Dropout

A Preprint

Authors mention main problem of deep NN as overfitting and suggest adressing this problem with a technique called DropOut. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights.

Dropout is a stochastic regularization technique that improves generalization by randomly omitting units during training. Each unit is retained with a fixed probability p, effectively sampling a "thinned"subnetwork at every iteration. This can be seen as training an ensemble of 2^n subnetworks (where n is the number of units), all sharing weights. At test time, dropout is deactivated and weights are scaled by p to match the expected output at training time (see Figure

Dropout can be interpreted as implicit model averaging, offering a computationally efficient alternative to explicitly training and ensembling multiple models. It significantly reduces overfitting across a variety of tasks and architectures, including feed-forward nets and graphical models like Restricted Boltzmann Machines (RBMs).

0.1 Biological and Evolutionary Analogy

The motivation behind dropout is analogous to sexual reproduction in biology. Whereas asexual reproduction preserves well-adapted gene combinations, sexual reproduction encourages robustness by disrupting such co-adaptations. Similarly, dropout discourages hidden units from relying on specific configurations, fostering redundancy and adaptability.

0.2 Theoretical Perspective and Relation to Noise

Dropout can also be viewed as injecting noise into hidden activations, extending the idea of denoising autoencoders. Unlike previous work that added noise only to inputs, dropout operates on hidden layers and generalizes to supervised tasks. Empirically, retaining 80% of input units and 50% of hidden units tends to perform well.

0.3 Marginalization and Deterministic Analogues

While dropout introduces stochasticity, its effect can be analytically marginalized in simple settings to yield deterministic regularizers. Recent work explores this by deriving closed-form equivalents for specific noise models, offering insights into the regularization properties of dropout.

0.4 Formal Model Description

Let $y^{(l)}$ denote activations at layer l, and $W^{(l)}, b^{(l)}$ be the weights and biases. Without dropout, a feedforward layer computes:

$$z_i^{(l+1)} = W_i^{(l+1)} y^{(l)} + b_i^{(l+1)}, \quad y_i^{(l+1)} = f(z_i^{(l+1)})$$

With dropout, a binary mask $r_j^{(l)} \sim \text{Bernoulli}(p)$ is applied elementwise:

$$\tilde{y}^{(l)} = r^{(l)} \odot y^{(l)}, \quad z_i^{(l+1)} = W_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}$$

where f is a nonlinearity (e.g., sigmoid or ReLU).

1 Training Dropout Networks

1.1 Backpropagation and Optimization

Dropout networks are trained using stochastic gradient descent (SGD) similarly to standard neural networks, with the key difference being that for each training example, a subnetwork is sampled by randomly dropping units. Forward and backward passes are performed only on this thinned network, and gradients are averaged over the mini-batch. Parameters not used in a training example contribute zero gradient.

Standard SGD enhancements—such as momentum, learning rate schedules, and L2 regularization—remain effective in this setting. Additionally, max-norm regularization, which constrains the ℓ_2 norm of incoming weights to each unit to be at most c, is particularly beneficial. This constraint is enforced by projecting weight vectors onto a norm ball after each update. The hyperparameter c is selected via validation.

Combining dropout with max-norm regularization, high momentum, and large learning rates leads to further gains. Dropout-induced noise allows broader exploration of the weight space, while norm constraints prevent divergence. As the learning rate decays, the optimizer converges to a robust solution.

1.2 Unsupervised Pretraining

Dropout can also be integrated with unsupervised pretraining strategies such as stacked RBMs, autoencoders, or Deep Boltzmann Machines. Pretraining remains unchanged, but to maintain consistency during finetuning, pretrained weights should be scaled by $1/p$ to match the expected unit outputs during dropout.

While aggressive learning rates during finetuning may erase pretrained knowledge due to dropout noise, using smaller learning rates preserves this information and can lead to improved generalization compared to finetuning without dropout.

1.3 Limitations of Dropout

Despite its effectiveness, dropout has several limitations:

- Increased Training Time: Since each mini-batch involves sampling different subnetworks, convergence can be slower compared to training without dropout. Larger models and more epochs are often required.
- Incompatibility with Batch Normalization: Dropout and batch normalization interact poorly when used together, particularly during training, due to the mismatch in statistics caused by dropout-induced noise. In practice, dropout is often removed or applied only after batch normalization layers.
- Not Ideal for All Architectures: Dropout is less effective in certain models, such as convolutional networks or recurrent architectures, unless applied selectively (e.g., only in fully connected layers or on activations rather than weights).
- Tuning Complexity: Dropout introduces additional hyperparameters (e.g., dropout rates for different layers), and suboptimal values can hurt performance. Furthermore, combining dropout with other regularization techniques (like weight decay) may require careful balancing.
- Noisy Gradient Estimates: The randomness introduced by dropout can lead to high-variance gradient estimates, which may destabilize training if not mitigated by small learning rates, momentum, or large mini-batches.
- Inference-Time Approximation: The scaling trick used at test time is an approximation of model averaging, which may not fully capture the behavior of the ensemble of subnetworks.