CS60010: Deep Learning

Spring 2023

Sudeshna Sarkar

Regularization and Optimization

1 Feb 2023

Regularization

- Machine learning is concerned more about the performance on the test data than on the training data.

- Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Regularization Strategies

- Adding restrictions on parameter values
- Adding constraints that designed to encode specific kinds of prior knowledge
- Use of ensemble methods/dropout
- Dataset augmentation

In practical Deep Learning scenarios, the model used is often a large model that has been regularized appropriately

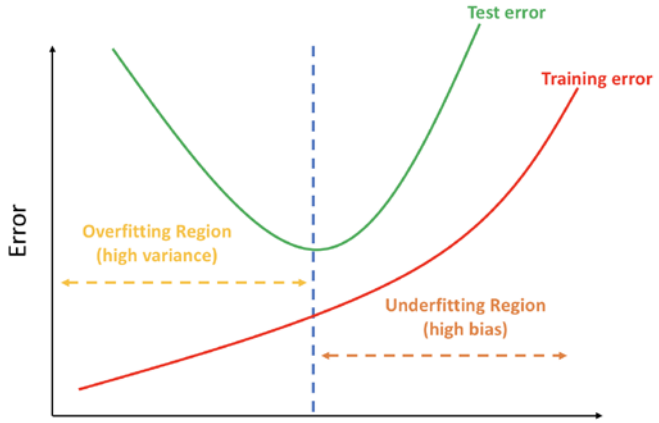
Parameter Norm Penalties

Adding penalties for high norm of parameters

- limits the capacity of the model by adding penalty $\Omega(\theta)$to the objective function resulting in

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- This will limit the parameters to grow in an unbounded manner, thus restricting the complexity



Strong regularization (large α) ➔ lower variance and higher bias

Weak regularization (small α) ➔ higher variance and lower bias

L2 Parameter Norm Regularization

$$\Omega(w) = \frac{1}{2}||w||_2^2 = \frac{1}{2}w^T w$$

The regularized objective function.

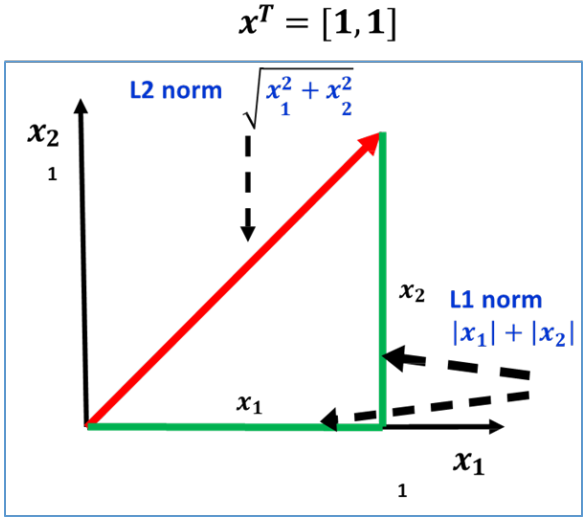
$$\tilde{J}(w) = J(w) + \frac{\alpha}{2}w^T w$$

The gradient is $\nabla_w \tilde{J}(w) = \nabla_w J(w) + \alpha w$

So, the update step is

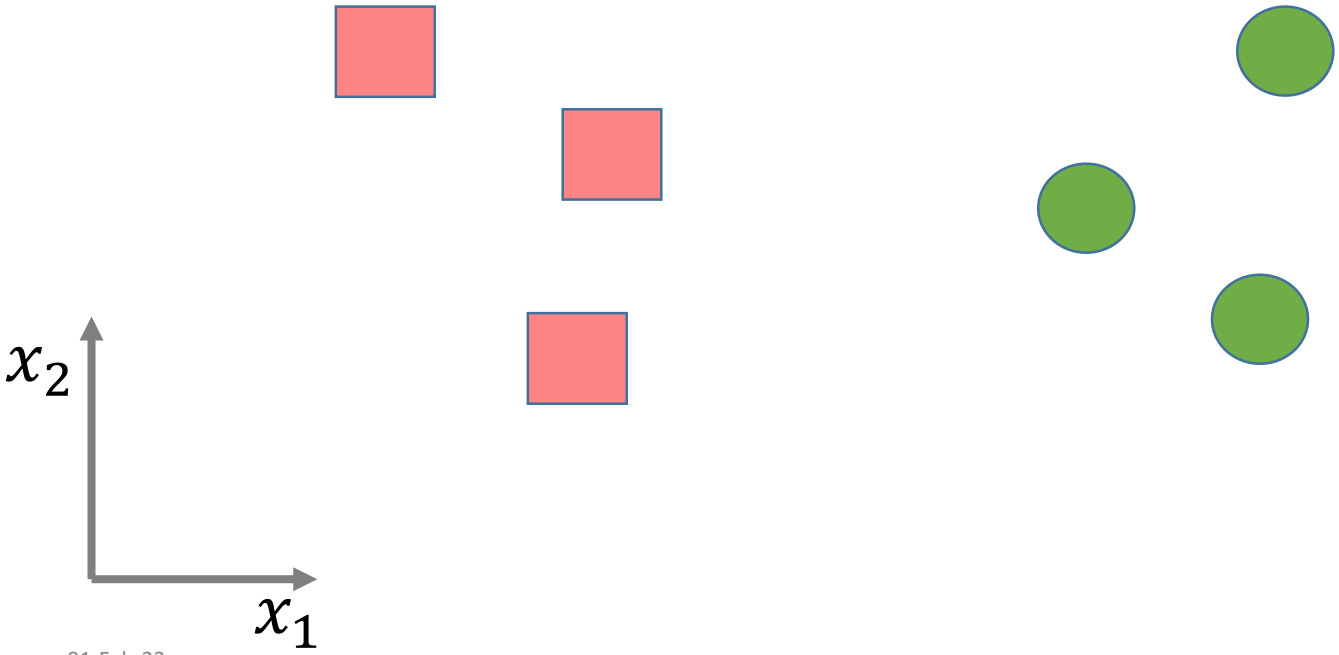
$$w \leftarrow w - \eta(\nabla_w J(w) + \alpha w)$$
$$= (1 - \eta\alpha)w - \eta\nabla_w J(w)$$

