Improving Neural Network Generalization

COMPSCI 371D — Machine Learning

Outline

Motivation

- Stochastic Gradient Descent (SGD) is the main algorithm for training neural networks
- However, without further attention, networks often fail to generalize
- Some fixes:
  - Regularization to shrink the hypothesis space: (momentum,) weight decay, early termination, and dropout
  - Making up data: data augmentation
- We can now increase depth
  - Vanishing and exploding gradients
  - Batch normalization

Regularization

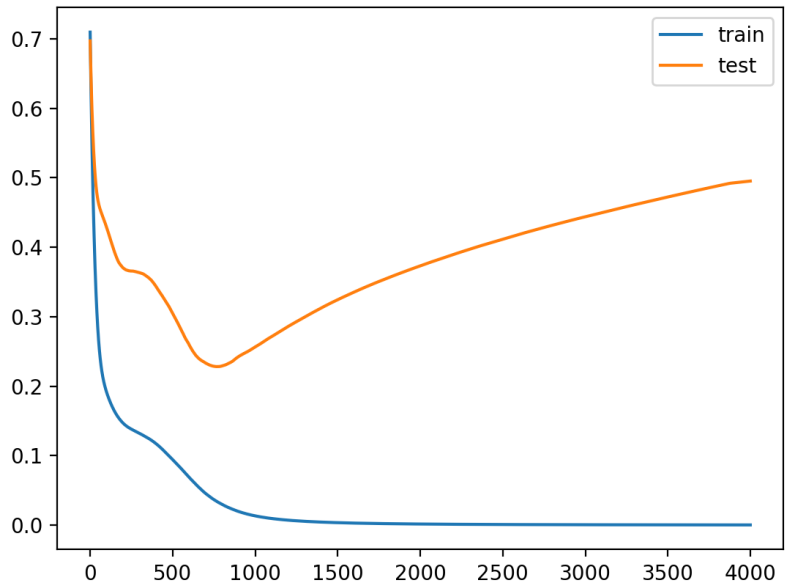
- The capacity of deep networks is very high: It is often possible to achieve near-zero training loss
- "Memorize the training set" $\Rightarrow$ overfitting
- All training methods use some type of regularization
- Regularization can be seen as *inductive bias*: Bias the training algorithm to find weights with certain properties
- Simplest method: *weight decay*, add a term $\lambda \|\mathbf{w}\|^2$ to $L_T(\mathbf{w})$
- Keeps the weights small (Tikhonov)
- Other proposals have been made, including early termination and dropout
- Often several or all methods are used in combination

Early Termination

- Early termination is also regularization
- Terminating training well before the L_T is minimized is somewhat similar to "implicit" weight decay
- Progress at each iteration is limited, so stopping early keeps us close to $\mathbf{w}_0$, which is a set of small random weights
- Therefore, the norm of $\mathbf{w}_t$ is restrained, albeit in terms of how long the learner takes to get there rather than in absolute terms

