TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Regularization

Over-Parameterization

Weight Decay

Dropout

Early Stopping

PAC-Bayesian Learning Theory

From Last Time: Momentum

A comment in the PyTorch source code for SGD states that the implementation uses

$$v^{t+1} = \mu v^t + \hat{g}^t$$

$$\Phi^{t+1} = \Phi^t - \eta v^{t+1}$$

rather than the more standard

$$v^{t+1} = \mu v^t + \eta \hat{g}^t$$

$$\Phi^{t+1} = \Phi^t - v^{t+1}$$

The Momentum Update is a Linear System

Think of the sequence v^t as a function of the sequence $\hat{g}^t$.

$$\begin{aligned} v^{t+1} &= \mu v^t + \hat{g}^t \\ a v^{t+1} &= \mu(a v^t) + a\hat{g}^t \\ u^{t+1} &= \mu u^t + a\hat{g}^t \qquad u^t = a v^t \end{aligned}$$

A Non-Difference

So

$$v^{t+1} = \mu v^t + \hat{g}^t$$

$$\Phi^{t+1} = \Phi^t - \eta v^{t+1}$$

is the same as

$$v^{t+1} = \mu v^t + \eta \hat{g}^t$$

$$\Phi^{t+1} = \Phi^t - v^{t+1}$$

Adjusting the Learning Rate

$$\begin{aligned}\eta_B &= B\eta_1 \\ \eta_\mu &= \eta_0/(1-\mu)\end{aligned}$$

For the convex combination update

$$\begin{aligned}\tilde{g}^{t+1} &= \mu\tilde{g}^t + (1-\mu)\hat{g}^t \\ \Phi^{t+1} &= \Phi^t - \eta\tilde{g}\end{aligned}$$

We should get

$$\eta_\mu = \eta_0$$

Over-Parameterization

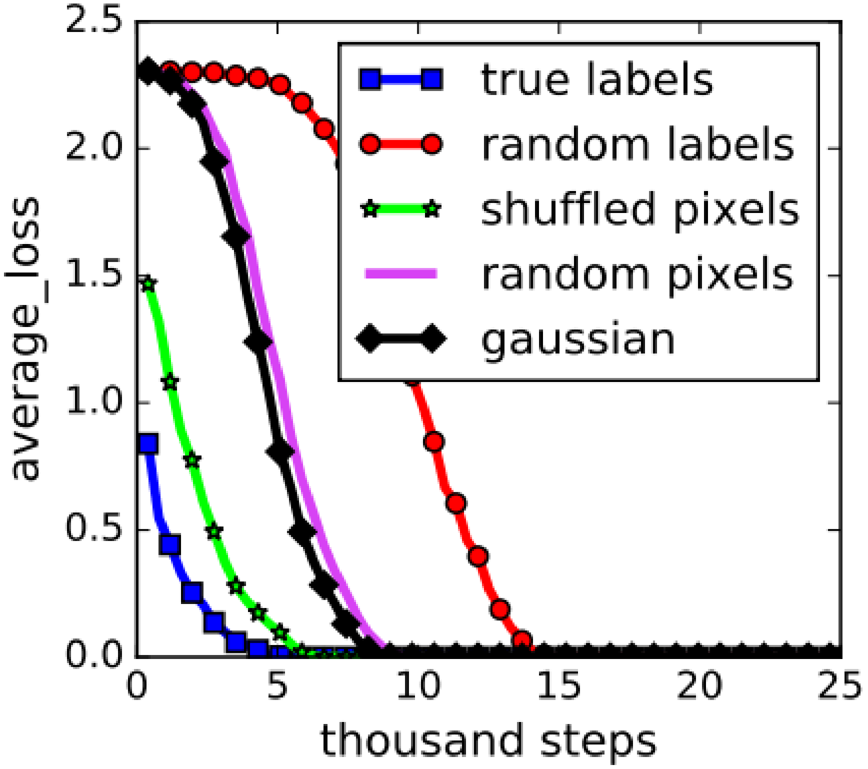
If we have more parameters than data then we can fit any set of labels.

> "Our experiments establish that state-of-the-art convolutional networks for image classification trained with stochastic gradient methods easily fit a random labeling of the training data."

