深度学习技术与应用（5）

Deep Learning: Techniques and Applications (5)

Ge Li

Peking University

本节内容

- 神经网络训练优化方法
- 卷积神经网络 之一

牛顿法

设待最小的函数为：$f(x)$，即：求取 x 使 $f(x)$ 最小：

$$\min_x f(x).$$

【带 Peano 余项的 Taylor 公式】设 $f(x)$ 在 x_k 处有 n 阶导数，则存在 x_k 的一个邻域，对于该邻域中的任一点 x，成立：

$$\begin{aligned} f(x) = & \\ & f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2!}(x - x_k)^2 + \ldots \\ & + \frac{f^{(n)}(x_k)}{n!}(x - x_k)^n + o((x - x_n)^n) \end{aligned}$$

余项 $o((x - x_n)^n)$ 为高阶无穷小。

牛顿法

设 x_k 为当前的极小值估计值，则在 x_k 邻域中的任一点 x 点做二阶泰勒展开，得到：

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2!}(x - x_k)^2$$

若在 x_k 邻域中的任一点 x 处能够取得最小值，即：$f'(x) = 0$ 于是得：

$$f'(x_k) + f''(x_k)(x - x_k) = 0$$

于是得：

$$x = x_k - \frac{f'(x_k)}{f''(x_k)}$$

得到 x 的迭代更新规律为：

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

牛顿法

将上式中的 x 推广至 N 维的情形（下文中 x 表示 N 向量），设 x_k 为第 k 步时，与目标函数极小值的估计值所对应的 x，则在 x_k 处目标函数 $f(x)$ 的二阶泰勒展开式为：

$$f(x) = f(x_k) + \nabla f(x_k) \cdot (x - x_k) + \frac{1}{2} \cdot (x - x_k)^T \cdot \nabla^2 f(x_k) \cdot (x - x_k)$$

其中，∇f 为 f 的梯度向量，$\nabla^2 f$ 为 Hessian 矩阵：

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix} \text{记为:} g, \quad \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix} \text{记为:} H$$

注：仅在上式中，x_i 表示 N 维向量 x 的第 i 维；

牛顿法

所以上式变为：

$$f(x) = f(x_k) + g_k(x - x_k) + \frac{1}{2} \cdot (x - x_k)^T \cdot H_k \cdot (x - x_k)$$

其中，g_k 表示 $\nabla f(x_k)$，H_k 表示 $\nabla^2 f(x_k)$；

若在 x_k 邻域中的任一点 x 处能够取得最小值，即：$f'(x) = 0$ 于是，对上式对 x 求导数，得：

$$g_k + H_k \cdot (x - x_k) = 0$$

若H_k非奇异，则得到：

$$x = x_k - H_k^{-1} \cdot g_k$$

得到 x 的迭代更新规律为：

$$x_{k+1} = x_k - H_k^{-1} \cdot g_k$$

$d_k = -H_k^{-1} \cdot g_k$ 被称为“牛顿方向”.

阻尼牛顿法

- 当目标函数为二次函数时，Hessian 矩阵退化成一个常数矩阵，从任一初始点出发只需要一步迭代即可达到 f(x) 的极小点 x*，这就是牛顿法的“二次收敛性”.
- 然而，牛顿法的迭代公式中，每步迭代都是固定长度，因此，并不一定能够收敛。对于非二次型目标函数，甚至可能会使函数值上升，导致计算失败。
- 为了克服这个问题，人们给牛顿法增加一个“步长因子”λ_k，且该步长因子满足：

$$\lambda_k = \underset{\lambda \in \mathcal{R}}{\operatorname{argmin}} f(x_k + \lambda d_k)$$

- 其中，难点在于计算 $d_k = -H_k^{-1} \cdot g_k$.
- 迭代终止条件可以设定为：$\| g_k \| < \epsilon$

阻尼牛顿法

1. Initialize: $k = 0; x_k =$ random number; $\epsilon > 0$;
2. Calculate: g_k and H_k;
3. If $\| g_k \| < \epsilon$, then return x_k;
   else calculate $d_k = -H_k^{-1} \cdot g_k$;
4. Calculate: $\lambda_k = \text{argmin}_{\lambda \in \mathcal{R}} f(x_k + \lambda d_k)$
5. Iterate: $k+ = 1$; goto step 2;

- 优点：利用目标函数的二阶导数，不但考虑了一阶方向性，且利用了二阶导数对变化趋势的预测能力；
- 缺点：
  - 对目标函数的要求很高——要求二阶可导，且 Hessian 矩阵必须为正定矩阵；
  - 计算量大，需要求解 Hessian 矩阵的逆，计算复杂度为 $O(n^3)$ 级；

拟牛顿法的基本思想

基本思想：针对牛顿法存在的上述缺点，通过构造 Hessian 矩阵的近似矩阵的方法，对目标函数进行优化。希望该矩阵具有以下性质：

- 该矩阵应该能够具有逼近（模拟）目标函数二阶导数的特性；
- 该矩阵不必求取 Hessian 矩阵；
- 更不必求取 Hessian 矩阵的逆矩阵；
- 也许，该矩阵应该具备递推性质，即由 k 情况下的矩阵值，可以递推出 $k+1$ 情况下的矩阵值。即：

$$\hat{H}_{k+1} = \hat{H}_k + \Delta\hat{H}_k$$

如果存在一个这样的矩阵 $\hat{H}_k$，那么原始牛顿法的计算过程，将改为：

拟牛顿法的计算过程

如上所述，设 $\hat{H}$ 为 Hessian 矩阵 H^{-1} 的同阶近似矩阵；

1. Initialize: $k = 0; x_k$ as random number; $\epsilon > 0$;
   $\hat{H}_0$ as positive definite and symmetric matrix;
2. Calculate: g_k;
3. If $\| g_k \| < \epsilon$, then return x_k;
4. Calculate $g_{k+1} = \nabla f(x_k)$, $\hat{H}_{k+1} = \hat{H}_k + \Delta\hat{H}_k$;
5. Calculate $d_{k+1} = -\hat{H}_{k+1} \cdot g_{k+1}$;
6. Calculate: $\lambda_{k+1} = \text{argmin}_{\lambda \in \mathcal{R}} f(x_k + \lambda d_{k+1})$;
7. Calculate: $x_{k+1} = x_k + \lambda_k d_{k+1}$;
8. Iterate: $k+ = 1$; goto step 2;

拟牛顿法的构造条件

在上述方法中，只有 $\hat{H}$ 未知，因此，现在的关键是：如何构造$\hat{H}$。
先来分析一下 $\hat{H}_{k+1}$ 应该满足的条件：

1. 条件 1：由上文可知，该矩阵应该具备递推性质，即由 k 情况下的矩阵值，可以递推出 $k+1$ 情况下的矩阵值。即：

$$\hat{H}_{k+1} = \hat{H}_k + \Delta\hat{H}_k$$

2. 条件 2：由 $x_{k+1} = x_k + \lambda_k d_{k+1}$ 且 $d_{k+1} = -\hat{H}_{k+1} \cdot g_{k+1}$ 可知，为了确保 x 的搜索方向稳定，最好保持$\hat{H}_{k+1}$为正定矩阵；
3. 条件 3　在数学上，必须满足“拟牛顿条件”：

拟牛顿条件

设当前要优化的目标函数为 $f(x)$，设已完成的迭代步骤为 k；需要推知 $k+1$ 情况下的计算方法；

下面开始推导：

1. 首先将 $f(x)$ 在 x_{k+1} 点展开：

$$\begin{aligned} f(x) =& f(x_{k+1}) + \nabla f(x_{k+1}) \cdot (x - x_{k+1}) \\ &+ \frac{1}{2} \cdot (x - x_{k+1})^T \cdot \nabla^2 f(x_{k+1}) \cdot (x - x_{k+1}) \end{aligned}$$

2. 对两边关于 x 求导：

$$\nabla f(x) = \nabla f(x_{k+1}) + H_{k+1} \cdot (x - x_{k+1})$$

3. 在上式中，取 x 为当前已经迭代到的值 x_k，得到：

$$\nabla f(x_k) - \nabla f(x_{k+1}) = H_{k+1} \cdot (x_k - x_{k+1})$$

即：

$$g_{k+1} - g_k = H_{k+1} \cdot (x_{k+1} - x_k)$$

拟牛顿条件

另一种推导方法是，直接利用微分中值定理，直接写出：

$$\nabla f(x_k) - \nabla f(x_{k+1}) = H_{k+1} \cdot (x_k - x_{k+1})$$

也得到：

$$g_{k+1} - g_k = H_{k+1} \cdot (x_{k+1} - x_k)$$

为简化表示，设 $s_k = x_{k+1} - x_k, y_k = g_{k+1} - g_k$，则得到拟牛顿条件：

$$y_k = H_{k+1} \cdot s_k \quad \text{即：} s_k = H_{k+1}^{-1} \cdot y_k$$

设 B_{k+1} 为 Hessian 矩阵 H_{k+1} 的近似，设 $\hat{H}_{k+1}$ 为 H_{k+1}^{-1} 的近似矩阵，则：得到拟牛顿条件的惯用表示：

$$y_k = B_{k+1} \cdot s_k \quad \text{即：} s_k = \hat{H}_{k+1} \cdot y_k$$

拟牛顿法

观察拟牛顿条件：

$$y_k = B_{k+1} \cdot s_k \quad \text{即：} s_k = \hat{H}_{k+1} \cdot y_k$$

可知，在 y_k 与 s_k 已知的条件下：

- 若保持 B_{k+1} 或 $\hat{H}_{k+1}$ 为对称矩阵，则其中有 $\frac{n^2+n}{2}$ 个未知数，而方程个数只有 k 个，所以满足条件的 B_{k+1} 或 $\hat{H}_{k+1}$ 有无穷多个；所以，一定有多种满足拟牛顿条件的近似方法；
- 这些方法要么是对 B_{k+1} 进行迭代修正，要么是对 $\hat{H}_{k+1}$ 进行迭代修正。

