COSI 165B
Deep Learning

Chuxu Zhang
Computer Science Department
Brandeis University

2/22/2021

❑ Backpropagation

1: Compute the values of $z \in \mathbb{R}^m$, $a \in \mathbb{R}^m$, and $o \in \mathbb{R}$
2: Compute

$$\delta^{[2]} \triangleq \frac{\partial J}{\partial o} = (o - y) \in \mathbb{R}$$
$$\delta^{[1]} \triangleq \frac{\partial J}{\partial z} = (W^{[2]\top}(o - y)) \odot 1\{z \geq 0\} \in \mathbb{R}^{m \times 1}$$
(by eqn. (3.12) and (3.13))

3: Compute

$$\frac{\partial J}{\partial W^{[2]}} = \delta^{[2]} a^\top \in \mathbb{R}^{1 \times m} \quad \text{(by eqn. (3.5))}$$
$$\frac{\partial J}{\partial b^{[2]}} = \delta^{[2]} \in \mathbb{R} \quad \text{(by eqn. (3.6))}$$
$$\frac{\partial J}{\partial W^{[1]}} = \delta^{[1]} x^\top \in \mathbb{R}^{m \times d} \quad \text{(by eqn. (3.7))}$$
$$\frac{\partial J}{\partial b^{[1]}} = \delta^{[1]} \in \mathbb{R}^m \quad \text{(as an exercise)}$$

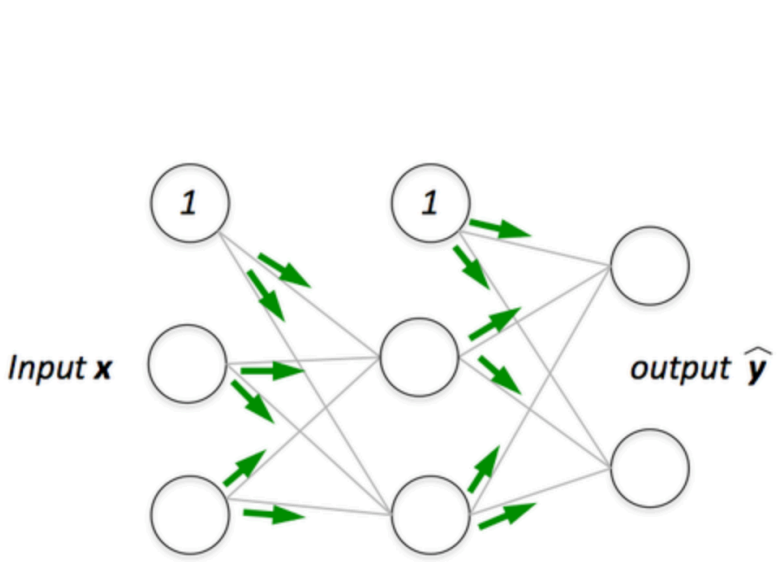
1: Compute and store the values of $a^{[k]}$'s and $z^{[k]}$'s for $k = 1, \ldots, r$, and J.
. ▷ This is often called the "forward pass"
2: .
3: **for** $k = r$ to 1 **do** ▷ This is often called the "backward pass"
4: **if** $k = r$ **then**
5: compute $\delta^{[r]} \triangleq \frac{\partial J}{\partial z^{[r]}}$
6: **else**
7: compute

