Machine Learning

Hang Zheng

June 2025

Contents

I Supervised Learning **1**

1 Linear Regression **3**
1.1 LMS Algorithm 4
1.2 The normal equation 4
1.2.1 Matrix derivatives 4
1.2.2 Least Squares Revisited 4
1.2.3 Probabilistic Interpretation 5

2 Classification and Logistic Regression **7**
2.1 Logistic Regression 7
2.2 The Perceptron Learning Algorithm 8
2.3 Multi-class classification 8
2.4 Newton's method 9
2.5 Derivation of $J(\theta)$ of the classifier 10
2.6 Making predicts 10
2.6.1 Confusion matrix 11
2.6.2 ROC curve 12
2.7 Generalized Linear Model 13
2.8 Stochastic gradient descent 13

II Deep Learning **15**

3 Neural Network and applications **17**
3.1 Neural networks 17
3.1.1 Multi-layer neural network 17
3.2 Convolutions 22
3.2.1 Convolutional layers 23
3.2.2 Example: MNIST 1D 24

Part I

Supervised Learning

Chapter 1

Linear Regression

We define an approximation to y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Here, θ_i are called the **parameters** (also called **weights**). Denote $x_0 \equiv 1$.(This is the **interception** term) Then:

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$$

We also define the cost function:

$$J(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

By using $L1$ regularization ($LASSO$), we can make the cost function to be:

$$J_1(\theta) = J(\theta) + \lambda \sum_{i=1}^{n} |\omega_i|$$

Or using $L2$ regularization ($Ridge$), we have:

$$J_2(\theta) = J(\theta) + \lambda \sum_{i=1}^{n} w_i^2$$

Where λ is a **hyperparameter**.

1.1 LMS Algorithm

We consider the **gradient descent method**, with initial θ. Here α is the learning rate:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

Repeat until convergence:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}, \text{ (for every } j)$$

This is also called **batch gradient descent**.

1.2 The normal equation

In this method, we will minimize J by explicitly taking its derivatives with respect to θ_j's, and setting them to 0.

1.2.1 Matrix derivatives

we define the derivative of f with respect to A to be:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nd}} \end{bmatrix}$$

1.2.2 Least Squares Revisited

$J(\theta)$ can be rewritten as:

$$J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T (X\theta - \vec{y})$$

Hence, the gradient of $J(\theta)$ is:

$$\nabla_\theta J(\theta) = X^T X\theta - X^T \vec{y}$$

Then the normal equation is given by $\nabla_\theta J(\theta) = 0$:

$$X^T X\theta = X^T \vec{y}$$

The closed form of θ is:

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

1.2.3 Probabilistic Interpretation

Assume the target variables and the inputs are related via the equation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

Here, we know that $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$, hence:

$$\theta^{(i)} \sim \mathcal{N}(\theta^T x^{(i)}, \sigma^2)$$

Then, we formulate the likelihood function:

$$L(\theta) \ = \ \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \ \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right).$$

The principle of **maximum likelihood** says that we should choose θ so as to make the data as high probability as possible, which is to choose θ to maximize $L(\theta)$. Use the **log likelihood** $l(\theta) = \log(L(\theta))$ we have:

$$l(\theta) = n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2.$$

Hence, maximizing $l(\theta)$ is the same answer as maximizing

$$\frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2$$

Chapter 2

Classification and Logistic Regression

We focus on the **binary classification** problem in which y can only take 0 and 1. Usually, 0 is called the **negative** class, and 1 is called the **positive** class. Given $x^{(i)}$, $y^{(i)}$ is also called the **label** for the training example.

