Topic 2: Efficient Learning

Inefficiencies in Learning

- Sampling Error and Overfitting
  - What if, the training data is sampled to have a certain pattern that is not present in the original input-output relationship?
- Variance in Stochastic Gradient Descent
  - Randomness in Gradient estimates can introduce a bias that is directly proportional to the variance in the gradient estimate.
- Complexity of Gradient Computation
  - Gradients for each complex model is very tedious to compute.

Can We Mitigate Overfitting?

- **Best Approach:** Get more data!
  - Train on different *bags* of data
- Use the right model...
  - Hard to accomplish... similar to model-based learning.
- Consider **model ensembles**...
  - Use different function classes (models of different forms)
  - Identify multiple weight parameters for different initializations and take their average
- How about **regularization** to limit the capacity of neural networks?
  - Limit the number of hidden layers and/or units per layer.
  - Early stopping criteria in optim algorithms, before overfitting begins
  - Penalize the objective for large weights using ℓ_1 penalty, or ℓ_2 penalty
  - Introduce hard constraint on weight capacity (*a.k.a.* max-norm, i.e. $||\mathbb{W}||_p \leq c$.).

Bootstrap Aggregation (in short, Bagging)[1]

- Reduce generalized error by aggregating the outcomes of several models.
- Generate a bootstrap sample, say $\mathcal{D}_k$, which follows the same distribution as the training set $\mathcal{D}$, for $k = 1, \cdots, K$.
- Train a new classifier, say $\hat{y}_k = \hat{f}_k(x)$, on each bootstrap sample $\mathcal{D}_k$.
- **Aggregation rule:** Count the number of times a class label appears amongst the K classifiers. The label with highest count is returned as output. Ties are broken by choosing the labels with lowest class label.
- Can also aggregate via averaging the outcomes.
- Let ϵ_k denote the error in the k^{th} classifier, with variance $\mathbb{E}(\epsilon_k^2) = v$, and covariance $\mathbb{E}(\epsilon_k \epsilon_j) = c$.

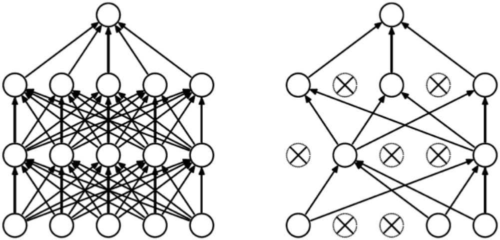
$$\begin{aligned}\mathbb{E}\left[\left(\frac{1}{K}\sum_{k=1}^{K}\epsilon_k\right)^2\right] &= \frac{1}{K^2}\mathbb{E}\left[\sum_{k=1}^{K}\left(\epsilon_k^2 + \sum_{j\neq k}\epsilon_k\epsilon_j\right)\right] \\ &= \frac{1}{K}v + \frac{K-1}{K}c\end{aligned}$$

- If $c = 0$, squared error reduces by a factor of K.

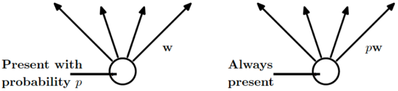
[1] Leo Breiman, "Bagging Predictors," *Machine learning*, vol. 24, no. 2, pp. 123-140, 1996.

Dropout[2]

- Bagging and other ensemble methods is hard, especially with large neural network models.
- **Simple Approach:** Drop neurons with a fixed probability p, during training.
- Aggregate by averaging the weight parameters across different models



(a) Standard Neural Net (b) After applying dropout.



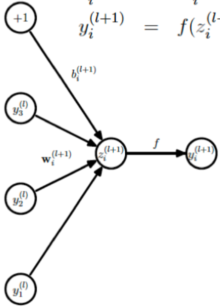
(a) At training time (b) At test time

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

Dropout (cont...)

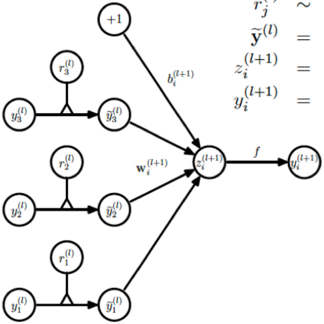
Formally, dropout is modeled as follows...

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$



(a) Standard network

$$
\begin{aligned}
r_j^{(l)} &\sim \mathrm{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$



(b) Dropout network

- **Note:** Backpropagation on thinned networks over several mini-batches of data
- Dropout + Norm-Regularization + AdaM (with large learning rates and high momentum) $\Rightarrow$ Huge boost in performance!

Regularization in Neural Networks

$$\hat{f}_N = \arg\min_{\mathbb{W}} \ L_N(\mathbb{W}) + \lambda \Phi(\mathbb{W}),$$

where $\Phi(\mathbb{W})$ is the term that penalizes undesirable weights.

