**دانشگاه صنعتی امیرکبیر**

**(پلی تکنیک تهران)**

**دانشکده ریاضی و علوم کامپیوتر**

**درس هوش مصنوعی و کارگاه**

گزارش ۷: پیاده سازی مقاله

# Hybrid MPSO-CNN: Multi-level Particle Swarm optimized hyperparameters of Convolutional Neural Network

**نگارش**

**کیارش مختاری دیزجی**

**۹۸۳۰۰۳۲**

**استاد اول**

**دکتر مهدی قطعی**

**استاد دوم**

**بهنام یوسفی مهر**

**تیر ۱۴۰۲**

# چکیده

پیشـرفت‌های اخیر در الگوریتم‌های بهینه‌سـازی الهام‌گرفته از Swarm، پذیرش گسـترده آن را در حل طیف گسـترده‌ای از مسـائل مختلف دنیای واقعی نشـان داده اسـت. Particle Swarm Optimization (PSO) یکی از کاوش‌شـده‌ترین الگوریتم‌های بهینه‌سـازی تصـادفی مبتنی بر جمعیت و مبتنی بر طبیعت است. در این مقاله، یک الگوریتم بهینه‌سازی ازدحام ذرات چند سطحی (MPSO) برای یافتن معماری و hyperparameterهای شـبکه عصـبی کانولوشـنال (CNN) به طور همزمان پیشـنهاد شـده اسـت. این یادگیری خودکار هزینه‌های سربار متخصصان انسانی را برای یافتن این پارامترها از طریق تجزیه و تحلیل دسـتی و آزمایش‌ها کاهش می‌دهد. راه حل پیشـنهادی از دسـته‌های متعدد در دو سـطح اسـتفاده می‌کند. ازدحام اولیه در سـطح ۱ معماری را بهینه می‌کند و ازدحام‌های متعدد در سـطح ۲ hyperparameterها را بهینه می‌کند. روش پیشـنهادی از وزن اینرسـی مانند سـیگموئید برای تنظیم خاصـیت اکتشـاف و بهره‌برداری ذرات و جلوگیری از همگرایی زودهنگام الگوریتم PSO به یک راه حل بهینه محلی، اسـتفاده کرده اسـت. در این مقالـه، رویکردی را برای پیشـنهاد بهترین معمـاری CNN بـا شـرایط خوب و hyperparameterهای آن با اسـتفاده از MPSO در یک فضـای جسـتجوی مشـخص بررسـی شـده اسـت. پیچیدگی و عملکرد MPSO-CNN به ابعاد فضـای جسـتجو بسـتگی دارد. نتایج تجربی بر روی ۵ مجموعه داده معیار MNIST، CIFAR-10، CIFAR-100، Convex Sets و MDRBI یک کاربرد مؤثر از PSO را در یادگیری معماری عصبی عمیق نشان داده‌اند.

