# Artificial Neural Networks
## 人工神经网络

权小军教授
中山大学计算机学院

quanxj3@mail.sysu.edu.cn

2023 年 4 月 10 日

# Lecture 5 - Neural Networks Optimization

(*Part of the slides are adapted from CMU 11-785*)

# Recap

# Story so far

- Neural networks are universal approximators (万能逼近器)
  - Can model any odd thing
  - Provided they have the right architecture
- We must train them to approximate any function
  - Specify the architecture
  - Learn their weights and biases
- Networks are trained to minimize total "loss" on training set
  - We do so by empirical risk minimization (经验风险最小化)
- We use gradient descent to do so
- The gradient of the error with respect to network parameters is computed through backpropagation (反向传播)

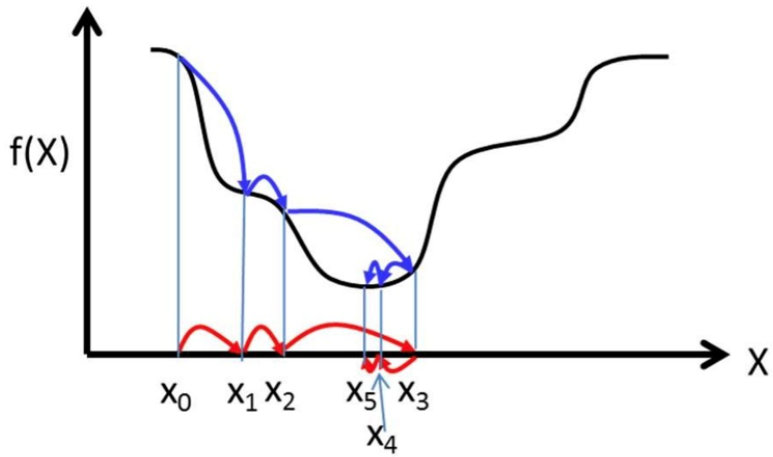
# Problem Setup

- Given a training set of input-output pairs $(X_1, d_1)$, $(X_2, d_d)$, ..., $(X_T, d_T)$
- Minimize the following function

$$Loss(W) = \frac{1}{T}\sum_i div(g(X_i; \mathbf{W}), d_i)$$

# Gradient Descent Algorithm

- In order to minimize any function $f(x)$ w.r.t. $x$
- Initialize:
  - $x^0$
  - $k = 0$
- Do
  - $k = k + 1$
  - $x^{k+1} = x^k - \eta \nabla_x f^T$
- while $\left| f\left(x^k\right) - f\left(x^{k-1}\right) \right| < \varepsilon$



# Stochastic gradient descent

- Update model parameters using each single data $(\mathbf{x}_n, \mathbf{y}_n)$

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t - \eta \nabla l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t)) \end{aligned}$$

# Stochastic gradient descent

- Update model parameters using each single data $(\mathbf{x}_n, \mathbf{y}_n)$

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t - \eta \nabla l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t)) \end{aligned}$$

- Very fast update, tractable for big data
- May jump to better local minimum
- Converge almost certainly to local minimum
- Convergence fluctuates

Above example used *single* data to update model parameters.

How to update parameters if we have lots of (training) data?

- Training dataset $D = \{(\mathbf{x}_n, \mathbf{y}_n) | n = 1, \ldots, N\}$
- Loss function

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}))$$

# Batch gradient descent

- Update model parameters using all data with gradient descent

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t - \frac{\eta}{N} \sum_{n=1}^{N} \nabla l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t)) \end{aligned}$$

where $\eta$ is learning rate, $\nabla L(\boldsymbol{\theta}_t)$ is gradient of loss function L over current model parameters $\boldsymbol{\theta}_t$.

# Mini-batch gradient descent

- Update model parameters using a mini-batch of data $S \subset D$

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t - \frac{\eta}{|S|} \sum_{(\mathbf{x}_n, \mathbf{y}_n) \in S} \nabla l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta}_t)) \end{aligned}$$

- In general $|S|$ ranges between tens and two/three hundreds
- Take the best of batch and stochastic gradient descents

# Issues

- Convergence (收敛): How well does it learn
  - And how can we improve it
- How well will it generalize (泛化) (outside training data)
- What does the output really mean?
- Etc..

# Lecture 5.1 Problems with Backprop

# Does backprop do the right thing?

Question: **Does gradient descent find the right solution, even when it finds the actual minimum?**

# Does backprop do the right thing?

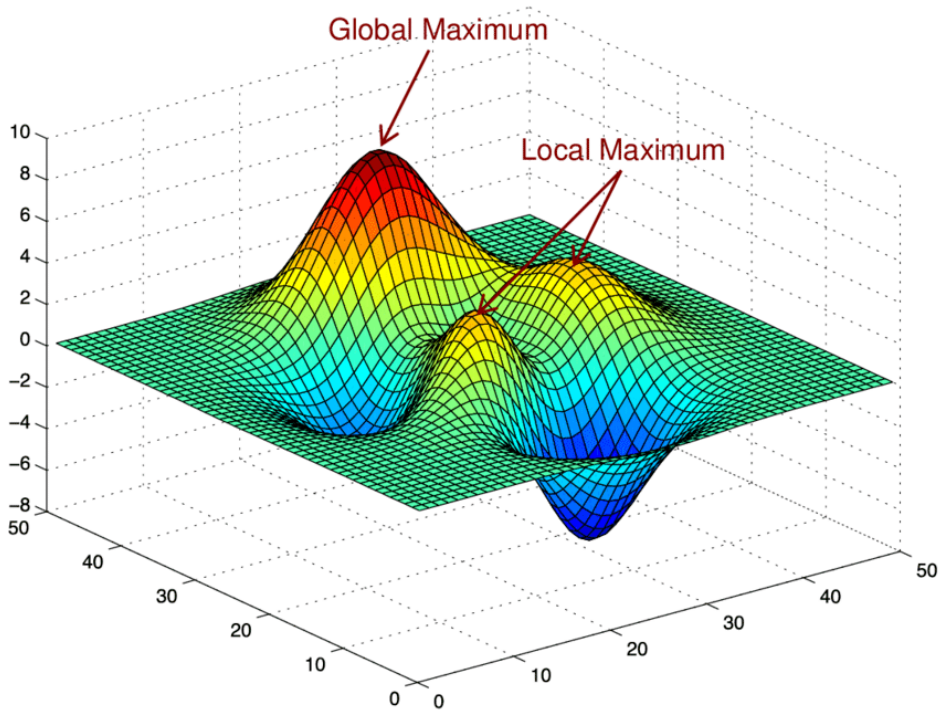
- **Does gradient descent find the right solution, even when it finds the actual minimum?**
- In classification problems, the classification error is a non-differentiable function of weights
- The divergence function minimized is only a proxy for classification error
- Minimizing divergence may not minimize classification error

# The Loss Surface

- The example (and statements) earlier assumed the loss objective had a single global optimum that could be found
- What about local optima (局部最优)

# The Loss Surface

- The example (and statements) earlier assumed the loss objective had a single global optimum that could be found
- What about local optima (局部最优)

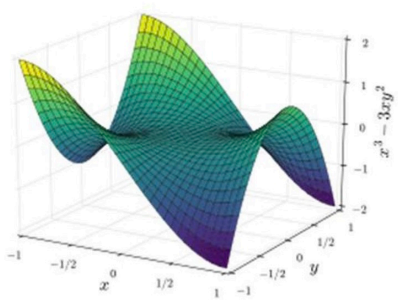


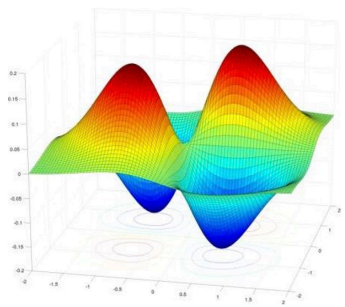
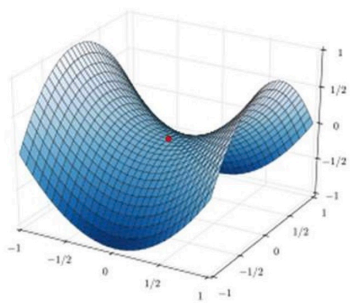
# The Loss Surface

- Saddle point (鞍点/拐点): A point where
  - The slope is zero
  - Surface increases in some directions, but decreases in others
  - Gradient descent algorithms often get "stuck" in saddle points

# The Loss Surface

- Saddle point (鞍点/拐点): A point where
  - The slope is zero
  - Surface increases in some directions, but decreases in others
  - Gradient descent algorithms often get "stuck" in saddle points
- Popular hypothesis:
  - In large networks, saddle points are far more common than local minima (在大网络中，鞍点的数量多过局部最后)
  - Most local minima are equivalent
  - This is not true for small networks



# Story so far

- Neural nets can be trained via gradient descent that minimizes a loss function
- Backpropagation can be used to derive the derivatives of loss
- Backprop is not guaranteed to find a "true" solution, even if it exists, and lies within the capacity of the network to model
  - The optimum for loss function may not be the "true" solution
- For large networks, the loss function may have a large number of unpleasant saddle points or local minima

# Lecture 5.2 Convergence

# Convergence

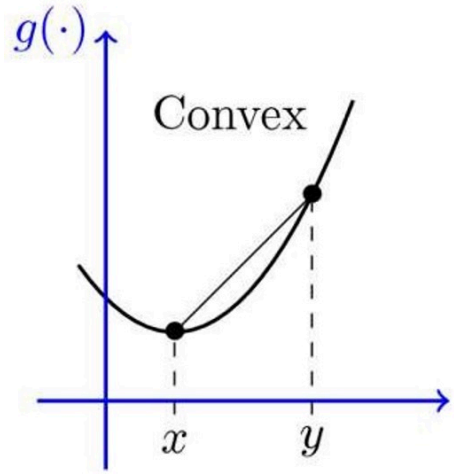
- In the discussion so far we have assumed the training arrives at a local minimum

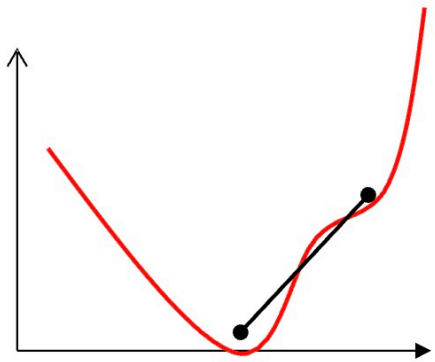
# Convergence

- In the discussion so far we have assumed the training arrives at a local minimum
- Does it always converge?
- How long does it take?
- Hard to analyze for an MLP, but we can look at the problem through the lens of convex optimization

# Convex Loss Functions

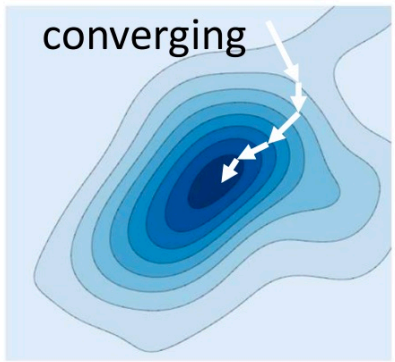
- A surface is "convex" if it is continuously curving upward
  - We can connect any two points on or above the surface without intersecting it
- Caveat: Neural network loss surface is generally not convex

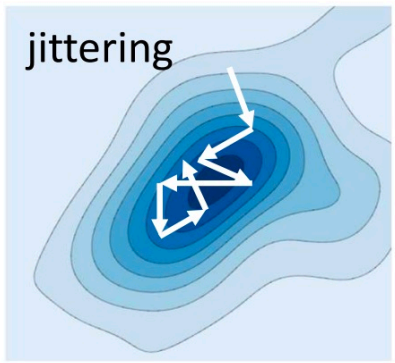


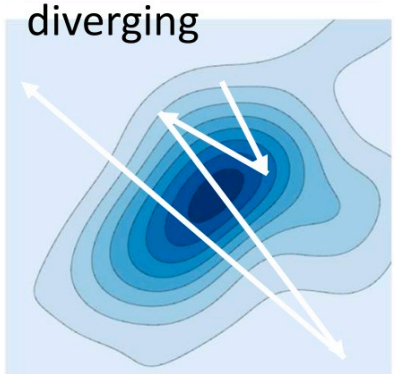



# Convergence of gradient descent

- An iterative algorithm is said to converge to a solution if the value updates arrive at a fixed point
  - Where the gradient is 0 and further updates do not change the estimate
- The algorithm may not actually converge
  - May jitter (抖动) around local minimum
  - It may even diverge (偏离)
- Conditions for convergence?



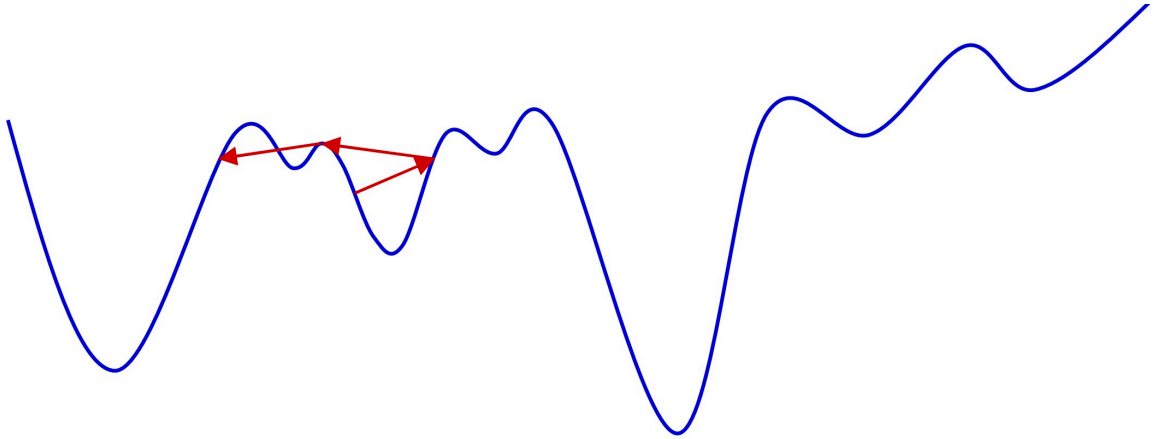
# Convergence and convergence rate (收斂率)

- Convergence rate: How fast the iterations reach the solution
- Generally quantified as

$$R = \frac{|f(x^{(k+1)}) - f(x^*)|}{|f(x^{(k)}) - f(x^*)|}$$

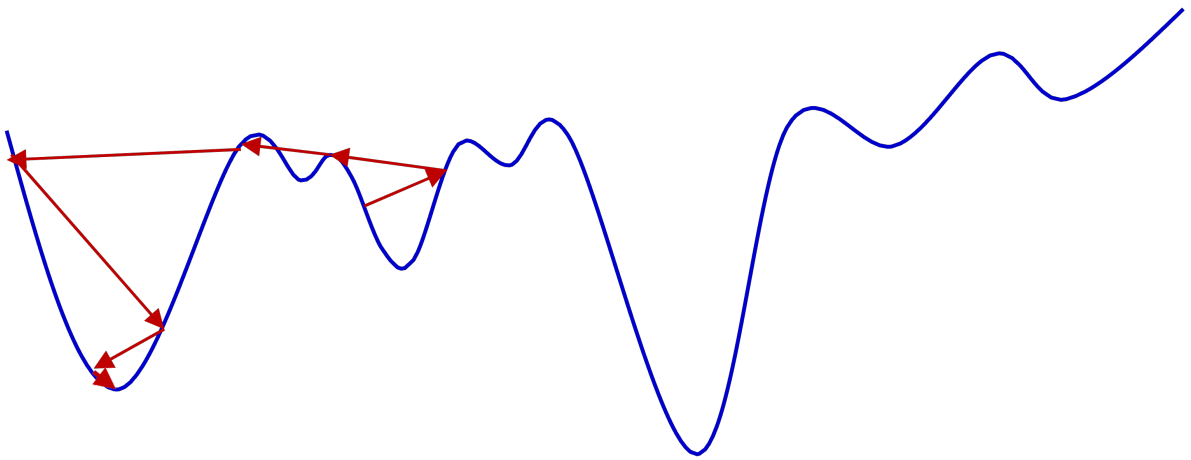
  - $x^{(k+1)}$ is the $k$-th iteration
  - $x^*$ is the optimal value of $x$
- If $R$ is a constant, the convergence is *linear*
  - In reality, its arriving at the solution exponentially fast

# The learning rate (学习率)



- For complex models such as neural networks, the loss function is often not convex
- Having a large learning rate can actually help escape local optima

# Decaying learning rate 衰减学习率



- Start with a large learning rate
- Gradually reduce it with iterations

# Decaying learning rate

- Typical decay schedules
  - Linear decay: $\eta_k = \frac{\eta_0}{k+1}$
  - Quadratic decay: $\eta_k = \frac{\eta_0}{(k+1)^2}$
  - Exponential decay: $\eta_k = \eta_0 e^{-\beta k}$, where $\beta > 0$

# Decaying learning rate

- Typical decay schedules
  - Linear decay: $\eta_k = \frac{\eta_0}{k+1}$
  - Quadratic decay: $\eta_k = \frac{\eta_0}{(k+1)^2}$
  - Exponential decay: $\eta_k = \eta_0 e^{-\beta k}$, where $\beta > 0$
- A common approach (for neural networks):
  - Train with a fixed learning rate until loss (or performance on a held-out data set) stagnates (停滞不前)
  - $\eta \longleftarrow \alpha\eta$, where $\alpha < 1$ (typically 0.1)
  - Return to step 1 and continue training from where we left off

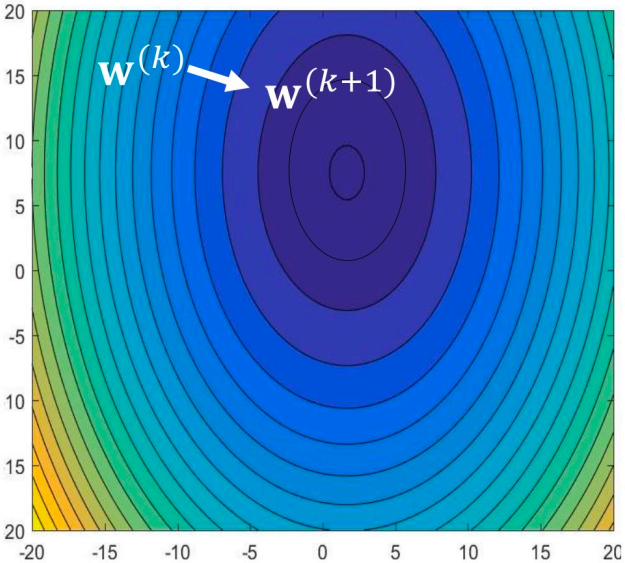
# Story so far : Convergence

- Gradient descent can miss obvious answers
  - And this may be a good thing
- May convergence issues
  - The loss surface has many saddle points
    - Gradient descent can stagnate on saddle points
  - Vanilla gradient descent may not converge, or may converge toooooo slowly
    - The optimal learning rate for one component may be too high or too low for others

# Story so far : Learning rate

- Divergence-causing learning rates may not be a bad thing
  - Particularly for ugly loss functions
- Decaying learning rates provide good compromise (折中) between escaping poor local minima and convergence
- **Many of the convergence issues arise because we force the same learning rate on all parameters**

# Lets take a step back




$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta(\nabla_{\mathbf{w}} E)^T$$

$$w_i^{(k+1)} = w_i^{(k)} - \eta \frac{dE\left(w_i^{(k)}\right)}{d\mathrm{w}}$$

- Problems arise because of requiring a fixed step size across all dimensions
  - Because step are "tied" to the gradient
- Let's try releasing this requirement

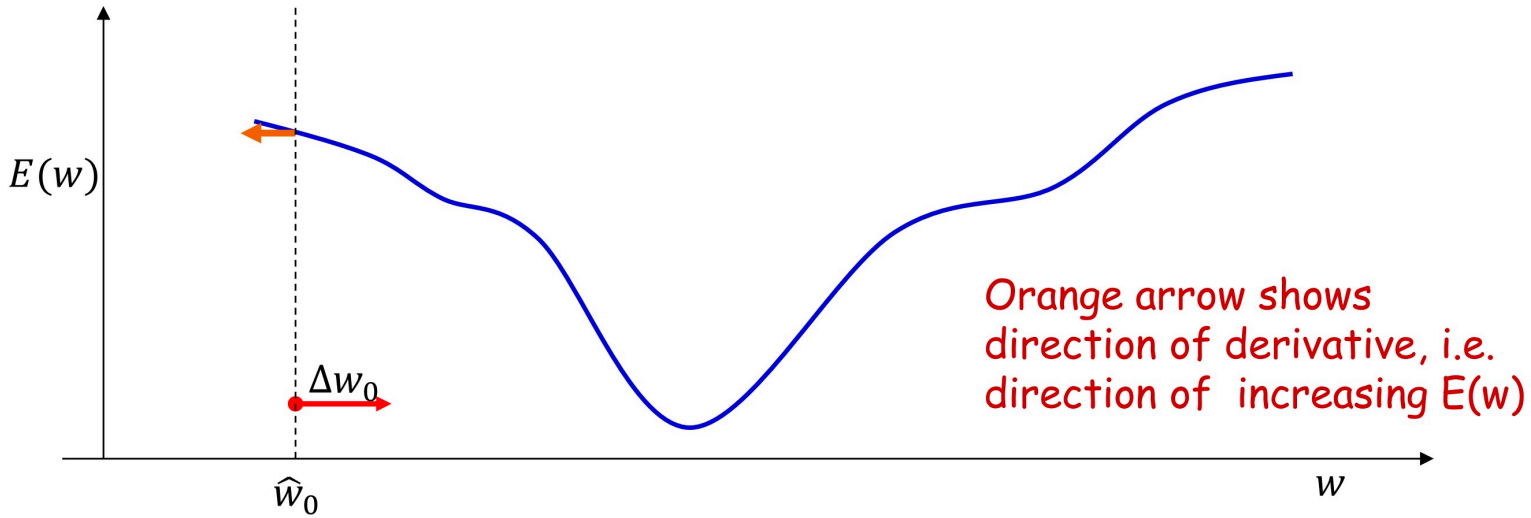
# Derivative-inspired algorithm

- Algorithms that use derivative information for trends, but do not follow them absolutely
  - Rprop

