深度学习技术与应用（7）

Deep Learning: Techniques and Applications (7)

Ge Li

Peking University

带有卷积权值的网络

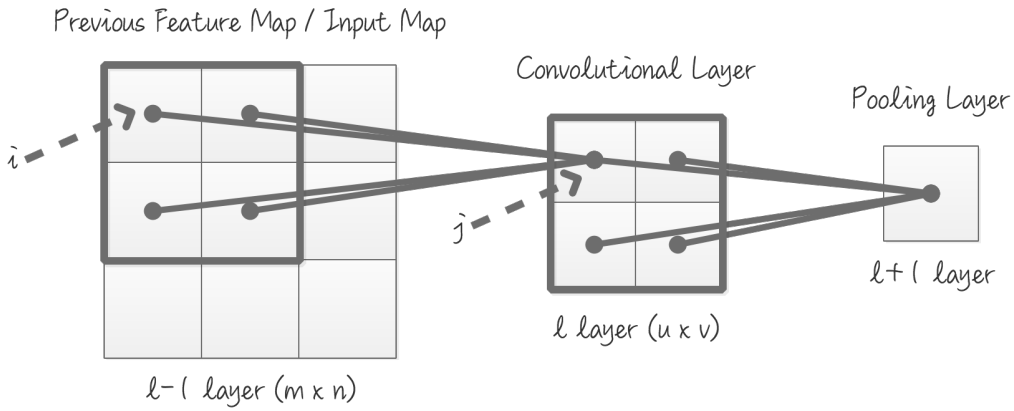
设 α_{ij} 为第 i 个输入特征图与第 j 个卷积核之间的连接强度；

$$a_j^l = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(x_i^{l-1} * k_i^l) + b_j^l\right)$$

其中：

- N_{in} 为输入特征图的个数；
- $\sum_i \alpha_{ij} = 1$ 且 $0 \leq \alpha_{ij} \leq 1$；
- 可以设 $\alpha_{ij} = \frac{exp(c_{ij})}{\sum_k exp(c_{kj})}$；
- 接下来的运算，将针对一个确定的输出特征图 j，因此上述公式可以简化为：$\alpha_i = \frac{exp(c_i)}{\sum_k exp(c_k)}$；

带有卷积权值的网络



接下来看一下，推导 α_{ij}，即 α_i 的求解方法：

$$\frac{\partial J}{\partial \alpha_i} = \frac{\partial J}{\partial z^l}\frac{\partial z^l}{\partial \alpha_i} = \sum_{u,v}(\delta^l \circ (a_i^{l-1} * k_i^l))_{uv}$$

继续求解关于 c_i 的导数：

$$\frac{\partial J}{\partial c_i} = \sum_k \frac{\partial J}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} = \alpha_i\left(\frac{\partial J}{\partial \alpha_i} - \sum_k \frac{\partial J}{\partial \alpha_k}\alpha_k\right)$$

带有卷积权值的网络

为了能够增强 α_i 的效果，我们可以想办法让它变得更稀疏：

$$\hat{J}(W,b) = J(W,b) + \Omega(\alpha) = J(W,b) + \lambda \sum_{i,j} |(\alpha)_{ij}|$$

在这个条件下，α 如何更新呢？

$$\frac{\partial \Omega}{\partial \alpha_i} = \lambda \ sign(\alpha_i)$$

将 α 的求解方式代入：

$$\frac{\partial \Omega}{\partial c_i} = \sum_k \frac{\partial \Omega}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial c_i} = \lambda \left(|\alpha_i| - \alpha_i \sum_k |\alpha_k| \right)$$

因此：

$$\frac{\partial \hat{J}}{\partial c_i} = \frac{\partial J}{\partial c_i} + \frac{\partial \Omega}{\partial c_i}$$

卷积神经网络的正则化处理

l_p 正则化处理：

$$\hat{J}(W,b) = J(W,b) + \Omega(\alpha) = J(W,b) + \lambda \sum_j \parallel W_j \parallel_p^p$$

其中：

- 若 $p > 1$，则 l_p 为突函数，相当于权值衰减；
- 若 $p < 1$，则 l_p 的作用相当于稀疏化处理，最小化代价函数使其趋向于零；

Dropout 处理

l_p 正则化处理：

在全连接情况下，对隐藏层神经元的输出进行如下处理：

$$y = r * f(W^T x + b)$$

其中：

- x 为 n 维输入向量，$W \in R^{(d \times n)}$；
- r 为 d 维向量，且 $r_i \sim Bernoulli(p)$，p 为参数；
- Dropout 方法是一种防止 overfitting 的有效方法，它相当于使用多个网络进行训练；

"Dropout can prevent the network from becoming too dependent on any one (or any small combination) of neurons, and can force the network to be accurate even in the absence of certain information."

Dropout 处理

几种 Dropout 方法的改进：

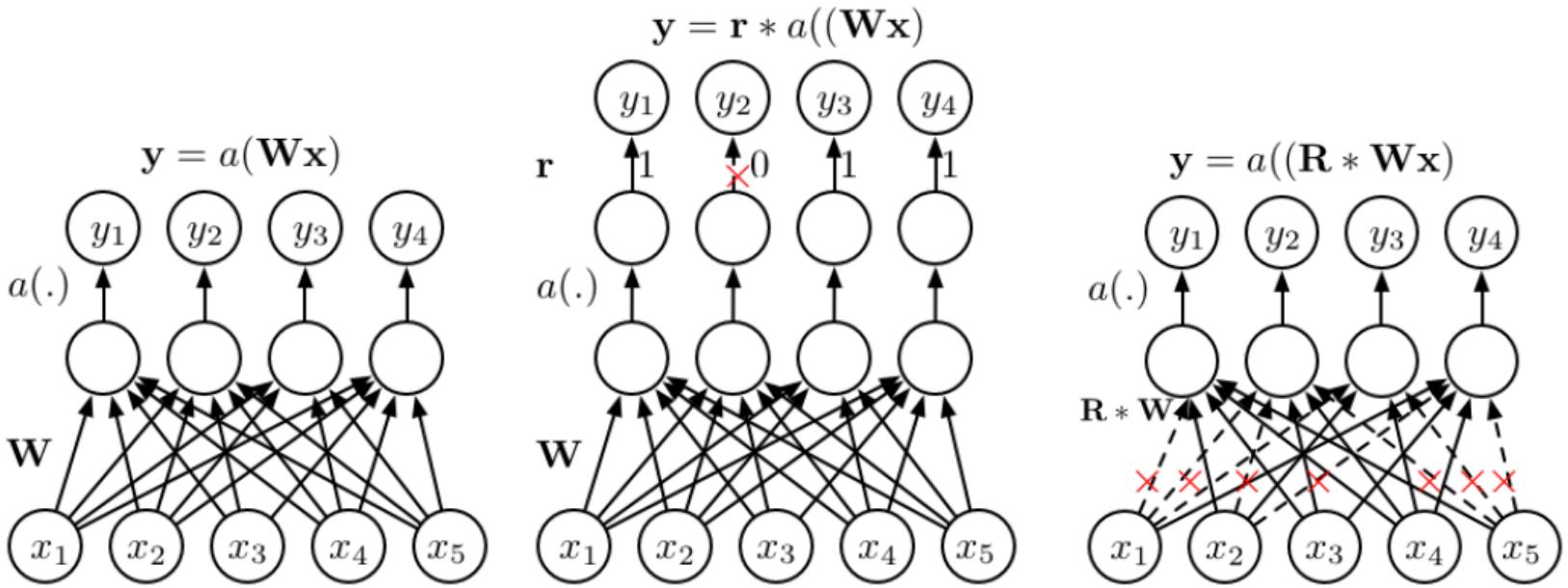
- Fast Dropout[1]: perform fast Dropout training by sampling from or integrating a Gaussian approximation.
- Adaptive Dropout[2]: the Dropout probability for each hidden variable is computed using a binary belief network that shares parameters with the deep network.
- SpatialDropout[3]: extends the Dropout value across the entire feature map, it works well especially when the training data size is small.

[1] S. Wang, C. Manning, Fast dropout training, in: ICML, 2013.

[2] J. Ba, B. Frey, Adaptive dropout for training deep neural networks, in: NIPS, 2013.

[3] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, C. Bregler, Efficient object localization using convolutional networks, in: CVPR, 2015.