那么，如何来选择构造 Hessian 矩阵的近似矩阵？

拟牛顿法之 DFP 方法

由数学家 William C. Davidon 在 1959 年最早提出，后经 Roger Fletcher、Michael J.D. Powell 在 1963 年完善，1991 年发表。

基本原理是，在前文分析的三个条件的限制下，对 $\hat{H}_k$ 进行秩 2 修正：

$$\begin{aligned}\hat{H}_{k+1} &= \hat{H}_k + \Delta\hat{H}_k \\ &= \hat{H}_k + \alpha_k u_k u_k^T + \beta_k v_k v_k^T\end{aligned}$$

其中，$\alpha_k, \beta_k \in \mathcal{R}; u_k, v_k \in \mathcal{R}^n$；
由于 $u_k u_k^T$ 与 $v_k v_k^T$ 的秩均为 1，所以被称为“对 $\hat{H}_k$ 进行秩 2 修正”；

接下来，关键看如何求解 $\alpha_k, \beta_k, u_k, v_k$：

拟牛顿法之 DFP 方法

DFP 秩 2 修正的好处：

1. 在 u_k 与 v_k 不为零，α_k, β_k 不为负的前提下，保证了 $\hat{H}_{k+1}$ 矩阵的正定特性，确保了对 x 稳定的调整方向；

   以 u_k 为例：$z^T u_k u_k^T z = (u_k^T z)^T u_k^T z = \| u_k^T z \| \geq 0; \Rightarrow$ 正定.

2. $u_k u_k^T$ 与 $v_k v_k^T$ 均为对称矩阵，从而延续了 $\hat{H}_{k+1}$ 的对称性；

3. 当然，接下来还得让它必须满足“拟牛顿条件”。

拟牛顿法之 DFP 方法

由拟牛顿条件：

$$s_k = \hat{H}_{k+1} \cdot y_k$$

代入 $\hat{H}_{k+1}$ 的秩 2 修正，得：

$$\begin{aligned} s_k &= (\hat{H}_k + \alpha_k u_k u_k^T + \beta_k v_k v_k^T) y_k \\ &= \hat{H}_k y_k + \alpha_k u_k u_k^T y_k + \beta_k v_k v_k^T y_k \end{aligned}$$

即：

$$s_k - \hat{H}_k y_k = \alpha_k u_k (u_k^T y_k) + \beta_k v_k (v_k^T y_k)$$

也就是说，要满足拟牛顿条件，就是要使这个式子成立。

拟牛顿法之 DFP 方法

使这个式子成立的办法很多，DFP 方法中选择如下条件：

$$s_k = \alpha_k u_k (u_k^T y_k)$$
$$\hat{H}_k y_k = -\beta_k v_k (v_k^T y_k)$$

进而，可以再选择：

$$\text{选择:} s_k = u_k, \text{ 得到：} \alpha_k (u_k^T y_k) = 1$$
$$\text{选择:} \hat{H}_k y_k = v_k, \text{ 得到：} \beta_k (v_k^T y_k) = -1$$

进而得到：

$$u_k = s_k, \quad \alpha_k = \frac{1}{(u_k^T y_k)}$$
$$v_k = \hat{H}_k y_k, \quad \beta_k = -\frac{1}{(v_k^T y_k)} = -\frac{1}{(y_k^T \hat{H}_k y_k)}$$

从 DFP 到 BFGS 方法

最终得到，DFP 方法近似矩阵 $\hat{H}_k$ 的迭代更新公式：

$$\hat{H}_{k+1} = \hat{H}_k + \frac{s_k^T s_k}{(u_k^T y_k)} - \frac{\hat{H}_k y_k y_k^T \hat{H}_k}{(y_k^T \hat{H}_k y_k)}$$

类似的，利用拟牛顿公式的另一个描述 $y_k = B_{k+1} \cdot s_k$，我们也可以对 B_{k+1} 进行迭代修正——BFGS 方法

- 以数学家 Broyden, Fletcher, Goldfarb, Shanno 的名字命名；
- 比 DFP 方法具有更好的性能，是当前常用的优化算法；

拟牛顿法之 BFGS 方法

对 B_{k+1} 进行秩 2 修正：

$$B_{k+1} = B_k + \alpha_k u_k u_k^T + \beta_k v_k v_k^T$$

由拟牛顿条件 $y_k = B_{k+1} \cdot s_k$，得：

$$y_k = B_k s_k + \alpha_k u_k u_k^T s_k + \beta_k v_k v_k^T s_k$$

选取：

$$y_k = \alpha_k u_k u_k^T s_k = \alpha_k u_k (u_k^T s_k)$$
$$B_k s_k = -\beta_k v_k v_k^T s_k = -\beta_k v_k (v_k^T s_k)$$

再选取：

$$u_k = y_k, \text{ 得到：} \alpha_k = \frac{1}{u_k^T s_k}$$

$$v_k = B_k s_k, \text{ 得到：} -\beta_k = -\frac{1}{v_k^T s_k} = -\frac{1}{s_k^T B_k s_k}$$

拟牛顿法之 BFGS 方法

从而得到，BFGS 方法近似矩阵 B_k 的迭代更新公式：

$$B_{k+1} = B_k + \frac{y_k y_k^T}{u_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

注意：$B_k + 1$ 是对 H_{k+1} 的近似，在牛顿法中计算 $d_{k+1} = -H_{k+1}^{-1} \cdot g_{k+1}$ 的公式相应变为 $d_{k+1} = -B_{k+1}^{-1} \cdot g_{k+1}$，可见，还需要计算 B_{k+1}^{-1}.

这很不爽，因此，改写公式：

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}$$

可见，BFGS 方法具有更好的计算性能，其时间复杂度为 $O(n^2)$ 级；
然而，其空间复杂度仍值得关注：B_{k+1} 为 $N \times N$ 方阵，存储量很大；

拟牛顿法之 L-BFGS 方法

L-BFGS (Limited-storage BFGS) 基本思想：

1. 在计算过程中，不选择存储 B_k，而选择利用 $\{s_i\}\{y_i\}$ 的历史序列计算得到；
2. 计算中，抛弃 m 步迭代之前的 $\{s_i\}\{y_i\}$ 序列，而仅利用最近的 m 个存储序列来估算；

对照公式：

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}$$

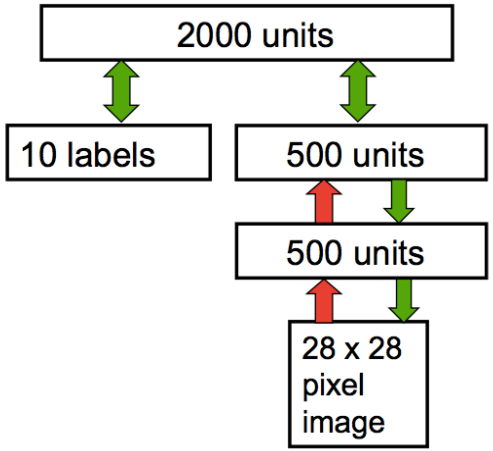
设：$\rho_k = \frac{1}{y_k^T s_k}, U_k = I - \rho_k y_k s_k^T, M_k = B_k^{-1}$，则原公式写为：

