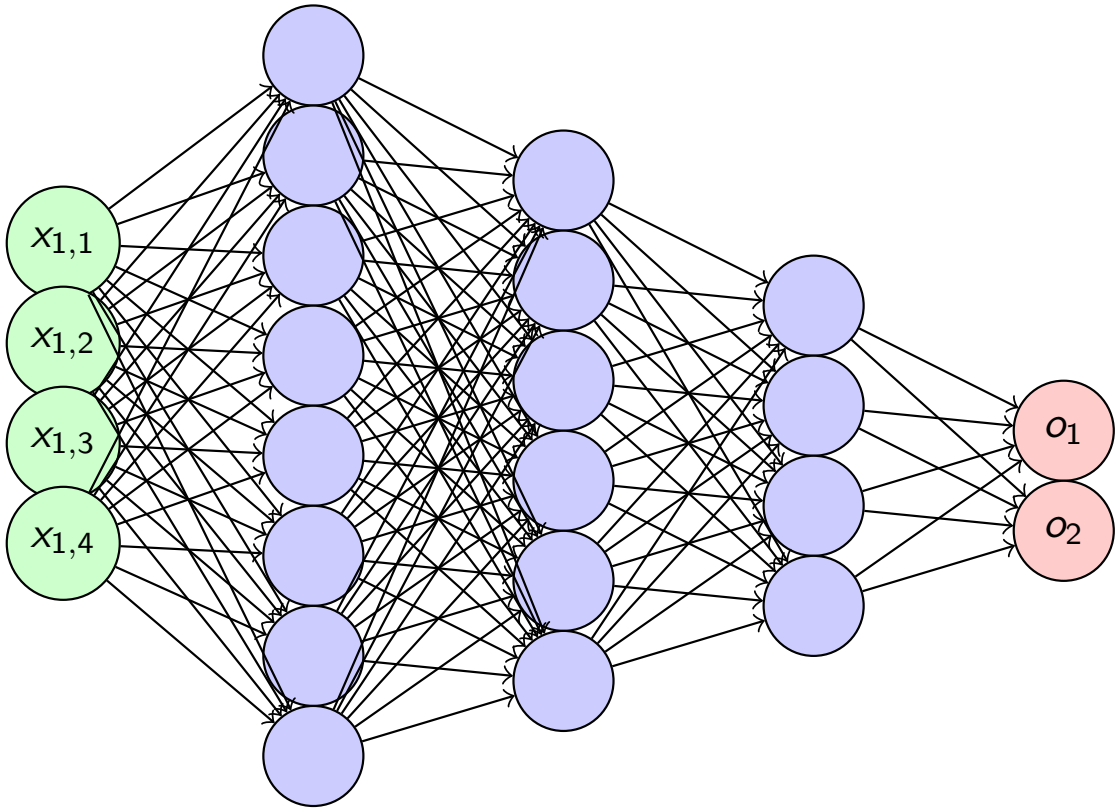
College of Engineering

Tue, Oct 21, 2025

Outline

Forward mode differentiation

Let us think of a simple feedforward with inputs $\boldsymbol{x} = \boldsymbol{x}_1 \in \mathbb{R}^{D=4}$, 3 hidden layers with $m_1 = 8$, $m_2 = 6$, and $m_3 = 4$ neurons, and outputs $\boldsymbol{o} \in \mathbb{R}^2$:



We consider each hidden unit as a function $\boldsymbol{f}_\ell(\cdot)$, where ℓ is the layer index, that maps the input $\boldsymbol{x}_\ell$ to the output of the hidden unit $\boldsymbol{x}_{\ell+1}$.

Function composition

We can then express the output of the network as a composition of functions:

$$\boldsymbol{o} = \boldsymbol{f}(\boldsymbol{x}) \tag{1}$$

where

$$\boldsymbol{f} = \boldsymbol{f}_4 \circ \boldsymbol{f}_3 \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1 \tag{2}$$

