Neural Network Basis

Yimeng Ren

November 29, 2019

Section 1

Neuron

Neuron Structure

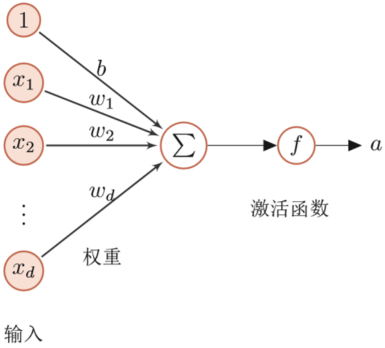


Figure 1: 典型神经元结构

Neuron Structure

$$z = \sum_{i=1}^{d} w_i x_i + b$$
$$= w^T x + b$$
$$a = f(z)$$

- 其中，$w = [w_1, w_2, ..., w_d] \in \mathbb{R}^d$ 是 d 维的权重向量，$b \in \mathbb{R}$ 是偏置。
- 非线性函数 $f(.)$ 称为**激活函数**（Activation Function）

Section 2

Common Activation Functions

Sigmoid

- 两端饱和
- Logistic Function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

- Tanh Function

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- Tanh 可以看作是放大并平移的 Logistic，值域为 (-1,1)。

Sigmoid

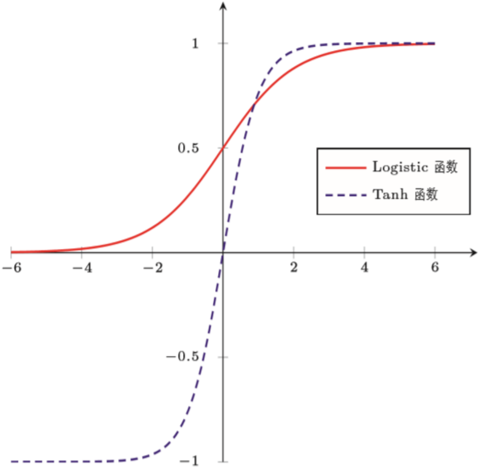


Figure 2: Logistic 函数和 Tanh 函数

Hard Sigmoid

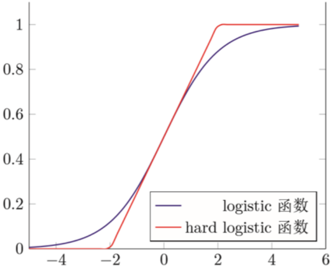
- $exp()$ 的求导计算量较大
- 对 Logistic 函数在 0 附近做一阶泰勒展开

$$
\begin{aligned}
g_l(x) &\approx \sigma(0) + x \times \sigma'(0) \\
&= 0.25x + 0.5
\end{aligned}
$$

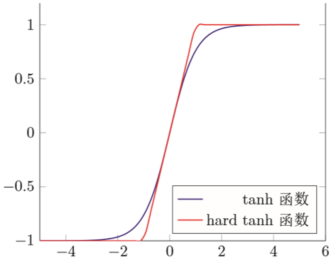
- 此时可以用分段函数 $hard - logistic(x)$ 来近似：

$$
\begin{aligned}
hard - logistic(x) &= \begin{cases} 1 & g_l(x) \geq 1 \\ g_l & 0 < g_l(x) < 1 \\ 0 & g_l(x) \leq 0 \end{cases} \\
&= \max\left(\min\left(g_l(x), 1\right), 0\right) \\
&= \max(\min(0.25x + 0.5, 1), 0)
\end{aligned}
$$

Hard Sigmoid



(a) Hard Logistic 函数



(b) Hard Tanh 函数

Figure 3: Hard Sigmoid 型激活函数

Rectified Linear Unit (ReLU)[1]

- 比 Sigmoid 函数具有更好的稀疏性，大约 50% 的神经元处于激活状态
- $x > 0$ 时导数为 1，一定程度缓解梯度消失问题

$$
\begin{aligned}
\mathrm{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\
&= \max(0, x) \\
\Delta_x \mathrm{ReLU}(x) &= 1(x > 0)
\end{aligned}
$$

- **Dying ReLU Problem**: 如果某种情况下（e.g. 大的梯度更新导致 ReLU 的输入小于 0），部分输入 W 经过 ReLU 函数能得到一个 0（ReLU is close），那么反向传播时，W 一般都不能通过反向传播得到更新。

[1] Nair and Hinton (2010)

ReLU Variant[2]