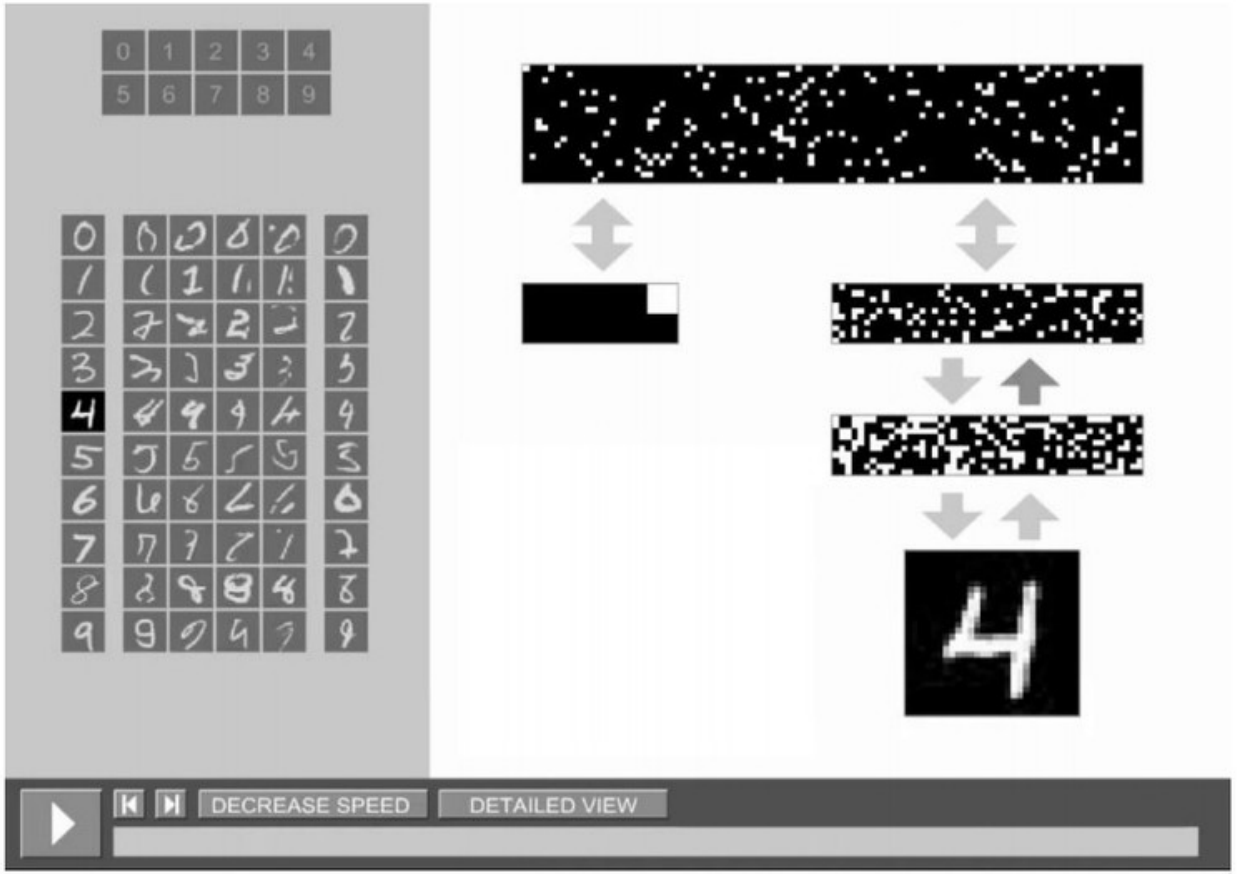
$$M_{k+1} = U_k^T M_k U_k + \rho_k s_k s_k^T$$

可见，计算 M_{k+1} 只需要对第一项进行 m 步迭代计算；

Problems for Fully Connected Neural Networks

- Hinton in 2006





Problems for Fully Connected Neural Networks

- Fully Connect Networks
  - With small images, it was computationally feasible to learn features on the entire image.
    - 28x28 images for the MNIST dataset
  - With larger images, learning features that span the entire image is very computationally expensive.
    - With 96x96 images, you would have about 10^4 input units, and assuming you want to learn 100 features, you would have on the order of 10^6 parameters to learn.
    - The feedforward and backpropagation computations would also be about 10^2 times slower, compared to 28x28 images.

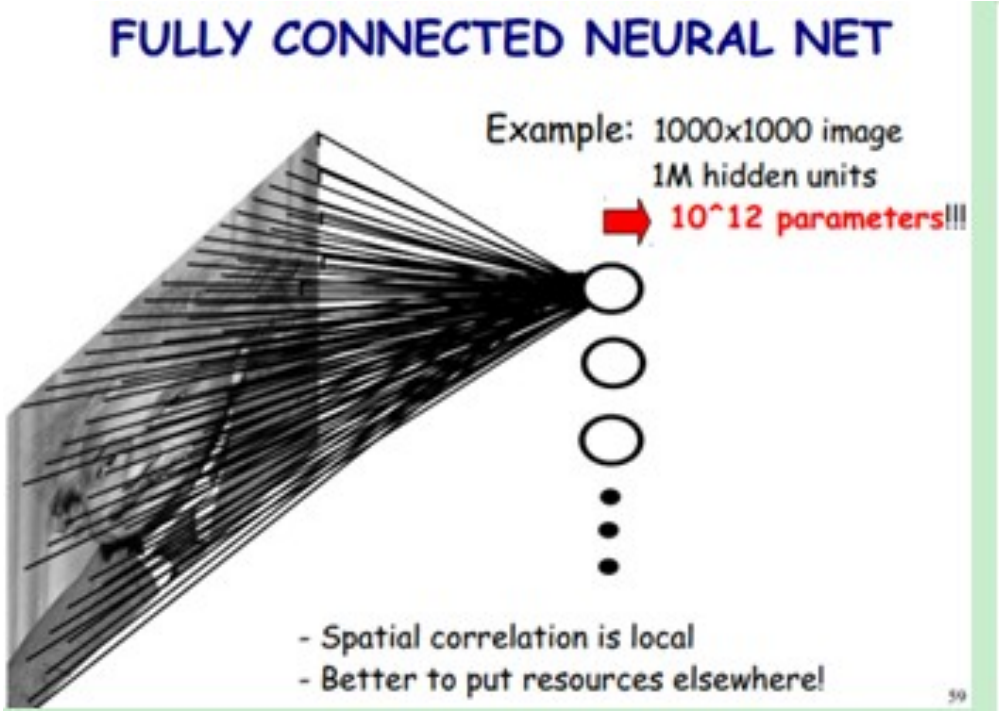
Locally Connected Networks

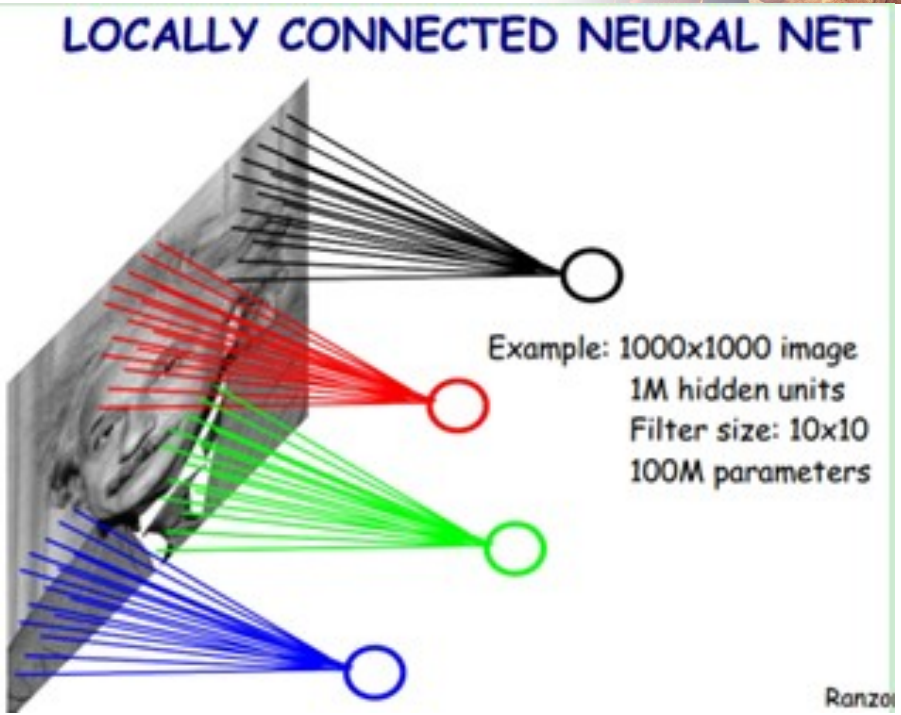
- One simple solution:
  - to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units.
  - Specifically, **each hidden unit will connect to only a small contiguous region of pixels in the input**.
    - there is often also a natural way to select "contiguous groups" of input units to connect to a single hidden unit as well;
  - This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology.
    - Specifically, neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimulit in a certain location).

Locally Connected Networks

- Natural images have the property of being "**stationary**"
  - meaning that the statistics of one part of the image are the same as any other part.
- So, **the features that we learn at one part of the image can also be applied to other parts of the image**, and we can use the same features at all locations.
  - More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image.
  - Specifically, we can take the learned 8x8 features and **convolve** them with the larger image, thus obtaining a different feature activation value at each location in the image.