Dropout 处理



(a) No-Drop (b) DropOut (c) DropConnect

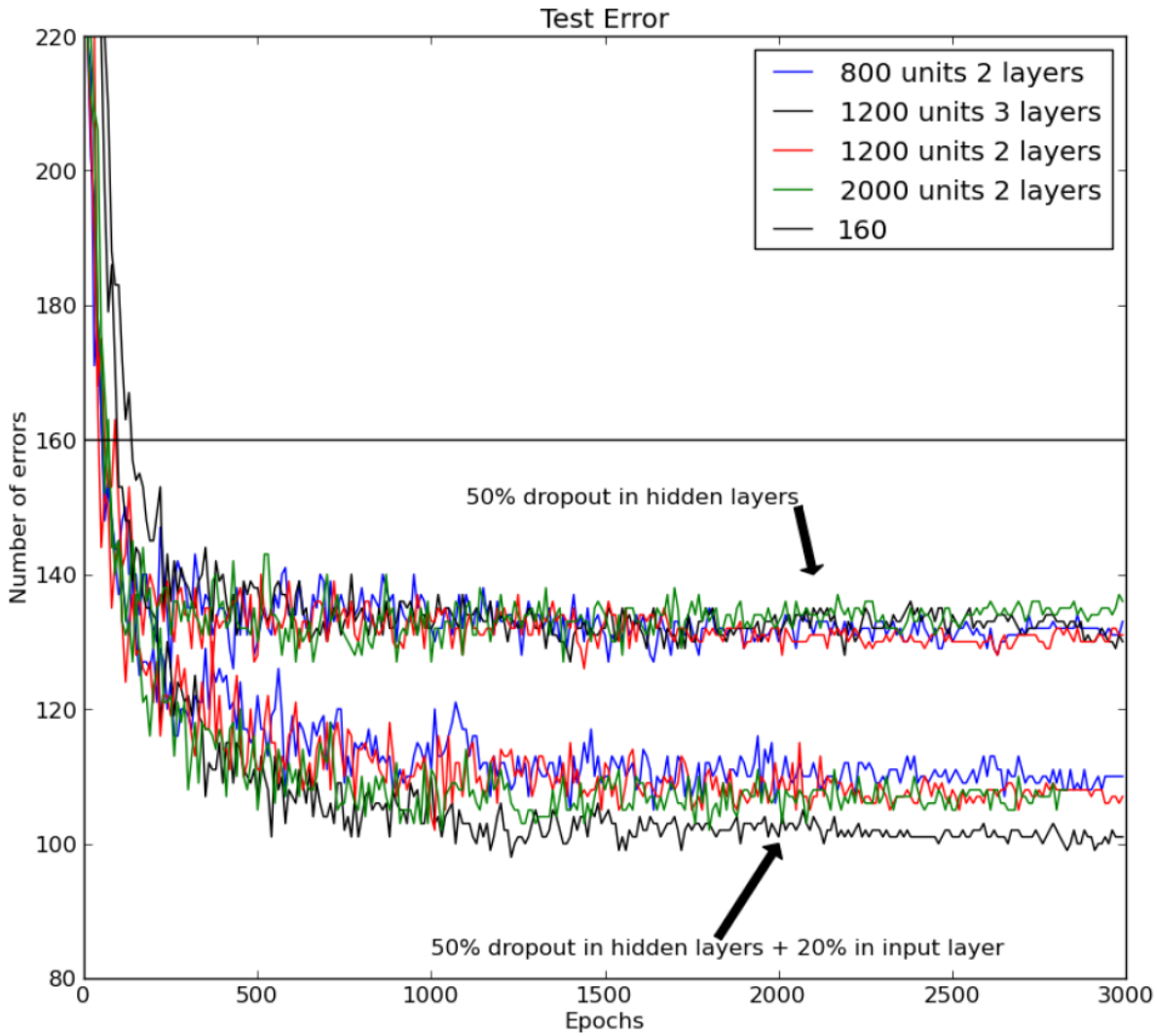
DropConnect 方法：

类似于 Dropout 方法，DropConnect 方法将权重矩阵 W 的某些值设置为 0；在全连接情况下，DropConnect 如下处理：

$$y = r * f(RW^T x + b) \text{其中：} R_{ij} \sim Bernoulli(p)$$

Dropout 处理

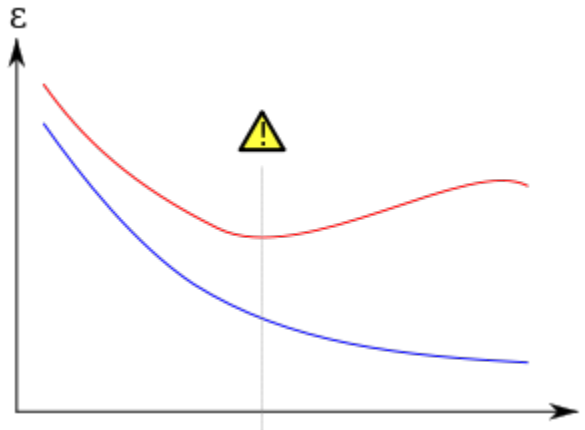
Dropout 的效果：



Overfitting vs. Regularization

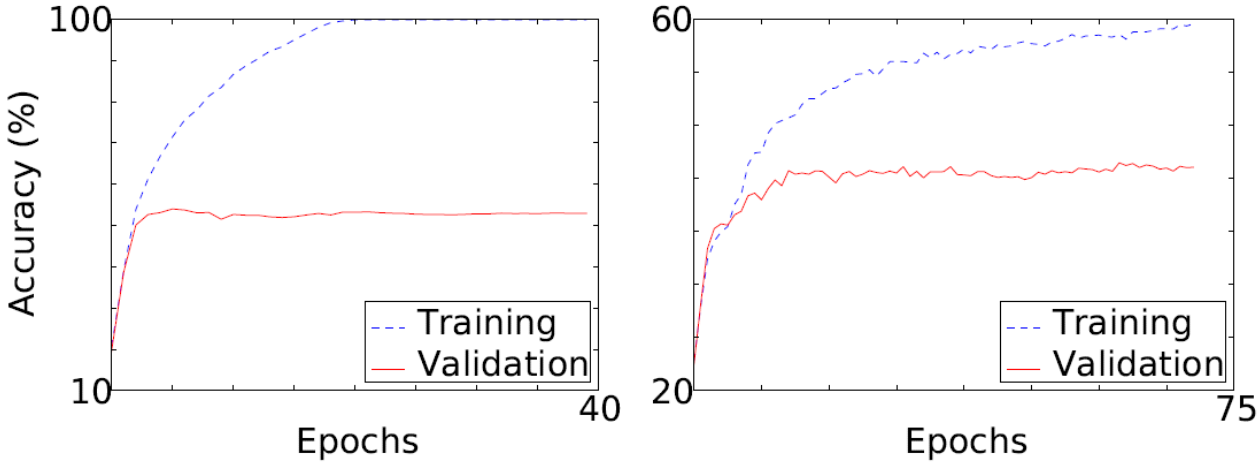
- Neural networks are very prone to overfitting
  - even with low dimensional embeddings (e.g., 50-dimensional), and shallow architectures (e.g., one convolutional layer).
  - Large numbers of parameters contribute much to the VC-dimension, and thus may lead to poor generalization.

If the validation error increases(positive slope) while the training error steadily decreases(negative slope) then a situation of overfitting may have occurred.



Overfitting vs. Regularization

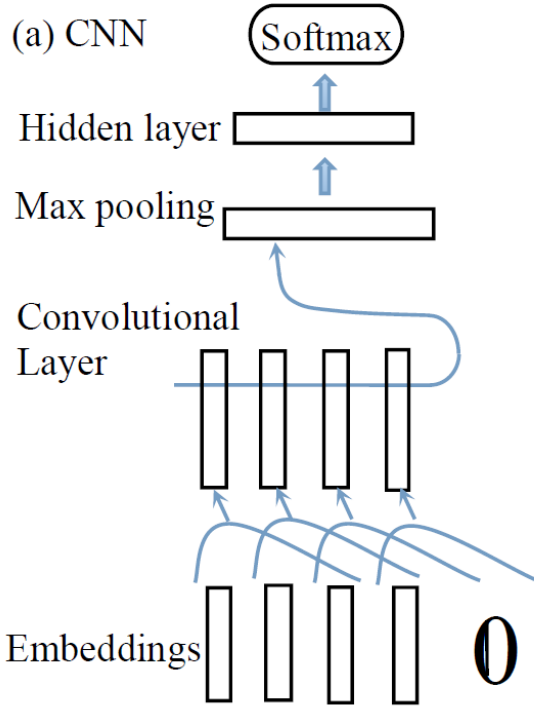
- Training and validation accuracy for relation extraction using the convolutional neural network (Collobert and Weston, 2008).



- Similar phenomenon also appears in sentiment analysis with the recursive neural network (Socher et al., 2011), where training accuracy surpasses validation accuracy by more than 20%.