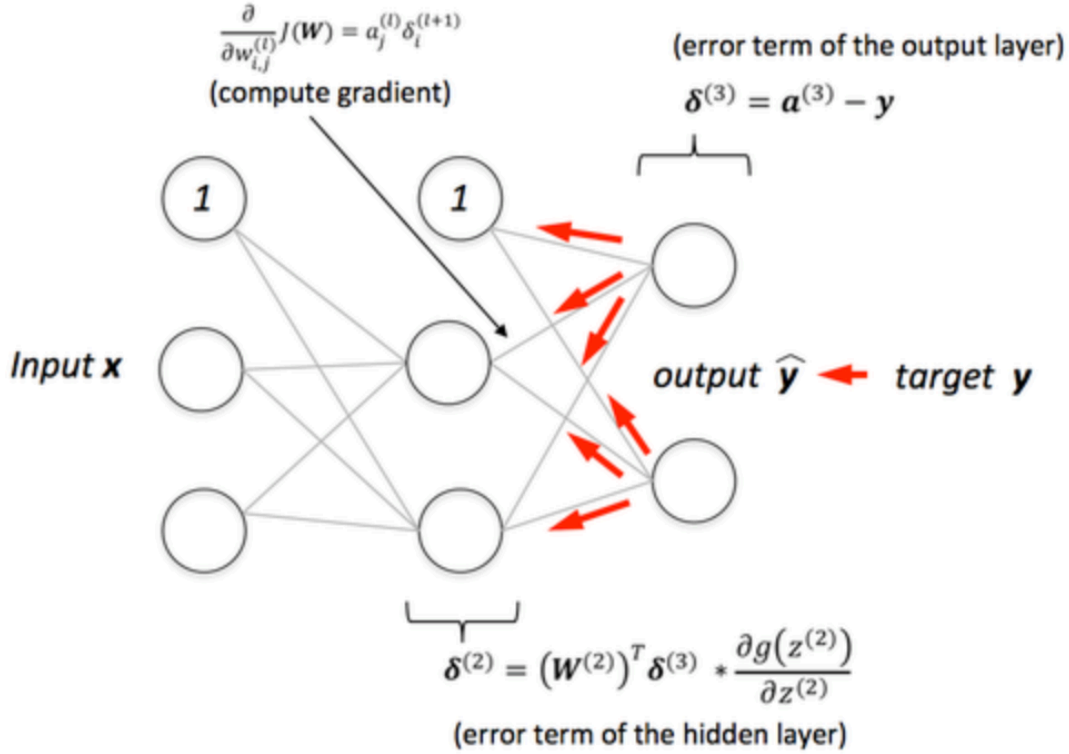
$$\delta^{[k]} \triangleq \frac{\partial J}{\partial z^{[k]}} = \left(W^{[k+1]\top} \delta^{[k+1]}\right) \odot \mathrm{ReLU}'(z^{[k]})$$

8: Compute

$$\frac{\partial J}{\partial W^{[k]}} = \delta^{[k]} a^{[k-1]\top}$$
$$\frac{\partial J}{\partial b^{[k]}} = \delta^{[k]}$$

