Deep Learning - 2019

Training 2 - Regularization

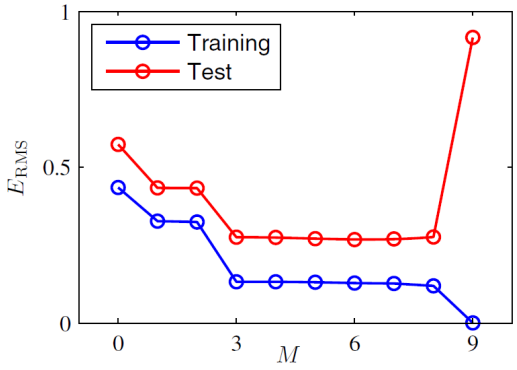
Prof. Avishek Anand

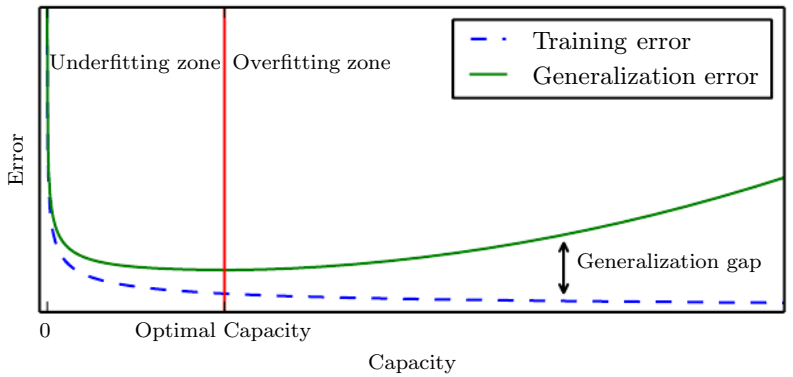
What we learnt before ?

"among competing hypotheses that explain known observations equally well, one should choose the simplest one. "

- How to do effective optimization using initialization, normalization and learning rate regulation.
- Today we talk about how do we do generalization
- Constrain the models or Limit the capacity of a model to avoid over-fitting
  - Larger the hypothesis class, easier to find a hypothesis that has a good trade-off
  - Throw away or penalize useless hypotheses

- Regularization: Limiting the hypothesis in the hypothesis space (reduces the gen. gap)





Regularization

- How do we tackle Under-fitting: Consider a higher model capacity (e.g. higher deg. poly)
- Answer to Overfitting – Regularization or restrict the models

Hard Constraints

$$\min_\theta \mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(\theta, x_i, y_i\right)$$

$$\text{subject to: } R(\theta) \leq r$$

Example – L2 Regularization

$$\min_\theta \mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(\theta, x_i, y_i\right)$$

$$\text{subject to: } \|\theta\|_2^2 \leq r^2$$

Soft Constraints – L2 Reg.

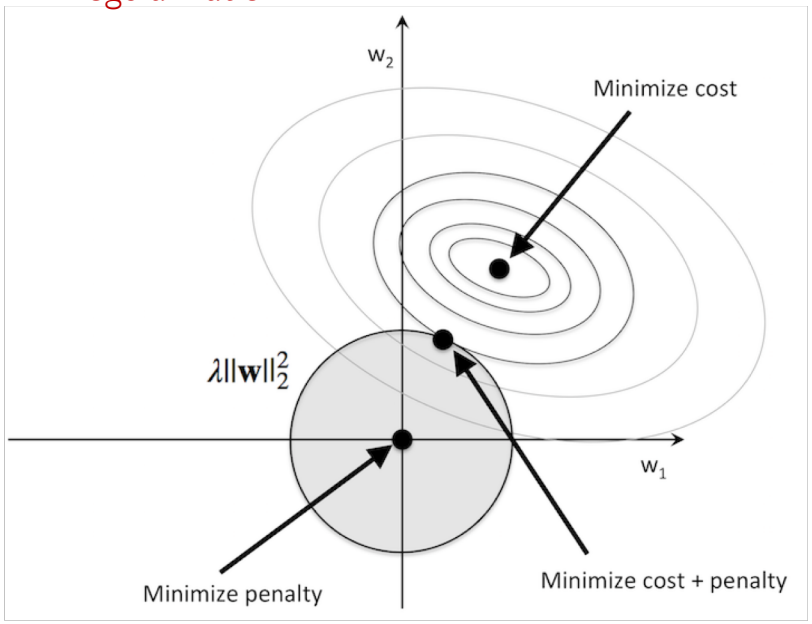
$$\min_\theta \mathcal{L}_{Reg}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(\theta, x_i, y_i\right) + \lambda\|\theta\|_2^2$$