Regularization Strategies Comparison

- Experiment I: Relation Extraction with CNN
  - Dataset: SemEval-2010 Task 8
  - Model: Collobert 's CNN (2008)



SemEval-2010 Task 8

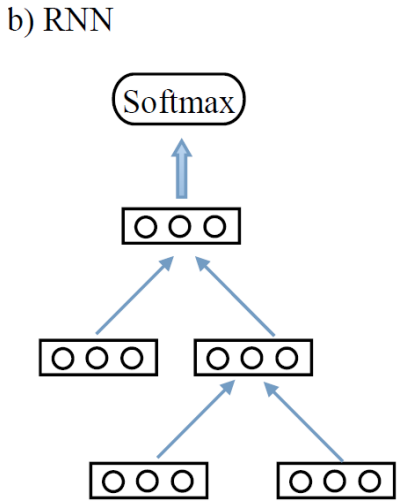
- Each sentence is tagged with two entities.
- The goal is to classify the relationship between these two entities.
- The dataset contains 8,000 samples with additional 3,000 for testing.

Regularization Strategies Comparison

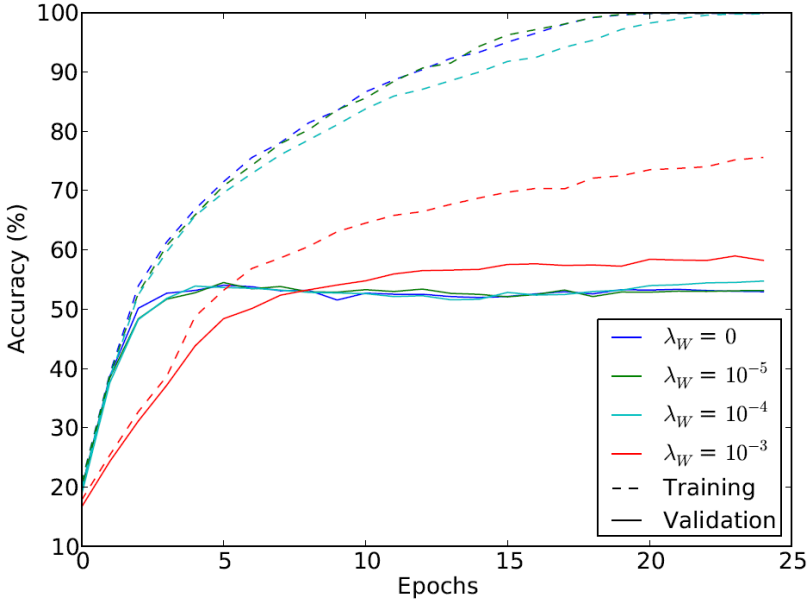
- Experiment II: Sentiment Analysis with RNNs
  - Dataset: Stanford treebank
  - Model : Recursive neural networks, Socher (2011 and 2012)

Stanford treebank

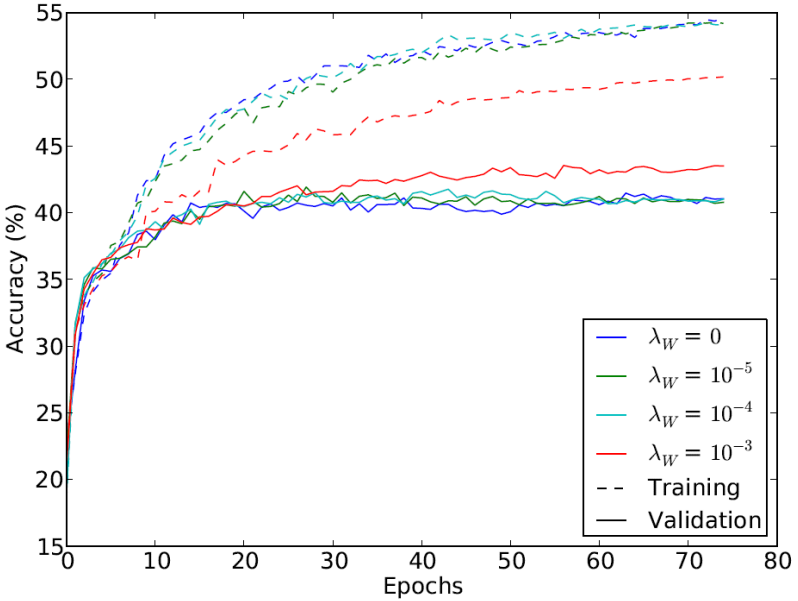
- consists of 152,847 human labeled sentences and phrases for training
- Additional 1101/2210 sentences for validation and testing.
- Target labels include 5 classes:
  - strongly positive, positive, neutral, negative, and strongly negative.



Regularization Strategies Comparison



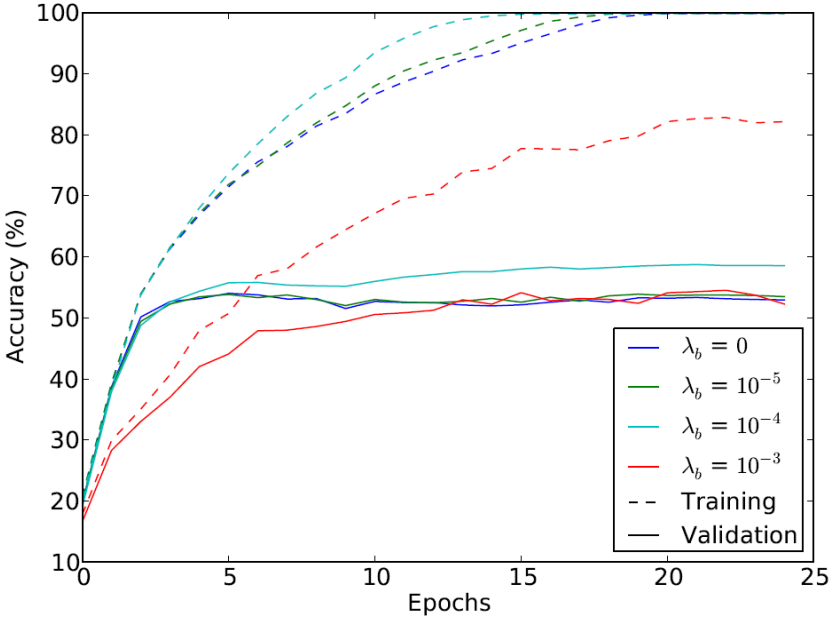
(a) Penalizing weights in Experiment I.



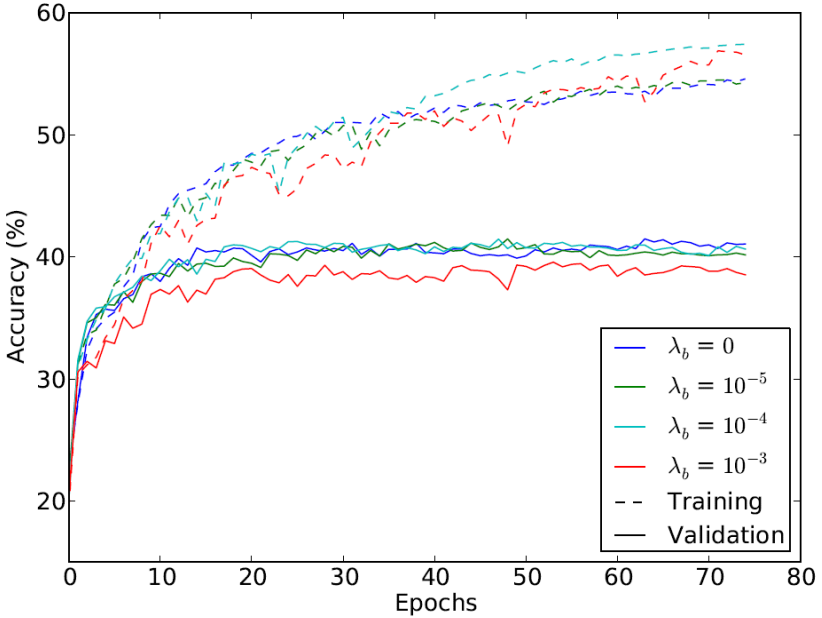
(b) Penalizing weights in Experiment II.