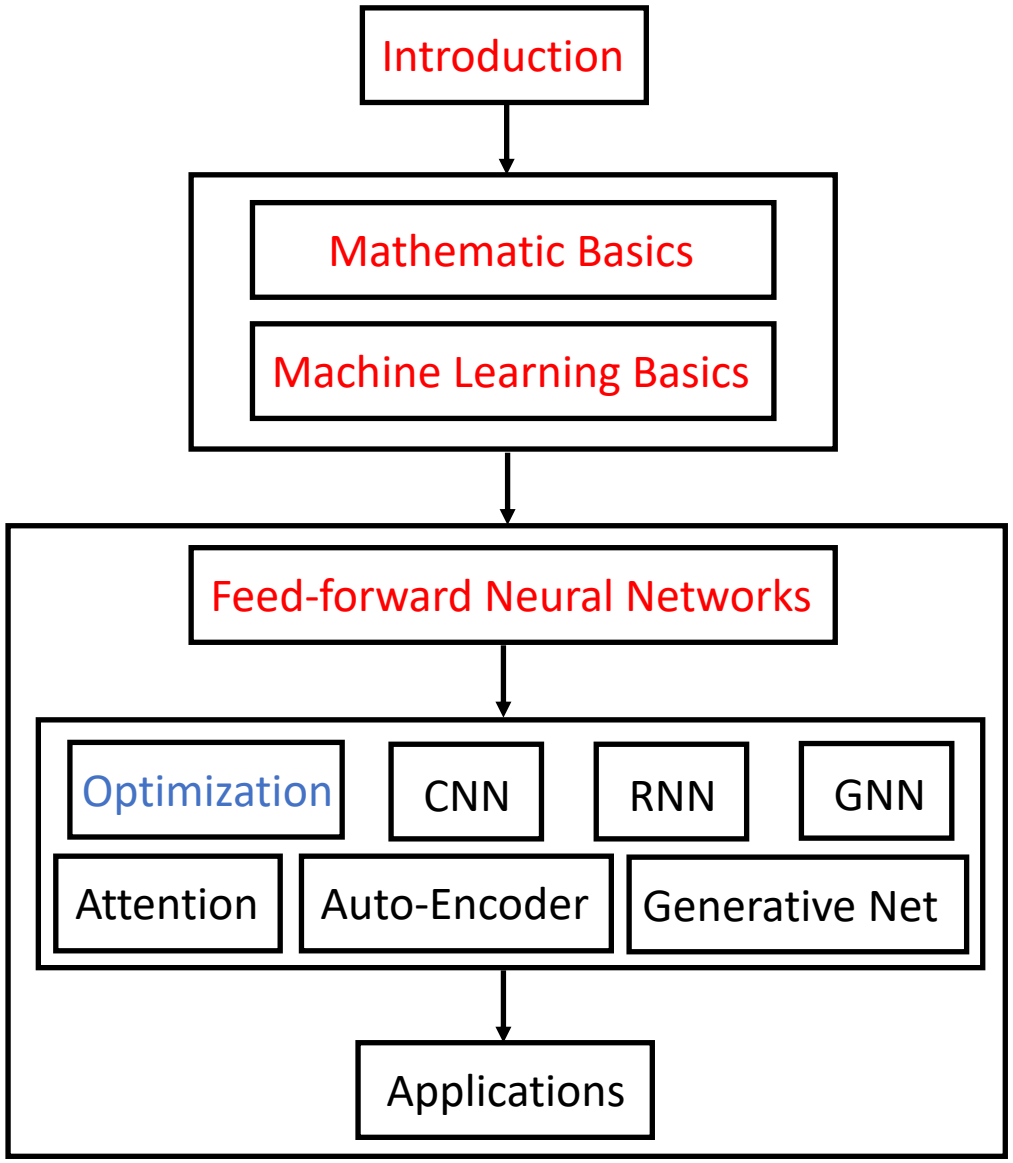
❑ Backpropagation





In backpropagation, we "simply" backpropagate the error to update model parameters.

Structure of This Course



Regularization

❑ Parameter Norm Penalties

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

ridge regression

❑ L2 Regularization

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

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

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

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

Single step: multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update

L2 Regularization

Whole process: analysis by quadratic approximation

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

Hessian matrix

set gradient to zero

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

add weight decay gradient

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$
$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$
$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*.$$

decomposition

$$\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top.$$

lambda

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*$$
$$= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*$$
$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*.$$

The effect of weight decay is to rescale w∗ along the axes defined by the eigenvectors

Along the directions where the eigenvalues of H are relatively large, the effect of regularization is relatively small

Regularization

❑ L1 Regularization

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$ sum of absolute values

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

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

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha|w_i|\right]$$ quadratic approximation

analytical solution $$w_i = \text{sign}(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

two cases $w_i^* \leq \frac{\alpha}{H_{i,i}}$ $w_i^* > \frac{\alpha}{H_{i,i}}$

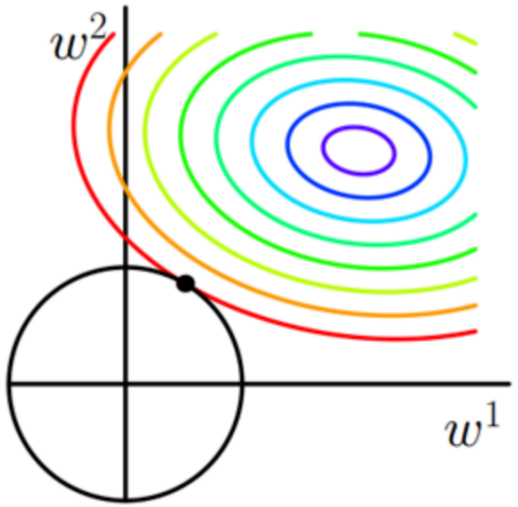
zero shift w by a distance

❑ L2 and L1 Comparison (Regularization as Constraint)

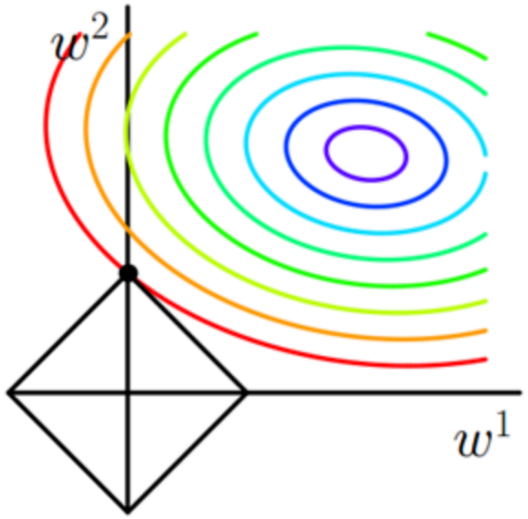
$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

