
بسم الله الرحمن الرحیم

Deep Learning

Mohammad Reza Mohammadi

2021

Parameter Norm Penalties

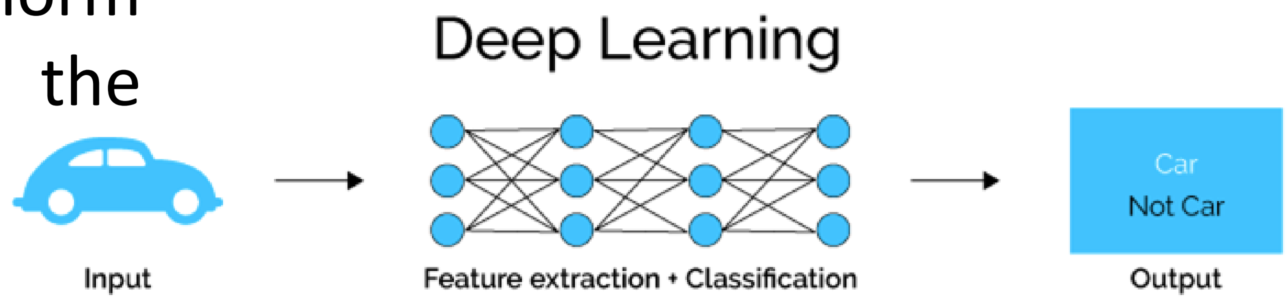
- We can limit the capacity of models by adding a parameter norm penalty to the objective function

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- We typically choose a norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized

$$g\left(\boldsymbol{W}^{(T)}\boldsymbol{x} + \boldsymbol{b}\right)$$

- We use $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while $\boldsymbol{\theta}$ denotes all of the parameters



L2 Parameter Regularization

- This regularization strategy drives the weights closer to the origin

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

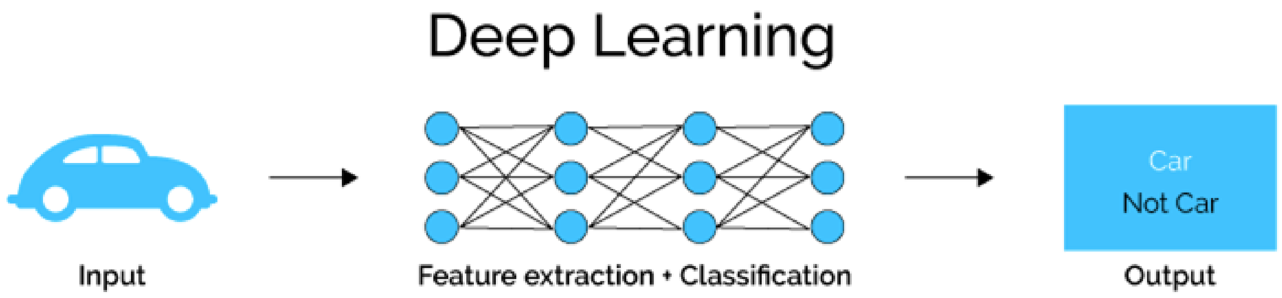
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

- Gradient of the regularized objective function:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

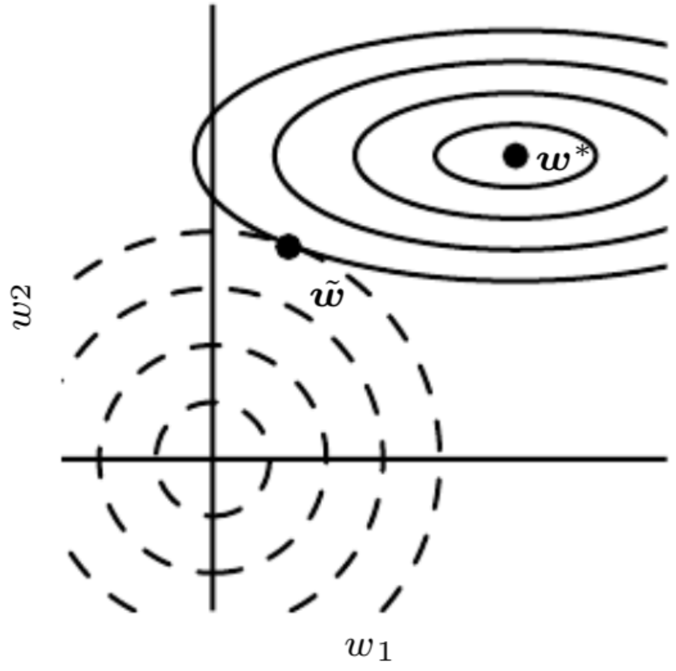
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})\right)$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$