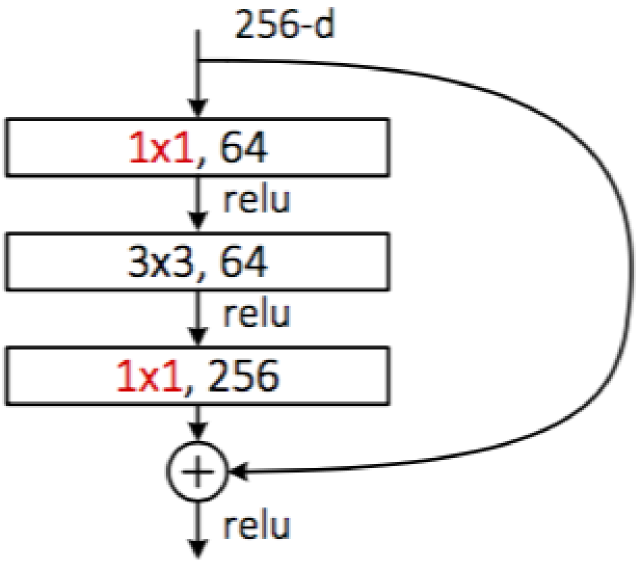
Rethinking Generalization, Zhang et al., ICLR 2017

Over-Parameterization

Inception (Google's net) on CIFAR10 can fit random labels on the training data.



Using Fewer Parameters :)



$$2 \times 256 \times 64 + 9 \times 64 \times 64 = 69,632$$

$$9 \times 256 \times 256 = 589,824$$

bottleneck

(for ResNet-50/101/152)

[Kaiming He]

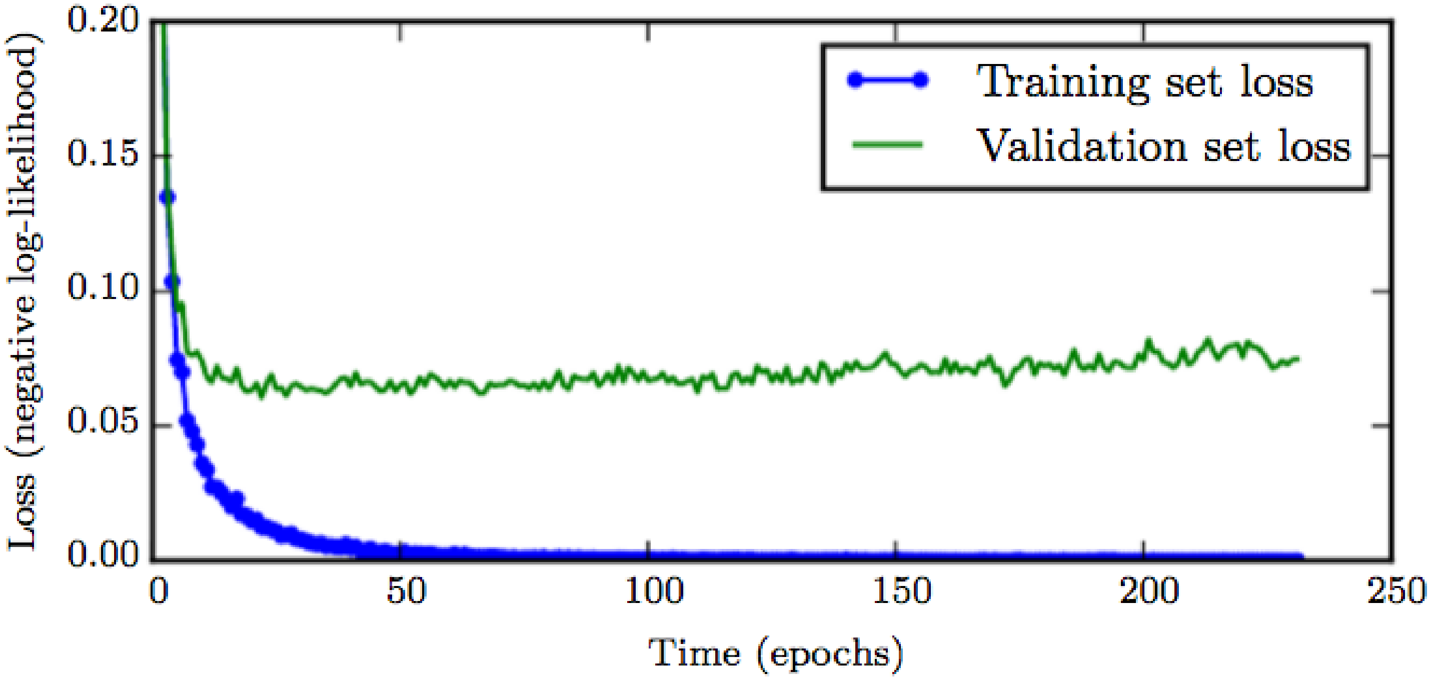
Early Stopping

During SGD one should be tracking validation loss and validation error.