- Regularization restricts the choice of parameters
- L2 Regularization is the most common regularization technique
  - Hard constraints can be translated to softer constraints that are easier to optimize

Regularization: Loss Surface

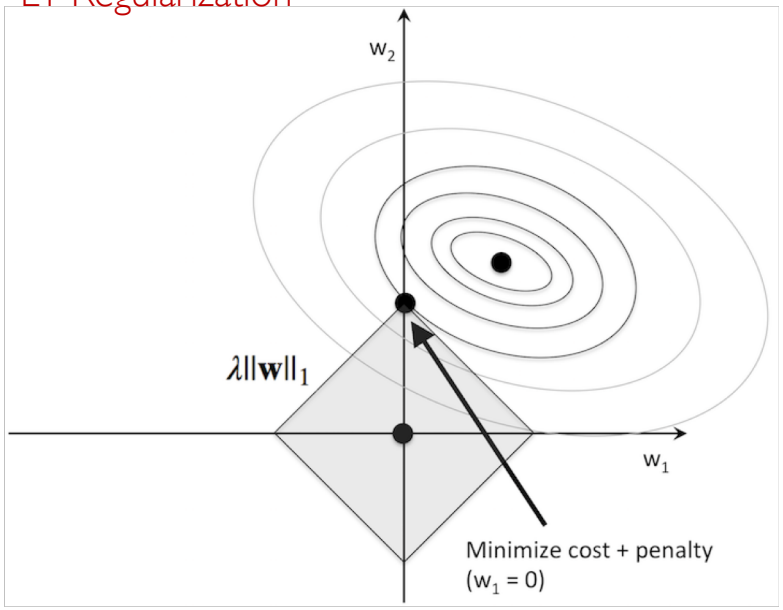
- How does the loss surface look like when there are 2 params for squared loss?

L2 Regularization



$$\min_{\theta} \mathcal{L}_{Reg}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(\theta, x_i, y_i\right) + \lambda\|\theta\|_2^2$$

L1 Regularization



$$\min_{\theta} \mathcal{L}_{Reg}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(\theta, x_i, y_i\right) + \lambda\|\theta\|_1$$

L2 Regularization or Weight Decay

- The L2 regularization penalizes high parameter values

$$\mathcal{L}_{Reg}(\theta; X, y) = \mathcal{L}(\theta; X, y) + \frac{\alpha}{2}\theta^T\theta$$

- Gradients of the cost w.r.t. the weights are

$$\nabla_\theta \mathcal{L}_{Reg}(\theta; X, y) = \nabla_\theta \mathcal{L}(\theta; X, y) + \alpha\theta$$

- Remember gradients are computed through back-propagation
- A simple gradient descent step with a learning rate ε is:

$$\theta \leftarrow \theta - \epsilon \left(\nabla_\theta \mathcal{L}(\theta; X, y) + \alpha\theta\right)$$

L1 Regularization in Neural Networks

- The regularization for L1

$$\begin{aligned}\mathcal{L}_{Reg}(\theta; X, y) &= \mathcal{L}(\theta; X, y) + \lambda \|\theta\|_1 \\ &= \mathcal{L}(\theta; X, y) + \lambda \sum_k |\theta_k|\end{aligned}$$

