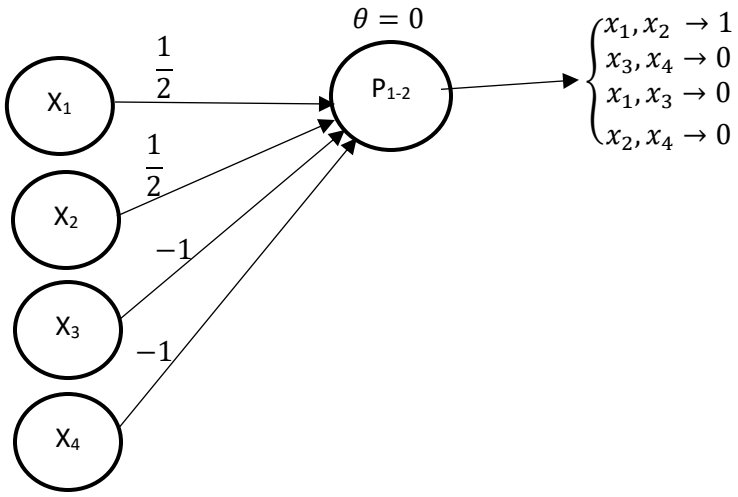
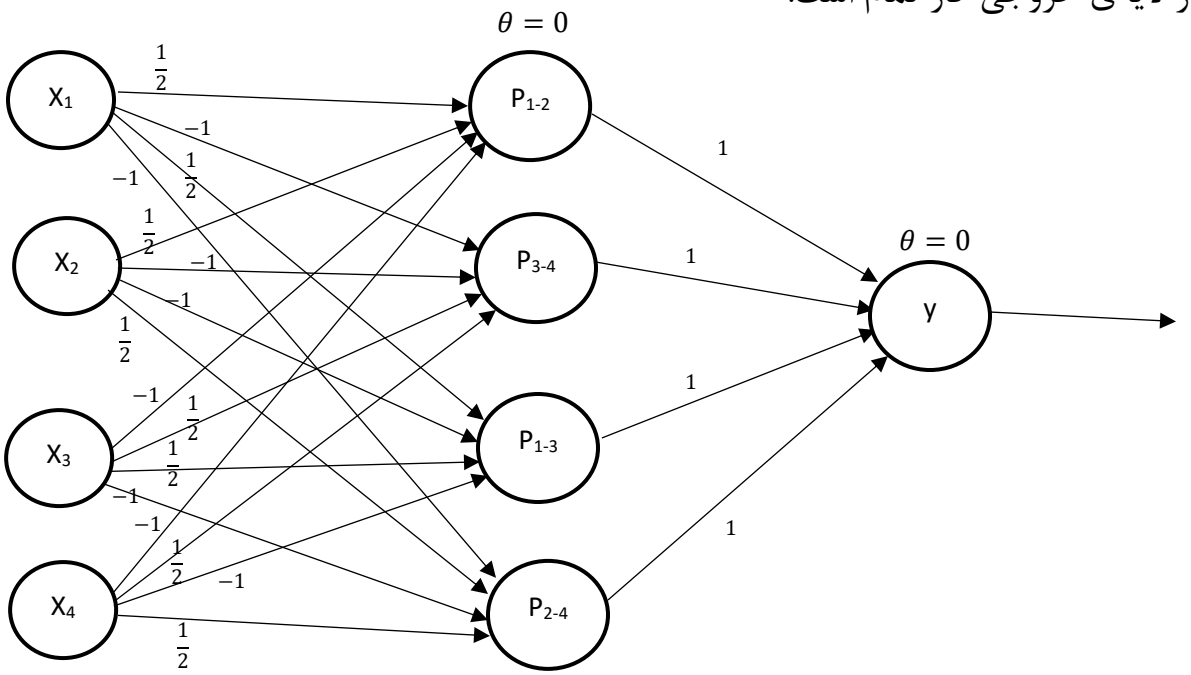
برای این کار کافی است فقط یک لایه‌ی مخفی با ۴ پرسپترون داشته باشیم. و در این لایه‌ی مخفی باید وزن‌ها را طوری قرار دهیم که در صورت وقوع هر کدام از چهار حالت ورودی فقط پرسپترون نظیر آن حالت رخ دهد. (می‌دانیم که می‌شود این کار را انجام داد؛ زیرا یکی از حالت‌های ورودی از بقیه به صورت خطی تفکیک‌پذیر است).