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

constraint optimization



tangent points

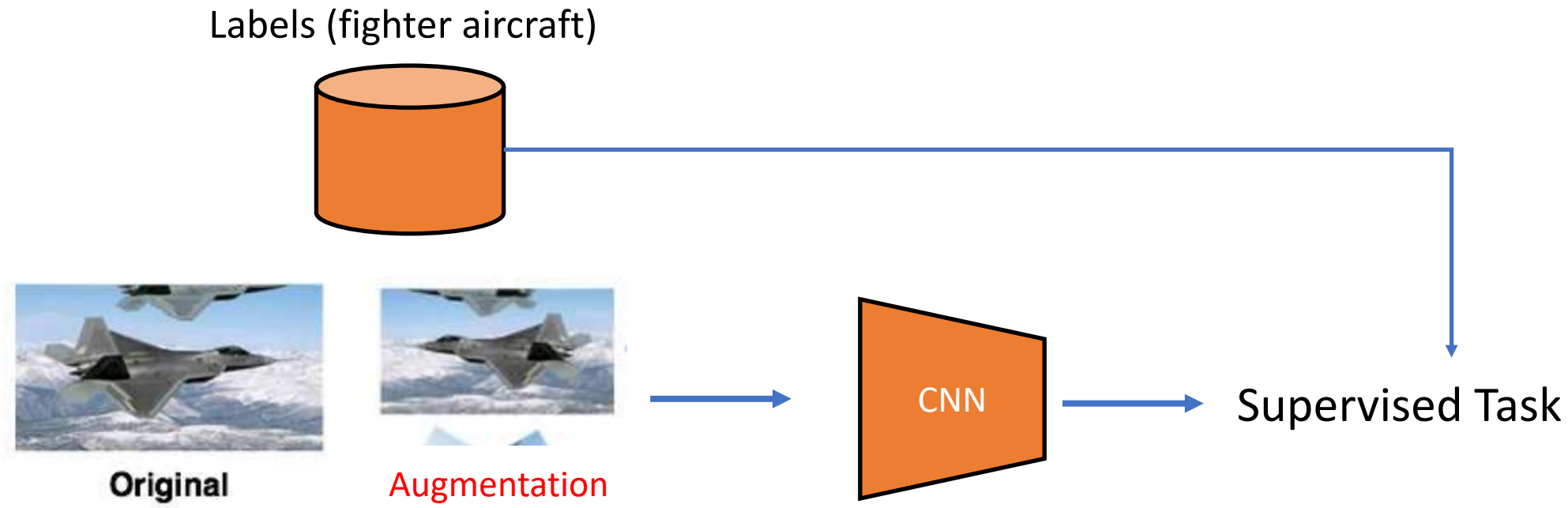


non-sparse weights
(most are non-zero)

sparse weights
(most are zero)