The addition of weight decay term modifies the learning rule to shrink the weight vector further before performing the usual gradient update

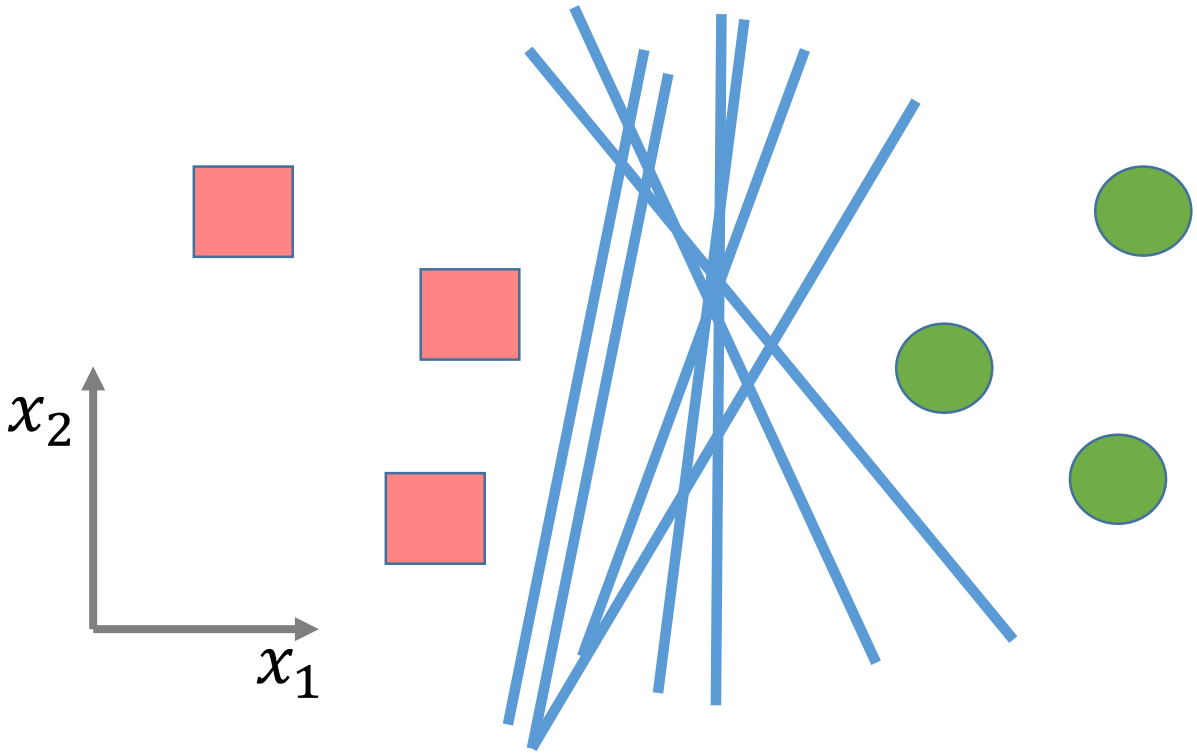


L2 (also known as weight decay)