# Rprop

- Resilient propagation (弹性传播)
- Simple algorithm, to be followed independently for each component
  - I.e. steps in different directions are not coupled
- At each time
  - If the derivative at the current location recommends continuing in the same direction as before (i.e. has not changed sign from earlier):
    - increase the step, and continue in the same direction
  - If the derivative has changed sign (i.e. overshot a minimum)
    - reduce the step and reverse direction

# Rprop

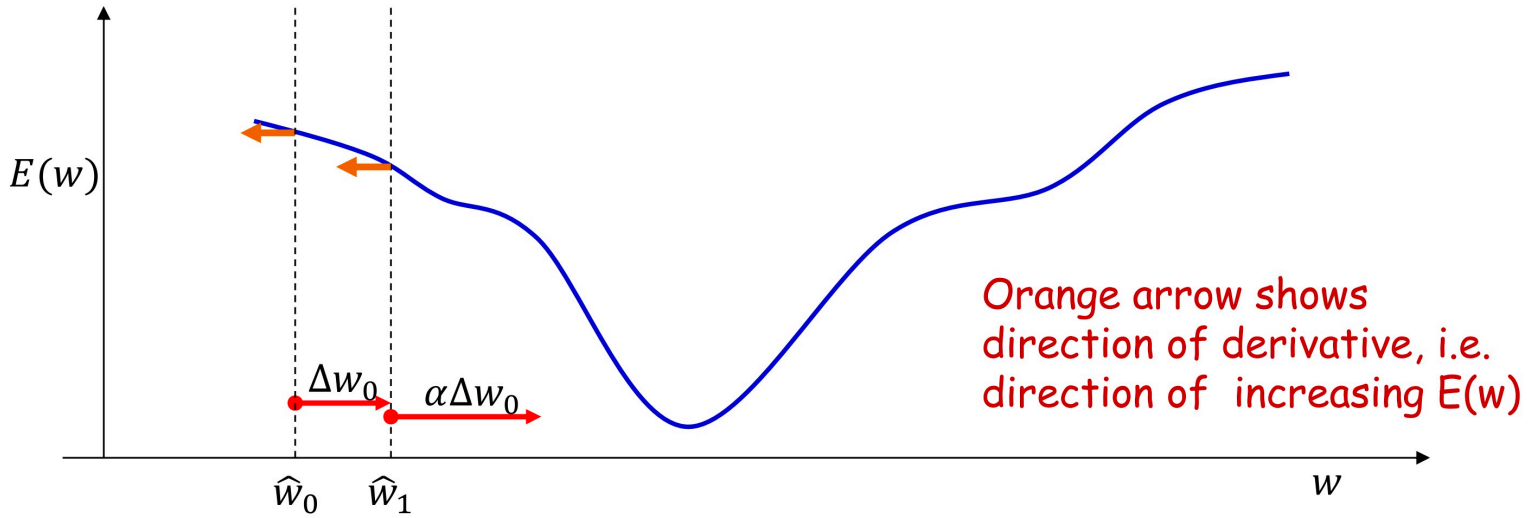



- Select an initial value $\hat{w}$ and compute the derivative
  - Take an initial step $\Delta w$ against the derivative
    - In the direction that reduces the function

$$\Delta w = sign\left(\frac{dE(\hat{w})}{dw}\right)\Delta w$$

$$\hat{w} = \hat{w} - \Delta w$$

# Rprop

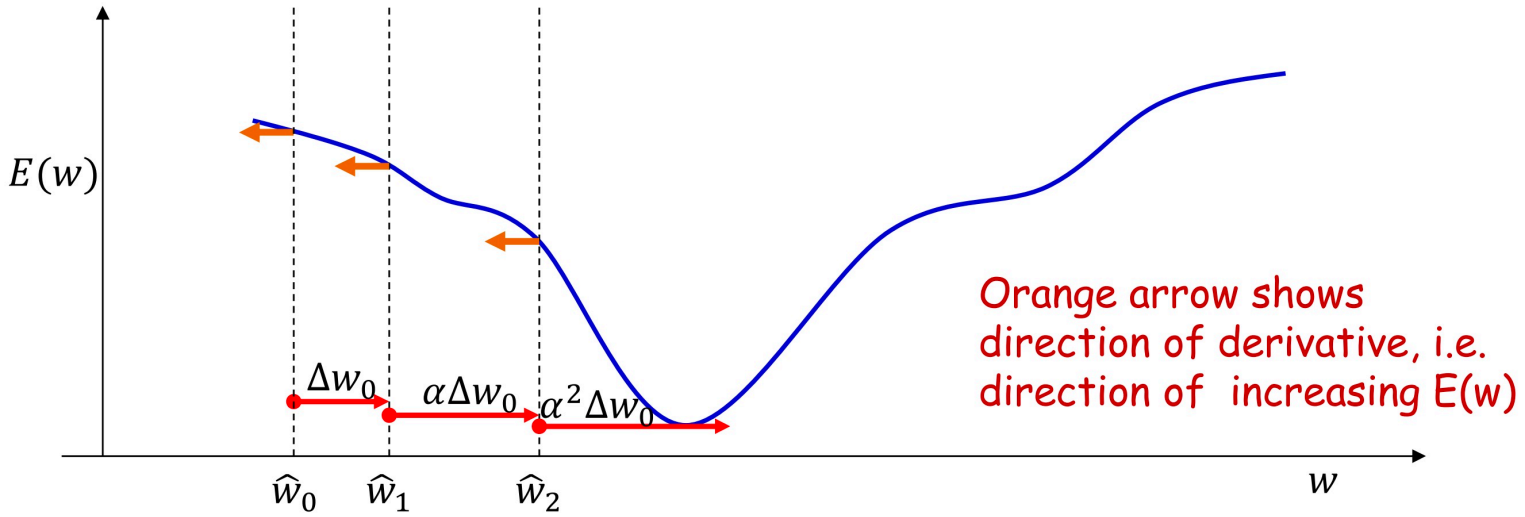



- Compute the derivative in the new location
  - If the derivative has not changed sign from the previous location, increase the step size and take a longer step

$\alpha > 1$

- $\Delta w = \alpha \Delta w$
- $\hat{w} = \hat{w} - \Delta w$

# Rprop

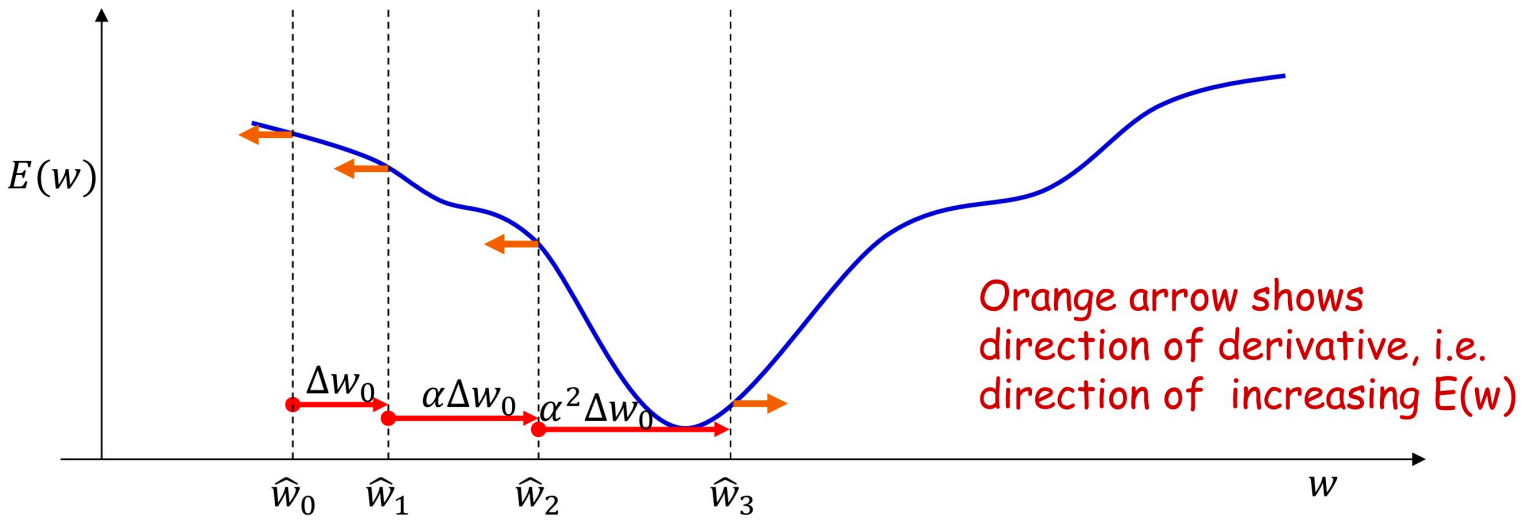



- Compute the derivative in the new location
  - If the derivative has not changed sign from the previous location, increase the step size and take a step

$\alpha > 1$

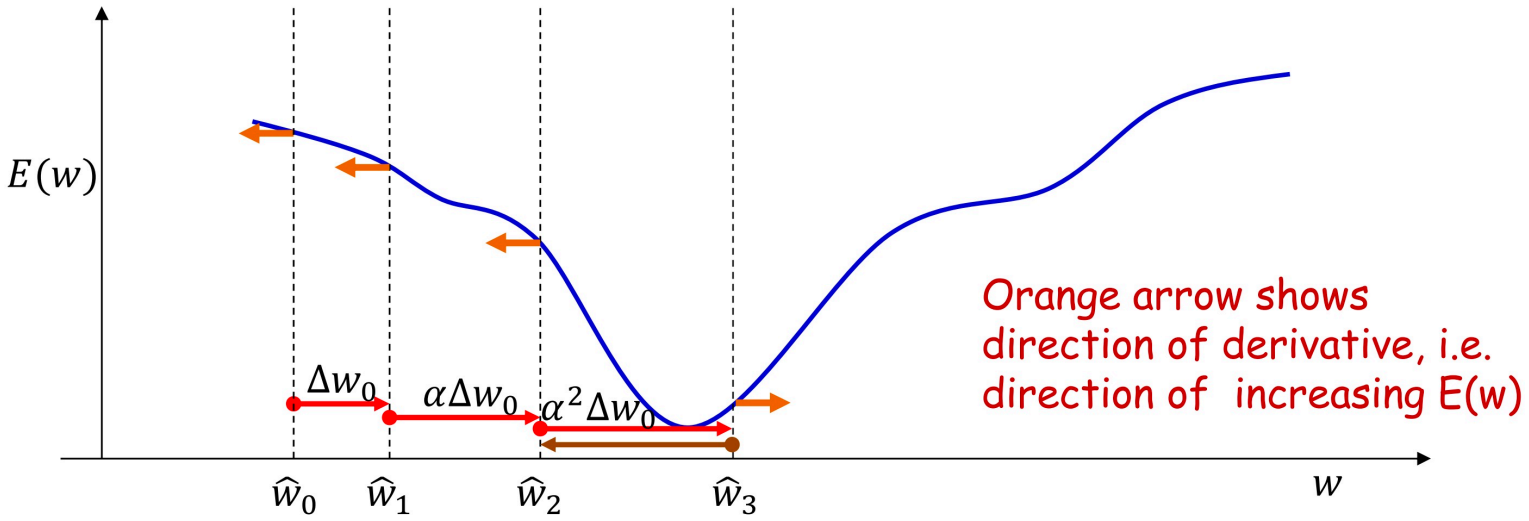
- $\Delta w = \alpha\Delta w$
- $\widehat{w} = \widehat{w} - \Delta w$

# Rprop



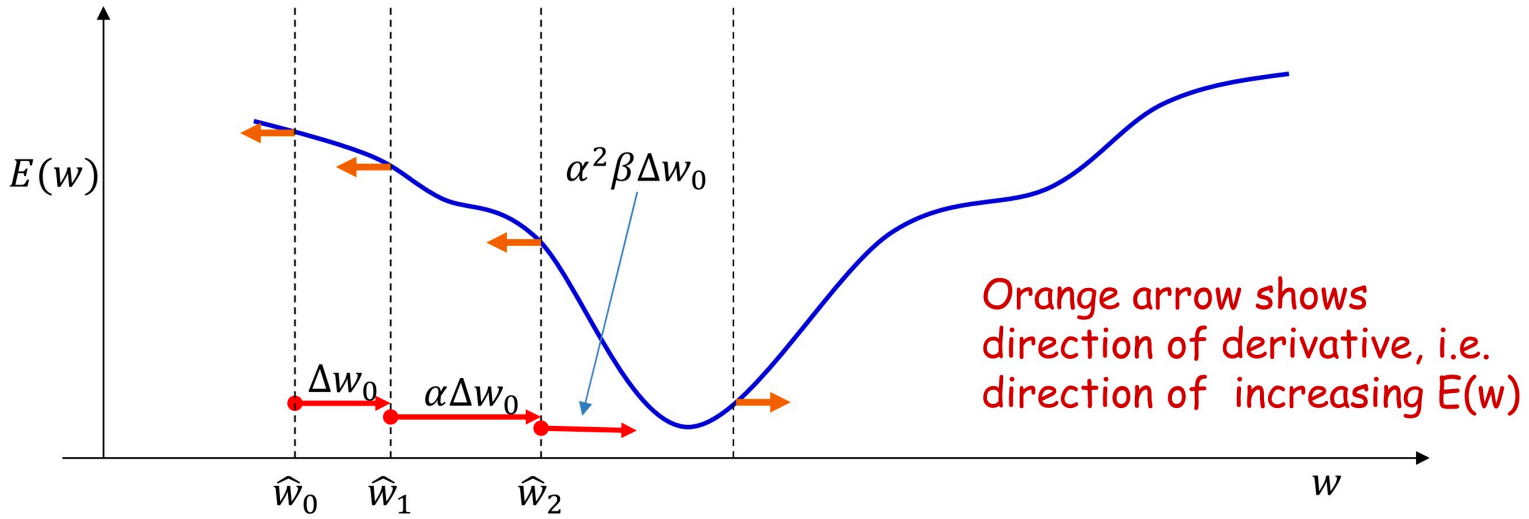

- Compute the derivative in the new location
  - If the derivative has changed sign

# Rprop




- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\widehat{w} = \widehat{w} + \Delta w$

# Rprop

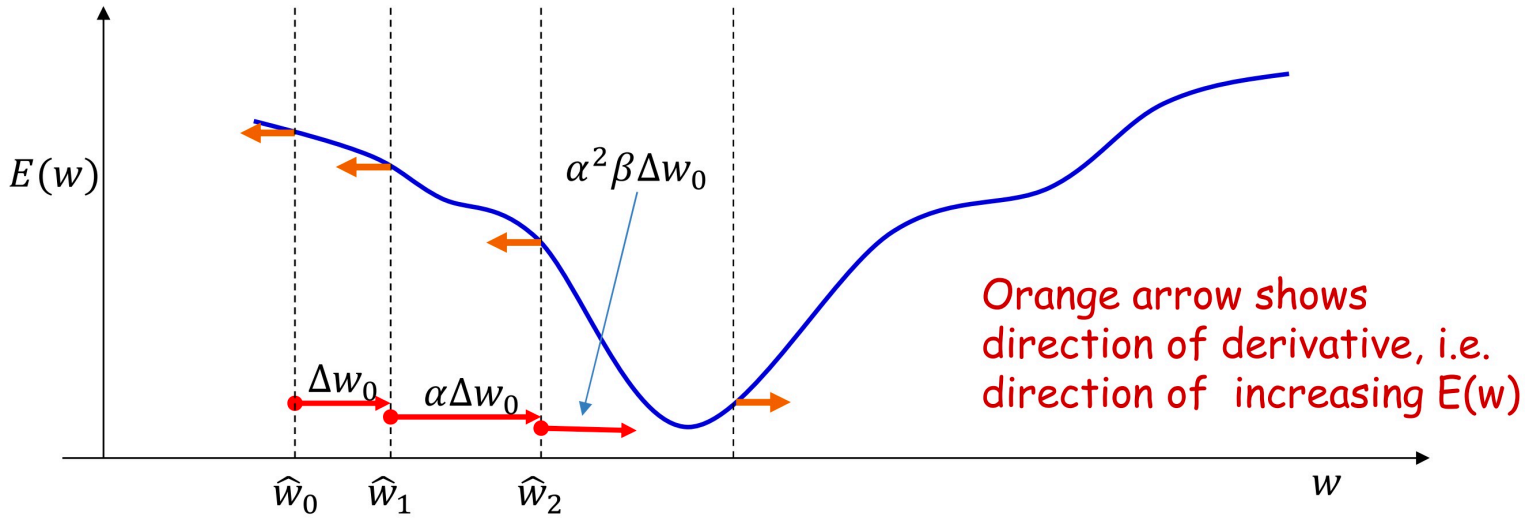



- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\widehat{w} = \widehat{w} + \Delta w$
  - Shrink the step

$\beta < 1$

- $\Delta w = \beta \Delta w$

# Rprop




- Compute the derivative in the new location
  - If the derivative has changed sign
  - Return to the previous location
    - $\widehat{w} = \widehat{w} + \Delta w$
  - Shrink the step

    $\beta < 1$
    - $\Delta w = \beta \Delta w$
  - Take the smaller step forward
    - $\widehat{w} = \widehat{w} - \Delta w$

# Rprop (simplified)

- Set $\alpha = 1.2, \beta = 0.5$
- For each layer $l$, for each $i, j$:
  - Initialize $w_{l,i,j}, \Delta w_{l,i,j} > 0$,
  - $prevD(l,i,j) = \frac{dLoss(w_{l,i,j})}{dw_{l,i,j}}$
  - $\Delta w_{l,i,j} = \text{sign}\big(prevD(l,i,j)\big)\Delta w_{l,i,j}$
  - While not converged:
    - $w_{l,i,j} = w_{l,i,j} - \Delta w_{l,i,j}$
    - $D(l,i,j) = \frac{dLoss(w_{l,i,j})}{dw_{l,i,j}}$
    - If $\text{sign}\big(prevD(l,i,j)\big) == \text{sign}\big(D(l,i,j)\big)$:
      - $\Delta w_{l,i,j} = \min(\alpha \Delta w_{l,i,j}, \Delta_{max})$
      - $prevD(l,i,j) = D(l,i,j)$
    - else:
      - $w_{l,i,j} = w_{l,i,j} + \Delta w_{l,i,j}$
      - $\Delta w_{l,i,j} = \max(\beta \Delta w_{l,i,j}, \Delta_{min})$

Ceiling and floor on step

# Rprop (simplified)

- Set $\alpha = 1.2, \beta = 0.5$
- For each layer $l$, for each $i, j$:
  - Initialize $w_{l,i,j}$, $\Delta w_{l,i,j} > 0$,
  - $prevD(l,i,j) = \frac{dLoss(w_{l,i,j})}{dw_{l,i,j}}$
  - $\Delta w_{l,i,j} = \text{sign}\big(prevD(l,i,j)\big)\Delta w_{l,i,j}$
  - While not converged:
    - $w_{l,i,j} = w_{l,i,j} - \Delta w_{l,i,j}$
    - $D(l,i,j) = \frac{dLos\ (w_{l,i,j})}{dw_{l,i,j}}$
    - If $\text{sign}\big(prevD(l,i,j)\big) == \text{sign}\big(D(l,i,j)\big)$:
      - $\Delta w_{l,i,j} = \alpha \Delta w_{l,i,j}$
      - $prevD(l,i,j) = D(l,i,j)$
    - else:
      - $w_{l,i,j} = w_{l,i,j} + \Delta w_{l,i,j}$
      - $\Delta w_{l,i,j} = \beta \Delta w_{l,i,j}$