## واژه‌های کلیدی:

PSO، CNN، hyperparameter

# فهرست مطالب

صفحه

# فهرست اشکال

صفحه

# ۱. فصل اول

## مقدمه

معماری‌های شبکه عصبی با بیش از سه لایه برای حل مسائل واقعی پیچیده‌تر ترجیح داده می‌شوند. برای آموزش مناسب، نیاز به حجم بزرگی از داده، منابع محاسباتی سریع و زمان محاسباتی زیاد است که چالش‌هایی ایجاد می‌کند. روش‌های سنتی یادگیری ماشین بستگی به ویژگی‌های دستساز شده دارند، اما یادگیری عمیق (DL)، به دلیل عملکرد مؤثر، آن روش‌های سنتی یادگیری ویژگی را بهبود می‌بخشد. در روش‌های یادگیری عمیق، نیاز به تنظیم تعداد زیادی پارامتر وجود دارد و پیچیدگی آن با افزایش تعداد لایه‌های مخفی افزایش می‌یابد. تخصص انسانی برای تعیین این پارامترها لازم است. CNN در بین تمام شبکه‌های عمیق بسیار محبوب است و کاربردهای گسترده‌ای در بینایی ماشین دارد. تحقیقات بسیاری در گذشته برای بهینه‌سازی انتخاب پارامتر در شبکه‌های عمیق با استفاده از الگوریتم‌های هوش مصنوعی انجام شده است. پژوهشگران نسخه‌های ترکیبی از شبکه‌های CNN برای بهینه‌سازی انتخاب پارامتر توسعه داده‌اند. PSO مشابه الگوریتم‌های تکاملی دیگر مانند الگوریتم ژنتیک (GA)، بهینه‌سازی کلونی مورچه‌ای (ACO) و غیره است. الگوریتم‌های هوش اجتماعی اصلی، ACO و PSO هستند. PSO به عنوان گزینه‌ای پرطرفدار در حل مسائل بهینه‌سازی استفاده می‌شود زیرا دارای تعداد کمتری پارامتر، فرمولاسیون ساده و محاسبات آسان است. پیشرفت‌های اخیر در برنامه‌های کاربردی PSO در زمینه شبکه‌های عصبی است که همچنین عنصر اصلی این مقاله می‌باشد. PSO با نسخه‌های مختلف خود بهبود یافته است تا راه حل‌های بهتری ارائه دهد. PSO عمدتاً از همگرایی زودهنگام رنج می‌برد که ممکن است به دلیل عدم تنوع در طبیعت ذرات رخ دهد. نسخه‌های PSO می‌توانند با تغییر مقداردهی اولیه جمعیت، بهبود انتخاب پارامترها، تنظیم ساختار توپولوژیکی آن و پیشنهاد نسخه‌های ترکیبی آن با سایر الگوریتم‌ها، این مشکل را به صورت‌های مختلفی حل کنند. Modified-PSO یکی از توسعه‌های اخیر در این زمینه است. این روش از مقداردهی اولیه مبتنی بر آشوب برای تولید ذرات با توزیع یکنواخت استفاده می کند.

# ۲. فصل دوم

# الگوریتم PSO و ساختار شبکه CNN

Particle Swarm Optimization (PSO) یک الگوریتم بهینه سازی مبتنی بر جمعیت است که از رفتار جمعی موجودات اجتماعی، مانند گله پرندگان یا پرورش ماهی الهام گرفته شده است. در PSO، جمعیتی از راه‌حل‌های کاندید، به نام ذرات، به طور مکرر به دنبال راه‌حل بهینه در یک فضای جستجو می‌گردند. هر ذره یک راه حل بالقوه را نشان می دهد و با تنظیم موقعیت خود بر اساس تجربه خود و تجربیات ذرات همسایه در فضای جستجو حرکت می کند.

در اینجا شبه کدی از الگوریتم PSO آورده شده است:

```
Step1: Randomly initialize Swarm population of N particles Xi
(i=1, 2, …, n)
Step2: Select hyperparameter values w, c1 and c2

Step 3: For Iter in range(max_iter):  # loop max_iter times
            For i in range(N):  # for each particle:
               a. Compute new velocity of ith particle
                     swarm[i].velocity =
                          w*swarm[i].velocity +
                          r1*c1*(swarm[i].bestPos -
swarm[i].position) + r2*c2*( best_pos_swarm - swarm[i].position)

               b. Compute new position of ith particle using its
new velocity
                     swarm[i].position += swarm[i].velocity
               c. If position is not in range [minx, maxx] then
clip it
                     if swarm[i].position < minx:
                         swarm[i].position = minx
                     elif swarm[i].position > maxx:
                         swarm[i].position = maxx
               d. Update new best of this particle and new best
of Swarm
                      if swaInsensitive to scaling of design
variables.rm[i].fitness < swarm[i].bestFitness:
                         swarm[i].bestFitness = swarm[i].fitness
                         swarm[i].bestPos = swarm[i].position

                      if swarm[i].fitness < best_fitness_swarm
                         best_fitness_swarm = swarm[i].fitness
                         best_pos_swarm = swarm[i].position
             End-for
         End -for.
```

