Optimizer

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION https://arxiv.org/pdf/1412.6980

Algorithm:

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Novelty:

In comparison with adagrad and RMSProp, Adam introduces the concept of second moment of the gradient to stabilize the gradient update. Theoretically and empirically, it is successful at smoothing the overshooting problem.

Explanation:

First Moment:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
&= (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i
\end{aligned}
$$

Let $l = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} (g_i - g_t)$, then

$$
\begin{aligned}
E[m_t] &= E[g_t](1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} + l \\
&= E[g_t](1 - \beta_1)\left(\frac{1 - \beta_1^t}{1 - \beta_1}\right) + l \quad \text{(Geometric Series)} \\
&= E[g_t](1 - \beta_1^t) + l
\end{aligned}
$$

If g is stationary, l = 0. Even if g is non-stationary, the author argued that l is relatively small. Due to the exponential decay factor β_1, the difference of the early timestep is

almost negligible.

So, by dividing m_t by $1-\beta_1^t$, we can get the unbiased estimate of the first moment $E[\hat{m}_t] \approx E[g_t]$.

Second Moment:

$$
\begin{aligned}
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
&= (1-\beta_2) \sum_{i=0}^{t} \beta_2^{t-i} g_i^2
\end{aligned}
$$

$$
\begin{aligned}
E[v_t] &= E[g_t^2](1-\beta_2) \sum_{i=0}^{t} \beta_2^{t-i} + l \\
&= E[g_t^2](1-\beta_2)\left(\frac{1-\beta_2^t}{1-\beta_2}\right) + l \quad \text{(Geometric Series)} \\
&= E[g_t^2](1-\beta_2^t) + l
\end{aligned}
$$

Similar to the first moment, we treat l as a negligible term.

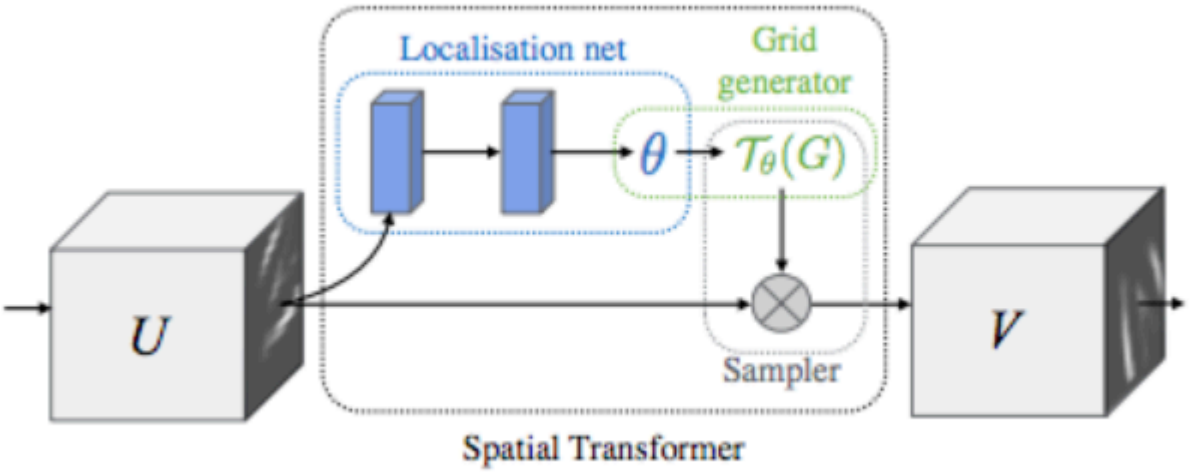
So, by dividing v_t by $1-\beta_2^t$, we can get the unbiased estimate of the second moment $E[\hat{v}_t] \approx E[g_t^2]$.

Why $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$ works?

- It makes the gradient update $\alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$ sit at a bounded region (estimated) [-1, 1] because $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \approx \frac{E[g_t]}{\sqrt{E[g_t^2]}} \lessapprox 1$
- Second moment represents the variance of the gradient. When the variance is large, it means it is uncertain about the true direction of the gradient to the local optimum. So, we want it to take a smaller step in case of overshooting from the local minima.

