Fast deterministic dropout training

NIPS workshop
Sida Wang, Chris Manning
What is dropout training

- Recently introduced by Hinton et al. in “Improving neural networks by preventing co-adaptation of feature detectors”
- Randomly select some inputs for each unit
  - zero them
  - compute the gradient, make an update
  - repeat

![Diagram](Diagram showing the process of dropout training.)
Dropout training is promising

- Won the ImageNet challenge by a margin
- George Dahl et al. won the Merck challenge
- Dropout seems to be an important ingredient
Preventing the danger of co-adaptation

• Observed in NLP and machine learning
• Model averaging
  • “Feature bagging”: train models on different subset of features [Sutton et al, 2006] to prevent “weight undertraining”.
• Deal with co-adaptation
  • Naïve Bayes can sometimes do better than discriminative methods
  • Regularize uninformative features more [Wang/Manning, 2012]
In this work...

- Sampling is inefficient – we integrate
  - 50% random dropout, after 5 passes of the data, $1/32 = 3\%$ of the data is still unseen

- No objective function was used in the original work – we use an implied objective function

- Understanding dropout as Bayesian model selection, and extensions
CLT and the Gaussian approximation

- Warm up with binary logistic regression

\[ Y(z) = w^t D_z x = \sum_i^{m} w_i x_i z_i \]
\[ S = E_z[Y(z)] + \sqrt{\text{Var}[Y(z)]} \epsilon \]
\[ \epsilon \sim N(0,1), \]
\[ E_z[Y(z)] = \sum_i^{m} p_i w_i x_i \]
\[ \text{Var}[Y(z)] = \sum_i^{m} p_i (1 - p_i)(w_i x_i)^2 \]
Computing the gradient

- The most naïve $E(XY) \approx E(X)E(Y)$ does not work

- But we can sample from the Gaussian and linearize

$$\frac{\partial L(w)}{\partial w_i} = E_z[f(Y(z))x_i z_i]$$

$$= \sum_{z_i \in \{0, 1\}} p(z_i) z_i x_i E_{z_{-i}|z_i}[f(Y(z))]$$

$$= p(z_i = 1) x_i E_{z_{-i}|z_i=1}[f(Y(z))]$$

$$\approx p_i x_i \left( E_{S \sim \mathcal{N}({\mu}_S, {\sigma}_S^2)}[f(S)] + \Delta {\mu}_i \frac{\partial E_{T \sim \mathcal{N}({\mu}_S, {\sigma}_S^2)}[f(T)]}{\partial {\mu}} \bigg|_{\mu={\mu}_S} \right) +$$

$$\Delta \sigma_i^2 \frac{\partial E_{T \sim \mathcal{N}({\mu}_S, {\sigma}^2)}[f(T)]}{\partial {\sigma}^2} \bigg|_{\sigma^2={\sigma}_S^2}$$

$$= p_i x_i (\alpha({\mu}_S, {\sigma}_S^2) + \Delta {\mu}_i \beta({\mu}_S, {\sigma}_S^2) + \Delta \sigma_i^2 \gamma({\mu}_S, {\sigma}_S^2))$$
Quality of the gradients

• The approximate gradient vs. true gradient
This method works well

- Reduces complexity from $O(Md)$ to $O(M+d)$ for $M$ samples and data dimension $d$.

![Graph](image_url)

Figure 1: Validation errors vs. time spent in training (left), and number of iterations (right) trained using batch gradient descent with Wolfe line search on the 20-newsgroup subtask alt.atheism vs. religion.misc. 100 samples are used. For MC dropout, $z_i$ is sampled only for non-zero $x_i$, with a dropout rate of $0.5$.

The accuracy and time taken are listed in table 1 for the datasets described in section A.1. The Gaussian approximation is generally around 10 times faster than MC dropout and performs comparably to NBSVM in [3]. Further speedup is possible by using one of the deterministic approximations instead of sampling. While each iteration of the Gaussian approximation is still slower than LR, it sometimes reaches a better validation performance in less time.
Document classification results

- Improvements over plain LR and most previous methods.

- Average accuracy and time on 9 datasets:

<table>
<thead>
<tr>
<th>Methods</th>
<th>Plain LR</th>
<th>Dropout</th>
<th>Fast Drop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>84.97</td>
<td>86.88</td>
<td>86.99</td>
</tr>
<tr>
<td>Time</td>
<td>92</td>
<td>2661</td>
<td>325</td>
</tr>
</tbody>
</table>
### Document classification results

- At or close to state of the art!