مثلا برای جداسازی حالت زیر از بقیه می‌توان به این شکل عمل کرد

| $x_1$ | $x_2$ |
|---|---|
| $x_3$ | $x_4$ |




حال کافی است چهار پرسپترون مانند بالا در لایه‌ی مخفی قرار دهیم، و در نهایت با یک پرسپترون در لایه‌ی خروجی کار تمام است.



در ابتدا به مطالعه‌ی توابع موجود در MATLAB می‌پردازیم و سپس بقیه‌ی بخش‌های سوال در گزارش نحوه‌ی انجام کار پوشش داده می‌شوند.

در MATLAB ابزار NEURAL NETWORK برای کار با شبکههای عصبی وجود دارد. این ابزار شامل چهار بخش اصلی زیر است:

- Function fitting
- Pattern recognition
- Data clustering
- Time-series analysis

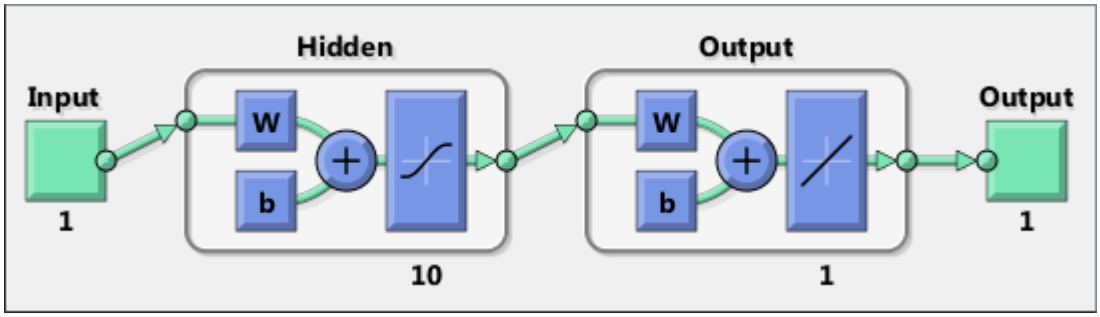
ما برای حل این مسئله باید از بخش Function fitting استفاده کنیم. زیرا در وردی و خروجی یک سری عدد داریم و می‌خواهیم از روی ورودی‌ها خورجی‌ها را تخمین بزنیم.

در MATLAB برای ایجاد شبکه‌ی چندلایه با الگوریتم یادگیری انتشار رو به عقب، از تابع feedforwardnet استفاده می‌شود. برای این تابع روش‌های یادگیری متفاوتی وجود دارد که همگی از ساختار انتشار رو به عقب استفاده می‌کنند. لیست این روش‌ها در زیر آمده است:

| Function | Algorithm |
|---|---|
| trainlm | Levenberg-Marquardt |
| trainbr | Bayesian Regularization |
| trainbfg | BFGS Quasi-Newton |
| trainrp | Resilient Backpropagation |
| trainscg | Scaled Conjugate Gradient |
| traincgb | Conjugate Gradient with Powell/Beale Restarts |
| traincgf | Fletcher-Powell Conjugate Gradient |
| traincgp | Polak-Ribiére Conjugate Gradient |
| trainoss | One Step Secant |
| traingdx | Variable Learning Rate Gradient Descent |
| traingdm | Gradient Descent with Momentum |
| traingd | Gradient Descent |

قابل استفاده از این تابع به این صورت است، که پارامتر اول تعداد گره‌های لایه‌ی مخفی و پارامتر دوم روش یادگیری است که لیست آن‌ها در بالا آمد.

```
net = feedforwardnet(hiddenSizes,trainFcn)
```




که با استفاده از تابع‌های موجود در کلاس net نیز می‌توان تعداد لایه‌ها و توابع انتقال و دیگر پارامترهای آن را تغییر داد.

در این مسئله از تابع fitnet استفاده می‌کنیم، که یک شبکه‌ی feedforward با یک لایه‌ی مخفی با تابع انتقال سیگموید و لایه‌ی خروجی با تابع انتقال خطی است.