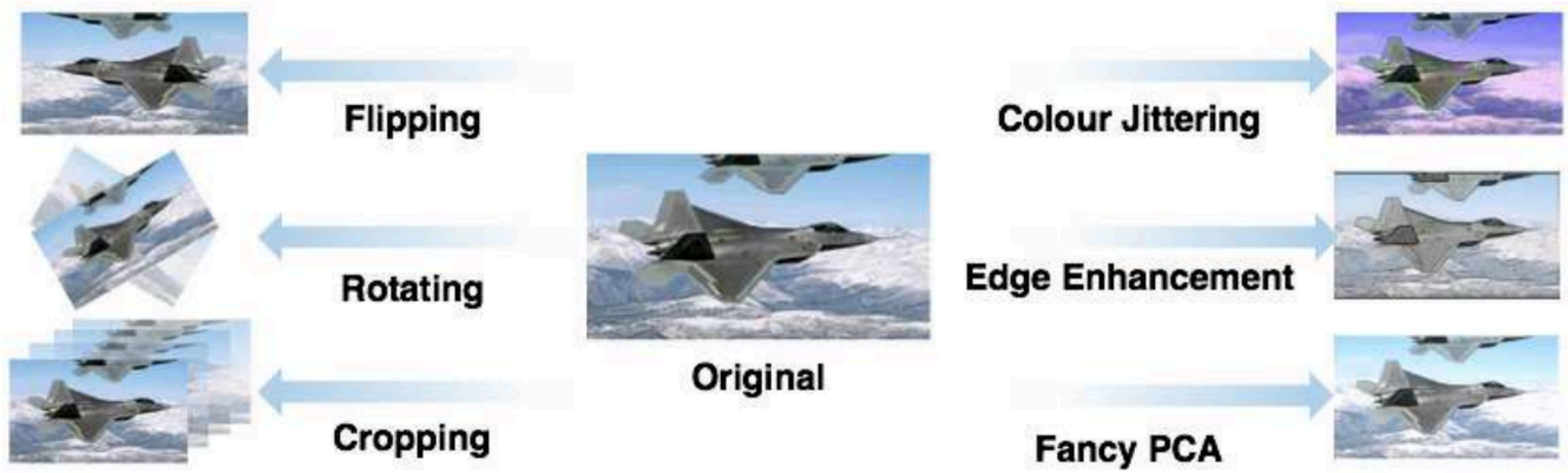
Data Augmentation

- More Data, Better Generalization



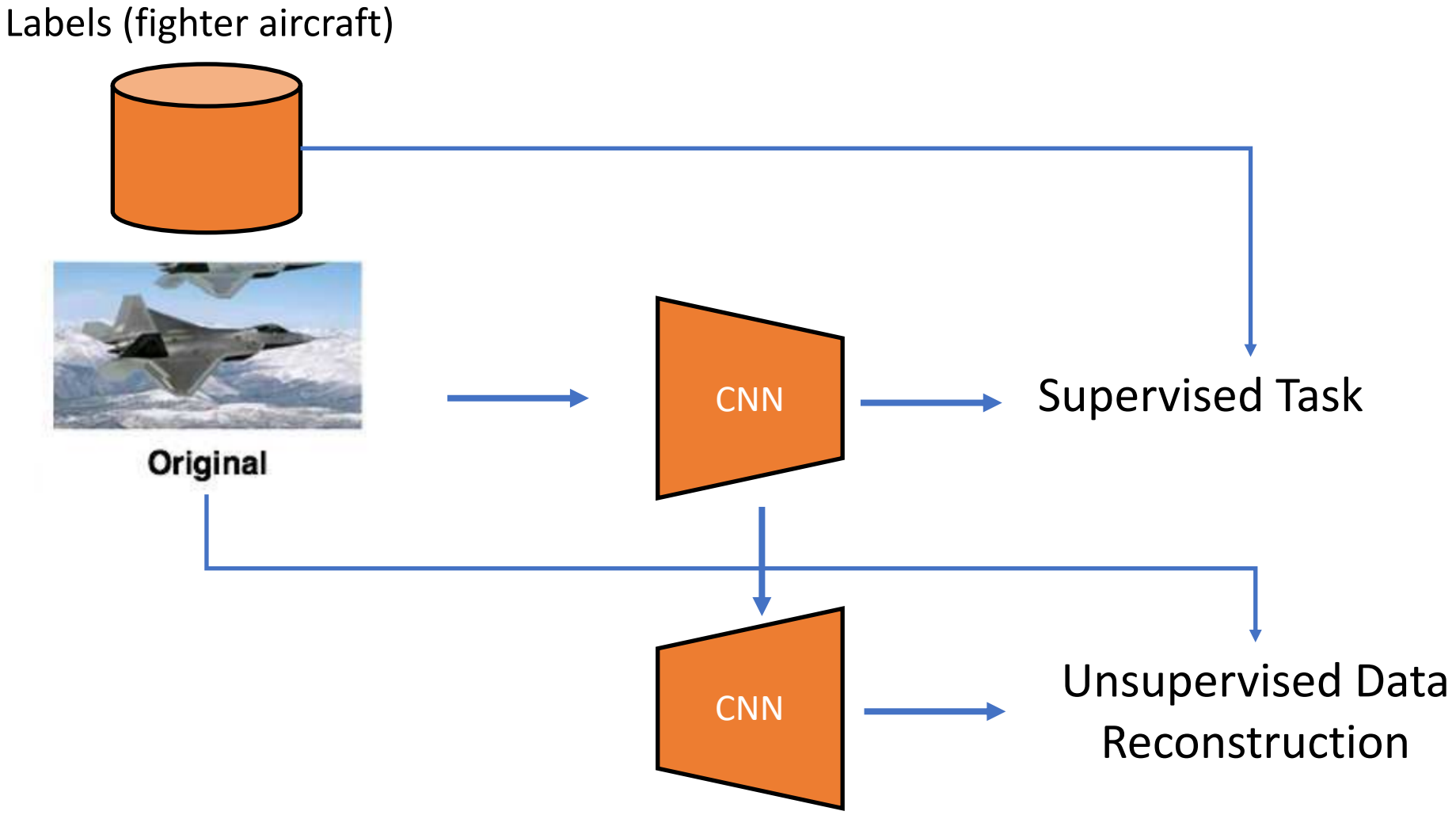
Data Augmentation

- Example (Image)