<table>
<thead>
<tr>
<th>Methods\Datasets</th>
<th>RT-2k</th>
<th>IMDB</th>
<th>RTs</th>
<th>subj</th>
<th>AthR</th>
<th>CR</th>
<th>MPQA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC dropout</td>
<td>89.8</td>
<td>91.2</td>
<td>79.2</td>
<td>93.3</td>
<td>86.7</td>
<td>82.0</td>
<td>86.0</td>
<td>86.88</td>
</tr>
<tr>
<td>training time</td>
<td>6363</td>
<td>6839</td>
<td>2264</td>
<td>2039</td>
<td>126</td>
<td>582</td>
<td>417</td>
<td>2661</td>
</tr>
<tr>
<td>Gaussian approx.</td>
<td>89.7</td>
<td>91.2</td>
<td>79.0</td>
<td>93.4</td>
<td>87.4</td>
<td>82.1</td>
<td>86.1</td>
<td>86.99</td>
</tr>
<tr>
<td>training time</td>
<td>240</td>
<td>1071</td>
<td>362</td>
<td>323</td>
<td>6</td>
<td>90</td>
<td>185</td>
<td>325</td>
</tr>
<tr>
<td>plain LR</td>
<td>88.2</td>
<td>89.5</td>
<td>77.2</td>
<td>91.3</td>
<td>83.6</td>
<td>80.4</td>
<td>84.6</td>
<td>84.97</td>
</tr>
<tr>
<td>training time</td>
<td>145</td>
<td>306</td>
<td>81</td>
<td>68</td>
<td>3</td>
<td>17</td>
<td>22</td>
<td>92</td>
</tr>
</tbody>
</table>

**Previous results**

<table>
<thead>
<tr>
<th></th>
<th>RT-2k</th>
<th>IMDB</th>
<th>RTs</th>
<th>subj</th>
<th>AthR</th>
<th>CR</th>
<th>MPQA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeCRF[7]</td>
<td>-</td>
<td>-</td>
<td>77.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>81.4</td>
<td>86.1</td>
</tr>
<tr>
<td>Vect. Sent.[8]</td>
<td>88.9</td>
<td>88.89</td>
<td>-</td>
<td>88.13</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RNN[9]</td>
<td>-</td>
<td>-</td>
<td>77.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>86.4</td>
<td>-</td>
</tr>
<tr>
<td>NBSVM[3]</td>
<td>89.45</td>
<td>91.22</td>
<td>79.4</td>
<td>93.2</td>
<td>87.9</td>
<td>81.8</td>
<td>86.3</td>
<td>87.03</td>
</tr>
</tbody>
</table>

| \(|\{i : x_i > 0\}\)| | 788   | 232  | 22  | 25   | 346  | 21  | 4     |

<table>
<thead>
<tr>
<th>Methods\Datasets</th>
<th>RT-2k</th>
<th>IMDB</th>
<th>RTs</th>
<th>subj</th>
<th>AthR</th>
<th>CR</th>
<th>MPQA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC dropout</td>
<td>89.8</td>
<td>91.2</td>
<td>79.2</td>
<td>93.3</td>
<td>86.7</td>
<td>82.0</td>
<td>86.0</td>
<td>86.88</td>
</tr>
<tr>
<td>training time</td>
<td>6363</td>
<td>6839</td>
<td>2264</td>
<td>2039</td>
<td>126</td>
<td>582</td>
<td>417</td>
<td>2661</td>
</tr>
<tr>
<td>Gaussian approx.</td>
<td>89.7</td>
<td>91.2</td>
<td>79.0</td>
<td>93.4</td>
<td>87.4</td>
<td>82.1</td>
<td>86.1</td>
<td>86.99</td>
</tr>
<tr>
<td>training time</td>
<td>240</td>
<td>1071</td>
<td>362</td>
<td>323</td>
<td>6</td>
<td>90</td>
<td>185</td>
<td>325</td>
</tr>
<tr>
<td>plain LR</td>
<td>88.2</td>
<td>89.5</td>
<td>77.2</td>
<td>91.3</td>
<td>83.6</td>
<td>80.4</td>
<td>84.6</td>
<td>84.97</td>
</tr>
<tr>
<td>training time</td>
<td>145</td>
<td>306</td>
<td>81</td>
<td>68</td>
<td>3</td>
<td>17</td>
<td>22</td>
<td>92</td>
</tr>
</tbody>
</table>
But it still requires sampling!