Obtained via backprop

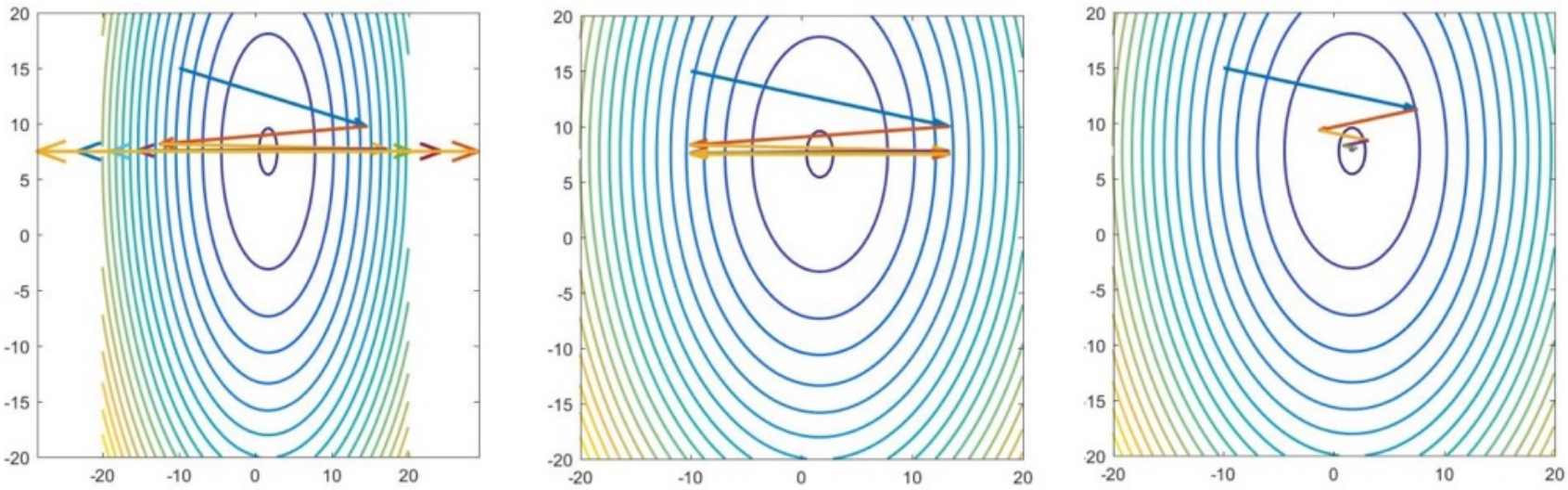
Note: Different parameters updated independently

# Rprop

- A remarkably simple algorithm that is frequently much more efficient than gradient descent
- Only makes minimal assumptions about the loss function
  - No convexity assumption

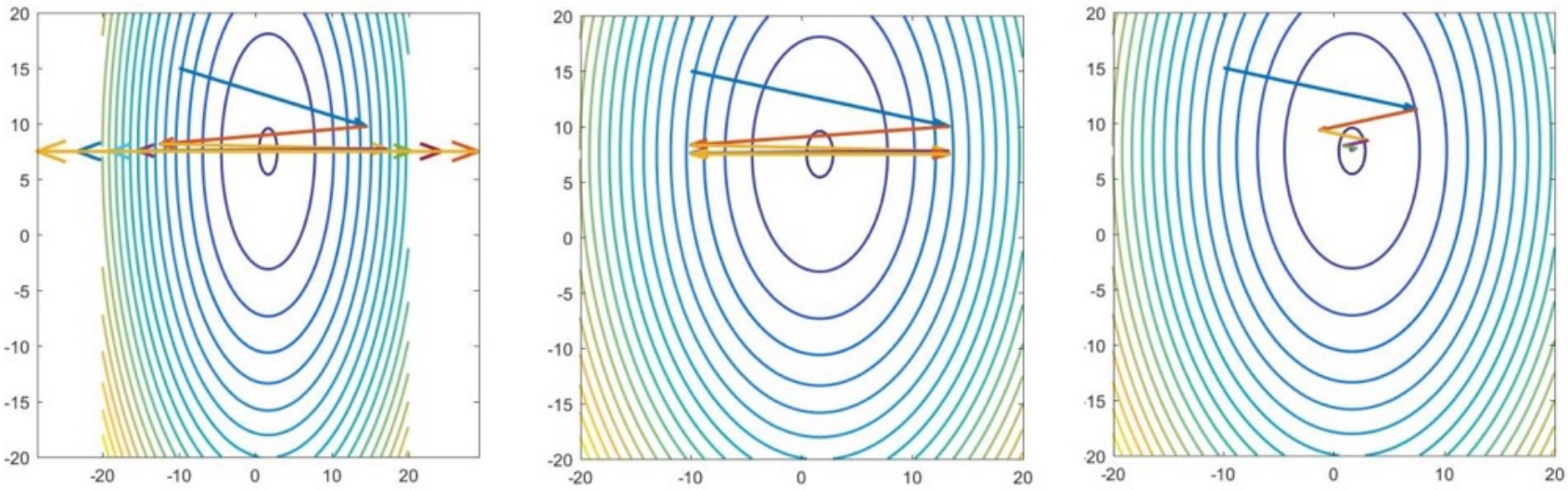
# Lecture 5.3 Momentum Methods 动量方法

# A closer look at the convergence problem




- With dimension-independent learning rates, the solution will converge smoothly in some directions, but oscillate (振荡) or diverge (背离) in others

# A closer look at the convergence problem




- With dimension-independent learning rates, the solution will converge smoothly in some directions, but oscillate (振荡) or diverge (背离) in others

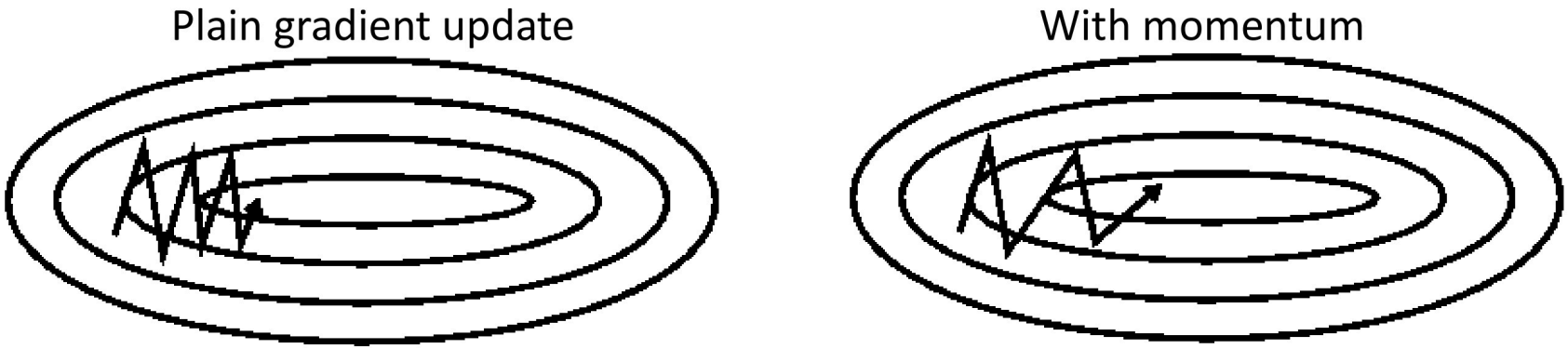
**Proposal:**

- Keep track of oscillations
- Emphasize steps in directions that converge smoothly
- Shrink steps in directions that bounce around..

# The momentum (动量) methods

- Maintain a running average (滑动平均) of all past steps
  - In directions in which the convergence is smooth, the average will have a large value
  - In directions in which the estimate swings, the positive and negative swings will cancel out in the average
- Update with the running average, rather than the current gradient

# Momentum Update



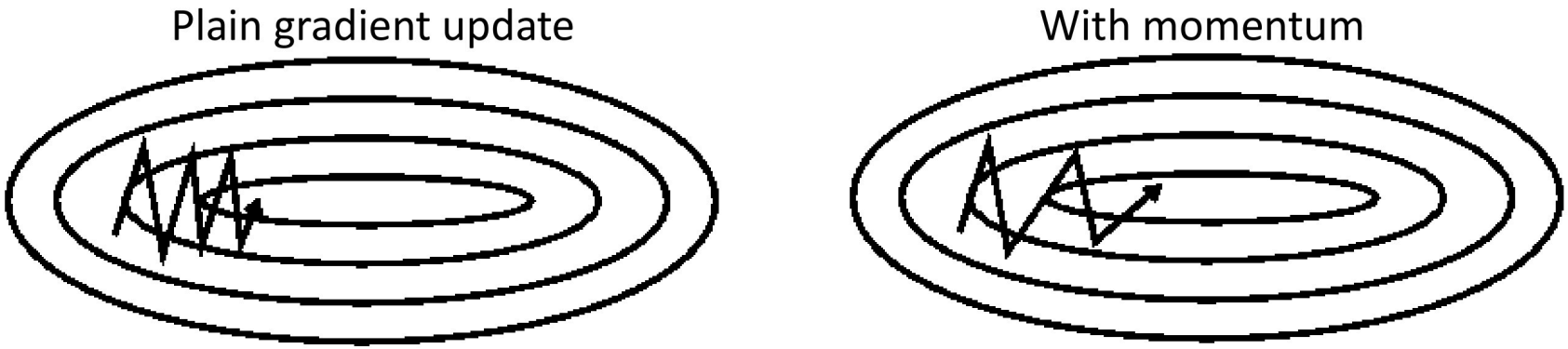

- The momentum method maintains a running average of all gradients until the current step

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss\left(W^{(k-1)}\right)^{\mathsf{T}}$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

  - Typical $\beta$ value is 0.9

# Momentum Update




- The momentum method maintains a running average of all gradients until the current step

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss\left(W^{(k-1)}\right)^{\mathsf{T}}$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

  - Typical $\beta$ value is 0.9
- The running average steps
  - Get longer in directions where gradient retains the same sign
  - Become shorter in directions where the sign flips (翻转)

# Training by gradient descent

- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_K$
- Do:
  - For all $i, j, k,$ initialize $\nabla_{W_k} Loss = 0$
  - For all $t = 1:T$
    - For every weight:
      - Compute $\nabla_{W_k} \boldsymbol{Div}(Y_t, d_t)$
      - Compute $\nabla_{W_k} Loss \mathrel{+}= \frac{1}{T} \nabla_{W_k} \boldsymbol{Div}(Y_t, d_t)$
  - For every weight:

$$W_k = W_k - \eta (\nabla_{W_k} Loss)^T$$

- Until $Loss$ has converged

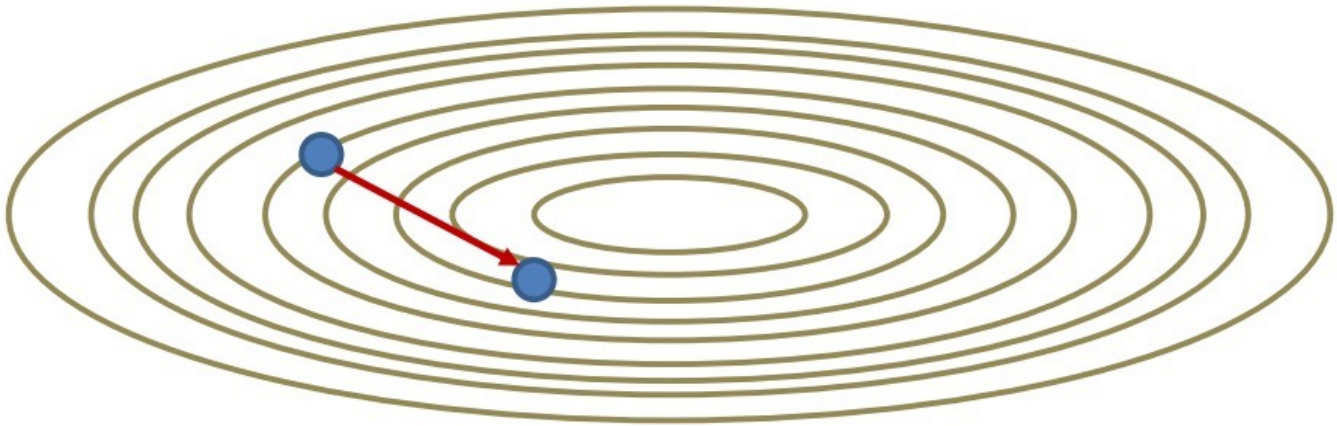
# Training with momentum

- Initialize all weights $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_K$
- Do:
  - For all layers $k$, initialize $\nabla_{W_k} Loss = 0, \Delta W_k = 0$
  - For all $t = 1:T$
    - For every weight:
      - Compute gradient $\nabla_{W_k} \boldsymbol{Div}(Y_t, d_t)$
      - $\nabla_{W_k} Loss \mathrel{+}= \frac{1}{T} \nabla_{W_k} \boldsymbol{Div}(Y_t, d_t)$
  - For every weight:

$$\Delta W_k = \beta \Delta W_k - \eta \left(\nabla_{W_k} Loss\right)^T$$
$$W_k = W_k + \Delta W_k$$

- Until $Loss$ has converged

# Momentum Update



- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss\left(W^{(k-1)}\right)^T$$

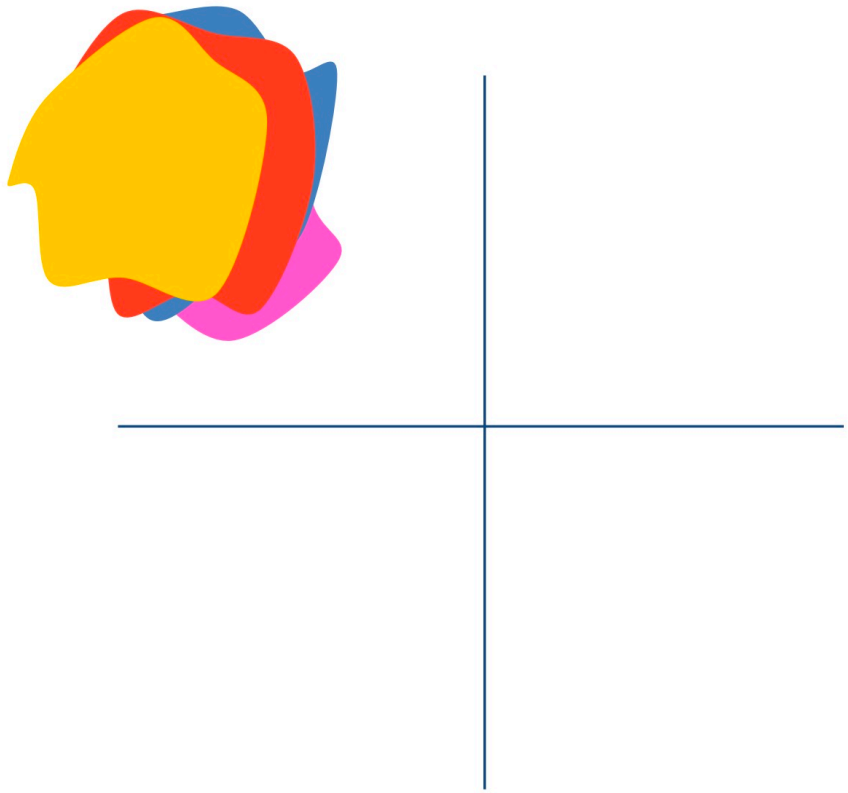
- At any iteration, to compute the current step:
  - First computes the gradient step at the current location
  - Then adds in the scaled previous step
    - Which is actually a running average (滑动平均)
  - To get the final step

# Momentum Update

- Momentum update steps are actually computed in two stages
  - First: We take a step against the gradient at the current location
  - Second: Then we add a scaled version of the previous step
- The procedure can be made more optimal by reversing the order of operations..

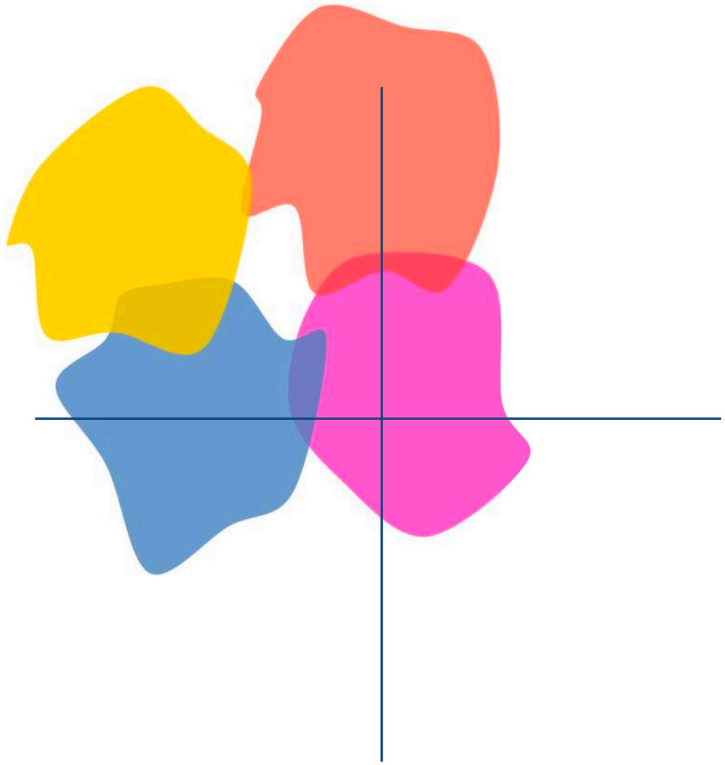
# Lecture 5.4 Covariate Shifts 协变量偏移

# The problem of covariate shifts



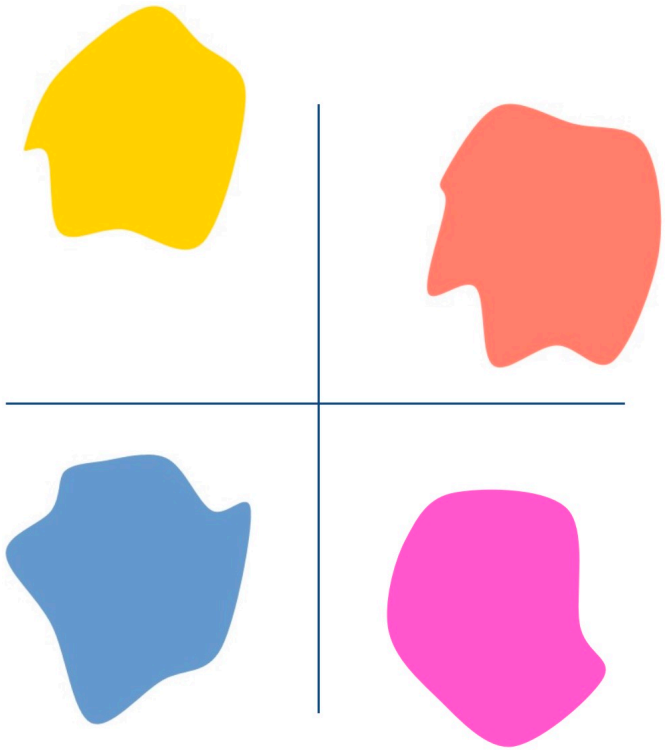
- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution

# The problem of covariate shifts



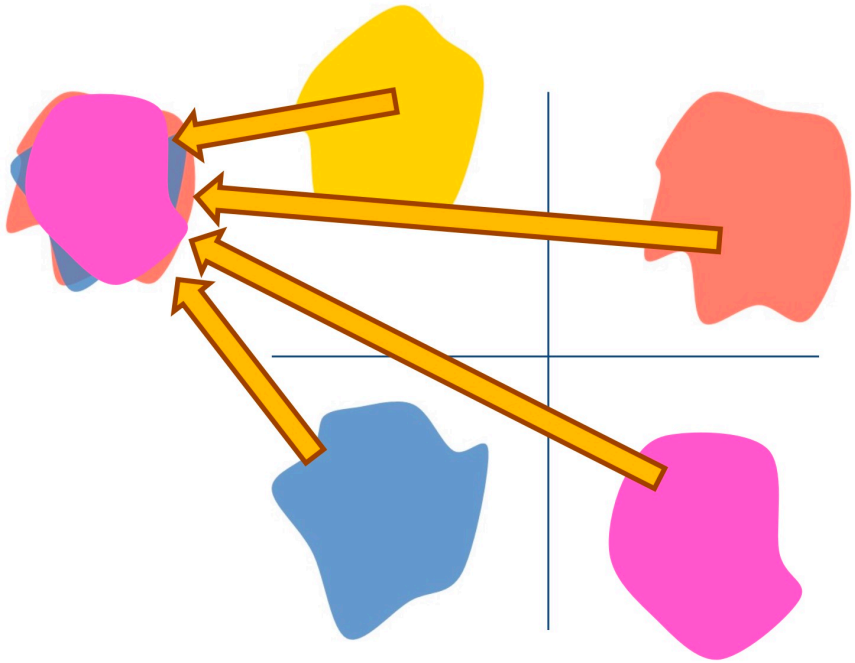
- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A "covariate shift" (协变量偏移)
  - Which may occur in each layer of the network