Locally Connected Networks



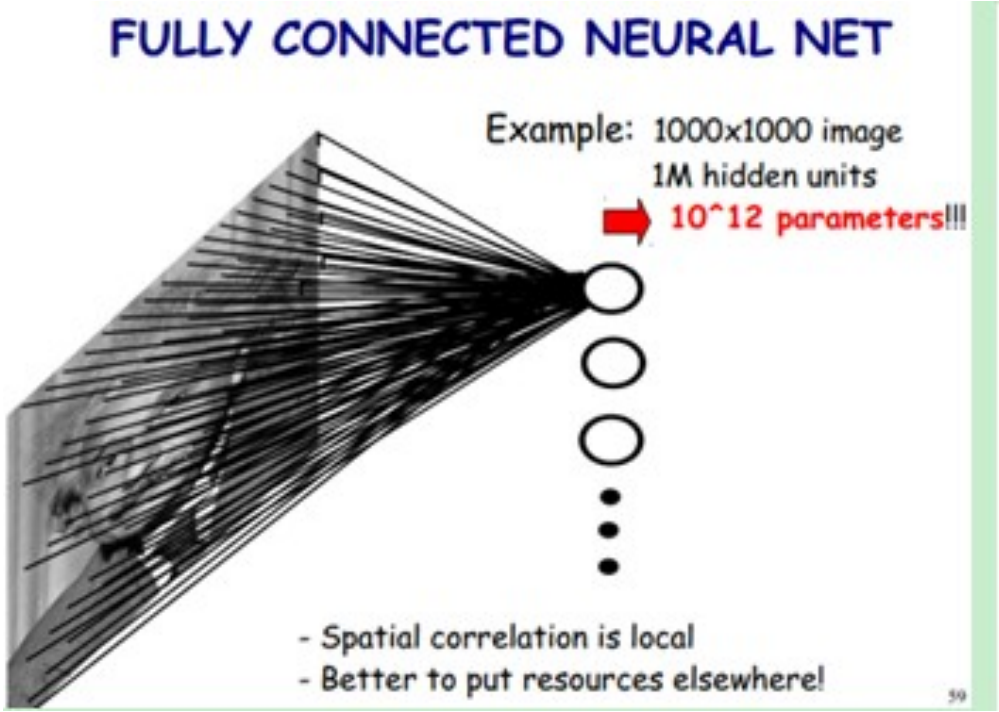


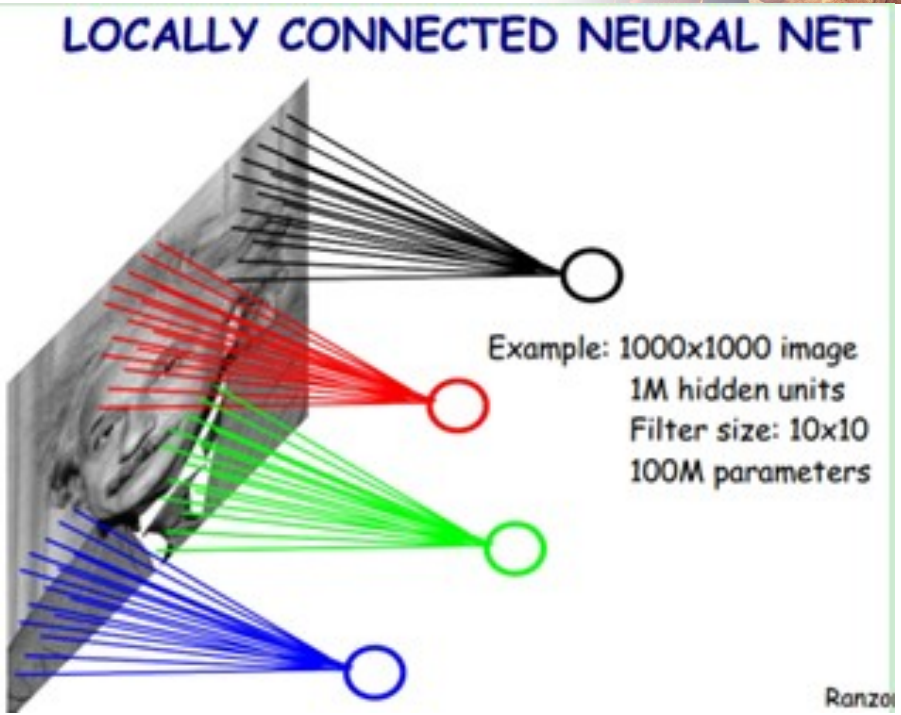
Suppose there are 1M hidden units:

Left: $1000 \times 1000 \times 1M = 10^{12}$

Right: $10 \times 10 \times 1M = 10^{8}$

Weights Sharing





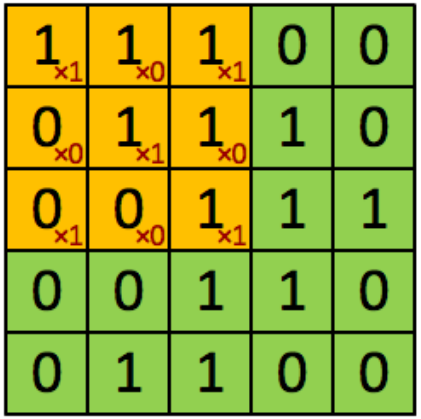
Suppose there are 1M hidden units:

Left: $1000\times1000\times1M = 10^{12}$

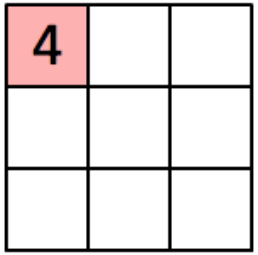
Right: $10\times10\times1 = 10^{2}$

Convolutions

- Suppose you want to learned 9 features from a 5x5 image.
  - With Fully Connected Neural Networks:
    $5 \times 5 \times 9 = 225$
  - With Locally Connected Neural Networks:
    $3 \times 3 \times 9 = 81$
  - With Weights Sharing:
    $3 \times 3 \times 1 = 9$

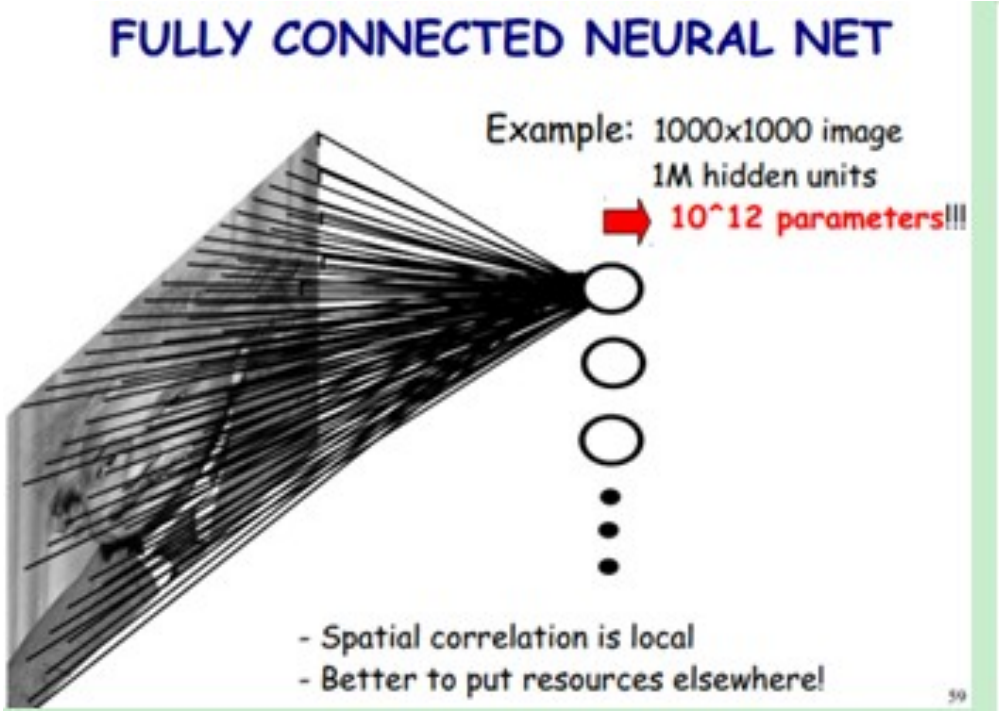


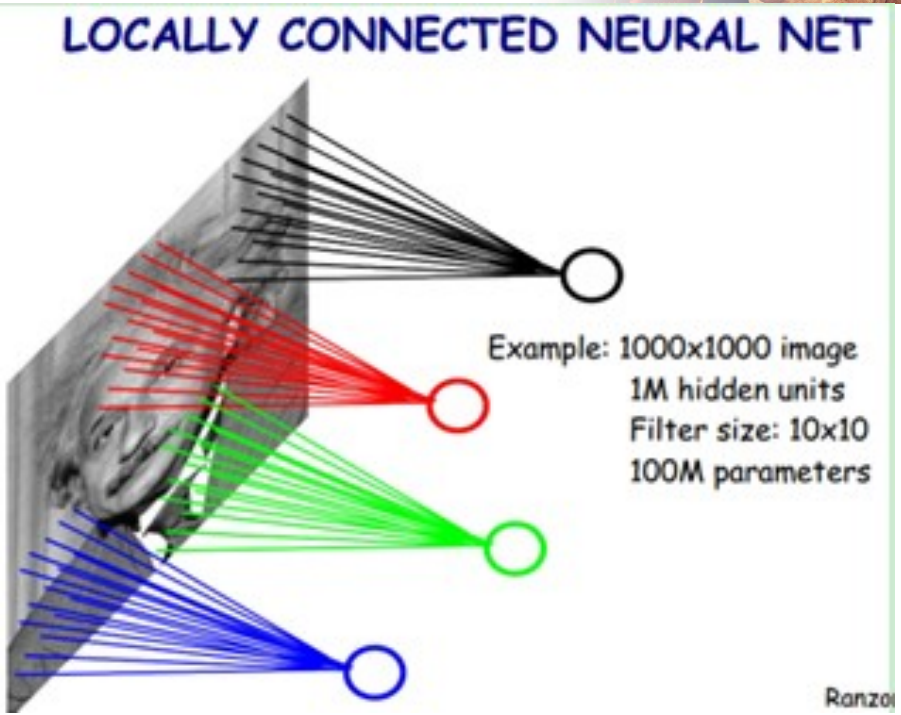
Image



Convolved Feature

Is 1 Hidden Unit Enough ? No !