Semi-Supervised Learning

- Supervised Task + Unsupervised Data Reconstruction

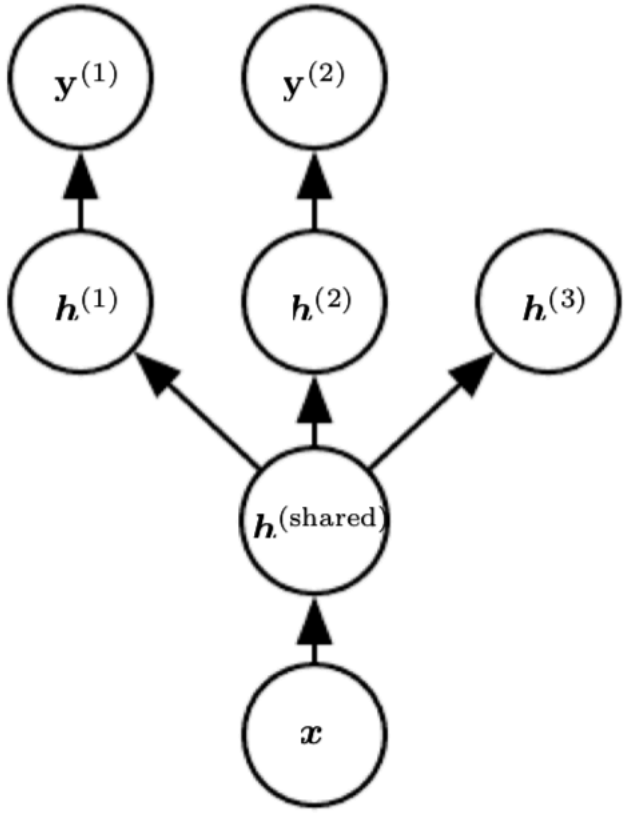


Multi-Task Learning

❑ Multiple Tasks

Different Tasks

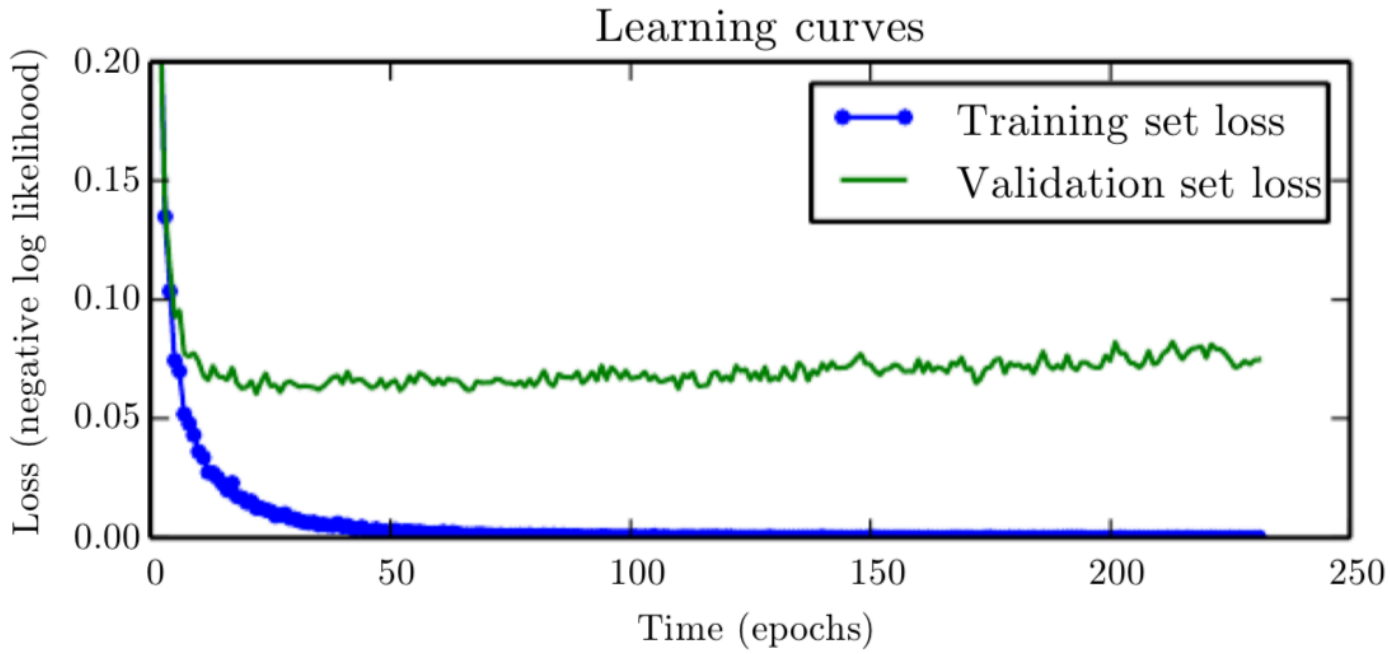
Task-specific parameters



Additional training examples put more pressure on the parameters of the model towards values that generalize well, when part of a model is shared across tasks, that part of the model is more constrained towards good values, often yielding better generalization.