1. Penalizing l_2-norm of weights does help generalization;

Regularization Strategies Comparison



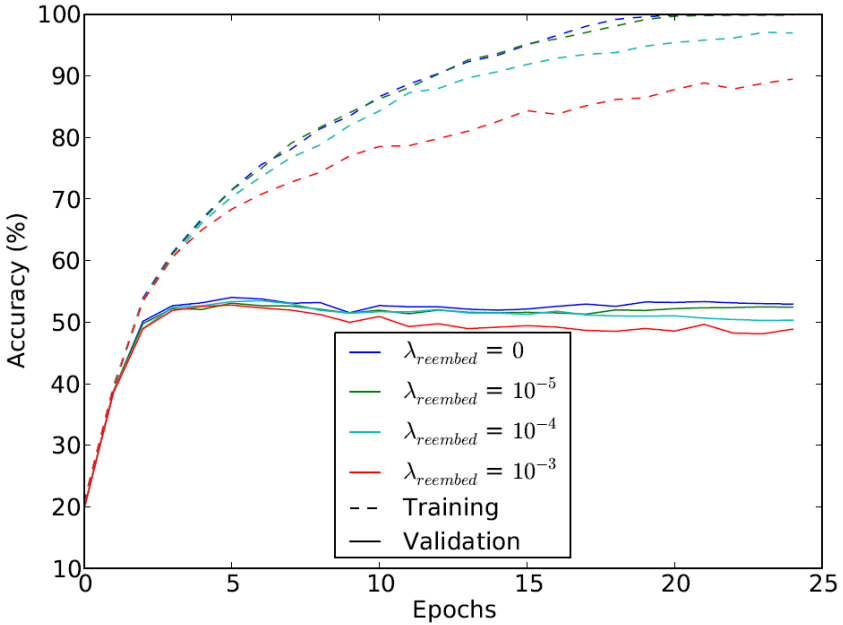
(c) Penalizing embeddings in Experiment I.



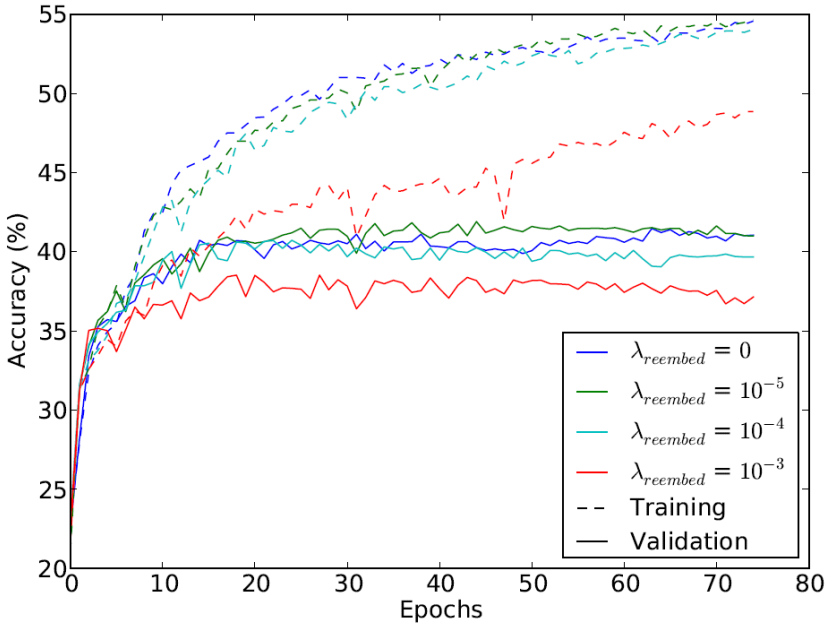
(d) Penalizing embeddings in Experiment II.

2. Penalizing l_2-norm of embeddings unexpectedly helps optimization (improves tranning accuracy).

Regularization Strategies Comparison



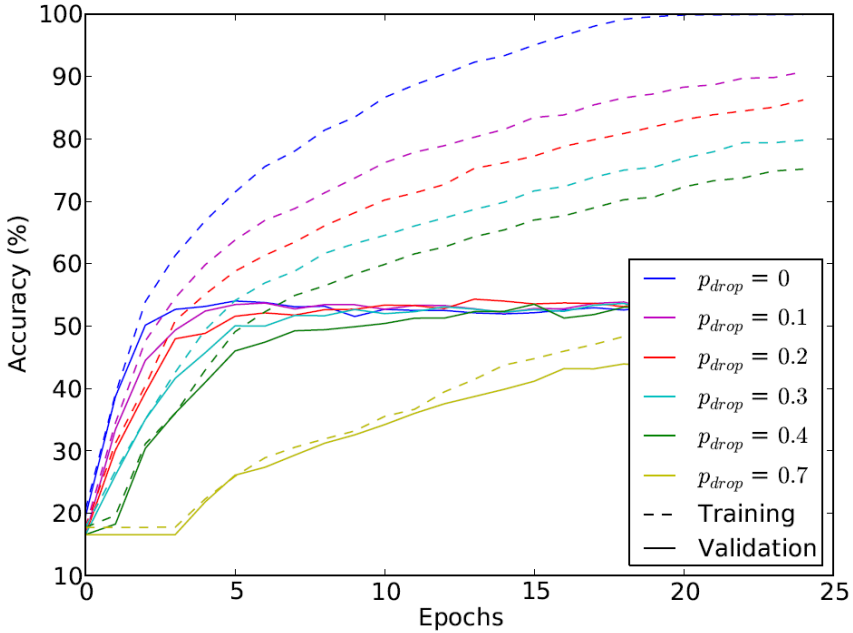
(e) Re-embedding words in Experiment I.



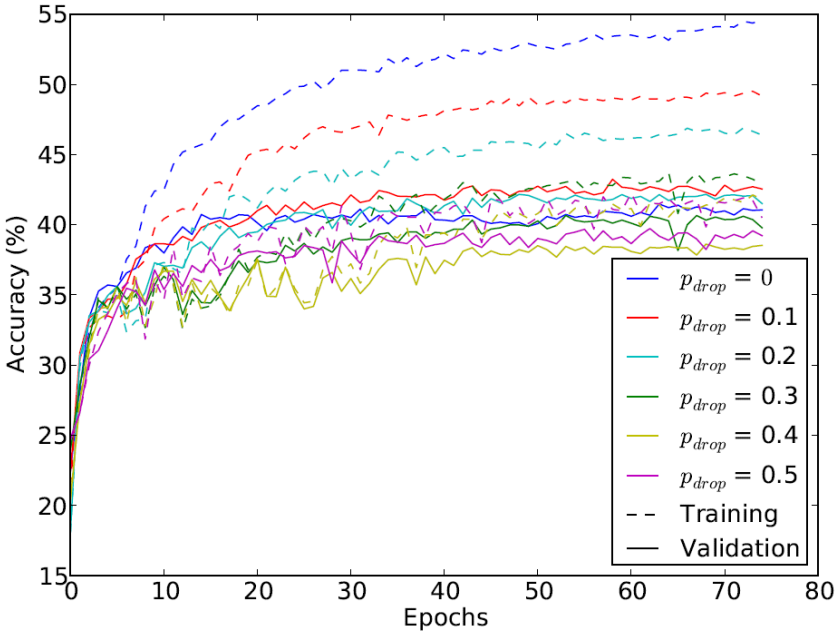
(f) Re-embedding words in Experiment II.

3. Re-embedding words, proposed in Labutov and Lipson (2013), does not improve generalization.

Regularization Strategies Comparison



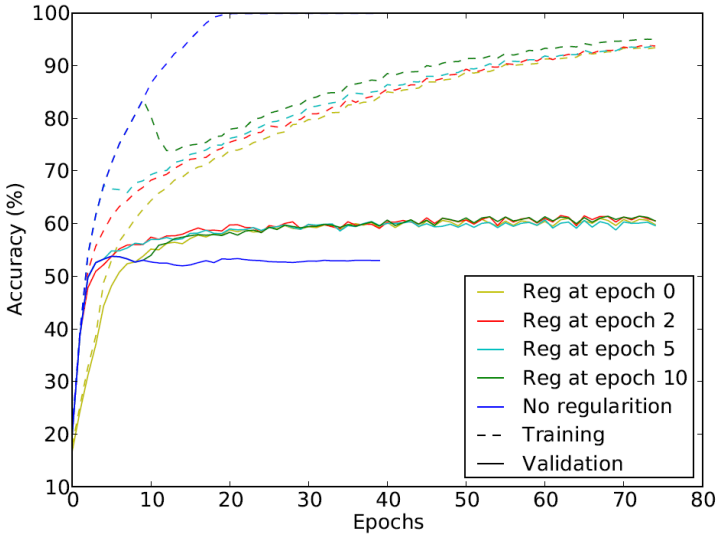
(g) Applying dropout in Experiment I. $p = 0.5, 0.6$ are omitted because they are similar to small values.