# The problem of covariate shifts



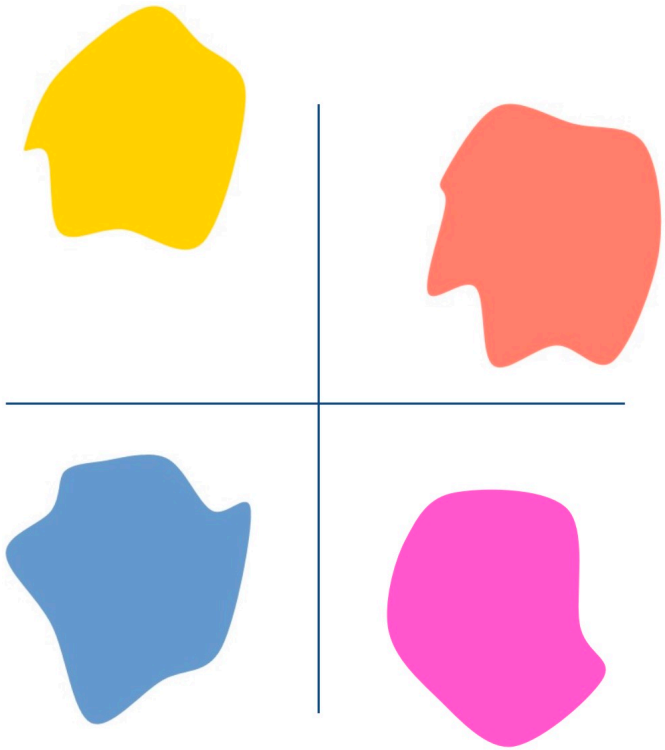
- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A "covariate shift"
- The shifts can be large!
  - Can affect training badly

# Solution: Move all minibatches to a "standard" location



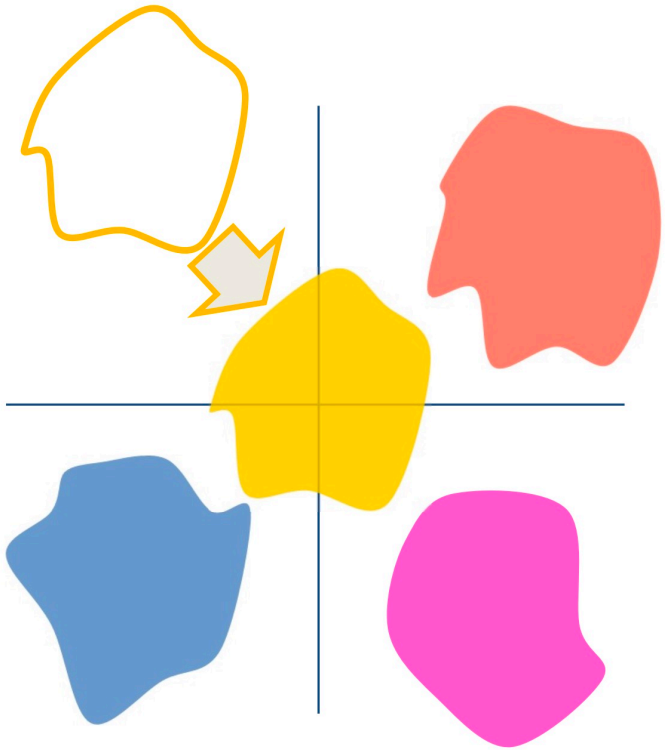
- "Move" all batches to a "standard" location of the space
  - But where?
  - To determine, we will follow a two-step process

# Move all minibatches to a "standard" location



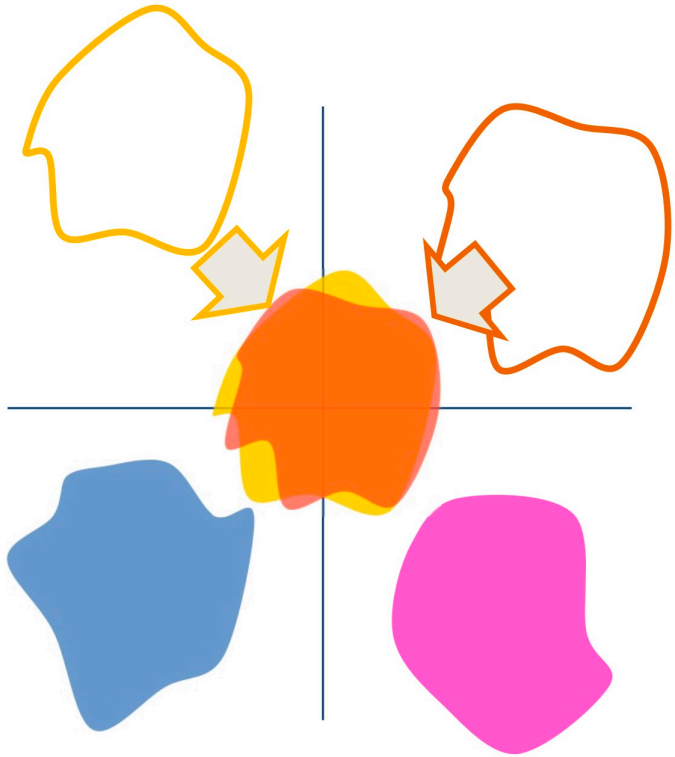
- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

# Move all minibatches to a "standard" location



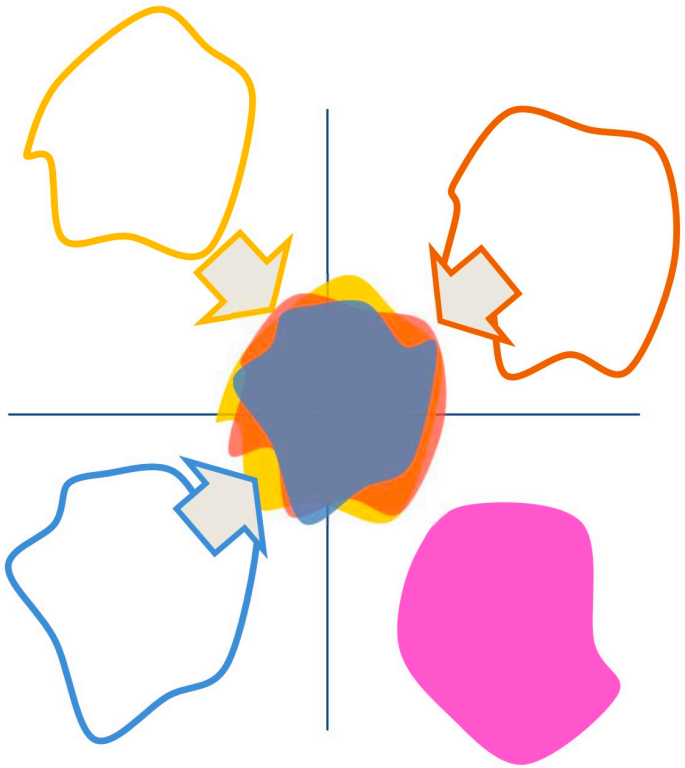
- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

# Move all minibatches to a "standard" location



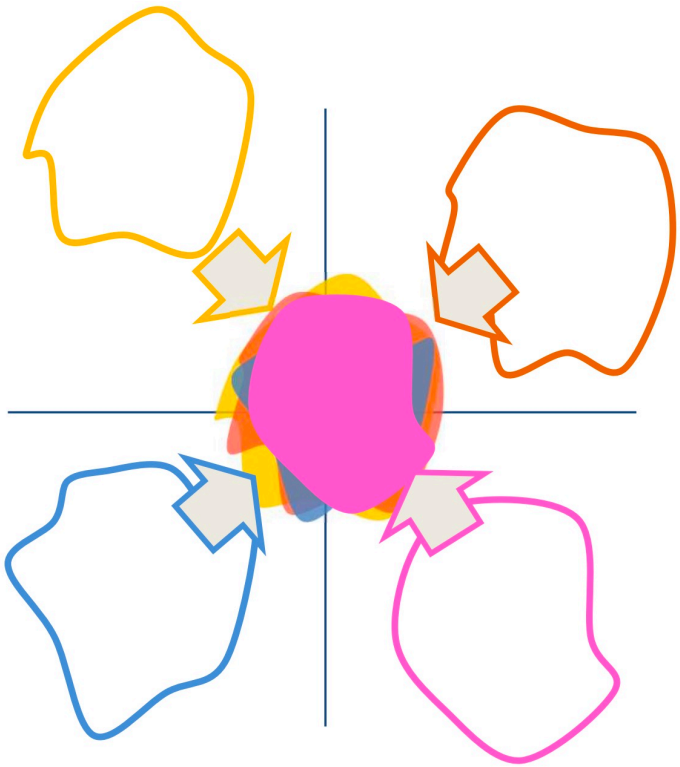
- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

# Move all minibatches to a "standard" location



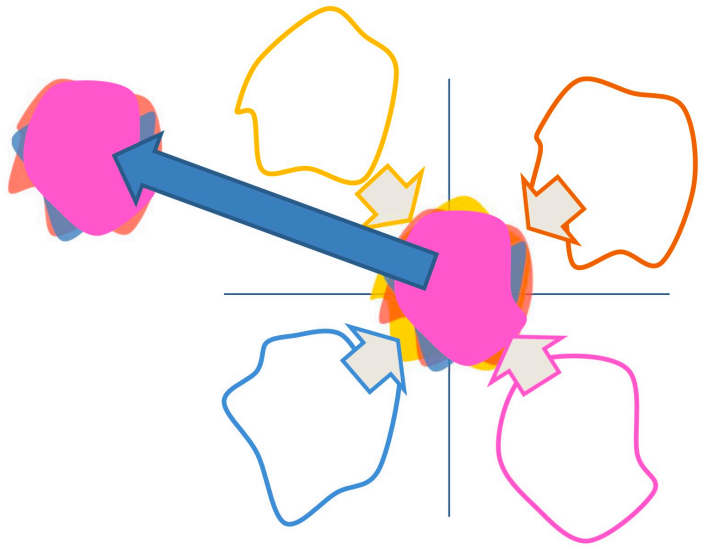
- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

# Move all minibatches to a "standard" location



- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

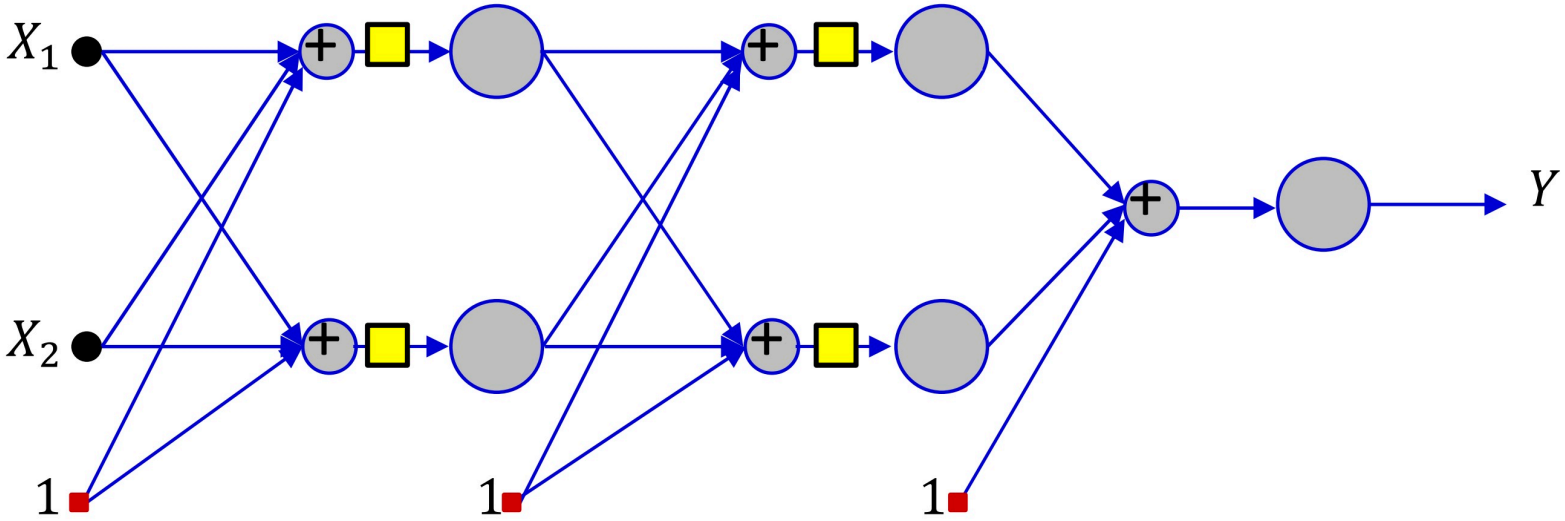
# (Mini)Batch Normalization



- "Move" all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
- **Then move the entire collection to the appropriate location**

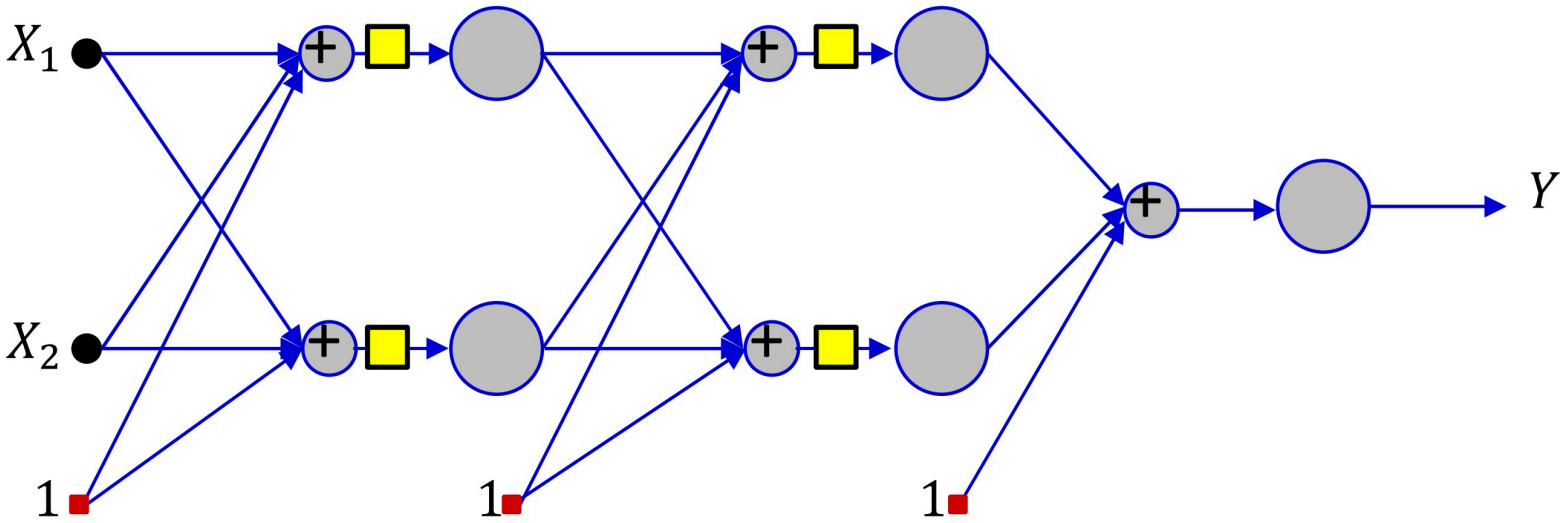
# Lecture 5.5 Batch Normalization 批归一化

# Batch normalization



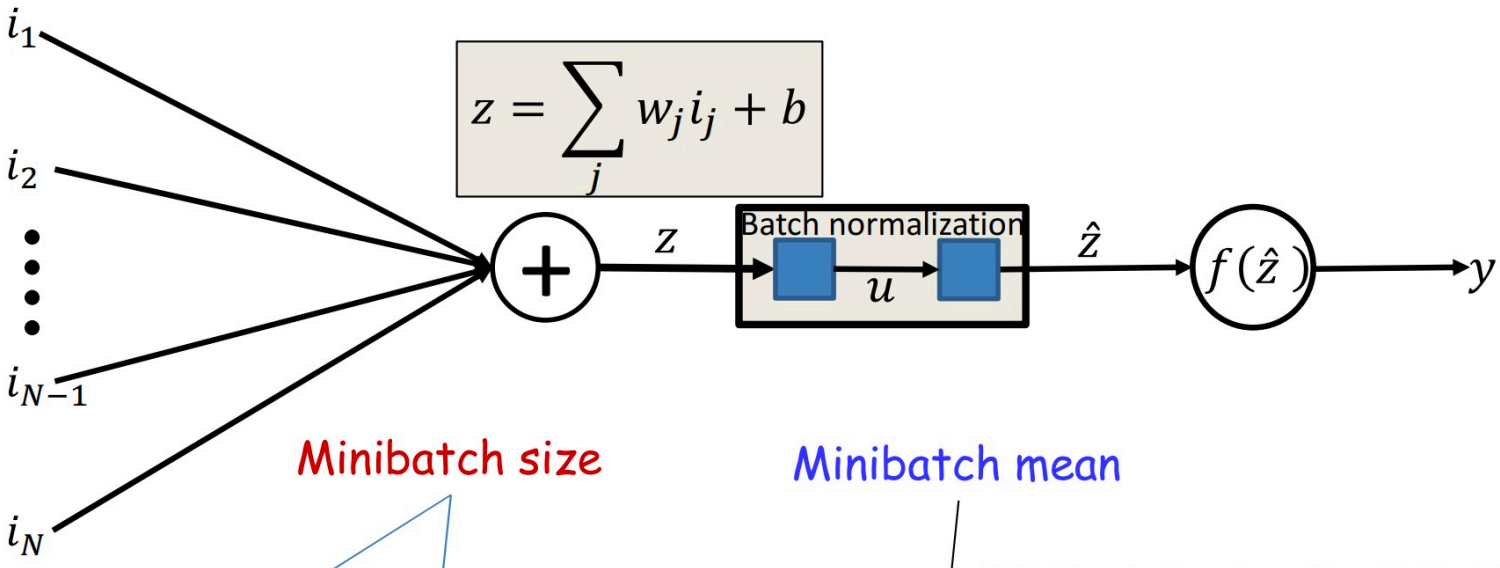

- Batch normalization is a shift-adjustment unit that happens after the weighted addition of inputs but before the application of activation (加权求和之后，激活之前)
  - Is done independently for each unit, to simplify computation

# Batch normalization




- Batch normalization is a shift-adjustment unit that happens after the weighted addition of inputs but before the application of activation (加权求和之后，激活之前)
  - Is done independently for each unit, to simplify computation
- Training: The adjustment occurs over individual minibatches

# Batch normalization: Training




Minibatch size

Minibatch mean

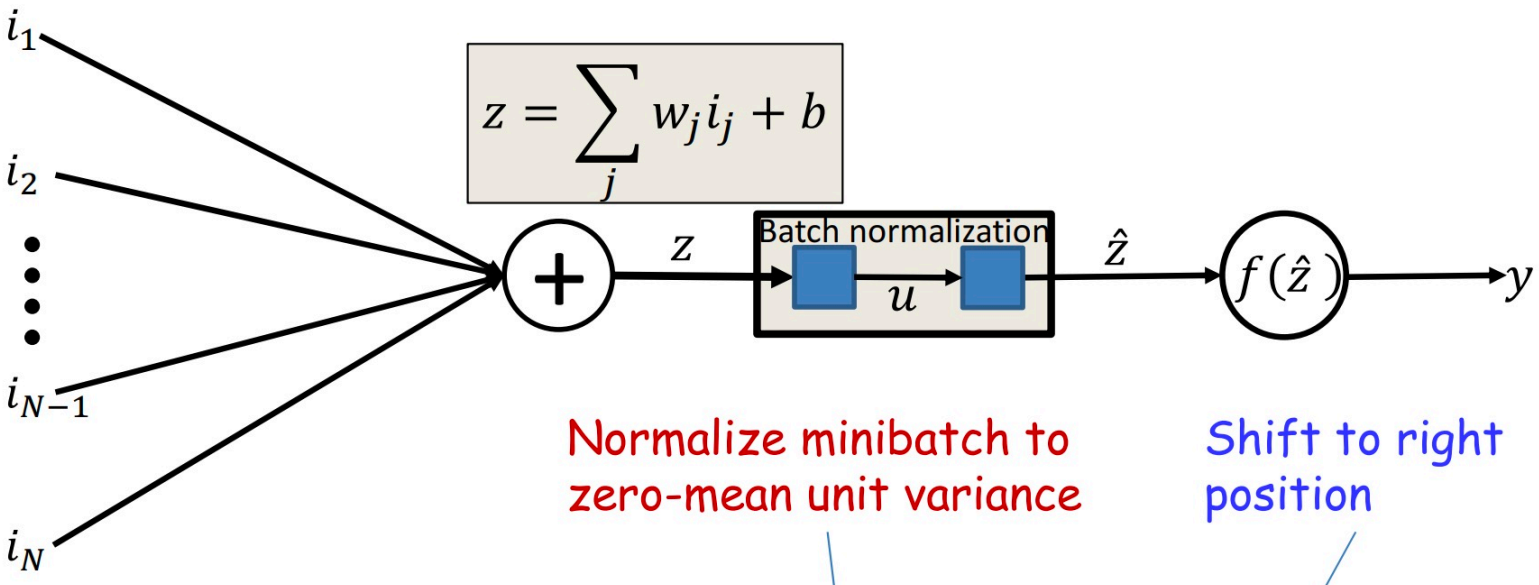
Minibatch standard deviation

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i \qquad \sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \hat{z}_i = \gamma u_i + \beta$$