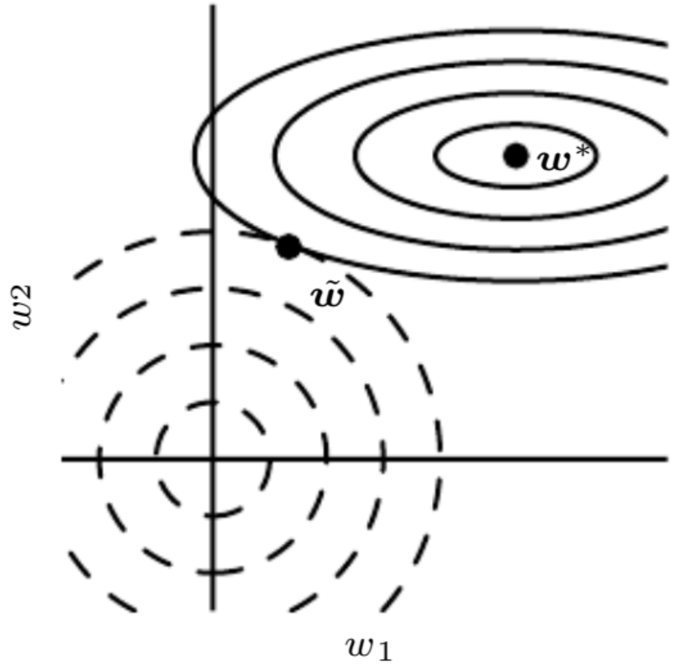
L2 Parameter Regularization

- Solid ellipses represent contours of equal value of the unregularized objective
- Dotted circles represent contours of equal value of the L2 regularizer
- At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium
- Objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$
  - the regularizer has a strong effect
- Objective function is very sensitive to movements away from $\boldsymbol{w}^*$ in the second dimension
  - the regularizer has a little effect



L2 Parameter Regularization

- Only directions along which the parameters contribute significantly to reducing the objective function are preserved relatively intact
- Unimportant directions are decayed away through the use of the regularization throughout training



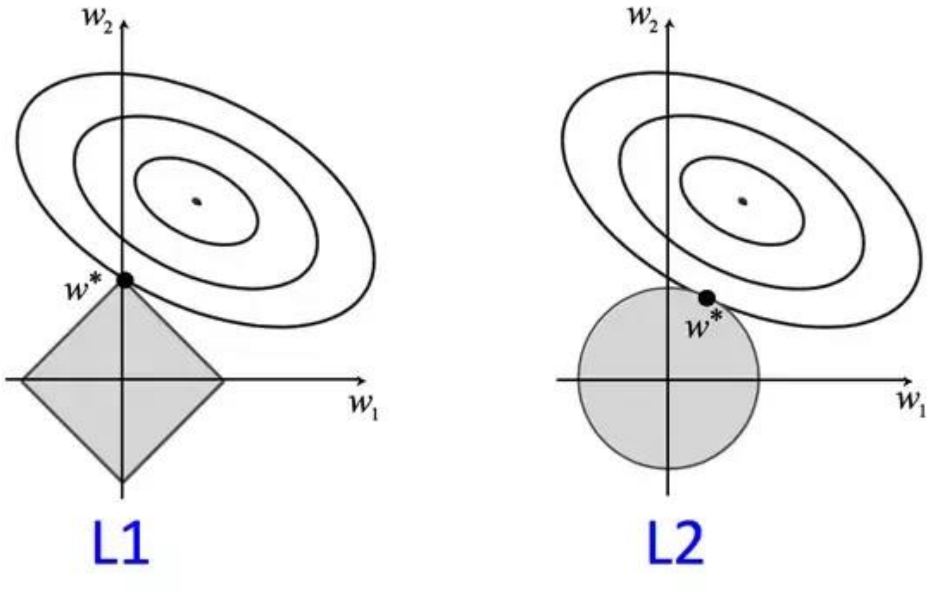
L1 Parameter Regularization

- L1 regularization on the model parameter $\boldsymbol{w}$ is defined as:

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

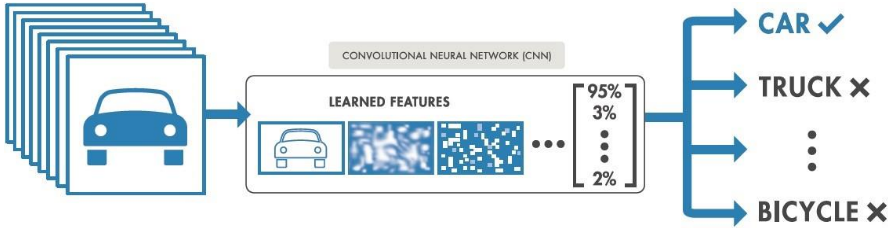
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

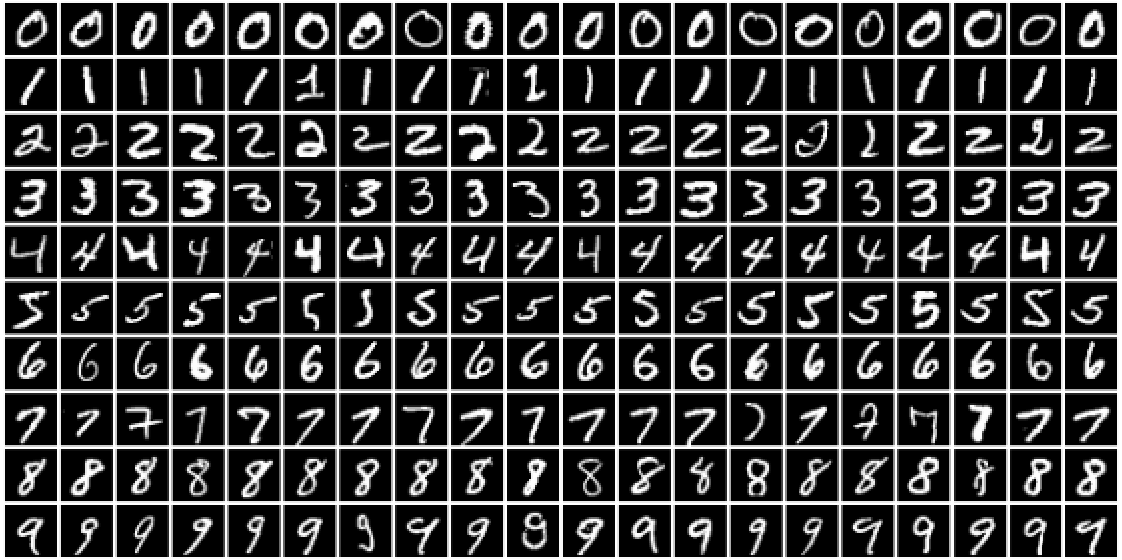
$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \, \mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$



