Универзитет „Св. Кирил и Методиј“ - Скопје

Факултет за информатички науки и компјутерско инженерство

Имплементација на алгоритми од машинско учење во Matlab/Octave

- семинарска работа -

Студиска програма:
Компјутерски науки и инженерство
Предмет:
MATLAB со примена на инженерски анализи
Студент:
М-р Томче Делев
tomche.delev@finki.ukim.mk

Септември 2012

Содржина

Листа на слики

1 Вовед

Машинско учење е подобласт на вештачката интелигенција. Со помош на алгоритми таа се обидува да извлече што е можно повеќе корист од компјутерите кои претставуваат најмоќни пресметковни машини. Примери на машинско учење се рударењете на податоци од огромни бази на податоци, медицински записи, биолошки податочни множества, како и ефикасното пребарување на веб. Голема примена наоѓа и во области како обработка на природни јазици, препознавање на ракопис, компјутерска визија итн.

Алгоритмите за машинско учење се подделени во две категории:

- надгледувано учење,
- ненадгледувано учење.

Други видови на машинско учење вклучуваат засилено учење и системи за препорачување.

Според резултатот од алгоритмот за машинско учење разликуваме алгоритми за регресија и алгоритми за класификација. Во алгоритмите за регресија се предвидува континуална променлива, додека во алгоритмите за класификација се одредува класата (дискретна вредност) на примерокот кој се класифицира.

2 Линеарна регресија

2.1 Линеарна регресија со една променлива

Ова е имплементација на алгоритам за линеарна регресија кој го предвидува профитот на камион кој превезува храна. Да предпоставиме дека вие сте извршен директор на ланец ресторани и сакате да го проширите вашиот ланец во некој друг град. Во ланецот веќе имате камиони во многу различни градови и имате податоци за профитот и популацијата во тие градови.

Сакате со помош на овие податоци да добиете помош кој да биде следниот ваш град во кој ќе се проширите.

Во датотеката population_profit.txt се наоѓа податочното множество за проблемот на линеарна регресија. Во првата колона е популацијата на градот, а во втората е профитот на камионот во тој град. Негативна вредност на профитот значи загуба.

2.1.1 Исцртување на податоците

Пред да се започне со било која обработка, вообичаено е корисно да се разберат податоците со помош на визуелизација. За да ги визуелизираме овие податоци користиме

scatter plot, поради тоа што има само две својства кои се прикажуваат (профит и популација).

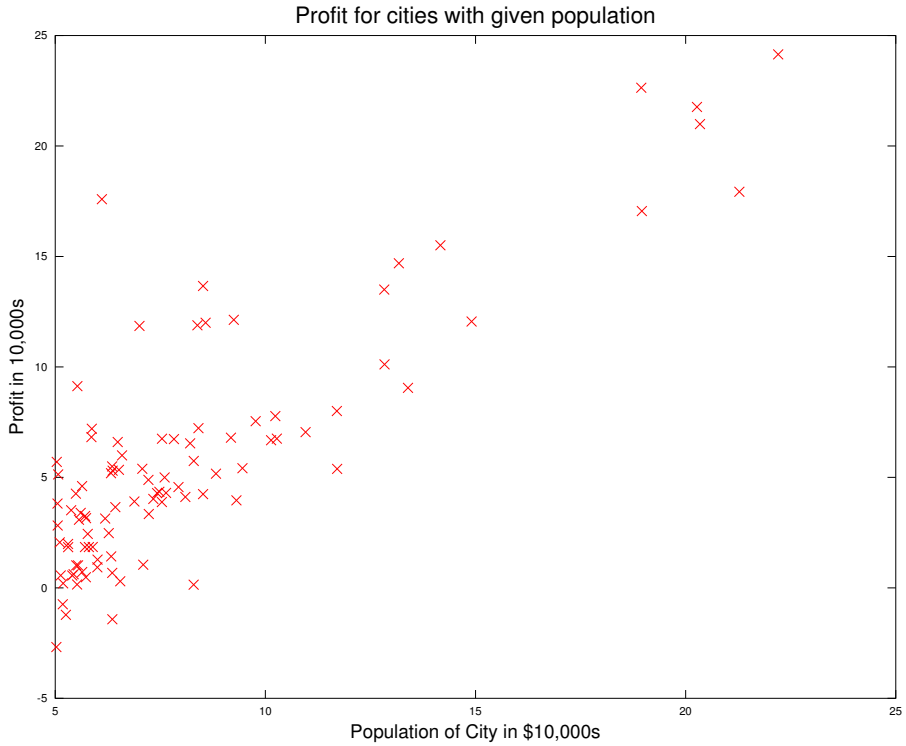
Код фрагмент 1: Вчитување на податоците во променливите X и y.

```
data = load('population_profit.txt');
X = data(:, 1);  % the population size in 10,000s
y = data(:, 2); % the profit in $10,000s
```

Код фрагмент 2: Исцртување на податоците

```
figure; % open a new figure window

plot(x, y, 'rx', 'MarkerSize', 10);
ylabel('Profit in 10,000s', 'fontsize', 20);
xlabel('Population of City in $10,000s', 'fontsize', 20);
```



Слика 1: Scatter plot на податочното множество

2.1.2 Пресметување со Gradient Descent

Следува имплементација на одредување на параметарот θ од линеарната регресија на податочното множество со помош на gradient descent.

