Regularization for Deep Learning

Dr. Josif Grabocka

ISMLL, University of Hildesheim

Deep Learning

Regularization

- Limit the capacity of a model to avoid over-fitting
- Extend the cost-function by adding a penalization term

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

- $\alpha \in [0, \infty)$ is also known as the regularization penalty
- Regularize the neuron weights, but not the bias terms
- For simplicity use the same α for all layers

Motivating Regularization

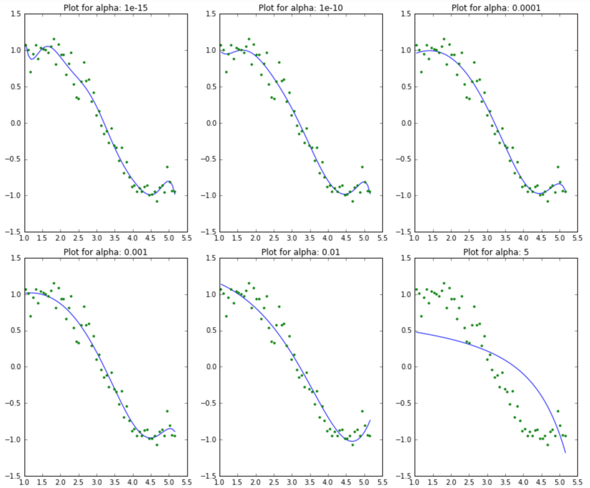


Figure 1: Regularizing polynomial regression (order 15), Source www.analyticsvidhya.com

L^2 Regularization

- The L^2 regularization penalizes high w values

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{\alpha}{2} w^T w$$

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

$$\nabla_w \tilde{J}(w; X, y) = \nabla_w J(w; X, y) + \alpha w$$

- Remember $\nabla_w J(w; X, y)$ is computed through back-propagation

- A simple gradient descent step with a learning rate ϵ is:

$$w \leftarrow w - \epsilon \left(\nabla_w J(w; X, y) + \alpha w \right)$$

L^1 Regularization

- The L^1 regularization:

$$\begin{aligned}\tilde{J}(w;X,y) &= J(w;X,y) + \alpha ||w||_1 \\ &= J(w;X,y) + \alpha \sum_k |w_k|\end{aligned}$$

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

$$\nabla_w \tilde{J}(w;X,y) = \nabla_w J(w;X,y) + \alpha \left(\begin{cases} 1 & \text{if } w_k > 0 \\ -1 & \text{if } w_k \leq 0 \end{cases} \right)$$

- A simple gradient descent step with a learning rate ϵ is:

$$w \leftarrow w - \epsilon \left(\nabla_w J(w;X,y) + \alpha \left(\begin{cases} 1 & \text{if } w_k > 0 \\ -1 & \text{if } w_k \leq 0 \end{cases} \right) \right)$$

L1 and L2 Regularizations - Illustration of the Principle

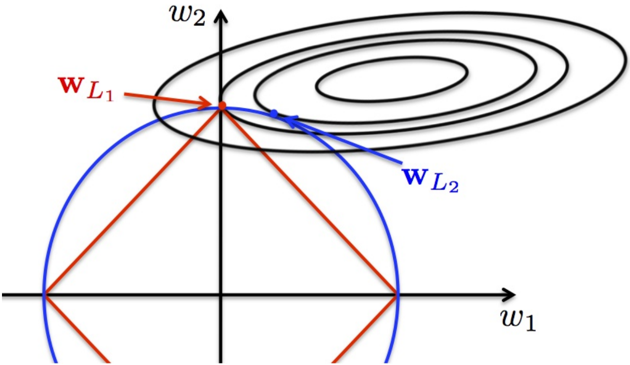


Figure 2: Competing objective terms. i) the blue line represents the L1 regularization, ii) the red line represents the L1 regularization, while iii) solid lines represent the cost function. Source: `g2pi.tsc.uc3m.es`

Constraint Optimization

- The standard regularized objective:

$$\tilde{J}(\theta; X, y) \quad = \quad J(\theta; X, y) + \alpha\Omega(\theta)$$

- A constrained problem forces $\Omega(w) < k$ as:

$$\tilde{J}(\theta, \alpha; X, y) \quad = \quad J(\theta; X, y) + \alpha\left(\Omega(\theta) - k\right)$$

- The solution is by deriving a new objective:

$$\theta^* \quad = \quad \operatorname*{argmin}_{\theta} \max_{\alpha, \alpha \geq 0} \left[J(\theta; X, y) + \alpha\left(\Omega(\theta) - k\right)\right]$$

