Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Srivastava et al

10.10.17

Summary by Justin Chen

Summary

Introduction:

- Dropout technique addresses better generalization for large, over-parameterized neural networks
- Technique **randomly** drops neurons during **training**
- Trains an ensemble of "**thinned**" networks and has the approximate effect of averaging predictions of ensemble during inference
- They focus on regularizing **fixed-sized models** by approximating an equally weighted geometric mean of the predictions from an ensemble of exponentially mean trained models that all share the same set of parameters
- **Naive approach** to averaging exponentially many models would require finding unique set of hyperparameters, unique dataset, training each individual network, and inferencing all networks simultaneously which is infeasible.
    - Dropout circumvents the aforementioned issues while also improving generalization
- All connections to dropped out neuron are also dropped during training
- Neurons are dropped with probability *P*
    - In practice, neurons in the input layer are dropped with much smaller probability, less than 0.8 and for hidden units and output units *P* can be fixed to 0.5
    - *P* can also be adjusted using validation set
- Dropout essentially samples subnetworks during training
- Original network with n neurons can also be thought of as having 2^n subnetworks
- All subnetworks share the same parameters
    - Total number of parameters is $O(n^2)$
        - Remember each row in a weight matrix is a neuron
- Training with dropout effectively
    - Trains at most 2^n networks, some aren't sampled during training

- During inference, weights of dropped out neurons are scaled down by probability *P* of it being dropped out during training
    - Neurons are ***never*** dropped out during inference. All neurons that were dropped out during training are **kept during inference** along with all other neurons!
    - Note that in this summary, *P* is the probability of being dropped and the paper refer to *P* as the probability of being retained
        - I think referring to the dropout probability in this way is more natural
        - *P* here is 1-*P* in the paper.
    - Scaling by *P* ensures that the expected output of that neuron is the same during training and inference
- They found that dropout significantly **decreases** generalization **error** compared to previous regularization techniques
    - Note: this paper was published in 2014
- Dropout encourages sparse activation during inference
- Dropout can be thought of as regularizing by adding noise to neurons
- “minimize loss function stochastically under noise distribution.”
    - Can be interpreted as minimizing expected loss function
- Does not require dimensionality reduction to perform well
- Gaussian noise can perform better than Bernoulli
- Drawbacks:
    - 2-3 times slower training time compared to same architectures without dropout
    - Noisy parameter updates

Motivation:

- Sexual reproduction in evolution
    - Half of a parent's genes are dropped *(innuendos - giggity ;)*
    - Asexual reproduction should work better since the genes have evolved (optimized) to work together
    - Sexual reproduction breaks out those co-adaptations, which you’d assume would decrease fitness, yet sexual reproduction has evolved the most advanced organisms
    - “Mix-ability” - ability to be useful alone or work with other random genes
        - This may be the desired property of natural selection not individual fitness (hypothesis)
        - Allows new genes to spread throughout population and reduce complexity that would ultimately reduce fitness

- Breaking co-adaptation "makes each hidden unit more robust and drive it towards creating useful features on its own without relying on other hidden units to correct its mistakes."

Model Description:

$$
\begin{aligned}
r_j^{(l)} &\sim \quad Bernoulli(p), \\
\widetilde{y}^{(l)} &= \quad r^{(l)} * y^{(l)} \\
z_i^{(l+1)} &= \quad w_i^{(l+1)} \widetilde{y}^{l} + b_i^{(l+1)} \\
y_i^{(l+1)} &= \quad f(z_i^{(l+1)})
\end{aligned}
$$

- In the equations above, everything is a vector and * means element-wise multiplication
- Essentially each weight has an extra Bernoulli coefficient (either 0 or 1)
- Derivative of loss function only backpropagated through subnetwork (neurons that were not dropped)

Backpropagation:

- Train as usual, but sample a subnetwork for each mini-batch
- "Gradients for **each parameter** are **averaged** over the training cases in **each mini-batch**"
- Dropped parameters receive **zero gradients**
- Combining dropout with max-norm regularization, decaying learning rate, and high momentum further improve generalization