شبکه‌های عصبی کانولوشنی (CNN) نوعی معماری یادگیری عمیق هستند که به طور خاص برای پردازش داده‌های شبکه‌ای مانند تصاویر یا دنباله‌ها طراحی شده‌اند. در ادامه توضیحی از معماری معمول CNN آمده است:

**۱. لایه ورودی:**

لایه ورودی اطلاعات ورودی را دریافت می‌کند، که معمولاً به صورت تصاویر با مقادیر پیکسلی نمایش داده می‌شوند. داده‌های ورودی معمولاً پیش‌پردازش می‌شوند، از جمله تغییر اندازه، نرمال‌سازی یا تکنیک‌های افزایش داده.

**۲. لایه‌های کانولوشنی:**

لایه‌های کانولوشنی به عنوان بخش اصلی شبکه‌های عصبی کانولوشنی عمل می‌کنند. این لایه‌ها از چندین فیلتر یا هسته یادگیری شده تشکیل شده‌اند. هر فیلتر با داده ورودی کانولوشن می‌شود و به طول فضایی به منظور تشخیص الگوها یا ویژگی‌های مرتبط در مناطق تصویری مشخص می‌گردد. لایه‌های کانولوشنی با یادگیری ویژگی‌های پیچیده‌تر در مقیاس‌های مختلف، سلسله مراتبی فضایی را ثبت می‌کنند.

**۳. تابع فعال‌سازی:**

معمولاً بعد از هر عمل کانولوشن، تابع فعال‌سازی غیرخطی مانند ReLU به صورت مؤلفه‌مند به اعمال کانولوشنی، اعمال می‌شود تا غیرخطی بودن را وارد کند و شبکه را قادر به یادگیری روابط پیچیده کند.

**۴. لایه‌های Pooling**:

لایه‌های Pooling برای کاهش ابعاد فضایی نقشه ویژگی استفاده می‌شوند در حالی که اطلاعات مهم‌تر را حفظ می‌کنند. عملیات‌های معمول Pooling شامل بیشینه‌گیری (انتخاب حداکثر مقدار درون حجم) یا میانگین‌گیری (محاسبه مقدار میانگین درون حجم) می‌باشد. حجم‌گیری پیچیدگی محاسباتی را کاهش می‌دهد، در استخراج ویژگی‌های قوی کمک می‌کند.

### ۵. لایه‌های Fully Connected:

لایه‌های کاملاً متصل، همچنین به عنوان لایه‌های چگال شناخته می‌شوند، معمولاً در انتهای معماری CNN قرار دارند. این لایه‌ها هر عصب در لایه قبلی را به هر عصب در لایه فعلی وصل می‌کنند، مشابه شبکه‌های عصبی سنتی. لایه‌های کاملاً متصل بر اساس ویژگی‌های استخراج شده توسط لایه‌های کانولوشنی قبلی عمل می‌کنند و استدلال و دسته‌بندی در سطح بالاتر را انجام می‌دهند.

### ۶. لایه خروجی:

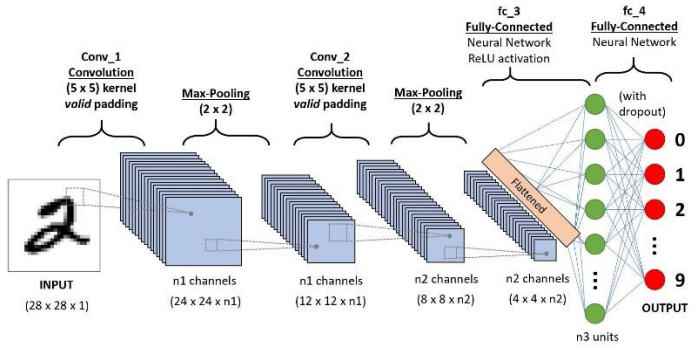
لایه نهایی شبکه، لایه خروجی است که پیش‌بینی‌های شبکه را تولید می‌کند. تعداد عصب‌هایی که در لایه خروجی بستگی به وظیفه خاص دارد. به عنوان مثال، در دسته‌بندی تصاویر، هر عصب متناظر با یک برچسب کلاس است و خروجی معمولاً از طریق تابع softmax به دست می‌آید تا توزیع احتمالی بدست آید.