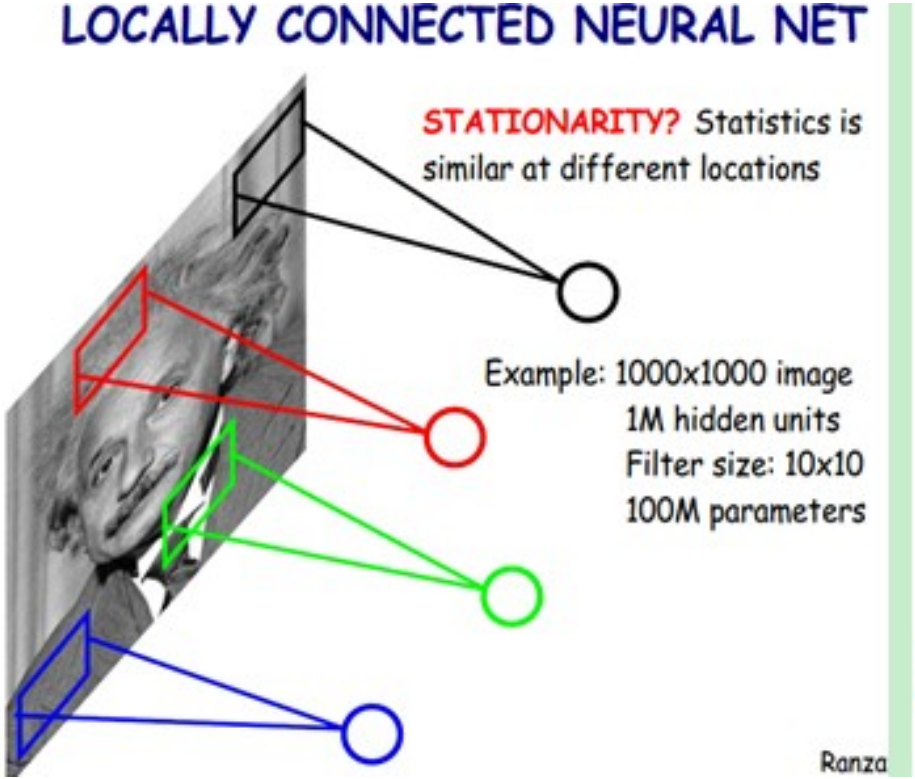


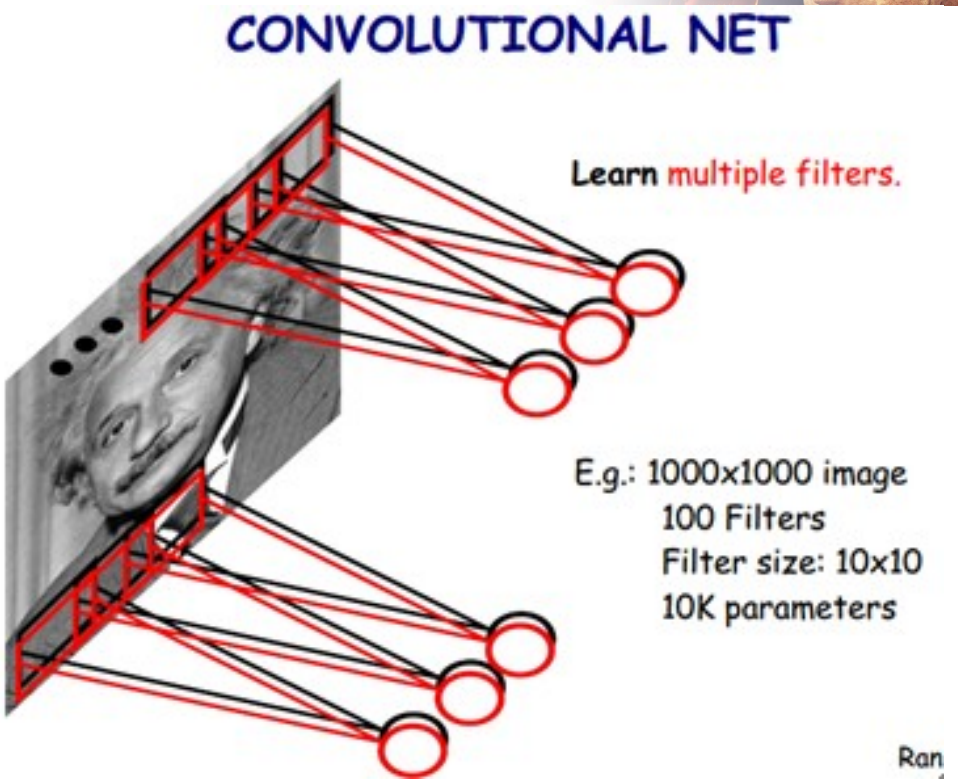
Suppose there are 1M hidden units:

Left: $1000\times1000\times1M = 10^{12}$

Right: $10\times10\times1 = 10^{2}$

Multiple Kernels Convolution



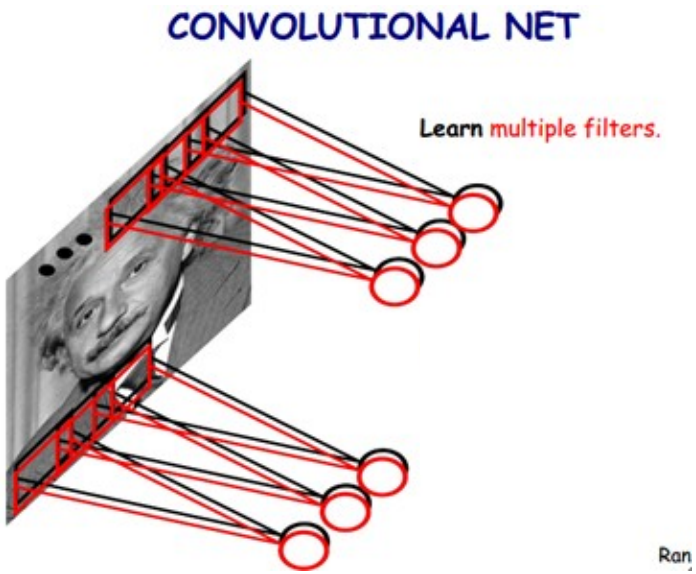


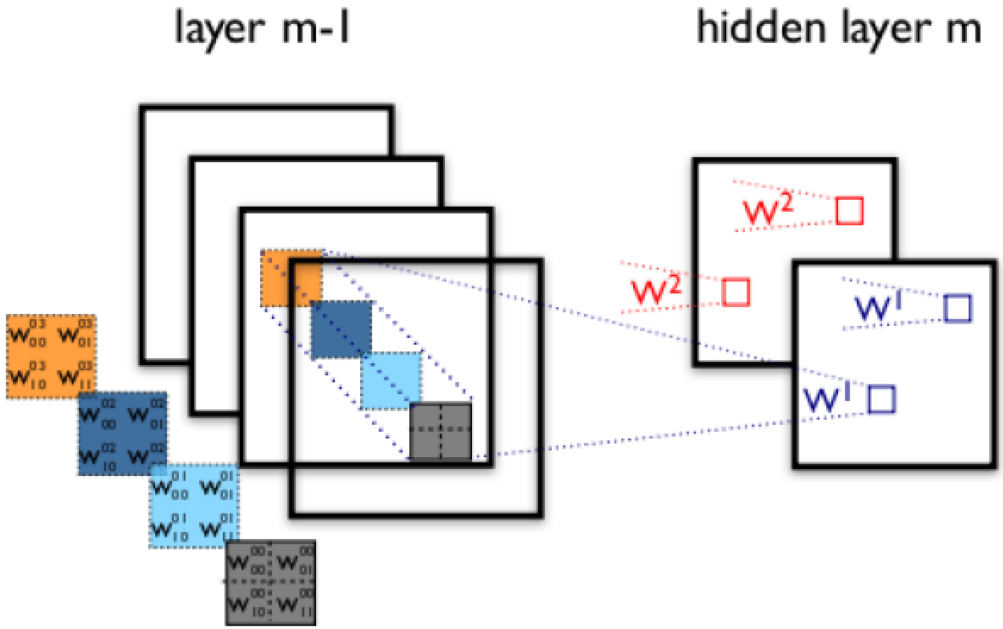
Left: $10\times10\times1M(\text{特征数}) = 10^8$

Right: $10\times10\times100$（特征数） $= 10^4$

Multiple Kernels Convolution

- 对于一张100×100的图片
  - One Kernel: $10\times10\times100 = 10^4$　　4 Kernels: $10^4\times4$
  - 2nd-Convolution: $2\times2\times4\times2 = 32$





Pooling

- If we use all the extracted features with a classifier, this can be computationally challenging.
  - images of size 96x96 pixels;
  - suppose we want to learn 400 features over 8x8 inputs;

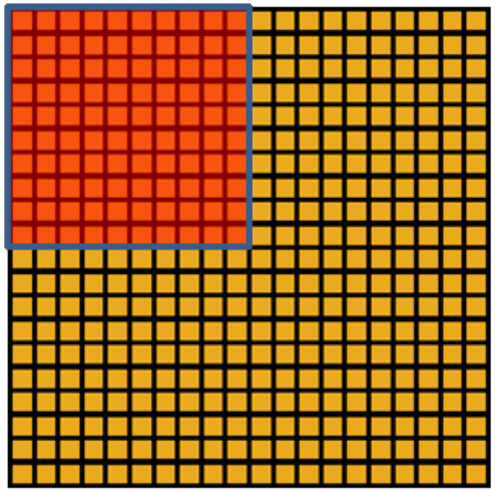
  Then we have to compute

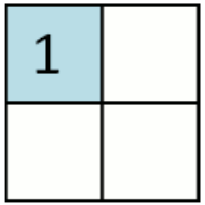
  $$(96-8+1)*(96-8+1)*400=3,168,400 \text{ features}$$

  Learning a classifier with inputs having 3+ million features can be unwieldy, and can also be prone to over-fitting.
- Pooling: to describe a large image, we can aggregate statistics of these features at various locations.