2.1.3 Равенки за пресметување

Целта на линеарната регресија е да ја минимизира функцијата на чинење

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

каде што хипотезата $h_\theta(x)$ е дадена со линеарниот модел:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Параметрите на моделот се вредностите θ_j. Алгоритмот го имплементираме со тоа што во секоја итерација ја извршуваме следата формула:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j$$

Симултано ги менуваме вредностите на θ_j за секое j.

2.1.4 Имплементација

Код фрагмент 3: Додавање на дополнителна колона θ_0 и иницијализција на параметрите.

```
theta = zeros(2, 1); % initialize fitting parameters

% Some gradient descent settings
iterations = 1500;
alpha = 0.01;
```

2.1.5 Пресметување на функцијата на чинење $J(\theta)$

Код фрагмент 4: Имплементација на $J(\theta)$

```
% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;
prediction = theta(1,:) .* X(:,1) + theta(2,:) .* X(:,2);
J = 1 / (2 * m) * sum((prediction - y(:,1)).^2);
```

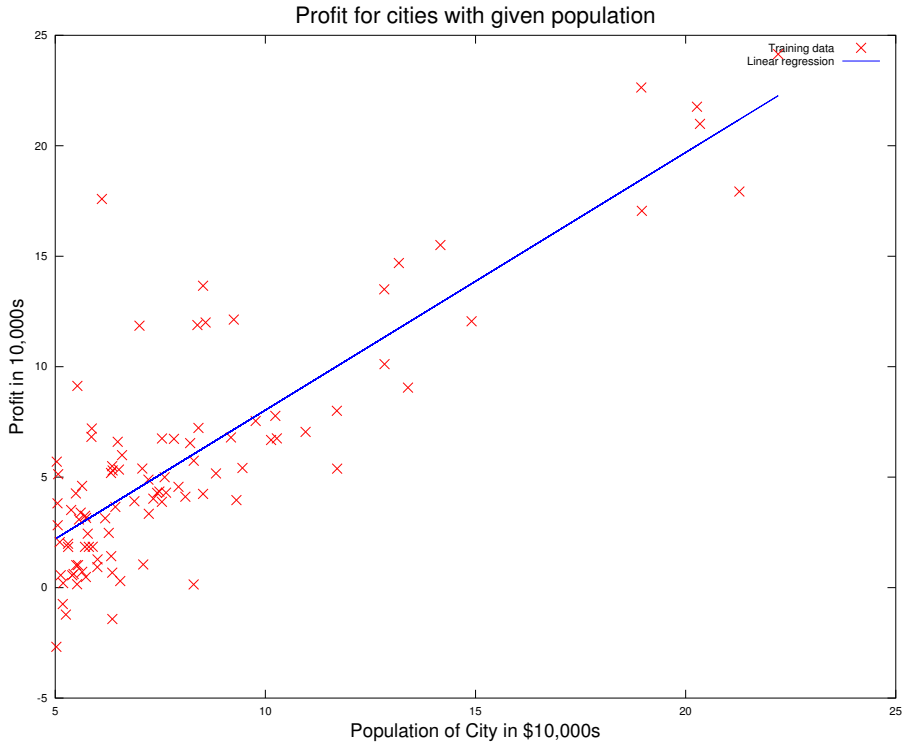
2.1.6 Gradient Descent

Код фрагмент 5: Имплементација на Gradient Descent

```
% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    Delta = ((theta' * X' - y') * X)';
    theta = theta - alpha / m * Delta;

    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);
end
```