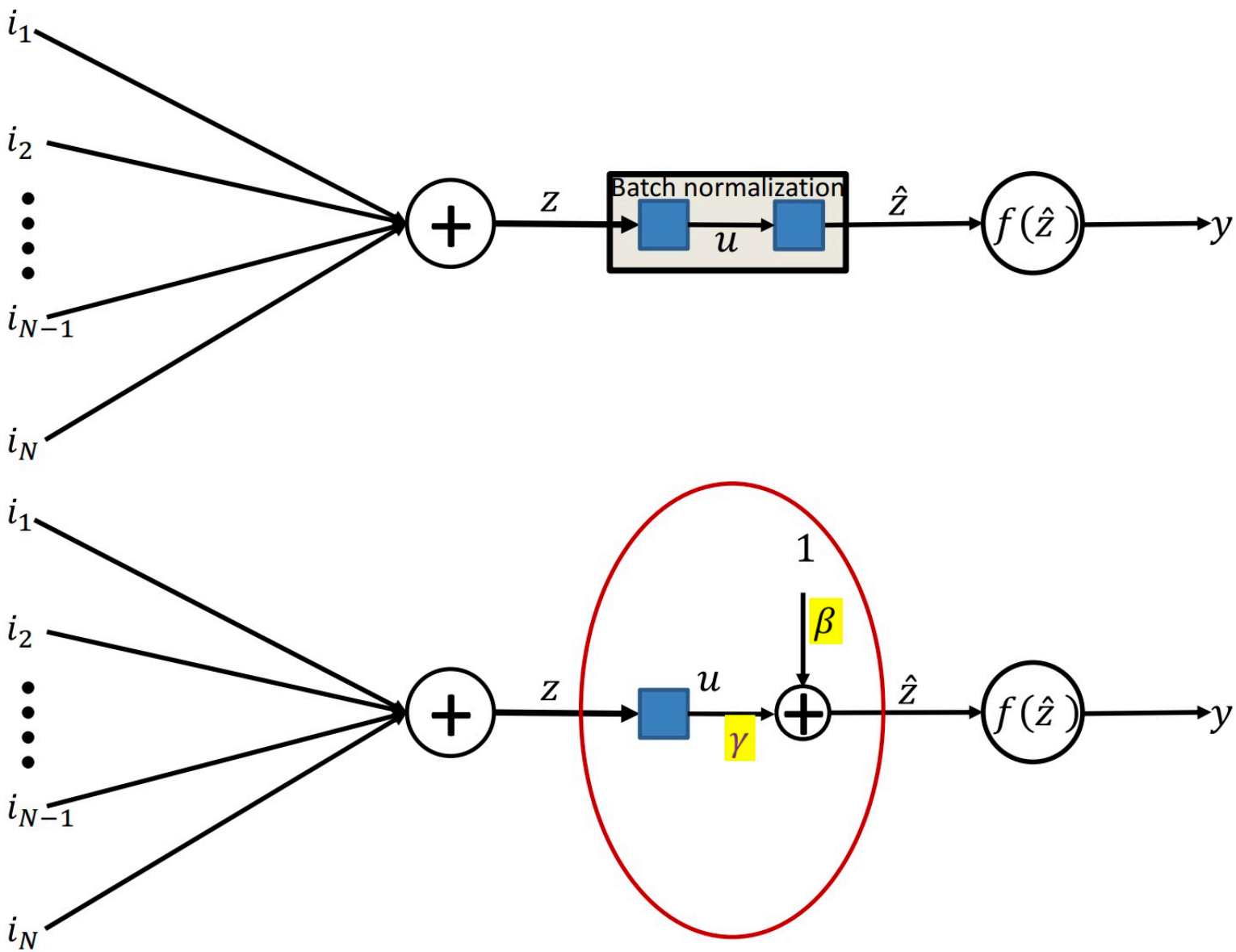
- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are "shifted" to a *unit-specific* location

# Batch normalization: Training




- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are "shifted" to a *unit-specific* location

# A better picture for batch norm



# A note on derivatives: Usual

In conventional training:

- The loss is the average of the divergence between the actual and desired outputs for all inputs in the minibatch

$$Loss(minibatch) = \frac{1}{B}\sum_{t} Div(Y_t(X_t), d_t(X_t))$$

- The derivative of the minibatch loss w.r.t. network parameters is the average of the derivatives of the divergences for the individual training instances w.r.t. parameters

$$\frac{dLoss(minibatch)}{dw_{i,j}^{(k)}} = \frac{1}{B}\sum_{t} \frac{dDiv(Y_t(X_t), d_t(X_t))}{dw_{i,j}^{(k)}}$$

- The output of the network in response to an input, and the derivative of the divergence for any input are independent of other inputs in the minibatch

# A note on derivatives: BatchNorm

- The outputs are now functions of $\mu_B$ and $\sigma_B^2$, which are functions of the entire minibatch

$$Loss(minibatch) = \frac{1}{B}\sum_t Div(\, Y_t(X_t, \mu_B, \sigma_B^2),\, d_t(X_t))$$

- The Divergence for each $Y_t$ depends on all the $X_t$ within the minibatch
  - Training instances within the minibatch are no longer independent

# The actual divergence with BN

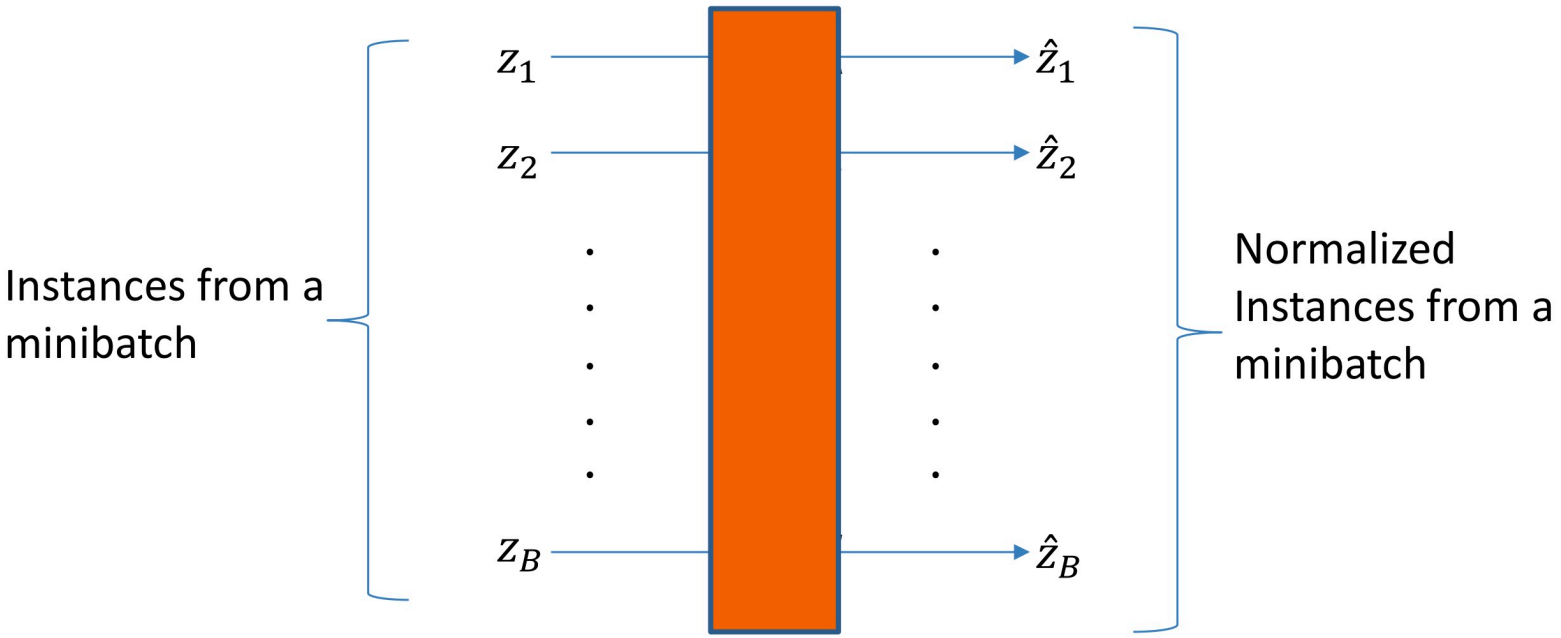
- The actual divergence for any minibatch

$$Loss(minibatch)$$

$$= \frac{1}{B}\sum_{t} Div\left(Y_t\left(X_t, \mu_B(X_t, X_{t'\neq t}), \sigma_B^2\left(X_t, X_{t'\neq t}, \mu_B(X_t, X_{t'\neq t})\right)\right), d_t(X_t)\right)$$

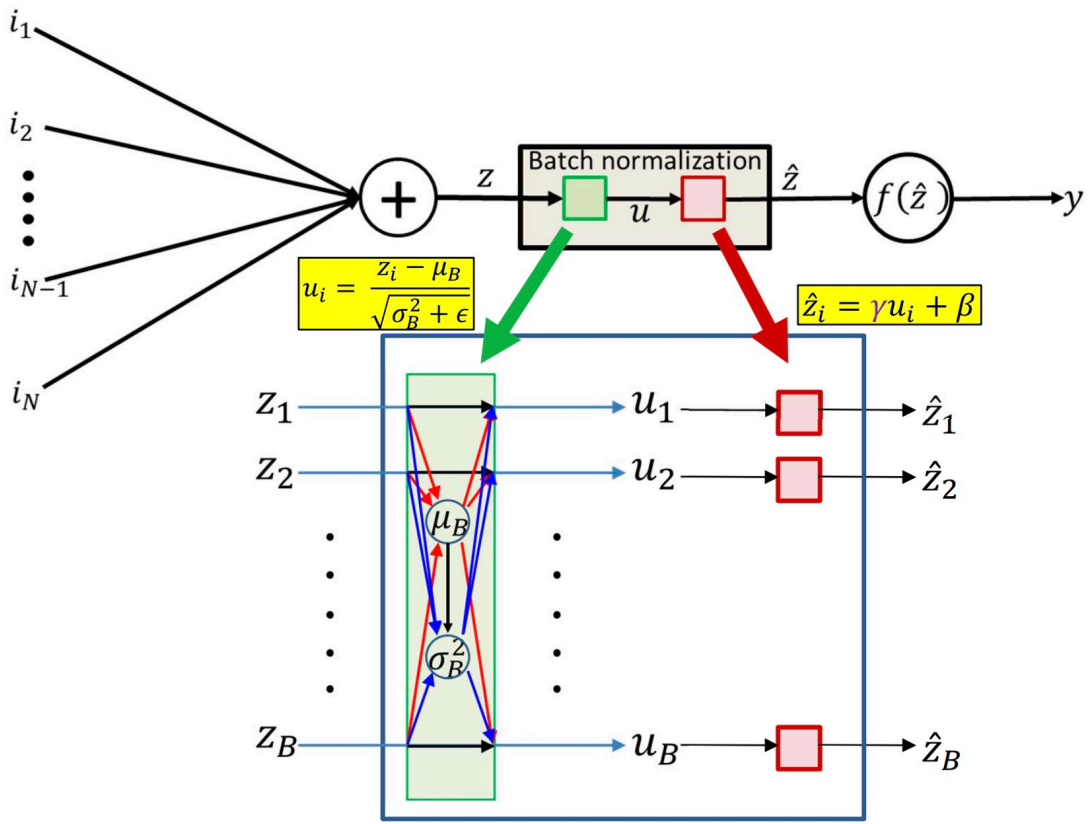
- We need the derivative for this function
- To derive the derivative let's consider the dependencies at a single neuron

# Batchnorm is a vector function over the minibatch



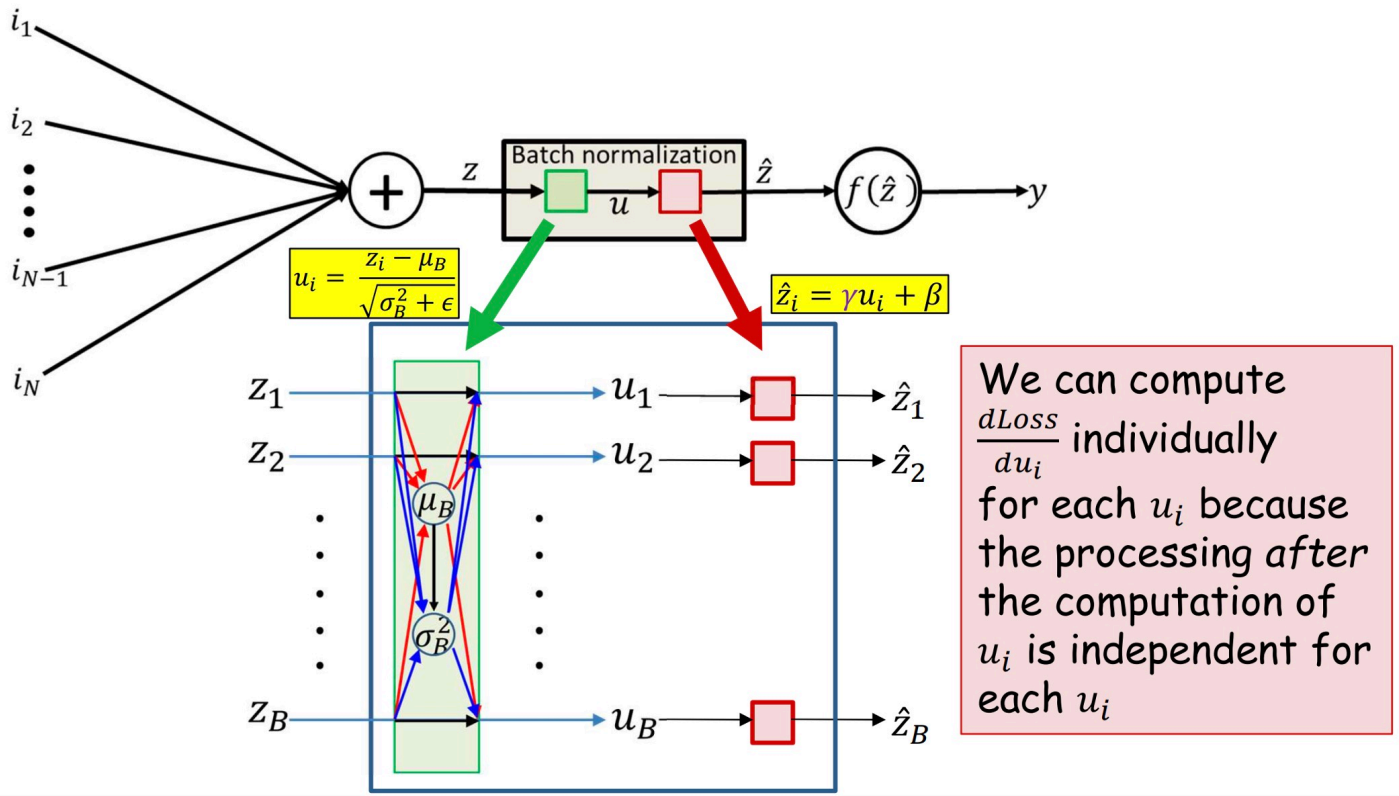

- Batch normalization is really a vector function applied over all the inputs from a minibatch
  - Every $z_i$ affects every $\hat{z}_i$
  - Shown on the next slide
- To compute the derivative of the minibatch loss w.r.t any $z_i$, we must consider all $\hat{z}_i$ in the batch

# Or more explicitly



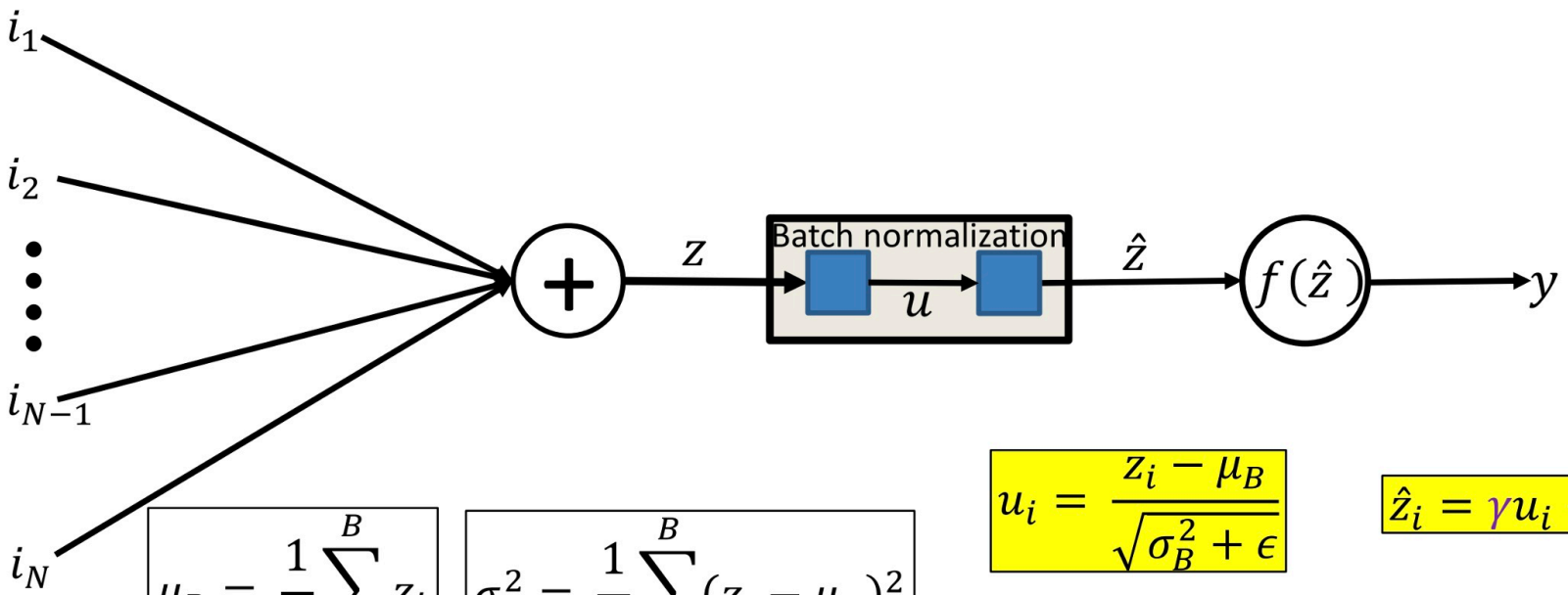

- Mini-batch normalized $u$'s is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each $u$ to compute the corresponding $\hat{z}_i$

# Or more explicitly




- Mini-batch normalized $u$'s is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each $u$ to compute the corresponding $\hat{z}_i$