A typical stopping rule (on a GPU) is to stop training when the validation error has not improved for two or three consecutive epochs.

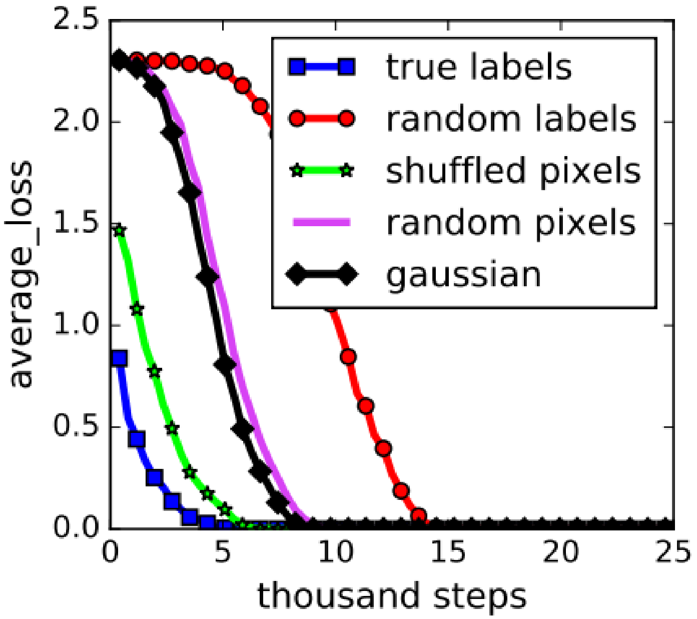
In a Pareto competition one can stop when the combined training time and validation error seems likely to be on the Pareto curve.

Early Stopping



[Goodfellow et al.]

Early Stopping



Implicit Regularization

Consider solving linear least squares regression with SGD.

SGD maintains the invariant that Φ is a linear combination of input vectors.

When over-parameterized the input vectors span a proper subspace.

SGD finds the zero training error solution minimizing $||\Phi||$.

PAC-Bayesian theory can be used to define an implicit regularization (an implicit parameter prior) for any given learning algorithm.

But driving the training error to zero is usually a mistake.

L_2 Regularization (Weight Decay)

Impose a prior probability on parameters

$$P(\Phi) \propto e^{-\frac{1}{2}||\Phi||^2}$$

This can be used to justify L_2 regularization (ridge regression is a special case).

$$\Phi^* = \operatorname*{argmin}_{\Phi}\; E_{(x,y)\sim \mathrm{Train}}\; \mathrm{loss}(\Phi, x, y) + \frac{1}{2}\lambda||\Phi||^2$$

PAC-Bayesian theory (and other theory) can be used to show that a small value of (a form of) this regularized optimization objective guarantees generalization independent of any truth of the prior.

Weight Decay

$$\nabla_\Phi \;\left(E_{(x,y)\sim\mathrm{Batch}}\;\mathrm{loss}(\Phi,x,y)+\frac{1}{2}\lambda||\Phi||^2\right)$$

$$= \;\Phi.\mathrm{grad}+\lambda\Phi$$

$$\Phi^{t+1} \;=\; \Phi^t-\eta\Phi.\mathrm{grad}-\gamma\Phi^t$$

$-\gamma\Phi$ is called weight decay where γ is the weight decay parameter.

Warning: PyTorch does $+\gamma\Phi$ rather than $-\gamma\Phi$ and it seems to be standard for the decay parameter to be negative :P

Dropout

Dropout can be viewed as an ensemble method.

To draw a model from the ensemble we randomly select a mask μ with

$$\begin{cases} \mu_i = 0 \text{ with probability } \alpha \\ \\ \mu_i = 1 \text{ with probability } 1 - \alpha \end{cases}$$

Then we use the model $(\Phi,\ \mu)$ with weight layers defined by

$$y_i = \mathrm{Relu}\left(\sum_j W_{i,j}\ \mu_j x_j\right)$$

Dropout Training

Repeat:

- Select a random dropout mask μ

- Φ `-=` $\nabla_\Phi\ \ell(\Phi, \mu)$

Backpropagation must use the same mask μ used in the forward computation.

Test Time Scaling

At train time we have

$$y_i = \mathrm{Relu}\left(\sum_j W_{i,j}\;\mu_j x_j\right)$$

At test time we have

$$y_i = \mathrm{Relu}\left((1-\alpha)\;\sum_j W_{i,j}\;x_j\right)$$

At test time we use the “average network”.