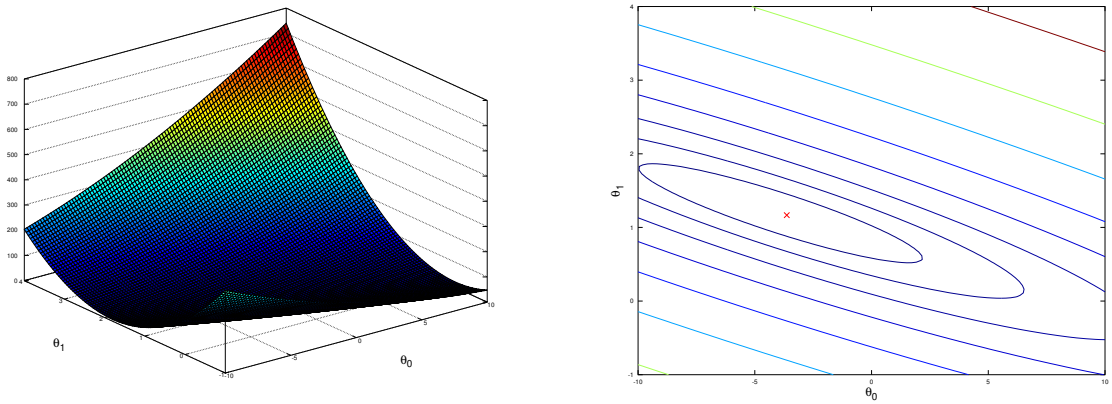


Слика 2: Линеарна регресија на тренинг множеството

2.1.7 Визуелизација на $J(\theta)$

За подобро согледување на вредностите на функцијата на чинење $J(\theta)$ ги исцртуваме нејзините вредност на 2-димензионален грид од вредностите на θ_0 и θ_1.

Целта на овие графици е да покажат како $J(\theta)$ се менува со менувањето на θ_0 и θ_1. Функцијата на чинење $J(\theta)$ е обликувана како вдлабнат сад и има глобален минимум. Ова најубаво се воочува на површинскиот график во 3Д. Минимумот е оптималната точка за θ_0 и θ_1, со што алгоритмот gradient descent во секој чекор се доближува поблиску до оваа точка.



(а) Површина (б) Контура која го прикажува минимумот

Слика 3: Функцијата на чинење $J(\theta)$

2.2 Линеарна регресија со повеќе променливи

Со помош на линеарна регресија со повеќе променливи ќе ја предвидуваме цената на куќите. Да претпоставиме дека продаваме куќа и сакаме да дознаеме која е нејзината цена на пазарот. Еден начин да го дознаеме ова е да собереме информации за цените на последно продадените куќи ид да изградиме модел на цените на куќите.

Во датотеката houses_prices.txt се содржи податочно множество со цените на куќите во Портланд, Орегон. Првата колона е големината на куќата (во квадратни стапки), втората колона е бројот на соби, а третата колона е цената на куќата.

2.2.1 Нормализација на обележјата

Помеѓу вредностите на некои обележја постојат многу големи разлики и тоа не е добро за алгоритмите за машинско учење. Затоа кога постојат вакви огромни разлики, вредностите на обележјата се нормализираат (скалираат) со што алгоритмот gradient descent многу побрзо конвергира.

Нормализацијата ја извршуваме со следните чекори:

- Ја одземаме средната вредност од секое обележје во податочното множество,
- Ги скалираме (делиме) вредностите на обележјата со нивните соодветни стандардни девијации.

Код фрагмент 6: Нормализација на обележјата

```
X_norm = X;
mu = zeros(1, size(X, 2));
sigma = zeros(1, size(X, 2));
```

```
mu = mean(X);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm);
X_norm = bsxfun(@rdivide, X_norm, sigma);
```

2.2.2 Gradient Descent

Код фрагмент 7: Имплементација на Gradient Descent за повеќе променливи

```
% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters

    △ = ((theta' * X' - y') * X)';
    theta = theta - alpha / m * △;

    % Save the cost J in every iteration
    J_history(iter) = computeCostMulti(X, y, theta);

end
```

2.2.3 Нормални равенки

Точното решение на линеарна регресија може да се определи со равенката:

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

За да се употребува оваа формула не е потребно скалирање на обележјата и ќе се добие точно решение во една пресметка: нема „циклус до конвергенција“ како во претходниот алгоритам.

3 Логистичка регресија

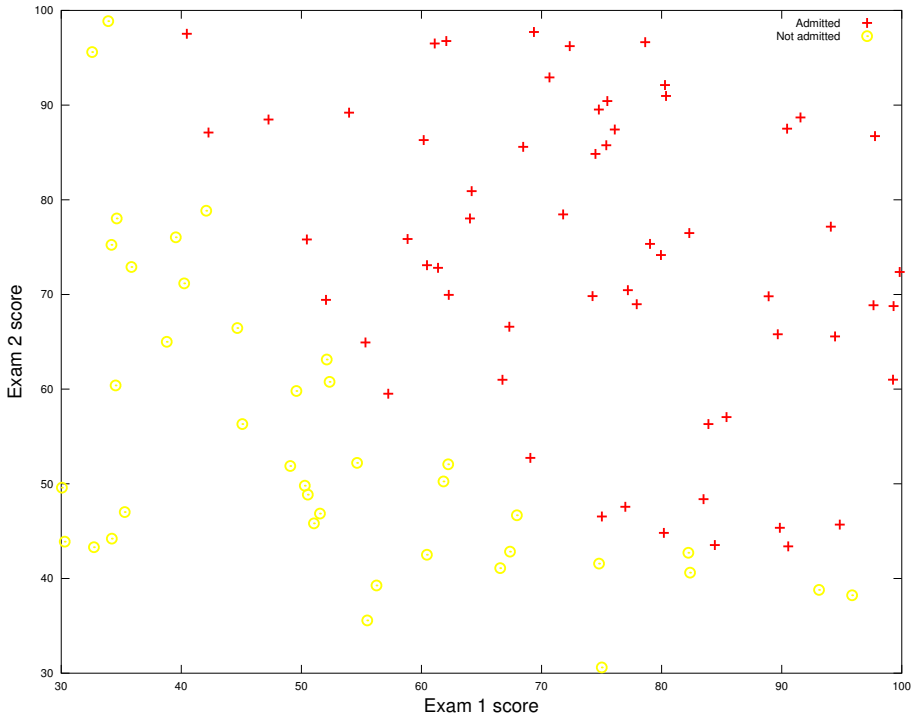
Со овој алгоритам ќе изградиме модел за логистичка регресија кој ќе предвидува дали кандидатот е примен или не на универзитет.