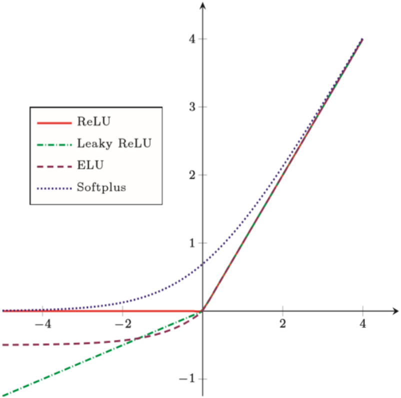


Figure 4: ReLU、Leaky ReLU、ELU 以及 Softplus 函数

[2] Maas (2013); Clevert, Unterthiner, and Hochreiter (2015); Dugas et al. (2000)

Section 3

Network Architectures

Network Architecture

- **前馈网络**
  - 非线性函数的多次复合，整个网络的信息只往一个方向传播，不反向传播
  - 全连接前馈网络、卷积神经网络（CNN）
- 记忆网络
  - 不但可以接收其它神经元的信息，也可以接收自己的历史信息
  - **循环神经网络（RNN）**、Hopfield 网络、玻尔兹曼机（Boltzmann Machine）、受限玻尔兹曼机（RBM）
- 图网络
  - 处理图结构的数据（知识图谱、社交网络、分子网络）
  - 图中每个节点都由一个或一组神经元构成。节点之间的连接可以是有向的，也可以是无向的。每个节点可以收到来自相邻节点或自身的信息。
  - 图卷积神经网络（GCN）、Graph Attention Networks (GAT)、Message Passing Neural Network (MPNN)

Network Architectures

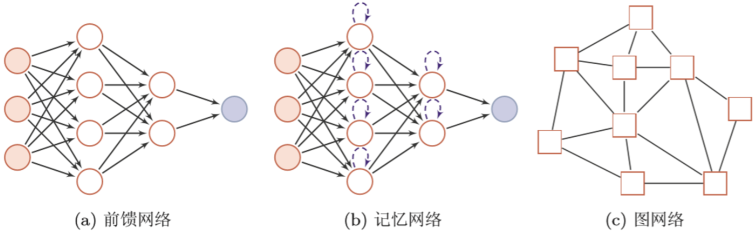


Figure 5: 三种不同的网络结构示例

Universal Approximation Theorem[3]

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R, under mild assumptions on the activation function.

[3] Hornik, Stinchcombe, and White (1989)

Feedforward Neural Network (FNN)

$$a^{(l)} = f_l\left(z^{(l)}\right) = f_l\left(W^{(l)} \cdot a^{(l-1)} + b^{(l)}\right)$$

$$x = a^{(0)} \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow \cdots \rightarrow a^{(L-1)} \rightarrow z^{(L)} \rightarrow a^{(L)} = \phi(x; W, b$$

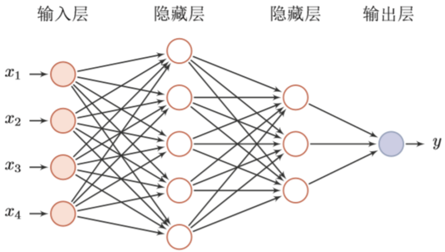


Figure 6: 多层前馈神经网络

Section 4

Back Propagation

Review and Notation

- 梯度下降（Gradient Descent）
  - BGD、SGD、MBGD
  - SGD Variant：SAG、SVRG、OCO, etc.
- 符号说明
  - L：表示神经网络的层数；
  - $m^{(l)}$：表示第 l 层神经元的个数；
  - $f_l(\cdot)$：表示第 l 层神经元的激活函数；
  - $W^{(l)} \in \mathbb{R}^{m^{(l)} \times m^{l-1}}$：表示第 $l-1$ 层到第 l 层的权重矩阵；
  - $b^{(l)} \in \mathbb{R}^{m^l}$：表示第 $l-1$ 层到第 l 层的偏移项；
  - $z^{(l)} \in \mathbb{R}^{m^l}$：表示第 l 层神经元的净输入；
  - $a^{(l)} \in \mathbb{R}^{m^l}$：表示第 l 层神经元的输出；