به این شکل آن را ایجا می‌کنیم:

```
net = fitnet(hiddenLayerSize);
```

مانند کد زیر می‌توان مقدار سهم train، validation و test را از مجموعه داده‌ی ورودی تعیین کرد.

```
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
```

مانند کد زیر می‌توان وزن‌های شبکه را با استفاده از ماتریس input و target یادگیری کرد.

```
[net,tr] = train(net,inputs,targets);
```

ماتریس input مقادیر ورودی‌اند که در هر ستون آن یک نمونه وجود دارد و سطرها ویژگی‌ها را مشخص می‌کنند و بردار سطری target مقادیر خروجی تابعی هستند که شبکه می‌خواهد آن را یاد بگیرد.

خروجی net تابع شبکه‌ی یادگیری شده‌ی خروجی است.

با استفاده از net بدست آمده نیز می‌توان خروجی اعمال شبکه بر روی داده‌ها را بدست آورد.

```
outputs = net(inputs);
```

برای این قسمت از فایل‌های زیر استفاده شده است.

1. input.m  2. myTrainNet.m  3. mainProgramm.m

**فایل input.m**:

این فایل فقط شامل یک تابع به نام input است.

- ورودی: نام(آدرس) ۴ فایلی که در این مسئله استفاده می‌شوند.
- خروجی این تابع دو ماتریسی است که می‌توان از آن‌ها به عنوان ماتریس‌های ورودی و هدف در شبکه عصبی استفاده کرد.

در ابتدا با استفاده از تابع xlsread چهار فایل داده شده را می‌خوانیم. فایل ها به صورت زیر هستند.

- **فایل مصرف برق**

| Year | Month | Day | 0:30 | 1:00 | 1:30 | ... |
|---|---|---|---|---|---|---|
| 1997 | 1 | 1 | 797 | 794 | 784 | … |
| 1997 | 1 | 2 | 704 | 697 | 704 | |
| 1997 | 1 | 3 | 753 | 720 | 710 | |
| 1997 | 1 | 4 | 720 | 704 | 711 | |
| 1997 | 1 | 5 | 704 | 674 | 672 | |
| 1997 | 1 | 6 | 701 | 704 | 688 | |
| ... | | | ... | | | ... |

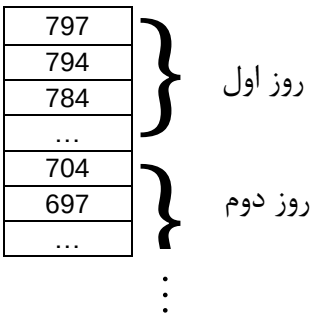
به همین دلیل با استفاده از کد زیر فقط قسمت حاوی داده ها را بر می‌داریم.

```
data1997(2:end,4:end)
```

سپس آن را به یک بردار با یک ستون تبدیل می‌کنیم. به طوری که تمام ساعت‌های کل سال پشت یر هم قرار بگیرند.

```
data1997 = reshape(data1997(2:end,4:end)', [], 1);
```

هم قرار بگیرند. که در نهایت به این صورت در می‌آیند.



در ادامه داده‌های هر دو سال را پشت سر هم در یک بردار قرا می‌دهیم.

```
allData = [data1997;data1998];
```

| |
|---|
| 797 |
| 794 |
| 784 |
| … |
| 704 |
| … |
| 728 |
| 738 |
| 708 |
| … |
| 668 |
| … |

سال ۱۹۹۷

سال ۱۹۹۸

**- فایل دما**

| Date | Temperature [oC] |
|---|---|
| 1/1/1997 | -7.6 |
| 1/2/1997 | -6.3 |
| 1/3/1997 | -3.0 |
| 1/4/1997 | 0.7 |
| … | … |

وقتی که تابع xlsread فایل را می‌خوانیم، یک بردار شامل دمای هر روز را به ما می‌دهد.

| |
|---|
| -7.6 |
| -6.3 |
| -3.0 |
| 0.7 |
| … |

برای این‌که متناظر با هر ساعت، دمای آن را بدانیم با استفاده از تابع repmat این بردار را به صورت ستونی ۴۸ بار تکرار می کنیم.

```
temper1997 = repmat(temper1997, 48,1);
```

که به این شکل خواهند شد.