Вие сте раководител на некој оддел во универзитет и сакате да ги одредите шансите на кандидатот да биде примен во зависност од неговите резултати на двата приемни испити. Поседувате податоци од претходните уписи со резултатите од двата испити и одлуката дали е примен кандидатот или не.

Целта е да се изгради модел за класификација кој ќе ја предвидува веројатноста за да биде примен кандидатот во зависност од резултатите од двата испити.

3.1 Визуелизација на податоците

На слика 4 е прикажана визуелизација на податоците од податочното множество, во кое со жолти кругчиња се означени одбиените, а со црвени плус знаци примените кандидати.



Слика 4: Scatter plot на податочното множество

Код фрагмент 8: Разделување на податочното множество на позитивни и негативни примероци и негова визуелизација

```
% Find indecies of Positive and Negative examples
pos = find(y == 1);
neg = find(y == 0);
% Plot examples
plot(X(pos, 1), X(pos, 2), 'r+', ...
    'LineWidth', 3, 'MarkerSize', 10);
plot(X(neg, 1), X(neg, 2), 'yo', ...
    'LineWidth', 3, 'MarkerSize', 10);
```

3.2 Имплементација

3.2.1 Sigmoid функција

Хипотезата во логистичка регресија е дефинирана со:

$$h_\theta(x) = g(\theta^T x),$$

каде функцијата g е sigmoid функцијата. Sigmoid функцијата е дефинирана со:

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

Код фрагмент 9: Имплементација на функцијата sigmoid

```
g = zeros(size(z));
g = 1 ./ (1 + exp(-z));
```

Функцијата на чинење во логистичка регресија е:

3.2.2 Функција на чинење и градиент

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

а градиентот на чинење е вектор со иста должина како θ каде што j^{th} елемент (за $j = 0, 1, \ldots, n$) е дефиниран на следниот начин:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

3.2.3 Учење на параметрите со `fminunc`

Учењето на параметрите θ во овој алгоритам ќе го правиме со употреба со готовата функција за минимизација `fminunc`. Ова е вградена функција која наоѓа минимум на неограничена [1] функција.

За логистичка регресија ја оптимизираме функцијата на чинење $J(\theta)$ со параметар θ.

Конкретно, ќе ја користиме fminunc да ги пронајдеме најдобрите параметри θ за функцијата на чинење на логистичката регресија, за дадено податочно множество (X и y вредности). Функцијата fminunc ќе ја повикаме со следните аргументи:

- Иницијалните вредности на параметрите кои сакаме да ги оптимизираме.
- Функција, која за дадено тренинг податочно множество и дадено θ, пресметува функција на чинење на логистичка регресија и градиент.

[1] Ограничувањата во оптимизацијата често се однесуваат на ограничувања на параметрите, на пример, ограничувања на вредностите на θ (пр., $\theta <= 1$). Логистичката регресија нема такви ограничувања затоа што θ може да ја прими вредноста на секој реален број.

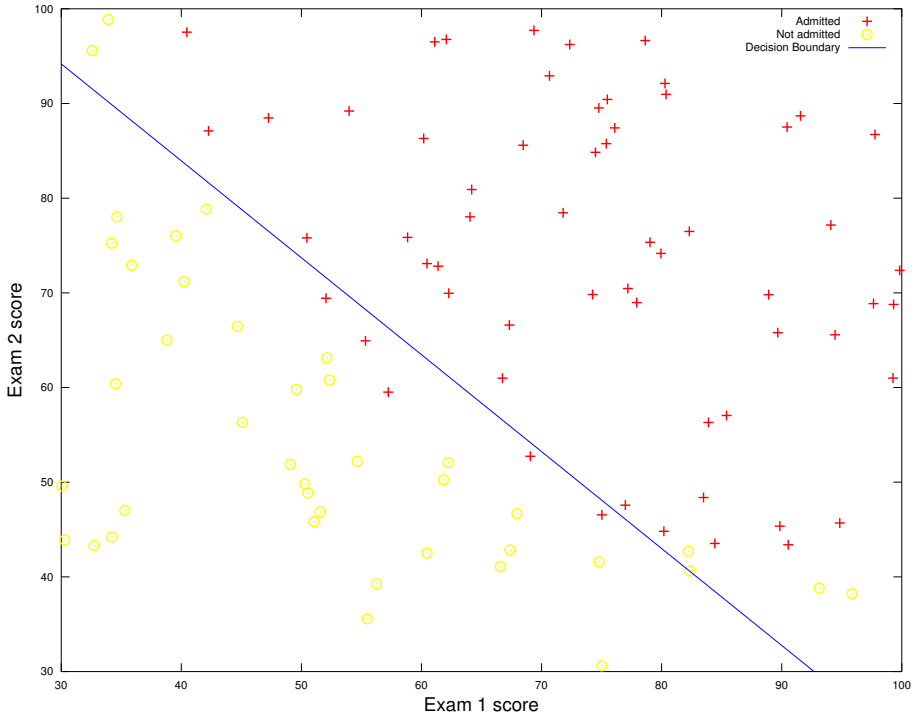
Код фрагмент 10: Повикување на функцијата за оптимизација `fminunc`

```
%% Optimizing using fminunc

%  Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

%  Run fminunc to obtain the optimal theta
%  This function will return theta and the cost
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)),...
    initial_theta, options);
```