Loss Function

- 如果使用交叉熵损失函数，对于样本 (x, y)，其损失函数为：

$$\mathcal{L}(y, \hat{y}) = -y^{\mathrm{T}} \log \hat{y}$$

$$\mathcal{R}(W, b) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\left(y^{(n)}, \hat{y}^{(n)}\right) + \frac{1}{2} \lambda \|W\|_F^2$$

$$\|W\|_F^2 = \sum_{l=1}^{L} \sum_{i=1}^{m^{(l)}} \sum_{j=1}^{m^{(l-1)}} \left(w_{ij}^{(l)}\right)^2$$

Gradiant Descend

$$
\begin{aligned}
W^{(l)} &\leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W, b)}{\partial W^{(l)}} \\
&= W^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial \mathcal{L}\left(y^{(n)}, \hat{y}^{(n)}\right)}{\partial W^{(l)}} \right) + \lambda W^{(l)} \right) \\
b^{(l)} &\leftarrow b^{(l)} - \alpha \frac{\partial \mathcal{R}(W, b)}{\partial b^{(l)}} \\
&= b^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}\left(y^{(n)}, \hat{y}^{(n)}\right)}{\partial b^{(l)}} \right)
\end{aligned}
$$

Gradiant Descend

- 对第 l 层中的参数 $W^{(l)}$ 和 $b^{(l)}$ 计算偏导数：

$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_{ij}^{(l)}} = \frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(y,\hat{y})}{\partial z^{(l)}}$$

$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial b^{(l)}} = \frac{\partial z^{(l)}}{\partial b^{(l)}} \frac{\partial \mathcal{L}(y,\hat{y})}{\partial z^{(l)}}$$

Partial Derivative 1

- 计算偏导数 $\frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}}$，因为 $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$，偏导数

$$
\begin{aligned}
\frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}} \right] \\
&= \left[0, \cdots, \quad \frac{\partial \left(w_{i:}^{(l)} a^{(l-1)} + b_i^{(l)} \right)}{\partial w_{ij}^{(l)}}, \cdots, 0 \right] \\
&= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0 \right] \\
&\triangleq \mathbb{I}_i \left(a_j^{(l-1)} \right) \quad \in \mathbb{R}^{m^{(l)}}
\end{aligned}
$$

Partial Derivative 2

- 计算偏导数 $\frac{\partial z^{(l)}}{\partial b^{(l)}}$，因为 $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$，偏导数

$$\frac{\partial z^{(l)}}{\partial b^{(l)}} = I_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

- 求导结果为 $m^{(l)} * m^{(l)}$ 的单位矩阵。

Partial Derivative 3

- 计算偏导数 $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial z^{(l)}} \triangleq \delta^{(l)}$
- 这个值表示第 l 层神经元对最终损失的影响（贡献程度），也反映了最终损失对第 l 层神经元的敏感程度，因此一般称为第 l 层神经元的**误差项**，用 $\delta^{(l)}$ 来表示。
- 根据 $z^{(l+1)} = W^{(l+1)}a^{(l)} + b^{(l+1)}$，有 $\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = \left(W^{(l+1)}\right)^{\mathrm{T}}$
- 根据 $a^{(l)} = f_l\left(z^{(l)}\right)$，其中 $f_l(\cdot)$ 按照位来计算，因此有

$$\begin{aligned}\frac{\partial a^{(l)}}{\partial z^{(l)}} &= \frac{\partial f_l\left(z^{(l)}\right)}{\partial z^{(l)}} \\ &= \operatorname{diag}\left(f_l'\left(z^{(l)}\right)\right)\end{aligned}$$

Partial Derivative 3

- 根据链式法则，第 l 层的误差项为

$$
\begin{aligned}
\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(y,\hat{y})}{\partial z^{(l)}} \\
&= \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \cdot \frac{\partial \mathcal{L}(y,\hat{y})}{\partial z^{(l+1)}} \\
&= \operatorname{diag}\left(f_l'\left(z^{(l)}\right)\right) \cdot \left(W^{(l+1)}\right)^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'\left(z^{(l)}\right) \odot \left(\left(W^{(l+1)}\right)^{\mathrm{T}} \delta^{(l+1)}\right)
\end{aligned}
$$

- 其中 $\odot$ 是向量的点积运算符，表示每个元素相乘；
- 理解反向传播算法：第 l 层的一个神经元的误差项（或敏感性）是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后再乘上该神经元激活函数的梯度。