Pooling

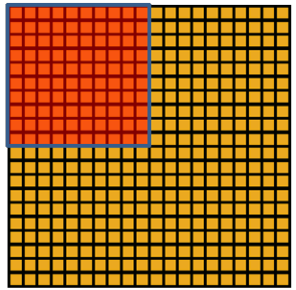
- Subsampling: “mean pooling” or “max pooling”
  - one could compute the mean (or max) value of a particular feature over a region of the image.
  - These summary statistics are much lower in dimension and can also improve results (less over-fitting).



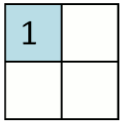


Convolutional Neural Networks

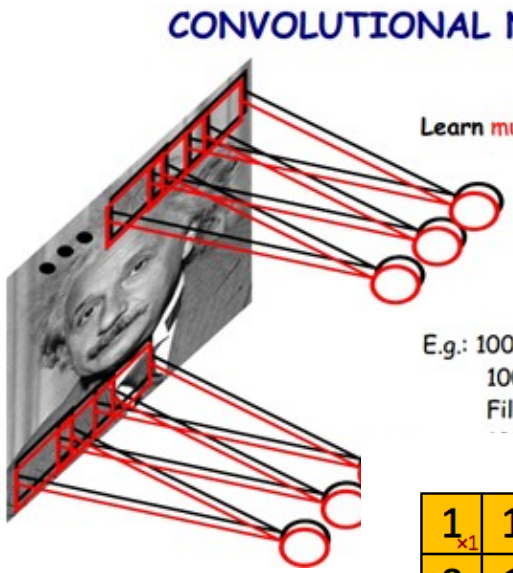
- Composed of
  - Convolution Layers
  - Pooling Layers

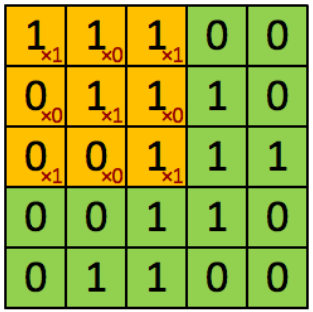


Convolved feature

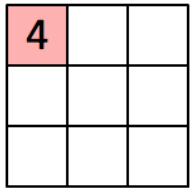


Pooled feature





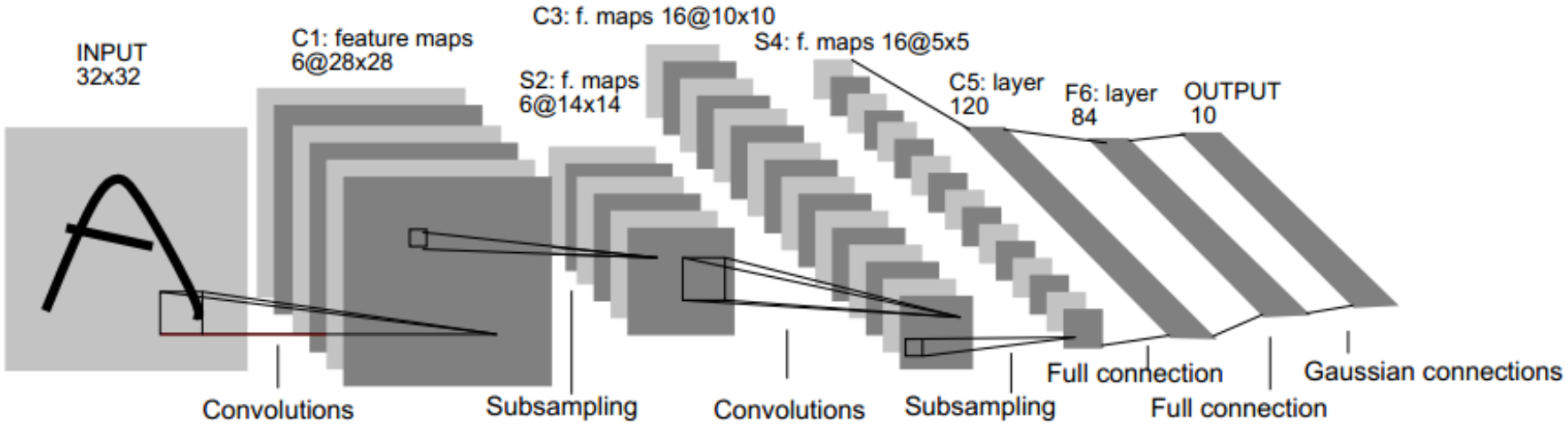
Image



Convolved Feature

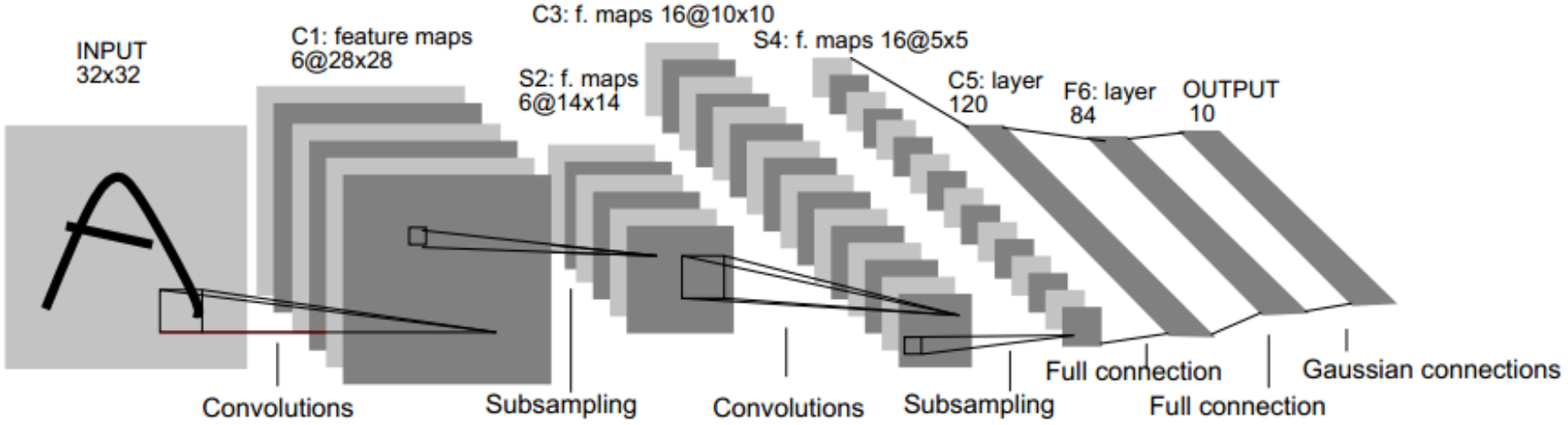
LeNet-5

- C1层是一个卷积层
  - 6个特征图，每个特征图中的每个神经元与输入中5*5的邻域相连，特征图大小为28*28，
  - 每个卷积神经元的参数数目：5*5=25个unit参数和一个bias参数，
  - 连接数目： (5*5+1)*6*(28*28)=122,304个连接
  - 参数共享：每个特征图内共享参数，因此参数总数：共(5*5+1)*6=156个参数