Со овој код фрагмент, најпрво ги дефинираме опциите со кои ќе ја повикаме `fminunc`. Со поставување на `GradObj` на `on`, ѝ кажуваме на `fminunc` дека нашата функција враќа и чинење и градиент. Ова овозможува `fminunc` да го користи градиентот за минимизирање на функцијата. Со поставување на `MaxIter` на 400, ја ограничуваме `fminunc` да се извршува максимум 400 итерации.



Слика 5: Податочното множество со границата на одлука

Вредноста на θ која се добива како резултат на оптимизацијата служи за исцртување на границата на одлука прикажана на слика 5.

3.2.4 Евалуација на логистичката регресија

Откако ќе ги научиме параметрите, го користиме моделот за да предвидиме дали одреден кандидат ќе биде примен. За кандидат кој на првиот испит имал 45 поени,

а на вториот 85, добиваме веројатност за успех од 0.776.

Нашиот модел го евалуираме на целото тренинг множество.

Код фрагмент 11: Функцијата за евалуација на тренинг множеството

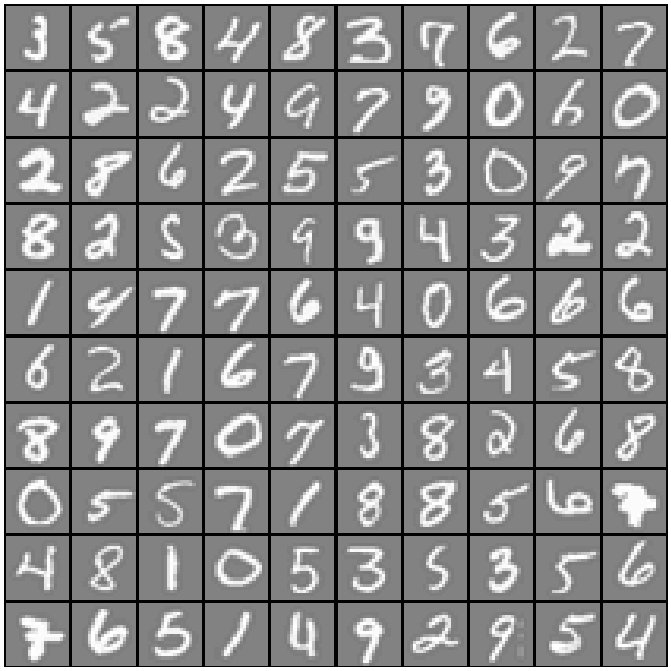
```
m = size(X, 1); % Number of training examples

p = zeros(m, 1);
p = sigmoid(X * theta) ≥ .5;
```

4 Невронски мрежи

Имплементација на алгоритмот за повратна пропагација за учење на параметрите на невронската мрежа. Овој алгоритам ќе го искористиме за препознавање на ракописно напишани цифри. Ова е корисно во автоматско читање на поштенски кодови, чекови, сметки и слично.