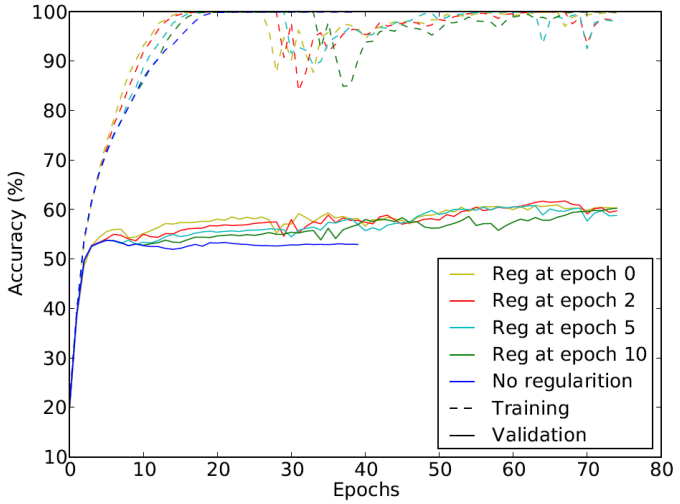
(h) Applying dropout in Experiment II.

- 4. Dropout does help generalization. Even though the training process becomes slower, the validation accuracy is still rising slightly after 200 epochs.

Regularization Strategies Comparison – Incrementally penalizing



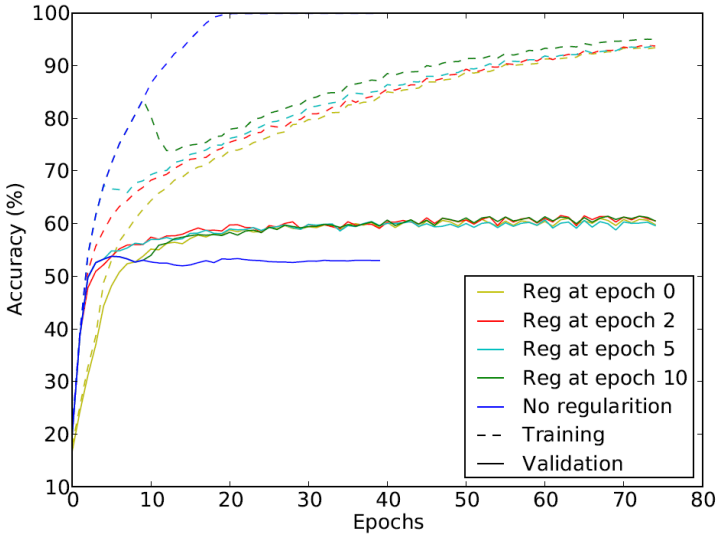
(a) Incrementally penalizing ℓ_2-norm of weights.



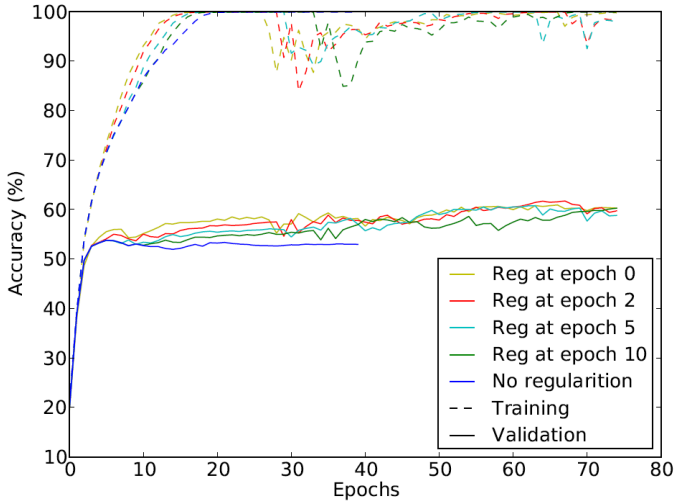
(b) Incrementally penalizing ℓ_2-norm of biases.

- all settings yield similar ultimate validation accuracies no matter when penalty is added.
- This shows l_2 regularization mainly serves as a "local" effect

Regularization Strategies Comparison – Incrementally penalizing



(a) Incrementally penalizing ℓ_2-norm of weights.



(b) Incrementally penalizing ℓ_2-norm of biases.

- (a) regularization helps generalization as soon as it is added: while the training accuracy first drops and then rises, the test accuracy goes up immediately.
- (b) that regularizing embeddings also helps optimization (training accuracy improves) right after the penalty is added.

Regularization Strategies Comparison – Combination of Regularizations

λ_{embed}	λ_{W} 0	10^{-4}	$3 \cdot 10^{-4}$	10^{-3}
0	54.02	57.88	59.96	61.00
10^{-5}	54.94	57.82	60.68	62.05
$3 \cdot 10^{-5}$	55.68	61.02	**64.00**	**63.15**
10^{-4}	60.91	**64.00**	**63.07**	60.56
$3 \cdot 10^{-4}$	58.92	61.33	59.85	42.93
10^{-3}	54.77	56.43	54.05	16.50

- Table 1 shows that combining l_2-norm of weights and embeddings further improved accuracy by 3–4 in percentage than applying either single one of them.
- In a certain range of coefficients, weights and embeddings are compensatory: given one hyperparameter, we can tune the other to achieve the results among highest ones.

Regularization Strategies Comparison – Combination of Regularizations

	λ_W			λ_{embed}		
p	10^{-4}	$3\cdot10^{-4}$	10^{-3}	10^{-5}	$3\cdot10^{-5}$	10^{-4}
0	57.88	**59.96**	**61.00**	54.94	55.68	**60.91**
1/6	58.36	59.36	43.42	58.49	59.59	**60.00**
2/6	58.22	**60.00**	16.60	59.34	**60.08**	59.61
3/6	58.63	59.73	16.60	59.59	**59.98**	58.82
4/6	56.43	54.63	16.60	56.76	59.19	56.64
5/6	38.07	16.60	16.60	49.79	53.63	49.75

- Such compensation is also observed in penalizing l_2-norm versus dropout
- the peak performance is obtained by pure l_2-norm regularization
- applying dropout with small l_2 penalty also achieves similar accuracy.
- The dropout rate is not very sensitive, provided it is small (less than 0.5, say).

权重初始化方法

初始化参数的目的：

- 初始化是消除神经网络隐藏层同层神经元之间存在的对称性;
- 不合理的初始化将导致怪异的输出结果：
  - 若采用统一的初始化值，梯度下降又促使所有参数按照相同的方式进行调整，将导致训练实效；
  - 不合理的初始化，可能导致训练结果持续变大，或持续变小，从而出现怪异的输出结果；
  - 不合理的分布，可能导致权重参数出现方差越来越大的现象；
- 应该尽量控制参数的变化范围：使其方差变化不至于太大。

权重初始化方法

常见的初始化方法有如下几种：

1. 论文 [1] 按照标准差为 0.01，均值为零的高斯分布随机生成数值，初始化权重 W，并将第 2，第 4 和第 5 个卷积层，及所有全连接层的偏置项设置为常数；
2. 论文 [2] 提出一种 Xavier 方法；论文 [3] 又给出了一种考虑 ReLU 函数的改进方法；

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: NIPS, 2012.

[2] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: AISTATS, 2010.

[3] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: ICCV, 2015.

权重初始化方法

根据 Xavier 方法：当激活函数在 0 值附近，导数接近 1 的条件下，其初始化参数可以按照如下范围内的均匀分布提取：

$$\left[-\sqrt{\frac{6}{n^{k+1}+n^{k}}}, \sqrt{\frac{6}{n^{k+1}+n^{k}}}\right]$$

接下来推证其合理性。先来回顾预备知识：
方差：如果随机变量 X 的数学期望 $\mu = EX$ 有限，则称：

$$E(X-\mu)^2$$

为 X 的方差，记作 $var(X)$.
若随机变量 X 和 Y 均服从均值为 0，方差为 σ 的分布，则：

- $X * Y$ 服从均值为 0，方差为 σ^2 的分布；
- $X * Y + X * Y$ 服从均值为 0，方差为 $2\sigma^2$ 的分布；

权重初始化方法

设输入数据 $x \in R^n$ 服从均值为 0，方差为 σ_x 的分布；
设 $w \in R^n$ 为输入层 n 个神经元与输出层第 j 个神经元之间的连接权重；
设 w 服从均值为 0，方差为 σ_w 的分布，则：

$$z_j = \sum_i^n w_i * x_i$$

根据方差的性质，z_j 满足均值为 0，方差为 $n\sigma_x\sigma_w$ 的分布；

注意到，在 0 值附近，神经网络激活函数近似服从 $a = x$，因此，在不考虑 w 的情况下有：$a_j = x$. 那么，若要使神经网络输入层与其下一层的输出值保持方差不变，则应令：

$$\sigma_w = \frac{1}{n}$$

权重初始化方法

若神经网络有 k 层，则根据前向传播公式，第 k 层神经元的 z 值方差为：

$$\sigma_x^k = \sigma_x^1 * \prod_{i=2}^{k} n^i * \sigma_w^i$$

可见，若要使神经网络输入层与其下一层的输出值保持方差不变，需令：

$$\sigma_w^k = \frac{1}{n^{k-1}}$$

即：第 k 层的权值的方差 σ_w^k 应为第 $k-1$ 层神经数目的倒数；
接下来，看反向传播过程中的约束关系。

权重初始化方法

因为：

$$x_i^k = f(\sum_{j=1}^{n^{k-1}} w_j^k * x_j^{k-1} + b)$$

则：

$$\frac{\partial J}{\partial x_j^{k-1}} = \sum_{i=1}^{n^k} \frac{\partial J}{\partial x_i^k} * w_j^k$$