- **Weight Decay (ℓ_2-Penalty):** $\Phi(\mathbb{W}) = \dfrac{1}{2}||\mathbb{W}||_2^2$
  - Also called ridge regression, or Tikhonov regularization
  - Drives weights closer to zero.
  - Let $\Psi_2(\mathbb{W}) = L_N(\mathbb{W}) + \lambda||\mathbb{W}||_2^2$.
  - Note that $\nabla \Psi_2(\mathbb{W}) = \nabla L_N(\mathbb{W}) + 2\lambda\mathbb{W} \to 0 \ \Rightarrow \ \mathbb{W} \to \dfrac{1}{2\lambda}\nabla L_N(\mathbb{W})$
- **ℓ_1-Penalty:** $\Phi_1(\mathbb{W}) = ||\mathbb{W}||_1$
  - Results in sparse $\mathbb{W}$.
  - Let $\Psi_1(\mathbb{W}) = L_N(\mathbb{W}) + \lambda||\mathbb{W}||_1 \ \Rightarrow \nabla \Psi_1(\mathbb{W}) = \nabla L_N(\mathbb{W}) + 2\lambda \cdot \text{sign}(\mathbb{W}) \to 0$
  - Weight update: $\mathbb{W}^{r+1} = \mathbb{W}^r - \delta \nabla L_N(\mathbb{W}) + 2\lambda\delta \cdot \text{sign}(\mathbb{W})$ (for each entry in $\mathbb{W}$)
    - $\mathbb{W}^{r+0.5} = \mathbb{W}^r - \delta \nabla L_N(\mathbb{W})$
    - If $\mathbb{W}^r > 0$, then $\mathbb{W}^{r+1} = \max\{0, \mathbb{W}^{r+0.5} + 2\lambda\delta\}$
    - If $\mathbb{W}^r < 0$, then $\mathbb{W}^{r+1} = \min\{0, \mathbb{W}^{r+0.5} - 2\lambda\delta\}$

Batch Normalization[3,4]

- **Internal Covariance Shift:** Change in distribution of activations due to change in model parameters – slows training
- **Solution:** Whitening activation functions
  - Linearly transform features to have zero mean and unit variances, i.e. decorrelated.

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

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

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

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

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

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

---

[3] S. Ioffe, and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *International Conference on Machine Learning*, pp. 448-456, 2015.

[4] Ping Luo, Xinjiang Wang, Wenqi Shao, Zhanglin Peng, "Towards Understanding Regularization in Batch Normalization," *ICLR*, 2019.

Batch Normalization (cont...)

Backpropagation for BatchNorm:

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

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

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

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

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

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

Mini-Batch Stochastic Gradient Descent

- Random coordinate descent[5] (RCD) provides same convergence rate as that of full gradient descent
- RCD takes a fraction of effort by evaluating gradients only for one (or a small subset of) random coodinate(s).
- But, computer vision problems require a lot of images for training – RCD is *not* sufficient!
- **Solution:** Mini-batch SGD[6]

$$\left[\nabla L_N(\mathbb{W}^{r-1})\right]_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell\big(y_i, \hat{y}_i | \mathbb{W}^{r-1}\big)$$

  - Estimate the gradient using a small sample (mini-batch) of training data.
  - Introduces bias which is a function of variance in the gradient estimate.
    - Variance in gradient estimate reduces as the size of mini-batch increases.
    - Alternatively, change the learning rate to $\delta_t = \delta \cdot \gamma^t$, where γ is a discounting factor that forces mini-batch SGD to converge.

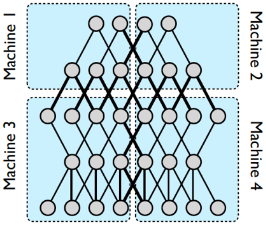
---

[5] Y. Nesterov, “Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems,” *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341-362, 2012.

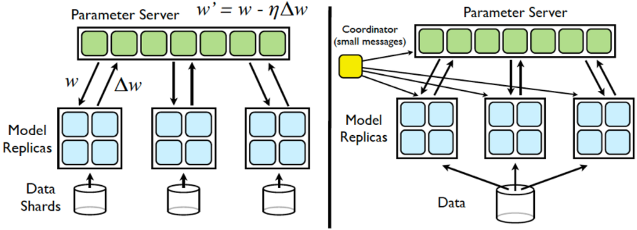
[6] M. Li, T. Zhang, Y. Chen, and A. J. Smola, “Efficient Mini-Batch Training for Stochastic Optimization,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 661-670. 2014.

Can We Take Advantage of Parallel Processing on Hardware?

Model Parallelism: Each model partition is handled/aggregated on a different CPU.



Model Aggregation: Models are trained on each mini-batch on a different GPU.



Source: J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato et al. "Large Scale Distributed Deep Networks," *Advances in neural information processing systems*, vol. 25, 2012.