# Batch normalization: Forward pass



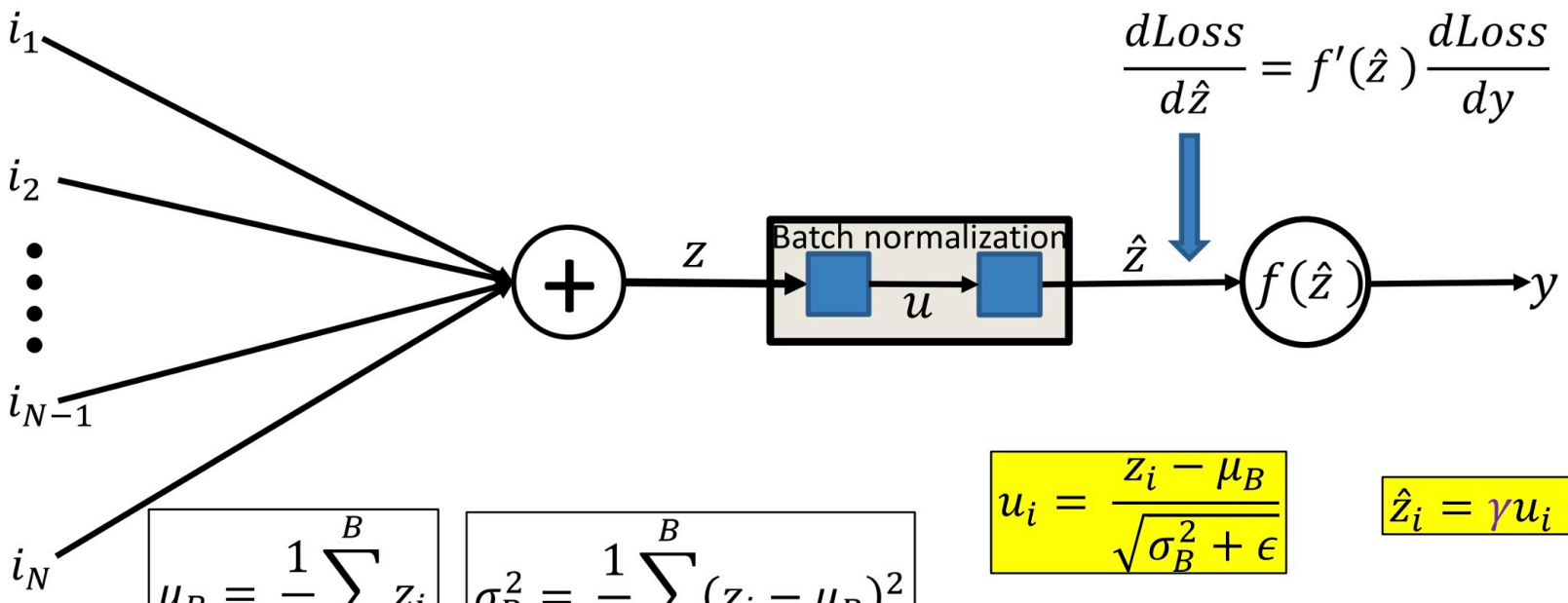

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i$$

$$\sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B} (z_i - \mu_B)^2$$

$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

# Batch normalization: Backpropagation

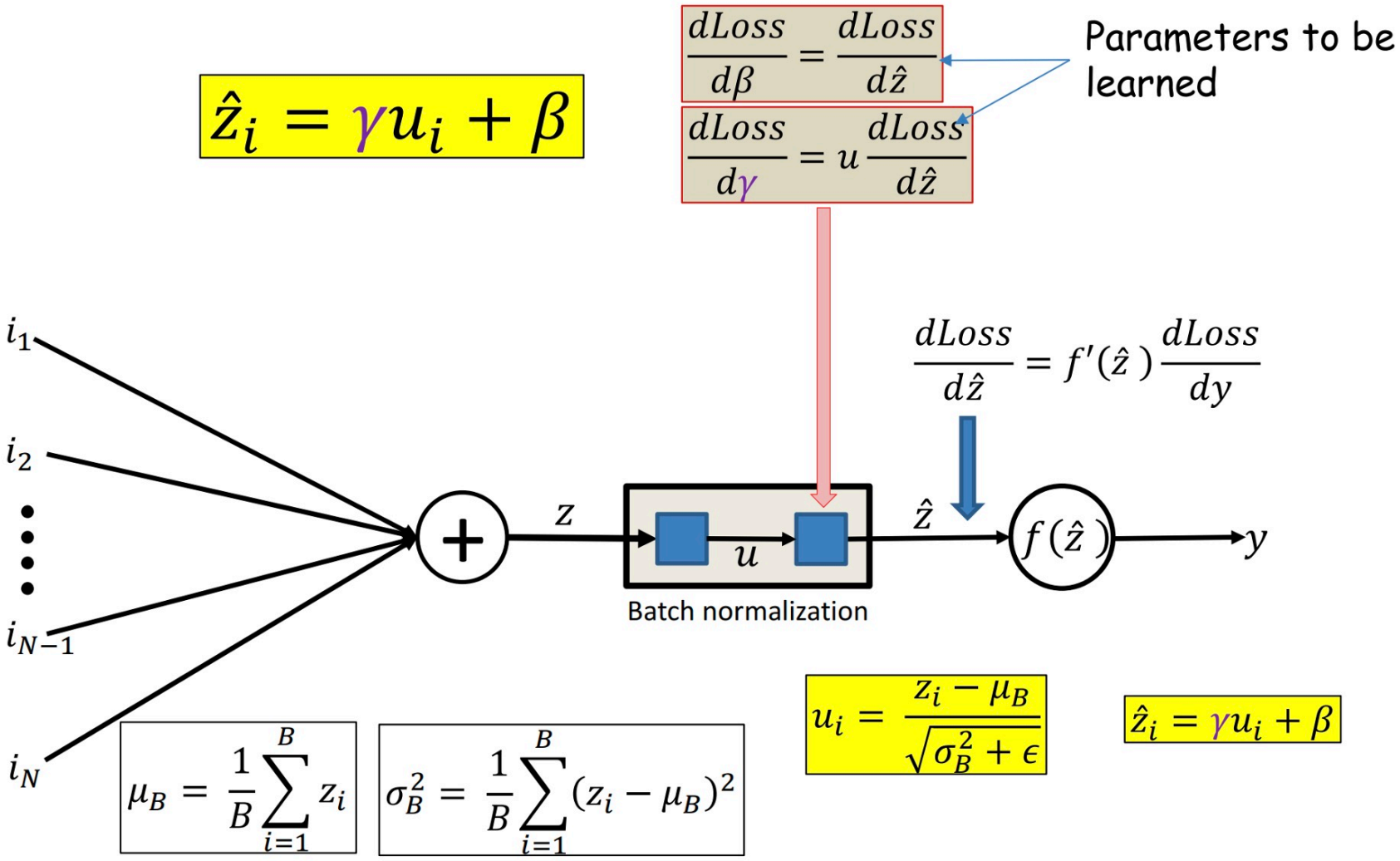



$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} z_i \qquad \sigma_B^2 = \frac{1}{B}\sum_{i=1}^{B}(z_i - \mu_B)^2$$

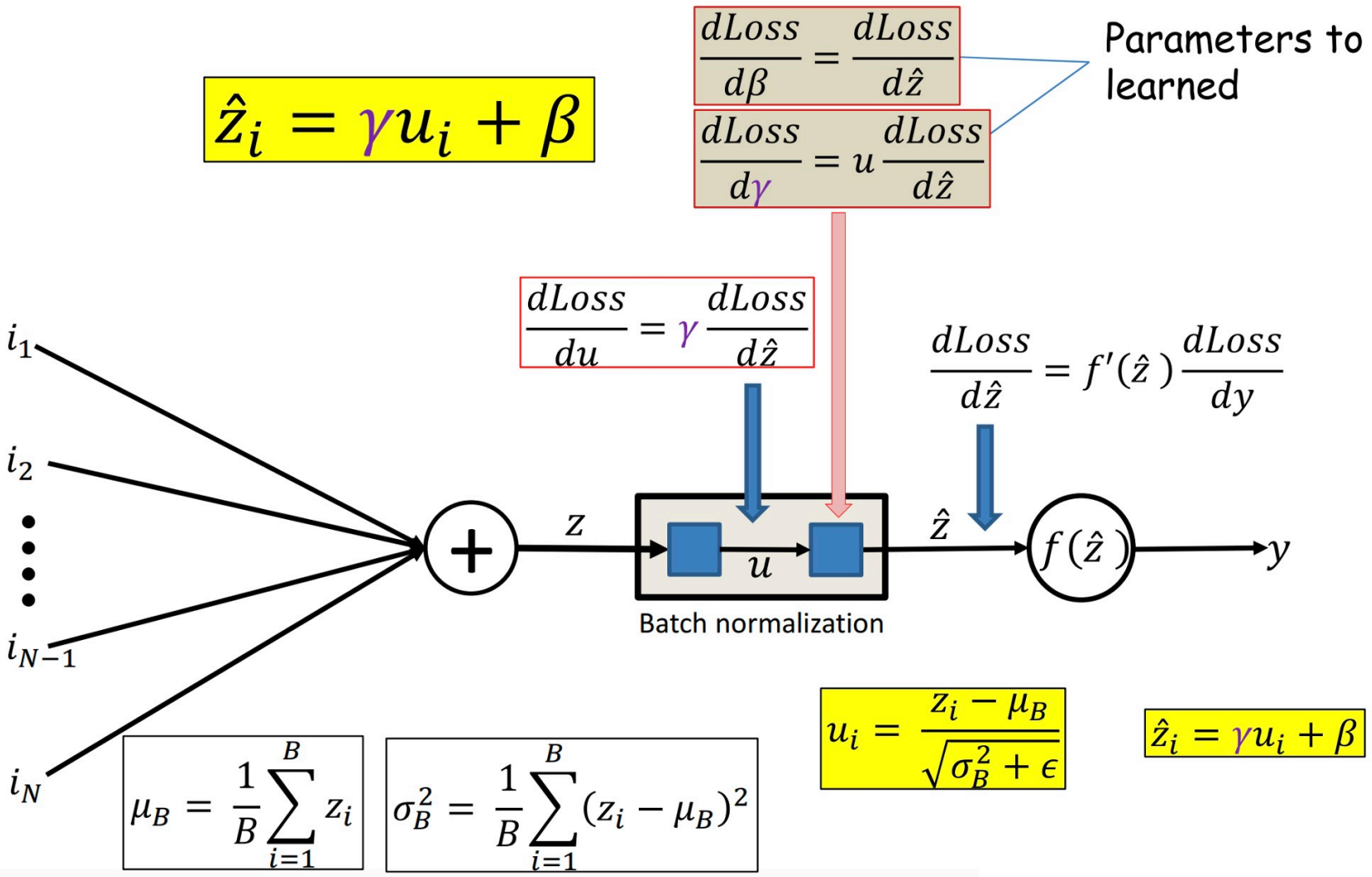
$$u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\hat{z}_i = \gamma u_i + \beta$$

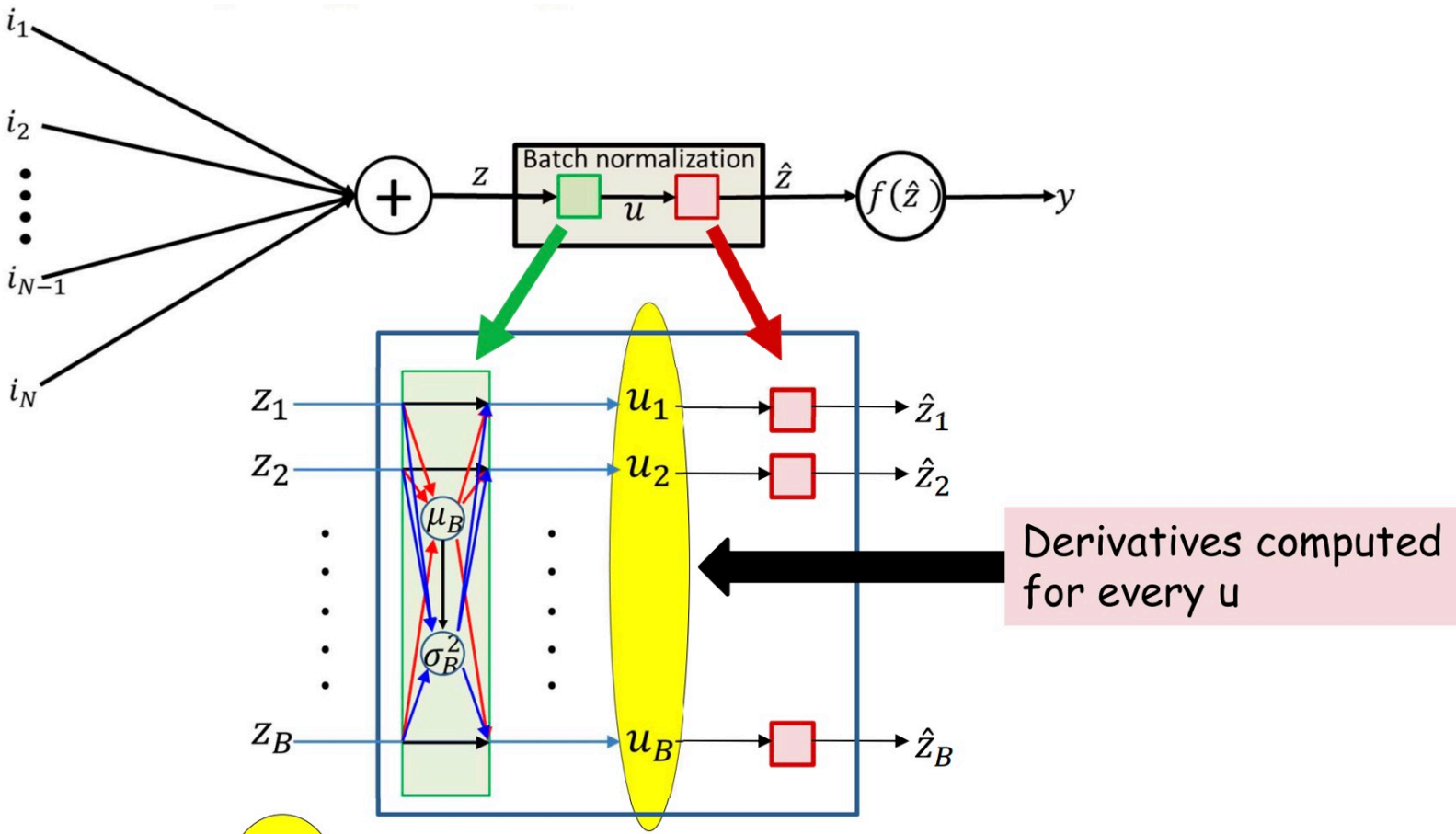
# Batch normalization: Backpropagation



# Batch normalization: Backpropagation

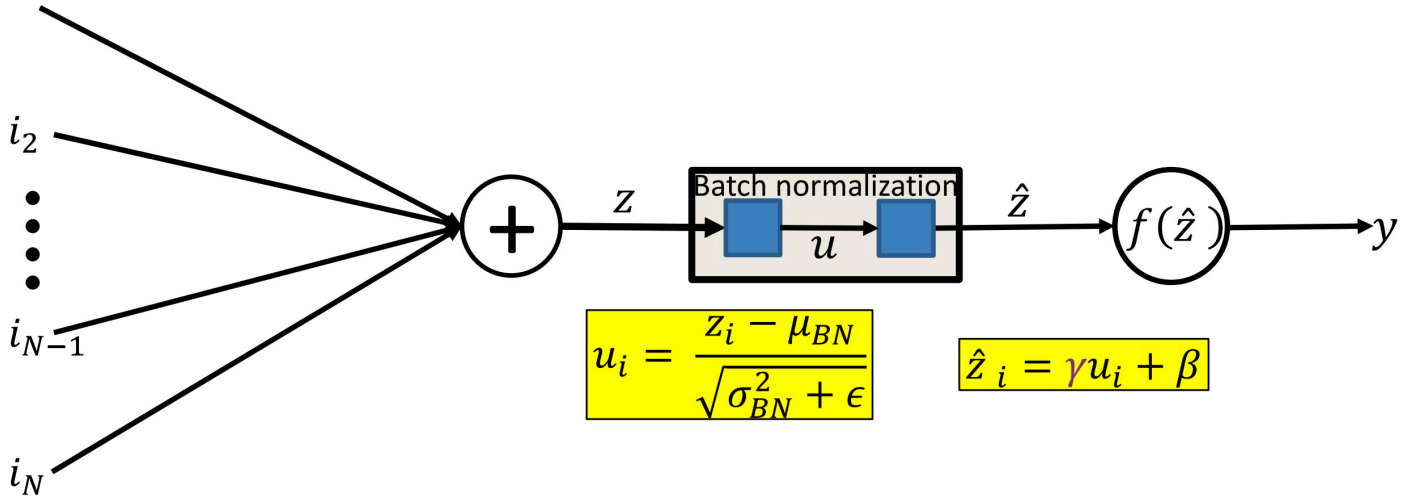


# Propogating the derivative




- We now have $\frac{dLoss}{du_i}$ for every $u_i$
- We must propagate the derivative through the first stage of BN
  - Which is a vector operation over the minibatch

# Batch normalization: Inference

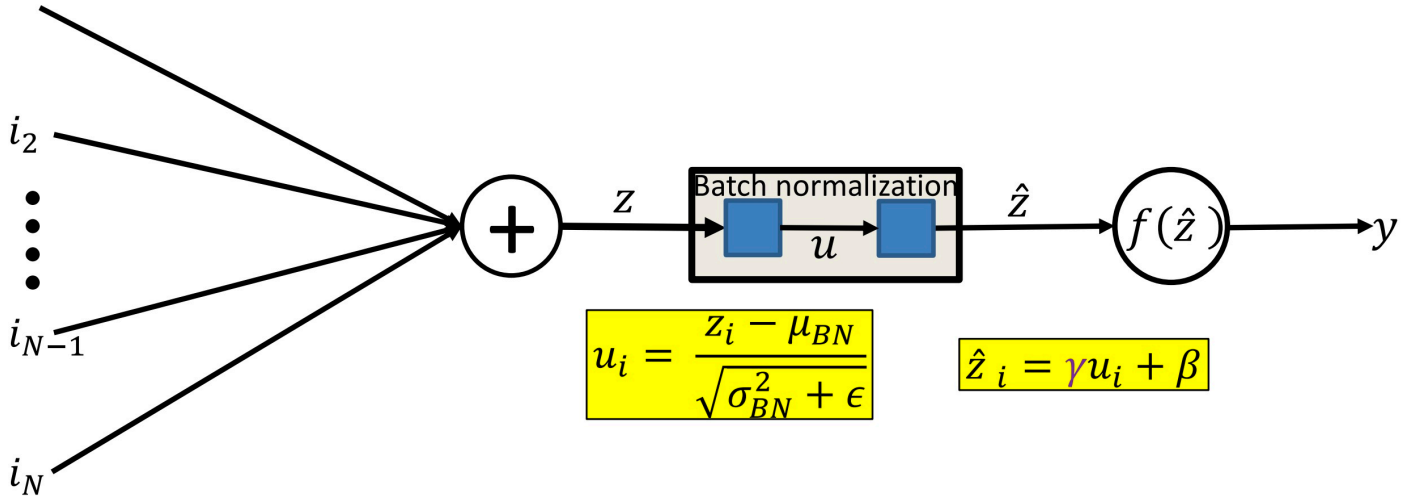



- On test data, BN requires $\mu_B$ and $\sigma_B^2$.
- We will use the average over all training minibatches

$$\mu_{BN} = \frac{1}{Nbatches} \sum_{batc} \mu_B(batch)$$

$$\sigma_{BN}^2 = \frac{B}{(B-1)Nbatches} \sum_{batc} \sigma_B^2(batch)$$

# Batch normalization: Inference



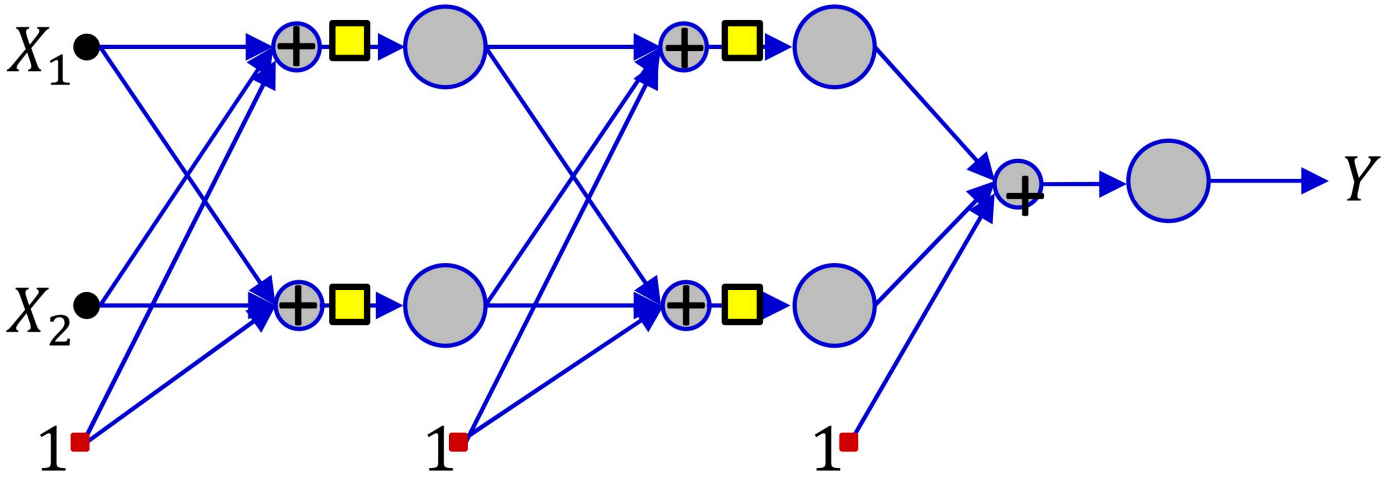

- On test data, BN requires $\mu_B$ and $\sigma_B^2$.
- We will use the average over all training minibatches

$$\mu_{BN} = \frac{1}{Nbatches} \sum_{batc} \mu_B(batch)$$

$$\sigma_{BN}^2 = \frac{B}{(B-1)Nbatches} \sum_{batc} \sigma_B^2(batch)$$

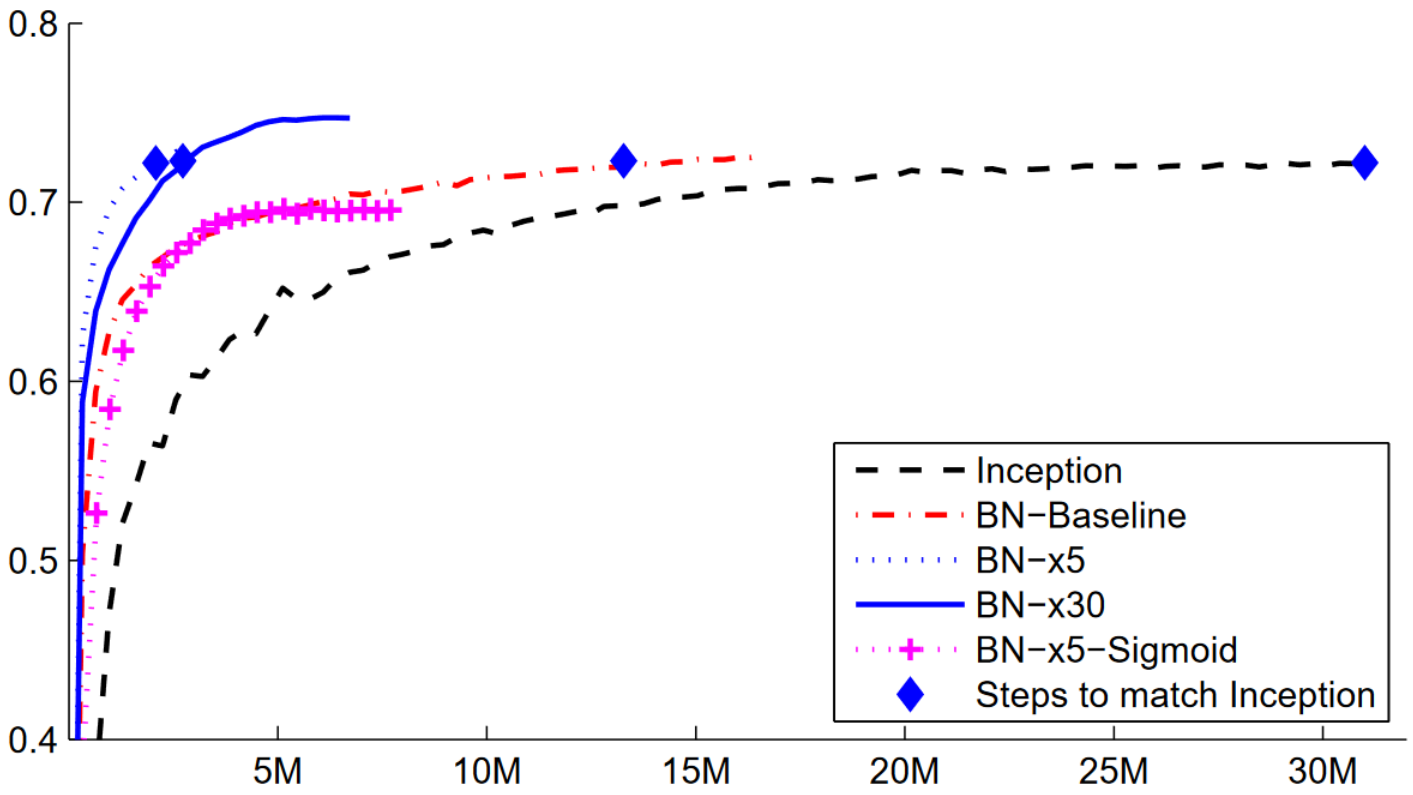
- Note: these are neuron-specific
  - $\mu_B(\cdot)$ and $\sigma_B^2(\cdot)$ are obtained from the final converged network
  - $B/(B-1)$ term gives an unbiased estimator for the variance

