Deep Learning

Regularization

Ali Ghodsi

University of Waterloo

Slides are based on Book in preparation, Deep Learning

by Bengio, Goodfellow, and Aaron Courville, 2015

Regularization

A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs.

Most machine learning tasks are estimation of a function $\hat{f}(\mathbf{x})$ parametrized by a vector of parameters θ.

Classical Regularization: Parameter Norm Penalty

Most classical regularization approaches are based on limiting the capacity of models, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J.

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

L^2 Parameter Regularization, weight decay

L^2 parameter norm penalty commonly known as *weight decay*.

Regularization term $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$

Gradient of the total objective function:

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

$$w := w - \epsilon(\alpha w + \nabla w J(w; X, y)).$$

Considering a quadratic approximation to the objective function

$$\hat{J}(\theta) = J(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*)$$

$$\nabla_w \hat{J}(w) = H(w - w^*).$$

$$\alpha w + H(w - w^*) = 0$$

$$(H + \alpha I)w = Hw^*$$

$$\tilde{w} = (H + \alpha I)^{-1} H w^*.$$

what happens as α grows?

$$H = Q\Lambda Q^T$$

$$\begin{aligned}
\tilde{w} &= (Q\Lambda Q^T \alpha I)^{-1} Q\Lambda Q^T w^* \\
&= [Q(\Lambda + \alpha I)Q^T]^{-1} Q\Lambda Q^T w^* \\
&= Q(\Lambda + \alpha I)^{-1} \Lambda Q^T w^*, \\
Q^T \tilde{w} &= (\Lambda + \alpha I)^{-1} \Lambda Q^T w^*.
\end{aligned}$$

The effect of weight decay is to rescale the coefficients of eigenvectors. The ith component is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$.

- If $\lambda_i \gg \alpha$, the effect of regularization is relatively small.
- Components with $\lambda_i \ll \alpha$ will be shrunk to have nearly zero magnitude.

Directions along which the parameters contribute significantly to reducing the objective function are preserved a small eigenvalue of the Hessian tell us that movement in this direction will not significantly increase the gradient
effective number of parameters, defined to be

$$\gamma = \sum_i \frac{\lambda_i}{\lambda_i + \alpha}.$$

As α is increased, the effective number of parameters decreases.

Dataset Augmentation

The best way to make a machine learning model generalize better is to train it on more data.

The amount of data we have is limited.

Create fake data and add it to the training set.

Not applicable to all tasks.

For example, it is difficult to generate new fake data for a density estimation task unless we have already solved the density estimation problem.

Dataset Augmentation

Operations like translating the training images a few pixels in each direction can often greatly improve generalization.

One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.

This same approach also works when the noise is applied to the hidden units, which can be seen as doing dataset augmentation at multiple levels of abstraction.

Noise injection

Two ways that noise can be used as part of a regularization strategy.

Adding noise to the input.

This can be interpreted simply as form of dataset augmentation.

Can also interpret it as being equivalent to more traditional forms of regularization.

Noise injection

Two ways that noise can be used as part of a regularization strategy.

Adding noise to the input.

This can be interpreted simply as form of dataset augmentation.

Can also interpret it as being equivalent to more traditional forms of regularization.

Adding it to the weights.

This technique has been used primarily in the context of recurrent neural networks (Jim et al., 1996; Graves, 2011a).

This can be interpreted as a stochastic implementation of a Bayesian inference over the weights.

Manifold Tangent Classifier

It is assumed that we are trying to classify examples and that examples on the same manifold share the same category.

The classifier should be invariant to the local factors of variation that correspond to movement on the manifold.

Use as nearest-neighbor distance between points $\mathbf{x}_1$ and $\mathbf{x}_2$ the distance between the manifolds M_1 and M_2 to which they respectively belong.

Manifold Tangent Classifier

Approximate M_i by its tangent plane at $\mathbf{x}_i$ and measure the distance between the two tangent.

Train a neural net classifier with an extra penalty to make the output $f(x)$ of the neural net locally invariant to known factors of variation (Tangent-Prop algorithm, Simard et al., 1992).

Manifold Tangent Classifier

Require $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$ to be orthogonal to the known manifold tangent vectors v_i at $\mathbf{x}$.

