regularization for deep learning

- the goal here is to improve generalization, even if we pay a small cost in training set error.
- if you think of the model training as a statistical estimation process, then regularization can be interpreted as reducing variance at the cost of bias → we are now biased towards some values.
- for most applications of deep learning, the true data-generating process is almost certainly outside the model family → for domains like images, audio sequences etc, the true data generating process involves simulating the entire universe.
- we are going for a large model that has been regularized adequately.
- Common parameter norm penalties are L^1 and L^2 norms. the second has the effect of reducing the weight by a fixed fraction of the weight on each iteration. If you take Hessians and work out the math, we will see by observing the eigendecomposition of the symmetric hessian matrix that L2 regularization reduces the weights in directions corresponding to the biggest eigenvalues of the hessian matrix most and reduces the change in directions along which the eigenvalues of the Hessian are less.
- In contrast, on doing the math we see that L1 regularization reduces the size of the weights by a constant, driving some to 0. This results in sparse weights → hence used as a feature selection mechanism.
- We can also use norm penalties as a constrained optimization problem. If we use a L2 constraint, then this is the same as restricting the weights to a norm ball. We usually do this if we have prior knowledge about what the norm ball should be that the weights should lie in. This can be shown to be a form of early stopping → the intuition is that in early stopping, the parameters can't get too far away from where they initially started, thus staying within some norm ball.
- Some problems and some ML algorithms involve inverting the $X^T X$ matrix and in these cases, sometimes the matrix is not invertible because the data generating process may not have any variance in some direction. in these cases, we invert $X^T X + \alpha I$ instead. This regularized matrix is invertible. This is like stabilizing underdetermined problems.
- dataset augmentation: for example, in images, we can introduce noise, rotations, transformations etc when training a net for classification problems. in speech, we can add a small amount of random noise to our training data. this both increases the training set size but also builds invariance to the things our classifier should be invariant to.
- another powerful technique is to introduce this noise to the hidden units' weights. in this case, what happens is that we add noise the learnt concepts, not just to pixels of an image and make the model robust to these learnt concept changes.
- another strategy is to sample this noise on the weights of hidden units. this pushes the model into regions where the model is relatively insensitive to small variations in the weights.
- another strategy is to do label smoothing: here, we train the model so that the output layer should match a probability $1 - \epsilon$ for the probability of the right label and $\frac{\epsilon}{k-1}$ for the other $k - 1$ labels. Maximum likelihood learning with a softmax classifier and hard targets may actually never converge - the softmax can never predict a probability of exactly 0 or 1, so it will continue to learn larger and larger weights, making it less robust and more gullible to adversarial attack.
- Semi-supervised learning (where not all examples have labels) and multitask learning with shared parameters for certain layers can also be viewed as regularization strategies.

- early stopping is the process of monitoring error on validation set and stopping when it stops going down after a few steps. this is simple and works very well.
- once we've used a validation set for early stopping, we can then retrain on the whole dataset.
- CNNs (which we will see later) are an example of parameter sharing. parameter sharing is when we have a belief that some parameters should share the same value because of some reason. the reason this invariant is expressed in CNNs will become clear when we study CNNs.
- We can also add L1 norm penalties to learnt representations. what this does is to induce representational sparsity.
- Bagging is a set of methods where we train a set of models, instead of one model. The idea is to have these models make independent errors and vote on the answer while running an inference. what this ends up doing is that hopefully they cancel out small errors and the true answer gets the most votes. To do bagging, it is good to construct k datasets (each with same size as original dataset) by sampling with replacement from the original dataset. This results in some repeats and some missing examples, encouraging independent errors.
- While benchmarking new approaches, one shouldn't inflate goodness of it by using bagging and model averaging methods.
- Dropout is a method where we approximate training an ensemble of neural nets efficiently by dropping out hidden units at random. we drop out hidden units with probability 0.5 and we dropout input units with probability 0.8. this forces the model to learn more general factors of variation, discouraging overfitting. during inference we divide each unit's value by 2 (since we dropped each unit with probability 0.5).
- Because we drop hidden units, we destroy extracted features rather than just original values. this makes us learn more stable, associations instead of overfitting.
- adversarial training is making up adversarial versions of the training example (there are methods to do this) and to add these to training set so as to become robust to these. This is like becoming robust to a local area in the input space.
- By adding a small amount of noise ϵ, we can show that the output from a weight can change by as much as $\epsilon||\mathbf{w}||$ where $||\mathbf{w}||$ is the weight vector.
- The tangent prop algorithm makes the model explicitly invariant to changes in input along a manifold by adding a penalty for the $\sum_i \nabla_x f(x)^T v_i$ term where we want to make it invariant to the vectors v_i. How do we know these vectors? either by some prior knowledge about the space or by training an autoencoder for the purpose of discovering these manifold vectors.