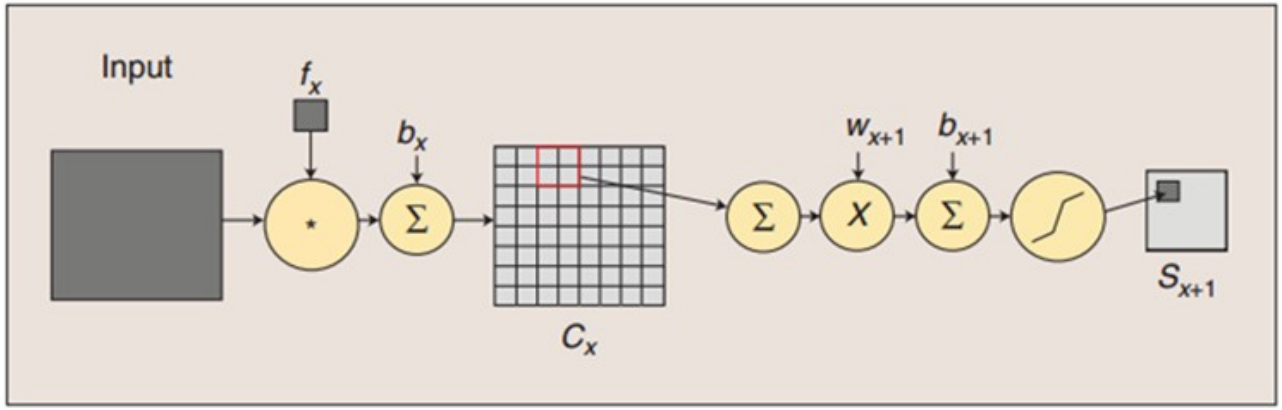
LeNet-5

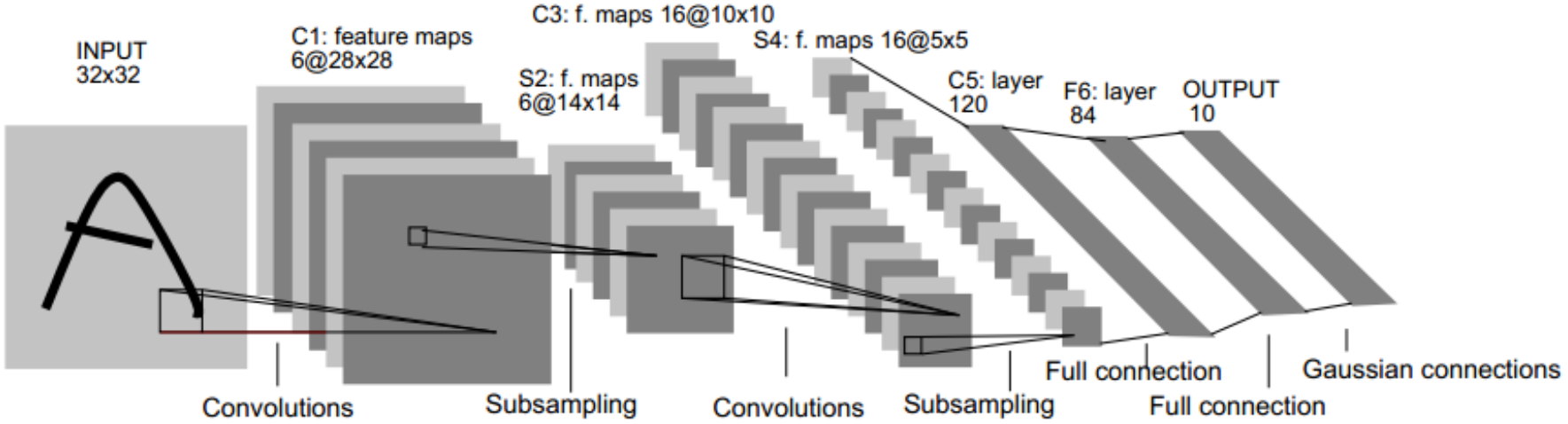
- S2层是一个下采样层
  - 6个14*14的特征图，每个图中的每个单元与C1特征图中的一个2*2邻域相连接，不重叠。因此，S2中每个特征图的大小是C1中特征图大小的1/4.
  - S2层每个单元的4个输入相加，乘以一个可训练参数w，再加上一个可训练偏置b，结果通过sigmoid函数计算。
  - 连接数：（2*2+1）*1*14*14*6 = 5880个
  - 参数共享：每个特征图内共享参数，因此有(2*2+1)*6=30个可训练参数



LeNet-5

- LeCun的表示法

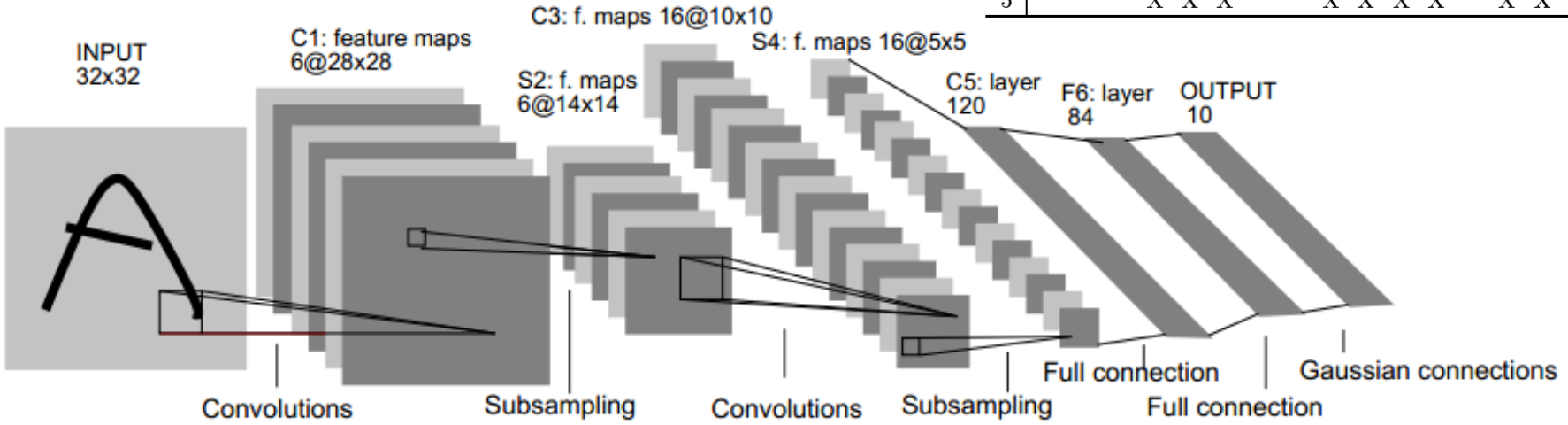




LeNet-5

- C3层是一个卷积层
  - 16个卷积核，得到16张特征图，特征图大小为10*10；
  - 每个特征图中的每个神经元与S2中某几层的多个5*5的邻域相连；

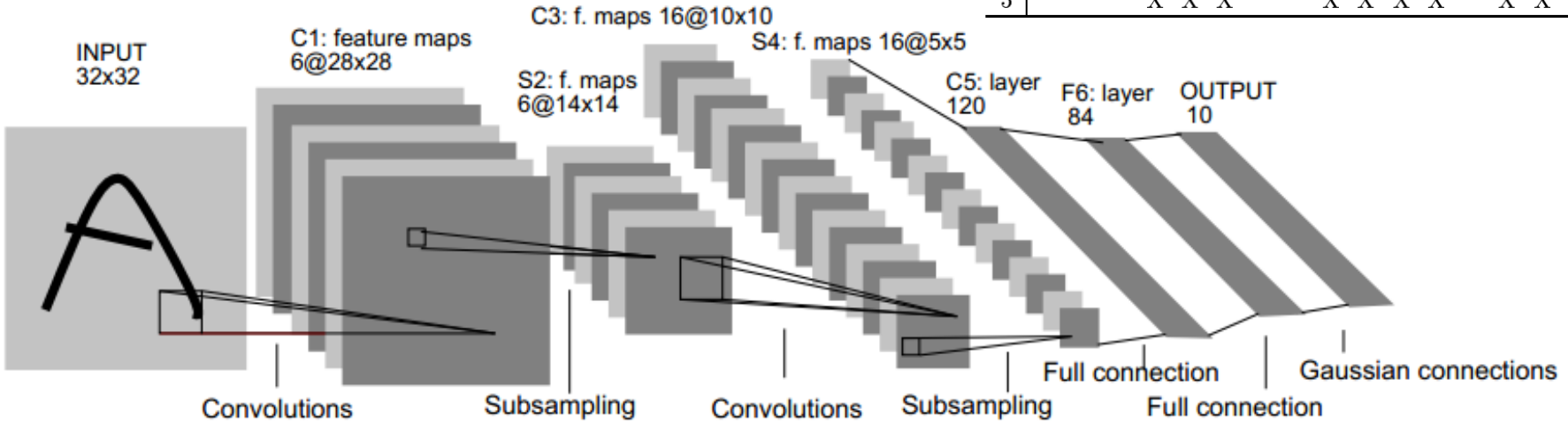
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



LeNet-5

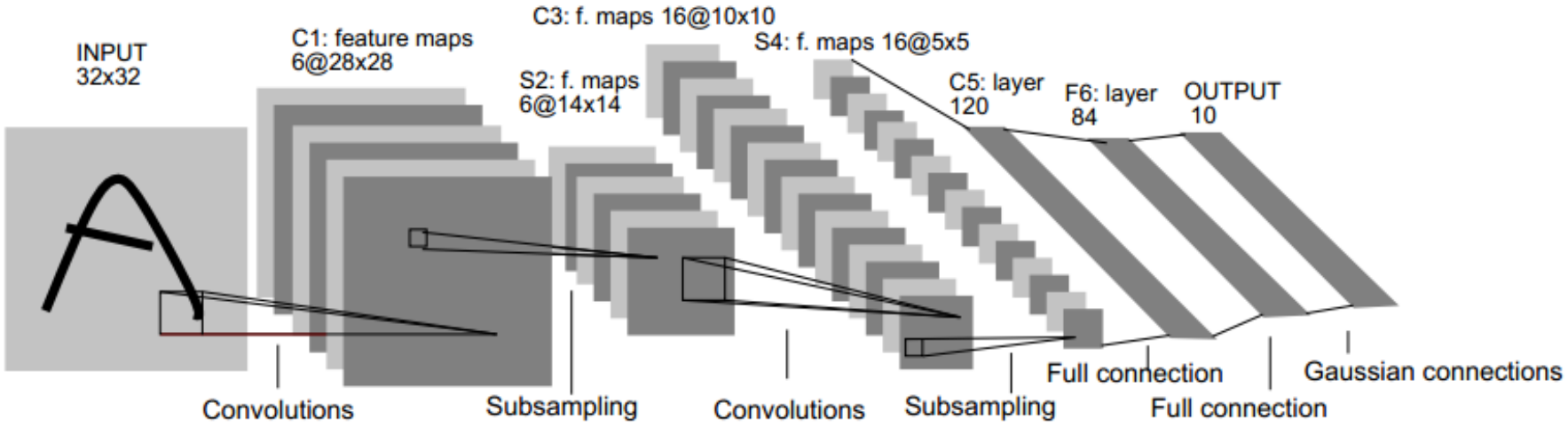
- C3层是一个卷积层
  - 16个卷积核，得到16张特征图，特征图大小为10*10；
  - 每个特征图中的每个神经元与S2中某几层的多个5*5的邻域相连；
    - 例如，对于C3层第0张特征图，其每一个节点与S2层的第0张特征图，第1张特征图，第2张特征图，总共3个5x5个节点相连接。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