Informed Early Termination

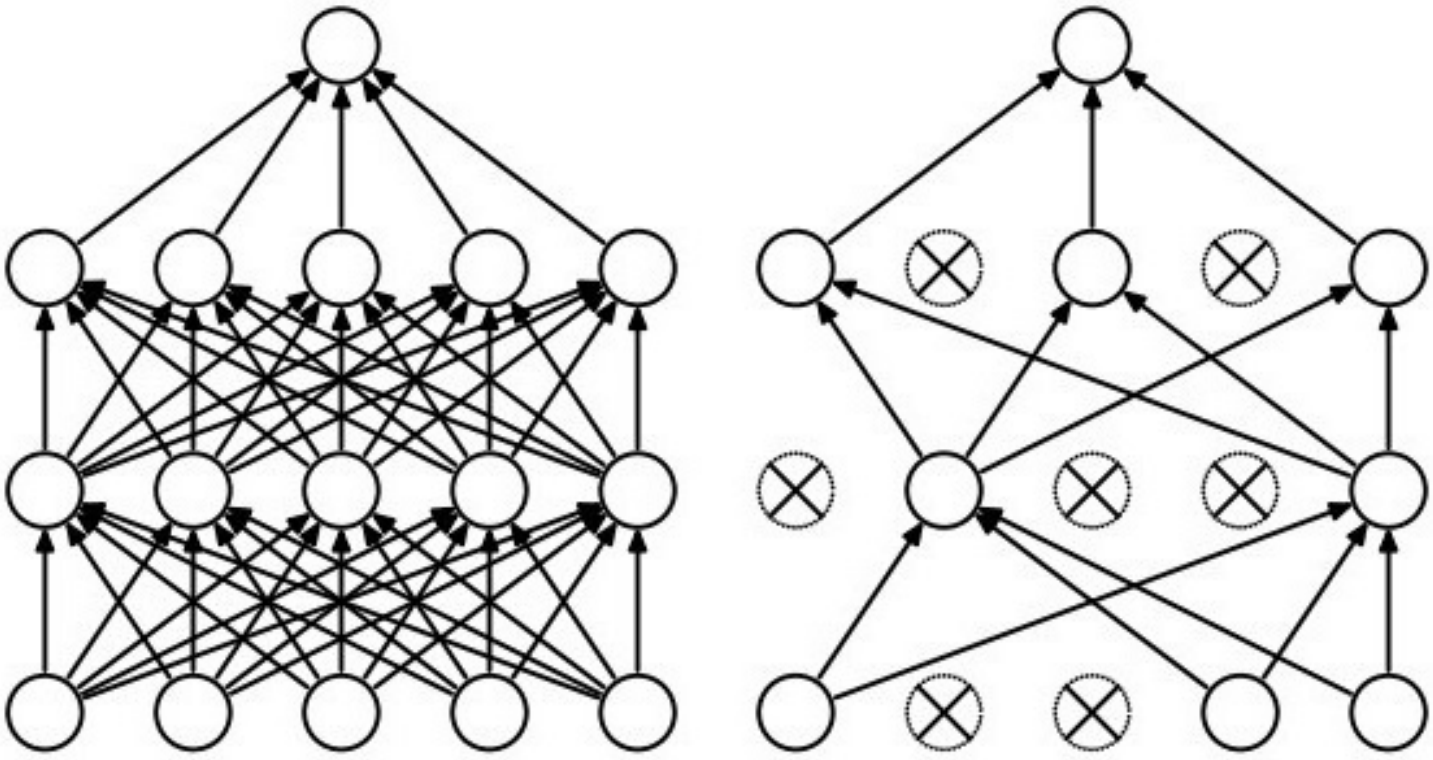
- A more informed approach to early termination stops when a validation risk (or, even better, error rate) stops declining
- This is arguably the most widely used regularization method



[plot from https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping]

Dropout

- *Dropout* inspired by ensemble methods (random forests): Regularize by averaging multiple predictors
- Key difficulty: It is too expensive to train an ensemble of deep neural networks
- Efficient (crude!) approximation:
  - Before processing a new mini-batch, flip a coin with $\mathbb{P}[\text{heads}] = p$ (typically $p = 1/2$) for each neuron
  - Turn off the neurons for which the coin comes up *tails*
  - Restore all neurons at the end of the mini-batch
  - When training is done, multiply all weights by p
- This is very loosely akin to training a different network for every mini-batch
- Multiplication by p takes the "average" of all networks
- There are flaws in the reasoning, but the method works