and thus

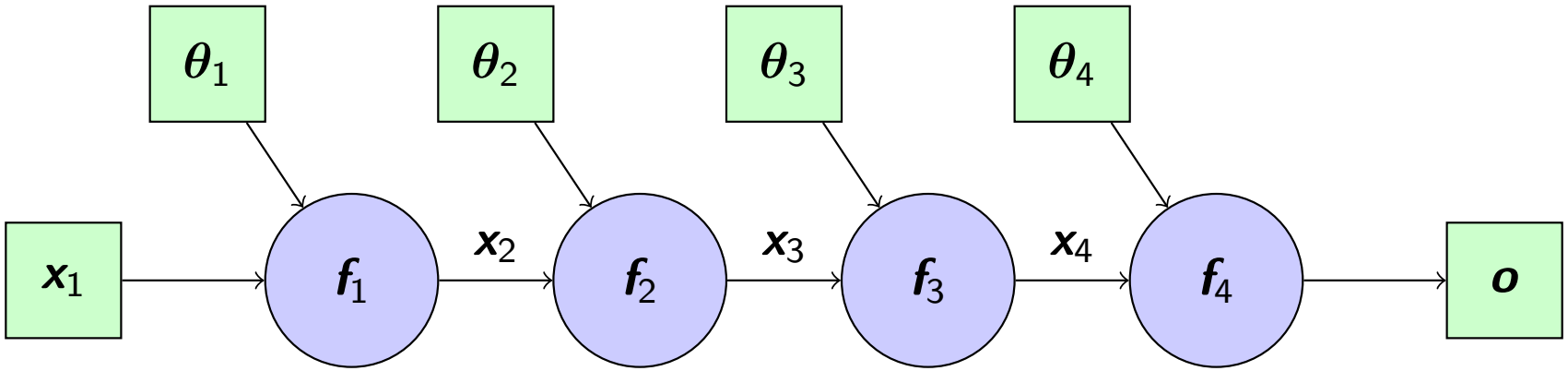
$$\boldsymbol{o} = \boldsymbol{f}_4(\boldsymbol{f}_3(\boldsymbol{f}_2(\boldsymbol{f}_1(\boldsymbol{x})))) \tag{3}$$

and

$$\begin{aligned}
\boldsymbol{x}_2 &= \boldsymbol{f}_1(\boldsymbol{x}_1), \quad \boldsymbol{f}_1 : \mathbb{R}^4 \to \mathbb{R}^8 \\
\boldsymbol{x}_3 &= \boldsymbol{f}_2(\boldsymbol{x}_2), \quad \boldsymbol{f}_2 : \mathbb{R}^8 \to \mathbb{R}^6 \\
\boldsymbol{x}_4 &= \boldsymbol{f}_3(\boldsymbol{x}_3), \quad \boldsymbol{f}_3 : \mathbb{R}^6 \to \mathbb{R}^4 \\
\boldsymbol{o} &= \boldsymbol{f}_4(\boldsymbol{x}_4), \quad \boldsymbol{f}_4 : \mathbb{R}^4 \to \mathbb{R}^2
\end{aligned}$$

Computational graph

We can further visualize the FFNN as a computational graph:



where θ_ℓ are the parameters (weights and biases) of layer ℓ.

Backpropagation

To compute the gradient, we need to find:

$$\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}_4(\boldsymbol{x}_4)}{\partial \boldsymbol{x}_4} \cdot \frac{\partial \boldsymbol{f}_3(\boldsymbol{x}_3)}{\partial \boldsymbol{x}_3} \cdot \frac{\partial \boldsymbol{f}_2(\boldsymbol{x}_2)}{\partial \boldsymbol{x}_2} \cdot \frac{\partial \boldsymbol{f}_1(\boldsymbol{x}_1)}{\partial \boldsymbol{x}_1} \tag{4}$$

We define the Jacobian matrix of each layer as:

$$\boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_\ell) = \frac{\partial \boldsymbol{f}_\ell(\boldsymbol{x}_\ell)}{\partial \boldsymbol{x}_\ell} = \begin{pmatrix} \frac{\partial f_{\ell,1}(\boldsymbol{x}_\ell)}{\partial x_{\ell,1}} & \frac{\partial f_{\ell,1}(\boldsymbol{x}_\ell)}{\partial x_{\ell,2}} & \cdots & \frac{\partial f_{\ell,1}(\boldsymbol{x}_\ell)}{\partial x_{\ell,D}} \\ \frac{\partial f_{\ell,2}(\boldsymbol{x}_\ell)}{\partial x_{\ell,1}} & \frac{\partial f_{\ell,2}(\boldsymbol{x}_\ell)}{\partial x_{\ell,2}} & \cdots & \frac{\partial f_{\ell,2}(\boldsymbol{x}_\ell)}{\partial x_{\ell,D}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{\ell,M}(\boldsymbol{x}_\ell)}{\partial x_{\ell,1}} & \frac{\partial f_{\ell,M}(\boldsymbol{x}_\ell)}{\partial x_{\ell,2}} & \cdots & \frac{\partial f_{\ell,M}(\boldsymbol{x}_\ell)}{\partial x_{\ell,D}} \end{pmatrix} \tag{5}$$

Thus, the gradient of the output with respect to the input is given by:

$$\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_4) \cdot \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_3) \cdot \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_2) \cdot \boldsymbol{J}_{\boldsymbol{f}}(\boldsymbol{x}_1) \tag{6}$$

Batch learning

The gradient descent method was described without referencing the observations.