| | | | | | | |
|---|---|---|---|---|---|---|
| -7.6 | -7.6 | -7.6 | -7.6 | | -7.6 | -7.6 |
| -6.3 | -6.3 | -6.3 | -6.3 | | -6.3 | -6.3 |
| -3.0 | -3.0 | -3.0 | -3.0 | ... | -3.0 | -3.0 |
| 0.7 | 0.7 | 0.7 | 0.7 | | 0.7 | 0.7 |
| … | … | … | … | | … | … |

حال مانند ماتریس مصرف برق آن را به یک بردار ستونی که تمامی ساعات سال پشت سر هم قرار دارند تبدیل می‌کنیم

```
temper1997 = reshape(temper1997', [], 1);
```

| |
|---|
| -7.6 |
| -7.6 |
| -7.6 |
| … |
| -6.3 |
| -6.3 |
| … |
| ⋮ |

روز اول

روز دوم

و در ادامه اطلاعات هر دو سال را پشت سر هم قرار می‌دهیم.

```
allTemper = [temper1997;temper1998];
```

| |
|---|
| -7.6 |
| -7.6 |
| -7.6 |
| … |
| -6.3 |
| … |
| 0.8 |
| 0.8 |
| 0.8 |
| … |
| 4.6 |
| … |

سال ۱۹۹۷

سال ۱۹۹۸

حال از این دو بردار بدست آمده برای ساختن ماتریس‌های ورودی و هدف مورد نیاز شبکه عصبی استفاده می‌کنیم. طبق صورت سوال ویژگی‌های مورد نیاز برای تخمین برق مصرفی در یک ساعت، برق مصرفی در ساعت قبل، روز قبل همان ساعت، هفته‌ی قبل همان ساعت، ماه قبل همان ساعت، سال قبل همان ساعت و دمای همه‌ی این ساعت‌هاست. یعنی ۱۰ ویژگی داریم. و البته می‌دانیم که یادگیری را روی داده‌های سال ۱۹۹۸ انجام می‌دهیم، زیرا داده های سال ۱۹۹۷ همه ویژگی برق مصرفی در در سال قبل همان ساعت را ندارند.

پس ماتریس ورودی و هدف باید به این صورت باشد:

| برق مصرفی ساعت قبل | برق مصرفی روز قبل | برق مصرفی هفته قبل | برق مصرفی ماه قبل | برق مصرفی سال قبل | دمای ساعت قبل | دمای روز قبل | دمای هفته قبل | دمای ماه قبل | دمای سال قبل |
|---|---|---|---|---|---|---|---|---|---|
| 692 | 673 | 641 | 650 | 797 | -1.4 | -1.4 | 2.3 | 6.1 | -7.6 |
| 728 | 668 | 633 | 641 | 794 | 0.8 | -1.4 | 2.3 | 6.1 | -7.6 |
| 738 | 648 | 625 | 627 | 784 | 0.8 | -1.4 | 2.3 | 6.1 | -7.6 |
| … | … | … | … | … | … | … | … | … | … |
| 659 | 728 | 606 | 654 | 704 | 0.8 | 0.8 | 2.2 | 4.4 | -6.3 |
| 704 | 738 | 601 | 646 | 697 | 4.6 | 0.8 | 2.2 | 4.4 | -6.3 |
| … | … | … | … | … | … | … | … | … | … |

ماتریس ورودی

بردار هدف

| برق مصرفی سال ۱۹۹۸ |
|---|
| 728 |
| 738 |
| 708 |
| … |
| 704 |
| 697 |
| … |

برای ساختن بردار هدف کافی است داده‌های برق مصرفی در سال ۱۹۹۸ را همان‌طور که در یک بردار پشت سر هم قرار دادیم در نظر بگیریم.

```
targets = data1998;
```

برای ایجاد ماتریس ورودی در ابتدا یک ماتریس با ۱۰ ستون و ۳۶۵*۴۸ سطر ایجاد می‌کنیم.

```
nh = 365*48;
inputs = zeros(nh, 10);
```

حال با استفاده از بردارهای alldata که برق مصرفی هر دوسال را (پشت سر هم) در خود دارد و allTemper که دمای هر دوسال را (پشت سر هم) در خود دارد، ستون‌های ماتریس input را به صورت زیر پر می‌کنیم.