Dropout for Least Squares Regression

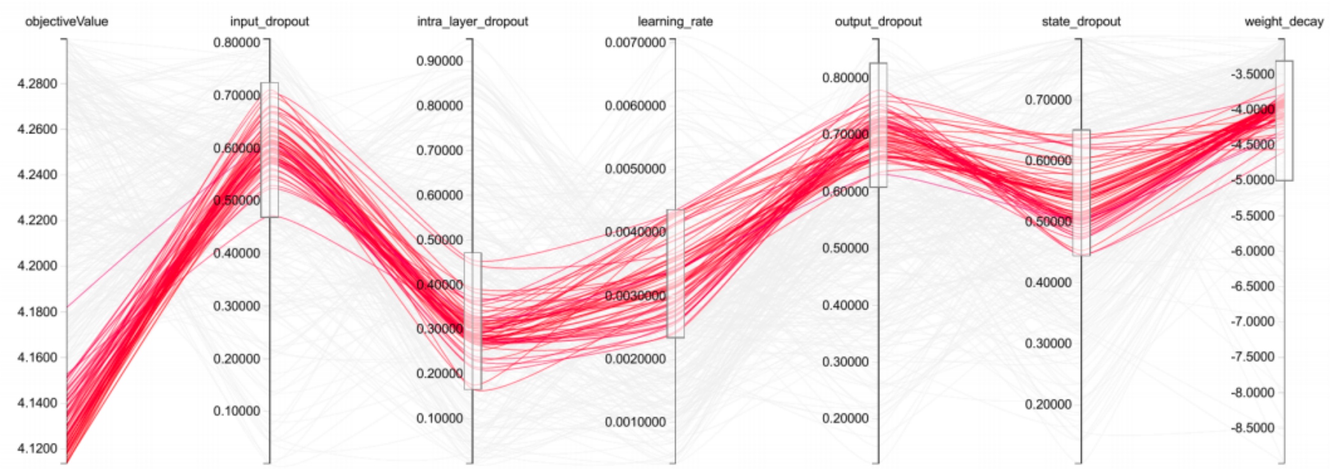
Consider simple least square regression

$$
\begin{aligned}
\Phi^* &= \operatorname*{argmin}_{\Phi} \;\; \mathrm{E}_{(x,y)}\, \mathrm{E}_{\mu} \left[(y - \Phi \cdot (\mu \odot x))^2 \right] \\
&= \mathrm{E} \left[(\mu \odot x)(\mu \odot x)^\top \right]^{-1} \mathrm{E}\left[y(\mu \odot x) \right] \\
&= \operatorname*{argmin}_{\Phi} \;\; \mathrm{E}_{(x,y)} (y - (1-\alpha)\Phi \cdot x)^2 + \sum_i \frac{1}{2}(\alpha - \alpha^2) \mathrm{E}\left[x_i^2 \right] \Phi_i^2
\end{aligned}
$$

In this case dropout is equivalent to a form of L_2 regularization — see Wager et al. (2013).

Search Over Regularization Parameters

Hyper-parameter search on Penn Tree Bank Language Modeling with a 4 layer LSTM.



Loss Input Drop Intra Lay Drop Learning Rate Output Drop State Drop Weight Decay

Melis et al. 2017.

Following Gal and Ghahrmani 2016, state dropout is an RNN parameter dropout with the same mask across the sequence.

Early Stopping

Early stopping can limit $||\Phi||$ — growing a large $||\Phi||$ can take a long time.

Early stopping seems more related to limiting $||\Phi - \Phi_{\mathrm{init}}||$

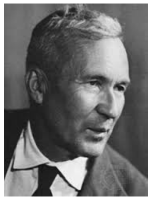
Theoretical guarantees work for $||\Phi - \Phi_{\mathrm{init}}||^2$ just as well as for $||\Phi||^2$.

This suggests replacing weight decay with

$$\Phi^{t+1} = \Phi^t - \eta\Phi.\mathrm{grad} - \gamma(\Phi - \Phi_{\mathrm{init}})$$

Learning Theory: Nature vs. Nurture

Noam Chomsky: Natural language grammar is unlearnable without with an innate linguistic capacity. This position is supported by the "no free lunch theorem".



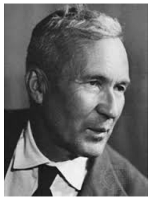
Andrey Kolmogorov, Geoff Hinton: Universal learning algorithms exist. This position is supported by the "free lunch theorem".

The No Free Lunch Theorem

Without prior knowledge, such as universal grammar, it is impossible to make a prediction for an input you have not seen in the training data.

Proof: Select a predictor h uniformly at random from all functions from $\mathcal{X}$ to $\mathcal{Y}$ and then take the data distribution to draw pairs $(x, h(x))$ where x is drawn uniformly from $\mathcal{X}$. No learning algorithm can predict $h(x)$ where x does not occur in the training data.