则，根据方法性质公式，得：

$$\text{var}\left(\frac{\partial J}{\partial x_j^{k-1}}\right) = n^k * \text{var}\left(\frac{\partial J}{\partial x_i^k}\right) * \sigma_w^k$$

权重初始化方法

若神经网络有 k 层，则在反向传播中，有：

$$\text{var}\left(\frac{\partial J}{\partial x_j^1}\right) = \text{var}\left(\frac{\partial J}{\partial x_i^k}\right) * \prod_{i=1}^{k-1} n^i * \sigma_w^i$$

可见，若要使神经网络输入层与其下一层的输出值保持方差不变，需令：

$$\sigma_w^k = \frac{1}{n^k}$$

即：第 k 层的权值的方差 σ_w^k 应为第 k 层神经数目的倒数；

权重初始化方法

综上所述，我们得到如下两个约束关系：

$$\sigma_w^k = \frac{1}{n^k} \qquad \sigma_w^k = \frac{1}{n^{k-1}}$$

对上述两个约束条件进行融合，得第 k 层 w 的约束条件为：

$$\sigma_w^k = \frac{2}{n^{k-1} + n^k}$$

这是初始化应满足的第一个条件。

设，我们对权重进行初始化的取值条件是：$[-u, u]$ 之间的均匀分布，则依据均匀分布的方差公式，又有初始化满足的第二个条件：

$$\text{var(uniform)} = \frac{(u - (-u))^2}{12} = \frac{u^2}{3}$$

权重初始化方法

联合上述两个条件，于是有：

$$\sigma_w^k = \frac{2}{n^{k-1} + n^k} = \frac{u^2}{3}$$

得：

$$u = \sqrt{\frac{6}{n^{k-1} + n^k}}$$

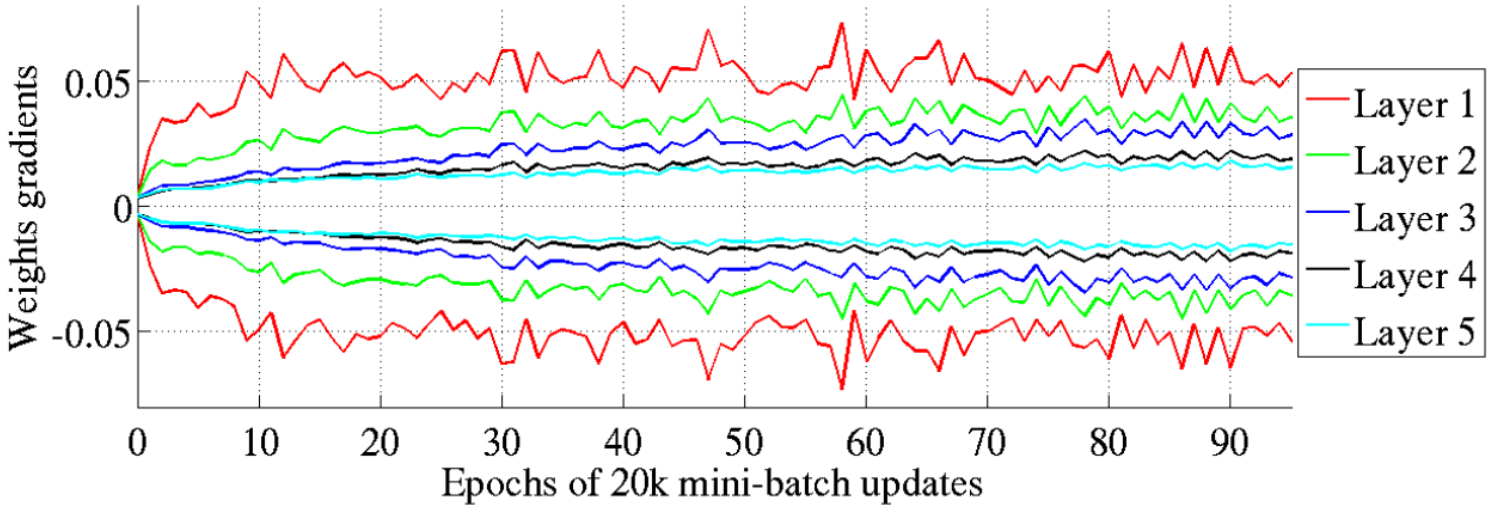
即得到 Xavier 方法：

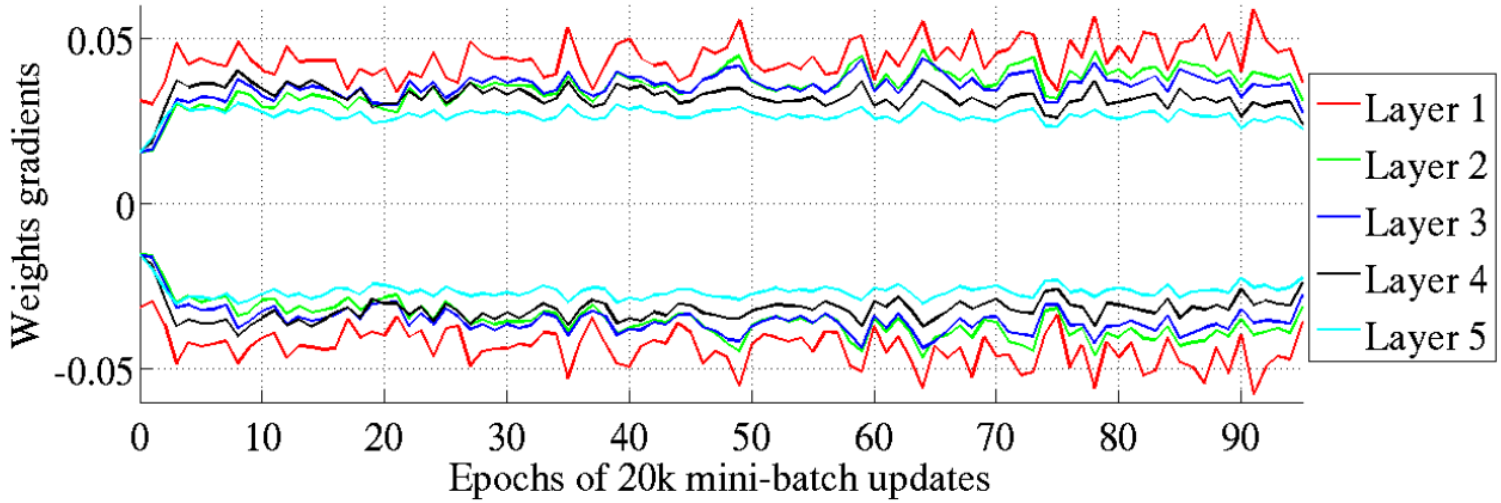
当激活函数在 0 值附近，导数接近 1 的条件下，其初始化参数可以按照如下范围内的均匀分布提取：

$$\left[-\sqrt{\frac{6}{n^{k+1} + n^k}}, \sqrt{\frac{6}{n^{k+1} + n^k}}\right]$$

权重初始化方法

Xavier 方法的效果：





Batch Normalization

进行 Batch Normalization 的原因：

- 神经网络的训练目标是在输出层得到原始输入数据数据分布的一个映射，即在训练过程中，应该力图保证数据分布的映射关系；若数据分布在训练过程中发生了偏移，则会降低网络的泛化能力；
- 在网络训练过程中，后一层网络的输入是前一层的输出，因此，前一层网络参数的变化，将导致后一层输入数据分布的改变，且这种改变会在训练过程中向后传递并被逐步放大；这种在训练过程中，数据分布的改变称为“Internal Covariate Shift”;
- 在 Batch-based Training 中，若个 Batch 的分布各不相同，网络需要在每个 Batch 的训练中适应不同的分布，从而大大降低训练速度；

所以，为了避免上述问题，可以考虑针对每层网络的每个 Batch 进行 Batch Normalization. 这是一种提高训练速度和效果的非常有效的方法.