### ۷. Backpropagation و آموزش:

شبکه‌های عصبی کانولوشنی با استفاده از Backpropagation آموزش داده می‌شوند، که در آن خطای پیش‌بینی شده و خطای صحیح را برای به‌روزرسانی وزن‌های شبکه استفاده می‌کنند. الگوریتم‌های بهینه‌سازی مانند کاهش گرادیان تصادفی (SGD) یا نسخه‌های آن معمولاً برای کمینه کردن خطا و به‌روزرسانی پارامترهای شبکه استفاده می‌شوند.

### ۸. تکرار پشته‌های لایه:

معماری شبکه‌های عصبی کانولوشنی معمولاً شامل چندین پشته از لایه‌های کانولوشنی، توابع فعال‌سازی و لایه‌های حجم‌گیری است. این تکرارهای پشته لایه‌ها در استخراج ویژگی‌های پیچیده و یادگیری تمثیل‌های سلسله مراتبی موثر هستند.




۱- نمونه‌ای از مدل CNN

**Algorithm 2**
Hybrid MPSO-CNN algorithm at swarm level-1.

***Input:*** maximum number of iterations ($tmax^1$) and search space for hyperparameters
***Output:*** (CNN hyperparameters: *[nC, nP, nF, c_nf, c_fs, c_pp, c_ss, p_fs, p_ss, p_fs, op]*, fitness value)

***Algorithm:***
initialize particle's position vector in specified range: [nC, nP, nF]
while (maximum number of iterations is not reached: $tmax^1$)
Calculate $\omega$ using Eq. (3)
    for each particle $i$ = 1 to m of swarm at level-1 do
        find hyperparameters and its fitness value (as in Algorithm 3)
        update fitness value: $F_i$ =fitness<$P_i$ > = fitness ($P_i, gbest_i$)
        update personal best: $\text{pbest}_i$
        update global best: gbest
        update particle's velocity and position: ($V_i, P_i$)
    end
end
return(gbest, CNN (gbest))

الگوریتم ترکیبی MPSO-CNN در سطح swarm ۱ یک الگوریتم بهینه‌سازی است که برای پیدا کردن ساختار و هایپرپارامترهای شبکه عصبی کانولوشنال (CNN) به صورت همزمان استفاده می‌شود. این الگوریتم از ترکیب الگوریتم بهینه‌سازی ذرات (PSO) با شبکه عصبی کانولوشنال استفاده می‌کند. در سطح swarm ۱، ذرات به صورت تصادفی در محدوده‌ای مشخص مقادیر اولیه می‌گیرند و سپس با محاسبه مقدار تابع فیتنس و به‌روزرسانی بهترین شخصی و بهترین جهانی، سرعت و موقعیت ذره‌ها به‌روزرسانی می‌شوند. این الگوریتم برای یافتن بهترین ساختار و هایپرپارامترهای CNN به صورت خودکار و بدون نیاز به دخالت انسانی استفاده می‌شود. نتایج تجربی نشان داده است که الگوریتم Hybrid MPSO-CNN قادر به بهبود عملکرد شبکه عصبی کانولوشنال در مسائل پیچیده و واقعی است.

**Algorithm 3**
Hybrid MPSO-CNN algorithm at swarm level-2.

***Input:*** particle ($P_i$) of swarm level-1, maximum number of iterations ($tmax^2$) and search space for hyperparameters
***Output:*** (CNN hyperparameters, fitness value))