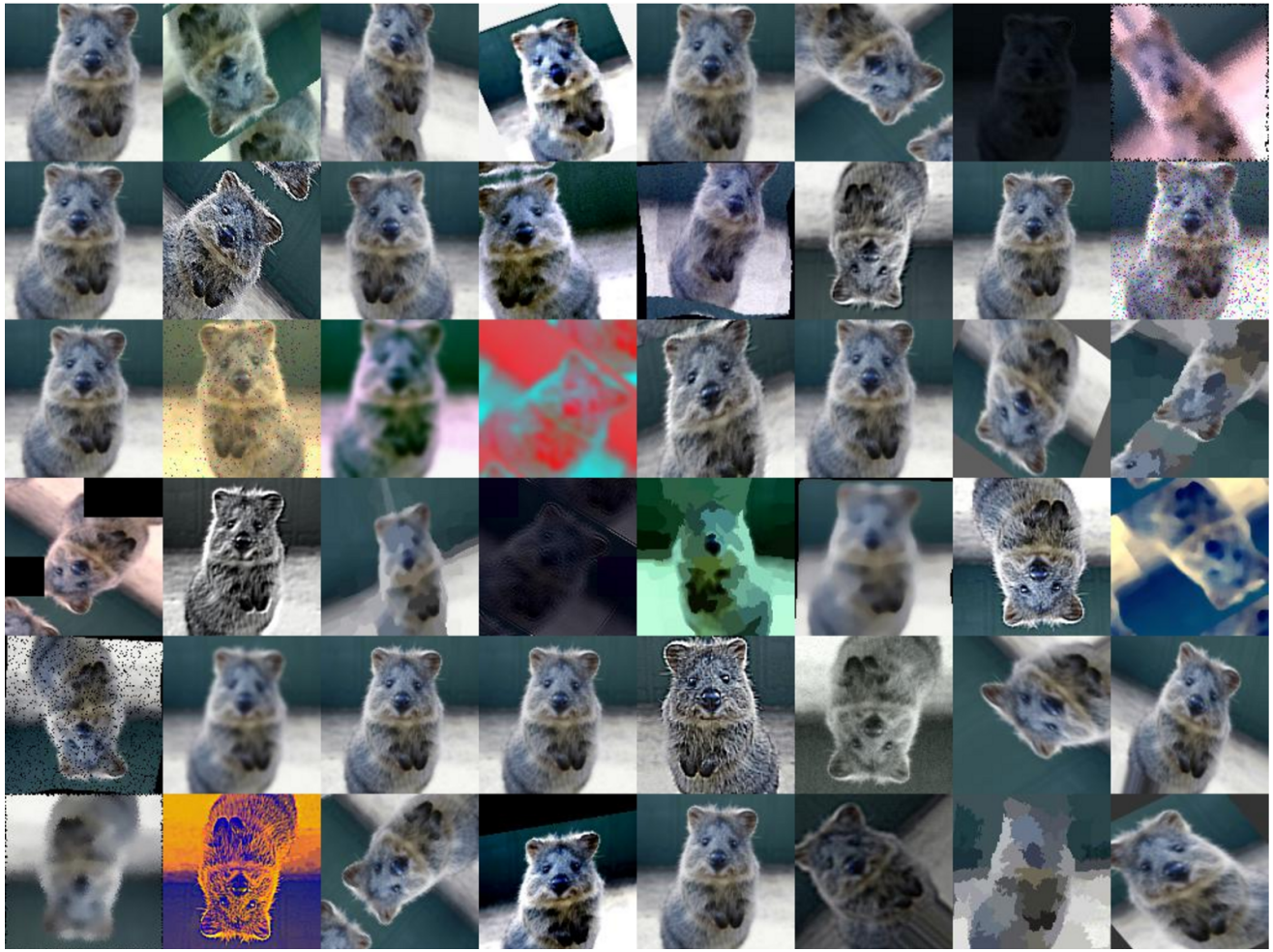
(a) Standard Neural Net

(b) After applying dropout.

Data Augmentation

- Data augmentation is not a regularization method, but combats overfitting
- Make new training data out of thin air
- Given data sample $(\mathbf{x}, y)$, create perturbed copies $\mathbf{x}_1, \ldots, \mathbf{x}_k$ of $\mathbf{x}$ (these have the same label!)
- Add samples $(\mathbf{x}_1, y), \ldots, (\mathbf{x}_k, y)$ to training set T
- With images this is easy. The $\mathbf{x}_i$s are cropped, rotated, stretched, re-colored, ... versions of $\mathbf{x}$
- One training sample generates k new ones
- T grows by a factor of $k + 1$
- Very effective, used almost universally
- Need to use realistic perturbations

Data Augmentation



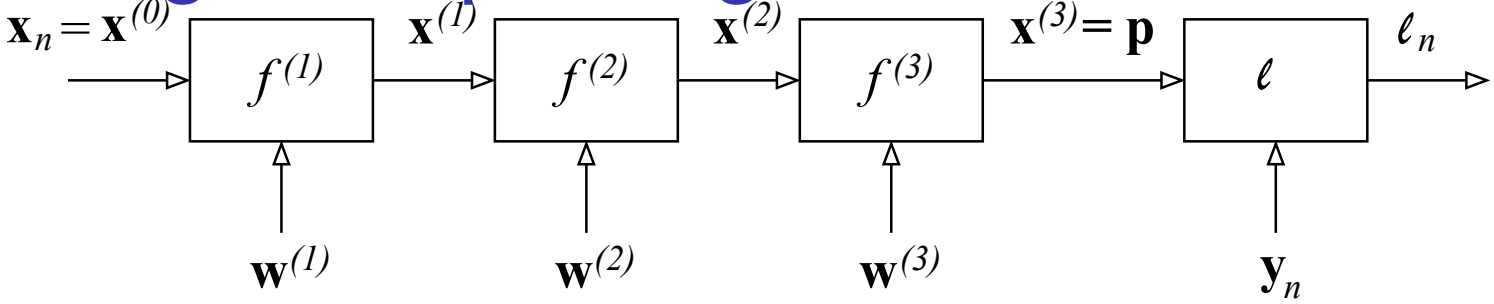
[image from https://algorithmia.com/blog/introduction-to-dataset-augmentation-and-expansion]

Current Trend: Go Deeper

[Material on this and subsequent slides is optional]

- If the output of the last layer comes from a ReLU, it is nonnegative
- Therefore, an additional layer, even with ReLU, can implement the identity by setting $V = I$ and $\mathbf{b} = 0$
- Therefore, more layers give more capacity (expressive power)
- So, why not go deeper?
- Two problems with greater capacity:
  - Overfitting
  - Vanishing or exploding gradients
- Overfitting can be controlled by regularization