The Free Lunch Theorem



Universal (knowledge-free) learning algorithms exists.

Let h be a C++ procedure taking an input from $\mathcal{X}$ and returning a value in $\mathcal{Y}$, where h is written using calls to prodecures in an (arbitrarily large) code library L. Let $|h|$ be the number of bit in a standard compression algorithm applied to the source code for h. We are compressing only the "main" procedure h and not the library L.

The Free Lunch Theorem

Consider a loss function $\mathrm{loss}(p, x, y)$ such that for any C program p and pair (x, y) we have $\mathrm{loss}(p, x, y) \in (0, L_{\mathrm{max}})$.

Theorem: For any standard library L fixed before the draw of the training data, we have that with probability at least $1 - \delta$ over the draw of the training data the following holds *simultaneously* for all main programs p and $\lambda > 1/2$.

$$E_{(x,y)\sim\mathrm{Population}}\ \mathrm{loss}(p, x, y)$$

$$\leq \frac{1}{1 - \frac{1}{2\lambda}}\left(E_{(x,y)\sim\mathrm{Train}}\ \mathrm{loss}(p, x, y) + \frac{\lambda L_{\mathrm{max}}}{N}\left((\ln 2)|h| + \ln\frac{1}{\delta}\right)\right)$$

PAC-Bayesian Generalization Bounds

The free lunch theorem is a special case of a PAC-Bayesian generalization bound.

PAC-Bayesian theory was introduced by me in 1999. The bounds have evolved over time with contributions by Langford, Blum, Shawe-Taylor, Catoni and others.

These bounds are of increasing interest today because of their applicability to deep networks.

A More General Free Lunch Theorem

Let $\mathcal{H}$ be a discrete (countable) hypothesis class. $\mathcal{H}$ might be the collection of all C programs.

Let P be a "prior" probability distribution on $\mathcal{H}$. $P(h)$ might be $2^{-8|h|}$ where $|h|$ is the length of h in bytes.

$$
\begin{aligned}
& E_{(x,y)\sim\text{Population}}\;\text{loss}(h,x,y) \\
\leq\; & \frac{1}{1-\frac{1}{2\lambda}}\left(E_{(x,y)\sim\text{Train}}\;\text{loss}(h,x,y)+\frac{\lambda L_{\max}}{N}\left(\ln\frac{1}{P(h)}+\ln\frac{1}{\delta}\right)\right)
\end{aligned}
$$

A Finite Precision Corollary

Suppose that we parameterize a classifier with a parameter vector Φ with d parameters and use b bits per parameter.

$$E_{(x,y)\sim\mathrm{Population}}\;\mathrm{loss}(\Phi,x,y)$$

$$\leq \frac{1}{1-\frac{1}{2\lambda}}\left(E_{(x,y)\sim\mathrm{Train}}\;\mathrm{loss}(\Phi,x,y)+\frac{\lambda L_{\max}}{N}\left((\ln 2)db+\ln\frac{1}{\delta}\right)\right)$$

Proof

$$L(h) = E_{(x,y)\sim \mathrm{Pop}} \, \mathrm{loss}(h, x, y)$$

$$\hat{L}(h) = E_{(x,y)\sim \mathrm{Train}} \, \mathrm{loss}(h, x, y)$$

Proof

Consider $L_{\max} = 1$ and define $\epsilon(h)$ by

$$\epsilon(h) = \sqrt{\frac{2L(h)\left(\ln \frac{1}{P(h)} + \ln \frac{1}{\delta}\right)}{N}}.$$

By the relative Chernoof bound we have

$$P_{\mathrm{Train}\sim\mathrm{Pop}}\left(\hat{L}(h) \leq L(h) - \epsilon(h)\right) \leq e^{-N\frac{\epsilon(h)^2}{2L(h)}} = \delta P(h).$$

Proof

$$P_{\text{Train}\sim\text{Pop}}\left(\hat{L}(h) \leq L(h) - \epsilon(h)\right) \leq \delta P(h).$$

$$P_{\text{Train}\sim\text{Pop}}\left(\exists h\ \hat{L}(h) \leq L(h) - \epsilon(h)\right) \leq \sum_h \delta P(h) = \delta$$

$$P_{\text{Train}\sim\text{Pop}}\left(\forall h\ L(h) \leq \hat{L}(h) + \epsilon(h)\right) \geq 1 - \delta$$