***Algorithm:***
for each particle $j$ = 1 to n of swarm at level-2 do
    initialize particle's position in specified range: [c_nf, c_fs, c_pp, c_ss, p_fs, p_ss, p_fs, op]
    setup a CNN model: CNN($P_i$, $P_{ij}$)
    compute fitness value using CNN: $F_{ij}$
    initialize personal best: $pbest_{ij}$
    initialize global best: $gbest_i$
end
while (maximum number of iterations is not reached: $tmax^2$)
    for each particle $j$ = 1 to n of swarm at level-2 do
        update particle's velocity and position: ($V_{ij}$, $P_{ij}$)
        setup a CNN model: CNN($P_i$, $P_{ij}$)
        compute fitness value using CNN: $F'_{ij}$
        update personal best: $pbest_{ij}$
        update global best: $gbest_i$
    end
end
return(($P_i$, $gbest_i$), CNN($P_i$, $gbest_i$))

الگوریتم ترکیبی MPSO-CNN در سطح swarm ۲ به منظور بهینه‌سازی هایپرپارامترهای شبکه عصبی کانولوشنال (CNN) طراحی شده است. در این الگوریتم، چندین swarm به منظور بهتر جستجو در فضای هایپرپارامترها استفاده می‌شود. هر ذره در swarm یک ترکیب خاص از هایپرپارامترها را نمایندگی می‌کند و مقدار تابع فیتنس آن محاسبه می‌شود. الگوریتم بهترین موقعیت شخصی و بهترین موقعیت جهانی را بر اساس مقادیر فیتنس به‌روزرسانی می‌کند. سپس با استفاده از مکانیزم MPSO، سرعت و موقعیت ذرات به‌روزرسانی می‌شوند. با ترکیب هوش گل‌ها و فرایند بهینه‌سازی، هدف از الگوریتم ترکیبی MPSO-CNN در سطح شاره ۲ یافتن مجموعه بهینه از هایپرپارامترها برای شبکه عصبی کانولوشنال است که منجر به بهبود عملکرد در حل مسائل پیچیده و واقعی می‌شود.

# ۳. فصل سوم

# پیاده سازی الگوریتم MPSO-CNN

۱. در این قسمت از کد، ما داده‌های MNIST را برای آموزش و آزمون مدل آماده می‌کنیم.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

در این جا ابتدا دیتاست MNIST را لود کرده‌ایم و سپس مقادیر پیکسل‌های تصاویر ورودی را از اعداد صحیح (بین ۰ تا ۲۵۵) به اعداد اعشاری بین ۰ تا ۱ تبدیل می‌کنیم و بعد یک بعد اضافی به تصاویر ورودی اضافه می‌کنیم برای اینکه بعد تصاویر به صورت (۱و۲۸و۲۸) بشود و در انتها برچسب های تصاویر را با استفاده از OneHot encoding به مقادیر عددی ۰ تا ۹ تبدیل می‌کنیم.

۲.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

در این بخش ما یک مدل sequential ساخته‌ایم و سپس لایه‌های کانولوشن و maxpooling و سپس دوباره لایه‌ی کانولوشن و maxpooling و در انتها لایه های flatten برای اتصال لایه کانولوشن به لایه dense استفاده شده است.

۳.

```
def fitness_function(position):
    model.set_weights(position)
    model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])
    model.fit(x_train, y_train, batch_size=128, epochs=5, verbose=0)
    y_pred = model.predict(x_test)
    y_pred = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_test, axis=1)
    accuracy = accuracy_score(y_true, y_pred)
    return -accuracy
```

در این قسمت از کد، تابع fitness_function تعریف شده است که برای محاسبه سطح فیتنس (fitness) یک موقعیت (position) در فضای جستجو استفاده می‌شود. این تابع عملکرد مدل را بر اساس موقعیت وزن‌ها تنظیم می‌کند و سپس مدل را بر روی داده‌های آموزش x_train و y_train آموزش می‌دهد.