Or the directional derivative of f at $\mathbf{x}$ in the directions v_i be small.

$$regulaizer = \lambda \sum_i \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} . v_i \right)^2$$

A recent paper introduces the Manifold Tangent Classifier (Rifai et al.,2011), which eliminates the need to know the tangent vectors a priori.

Injecting Noise at the Input

Consider a regression setting

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

$\epsilon \sim (0, \nu I)$,

$$\begin{aligned}
\tilde{J}_x &= \mathbb{E}_{p(x,y\epsilon)}[(\hat{y}(x+\epsilon) - y)^2] \\
&= \mathbb{E}_{p(x,y\epsilon)}[\hat{y}^2(x+\epsilon) - 2y\hat{y}(x+\epsilon) + y^2] \\
&= \mathbb{E}_{p(x,y\epsilon)}[\hat{y}^2(x+\epsilon)] - 2\mathbb{E}_{p(x,y\epsilon)}[y\hat{y}(x+\epsilon)] + \mathbb{E}_{p(x,y\epsilon)}[y^2]
\end{aligned}$$

Assuming small noise, we can consider the Taylor series expansion of $\hat{y}(x+\epsilon)$ around $\hat{y}(x)$.

$$\hat{y}(x+\epsilon) = \hat{y}(x) + \epsilon^T \nabla x \hat{y}(x) + \frac{1}{2}\epsilon^T \nabla_x^2 \hat{y}(x)\epsilon + 0(\epsilon^3)$$

$$\mathbb{E}_{p(\epsilon)}[\epsilon] = 0$$

$$\mathbb{E}_{p(\epsilon)}[\epsilon\epsilon^T] = \gamma I$$

$$
\begin{aligned}
\hat{J}_x &\approx \mathbb{E}_{p(x,y,\epsilon)}\left[\left(\hat{y}(x) + \epsilon^T \nabla_x \hat{y}(x) + \frac{1}{2}\epsilon^T \nabla_x^2 \hat{y}(x)\epsilon\right)^2\right] \\
&- 2\mathbb{E}_{p(x,y,\epsilon)}\left[y\hat{y}(x) + y\epsilon^T \nabla_x y\hat{y}(x) + \frac{1}{2}y\epsilon^T \nabla_x^2 \hat{y}(x)\epsilon\right] + \mathbb{E}_{p(x,y,\epsilon)}[y^2] \\
&= \mathbb{E}_{p(x,y,\epsilon)}[(\hat{y}(x) - y)^2] + \mathbb{E}_{p(x,y,\epsilon)}\left[\hat{y}(x)\epsilon^T \nabla_x^2 \hat{y}(x)\epsilon + (\epsilon^T \nabla_x \hat{y}(x))^2 + 0(\epsilon^3)\right] \\
&- 2\mathbb{E}_{p(x,y,\epsilon)}\left[\frac{1}{2}y\epsilon^T \nabla_x^2 \hat{y}(x)\epsilon\right] \\
&= j + \gamma\mathbb{E}_{p(x,y,\epsilon)}[(\hat{y}(x) - y)\nabla_x^2 \hat{y}(x)] + \gamma\mathbb{E}_{p(x,y,\epsilon)}[\|\nabla_x \hat{y}(x)\|^2]
\end{aligned}
$$

Consider minimizing this objective function, by taking the functional gradient of $\hat{y}(x)$ and setting the result to zero

$$\hat{y}(x) = \mathbb{E}_{p(y|x)}[y] + 0(\nu)$$

$$\mathbb{E}_{p(x,y,\epsilon)}[(\hat{y}(x) - y)\nabla_x^2 \hat{y}(x)],$$

reduces $0(\nu)$

For small ν, the minimization of J with added noise on the input (with covariance νI) is equivalent to minimization of J with an additional regularization term given by $\nu \mathbb{E}_{p(x,y,\epsilon)}[\|\nabla_x y(x)\|^2]$. it has the effect of penalizing large gradients of the function $\hat{y}(x)$.
it has the effect of reducing the sensitivity of the output of the network with respect to small variations in its input x.
for linear networks, this regularization term reduces to simple weight decay

Early Stopping as a Form of Regularization

Instead of running our optimization algorithm until we reach a (local) minimum of validation error:

Run it until the error on the validation set has not improved for some amount of time.

Every time the error on the validation set improves, we store a copy of the model parameters.

When the training algorithm terminates, we return these parameters, rather than the latest parameters.

It is probably the most commonly used form of regularization in deep learning.