Convolutional Neural Networks

Spatial Transformer Networks https://arxiv.org/pdf/1506.02025



Novelty:

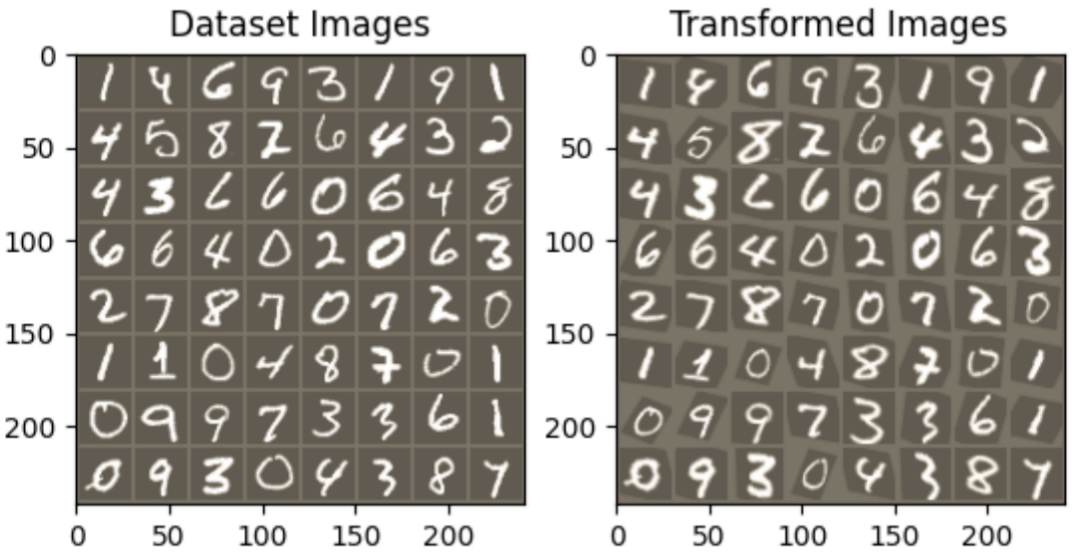
- It acts as a pre-processing step to the CNN to make it more robust to the spatial transformation.
- The spatial transformer network (STN) is often used at the first layer of the network. It transforms the input image to a canonical form, and then feed into the CNN.
- Affine transformation is used to transform the input image. And the transformation matrix is learned by the network.

Architecture:

The spatial transformer network (STN) consists of three parts:

1. Localization Network:
    - It is a network (can be a CNN, FCN, RNN, etc.) that outputs the parameters for the affine transformation matrix.
2. Grid Generator (Affine Transformation):
    - Apply the affine transformation to the input image.
3. Grid Sampler:
    - It samples the input image using the grid of coordinates.
    - A common sampling method is bilinear interpolation.

Image Comparion between original and STN:



Based on the above image, we can see that the STN is able to transform the input image to a "canonical form". In which, the images of the same digits are more aligned in terms of rotation. We can conclude that STN helps to mitigate the problem of the CNN being sensitive to the spatial transformation like rotation.

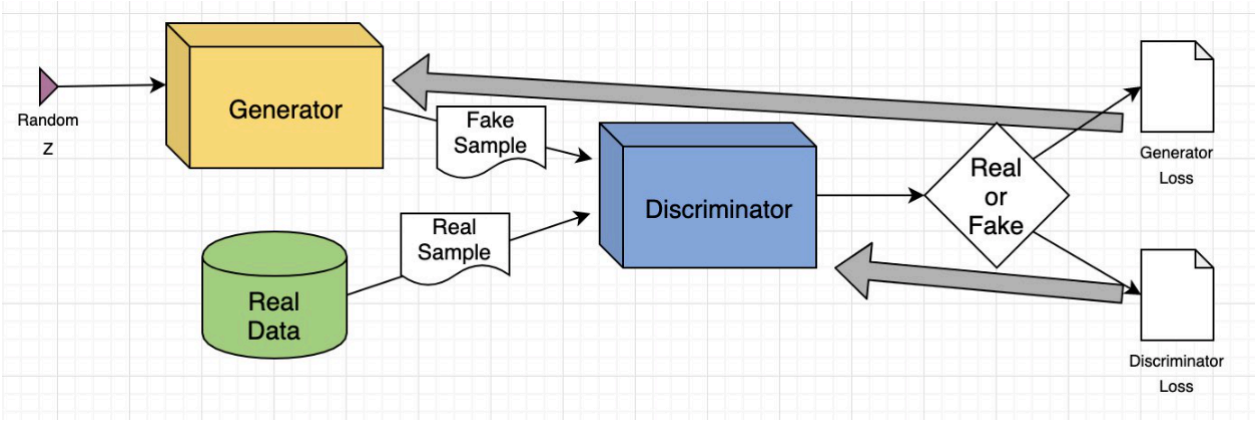
Code:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