Training Data

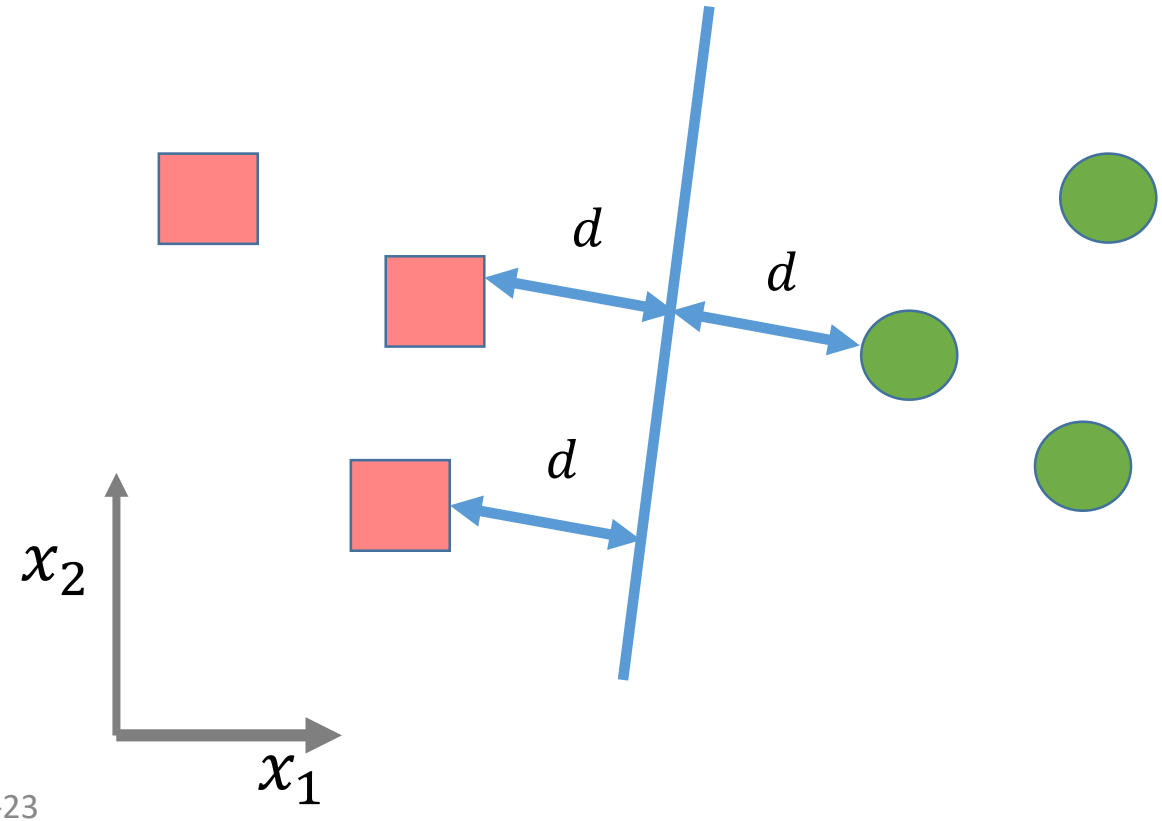


Training Data



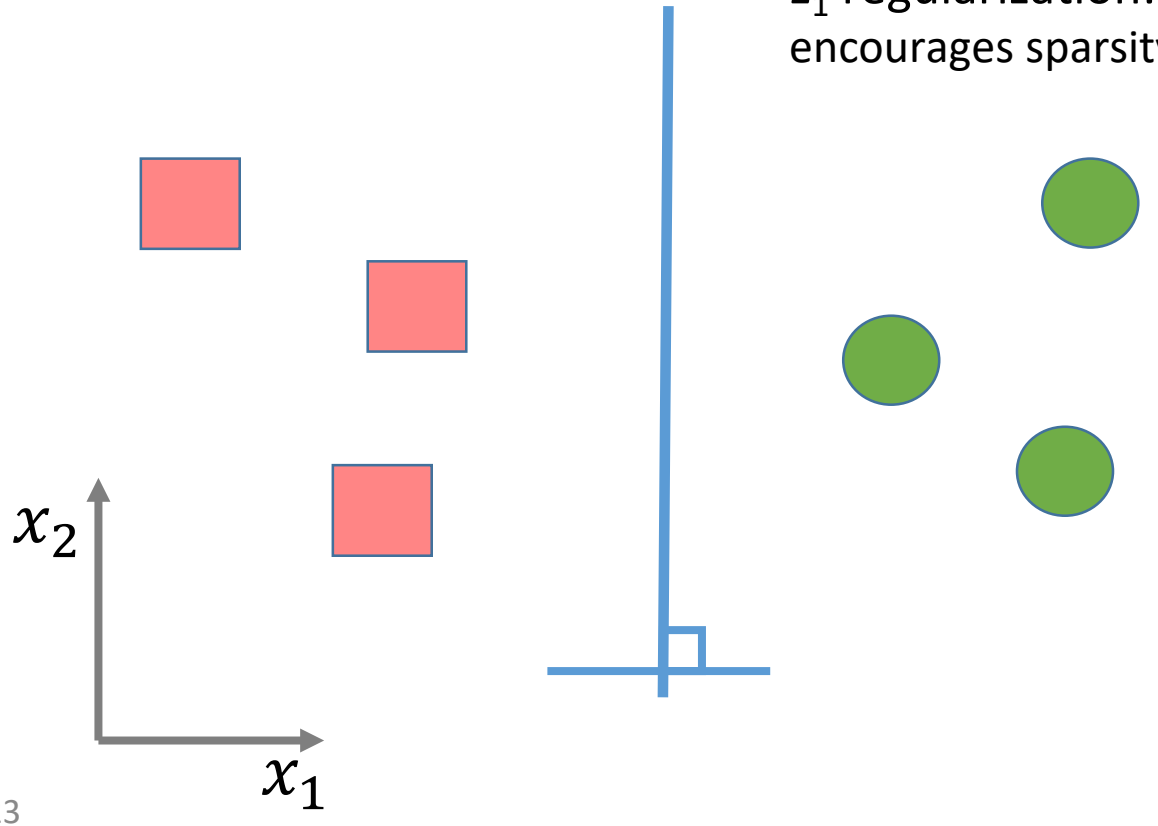
Optimal Hyperplane (L2)