Partial Derivative

- 计算出上面三个偏导数之后，损失函数对一个权重 $w_{ij}^{(l)}$ 的梯度就可以写成

$$
\begin{aligned}
\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_{ij}^{(l)}} &= \mathbb{I}_i \left(a_j^{(l-1)}\right) \delta^{(l)} \\
&= \left[0,\cdots,a_j^{(l-1)},\cdots,0\right] \left[\delta_1^{(l)},\cdots,\delta_i^{(l)},\cdots,\delta_{m^{(l)}}^{(l)}\right]^{\mathrm{T}} \\
&= \delta_i^{(l)} a_j^{(l-1)}
\end{aligned}
$$

- 其中 $\delta_i^{(l)} a_j^{(l-1)}$ 相当于向量 $\delta^{(l)}$ 和向量 $a^{(l-1)}$ 的**外积**的第 $\{\text{i,j}\}$ 个元素，因此上式进一步写成

$$
\left[\frac{\partial \mathcal{L}(y,\hat{y})}{\partial W^{(l)}}\right]_{ij} = \left[\delta^{(l)} \left(a^{(l-1)}\right)^{\mathrm{T}}\right]_{ij}
$$

Gradient

- $\mathcal{L}(y, \hat{y})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为:

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial W^{(l)}} = \delta^{(l)} \left(a^{(l-1)}\right)^{\mathrm{T}}$$

- $\mathcal{L}(y, \hat{y})$ 关于第 l 层偏移项 $b^{(l)}$ 的梯度为:

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial b^{(l)}} = \delta^{(l)}$$

Back Propagation

算法 4.1: 使用反向传播算法的随机梯度下降训练过程

输入: 训练集 $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, 验证集 $\mathcal{V}$, 学习率 α, 正则化系数 λ, 网络层数 L, 神经元数量 $m^{(l)}$, $1 \le l \le L$.

```
随机初始化 W, b ;
repeat
    对训练集 D 中的样本随机重排序;
    for n = 1 ··· N do
        从训练集 D 中选取样本 (x^(n), y^(n));
        前馈计算每一层的净输入 z^(l) 和激活值 a^(l), 直到最后一层;
        反向传播计算每一层的误差 δ^(l);                      // 公式 (4.63)
        // 计算每一层参数的导数
        ∀l, ∂L(y^(n), ŷ^(n)) / ∂W^(l) = δ^(l) (a^(l-1))^T;    // 公式 (4.68)
        ∀l, ∂L(y^(n), ŷ^(n)) / ∂b^(l) = δ^(l);               // 公式 (4.69)
        // 更新参数
        W^(l) ← W^(l) − α(δ^(l) (a^(l-1))^T + λW^(l));
        b^(l) ← b^(l) − αδ^(l);
    end
until 神经网络模型在验证集 V 上的错误率不再下降;
```

其中第 8–11 行为:

$$\forall l, \quad \frac{\partial \mathcal{L}(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)})}{\partial W^{(l)}} = \delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}; \qquad \text{// 公式 (4.68)}$$

$$\forall l, \quad \frac{\partial \mathcal{L}(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)})}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)}; \qquad \text{// 公式 (4.69)}$$

$$W^{(l)} \leftarrow W^{(l)} - \alpha(\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}} + \lambda W^{(l)});$$

$$\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \alpha\delta^{(l)};$$

输出: $W, \boldsymbol{b}$

Figure 7: 反向传播算法

Section 5

Regularization

Regularization

- 训练数据集上的经验风险最小化和样本真实分布上的**期望风险**并不一致
- 神经网络的拟合能力非常强，其在训练数据上的错误率往往都可以降到非常低，甚至可以到 0，从而导致过拟合
- 正则化：限制模型复杂度
  - 传统机器学习：$l1, l2$ 正则化
  - 深度神经网络：由于过度参数化的问题（over-parameterization），一般使用数据增强（Data Augmentation）、权重衰减（Weight Decay）、提前停止（Early Stop）、**丢弃法**（Dropout）等

Weight Decay[5]

- 在每次参数更新时，引入一个衰减系数

$$\theta_t \leftarrow (1-w)\theta_{t-1} - \alpha \mathbf{g}_t$$

- 在标准的随机梯度下降中，权重衰减正则化和 $l2$ 正则化的效果相同。因此，权重衰减在一些深度学习框架中通过 $l2$ 正则化来实现。但是，在较为复杂的优化方法（比如 Adam）中，权重衰减和 $l2$ 正则化并不等价[4]。

[4] Loshchilov and Hutter (2017)
[5] Hanson and Pratt (1988)