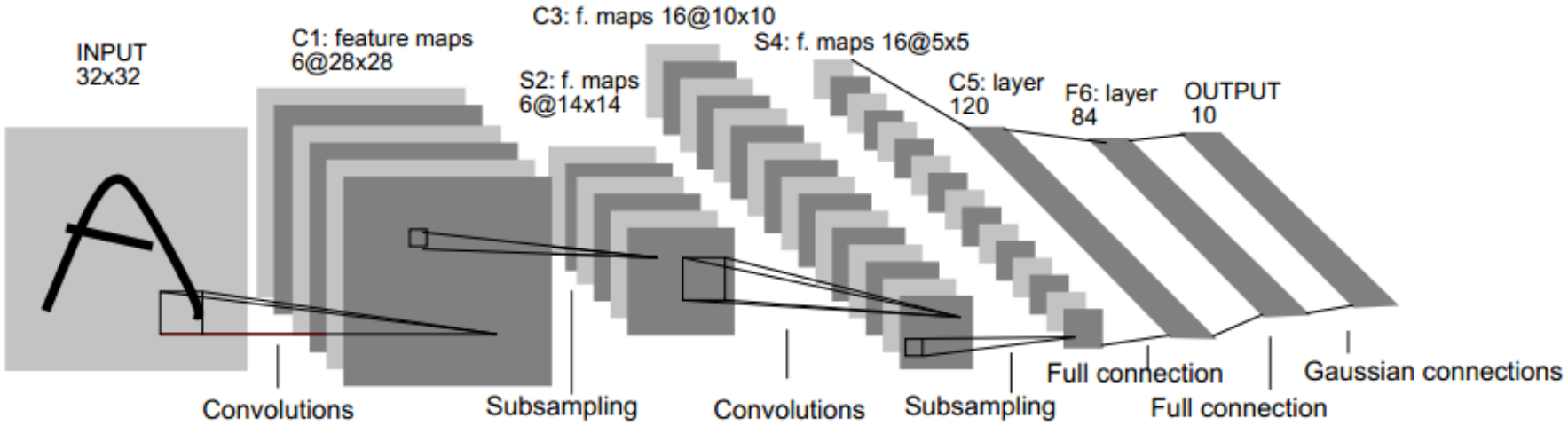
LeNet-5

- S4层是一个下采样层
  - 由16个5*5大小的特征图构成，特征图中的每个单元与C3中相应特征图的2*2邻域相连接；
  - 连接数：（2*2+1）*5*5*16=2000个
  - 参数共享：特征图内共享参数，每张特征图中的每个神经元需要1个因子和一个偏置，因此有 2*16 个可训练参数



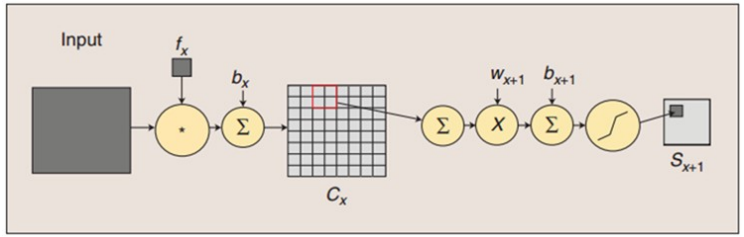
LeNet-5

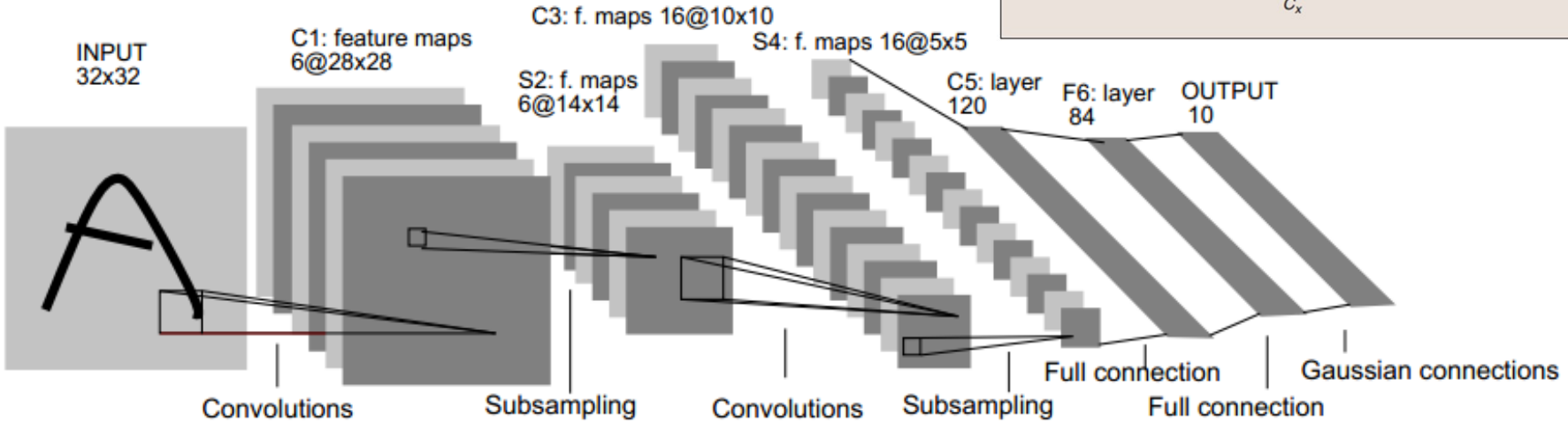
- C5层
  - 120个神经元，可以看作120个特征图，每张特征图的大小为1*1
  - 每个单元与S4层的全部16个单元的5*5邻域相连（S4和C5之间的全连接）
  - 连接数=可训练参数：（5*5*16+1）*120=48120个



LeNet-5

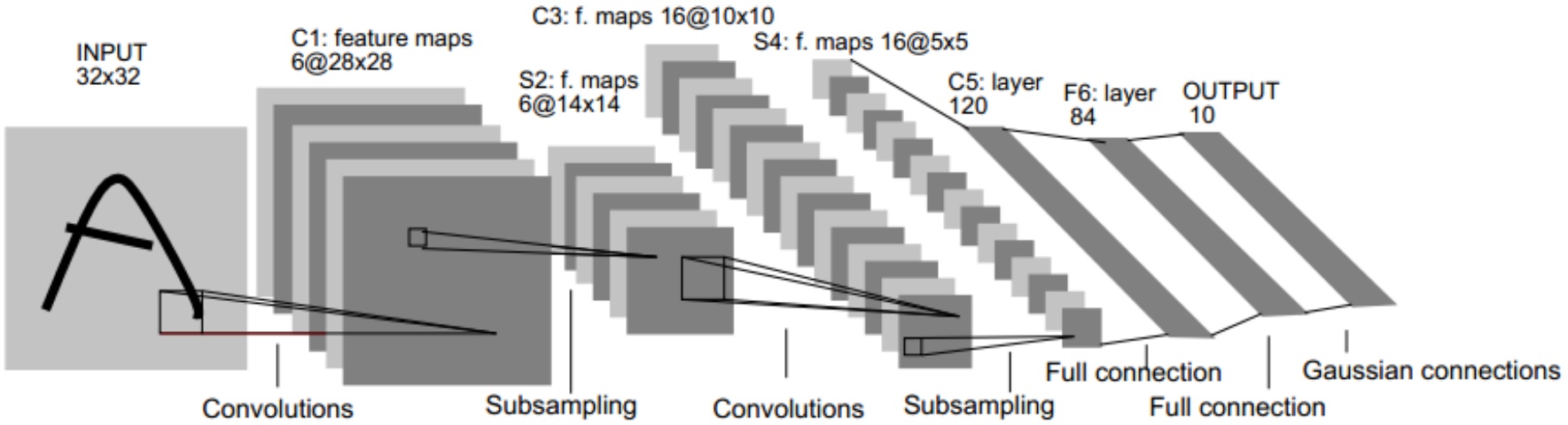
- F6层
  - 有84个单元（之所以选这个数字的原因来自于输出层的设计），与C5层全相连。
  - F6层计算输入向量和权重向量之间的点积，再加上一个偏置。
  - 连接数=可训练参数：（120+1）*84=10164
  - 84：stylized image：7*12





LeNet-5

- 输出层采用欧式径向基函数（Euclidean Radial Basis Function）单元
  - 给定一个输入模式，损失函数应能使得F6的配置与RBF参数向量（即模式的期望分类）足够接近。
  - 每类一个单元，每个单元连接84个输入；每个输出RBF单元计算输入向量和参数向量之间的欧式距离。
  - RBF输出可以被理解为F6层配置空间的高斯分布的【-log-likelihood】



Thanks.