Early Stopping

❑ Learning Curve



Training error decreases steadily over time, but validation set error begins to rise again

Instead of running model until it reaches minimum validation error, run it until the error on validation set has not improved for some amount of time.

Early Stopping

- Algorithm

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Early Stopping

❑ Algorithm

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Algorithm 7.3 Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**
  Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for n steps.
end while

Early Stopping

❑ How It Acts As Regularizer

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

quadratic approximation

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$$\begin{aligned}\boldsymbol{w}^{(\tau)} &= \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(\tau-1)}) \\ &= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)\end{aligned}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

$$\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top. \qquad \boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

$$\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

$$\boldsymbol{Q}^\top \boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau]\boldsymbol{Q}^\top \boldsymbol{w}^*$$

L2 regularization

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*$$

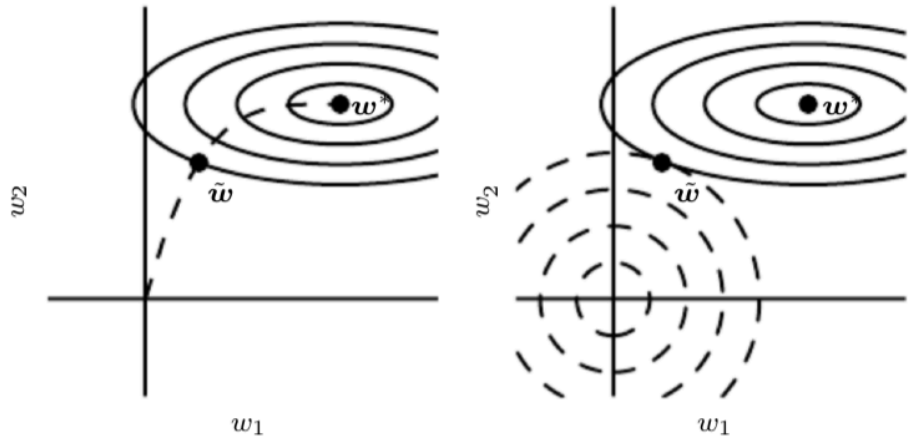
$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^\top \boldsymbol{w}^*$$

$$(\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha.$$

$$\tau \approx \frac{1}{\epsilon \alpha},$$

$$\alpha \approx \frac{1}{\tau \epsilon}.$$

L2 regularization and early stopping can be seen to be equivalent.



Bagging

❑ Ensemble Method

How: train several different models separately, then have all of the models vote on the output for test examples

Expected square error

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

errors drawn from normal distribution

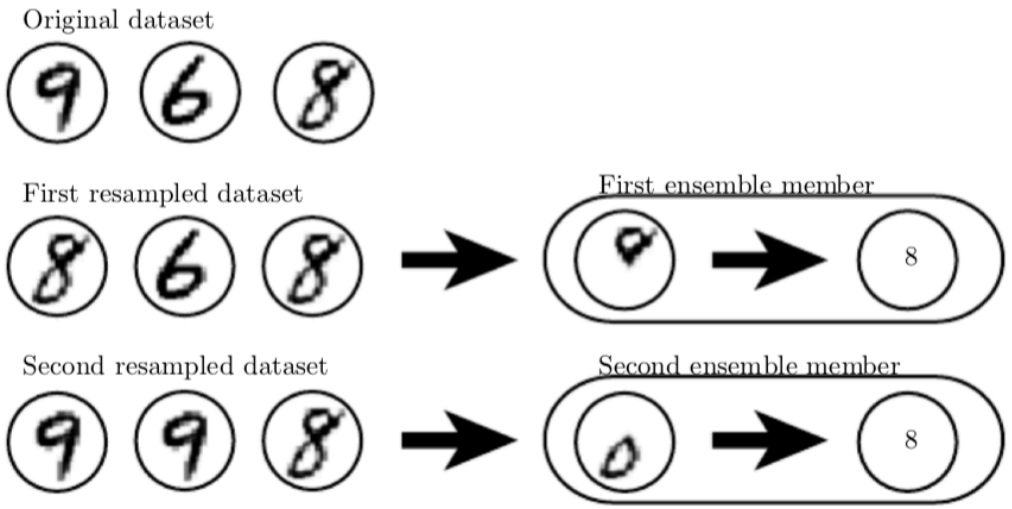
When errors are perfectly correlated and c = v, the mean squared error reduces to v, so the model averaging does not help at all.

When errors are perfectly uncorrelated and c = 0, the expected squared error of the ensemble is only v/k. This means that the expected squared error of the ensemble k decreases linearly with the ensemble size.