۴.

```
class Particle:
    def __init__(self, position):
        self.position = position
        self.velocity = [np.zeros_like(p) for p in position]
        self.best_position = copy.deepcopy(self.position)
        self.best_fitness = fitness_function(self.position)

    def update_velocity(self, global_best_position, w, c1, c2):
        r1 = [np.random.random(p.shape) for p in self.position]
        r2 = [np.random.random(p.shape) for p in self.position]
        self.velocity = [w * v + c1 * r1i * (p_best - p) + c2 * r2i *
(g_best - p)
                         for v, r1i, r2i, p_best, g_best, p in
zip(self.velocity, r1, r2, self.best_position, global_best_position,
self.position)]

    def update_position(self):
        self.position = [p + v for p, v in zip(self.position,
self.velocity)]
        fitness = fitness_function(self.position)
        if fitness < self.best_fitness:
            self.best_position = copy.deepcopy(self.position)
            self.best_fitness = fitness
```

```
def mpso_optimization(num_particles, num_iterations, w, c1, c2):
    global_best_position = None
    global_best_fitness = -np.inf
    particles = []

    for _ in range(num_particles):
        position = [layer + np.random.uniform(low=-1, high=1,
size=layer.shape) for layer in model.get_weights()]
        particle = Particle(position)
        particles.append(particle)

        if particle.best_fitness > global_best_fitness:
            global_best_position = [np.copy(pos) for pos in
particle.best_position]
            global_best_fitness = particle.best_fitness

    for _ in range(num_iterations):
        for particle in particles:
            particle.update_velocity(global_best_position, w, c1, c2)
            particle.update_position()

            if particle.best_fitness > global_best_fitness:
                global_best_position = [np.copy(pos) for pos in
particle.best_position]
                global_best_fitness = particle.best_fitness

    return global_best_position
```

در این قسمت از کد، یک کلاس به نام Particle و همچنین تابع mpso_optimization تعریف شده است. این تابع ذرات را با موقعیت‌های تصادفی مقداردهی اولیه کرده، فیتنس آن‌ها را محاسبه کرده و بهترین موقعیت و فیتنس سراسری را به‌روز می‌کند. سپس در هر تکرار، سرعت و موقعیت هر ذره را به‌روزرسانی کرده و در صورت پیدا کردن یک جواب بهتر، بهترین موقعیت و فیتنس سراسری را به‌روز می‌کند .در نهایت، مقدار بهترین موقعیت سراسری را برمی‌گرداند که معادل مجموعه بهینه شده وزن‌ها برای مدل شبکه عصبی است.

# ٤. فصل چهارم

# نتیجه‌گیری

این الگوریتم، که از ترکیب روش MPSO و شبکه‌های عصبی کانولوشنی (CNN) استفاده می‌کند، برای بهینه‌سازی معماری و هایپر پارامترهای شبکه‌های عصبی توسعه یافته است. با استفاده از الگوریتم MPSO، به دنبال پیدا کردن ساختار مناسبتری برای شبکه‌های عصبی هستیم. MPSO با رویکرد سلسله مراتبی، در سطح اول بهینه‌سازی لایه‌های شبکه را انجام می‌دهد و در سطح دوم بهینه‌سازی هایپر پارامترهای هر لایه را انجام می‌دهد. از این روش برای انتخاب تعداد لایه‌ها و هایپر پارامترهای مختلف هر لایه استفاده می‌کنیم. با استفاده از تکنیک اینرسی مشابه سیگموئید، همگرایی زودرس را کنترل می‌کنیم. نتایج این الگوریتم روی پنج مجموعه داده نشان می‌دهد که MPSO-CNN موفقیت‌آمیز در بهینه‌سازی ساختار شبکه‌های عصبی بدون نیاز به دخالت انسانی است. این رویکرد کارآمد است و می‌تواند برای حل مشکلات مشابه دیگر نیز استفاده شود.

# منابع

Hybrid MPSO-CNN: Multi-level Particle Swarm optimized hyperparameters of Convolutional Neural Network Pratibha Singh * , Santanu Chaudhury , Bijaya Ketan Panigrahi

https://www.geeksforgeeks.org/particle-swarm-optimization-pso-an-overview/

https://github.com/vinthony/pso-cnn/blob/master/mnist_simpleNet.py

**لازم‌به ذکر اســـت که بخش مقدمه و چکیده تقریبا ترجمه بخش‌های چکیده و مقدمه مقاله می‌باشد.**

**و همچنین کد این پروژه با همکاری سایا هاشمیان زده شده است.**