2.1 Logistic Regression

We approach the problem by ignoring the fact that y is discrete-valued. To fix this, change the form of the hypotheses:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is called the **logistic function** or **sigmoid function**. Let us assume

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Hence,

$$L(\theta) = \prod_{i=1}^{n} (h_\theta)^{y^{(i)}} (1 - h_\theta)^{1 - y^{(i)}}$$

Use the log transformation and the fact:

$$g(z)' = g(z)(1 - g(z))$$

This gives the **stochastic gradient ascent** rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

2.2 The Perceptron Learning Algorithm

Consider modifying the logistic regression method to 'force' it to output 0 or 1 exactly. It is natural to change the definition of $g(z)$ to be the *threshold* function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

If we let $h_\theta(x) = g(\theta^T x)$ and update the parameter by

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Then this is called **perceptron learning algorithm**.

2.3 Multi-class classification

Consider the response variable $y \in \{1, 2, 3, \cdots, k\}$ We use **softmax function** to turn $(\theta_1^T x, \cdots, \theta_k^T x)$ into a probability vector with non-negative entries that sum up to 1. Define the softmax function as:

$$\text{softmax}(t_1, \ldots, t_k) = \begin{bmatrix} \frac{\exp(t_1)}{\sum_{j=1}^k \exp(t_j)} \\ \vdots \\ \vdots \\ \frac{\exp(t_k)}{\sum_{j=1}^k \exp(t_j)} \end{bmatrix}.$$

Then we denote

$$P(y = i|x; \theta) = \frac{\exp\left(\theta_i^T x\right)}{\sum_{j=1}^k \exp\left(\theta_j x\right)} = \phi_i$$

By taking log-likelihood of a single sample (x, y) and find its derivative:

$$\frac{\partial \ell(\theta)}{\partial \theta_i} = \sum_{j=1}^{n} (\phi_i^{(j)} - 1\{y^{(j)} = i\} \cdot x^{(j)},$$

where

$$\phi_i^{(j)} = \frac{\exp\left(\theta_i^T x^{(j)}\right)}{\sum_{s=1}^{k} \exp\left(\theta_s x^{(j)}\right)}$$

With the gradient above, one can implement gradient descent method to minimize the loss function $\ell(\theta)$.

2.4 Newton's method

Newton's method performs the following update:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

or in higher space:

$$\theta := \theta - H^{-1} \nabla_\theta \ell(\theta).$$

Where

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}.$$

Newton's method typically enjoys faster convergence than (batch) gradient descent, and requires many fewer iterations to get very close to the minimum. One iteration of Newton's can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an d-by-d Hessian; but so long as d is not too large, it is usually much faster overall. When Newton's method is applied to maximize the logistic regression log likelihood function $\ell(\theta)$, the resulting method is also called **Fisher scoring**.

2.5 Derivation of $J(\theta)$ of the classifier

Idea: find parameters that would maximize the probability of observed data.

$$J(\theta) = \arg\max_{\theta} \log P(\text{data}) = \arg\min_{\theta}(-\frac{1}{n}\log P(\text{data}))$$

Therefore, after simplification, we have:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{y^{(i)}=1\}\log\sigma(\theta^T x^{(i)}+\theta_0)+\mathbf{1}\{y^{(i)}=-1\}\log(1-\sigma(\theta^T x^{(i)}+\theta_0))$$

If we take $y = \{0, 1\}$, then

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)}\log\sigma(\theta^T x^{(i)}+\theta_0) + (1-y^{(i)})\log(1-\sigma(\theta^T x^{(i)}+\theta_0))$$

It is the **Average negative log likelihood**. The gradient is:

$$\nabla_\theta J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - g^{(i)})x^{(i)}$$

where

$$g^{(i)} = \frac{1}{1+e^{-(\theta^T x^{(i)}+\theta_0)}}$$

2.6 Making predicts

The setting is:

$$P(Y = +1 \mid X; \omega) = \text{Sigmoid}(\omega^T h(x))$$

Where Sigmoid is never +1 or 0 exactly. We make a threshold:

- +1 if Sigmoid($\omega^T x$)$> \gamma$.
- -1 (or 0). Otherwise.