        # Spatial transformer localization-network
        self.localization = nn.Sequential(
            nn.Conv2d(1, 8, kernel_size=7),
            nn.MaxPool2d(2, stride=2),
            nn.ReLU(True),
            nn.Conv2d(8, 10, kernel_size=5),
            nn.MaxPool2d(2, stride=2),
            nn.ReLU(True)
        )

        # Regressor for the 3 * 2 affine matrix
        self.fc_loc = nn.Sequential(
```

```
            nn.Linear(10 * 3 * 3, 32),
            nn.ReLU(True),
            nn.Linear(32, 3 * 2)
        )

        # Initialize the weights/bias with identity transformation
        self.fc_loc[2].weight.data.zero_()
        self.fc_loc[2].bias.data.copy_(torch.tensor([1, 0, 0, 0, 1, 0],
dtype=torch.float))

    # Spatial transformer network forward function
    def stn(self, x):
        xs = self.localization(x)
        xs = xs.view(-1, 10 * 3 * 3)
        theta = self.fc_loc(xs)
        theta = theta.view(-1, 2, 3)

        grid = F.affine_grid(theta, x.size())
        x = F.grid_sample(x, grid)

        return x

    def forward(self, x):
        # transform the input
        x = self.stn(x)

        # Perform the usual forward pass
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)


model = Net().to(device)
```

Generative Model

Generative Adversarial Networks https://arxiv.org/pdf/1406.2661



Novelty

- Adversarial Training Paradigm: Before GAN, the generative models often relied on the maximum likelihood estimation. GAN introduced a discrimitive network as the adversary to the generative network, which drives the generator to generate the samples that are close to the real samples.
- Implicit density estimation: The generative model is not trained with the explicit density estimation. Instead, it is trained with purely the log-likelihood loss.

Architecture:

Components:

- $G(z)$: The generator network. It takes the noise z as the input and generates the sample x.
- $p_z(z)$: The prior distribution of the noise.
- $p_d(x)$: The real data distribution.
- $D(x)$: The discriminator network. It takes the sample x as the input and outputs the probability of the sample being real.

Objective:

$$\min_G \max_D V(D,G) = E_{x\sim p_d}[\log D(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))]$$

Intuition:

- The discriminator is trying to maximize the expected value of $logD(x)$ for x sampled from the real data distribution $p_d(x)$. And it is trying to minimize the expected value of $log(D(G(z)))$ for z sampled from the prior distribution (noise) $p_z(z)$ and image

generated by the generator function $G(z)$, which is then re-written as maximizing the expected value of $log(1 - D(G(z)))$ in the objective function.

- The generator is trying to maximize the expected value of $log(D(G(z)))$ for the fake samples generated by the generator $G(z)$. In other words, it is trying to generate the samples that can fool the discriminator, and it is written as minimizing expected value of $log(1 - D(G(z)))$ in the objective function.

Theorem:

Theorem 1: For any generator G, the optimal discriminator D is

$$D_G^*(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