Conclusion: if the members make independent errors, the ensemble will perform significantly better than its members.

Bagging

❑ Example



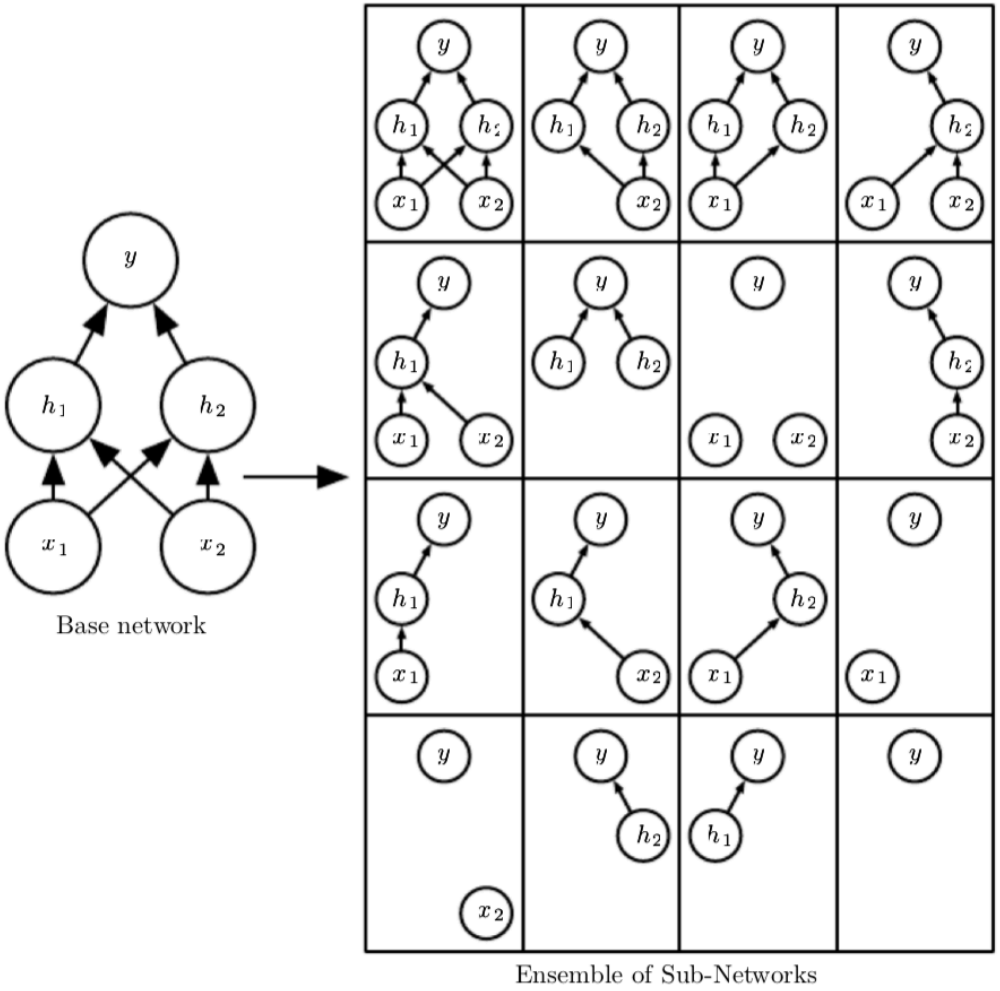
K different datasets. Each one has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the original dataset.

Model i is then trained on dataset i.

Missing some of the examples from the original dataset and also contains several duplicate examples.

Dropout

❑ Method



How: Randomly some portions of weights to zero (e.g., 50%)

Trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.

Dropout

❑ Strengths

Very computationally cheap. requires only O(n) computation per example per update, to generate n random binary numbers and multiply them by the state. It may also require O(n) memory to store these binary numbers until the back-propagation stage.

Does not significantly limit the type of model or training procedure that can be used. It works well with nearly any model that uses a distributed representation and can be trained with stochastic gradient descent.

Can work with other regularization methods.

Require sufficient training data.

Dropout

❑ Bagging vs Dropout

Bagging: k different models, construct k different datasets, train model i on dataset i.
Dropout: approximate this process, but with an exponentially large number of neural networks.

Bagging: the models are all independent, each model is trained to convergence on its respective training set
Dropout: the models share parameters, with each model inheriting a different subset of parameters from the parent neural network

Bagging: accumulate votes from all of its members for prediction
Dropout: each sub-model defined by mask vector u defines a probability distribution

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

exponential number of terms, it is intractable

approximate the inference with sampling, by averaging together the output from many masks.

Q & A