Batch Normalization

进行 Batch Normalization 的条件：

1. 起到 Normalize 的作用：控制数据的均值与方差在一定范围内；
2. 保持 Normalize 之前的数据与 Normalize 之后数据之间的映射关系；
3. 保证 Normalize 方法 / 函数的可导性；

论文 [1] 提出了一种针对每个 Batch 进行 Normalize 的方法：

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

[1]Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

反向传播阶段的计算：

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Batch Normalization

数据测试阶段：

因为测试阶段输入数据可能只有一个，因此，我们使用所有 Batch 的 μ_B 的期望值代替上述公式中的 $E[x]$；使用所有 Batch 方差 δ_B^2 的无偏估计代替上述公式中的 $\text{Var}[x]$，即：

$$\begin{aligned}\mathrm{E}[x] &\leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}] \\ \mathrm{Var}[x] &\leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]\end{aligned}$$

又因为，上述公式中：

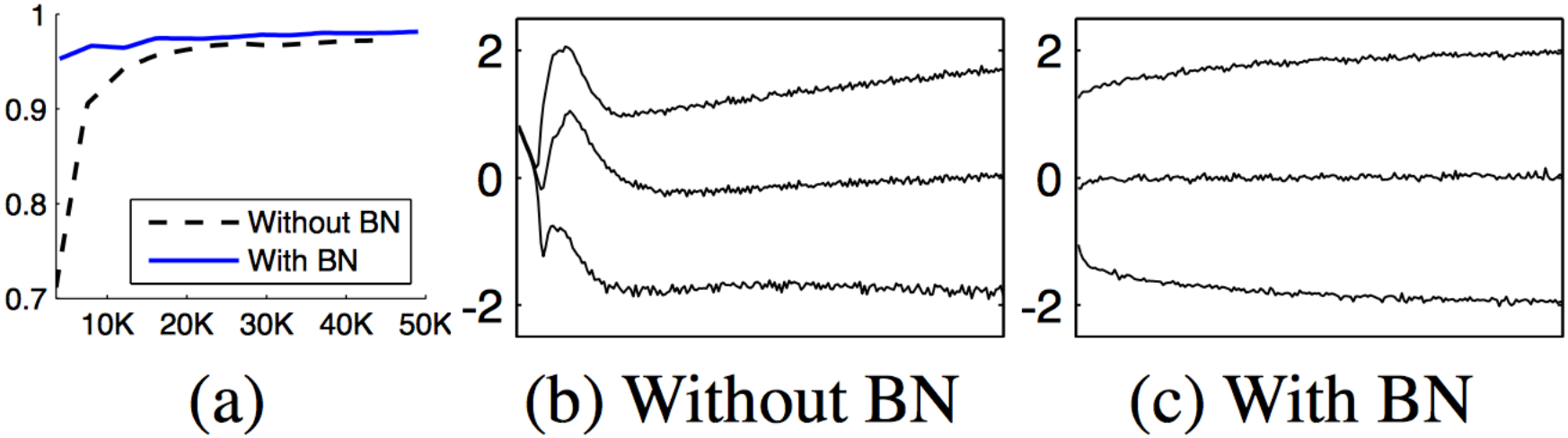
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

则，Batch Normalization 层计算的公式为：

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

Batch Normalization

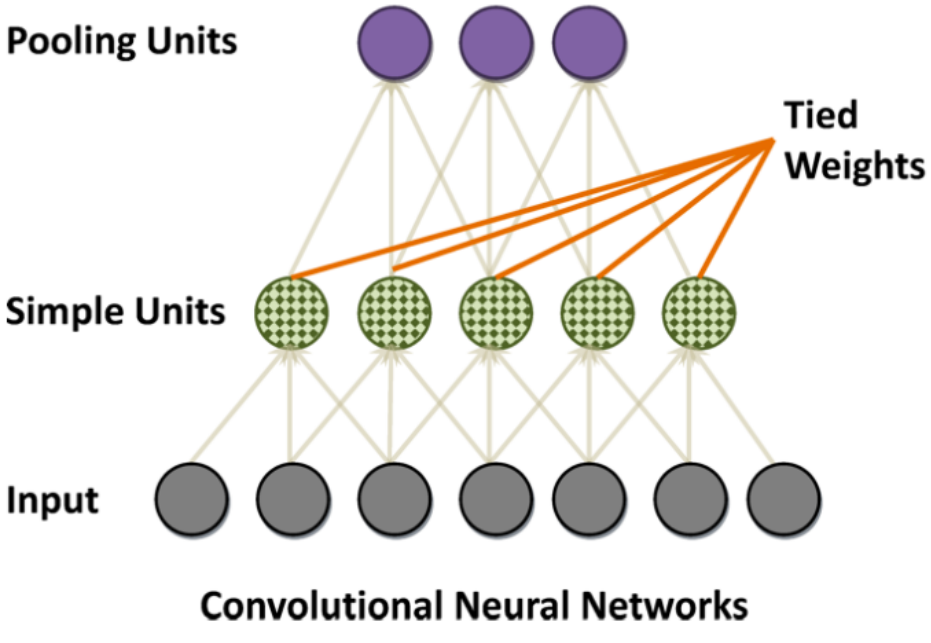
Batch Normalization 的效果：



(a) (b) Without BN (c) With BN

Tiled Convolution

Tiled Convolution :



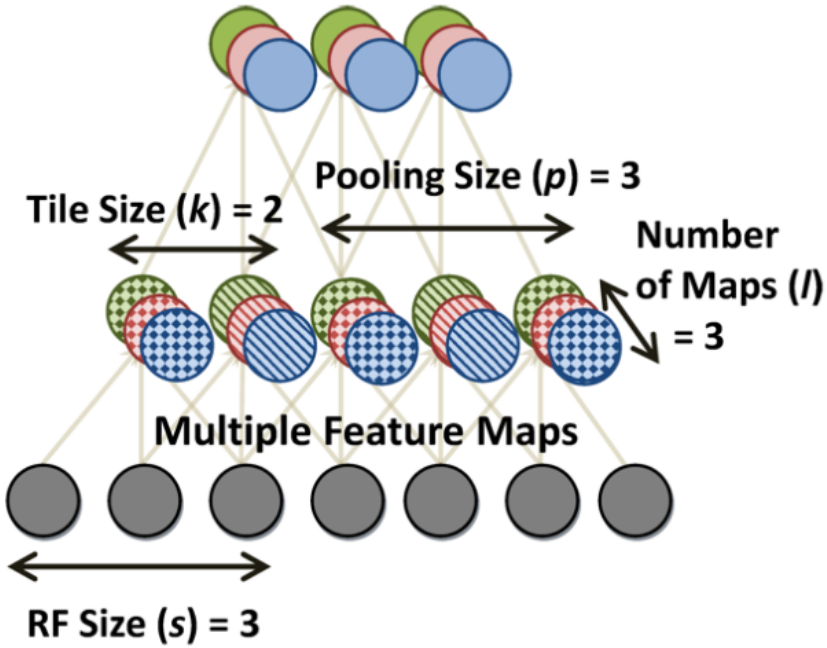
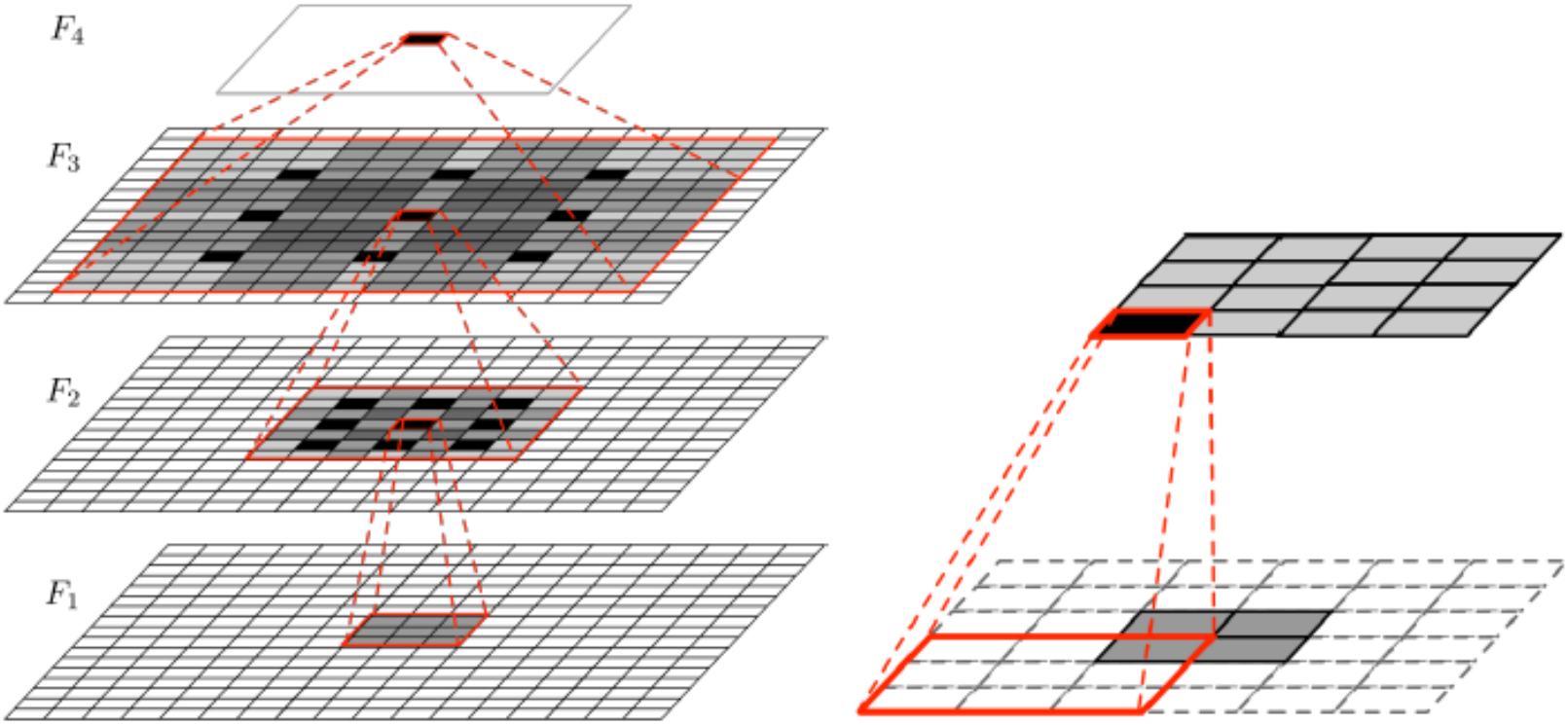