Where we call γ the threshold.

2.6.1 Confusion matrix

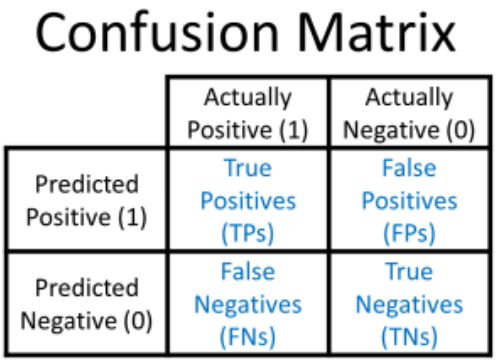

Figure 2.1: confusion matrix

We consider the **False Positive Rate**, **True Positive Rate** and the **Misclassification Rate**.

2.6.2 ROC curve

A ROC (Receiver Operating Characteristic) curve is a graphical plot that illustrates the performance of a binary (or multi-class) classification model at various threshold settings. When the threshold approaches 1, it means

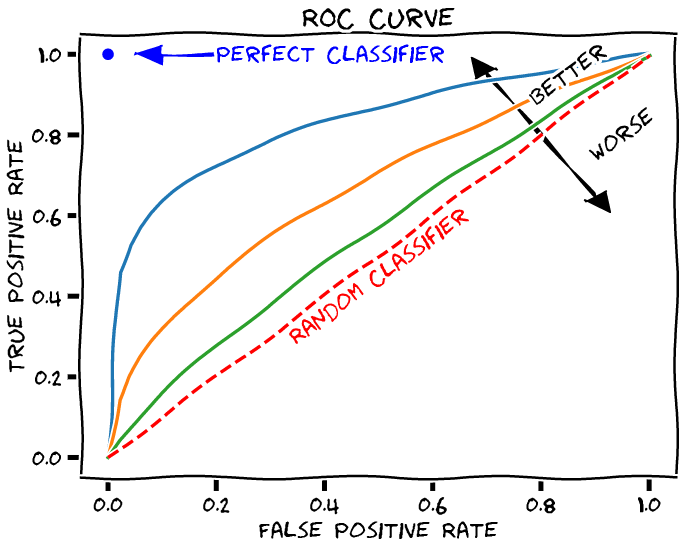

Figure 2.2: ROC curve

that the FPR and TPR will both become 1, and when it approaches 0, they both become 0. **We want the TPR to be 1 and the FPR to be 0**. Hence, the curve should be on upper left.

2.7 Generalized Linear Model

We denote the exponential family to be:

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$$

And we have:

$$\sum_y P(y;\eta) = 1$$

Here η is the **canonical parameter** of the distribution. $T(y)$ is the **sufficient statistic** and $a(\eta)$ is the **log partition function**.

2.8 Stochastic gradient descent

Algorithm 1 Stochastic Gradient Descent (SGD) Method

1: Initialize: w_0
2: **Iterate:**
3: **for** $t = 0, 1, 2, \ldots$ **do**
4: Choose a step size (i.e., learning rate) $\eta_t > 0$.
5: Generate a random variable ξ_t.
6: Compute a stochastic gradient $\nabla f(w_t; \xi_t)$.
7: Update the new iterate $w_{t+1} = w_t - \eta_t \nabla f(w_t; \xi_t)$.
8: **end for**

Characteristics:

- Randomly shuffle data, reduce the training time.
- Stochastic gradient achieves higher likelihood sooner, but it is noisier.(with more waves in the graph)