Proof

$$L(h) \leq \widehat{L}(h) + \sqrt{L(h)\left(\frac{2\left(\ln\frac{1}{P(h)} + \ln\frac{1}{\delta}\right)}{N}\right)}$$

using

$$\sqrt{ab} = \inf_{\lambda > 0}\ \frac{a}{2\lambda} + \frac{\lambda b}{2}$$

we get

$$L(h) \leq \widehat{L}(h) + \frac{L(h)}{2\lambda} + \frac{\lambda\left(\ln\frac{1}{P(h)} + \ln\frac{1}{\delta}\right)}{N}$$

Proof

$$L(h) \leq \widehat{L}(h) + \frac{L(h)}{2\lambda} + \frac{\lambda\left(\ln\frac{1}{P(h)} + \ln\frac{1}{\delta}\right)}{N}$$

Solving for $L(h)$ yields

$$L(h) \leq \frac{1}{1 - \frac{1}{2\lambda}}\left(\hat{L}(h) + \frac{\lambda L_{\max}}{N}\left(\ln\frac{1}{P(h)} + \ln\frac{1}{\delta}\right)\right)$$

A KL Divergence Bound

Let P be any "prior" and Q be any "poterior" on any model space.
Define

$$L(Q) = \mathrm{E}_{h \sim Q}\left[L(h)\right]$$

$$\hat{L}(Q) = \mathrm{E}_{h \sim Q}\left[\hat{L}(h)\right]$$

For any P and any $\lambda > \frac{1}{2}$, with probability at least $1 - \delta$ over the draw of the training data, the following holds simultaneously for all Q.

$$L(Q) \leq \frac{1}{1 - \frac{1}{2\lambda}}\left(\hat{L}(Q) + \frac{\lambda L_{\max}}{N}\left(KL(Q, P) + \ln \frac{1}{\delta}\right)\right)$$

L_2 Bounds

$$P(w) = \mathcal{N}(0,1)^d$$

$$L(Q_\Theta) = \mathrm{E}_{\epsilon \sim \mathcal{N}(0,1)^d}\left[L(\Theta + \epsilon)\right]$$

$$\hat{L}(Q_\Theta) = \mathrm{E}_{\epsilon \sim \mathcal{N}(0,1)^d}\left[\hat{L}(\Theta + \epsilon)\right]$$

$$KL(Q_\Theta, P) = \frac{1}{2}||\Theta||^2$$

$$L(Q_\Theta) \leq \frac{1}{1 - \frac{1}{2\lambda}}\left(\hat{L}(Q_\Theta) + \frac{\lambda L_{\max}}{N}\left(\frac{1}{2}||\Theta||^2 + \ln\frac{1}{\delta}\right)\right)$$