- ستون اول:

میزان برق مصرفی هر ساعت در سطر قبل از آن قرار دارد، بنابراین با شیفت دادن بردار alldata به اندازه‌ی یک واحد، و انتخاب داده‌های متناظر با سال ۱۹۹۸(۳۶۵*۴۸ سطر آخر)، برق مصرفی ساعت‌های قبل متناظر با تمام ساعات سال ۱۹۹۸ بدست می‌آیند.

```
temp = circshift(allData, 1);
inputs(:, 1) = temp(end-nh+1:end);
```

- برای ستون دوم تا پنجم فقط میزان شیفت دادن متفاوت است
  - ستون دوم ۴۸ شیفت
  - ستون سوم ۷*۴۸ شیفت
  - ستون چهارم ۳۰*۴۸ شیفت
  - ستون چنجم داده‌های سال ۱۹۹۷

```
temp = circshift(allData, 48);
inputs(:, 2) = temp(end-nh+1:end);
temp = circshift(allData, 7*48);
inputs(:, 3) = temp(end-nh+1:end);
temp = circshift(allData, 30*48);
inputs(:, 4) = temp(end-nh+1:end);
% temp = circshift(allData, 365*48);
% inputs(:, 5) = temp(end-nh+1:end);
inputs(:, 5) = data1997;
```

- برای ستون ششم تا دهم نیز مانند بالا عمل می‌کنیم، فقط با این تفاوت که از بردار دما یا allTemper استفاده خوایم کرد.

```
temp = circshift(allTemper, 1);
inputs(:, 6) = temp(end-nh+1:end);
temp = circshift(allTemper, 48);
inputs(:, 7) = temp(end-nh+1:end);
temp = circshift(allTemper, 7*48);
inputs(:, 8) = temp(end-nh+1:end);
temp = circshift(allTemper, 30*48);
inputs(:, 9) = temp(end-nh+1:end);
% temp = circshift(allTemper, 365*48);
% inputs(:, 10) = temp(end-nh+1:end);
inputs(:, 10) = temper1997;
```

## فایل mTtrainNet.m :

این فایل فقط شامل یک تابع به نام trainNet است.

- ورودی: ماتریس ورودی (input) و هدف (target) و تعداد گره‌های لایه ی مخفی (hiddenLayerSize)
- خروجی این تابع مقدار خطای شبکه (MAPE) بر اساس فرمول گفته شده در صورت سوال است..

روند کار به این صورت است که ابتدا یک شبکه عصبی چندلایه با استفاده از fitnet ایجاد می‌کنیم.

```
net = fitnet(hiddenLayerSize);
```

میزان سهم train و test را از داده‌های اصلی مشخص می‌کنیم.

```
net.divideParam.trainRatio = 70/100;
%net.divideParam.valRatio = 0;%15/100;
net.divideParam.testRatio = 30/100;
```

داده‌ها را مطابق آن‌چه در صورت سوال گفته نرمالایز می‌کنیم.

```
mins = min(inputs);
maxs = max(inputs);
mins = repmat(mins, size(inputs,1),1);
maxs = repmat(maxs, size(inputs,1),1);
inputs = (inputs - mins)./(maxs - mins);
```

ماتریس input و بردار ستونی target را ترانهاده می‌کنیم تا داده‌ها مناسب استفاده در تابع fitnet شوند.

```
inputs = inputs';
targets = targets';
```

سپس با استفاده از تابع train و net شبکه را یاد می‌گیریم و خروجی اعمال آن‌ها را روی داده‌ها بدست می‌آوریم. سپس خطا را حساب می‌کنیم.

```
[net,tr] = train(net,inputs,targets);
outputs = net(inputs);
MAPE = sum((outputs-targets)/outputs)*100/nh;
```

**فایل mainProgramm.m** :

در این فایل از تابع input و myTrainNet استفاده می‌کنیم و برای هر hiddenLayerSize، ۱۰ بار آزمایش را تکرار می‌کنیم و میانگین خطا را بدست می‌آوریم. در زیر جدول خواسته شده در صورت سوال آمده است.

| تعداد نورون‌های مخفی | 40 | 20 | 14 | 11 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|
| MAPE | 0.0000014536 | 0.0000018066 | 0.0000010778 | 0.0000015460 | 0.0000014158 | 0.0000011034 | 0.0000014023 |

به این دلیل باید هر آزمایش را چندین بار تکرار کنیم که مقادیر اولیه‌ی شبکه تصاوفی هستند و همین‌طور در رسیدن به مقادیر نهایی، شبکه ممکن است هر بار به مقادیر متفاوتی همگرا شود.