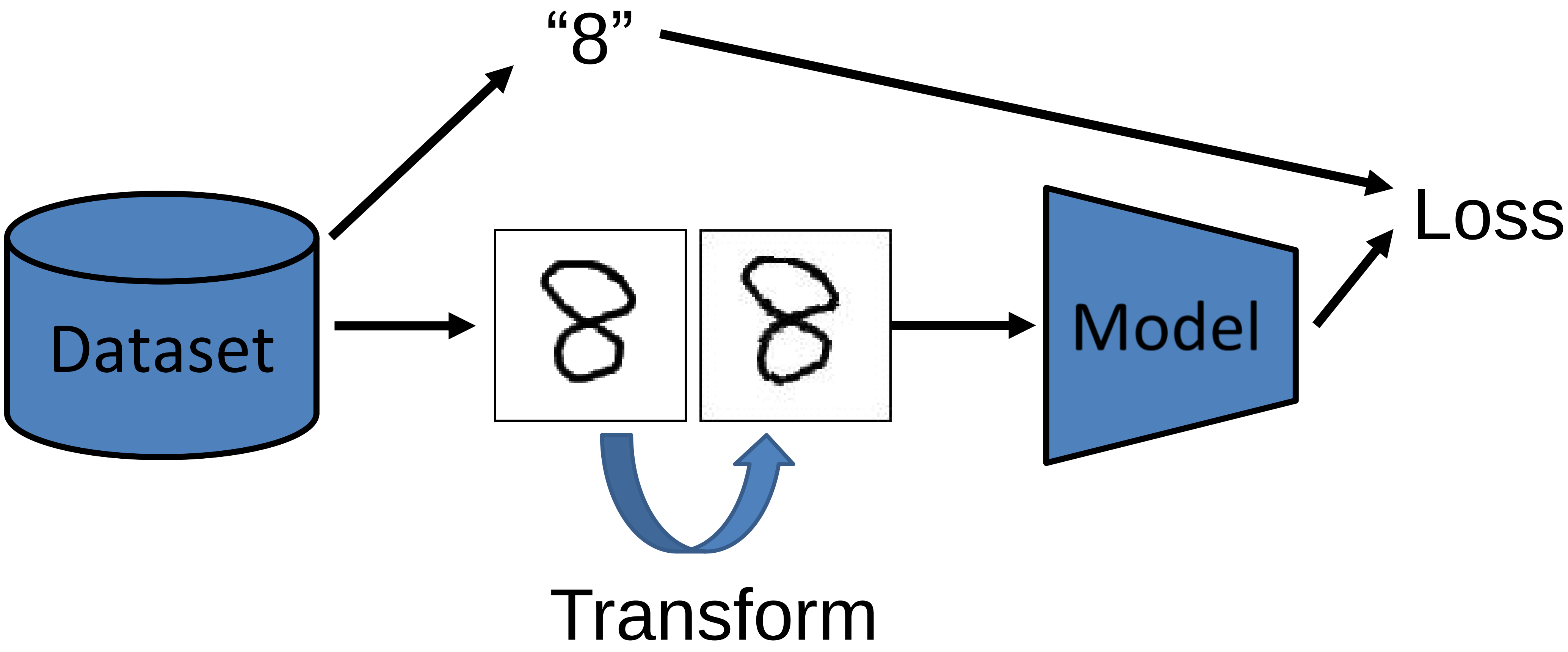
Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data
- The data collection process is a challenging and tedious task
- We can create fake data and add it to the training set
- This approach is easiest for classification
- We can generate new (x, y) pairs easily just by transforming the x