Early Stopping as a Form of Regularization

Assume $\theta = w$

Take a quadratic approximation to the objective function J in the neighborhood of the empirically optimal value of the weights w^*.

$$\hat{J}(\theta) = J(w^*) + \frac{1}{2}(w - w^*)^T H(\theta - \theta)$$

$$\nabla_w \hat{J}(s) = H(w - w^*).$$

$w^{(0)} = 0$

$$
\begin{aligned}
w^{(\tau)} &= w^{(\tau+1)} - \eta \nabla_w J\big(w^{(\tau+1)}\big) \\
&= w^{\tau+1} - \eta H\big(w^{(\tau+1)} - w^*\big) \\
w^{(\tau)} - w^* &= (I - \eta H)\big(w^{(\tau-1)} - w^*\big)
\end{aligned}
$$

$H : H = Q \Lambda Q$

$$
\begin{aligned}
w^{(\tau)} - w^* &= (I - \eta Q \Lambda Q^T)\big(w^{(\tau+1)} - w^*\big) \\
Q^T\big(w^{(\tau)} - w^*\big) &= (I - \eta \Lambda) Q^T \big(w^{(\tau+1)} - w^*\big)
\end{aligned}
$$

Assuming $w^0 = 0$, and that $|1 - \eta\lambda_i| < 1$, we have after τ training updates,

$$Q^T w^{(\tau)} = [I - (I - \eta\Lambda)^\tau] Q^T w^*.$$

$$Q^T \tilde{w} = (\Lambda + \alpha I)^{-1} \Lambda Q^T w^*$$
$$Q^T \tilde{w} = [I - (\Lambda + \alpha I)^{-1} \alpha] \Lambda Q^T w^*.$$

$$(I - \eta\Lambda)^\tau = (\Lambda + \alpha I)^{-1} \alpha$$

,

by taking logs and using the series expansion for $log(1+x)$, we can conclude that if all λ_i are small (i.e. $\eta\lambda_i \ll 1$ and $\lambda_i/\alpha \ll 1$) then

$$\tau \approx \frac{1}{\eta\alpha},$$

$$\alpha \approx \frac{1}{\tau\eta}.$$

the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\eta$ plays the role of the weight decay coefficient.

Parameter Tying and Parameter Sharing

Sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.

Certain parameters should be close to one another

Parameter Tying and Parameter Sharing

Consider model a with parameters $w^{(a)}$ and model b with parameters $w^{(b)}$.

Suppose the two models map the input to two different, but related outputs: $\hat{y}_a = f(w^{(a)}, x)$ and $\hat{y}_b = g(w^{(b)}, x)$.

Imagine that the tasks are similar enough (perhaps with similar input and output distributions) that we believe the model parameters should be close to each other, i.e. $\forall_i, w_i^{(a)}$ should be close to $w^{(b)}$

use a parameter norm penalty of the form:
$\Omega(w^{(a)}, w^{(b)}) = \|w^{(a)} - w^{(b)}\|_2^2$.

Bagging and Other Ensemble Methods

Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models (Breiman, 1994).

Train several different models separately, then have all of the models vote on the output for test examples.

This is an example of a general strategy in machine learning called model averaging.

Techniques employing this strategy are known as ensemble methods.

Bagging and Other Ensemble Methods

Consider for example a set of k regression models.

Suppose that each model makes an error ϵ_i on each example.

Suppose the errors drawn from a zero-mean multivariate normal distribution with:

variances $\mathbb{E}[\epsilon_i^2] = \nu$

and covariances $\mathbb{E}[\epsilon_i \epsilon_j] = c$.

Bagging and Other Ensemble Methods

Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.

The expected squared error of the ensemble predictor is

$$\mathbb{E}[\frac{1}{k}\sum_i \epsilon_i^2]$$

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

$$\frac{1}{k}\nu + \frac{k-1}{k}c.$$

Bagging and Other Ensemble Methods

$$\frac{1}{k}\nu + \frac{k-1}{k}c.$$

the errors are perfectly correlated and $c = \nu$, this reduces to ν, and the model averaging does not help at all.

The errors are perfectly uncorrelated and $c = 0$, then the expected squared error of the ensemble is only $\frac{1}{k}\nu$.

On average,the ensemble will perform at least as well as any of its members

If the members make independent errors, the ensemble will perform significantly better than its members.