L_2 regularization:
Weights decay



Sparsity (L1)

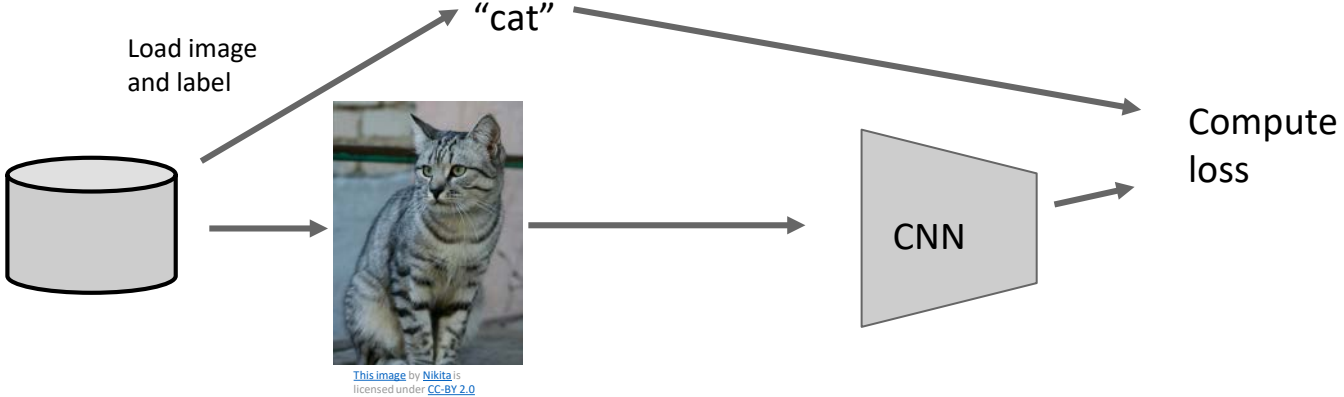
L_1 regularization:
encourages sparsity



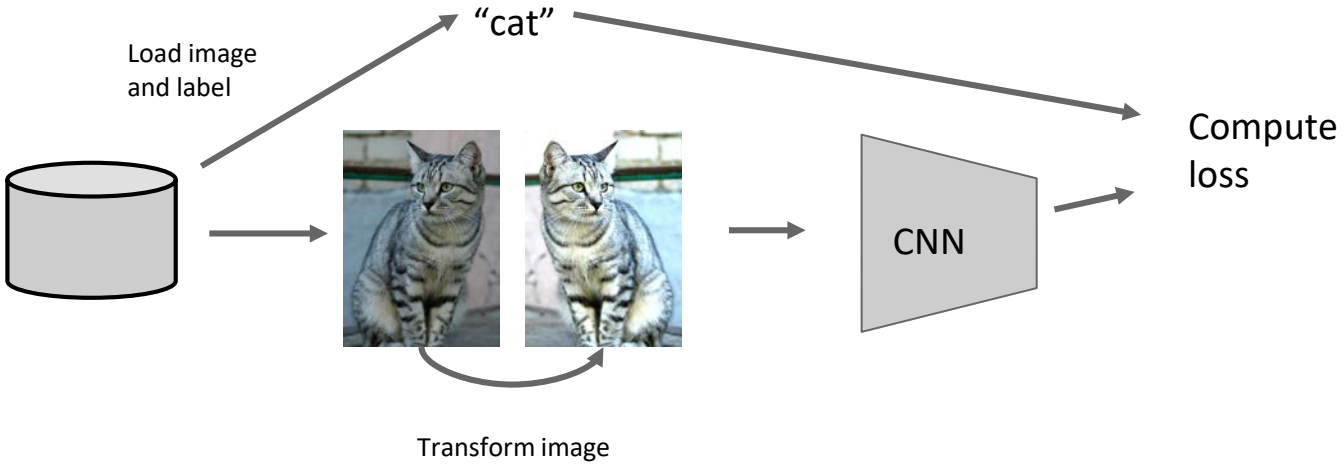
Data Augmentation and Noise Robustness

Regularization: Data Augmentation

Dataset augmentation is a technique that creating fake data and adds it to the training set