In reality, the forward and backward passes are performed for each observation in the training set, with parameter updates obtained by **averaging** the gradients:

$$\boldsymbol{\theta}_{\ell,t+1} = \boldsymbol{\theta}_{\ell,t} - \rho \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}_n(\boldsymbol{\theta}_{\ell,t})}{\partial \boldsymbol{\theta}_\ell} \tag{7}$$

This approach is called **batch learning**.

- One sweep through all the training observations is called an *epoch*
- In batch learning, only one update results from a training epoch
- Thus, several epochs are required for convergence

Stochastic gradient descent

In standard gradient descent (batch learning, the updates are performed only after the gradient is computed for *all* training observations.

The stochastic gradient descent approach approximates the gradient in each iteration using a randomly selected observation n:

$$\boldsymbol{\theta}_{\ell,t+1} = \boldsymbol{\theta}_{\ell,t} - \rho \frac{\partial \mathcal{L}_n(\boldsymbol{\theta}_{\ell,t})}{\partial \boldsymbol{\theta}_\ell} \tag{8}$$

$$\tag{9}$$

- Each iteration over observation n results in a weight/bias update
- Thus, one sweep through the entire training set (an epoch) produces N updates

This procedure is also known as **online learning**

Mini-batch learning

To introduce stability, we can compute weight updates over a *subset* of training observations

1. Randomly sample a mini-batch $\mathcal{B}_t$ of B samples (this means that $|\mathcal{B}_t| = B$), where $B \ll N$.
2. Perform a forward and backward pass through each mini-batch to update the weights:

$$\boldsymbol{\theta}_{\ell,t+1} = \boldsymbol{\theta}_{\ell,t} - \rho \frac{1}{|\mathcal{B}_t|} \sum_{n \in \mathcal{B}_t} \frac{\partial \mathcal{L}_m(\boldsymbol{\theta}_{\ell,t})}{\partial \boldsymbol{\theta}_\ell} \tag{10}$$

   Thus, for each iteration, average the gradient over a *mini-batch* of B randomly selected observations
3. Repeat step 2 until convergence

Considerations

- Online learning is more efficient for very large datasets
- In practice, mini-batch learning is employed, as batch sizes (usually, $M = 16$ or $M = 32$) can be chosen to take advantage of parallel computing architectures
- An **adpative** learning rate ρ can guarantee convergence, e.g. $\rho_r = \frac{1}{r}$
- Input **standardization** (mean zero, SD 1) is recommended for consistent weight initialization and regularization
- Weights/biases are **initialized** to *small* values near 0 for better performance
- Cost function C is nonlinear and nonconvex; other optimization approaches (e.g. conjugate gradient) can provide faster convergence compared to stochastic gradient descent

Regularization

DNNs are prone to overfitting. To migitate this, we can regularize them by:

- Early stopping: terminating training if objective does not improve after a specified number of epochs (patience)
- Weight decay:

$$C \leftarrow C + \lambda J = C + \lambda \left(\sum w^2 + \sum b^2 \right) \tag{11}$$

  where λ is tuning parameter estimated via cross-validation
- Model compression via ℓ_1 regularization of weights (sparse DNNs)
- Dropout: randomly switching off connections from each neuraon with probability p

Regression MLP architecture

Typical hyperparameter values are:

Hyperparameter	**Value**
# input neurons	1 per input feature
# hidden layers	Usually 1 – 5
# neurons per hidden layer	Usually 10 – 100
# output neurons	1 per prediction dimension
hidden layer activation	ReLU
output activation	None (if unbounded)
loss function	MSE or MAE/Huber

Classification MLP architecture

- For classifcation, input and hidden layers are chosen in similar fashion to the regression case
- However, the number of output neurons is given by the name of classes/labels
- The output layer activation is typically the softmax function:

$$o_c = \mathcal{S}(\boldsymbol{a}_c) = \frac{e^{\boldsymbol{a}_c}}{\sum_{c'=1}^{C} e^{\boldsymbol{a}_{c'}}} \tag{12}$$

  where $\boldsymbol{a}_c$ is the unnormalized log probability of each class c
- The loss function is taken as the **categorical cross-entropy**:

$$\mathcal{L} = -\sum_{c=1}^{C} y_c \log p_c = -\sum_{c=1}^{C} y_c \log(\mathcal{S}(\boldsymbol{a}_c)) \tag{13}$$

Other types of neural networks

The standard ANN architecture (MLP) we have studied is also called the feed-forward network.

Other architectures have been shown to give better performance for various applications:

- Recurrent neural networks (RNNs): time-series forecasting
- Convolutional neural networks (CNNs): image classification
- Long short-term memory networks (LSTMs): time-series, pattern identification, etc.