4.1 Визуелизација на податоците



Слика 6: Примероци од податочното множество

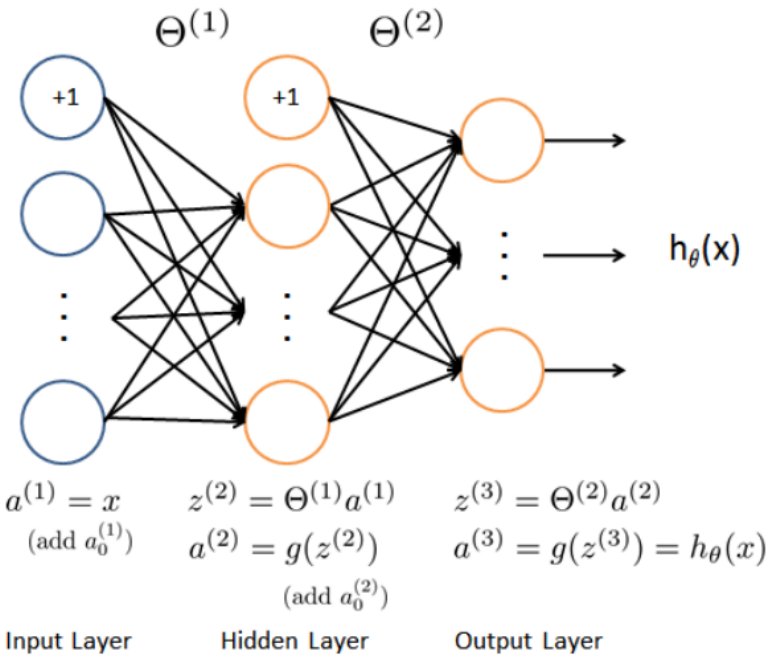
Податочното множество (дел е прикажан на слика 6) за овој алгоритам е составено од 5000 тренинг примероци, од кој секој примерок е црно-бела слика од 20x20

пиксели. Секој пиксел се репрезентира со децимален број кој го означува интензитетот. Овие 400 пиксели се сместени во еднодимензионален вектор. Секој од овие тренинг примероци претставува ред од матрицата X. Со ова се добива матрица од 5000 по 400 во која секој ред е примерок за слика од ракописно напишана цифра.

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

Вториот дел од податочното множество е 5000-димензионален вектор y кој ги содржи целните ознаки за податочното множество. За да се постигне компатибилност со индексирањето во Octave/Matlab, каде што не постои 0 индекс, цифрата 0 е пресликана во вредноста 10. Така, цифрата „0“ е означена како „10“, додека цифрите од „1“ до „9“ се означени како „1“ до „9“ во нивниот природен редослед.

4.2 Репрезентација на моделот



Слика 7: Моделот на невронска мрежа

На слика 7 е прикажан моделот на невронската мрежа. Се состои од 3 слоеви: влезен слој, скриен слој и излезен слој. На влезот на оваа невронска мрежа се дигиталните слики. Затоа што секоја слика е со големина 20 x 20, влезниот слој е составен од 400 влезни единици.

4.3 Feedforward пропагација и предикција

Алгоритмот за feedforward пропагација ја пресметува $h_\theta(x^{(i)})$ за секој примерок i и ја враќа предвидената вредност. Слично како кај стратегијата за класификација еден-против-сите, предвидувањето на невронската мрежа ќе биде ознаката со најголема вредност $(h_\theta(x))_k$.

Код фрагмент 12: Имплементација на функцијата predict

```
% Useful values
m = size(X, 1);
num_labels = size(Theta2, 1);

p = zeros(size(X, 1), 1);

X = [ones(m, 1) X];
a1 = sigmoid(X * Theta1');
a1 = [ones(size(a1, 1), 1) a1];
a2 = sigmoid(a1 * Theta2');
h = sigmoid(a2);
[temp, p] = max(h, [], 2);
```

4.4 Feedforward и cost функција

Функцијата на чинење на невронска мрежа (без регуларизација) е

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)],$$

каде што $h_\theta(x^{(i)})$ се пресметува како на слика 7 и $K = 10$ е бројот на можни ознаки. Со $(h_\theta(x^{(i)}))_k = a_k^{(3)}$ се означува активацијата (излезната вредност) на $k-th$ излезна единица. Оригиналните вредности за излезната ознака y се 1, 2, ..., 10, за тренирање на невронска мрежа, треба сите ознаки да се претворат во соодветни вектори кои ги содржат единствено вредностите 0 и 1:

$$y = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \ldots or \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$$

На пример, ако $x^{(i)}$ е слика од цифрата 5, тогаш соодветното $y^{(i)}$ (кое треба да се искористи во функцијата на чинење) треба да биде 10-димензионален вектор со $y_5 = 1$, а останатите елементи 0.

Код фрагмент 13: Имплементација на функцијата на чинење

```
% Reshape nn_params back into the parameters Theta1 and Theta2, the weight matrices
% for our 2 layer neural network
Theta1 = reshape(nn_params(1:hidden_layer_size * ...
(input_layer_size + 1)), hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * ...
(input_layer_size + 1))):end), num_labels, (hidden_layer_size + 1));

% Setup some useful variables
m = size(X, 1);
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));

X = [ones(m, 1) X];
y = eye(num_labels)(y,:);

a1 = X;
z2 = a1 * Theta1';
a2 = sigmoid(z2);

n = size(a2, 1);
a2 = [ones(n,1) a2];

z3 = a2 * Theta2';
a3 = sigmoid(z3);

regularization = (lambda/(2*m)) * (sum(sum((Theta1(:,2:end)).^2))...
 + sum(sum((Theta2(:,2:end)).^2)));
J = ((1/m) * sum(sum((-y .* log(a3))-((1-y) .* log(1-a3))))))...
 + regularization;

Delta_3 = a3 - y;
Delta_2 = (Delta_3 * Theta2(:,2:end)) .* sigmoidGradient(z2);

Delta_cap2 = Delta_3' * a2;
Delta_cap1 = Delta_2' * a1;

Theta1_grad = ((1/m) * Delta_cap1) + ((lambda/m) * (Theta1));
Theta2_grad = ((1/m) * Delta_cap2) + ((lambda/m) * (Theta2));

Theta1_grad(:,1) -= ((lambda/m) * (Theta1(:,1)));
Theta2_grad(:,1) -= ((lambda/m) * (Theta2(:,1)));
```