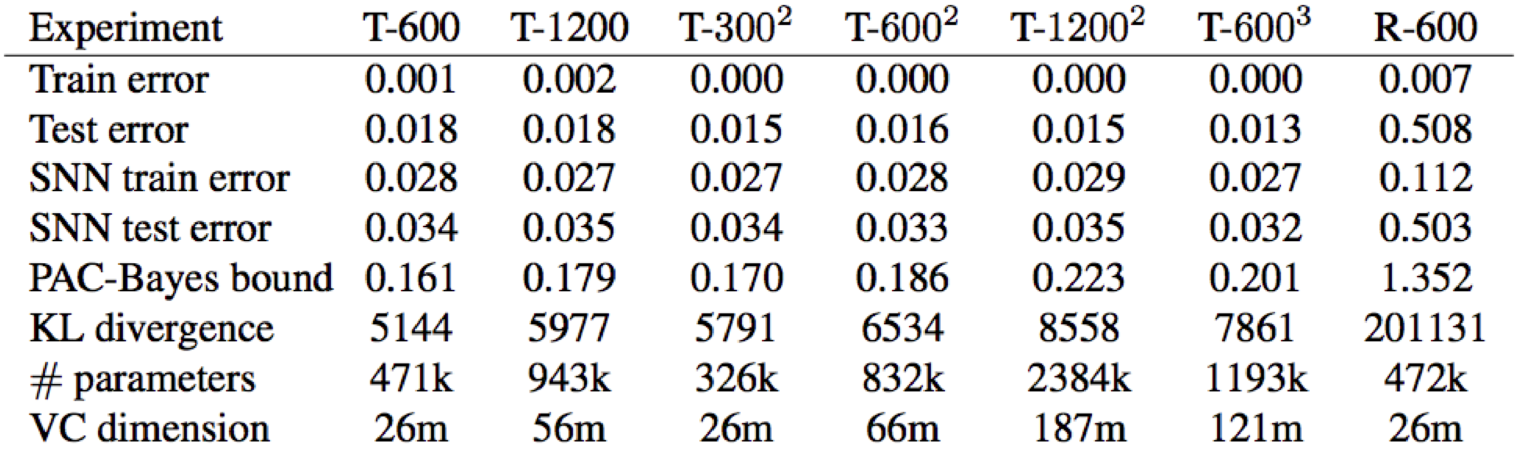
A Dropout Bound

$$\begin{aligned} KL(Q_{\alpha,\Phi},\ Q_{\alpha,0}) &= \mathrm{E}_{\mu\sim P_\alpha,\epsilon\sim\mathcal{N}(0,1)^d}\left[\ln\frac{P_\alpha(\mu)e^{-\frac{1}{2}||\mu\odot\epsilon||^2}}{P_\alpha(\mu)e^{-\frac{1}{2}||\mu\odot(\Phi+\epsilon)||^2}}\right] \\ &= \mathrm{E}_{\mu\sim P_\alpha}\left[\frac{1}{2}||\mu\odot\Phi||^2\right] \\ &= \frac{1-\alpha}{2}||\Phi||^2 \end{aligned}$$

$$L(Q_{\alpha,\Phi}) \leq \frac{1}{1-\frac{1}{2\lambda}}\left(\hat{L}(Q_{\alpha,\Phi}) + \frac{\lambda L_{\max}}{N}\left(\frac{1-\alpha}{2}||\Phi||^2 + \ln\frac{1}{\delta}\right)\right)$$

L_2 PAC-Bayesian Bounds in Action

Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data, (Dziugaite and Roy, arXiv, 2017)



Experiment	T-600	T-1200	T-300^2	T-600^2	T-1200^2	T-600^3	R-600
Train error	0.001	0.002	0.000	0.000	0.000	0.000	0.007
Test error	0.018	0.018	0.015	0.016	0.015	0.013	0.508
SNN train error	0.028	0.027	0.027	0.028	0.029	0.027	0.112
SNN test error	0.034	0.035	0.034	0.033	0.035	0.032	0.503
PAC-Bayes bound	0.161	0.179	0.170	0.186	0.223	0.201	1.352
KL divergence	5144	5977	5791	6534	8558	7861	201131
# parameters	471k	943k	326k	832k	2384k	1193k	472k
VC dimension	26m	56m	26m	66m	187m	121m	26m

The bounds are based on L_2 distance of the weight vector to the initialization.

The weight vector is retrained to minimize the bound.

A Formal Definition of Implicit Prior

We now consider any learning algorithm $\mathcal{A}$ which takes training data and returns a probability distribution $Q_{\mathcal{A}}(\mathrm{Train})$ on parameters.

define $Q_{\mathcal{A}}(\mathrm{Pop}) = E_{\mathrm{Train} \sim \mathrm{Pop}}\; Q_{\mathcal{A}}(\mathrm{Train})$

To draw Φ from $Q_{\mathcal{A}}(\mathrm{Pop})$ I first draw Train from Pop and then draw Φ from $Q_{A}(\mathrm{Train})$.

We will show that $Q_{\mathcal{A}}(\mathrm{Pop})$ is an optimal prior for $\mathcal{A}$ running on Pop. We can think of $Q_{\mathcal{A}}(\mathrm{Pop})$ as an implicit prior for $\mathcal{A}$.

Optimality of $Q_{\mathcal{A}}(\mathrm{Pop})$

Consider an arbitrary prior P.

$$E_{\mathrm{Train}\sim\mathrm{Pop}}\ KL(Q_{\mathcal{A}}(\mathrm{Train}), P)$$

$$= E_{\mathrm{Train}\sim\mathrm{Pop}}\ \ln\frac{Q_{\mathcal{A}}(\mathrm{Train})(h)}{P(h)}$$

$$= \mathrm{E}_{\mathrm{Train}\sim\mathrm{Pop},\ h\sim Q_{\mathcal{A}}(\mathrm{Train})}\left[\ln\frac{Q_{\mathcal{A}}(\mathrm{Train})(h)}{Q_{\mathcal{A}}(\mathrm{Pop})(h)}\right]$$

$$+\mathrm{E}_{h\sim Q_{\mathcal{A}}(\mathrm{Pop})}\left[\ln\frac{Q_{\mathcal{A}}(\mathrm{Pop}(h))}{P(h)}\right]$$

Optimality of $Q_{\mathcal{A}}(\mathrm{Pop})$

$$
\begin{aligned}
& E_{\mathrm{Train}\sim\mathrm{Pop}}\; KL(Q_{\mathcal{A}}(\mathrm{Train}), P) \\
= \; & \mathrm{E}_{\mathrm{Train}\sim\mathrm{Pop}}\left[KL(Q_{\mathcal{A}}(\mathrm{Train}), Q_{\mathcal{A}}(\mathrm{Pop}))\right] + KL(Q_{\mathcal{A}}(\mathrm{Pop}), P)
\end{aligned}
$$

So $Q_{\mathcal{A}}(\mathrm{Pop})$ is an **optimal prior** (or **implicit prior**) for algorithm $\mathcal{A}$.