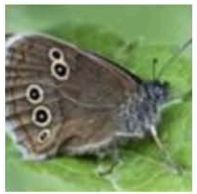


Regularization: Data Augmentation

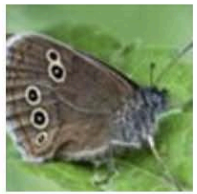


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Data Augmentation

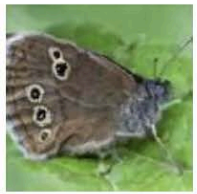


Affine Distortion

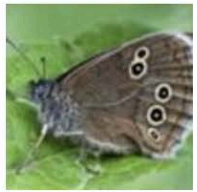


Noise

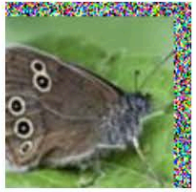
Elastic Deformation



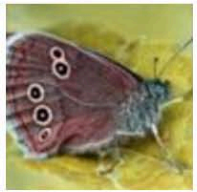
Horizontal flip



Random Translation

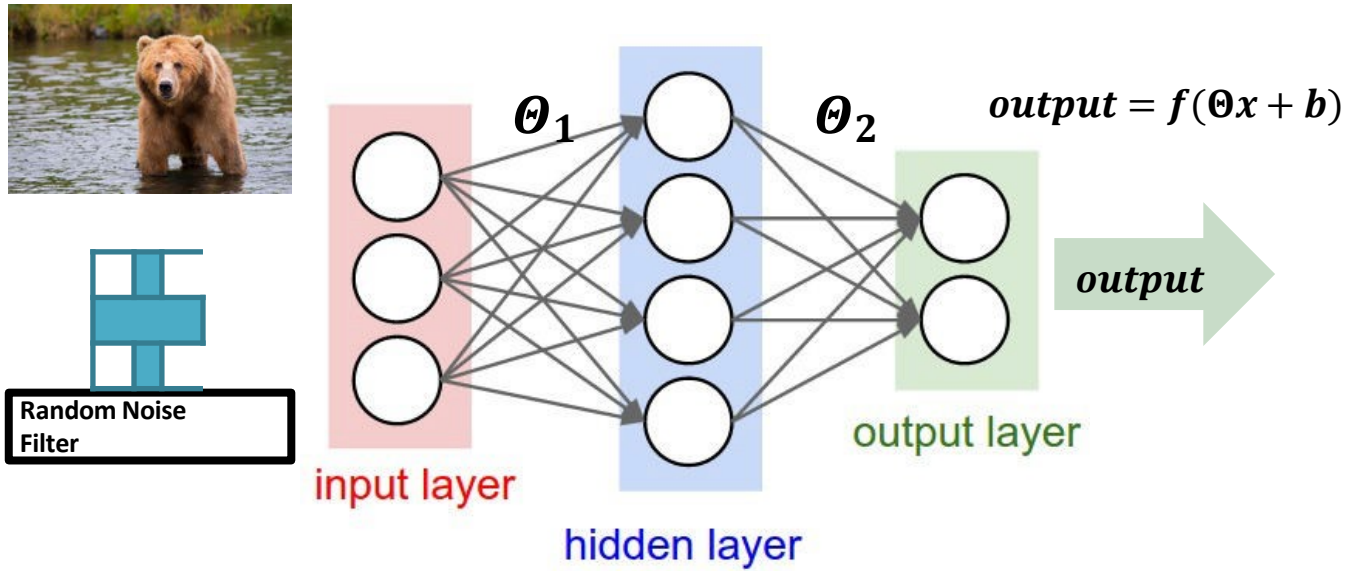


Hue Shift



Injecting noise (Training data)