Proof:

$$\begin{aligned} V(D, G) &= E_{x \sim p_d}[\log D(x)] + E_{z \sim p_z}[\log(1 - D(G(z)))] \\ &= \int_x p_d(x) \log D(x) dx + \int_z p_z(z) \log(1 - D(G(z))) dz \\ &= \int_x p_d(x) \log D(x) dx + \int_x p_g(x) \log(1 - D(x)) dx \\ &= \int_x (p_d(x) \log D(x) + p_g(x) \log(1 - D(x))) dx \end{aligned}$$

Consider the function $f(y) = a \log y + b \log(1 - y)$, the function achieves the maximum when $y = \frac{a}{a+b}$ (by taking the derivative and set it to 0).

Therefore, the optimal discriminator is $D^*(x) = \frac{p_d(x)}{p_d(x)+p_g(x)}$

Theorem 2: Give an optimal discriminator D^*, the optimal generator G^*, which aims to minimize the objective function, is obtained when $p_g = p_d$

Proof:

$$
\begin{aligned}
V(D^*, G) &= E_{x\sim p_d}[\log D^*(x)] + E_{x\sim p_g}[\log(1 - D^*(x))] \\
&= E_{x\sim p_d}[\log(\frac{p_d(x)}{p_d(x)+p_g(x)})] + E_{x\sim p_g}[\log(\frac{p_g(x)}{p_d(x)+p_g(x)})] \\
&= E_{x\sim p_d}[\log(\frac{p_d(x)}{p_d(x)+p_g(x)})] + E_{x\sim p_g}[\log(\frac{p_g(x)}{p_d(x)+p_g(x)})] \\
&= \int_x (p_d(x)\log(\frac{p_d(x)}{p_d(x)+p_g(x)}) + p_g(x)\log(\frac{p_g(x)}{p_d(x)+p_g(x)}))dx
\end{aligned}
$$

Let a_i and b_i to denote the probability of x_i being sampled from $p_d(x)$ and $p_g(x)$ respectively.

We can rewrite the integral as the summation of the probabilities in the discrete case:

$$
\sum_{i=1}^{n} [a_i \log(\frac{a_i}{a_i + b_i}) + b_i \log(\frac{b_i}{a_i + b_i})]
$$

And we know that the summation of probability distribution function is 1. Then we get the following constraint:

$$
\sum_{i=1}^{n} a_i = 1 \quad \text{and} \quad \sum_{i=1}^{n} b_i = 1
$$

Then, we can formulate the Lagrangian multiplier function:

$$
\begin{aligned}
L(a, b, \lambda, \mu) &= \sum_{i=1}^{n} [a_i \log(\frac{a_i}{a_i + b_i}) + b_i \log(\frac{b_i}{a_i + b_i})] - \lambda([\sum_{i=1}^{n} a_i] - 1) - \mu([\sum_{i=1}^{n} b_i] - 1) \\
&= \sum_{i=1}^{n} [a_i \log(\frac{a_i}{a_i + b_i}) - \lambda a_i + b_i \log(\frac{b_i}{a_i + b_i}) - \mu b_i] + \lambda + \mu
\end{aligned}
$$

Let $F_i = a_i \log(\frac{a_i}{a_i+b_i}) - \lambda a_i + b_i \log(\frac{b_i}{a_i+b_i}) - \mu b_i$.

We want to have $\frac{\partial F_i}{\partial a_i} = 0 \quad \text{and} \quad \frac{\partial F_i}{\partial b_i} = 0$ for every i, to make sure F is at the minimum.

By taking the partial derivative of F_i with respect to a_i and b_i, we get:

$$
\begin{aligned}
\frac{\partial F_i}{\partial a_i} &= \log \frac{a_i}{a_i + b_i} - \lambda = 0 \implies \lambda = \log \frac{a_i}{a_i + b_i} \\
\frac{\partial F_i}{\partial b_i} &= \log \frac{b_i}{a_i + b_i} - \mu = 0 \implies \mu = \log \frac{b_i}{a_i + b_i}
\end{aligned}
$$

We then have $\frac{a_i}{a_i+b_i} = e^{\lambda}$ and $\frac{b_i}{a_i+b_i} = e^{\mu}$,

and $\frac{a_i}{a_i+b_i} + \frac{b_i}{a_i+b_i} = 1 = e^{\lambda} + e^{\mu}$.

If we let $e^{\lambda} = e^{\mu} = \frac{1}{2}$, then we have $a_i = b_i$.

Thus, we have $\log \frac{a_i}{a_i+b_i} = \log \frac{b_i}{a_i+b_i} \implies a_i = b_i$.

Therefore, if we have $a_i = b_i$ for all x_i, then the objective function $V(D^*, G^*)$ is minimized with value of $-2log2$