- Gradients of the cost w.r.t the weights are

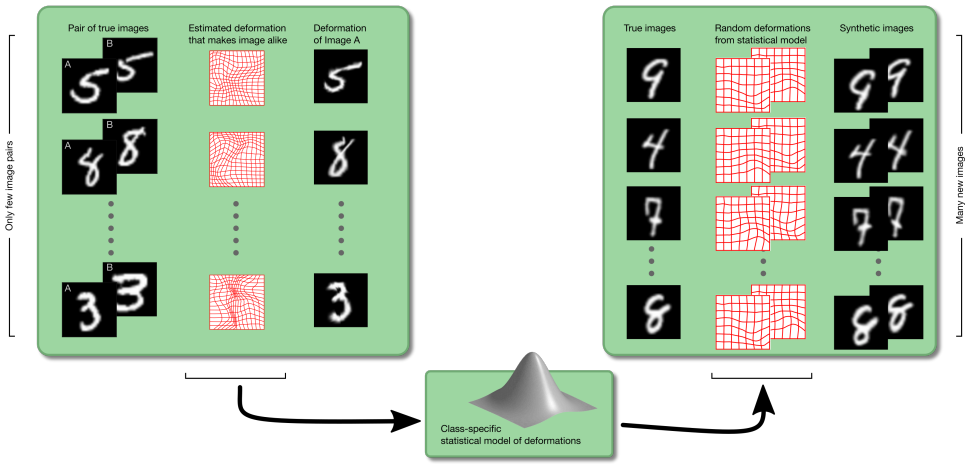
$$\nabla_w \mathcal{L}_{Reg}(\theta; X, y) = \nabla_w \mathcal{L}(\theta; X, y) + \lambda \left(\begin{cases} 1 & \text{if } \theta_k > 0 \\ -1 & \text{if } \theta_k \leq 0 \end{cases} \right)$$

- A simple gradient descent step with a learning rate **ε**

$$\theta \leftarrow \theta - \epsilon \left(\nabla_w \mathcal{L}(\theta; X, y) + \lambda \left(\begin{cases} 1 & \text{if } \theta_k > 0 \\ -1 & \text{if } \theta_k \leq 0 \end{cases} \right) \right)$$

Data Augmentation

- Train the network with more data to improve generalization
- Create ”fake” data by perturbing existing training set instances
- Effective for object recognition:
  - Translation, rotation, scaling of images; or deformation strategies:



Noise Robustness

- Noise to weights reduces over-fitting and is used primarily with recurrent neural networks
- Consider a regression problem:

$$\mathcal{L} = \mathbb{E}_{p(x,y)}\left[(\hat{y}(x) - y)^2\right]$$

- Adding a perturbation to the network weights yield a perturbed prediction, such that:

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{p(x,y,\epsilon_\theta)}\left[\left(\hat{y}_{\epsilon_\theta}(x) - y\right)^2\right] \\ &= \mathbb{E}_{p(x,y,\epsilon_\theta)}\left[\hat{y}_{\epsilon_\theta}(x)^2 - 2y_{\epsilon_\theta}(x)y + y^2\right]\end{aligned}$$

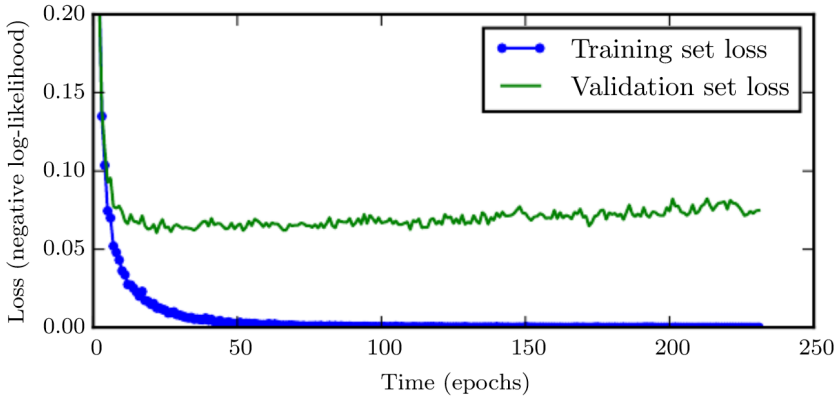
$\epsilon_\theta \sim \mathcal{N}(0, \eta I)$