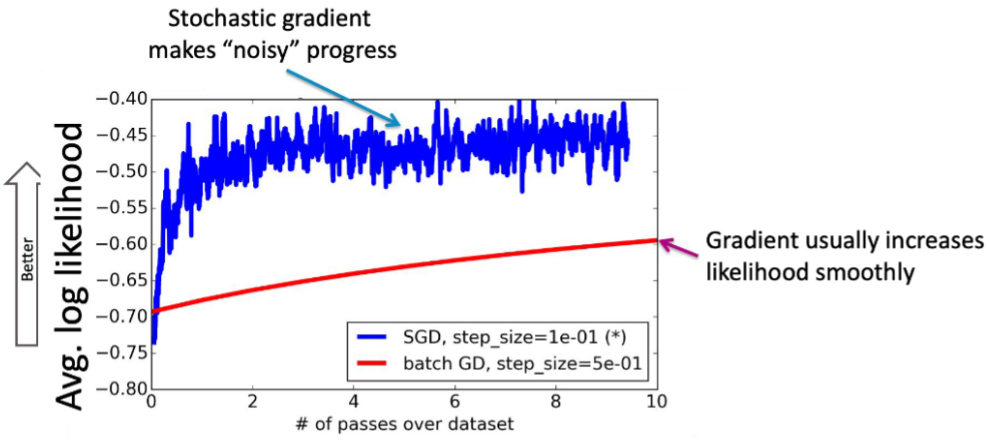

Figure 2.3: SGD compared to BGD

Part II

Deep Learning

Chapter 3

Neural Network and applications

3.1 Neural networks

We will predict as

$$\omega^T h(x) \quad \text{where} \quad h(x) = \sigma(Ax + b)$$

We will train as:

$$\min \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, w^T \sigma(Ax^{(i)} + b)).$$

3.1.1 Multi-layer neural network

The neural networks can be described as below: It is saying that given the input, the network will recognize what matters. Then the optimization takes the form:

$$\min_{\mathbf{w}, \mathbf{A}^{[1]}, \ldots, \mathbf{A}^{[L-1]}, \mathbf{b}_1, \ldots, \mathbf{b}_{L-1}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, \mathbf{w}^\top \sigma_1\left(\cdots \sigma_{L-1}\left(\mathbf{A}^{[L-1]} \mathbf{x}^{(i)} + \mathbf{b}^{[L-1]}\right) \cdots\right)\right)$$

The objective function is a form of **multi-layer neural networks**. A rich hypothesis class of parameterized functions with multiple layers of **hidden units** that can learn to represent complex features of the input.

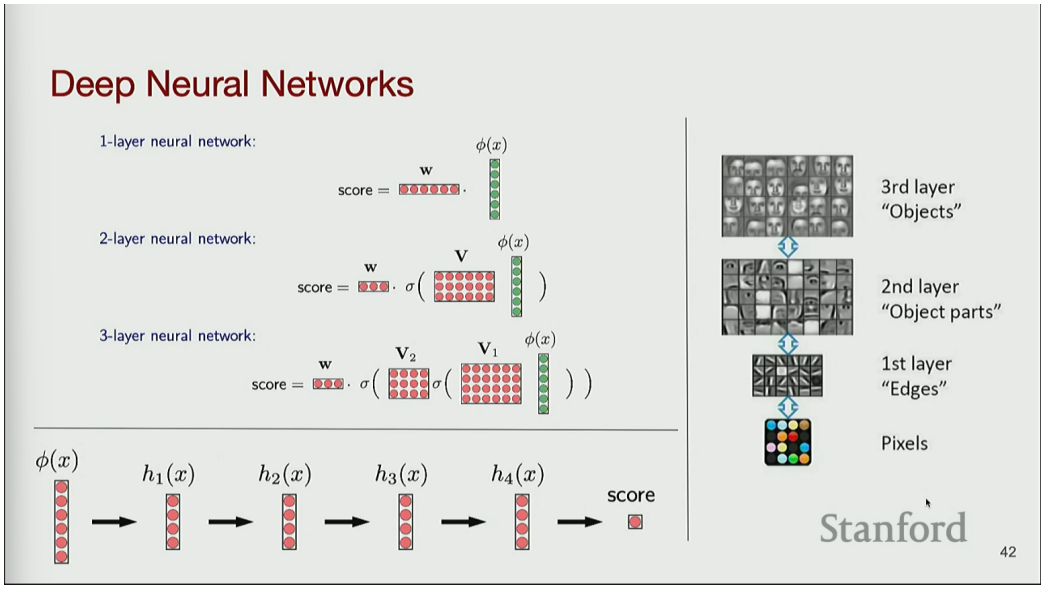

Figure 3.1: Deep neural network

Classical representation as graphs

The parameters are shown in the graph. And the loss function is given as:

$$J(w, w_0) = \sum_i \ell(NN(x^{(i)}; w, w_0), y^{(i)})$$

Where

$$\ell(a^{(i)}, y^{(i)}) = y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})$$

NN is **Neural Network**

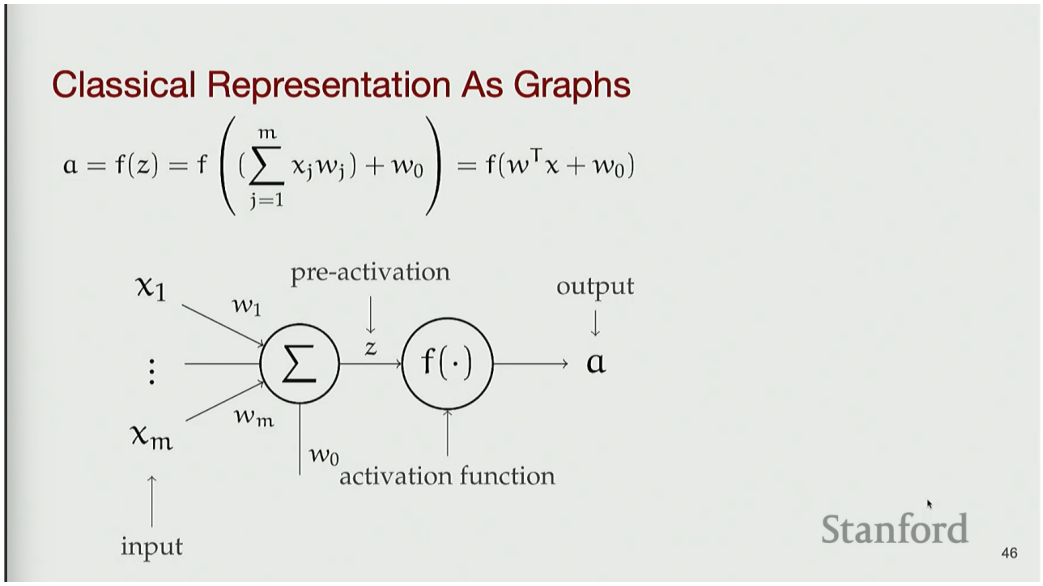

Figure 3.2: Graph representations

We shall combine the activated characters into a scalar.