- We need to compute several expectations
- Could just build a table, and table of partial derivatives, a pain to implement. Here is a good numerical approximation:

\[
\int_{-\infty}^{\infty} \sigma(x) N(x | \mu, s^2) dx \approx \sigma \left( \frac{\mu}{\sqrt{1 + \pi s^2/8}} \right)
\]
We can even approximate the objective function directly!

- For binary LR, no linearizing and sampling is needed at all!

\[
E_{Y \sim \mathcal{N}(\mu, s^2)}[\log(\sigma(Y))] = \int_{-\infty}^{\infty} \log(\sigma(x)) \mathcal{N}(x|\mu, s^2) dx \\
\approx \sqrt{1 + \pi s^2 / 8} \log \sigma \left( \frac{\mu}{\sqrt{1 + \pi s^2 / 8}} \right)
\]

- Works slightly worse in practice compared to sampling
- still retains over 80% of the improvement over baseline
Relation to Bayesian model selection

- We are in effect maximizing a lower bound on the Bayesian evidence:

\[
M_{\mu} = \int p(D|w)p(w|\mu)dw
\]

- Under the model

\[
p(w_i|\mu_i) = \mathcal{N}(w_i|\mu_i, \alpha \mu_i^2)
\]
\[
p(y|x, w) = \sigma(yw^T x)
\]

\[
L(w) = E_{z; z_i \sim \text{Bernoulli}(p_i)}[\log p(y|w^T D_z x)]
\]
\[
\approx E_{y \sim \mathcal{N}(E[w^T D_z x], \text{Var}[w^T D_z x])}[\log p(y|Y)]
\]
\[
= E_{v:v_i \sim \mathcal{N}(\mu_i, \alpha \mu_i^2)}[\log p(y|v^T x)]
\]
\[
\leq \log E_{v:v_i \sim \mathcal{N}(\mu_i, \alpha \mu_i^2)}[p(y|v^T x)]
\]
\[
= \log(M_{\mu})
\]
Fast dropout in neural networks

- The fast dropout idea can be applied to neural networks
- Each hidden unit has an input mean and an input variance
- Outputs a mean and variance
Applies to different types of units

- Sigmoid unit:
  \[ \nu = \int_{-\infty}^{\infty} \sigma(x) \mathcal{N}(x|\mu, s^2) dx \approx \sigma \left( \frac{\mu}{\sqrt{1 + \pi s^2 / 8}} \right) \]

- Rectified linear unit: \( f(x) = \max(0, x) \quad r = \mu / s \)
  \[ \nu = \int_{-\infty}^{\infty} f(x) \mathcal{N}(x|\mu, s^2) dx = \Phi(r) \mu + s \mathcal{N}(r|0, 1) \]

- Can also apply to different classifiers, and regression.
Can now train neural networks with deterministic dropout!

- We can now adjust $\alpha$ and add more variance freely if the one decided by dropout is suboptimal.

<table>
<thead>
<tr>
<th>Method name</th>
<th>Number of errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN-1200-1200 plain</td>
<td>182</td>
</tr>
<tr>
<td>NN-1200-1200 det. Dropout</td>
<td>134</td>
</tr>
<tr>
<td>NN-1200-1200 det. Dropout + Var</td>
<td>109</td>
</tr>
<tr>
<td>NN-1200-1200 det. Dropout + Var</td>
<td>110</td>
</tr>
<tr>
<td>NN-300 MSE [LeCun 1998]</td>
<td>360</td>
</tr>
<tr>
<td>NN-800 [Simard 2003]</td>
<td>160</td>
</tr>
<tr>
<td>Real dropout [Hinton 2012]</td>
<td>105-120</td>
</tr>
<tr>
<td></td>
<td>79 with pretraining</td>
</tr>
</tbody>
</table>
More classification results

- Classification on some more datasets

<table>
<thead>
<tr>
<th></th>
<th>SmallM</th>
<th>USPS</th>
<th>Isolet</th>
<th>hepatitis</th>
<th>soybean</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoDrop</td>
<td>87</td>
<td>94.6</td>
<td>90.5</td>
<td>83</td>
<td>94</td>
</tr>
<tr>
<td>F.Dropout</td>
<td>90</td>
<td>96.3</td>
<td>93.2</td>
<td>87</td>
<td>88</td>
</tr>
</tbody>
</table>
Regression