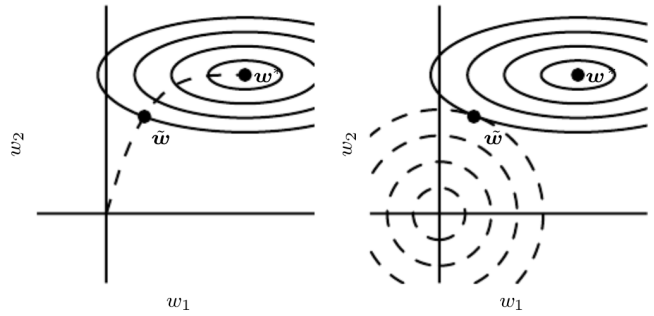
perturbation

- The optimization of this objective for small η is equivalent to adding an additional regularization

$$\eta\mathbb{E}_{p(x,y,\epsilon_\theta)}\left[\|\nabla_\theta \hat{y}(x)\|^2\right]$$

Early Stopping

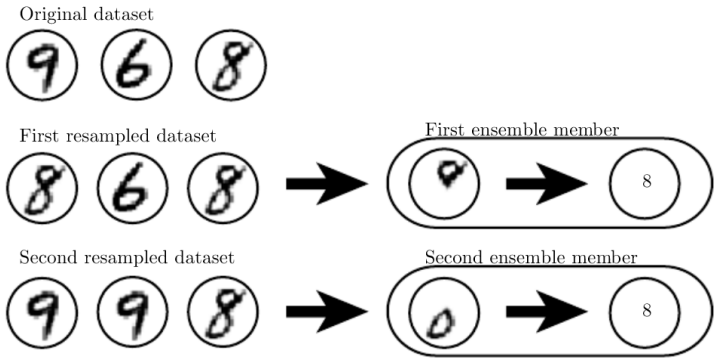




Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Bagging (Bootstrap Aggregation)

- Sample the training dataset with replacement and create subsets
- Learn one model for each subset and then aggregate the predictions of each model
- Also known as ensemble methods with model averaging
- Bagging helps reducing the generalization error



Bagging

- Ensemble models make errors in a regression task
  - Each error is drawn from a multivariate normal distribution of mean 0, and variance c and co-variance v
  - The overall error of an ensemble

$$\frac{1}{k}\sum_{i=1}^{k}\epsilon_i$$

  - The expected squared error of the ensemble is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_{i=1}^{k}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

  - (A) If errors are perfectly correlated, c = v then the squared error is v
  - (B) If errors are perfectly uncorrelated, c = 0 then squared error is v/k
  - In (A) ensemble doesn't help and in (B) the error is reduced linearly