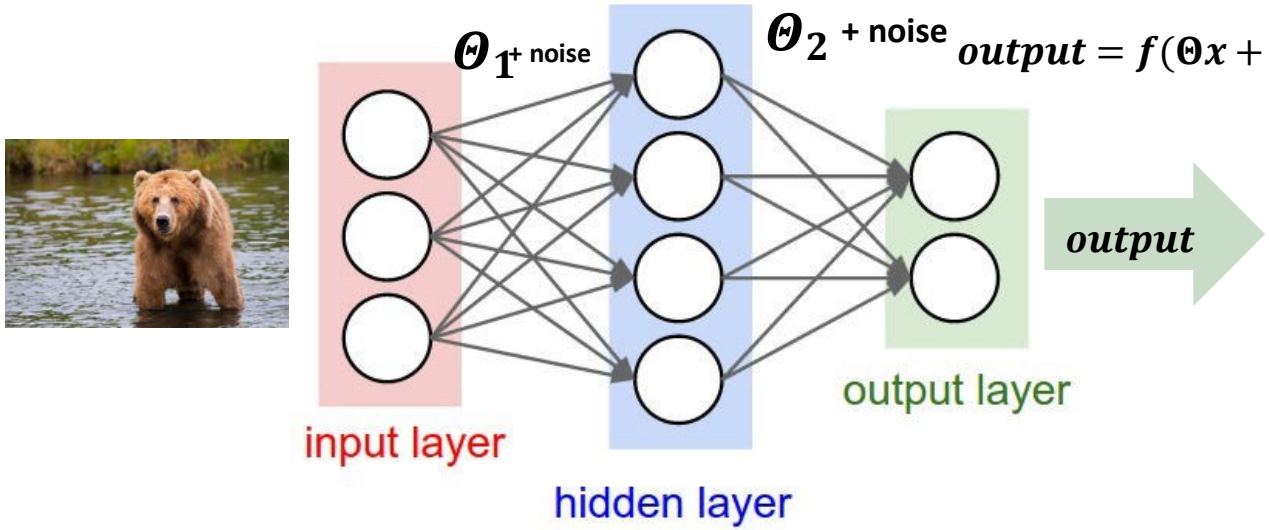
Injecting random noise into input data to improve robustness



Injecting noise (Weight)

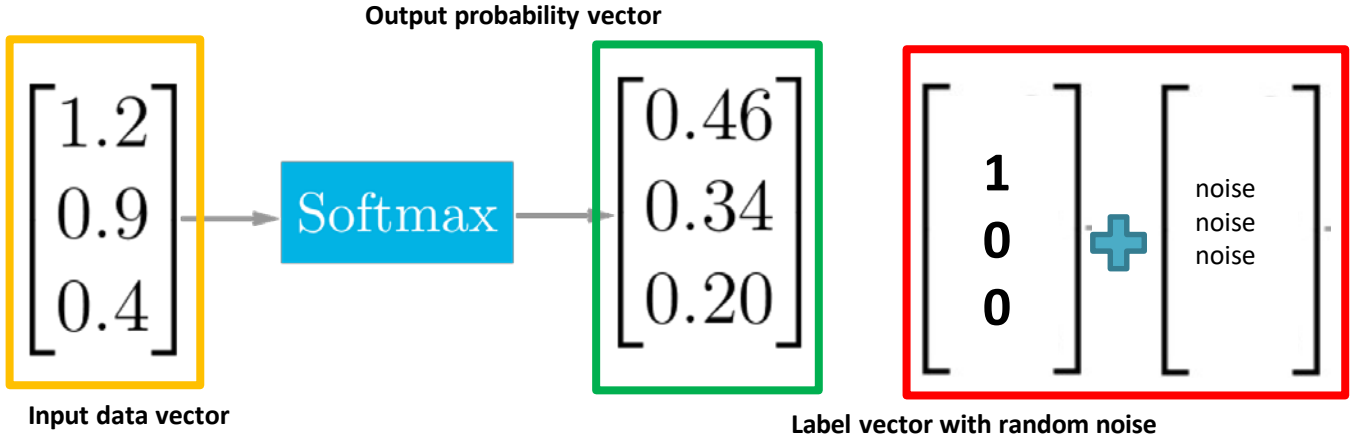
2. Injecting random noise into weight to improve robustness

This makes the model relatively insensitive to small variations in the weights



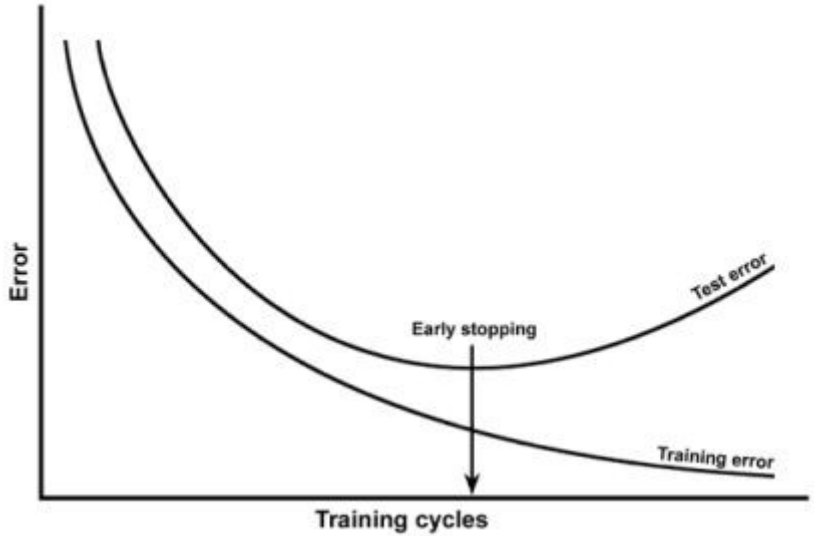
Injecting noise (Label)