# Batch normalization




- Batch normalization may only be applied to some layers
  - Or even only selected neurons in the layer
- Improves both convergence rate and network performance
  - Anecdotal evidence that BN eliminates the need for dropout
  - To get maximum benefit from BN, learning rates must be increased and learning rate decay can be faster
- Also needs better randomization of training data order

# Batch Normalization: Typical result



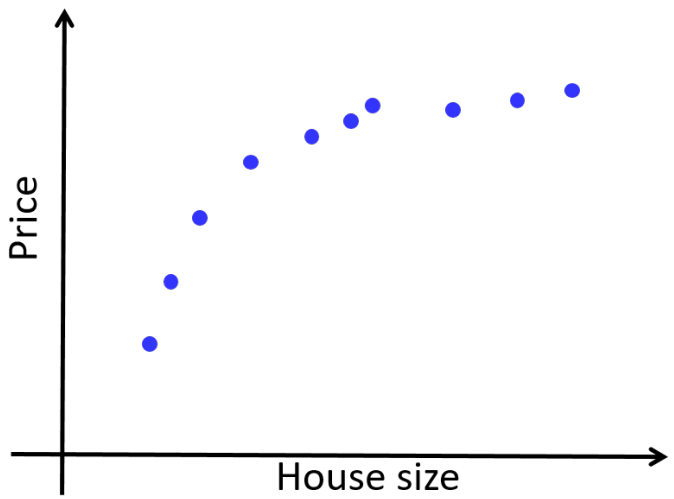

- Performance on Imagenet, from Ioffe and Szegedy, JMLR 2015

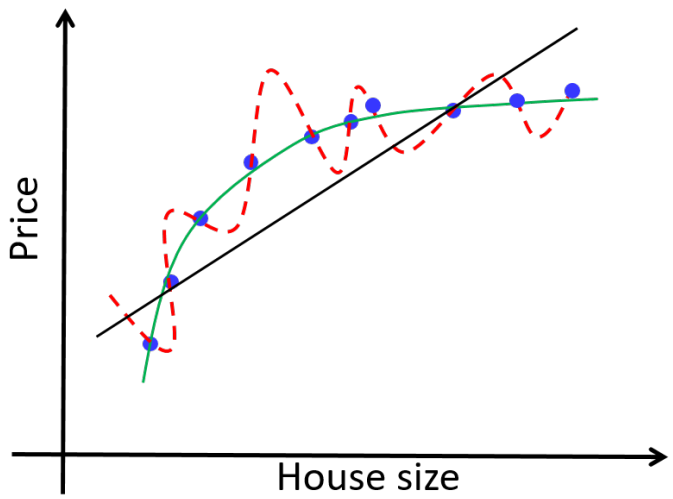
# Story so far

- Gradient descent can be sped up by incremental updates
  梯度下降可以通过增量更新的方式加速
- Convergence can be improved using smoothed updates
  通过平滑更新可以改进神经网络的收敛
- The choice of divergence affects both the network and results
  损失函数的选择很重要
- Covariate shift between samples may cause problems and may be handled by batch normalization
  协方差偏移可以通过批归一化进行处理

# Lecture 5.6 Overfitting 过拟合

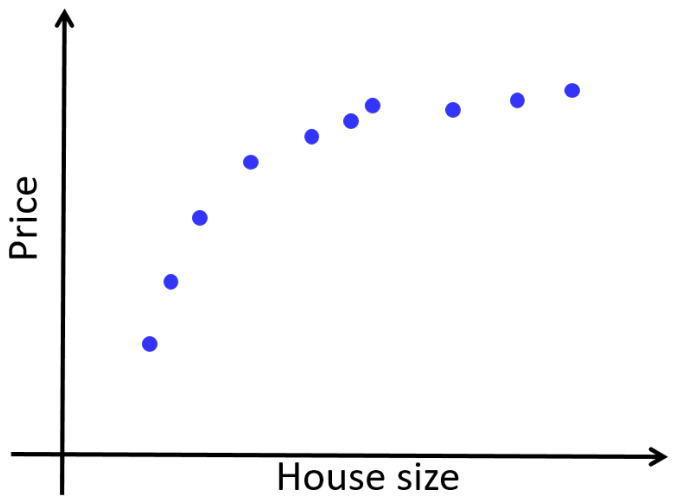
# Overfitting issue

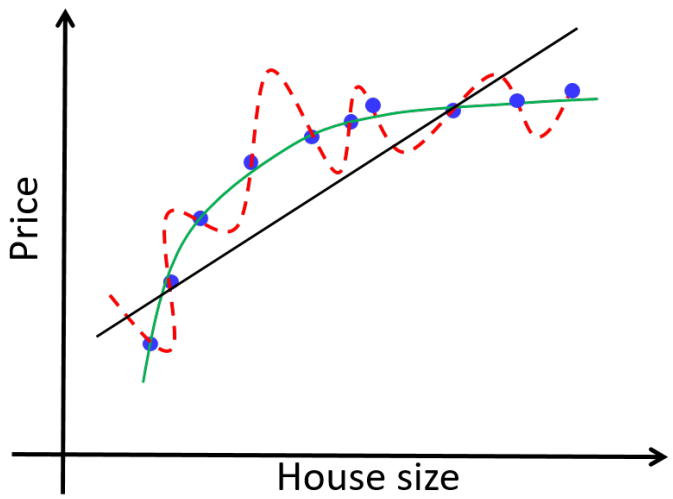



- Overfitting (red curve): trained to predict training data too accurate to be generalizable!

# Overfitting issue




- Overfitting (red curve): trained to predict training data too accurate to be generalizable!

死记硬背 ➡ 过犹不及

# How to reduce overfitting?

- Rule: train the network model to make it have better *generalization* (泛化) performance!

# How to reduce overfitting?

- Rule: train the network model to make it have better *generalization* (泛化) performance!

## Generalization ability

How well does the trained model work for unseen data?

# How to reduce overfitting?

- Rule: train the network model to make it have better *generalization* (泛化) performance!

## Generalization ability

How well does the trained model work for unseen data?

举一反三

# Data set splitting for model generalization

Whole data set divided:

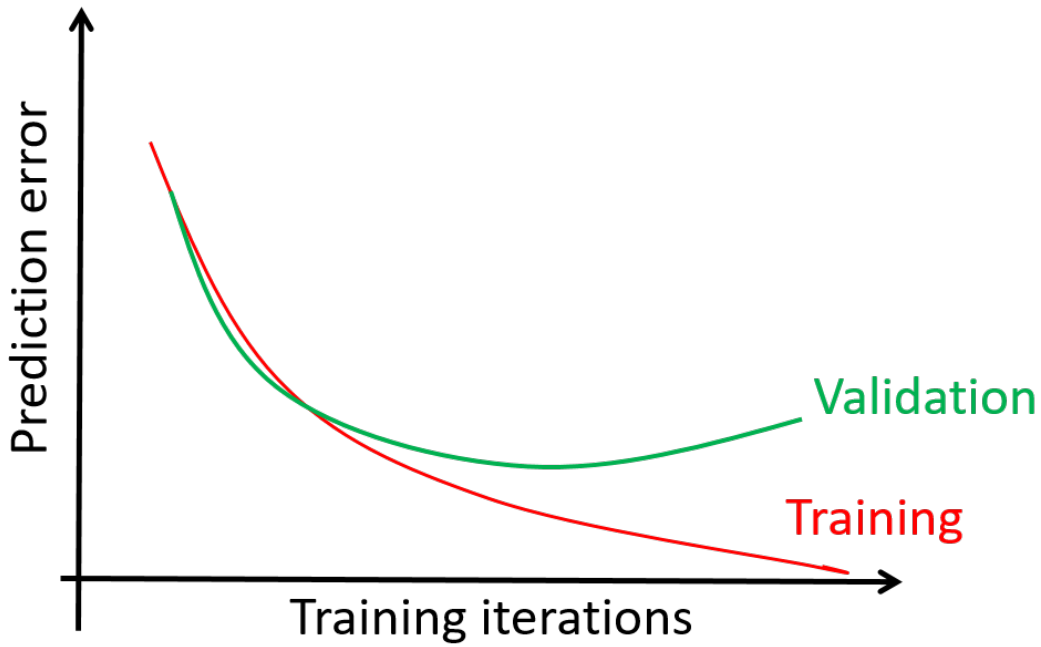
- Training set (e.g., 60%): used to train model parameter
- Validation set (e.g., 20%): to determine when to stop training
- Test set (e.g., 20%): to evaluate final model's performance

Training set

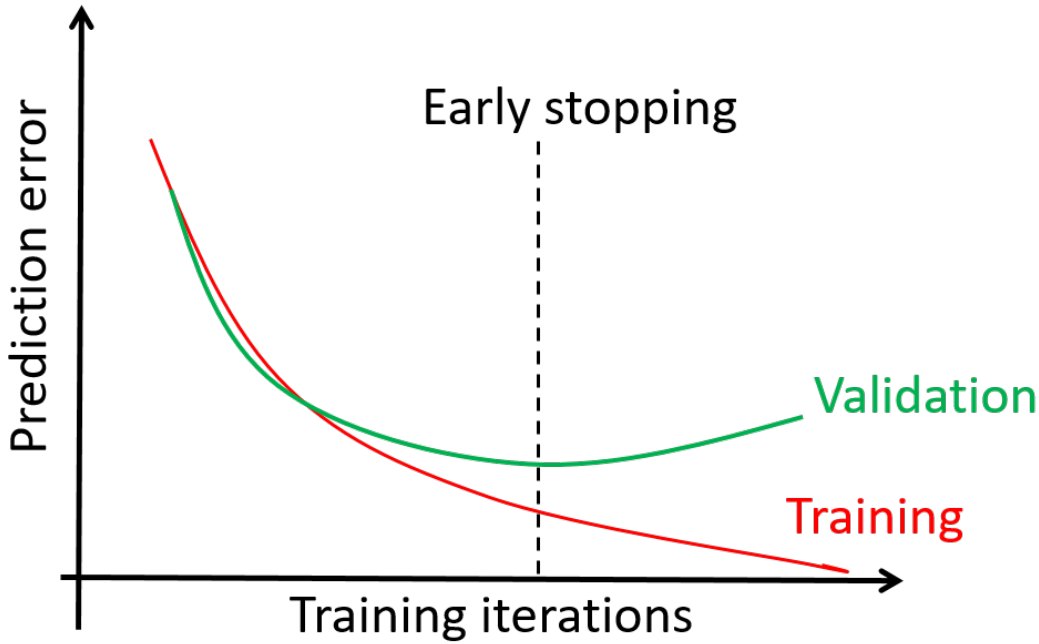
Validation set

Test set

# Prevent overfitting: early stopping 早停

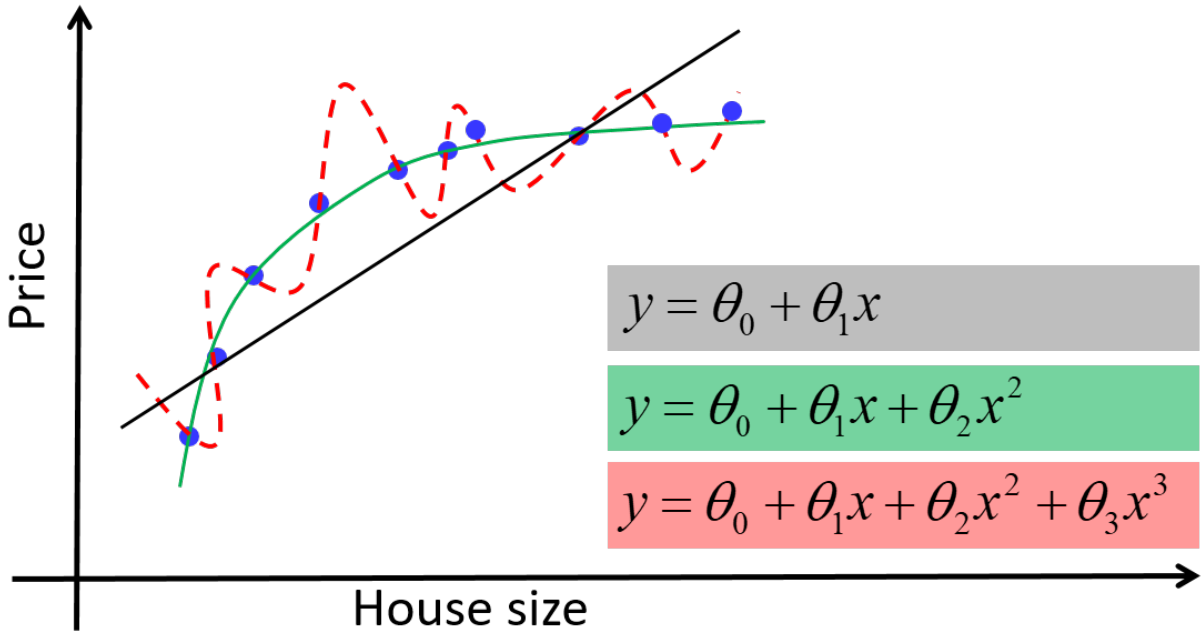


# Prevent overfitting: early stopping 早停



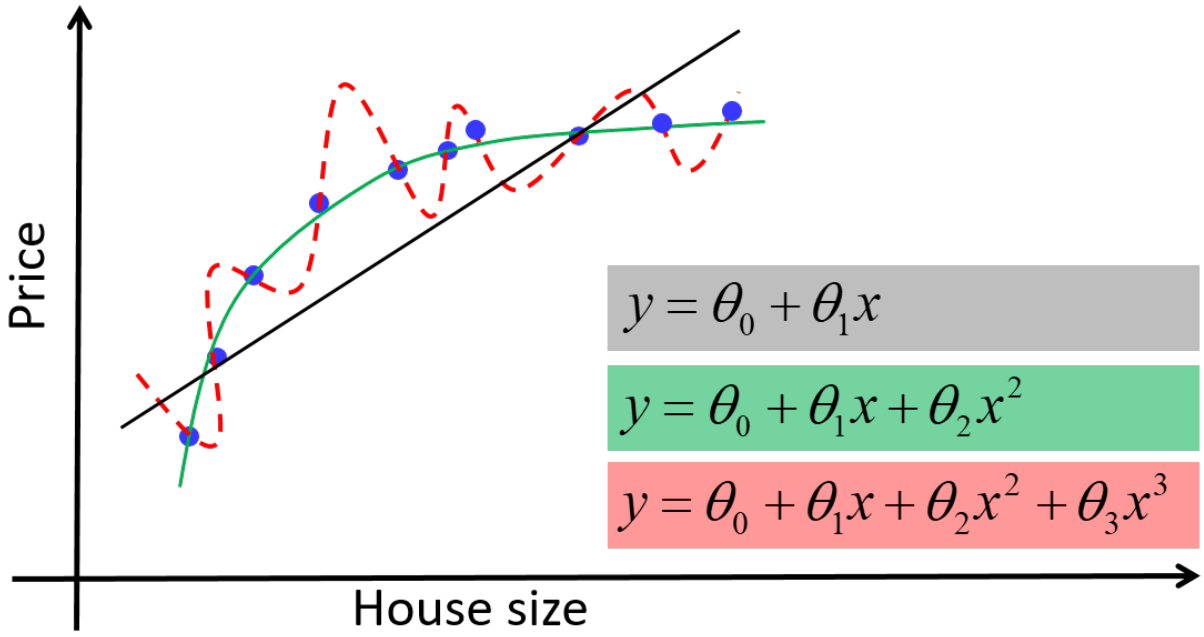

- Early stopping: stop training when prediction error on validation set does not decrease.

# Regularization (正则化) for model generalization: $L_p$ norm




- More model parameters, more likely to be overfitting
- Fewer model parameters, more likely to have larger loss

# Regularization (正则化) for model generalization: $L_p$ norm




- More model parameters, more likely to be overfitting
- Fewer model parameters, more likely to have larger loss
- So: need trade-off between loss and number of working parameters.

# Regularization: $L_p$ norm (cont')

## $L_p$ regularization

Adding a penalty on large parameter values with $L_p$ norm in the loss function to reduce overfitting:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta})) + \lambda \|\boldsymbol{\theta}\|_p$$

- $L_p$ norm $\|\boldsymbol{\theta}\|_p \equiv (\sum_i |\theta_i|^p)^{1/p}$
- $\lambda$: a hyper-parameter to balance two terms

# Regularization: $L_p$ norm (cont')

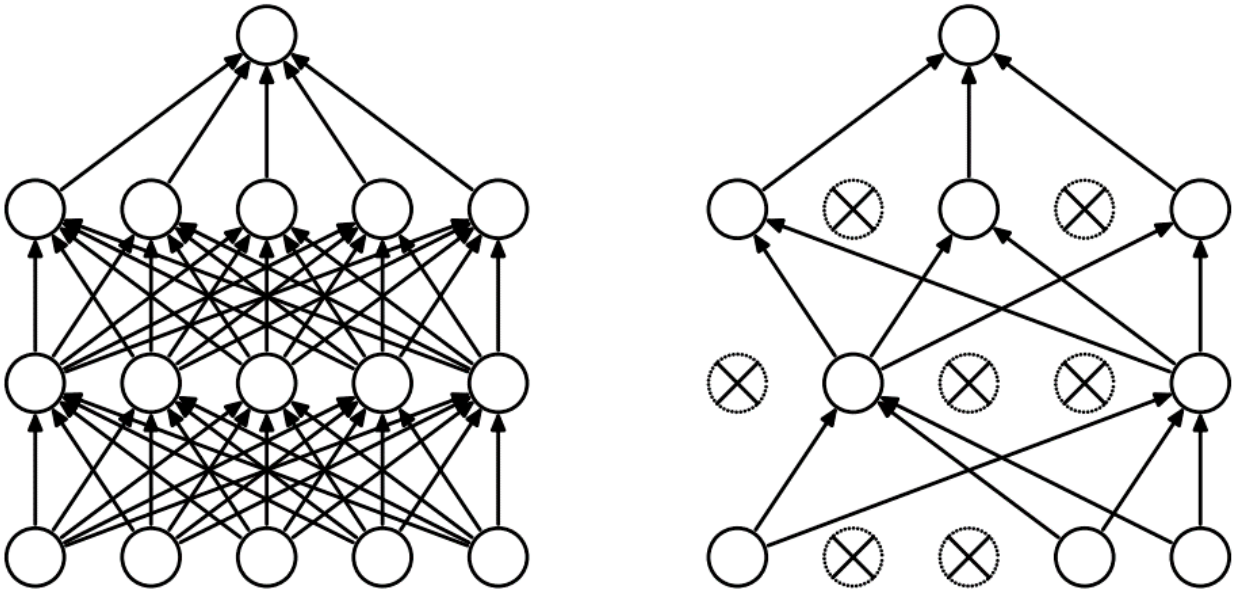
## $L_p$ regularization

Adding a penalty on large parameter values with $L_p$ norm in the loss function to reduce overfitting:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} l(\mathbf{y}_n, \mathbf{f}(\mathbf{x}_n; \boldsymbol{\theta})) + \lambda \|\boldsymbol{\theta}\|_p$$

- $L_p$ norm $\|\boldsymbol{\theta}\|_p \equiv (\sum_i |\theta_i|^p)^{1/p}$
- $\lambda$: a hyper-parameter to balance two terms
- $p = 2$: "weight decay", causing smaller weight values
- $p = 1$: causing fewer non-zero weight parameters

# Regularization: Dropout

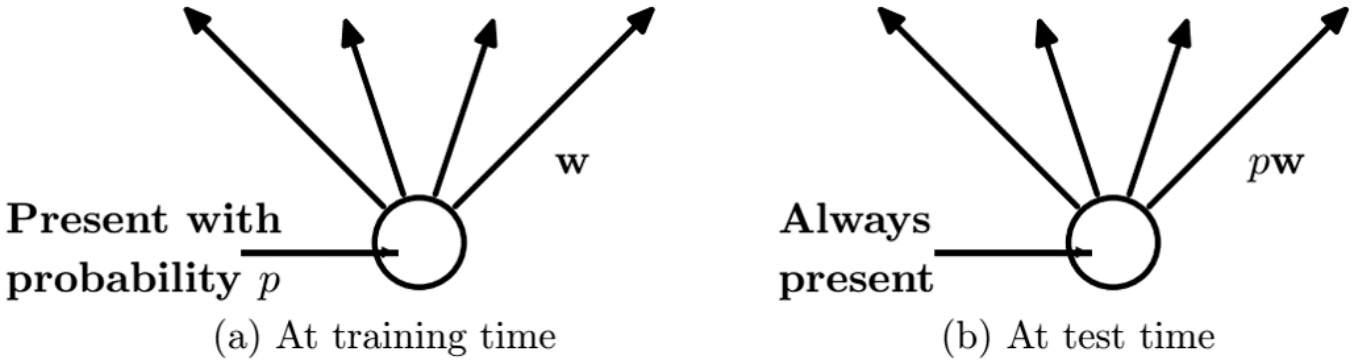


(a) Standard Neural Net

(b) After applying dropout.