An Implicit Prior Generalization Bound

For any given learning algorithm $\mathcal{A}$ and $\lambda > \frac{1}{2}$ we have the following with probability at least $1 - \delta$ over the draw of the training data.

$$L(Q_{\mathcal{A}}(\mathrm{Train}))$$

$$\leq \frac{1}{1 - \frac{1}{2\lambda}} \left(\begin{array}{l} \hat{L}(Q_{\mathcal{A}}(\mathrm{Train})) \\ \\ + \frac{\lambda L_{\max}}{N} \left(KL(Q_{\mathcal{A}}(\mathrm{Train}), Q_{\mathcal{A}}(\mathrm{Pop})) + \ln \frac{1}{\delta} \right) \end{array} \right)$$

Ensembles under Square Loss

We average k regression models

$$f(x) = \frac{1}{k}\sum_{i=1}^{k} f_i(x)$$

$$f(x) - y = \frac{1}{k}\sum_{i=1}^{k} (f_i(x) - y)$$

$$\epsilon = \frac{1}{k}\sum_{i=1}^{k} \epsilon_i, \quad \epsilon_i = f_i - y \quad \text{(residuals)}$$

Ensembles

Assume that $\mathrm{E}\left[\epsilon_i^2\right] = \sigma^2$ and $\mathrm{E}\left[\epsilon_i \epsilon_j\right] = \sigma^2 \rho$ for $i \neq j$.

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

$$= \frac{1}{k}\sigma^2 + \frac{k-1}{k}\sigma^2\rho \;=\; \sigma^2\left(\frac{1}{k} + \left(1 - \frac{1}{k}\right)\rho\right)$$

If Pearson's correlation $\rho = \mathrm{E}\left[\epsilon_i \epsilon_j\right]/\sigma^2 < 1$ we win.

Ensembles Under Log Loss

For log loss we average the probability vectors.

$$P(y|x) = \frac{1}{k} \sum_i \; P_i(y|x)$$

$-\log P$ is a convex function of P. For any convex $\ell(P)$ Jensen's inequality states that

$$\ell\left(\frac{1}{k}\sum_i P_i\right) \leq \frac{1}{k}\sum_i \ell(P_i)$$

This implies that the loss of the average model cannot be worse (can only be better) than the average loss of the models.

L_1 Regularization and Sparse Weights

$$p(\Phi) \propto e^{-||\Phi||_1} \qquad ||\Phi||_1 = \sum_i |\Phi_i|$$

$$\Phi^* = \underset{\Phi}{\mathrm{argmin}} \quad \ell_{\mathrm{train}}(\Phi) \;+\; \lambda||\Phi||_1$$

$$\begin{array}{rcl} \Phi & \texttt{-=} & \eta\nabla_\Phi\, \ell_{\mathrm{train}}(\Phi) \\ \Phi_i & \texttt{-=} & \eta\lambda\;\mathrm{sign}(\Phi_i) \qquad \text{(shrinkage)} \end{array}$$

At equilibrium (sparsity is difficult to achieve with SGD)

$$\begin{array}{rcll} \Phi_i & = & 0 & \text{if } |\partial\ell/\partial\Phi_i| < \lambda \\ \partial\ell/\partial\Phi_i & = & -\lambda\mathrm{sign}(\Phi_i) & \text{otherwise} \end{array}$$

Sparse Activation

We can impose an L_1 regularization on the activations of the network (the output of the activation function of each neuron).

$$\Phi^* = \operatorname*{argmin}_{\Phi} \ell(\Phi) + \lambda ||h||_1$$

where h is the vector of neuron activation outputs.

This will tend to make activations sparse.

Sparse Coding

Let W be a matrix where we view $W_{\cdot,i}$ is the ith "dictionary vector".

For input x we can construct a k-sparse representation $h(x)$.

$$h(x) = \underset{h, ||h||_0 = k}{\operatorname{argmin}} \; ||x - Wh||^2$$

Note

$$Wh = \sum_{i \in I(x)} h_i \, W_{\cdot,i} \qquad |I(x)| = k$$

We can now replace x by its sparse code $h(x)$.

END