- The final layer is like L2 penalty:
  \[ s^2 = \sum_j \alpha w_j^2 x_j^2 \]
  \[
  E_{X \sim \mathcal{N}(\mu, s^2)}[(X - t)^2] = \int_{-\infty}^{\infty} (x - t)^2 \mathcal{N}(x | \mu, s^2) dx \\
  = s^2 + (\mu - t)^2
  \]

- Test error (training error) for X-200-100-y

<table>
<thead>
<tr>
<th>Sq.Error</th>
<th>liver</th>
<th>cardio</th>
<th>housing</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoDrop</td>
<td>16 (3.1)</td>
<td>4320 (90)</td>
<td>39 (29)</td>
<td>1.5 (1.2)</td>
</tr>
<tr>
<td>F.Dropout</td>
<td>10 (9.5)</td>
<td>298 (217)</td>
<td>35 (32)</td>
<td>1.3 (1.4)</td>
</tr>
</tbody>
</table>
Conclusions

• Dropout training seems promising
• But doing real dropout is slow, sampling is expensive
• Apply the Gaussian approximation
• Cheaper samples or completely deterministic approximations!
The output variance of hidden units

- Wanted an overestimate. Not exact, but fairly accurate

\[
\text{Var}_{Y \sim \mathcal{N}(\mu, s^2)}[\sigma(Y)] :
\]

\[
= E[\sigma(Y)^2] - E[\sigma(Y)]^2
\]

\[
\approx E[\sigma(a(Y - b))] - E[\sigma(Y)]^2
\]

\[
\approx \sigma \left( \frac{a(\mu - b)}{\sqrt{1 + \pi/8a^2s^2}} \right) - \sigma \left( \frac{\mu}{\sqrt{1 + \pi/8s^2}} \right)
\]
Covariance matrices

This integral can be evaluated exactly for the rectified linear unit \( f(x) = \max(0, x) \). Let \( r = \mu/s \), then
\[
\mathcal{N}(x|\mu, s^2) \]
\[
\int f(x) \mathcal{N}(x|\mu, s^2) \, dx = (r)\mu + s N(r|0, 1) (13)
\]

With dropout training, each hidden unit also has an output variance, which can be approximated fairly well (see A.4).

Figure 1: Top: MC dropout covariance matrix of the inputs of 10 random hidden units; Top-left: trained to convergence; Top-right: at random initialization. The covariance is not completely diagonal once trained to convergence. Bottom: empirical input distribution of the input of a hidden unit. Bottom-left: trained to convergence; Bottom-right: at initialization. We lose almost nothing here.

3.2 Training with backpropagation

The resulting neural network can be trained by backpropagation with an additional set of derivatives. In normal backpropagation, one only need to keep \( \frac{\partial L}{\partial \mu_i} \) for each hidden unit \( i \) with input \( \mu_i \). For approximate dropout training, we need \( \frac{\partial L}{\partial s^2_i} \) as well for input variance \( s^2_i \). Where
\[
\mu_i = p P_j w_{ij} \xi_j
\]
\[
s^2_i = p (1/p) P_j \xi_j^2 w_{ij}^2 + p \xi_0^2 w_{ij}^2 \]
and \( \xi_0 \), \( \xi_2 \) are the output mean and variance of the previous layer.

3.3 The output layer

We still need to define what the cost function \( L \) is, which is task dependent. We outline how to do approximate dropout for the final layer for one-vs-rest logistic units, linear units under squared loss, and softmax units under their representative cost functions.

Logistic units with the cross-entropy loss function that can be well-approximated using the following:
\[
E_{X \sim \mathcal{N}(\mu, s^2)} [\log(f(X))] = \int \log(f(x)) \mathcal{N}(x|\mu, s^2) \, dx (14)
\]
\[
p p \frac{1 + p \frac{1 + s^2}{8} \log(p \frac{1 + p \frac{1 + s^2}{8}})} (15)
\]

Linear units with squared error loss can be computed exactly
\[
E_{X \sim \mathcal{N}(\mu, s^2)} [(X_t)^2] = \int (x_t)^2 \mathcal{N}(x_t|\mu, s^2) \, dx (16)
\]
\[
s^2 + (\mu_t)^2 (17)
\]
Since \( s^2 = \sum_j \xi_j^2 w_{ij}^2 x_j^2 \), this is L2 regularization.
Testing the assumptions

- The training objective functions
  - Left: training on expected cross entropy (dropout)
  - Right: training on cross entropy (plain LR)