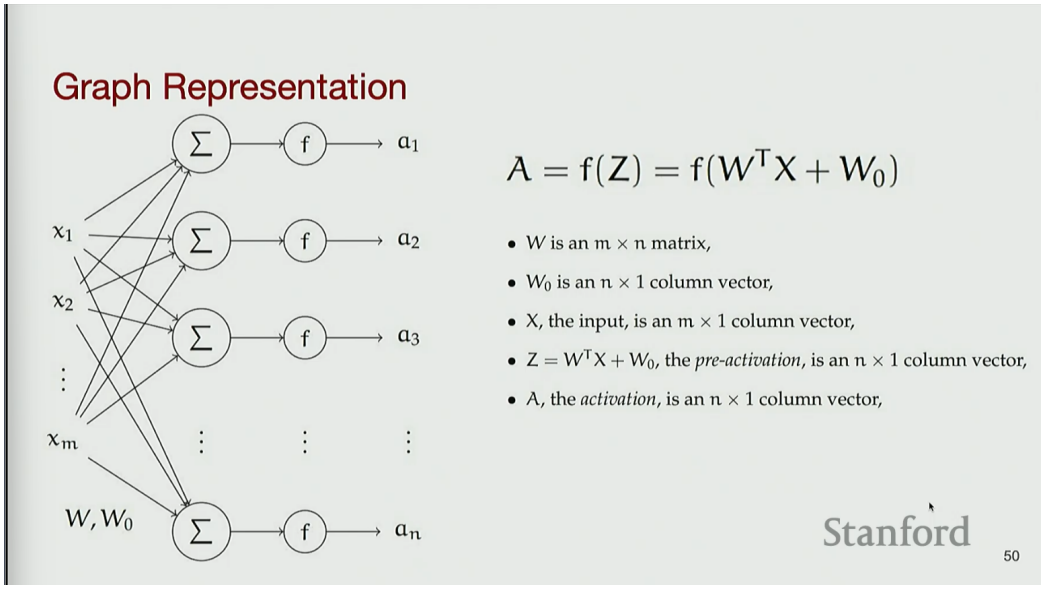

Figure 3.3: Concrete graph

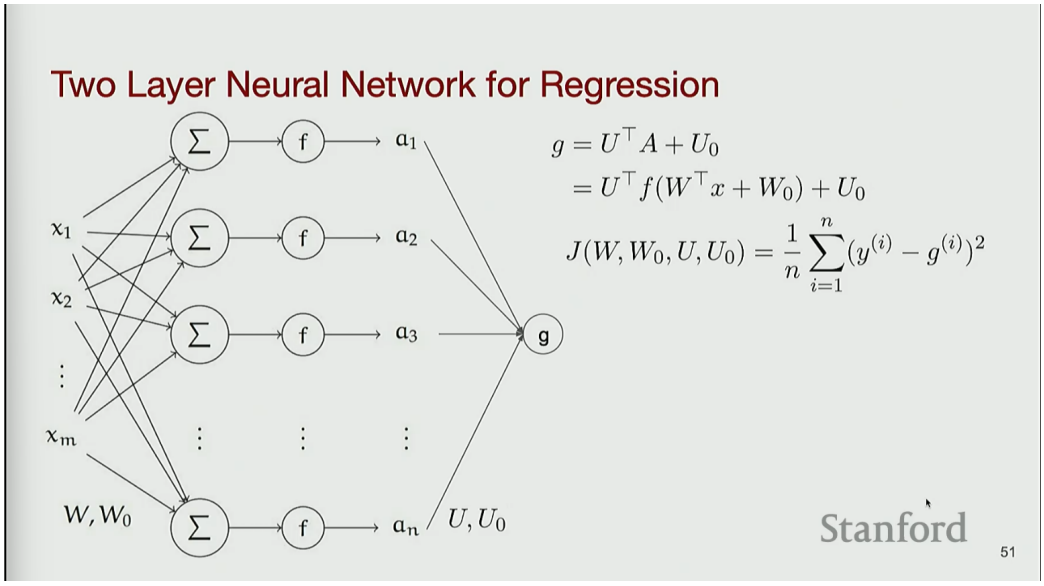

Figure 3.4: Regression sample

We just generate a g function, where:

$$g = U^T A + U_0$$

When it comes to classification, we derive:

$$g = \sigma(U^T f(W^T x + W_0) + U_0)$$

And the loss function is

$$L(W, W_0, U, U_0) = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} \log g^{(i)} + (1 - y^{(i)}) \log(1 - g^{(i)})$$