- At training, each hidden neuron is present (not dropped out) with probability $p$
- So, each mini-batch is to train a different random structure

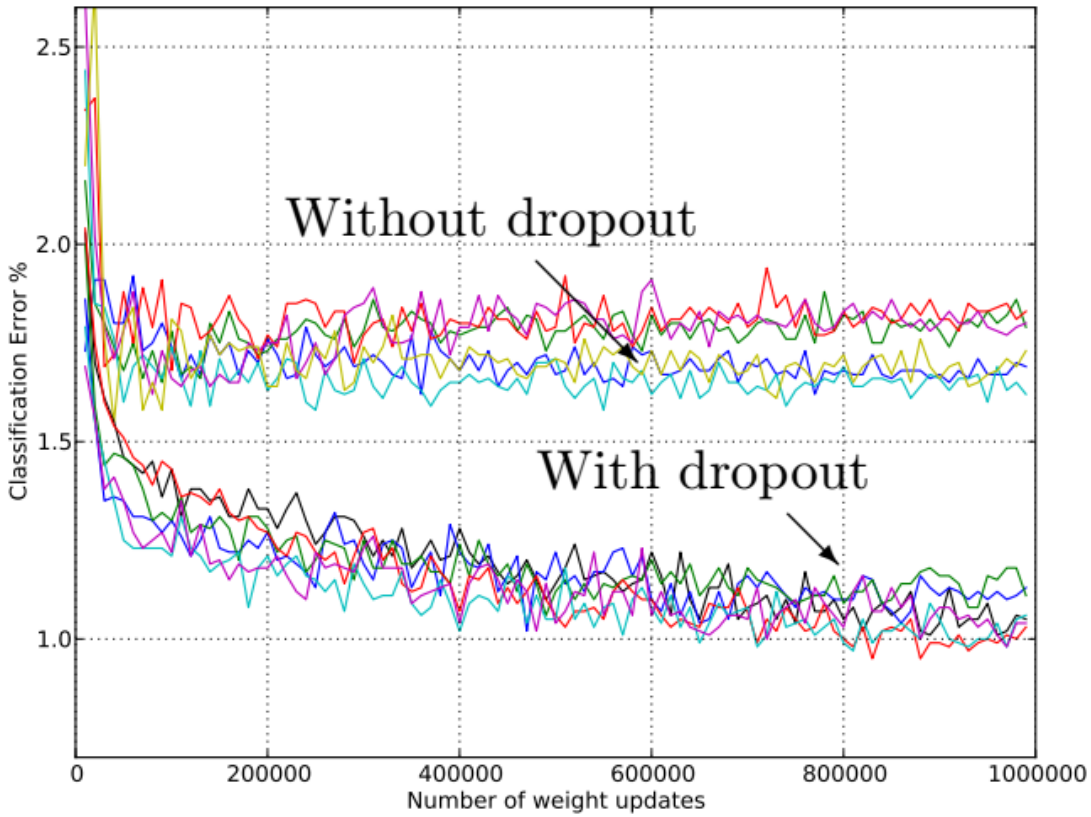
# Regularization: Dropout (cont')




(a) At training time (b) At test time

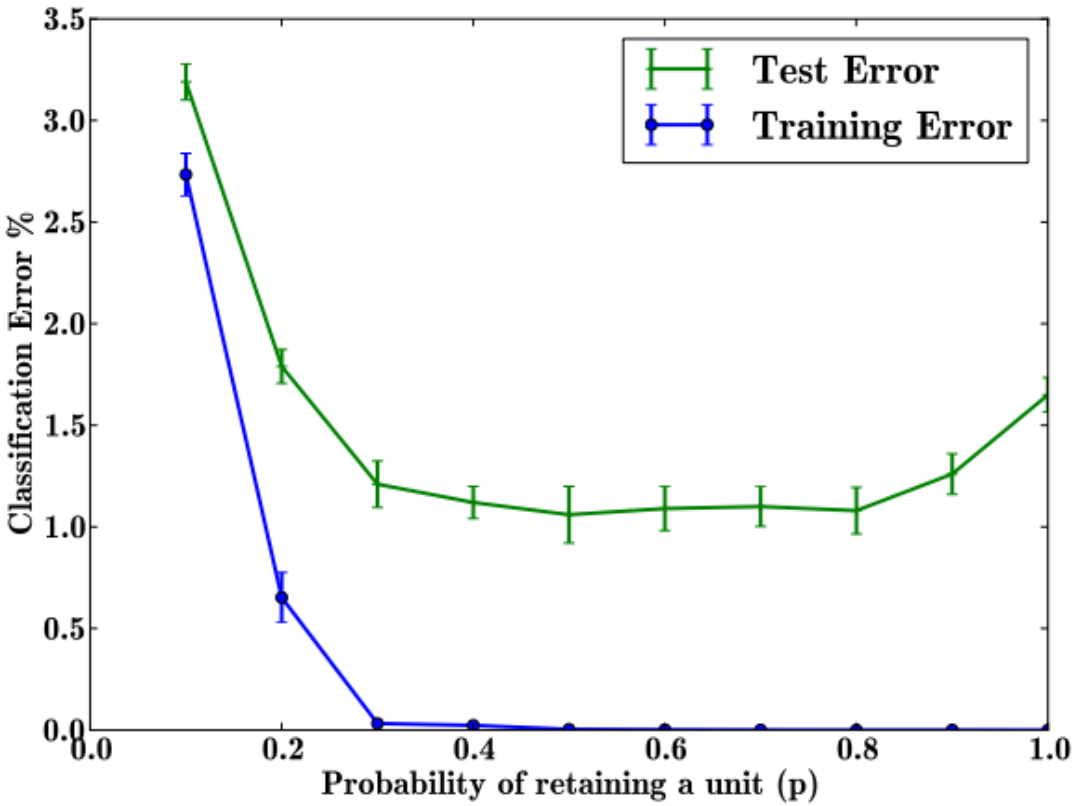
- At test, every neuron is always present. Weights are (down-) scaled by $p$, such that output at test time is same as expected output at training time.

# Regularization: Dropout (cont')




- Dropout reduces test errors on different model architectures (each architecture with a unique color)

# Regularization: Dropout (cont')




- Dropout works well at large range of rate $p$.

# Regularization: Dropout (cont')

Why does dropout work?

- At each training, every retaining neuron is forced to finish the task with less help from other neurons.
- At test time, the whole network approximates the average over many "thinned" (with some neurons dropped) networks.

# Regularization: Dropout (cont')

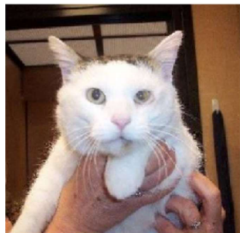
Why does dropout work?

- At each training, every retaining neuron is forced to finish the task with less help from other neurons.
- At test time, the whole network approximates the average over many "thinned" (with some neurons dropped) networks.

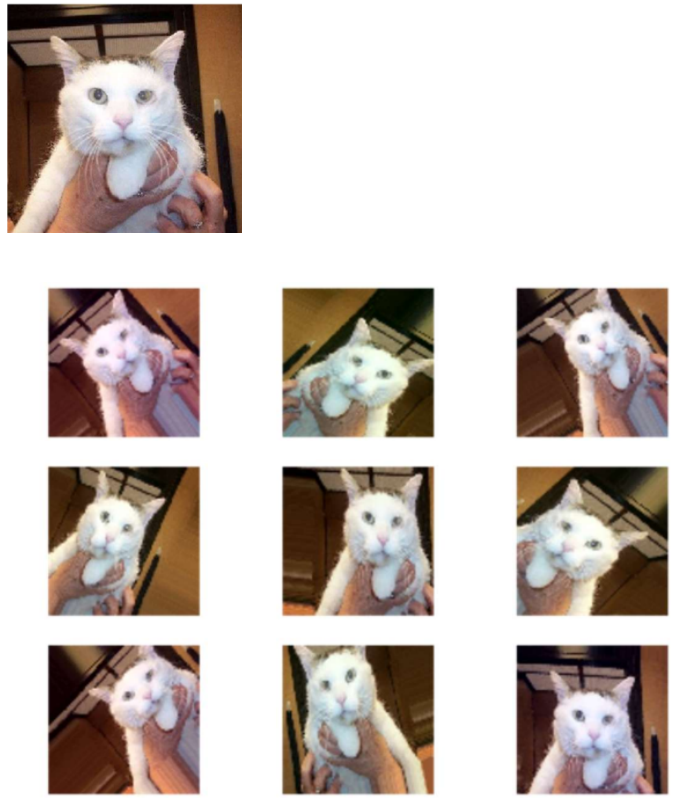
Drawback of dropout:

- It takes 2-3 times longer in training

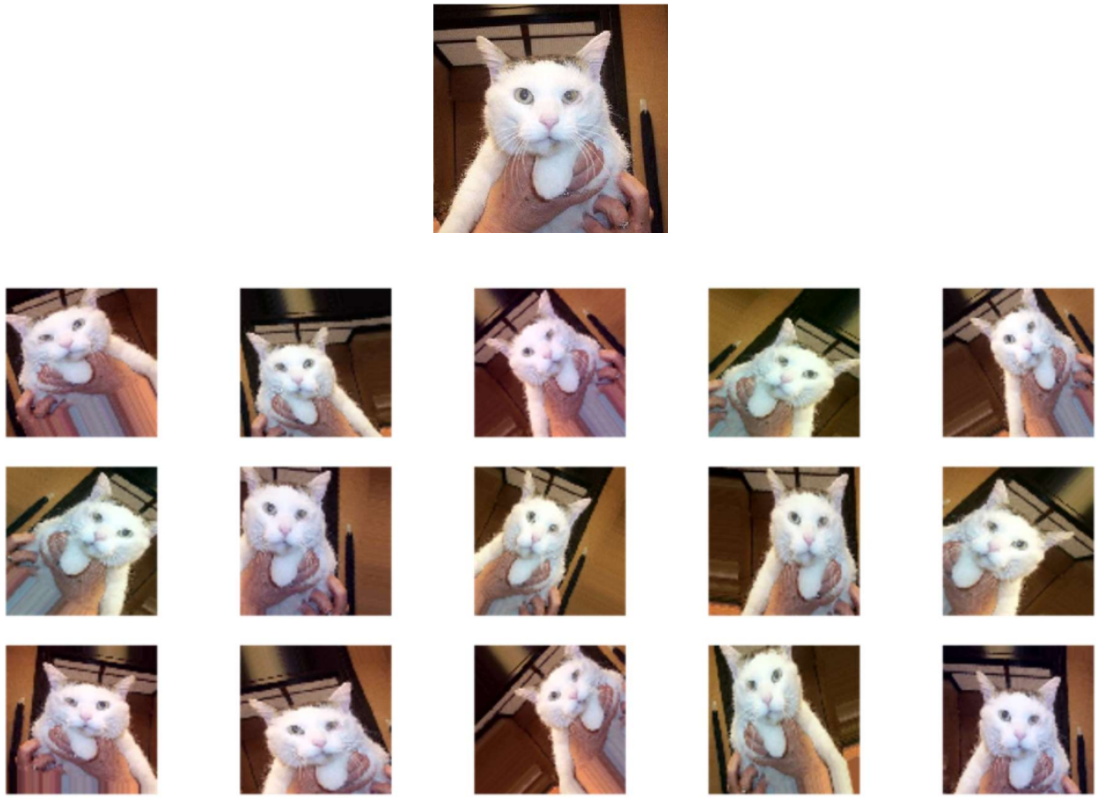
# More generalization ideas: data augmentation 数据增强



# More generalization ideas: data augmentation 数据增强



# More generalization ideas: data augmentation 数据增强



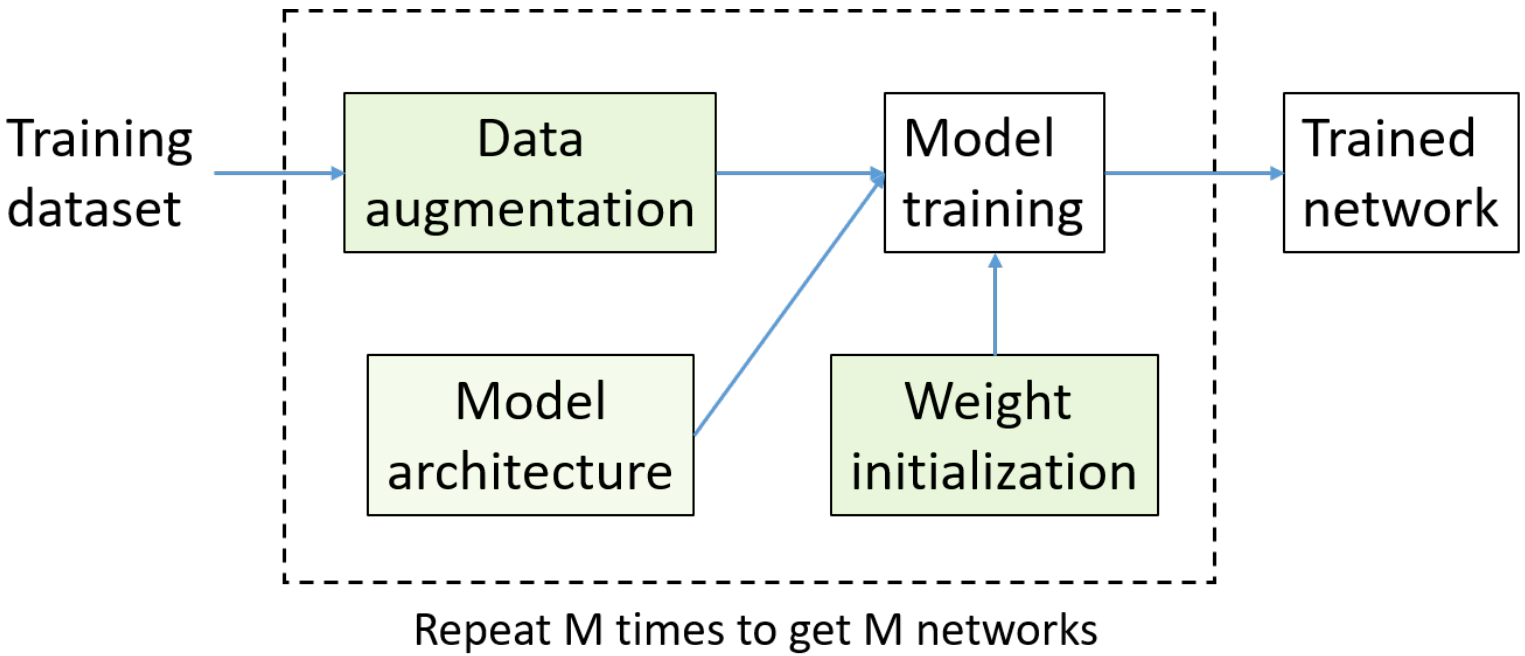
- Augmentation ways: rotate, scale, translate, flip, shear, deform, color and illumination change, etc (旋转、缩放、平移、翻转、剪切、变形、颜色和照明更改)
- Data augmentation produces more training data

# Ensemble model 集成模型

- Use a **group** of models (experts) to predict result!
- First, train multiple slightly different networks

# Ensemble model 集成模型

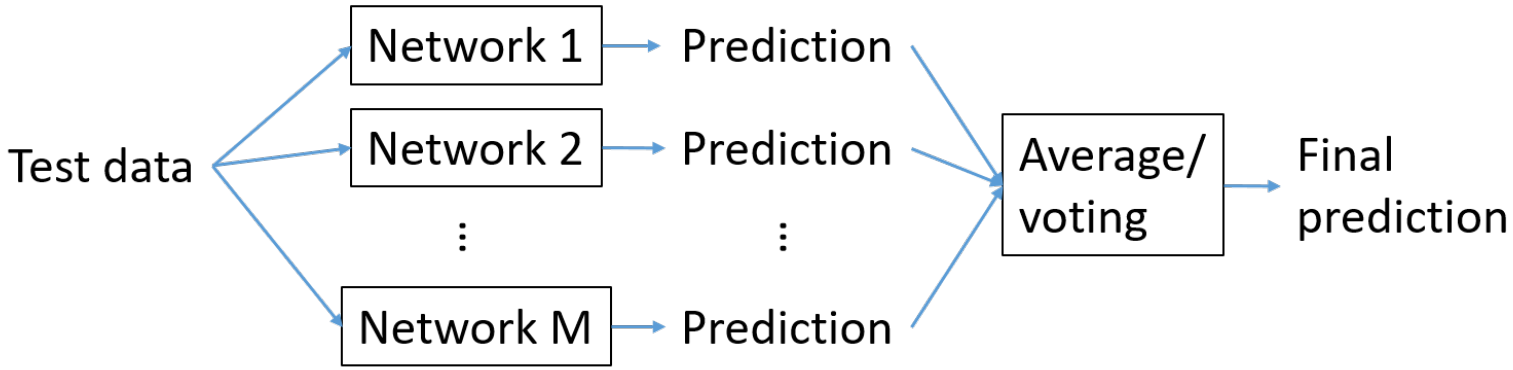
- Use a **group** of models (experts) to predict result!
- First, train multiple slightly different networks




- Networks are different due to different weight initialization, augmented data, and possibly different model architectures.

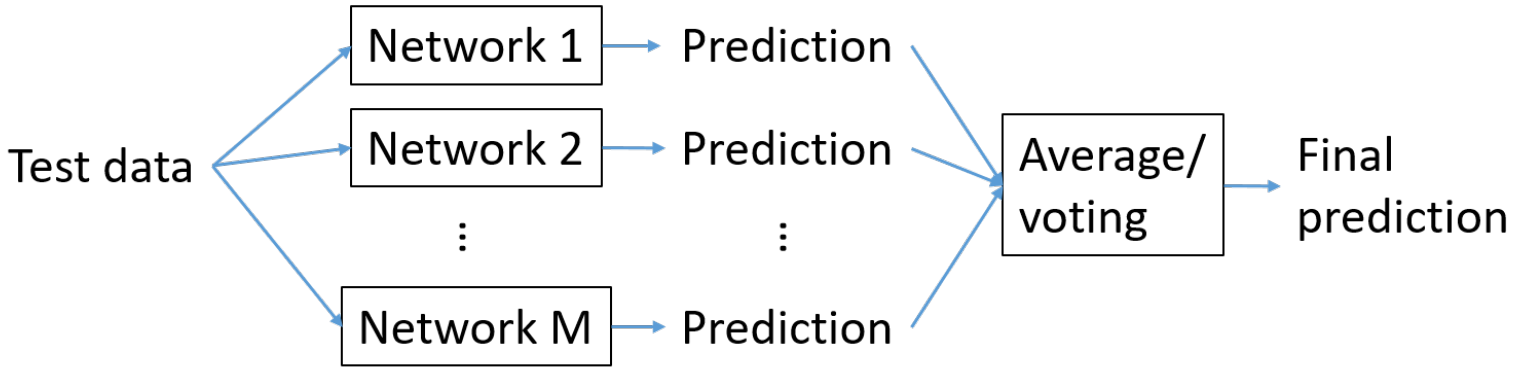
# Ensemble model (cont')

- Then, collect predictions of all experts for final prediction



# Ensemble model (cont')

- Then, collect predictions of all experts for final prediction




- Ensemble model generalizes better (lower test error)

# In-class Quiz

- 有一个图像分类任务，共有 4 个类别（tree, animal, person, bicycle），某个输入样本的类别是 person, 如果表达该类别标签？结果是什么？
- 为什么要进行学习率衰减（Decaying learning rate）？常见的方法有哪些？
- 弹性传播（Rprop）和动量方法（Momentum Methods）提出的动机是什么？简述弹性传播和动量方法的思想。
- 为什么要进行协变量偏移（Covariate Shifts）？简述批归一化（Batch Normalization）步骤。
- 什么是过拟合（overfitting）？解决过拟合的方法有哪些？

Thank you!