گزارش تمرین چهارم هوش محاسباتی

غزاله محمودی

Inverted pendulum

در این قسمت با کمک مباحث مطرح شده درباره کنترلر فازی به پیاده سازی آن پرداختیم . سیستم رو ورودی theta , theta dot و خروجی F می باشد .

اساس کلی کار به این صورت است که بر اساس دامنه ورودی به یک متغیرهای ورودی را به term های مختلف تقسیم می کنیم .

و سپس با توجه به مشاهدات و آزمون و خطا چندین قاعده می نویسیم که کنترلر ورودی بر اساس ورودی و قواعد تصمیم می گیرد F چه term فازی باشد و چه مقدار عددی را بگیرد .

Hopfield network

در این سوال تصاویر ۱۰ حروف اول الفبای انگلیسی ساخته و ذخیره می‌شوند و سپس درصد های متفاوتی از noise به این تصاویر اعمال می‌شود و سپس با استفاده از شبکه هاپفیلد سعی می‌شود که تصاویر را تا حدی به شکل اصلیشان بازیابی کنیم.

ابتدا داده های اصلی را می سازیم .

```
for char in alphabets:
    img =  Image.Image()._new(font.getmask(char, mode='L'))
    img.save("train - " + str(font_size)+ " - "+ char + ".jpg")
    shape[char] = img.size
    imgs[char] = img
```

سپس داده اصلی را با درصد 10 و 30 و 60 نویزی می کنیم و تصاویر حاصل را مجدد ذخیره می کنیم .

```
for char in alphabets:
    img = np.array(train_imgs[font_size][char])

    pixels_count = img.shape[0] * img.shape[1]
    noisypx_count = int(noise * pixels_count)

    random_pxs = random.sample(range(0, pixels_count), noisypx_count)

    for i in random_pxs:
        row = int(i / img.shape[1])
        column = int(i % img.shape[1])

        img[row,column] = 255 if img[row,column] < 125  else 0

    img = Image.fromarray(img, mode="L")
    imgs[noise][char] = img
    img.save("test -" + str(font_size) + " - noise - "+ str(noise)+ " - "+ char + " - "+ ".jpg")
```

در اینجا به اندازه درصد نویز از پیکسل ها انتخاب می کنیم و رنگ آن ها را برعکس میکنیم . (سفید ها سیاه و برعکس)

و کار ساخت دیتا را برای همه فونت ها (16-32-64) و همه نویز ها (10 و 30 و 60) انجام می دهیم

پس از اتمام کار ساخت دیتا باید با استفاده از عکس های اصلی شبکه را train کنیم و ماتریس وزن متناسب با آن سایز را پیدا کنیم .

```
# num_files is the number of files -- Training
w = []
for char in alphabets:
    print (char, "   ", "font : ", font)
    x = image_to_array(train_files[char])
    x_vec = matrix_to_vector(x)

    if char == 'A':
        w = create_matrix_w(x_vec)
    else:
        w = w + create_matrix_w(x_vec)

print(w)
```

و می دانیم ماتریس وزن در شبکه هاپفیلد متقارن است . سپس با قواعدی که می دانیم ماتریس وزن را میسازیم . (در اینجا می خواهیم می خواهیم 10 پترن را ذخیره کنیم)

```
# Create Weight matrix for a single image
def create_matrix_w(x):
    w = np.zeros([len(x),len(x)])
    for i in range(len(x)):
        for j in range(i,len(x)):
            if i == j:
                w[i, j] = 0
            else:
                w[i, j] = x[i]*x[j]
                w[j, i] = w[i, j]
    return w
```

سپس عکس های نویزی را به ترتیب به ماتریس وزن می دهیم و شبکه به دنبال نزدیک ترین پترن برای مطابقت دادن می گردد .

```
#Import test data -- testing
for noise in noises :
    for char in alphabets:
        y = image_to_array(test_files[noise][char])

        oshape = y.shape
        print( "test data : ", "noise = ", noise,"font :",    font )

        y_vec = matrix_to_vector(y)

        y_vec_after = update(w=w, y_vec=y_vec, theta=theta, time=time)
        y_vec_after = y_vec_after.reshape(oshape)
```

بسته به آستانه ای که در تابع image_to_array می گذاریم و همچنین اینکه تصویر پایانی را به سایز اولیه ذخیره کنیم یا با سایز 100 در 100 نتایج مختلفی پدید می آید .

اما به طور کلی هر چه فونت بزرگتر باشد نتیجه بهتری حاصل می شود و بدیهی است هر چه نویز کتر باشد بازسازی عکس بهتر است .

برای نویز 10 در صد شبکه می تواند پترن را تشخیص دهد ولی با افزایش نویز به 60 درصد اکثر ورودی ها را E و G تشخیص میدهد که نشان دهنده عدم مقاومت هاپفیلد در مقابل نویز بالا است .

نتایج در فایل output قابل مشاهده است .