Early Stop

- 在使用梯度下降法进行优化时，我们可以使用一个和训练集独立的样本集合，称为验证集（Validation Set），并用验证集上的错误来代替期望错误。
- 当验证集上的错误率不再下降，就停止迭代[6]。

[6] Prechelt (1998)

Dropout

- 当训练一个深度神经网络时，我们可以随机丢弃一部分神经元（同时丢弃 其对应的连接边）来避免过拟合[7]。
- 对于一个神经层 $y = f(Wx + b)$，我们可以引入一个丢弃函数 $d(\cdot)$，使得 $y = f(Wd(x) + b)$。丢弃函数 $d(\cdot)$ 的定义为

$$\mathrm{d}(x) = \begin{cases} m \odot x & \text{Training Set} \\ px & \text{Test Set} \end{cases}$$

- 其中 $m \in \{0,1\}^d$ 是丢弃掩码（Dropout Mask），通过以概率为 p 的伯努利分布随机生成。

[7] Srivastava et al. (2014); Wan et al. (2013)

Dropout

- 对于隐藏层的神经元，其丢弃率 $p = 0.5$ 时效果最好
- 对于输入层的神经元，其丢弃率通常设为更接近 1 的数，使得输入变化不会太大
- 丢弃法一般是针对神经元进行随机丢弃，也可以扩展到对神经元之间的连接进行随机丢弃，或每一层进行随机丢弃

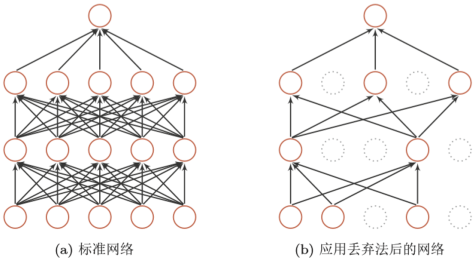


Figure 8: 丢弃法示例

Two Explanations for Dropout

- 集成学习角度
  - 每做一次丢弃，相当于从原始的网络中采样得到一个子网络。如果一个神经网络有 n 个神经元，那么总共可以采样出 2^n 个子网络。
- 贝叶斯学习角度[8]
  - 贝叶斯学习是假设参数 θ 为随机向量，并且先验分布为 $q(\theta)$，贝叶斯方法的预测为

$$\mathbb{E}_{q(\theta)}[y]=\int_{q} f(x;\theta)q(\theta)d\theta \approx \frac{1}{M}\sum_{m=1}^{M} f\left(x,\theta_{m}\right)$$

- 其中 $f(x,\theta_m)$ 为第 m 次应用丢弃方法后的网络，其参数 θ_m 为对全部参数 θ 的一次采样。

[8] Gal and Ghahramani (2016)

Data Augmentation

- 在数据量有限的情况下，可以通过数据增强来增加数据量，提高模型鲁棒性，避免过拟合。
- 主要运用在图像处理中，对图像进行转变，引入噪声等方法来增加数据的多样性。
  - 旋转（rotation)、翻转（flip)、缩放（scale)、裁剪（crop)、平移（translation)、加噪声（noise）[9]

[9] https://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-limited-data-f68c0b512cab

References I

Clevert, Djorkarne, Thomas Unterthiner, and Sepp Hochreiter. 2015. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." *arXiv: Learning*.

Dugas, Charles, Yoshua Bengio, Francois Belisle, Claude Nadeau, and Rene Garcia. 2000. "Incorporating Second-Order Functional Knowledge for Better Option Pricing," 472–78.

Gal, Yarin, and Zoubin Ghahramani. 2016. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," 1050–59.

Hanson, Stephen Jose, and Lorien Y Pratt. 1988. "Comparing Biases for Minimal Network Construction with Back-Propagation," 177–85.

Hornik, Kurt, Maxwell B Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–66.

Loshchilov, Ilya, and Frank Hutter. 2017. "Fixing Weight Decay Regularization in Adam." *CoRR* abs/1711.05101. http://arxiv.org/abs/1711.05101.

References II

Maas, Andrew L. 2013. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." In.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified Linear Units Improve Restricted Boltzmann Machines," 807–14.

Prechelt, Lutz. 1998. "Early Stopping-but When?" *Neural Information Processing Systems*, 55–69.

Srivastava, Nitish, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (1): 1929–58.

Wan, Li, Matthew D Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. "Regularization of Neural Networks Using DropConnect," 1058–66.