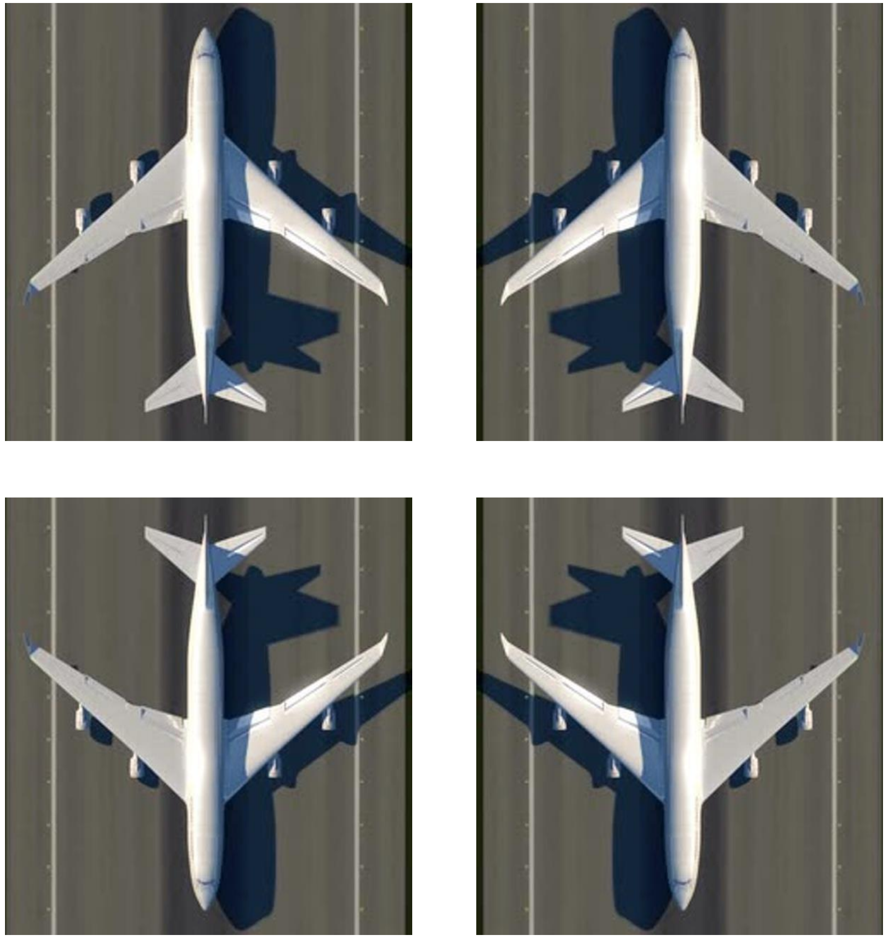


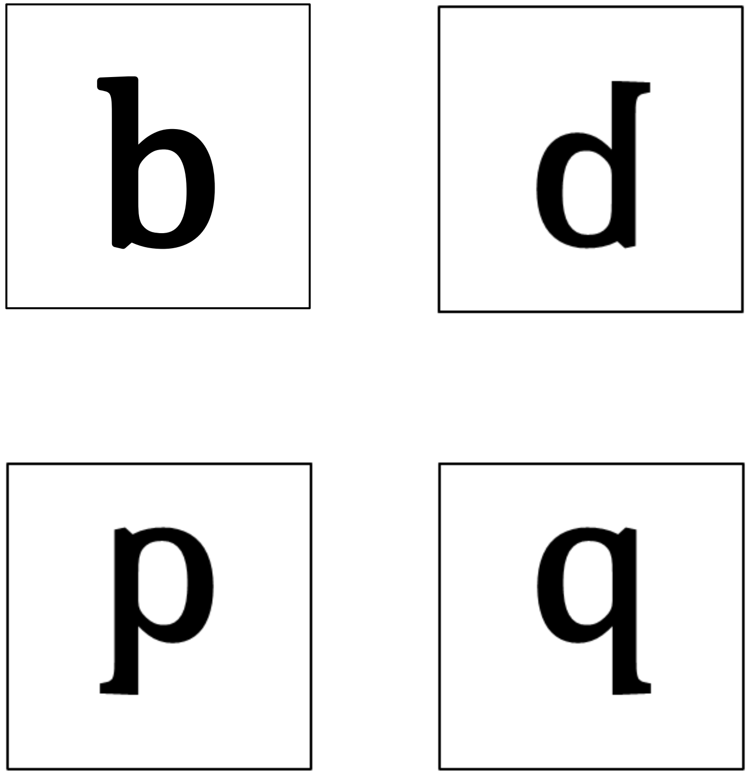


Dataset Augmentation



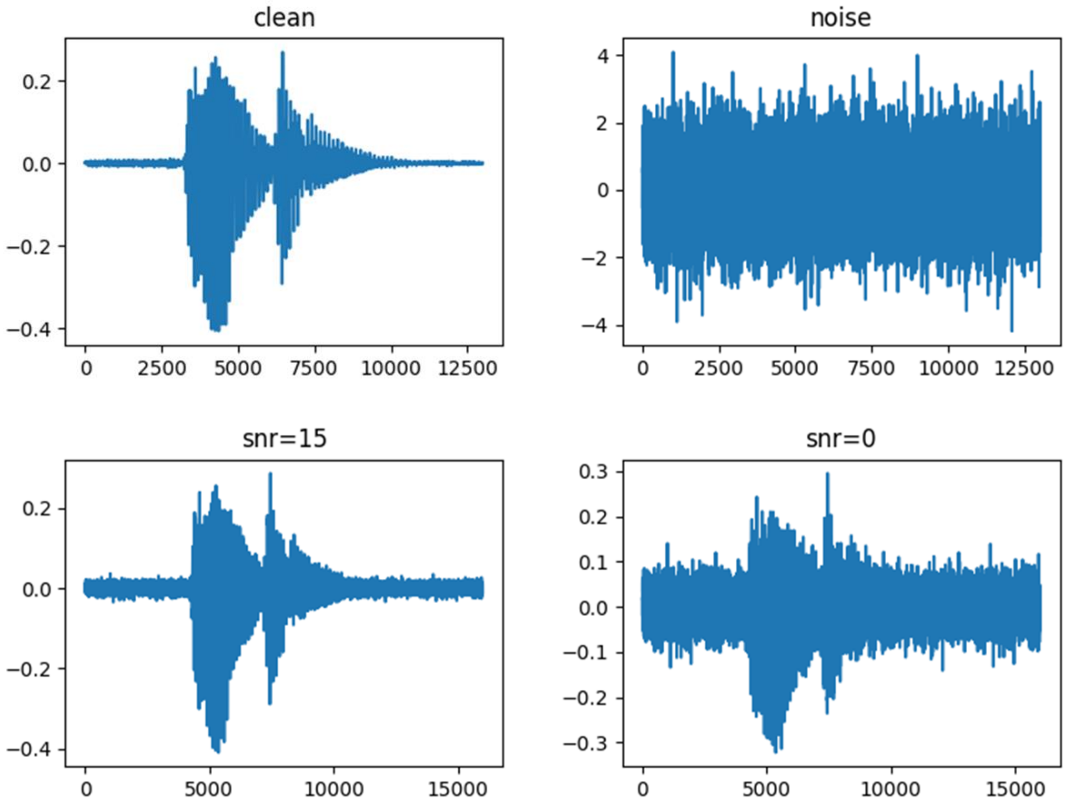
Dataset Augmentation: Flip image





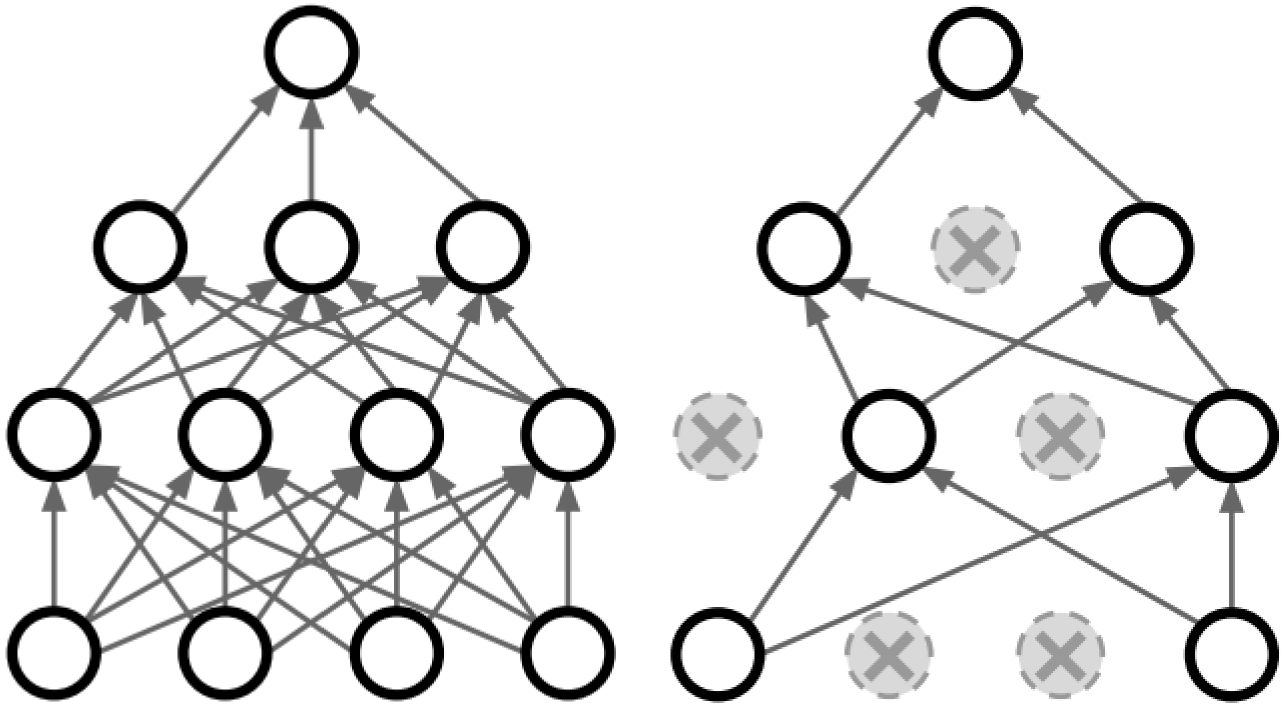
Dataset Augmentation

- Injecting noise in the input to a neural network can also be seen as a form of data augmentation