Then the general picture of multi-layer neural network is given by

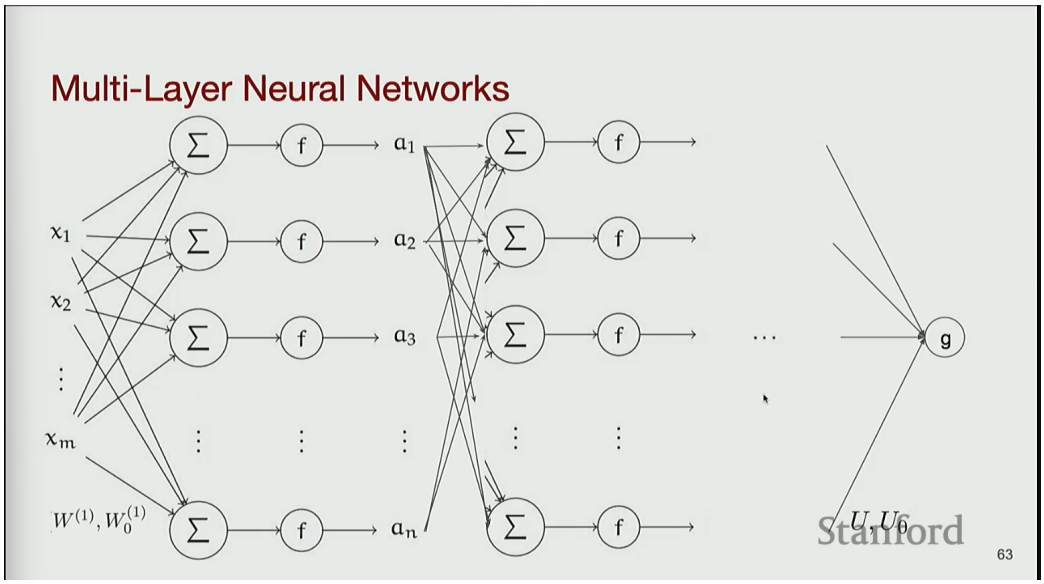

Figure 3.5: Multi-layer neural network

3.2 Convolutions

The main problem is:

- Shared weights
- Help with spatial locality and translation invariance.

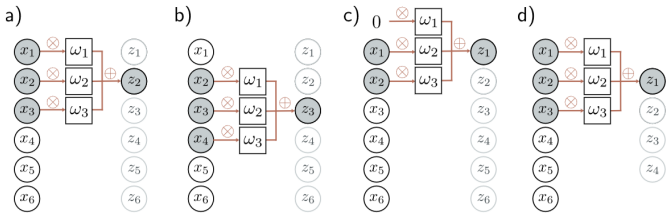

Figure 3.6: Convolutional Network

Each output z_i is a weighted sum of the nearest three inputs x_{i-1}, x_i, and x_{i+1}, where the weights are $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]$.

a) Output z_2 is computed as $z_2 = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3$.

b) Output z_3 is computed as $z_3 = \omega_1 x_2 + \omega_2 x_3 + \omega_3 x_4$.

c) At position z_1, the kernel extends beyond the first input x_1. This can be handled by zero-padding, in which we assume values outside the input are zero. The final output is treated similarly.

d) Alternatively, we could only compute outputs where the kernel fits within the input range ("valid" convolution); now, the output will be smaller than the input.

3.2.1 Convolutional layers

A convolutional layer computes its output by convolving the input, adding a bias β, and passing each result through an activation function $a[\bullet]$. With kernel size three, stride one, and dilation rate one, the i^{th} hidden unit h_i would be computed as:

$$\begin{aligned} h_i &= \mathrm{a}\left[\beta + \omega_1 x_{i-1} + \omega_2 x_i + \omega_3 x_{i+1}\right] \\ &= \mathrm{a}\left[\beta + \sum_{j=1}^{3} \omega_j x_{i+j-2}\right] \end{aligned}$$

This is a special case of a fully connected layer that computes the i_{th} hidden unit as:

$$h_i \quad = \quad \mathrm{a}\left[\beta_i + \sum_{j=1}^{D} \omega_{ij} x_j\right].$$

3.2.2 Example: MNIST 1D

This network was trained for 100,000 steps using SGD without momentum, a learning rate of 0.01, and a batch size of 100 on a dataset of 4,000 examples. We compare this to Problem 10.12 a fully connected network with the same number of layers and hidden units (i.e., three hidden layers with 285, 135, and 60 hidden units, respectively). The convolutional network has 2,050 parameters, and the fully connected network has 59,065 parameters. By the logic of figure 10.4, the convolutional network is a special case of the fully connected.