4.5 Backpropagation

4.5.1 Сигмоид градиент

Градиентот за сигмоид функцијата се пресметува како

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1 - g(z))$$

каде што

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

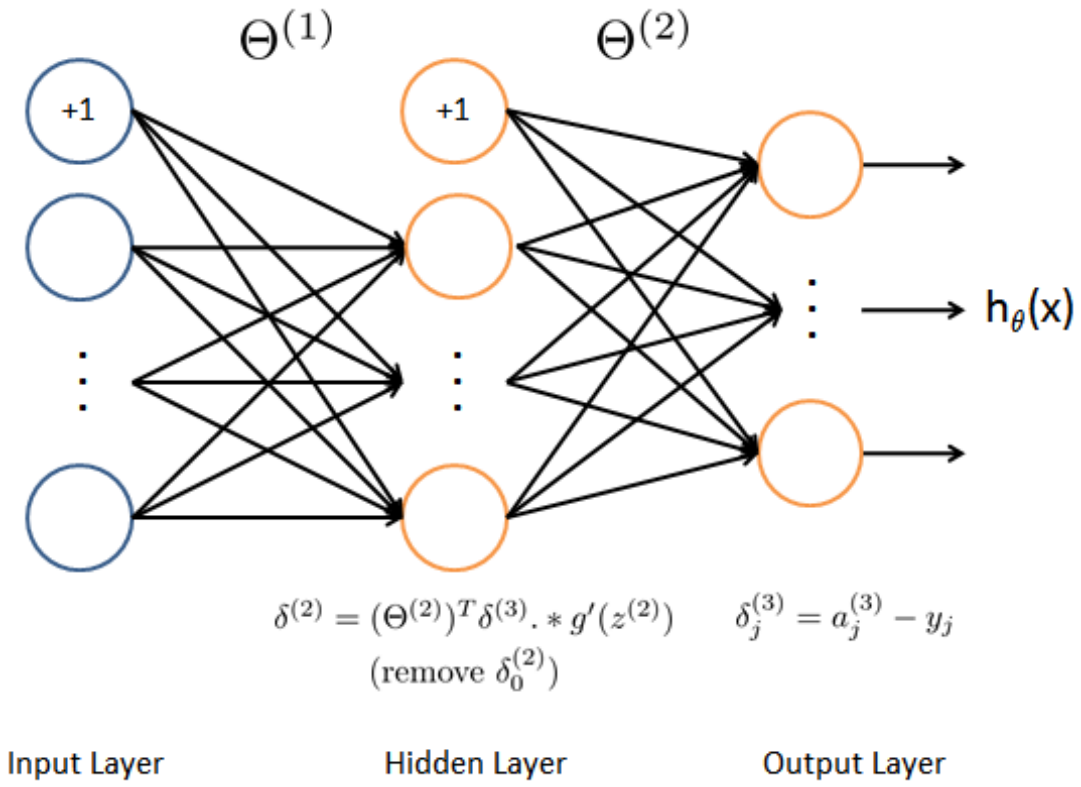
4.5.2 Случајна иницијализација

Кога се тренираат невронски мрежи, важно е да параметрите да се иницијализираат случајно за да се наруши симетријата. Една ефективна стратегија за случајна иницијализација е случајно да се изберат вредности за $\theta(l)$ рамномерно во опсегот $[-\epsilon_{init}, \epsilon_{init}]$. Овој опсег не осигурува дека вредностите на параметрите ќе бидат мали, а со тоа и учењето ќе биде поефикасно.

Код фрагмент 14: Имплементација на случајна иницијализација

```
% Randomly initialize the weights to small values
epsilon_init = 0.12;
W = rand(L_out, 1+L_in) * 2 * epsilon_init - epsilon_init;
```

4.5.3 Backpropagation



Слика 8: Освежување на вредностите во Backpropagation

За даден тренинг примерок $(x^{(t)}, y^{(t)})$, најпрво се извршува „изминување напред“ за да се добијат активациските вредности во мрежата, вклучувајќи ги и излезните вредности на хипотезата $h_\theta(x)$. Потоа, за секој јазел j во слојот l, сакаме да ја пресметаме „грешката“ $\delta_j^{(l)}$ која означува колку секој јазел во мрежата е „одговорен“ за грешките во излезот.