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input
- Noise injection also works when the noise is applied to the hidden units



Dropout

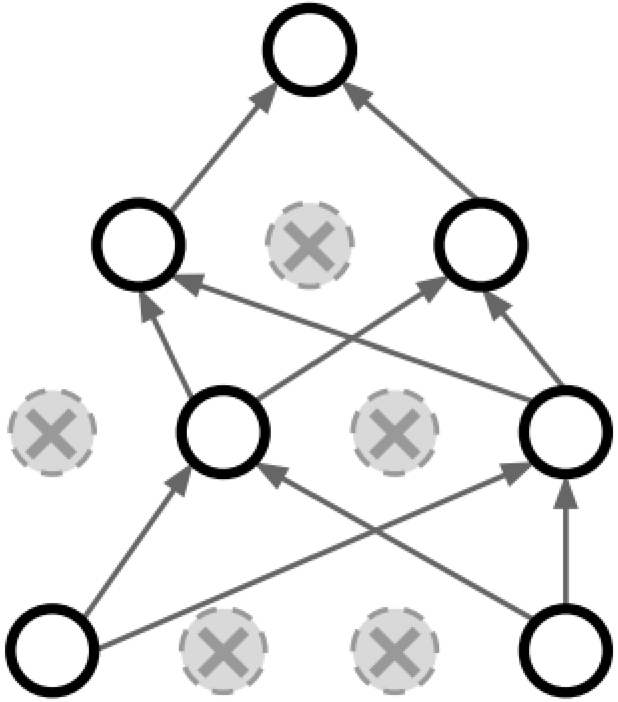
- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyperparameter; 0.5 is common



Dropout

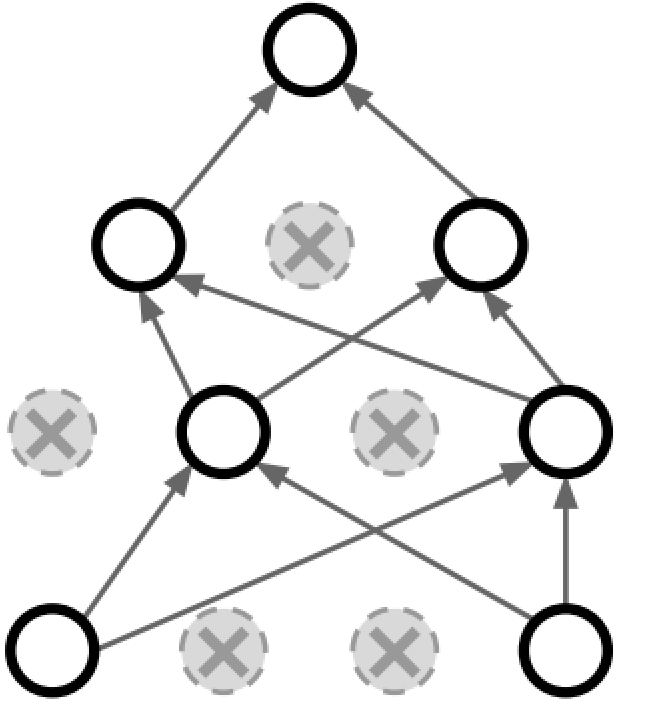
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  """ X contains the data """
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```



Dropout

- How can this possibly be a good idea?
- Forces the network to have a redundant representation
- Prevents co-adaptation of features



Dropout

- Dropout is training a large ensemble of models (that share parameters)
- Each binary mask is one model
- An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!