Most datasets have some amount of mistakes in the $\boldsymbol{y}$ labels



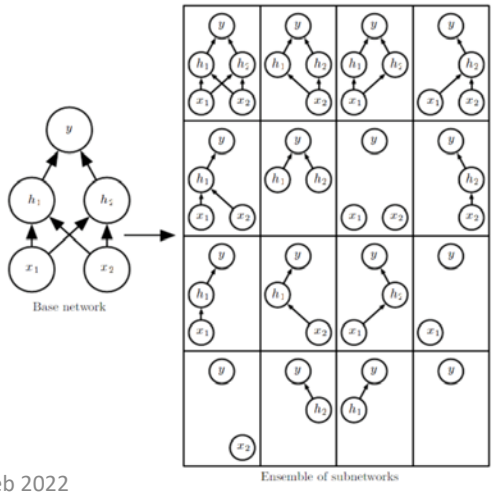
Early stopping

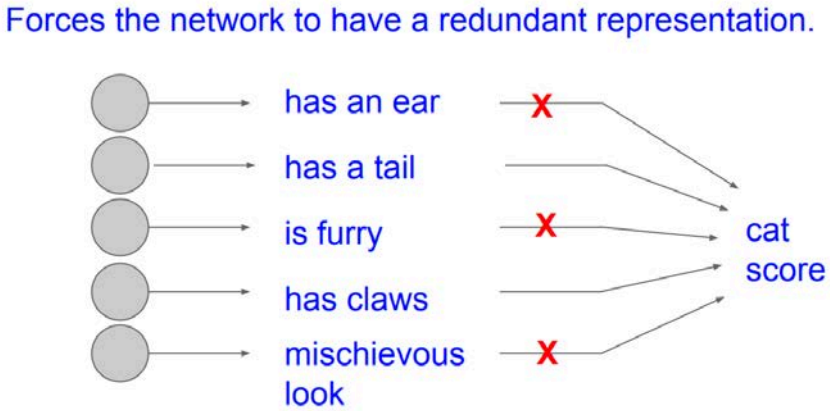
Learning too much iterations causes overfitting



Regularization Strategies: Dropout

- Bagging is a technique for reducing generalization error through combining several models (Breiman, 1994)
- Bagging: (1) Train k different models on k different subsets of training data, constructed to have the same number of examples as the original dataset through random sampling from that dataset with replacement
- Bagging: (2) Have all of the models vote on the output for test examples
- Dropout is a computationally inexpensive but powerful extension of Bagging
- Training with dropout consists of training sub-networks that can be formed by removing non-output units from an underlying base network

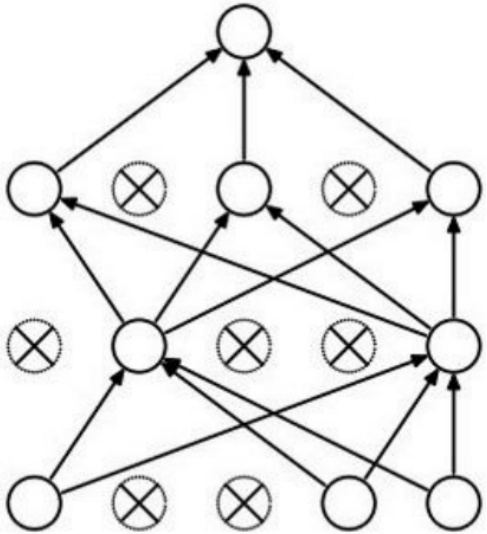




Images courtesy: Goodfellow et. al., Karpathy et. al.

Dropout – At Test Time

- Ideally, the randomness would have to be integrated out.
- Monte Carlo approximation: Do many forward passes with different random neurons dropped out. Then average out all predictions.
- An approximation to this approximation:
  - Can this be done in a single forward pass!
  - Can this be done without dropping out any neuron during forward pass at test time!
  - 1st way: Get the output of the network at test time with all neurons on. Scale down this by multiplying it with the probability value with which neurons are dropped during training.
  - 2nd way: During training compute the output of the network that you get after dropping out neurons with probability ‘p’. During training itself, scale up this by multiplying it with (1/p). At test time, get the output as what is coming by keeping all the neurons on.