Dataset Augmentation (Noise to Input)

- 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:

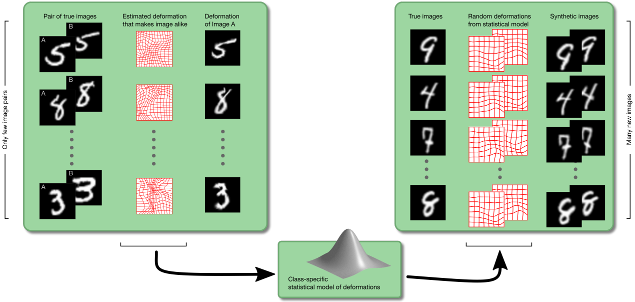


Figure 3: An illustrative strategy for digit images augmentation, Source: compute.dtu.dk

Noise Robustness (Noise to Weights)

- Noise to weights reduces over-fitting and is used primarily with recurrent neural networks

- Consider a regression problem:

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

- Adding a perturbation $\epsilon_w \sim \mathcal{N}(0, \eta I)$ to the network weights yield a perturbed prediction $\hat{y}_{\epsilon_w}(x)$, such that:

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

- The optimization of this objective for small η is equivalent to adding an additional regularization $\eta\mathbb{E}_{p(x,y,\epsilon_w)}\left[||\nabla_w \hat{y}(x)||^2\right]$

Early Stopping - Motivation

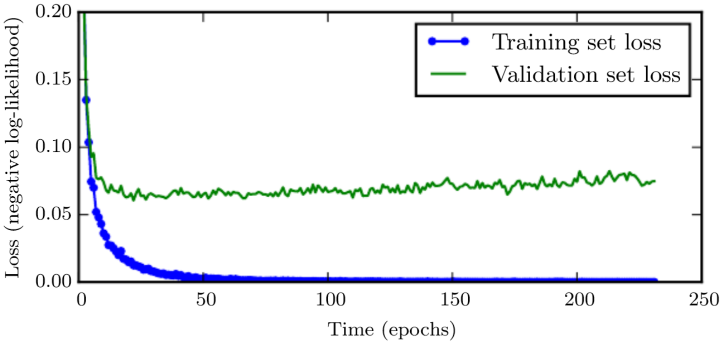


Figure 4: There is a better generalization in the earlier epochs of the optimization, Source: Goodfellow et al., 2016

Early Stopping (Source: Goodfellow et al., 2016)

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^*

Early Stopping as a Regularizer

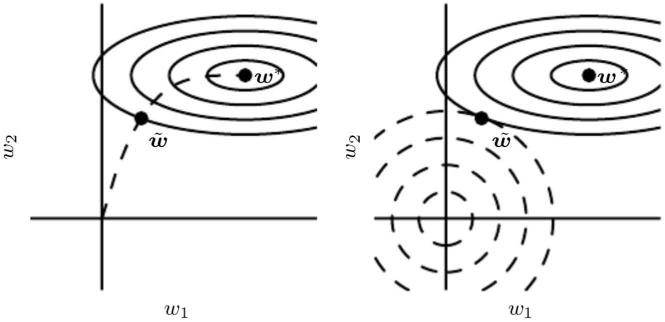


Figure 5: Effect of early stopping (left) on the parameter weights, compared to L2 regularization (right). Source: Goodfellow et al., 2016

Bagging (Boostrap Aggregating)

- 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

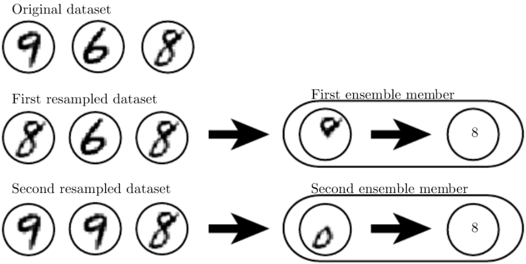


Figure 6: Classify 8-vs-others in digit recognition, Source: Goodfellow et al., 2016

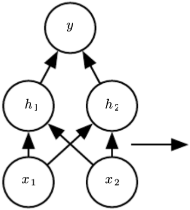
Understanding Bagging

- Ensemble models make errors $\epsilon_i, i = 1, \ldots, k$ in a regression task:
  - Each ϵ_i is drawn from a multivariate normal distribution with mean 0, variance $\mathbb{E}\left[\epsilon_i^2\right] = v$ and covariances $\mathbb{E}\left[\epsilon_i \epsilon_j\right] = c$
- The overall error of an ensemble is $\frac{1}{k}\sum_{i=1}^{k} \epsilon_i$
- The expected squared error of the ensemble is