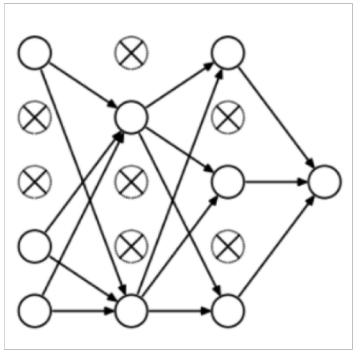
$$\epsilon_i, i = 1, \ldots, k$$

$$\mathbb{E}\left[\epsilon_i^2\right] = v$$

$$\mathbb{E}\left[\epsilon_i\epsilon_j\right] = c$$

Dropout

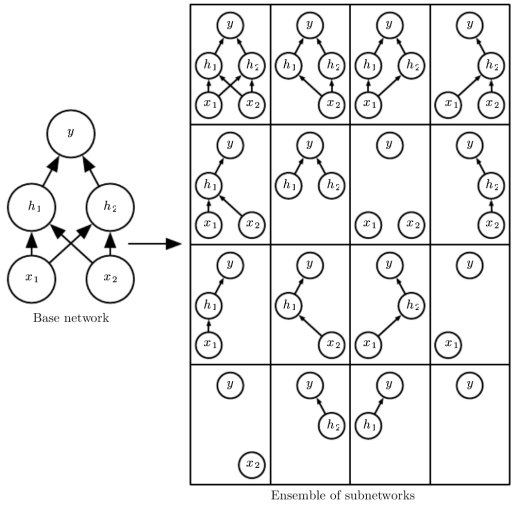
- Dropout a node by multiplying its output by zero
- Only input and hidden nodes are dropped out



- Minibatch-based learning
  - for each batch of training instances we sample different **binary masks** for input/hidden units

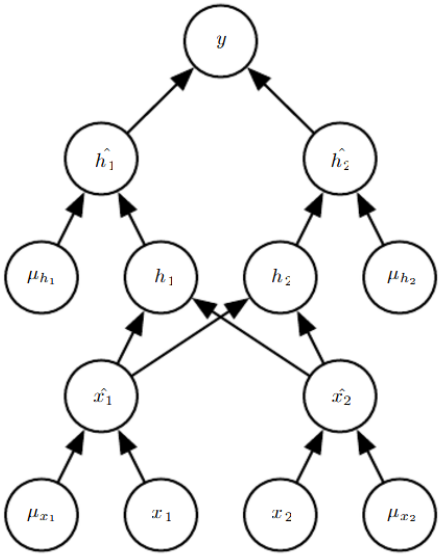
- Typically an input unit is included with a probability of 0.8 and hidden unit with a probability of 0.5

- Dropout as ensembles with shared weights



Dropout – forward computation

- For every mini-batch of the training set
- For input layer sample drop-out masks μ (0) ~Bernoulli(p_{input})N
- For hidden layer l = 1, . . . , L sample μ (l) ~ Bernoulli(p_{hidden})N_l



$$h^{(1)} = g^{(1)}\left(W^{(1)^T}\left(x \bigotimes \mu^{(0)}\right) + b^{(1)}\right)$$

$$h^{(2)} = g^{(2)}\left(W^{(2)^T}\left(h^{(1)} \bigotimes \mu^{(1)}\right) + b^{(2)}\right)$$

$$h^{(L)} = g^{(L)}\left(W^{(L)^T}\left(h^{(L-1)} \bigotimes\right)\mu^{(L-1)}\right) + b^{(L)})$$

$\bigotimes$ Element wise multiplication

Dropout -- BackProp and inference

- When running backpropagation multiply gradients by masks

$$\frac{\partial \mathcal{L}\left(y, h^{(L)}\right)}{\partial W_{j,i}^{(I)}} = \frac{\partial \mathcal{L}\left(y, h^{(L)}\right)}{\partial a_i^{(I)}} \frac{\partial a_i^{(I)}}{\partial W_{j,i}^{(I)}}$$

$$= \frac{\partial \mathcal{L}\left(y, h^{(L)}\right)}{\partial a_i^{(I)}} h_j^{(I-1)} \mu_j^{(I-1)} \quad = \frac{\partial \mathcal{L}\left(y, h^{(L)}\right)}{\partial a_i^{(I)}} \begin{cases} h_j^{(I-1)} \mu_j^{(I-1)} & \text{if } I-1 > 0 \\ x_j \mu_j^{(0)} & \text{if } I-1 = 0 \end{cases}$$

- Use weight scaling for inference
  - it is not feasible to explicitly average the predictions from exponentially many thinned models
  - The network is used as a whole
  - The weights are scaled-down by a factor of p

References

- Deep Learning book – Chapter 7