Figure 1: Left: Convolutional Neural Networks with local receptive fields and tied weights. Right: Partially untied local receptive field networks – Tiled CNNs. Units with the same color belong to the same map; within each map, units with the same fill texture have tied weights. (Network diagrams in the paper are shown in 1D for clarity.)

[1] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, A. Y. Ng, Tiled convolutional neural networks, in: NIPS, 2010.

Transposed Convolution, Dilated Convolution



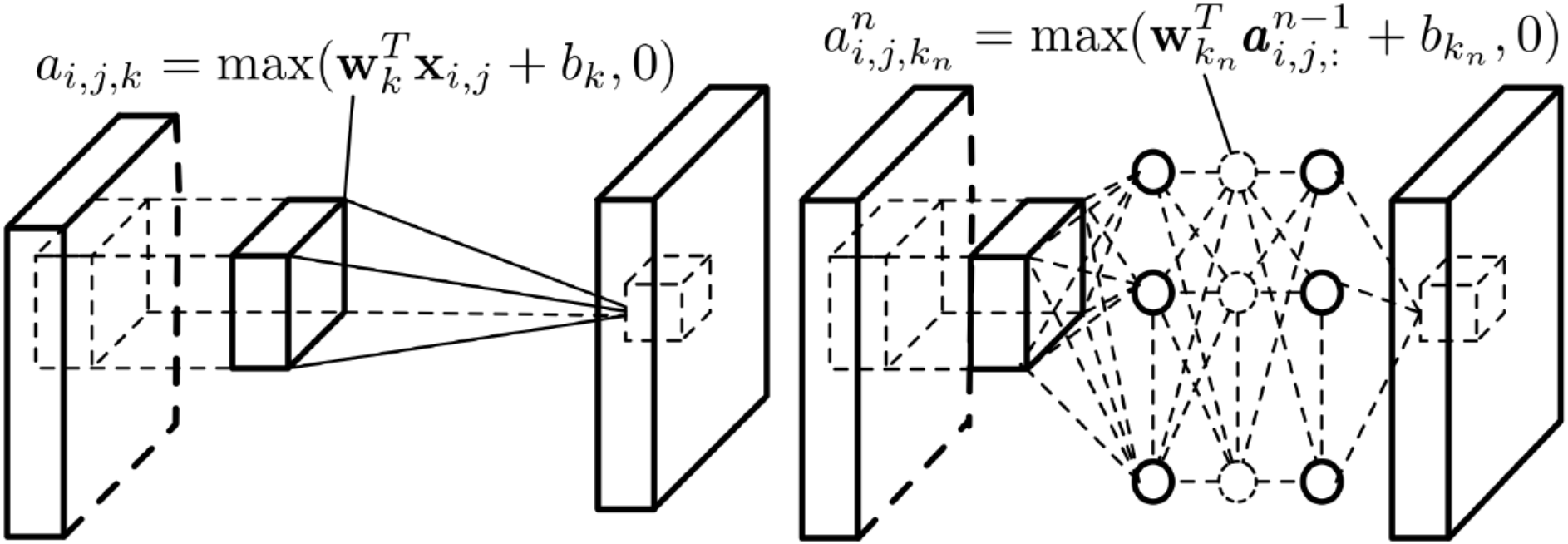
(c) Dilated Convolution (d) Deconvolution

[d] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: ECCV, 2014.

[c] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: ICLR, 2016.

Network in Network

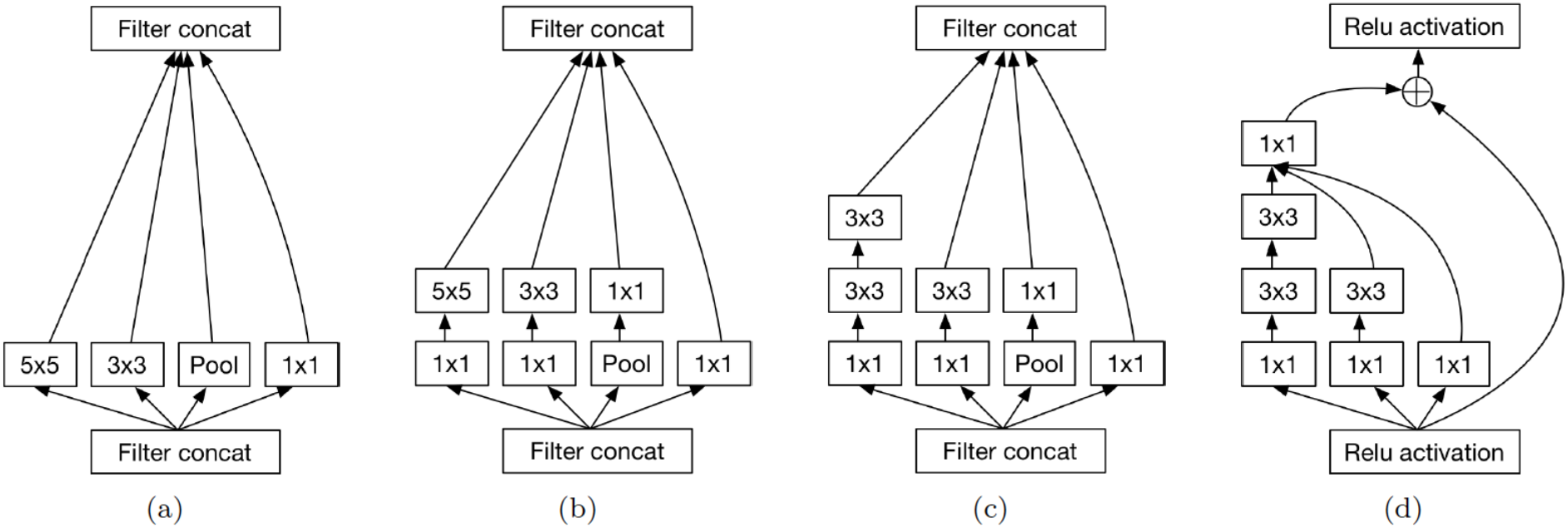
Network in Network:



(a) Linear convolution layer (b) Mlpconv layer

[1] M. Lin, Q. Chen, S. Yan, Network in network, in: ICLR, 2014.

Inception Module



[b] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: CVPR, 2015.
[c] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: arXiv preprint arXiv:1512.00567, 2015.
[d] C. Szegedy, S. Ioffe, V. Vanhoucke, Inception-v4, inception-resnet and the impact of residual connections on learning, in: arXiv preprint arXiv:1602.07261, 2016.

Thanks.