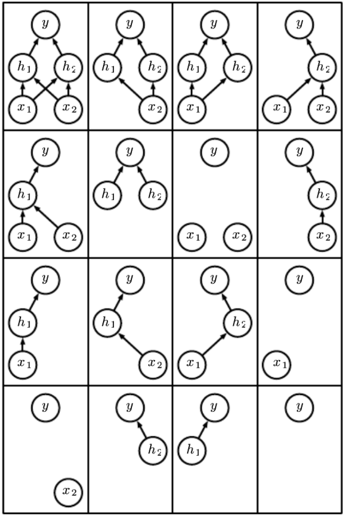
$$\mathbb{E}\left[\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_{i=1}^{k}\left(\epsilon_i^2 + \sum_{j \neq i}\epsilon_i\epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

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

Dropout - Bagging of random neural subnetworks



Base network



Ensemble of subnetworks

Dropout Mechanism

- 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

- Compute back-propagation as usual but multiplying the activations by the mask

Dropout - Forward computations (i)

- For every mini-batch of the training set,
- For input layer sample drop-out masks $\mu^{(0)} \sim \text{Bernoulli}(p_{\text{input}})_N$
- For hidden layer $l = 1, \ldots, L$ sample $\mu^{(l)} \sim \text{Bernoulli}(p_{\text{hidden}})_{N_l}$

$$
\begin{aligned}
h^{(1)} &= g^{(1)}\left(W^{(1)^T}\left(x \bigotimes \mu^{(0)}\right) + b^{(1)}\right) \\
h^{(2)} &= g^{(2)}\left(W^{(2)^T}\left(h^{(1)} \bigotimes \mu^{(1)}\right) + b^{(2)}\right) \\
&\cdots \\
h^{(L)} &= g^{(L)}\left(W^{(L)^T}\left(h^{(L-1)} \bigotimes \mu^{(L-1)}\right) + b^{(L)}\right)
\end{aligned}
$$

where $\bigotimes$ is the element-wise multiplication.

Dropout - Forward computation (ii)

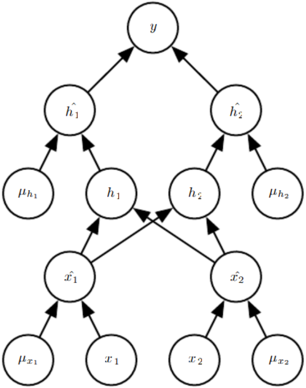


Figure 7: Illustrating drop-out masks, Source: Goodfellow et al., 2016

Dropout - Back-propagation

When running backpropagation mulptiply gradients by the masks:

$$
\begin{aligned}
\frac{\partial \mathcal{L}(y, h^{(L)})}{\partial W_{j,i}^{(l)}} &= \frac{\partial \mathcal{L}(y, h^{(L)})}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{j,i}^{(l)}} \\
&= \frac{\partial \mathcal{L}(y, h^{(L)})}{\partial a_i^{(l)}} h_j^{(l-1)} \mu_j^{(l-1)} \\
&= \frac{\partial \mathcal{L}(y, h^{(L)})}{\partial a_i^{(l)}} \begin{cases} h_j^{(l-1)} \mu_j^{(l-1)} & \text{if } l-1 > 0 \\ x_j \mu_j^{(0)} & \text{if } l-1 = 0 \end{cases}
\end{aligned}
$$

Remember $\frac{\partial \mathcal{L}(y, h^{(L)})}{\partial a_i^{(l)}}$ are stored during back-propagation

Dropout - Inference

Use the weight scaling rule for inference:

$$
\begin{aligned}
h^{(1)} &= g^{(1)}\left(W^{(1)^T}\left(x \bigotimes p_{\vec{\text{input}}_N}\right) + b^{(1)}\right) \\
h^{(2)} &= g^{(2)}\left(W^{(2)^T}\left(h^{(1)} \bigotimes p_{\vec{\text{hidden}}_{N_1}}\right) + b^{(2)}\right) \\
&\cdots \\
h^{(L)} &= g^{(L)}\left(W^{(L)^T}\left(h^{(L-1)} \bigotimes p_{\vec{\text{hidden}}_{N_{L-1}}}\right) + b^{(L)}\right)
\end{aligned}
$$