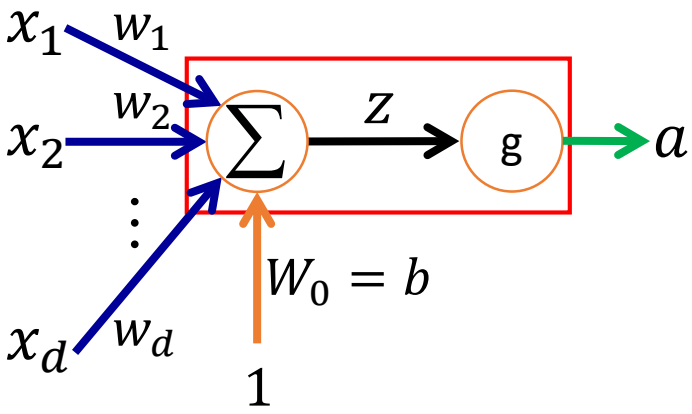
Batch Normalization (batchnorm)

- Training deep neural networks can be sensitive to the initial random weights and configuration of the learning algorithm.
- The distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated.
- *Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch*. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Batch Normalization

- BN mitigates the interdependency between layers during training.
- BN makes the optimisation landscape smoother

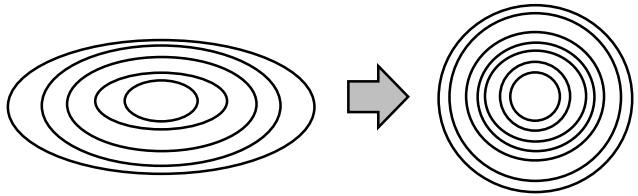
Batch Normalization

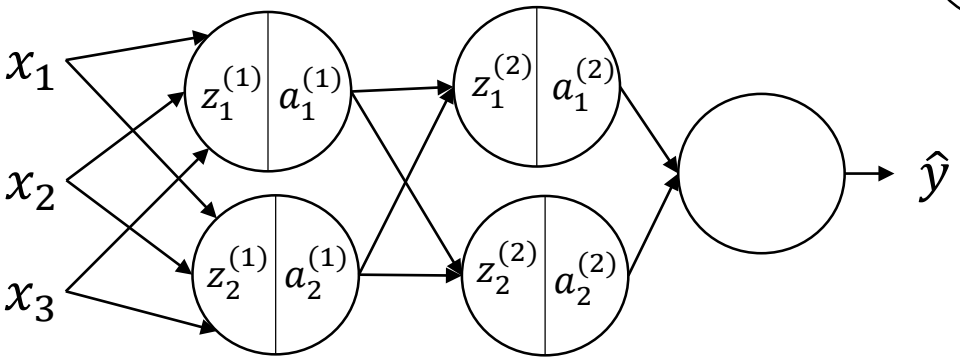


$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

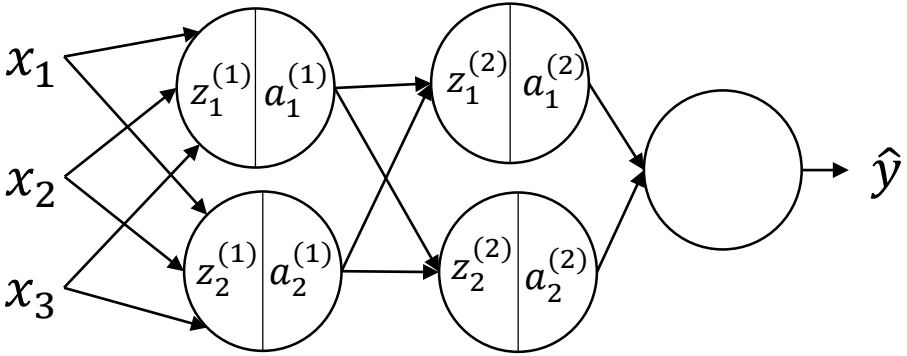
$$X = \mathrm{X} - \mu$$

$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)^2}$ (elementwise)





Batch Normalization:1. Normalize net inputs



$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} z_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(z_j^{(i)} - \mu_j\right)^2$$

$$z'^{[i]}_j = \frac{z_j^{(i)} - \mu_j}{\sigma_j}$$

In practice,

$$z'^{[i]}_j = \frac{z_j^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

BatchNorm Step 2: Pre-Activation Scaling

$$z'^{[i]}_j = \frac{z^{(i)}_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

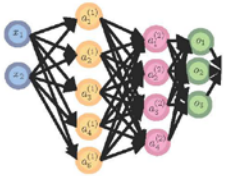
$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

Controls the spread

Controls the mean

Learnable parameters

BatchNorm Step 1 & 2 Summarized



$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sigma_j}$$

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

x_1 → $z^{(1)}_1$ → $z'^{(1)}_1$ → $a'^{(1)}_1$ → $a^{(1)}_1$ first hidden layer

$z^{(2)}_1$ → $z'^{(2)}_1$ → $a'^{(2)}_1$ → $a^{(2)}_1$ second hidden layer

...