За излезните јазли, можеме директно да ја измериме разликата меѓу активацијата на мрежата и вистинската вредност и да ја искористиме за да ја дефинираме $\delta_j^{(3)}$ (слојот 3 е излезниот слој). а скриените единици, $\delta_j^{(l)}$ го пресметуваме како тежински просек на изразите за грешка на слојот $(l+1)$.

Детално чекорите на backpropagation алгоритмот се следните:

1. Поставете ги вредностите на влезниот слој $(a^{(1)})$ на вредностите од $t-$тиот тренинг примерок $x^{(t)}$. Следува feedforward (Слика 8), за пресметување на активациите $z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)}$ за слоевите 2 и 3. Исто така треба да се додаде и

изразот a + 1 за да активациските вектори за слоевите $a^{(1)}$ и $a^{(2)}$ исто така ја вклучуваат единицата за наклонетост. Во Matlab и Octave, ако 1 е вектор колона, додавање 1 е еквивалентно на

`a_1 = [1; a_1]`.

2. За секоја излезна единица k во слојот 3 (излезниот слој), пресметајте:

$$\delta_k^{(3)} = (a^{(k)} - y_k),$$

каде што y_k покажува дали тековниот примерок припаѓа на класата $k(y_k = 1)$ или припаѓа на друга класа $(y_k = 0)$.

3. За скриениот слој $l = 2$, пресметајте

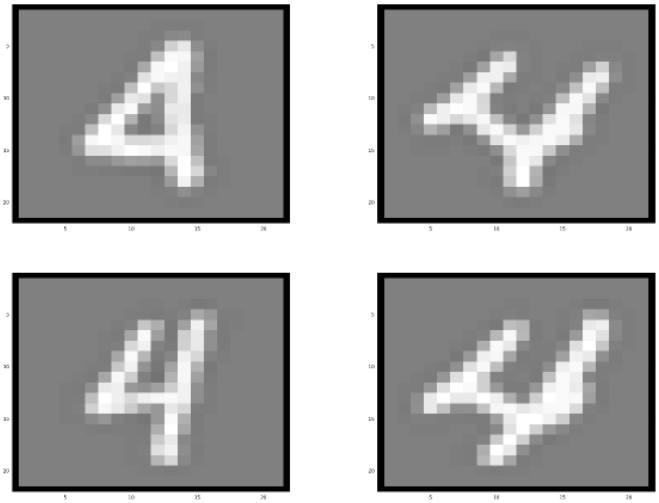
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}),$$

4. Соберете го градиентот од овој примерок со користење на следната формула:

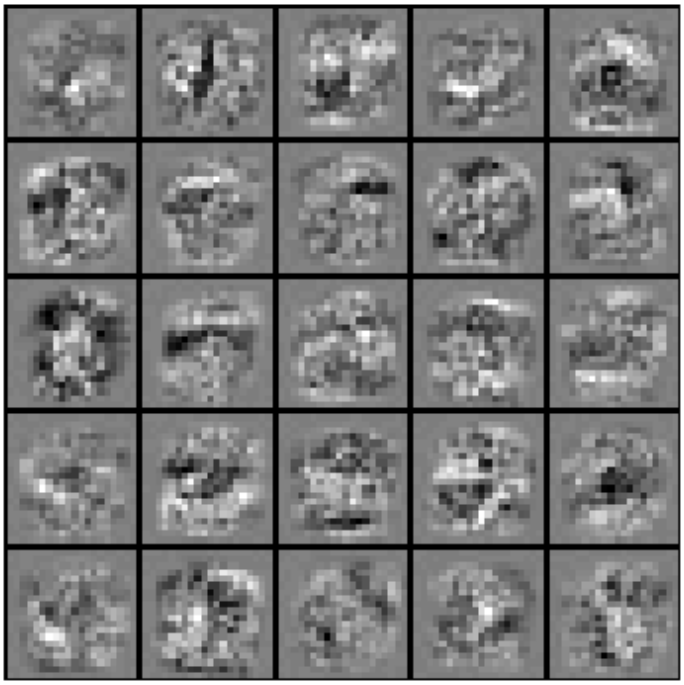
$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Пресметајте го (ненормализираниот) градиент за функцијата на чинење на неверонската мрежа со множење на збирот на градиентите со $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$



Слика 9: Четири различни прикази на бројот 4



Слика 10: Визуелизација на тренираната невронска мрежа

4.6 Примери

На слика 9 се прикажани четири различни примериоци на бројот 4 и во сите случаи алгоритмот точно ја предвидува прикажаната бројка. На слика 10 е прикажана визуелизација на тренираната невронска мрежа.

Литература

[1] *Coursera.org*, http://class.coursera.org/ml.

[2] *Stanford cs229*, http://cs229.stanford.edu/materials.html.

[3] C.M. Bishop et al., *Neural networks for pattern recognition*, (1995).

[4] T. Mitchell, *Machine learning (mcgraw-hill international edit)*, (1997).