Vanishing or Exploding Gradients



- The recursion $\frac{\partial \ell_n}{\partial \mathbf{x}^{(k-1)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(k)}} \frac{\partial \mathbf{x}^{(k)}}{\partial \mathbf{x}^{(k-1)}}$ yields
  $\frac{\partial \ell_n}{\partial \mathbf{x}^{(i)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(K)}} \frac{\partial \mathbf{x}^{(K)}}{\partial \mathbf{x}^{(K-1)}} \cdots \frac{\partial \mathbf{x}^{(i+1)}}{\partial \mathbf{x}^{(i)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(K)}} J_K \cdot \ldots \cdot J_{i+1}$
- Feedback signal (gradient) from loss ℓ_n to layer i, and therefore also $\frac{\partial \ell_n}{\partial \mathbf{w}^{(i)}} = \frac{\partial \ell_n}{\partial \mathbf{x}^{(i)}} \frac{\partial \mathbf{x}^{(i)}}{\partial \mathbf{w}^{(i)}}$, depends on the product $J^{(i)} = J_K \cdot \ldots \cdot J_{i+1}$ of layer Jacobians
- $\det(J^{(i)}) = \det(J_K) \cdot \ldots \cdot \det(J_{i+1})$ determines (pun intended) the magnitude of the gradient
- Vanishing gradients choke information flow: No progress in early layers
- Exploding gradients cause instability during training

Batch Normalization

- Ideally, we would like the norms of all activations $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(K)}$ to be equal ($\det(J_i) \approx 1$)
- Suppose that we could interpose a layer β_k between layers k and $k+1$ that subtracts the mean of all possible outputs $\mathbf{x}^{(k)}$ from layer k and divides by their standard deviation:

  $\hat{x}_{nk}^{(c)} = \frac{x_{nk}^{(c)} - \mu_k^{(c)}}{\sigma_k^{(c)}}$ for component c of $\mathbf{x}_{nk}$ (sample n, layer k)
- Then, layer k together with β_k has normalized outputs
- If we do this for all layers, all layers transform normalized inputs to normalized outputs
- Problem 1: We don't know "all possible outputs $\mathbf{x}^{(k)}$ from layer k" because the network changes during training
- Problem 2: We limit the expressive power of the network

Batch Normalization

- Normalize each activation by an *estimate* of its mean and standard deviation
- During training, compute the estimate over each mini-batch
- During inference, use the the mean estimate over all mini-batches
- Let x be a scalar activation just before a non-linearity
- Let μ, σ be the sample mean and standard deviation of x over the current mini-batch
- Pass x through a Batch Normalization (BN) module that
  - Normalizes each component of $\mathbf{x}$: $\hat{x} = \frac{x-\mu}{\sigma}$
  - Computes $z = \gamma\hat{x} + \beta$
- The *learnable* parameters γ and β restore the layer's expressive power

Normalization and De-Normalization

- Wait, what? What is the point of normalizing x to $\hat{x} = \frac{x-\mu}{\sigma}$ and then let the network undo the normalization by $z = \gamma\hat{x} + \beta$?
- Why we *must* do this: If we don't, we restrict the expressive power of the layer
- Why we *can* do this: The de-normalization is local
- If, say, $\gamma = 2$ in layer k, then mini-batch inputs to layer $k+1$ are twice as large, and will be normalized again in layer $k+1$ by a σ that is also twice as big
- BN in layer k accounts for *all* the γs in previous layers
- The γs in different layers do not multiply
- Last layer does not have batch normalization

Example

- Only look at standard deviations for simplicity
  (Similar considerations hold for means)
- Start with $\gamma_1 = 1$ in layer 1
- Outputs of layer 2 have standard deviation σ_2 before BN
- Now change γ_1 to $\gamma_1' = 2$
- Outputs from layer 2 now have $\sigma_2' = 2\sigma_2$ before BN
- They are twice as big, but BN divides them by a standard deviation that is twice as big as well
- μ, σ statistics of the outputs from layer 2 are unchanged *after BN*
- Key point: γ_1, β_1 affect μ_2, σ_2

Going Deeper

- With batch normalization, gradients are tame
- Need to compute BN Jacobians for back-propagation
- Need to store estimates of μ, σ for inference
- Everything else remains the same
- Network depth is no longer a problem for training
- Regularization reduces overfitting for deep networks
- Networks with BN often have tens or hundreds of layers
- A network with 1000 layers was shown to be trainable

  Deep Residual Learning for Image Recognition, He *et al.*, ArXiv, 2015
- Of course, regularization and data augmentation are now even more crucial