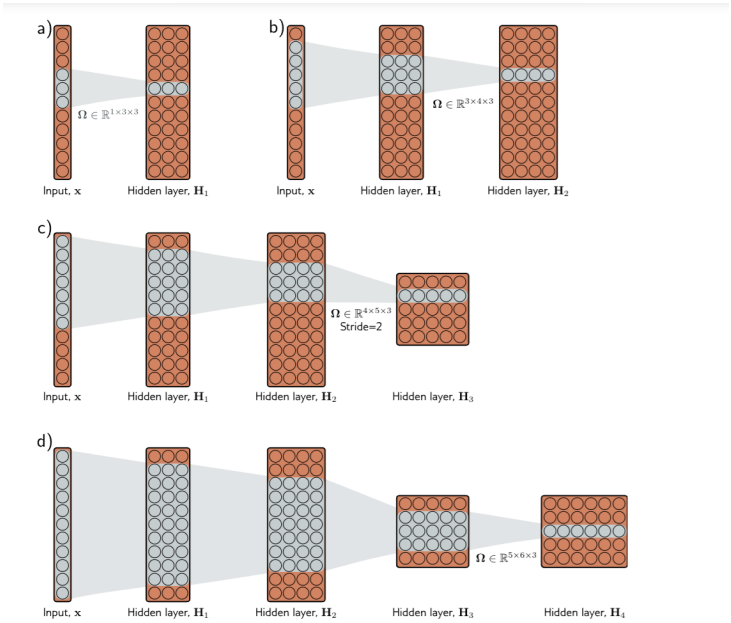

Figure 3.7: MNIST network

This is effective as shown in the graphs below: a) The convolutional

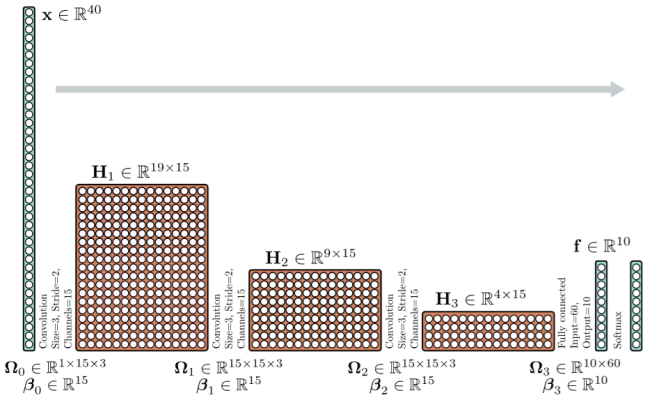

Figure 3.8: Convolutional network for classifying MNIST-1D data

network from figure 3.9 eventually fits the training data perfectly and has 17% test error. b) A fully connected network with the same number of hidden layers and the number of hidden units in each learns the training data faster but fails to generalize well with 40% test error. The latter model can reproduce the convolutional model but fails to do so. The convolutional structure restricts the possible mappings to those that process every position similarly, and this restriction improves performance.

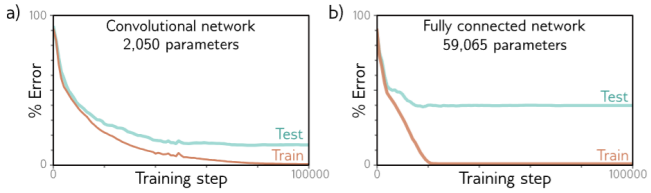

Figure 3.9: Comparision