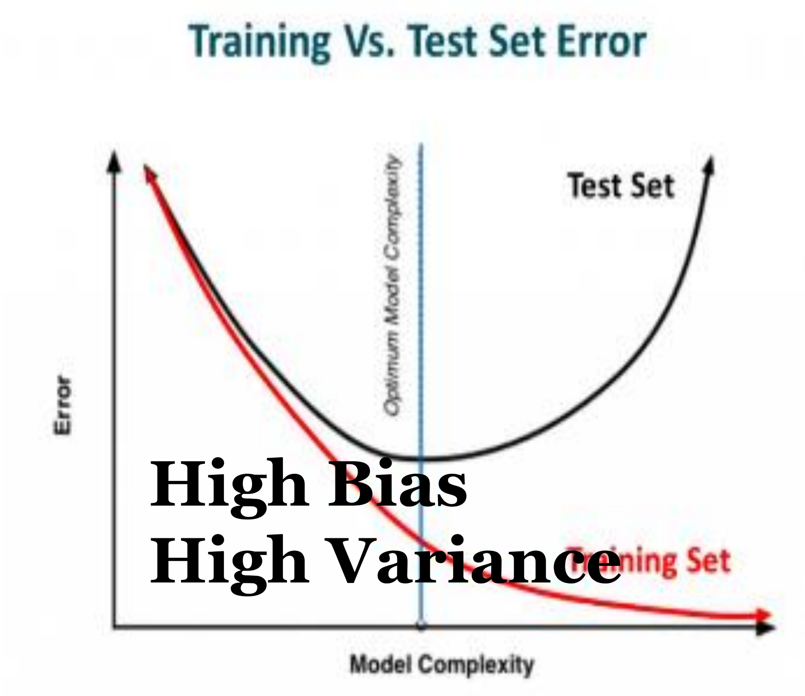
Deep Learning Regularization and Optimization

Deep Learning
E. Fatemizadeh Fall 2021

Regularization

› From Previous Discussion:
  – The Bias-Variance Tradeoff (Dilemma)



Regularization

› The Bias-Variance Tradeoff (Dilemma)

$$y = f(x) + \varepsilon$$

$$\begin{aligned}
\mathrm{E}\left[(y-\hat{f})^2\right] &= \mathrm{E}\left[(f+\varepsilon-\hat{f})^2\right] \\
&= \mathrm{E}\left[(f+\varepsilon-\hat{f}+\mathrm{E}[\hat{f}]-\mathrm{E}[\hat{f}])^2\right] \\
&= (f-\mathrm{E}[\hat{f}])^2+\mathrm{Var}[y]+\mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2+\mathrm{Var}[y]+\mathrm{Var}\left[\hat{f}\right] \\
&= \mathrm{Bias}[\hat{f}]^2+\sigma^2+\mathrm{Var}\left[\hat{f}\right]
\end{aligned}$$

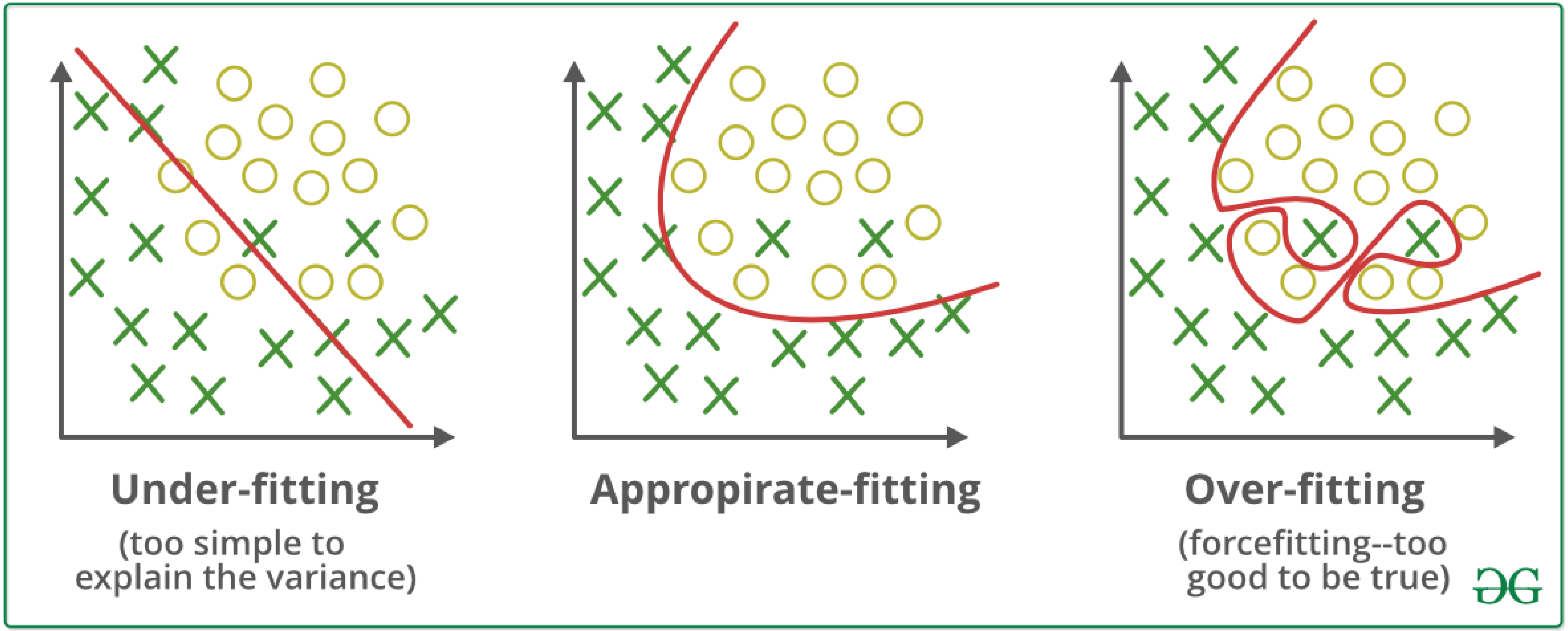
Lower Bound on the Expected Error on Unseen/Validation Samples

Regularization

- The Bias-Variance Tradeoff (Dilemma)
- Simple models trained on different samples of data **do not differ much** from each other. However, far from the true (underfitting).
- On the other hand, complex models trained on different subsets of data **are very different** from each other (overfitting).
- Simple model: high bias, low variance
- Complex model: low bias, high variance

Regularization

› Motivation:
  – Overfitting Challenge



Regularization

› Overfitting Challenge (What to do?)

Training Error	**Valid Error**	**Cause**	**Solution**
High	High	High bias	- Increase model complexity - Train for more epochs
Low	High	High variance	- Add more training data (e.g., **dataset augmentation**) - Use **regularization** - User **early stopping** (train less)
Low	Low	Perfect tradeoff	- You are done!

Regularization

- Definition:

*“**Any** modification we make to a learning algorithm that is intended to **reduce** its **generalization error** but not its **training error**.”*

- *Solution:*
  - *Parameter (weights) Penalties*
  - *Dataset Augmentation*
  - *Noise Robustness (input/output)*
  - *Early Stopping*
  - *Bagging and Other Ensemble Methods*
  - *Dropout*

Parameter Penalties

- Instead *empirical risk minimization*:

$$\text{minimize}\left(\text{Loss}\left(\text{Data}\middle|\text{Model}\right)\right)$$

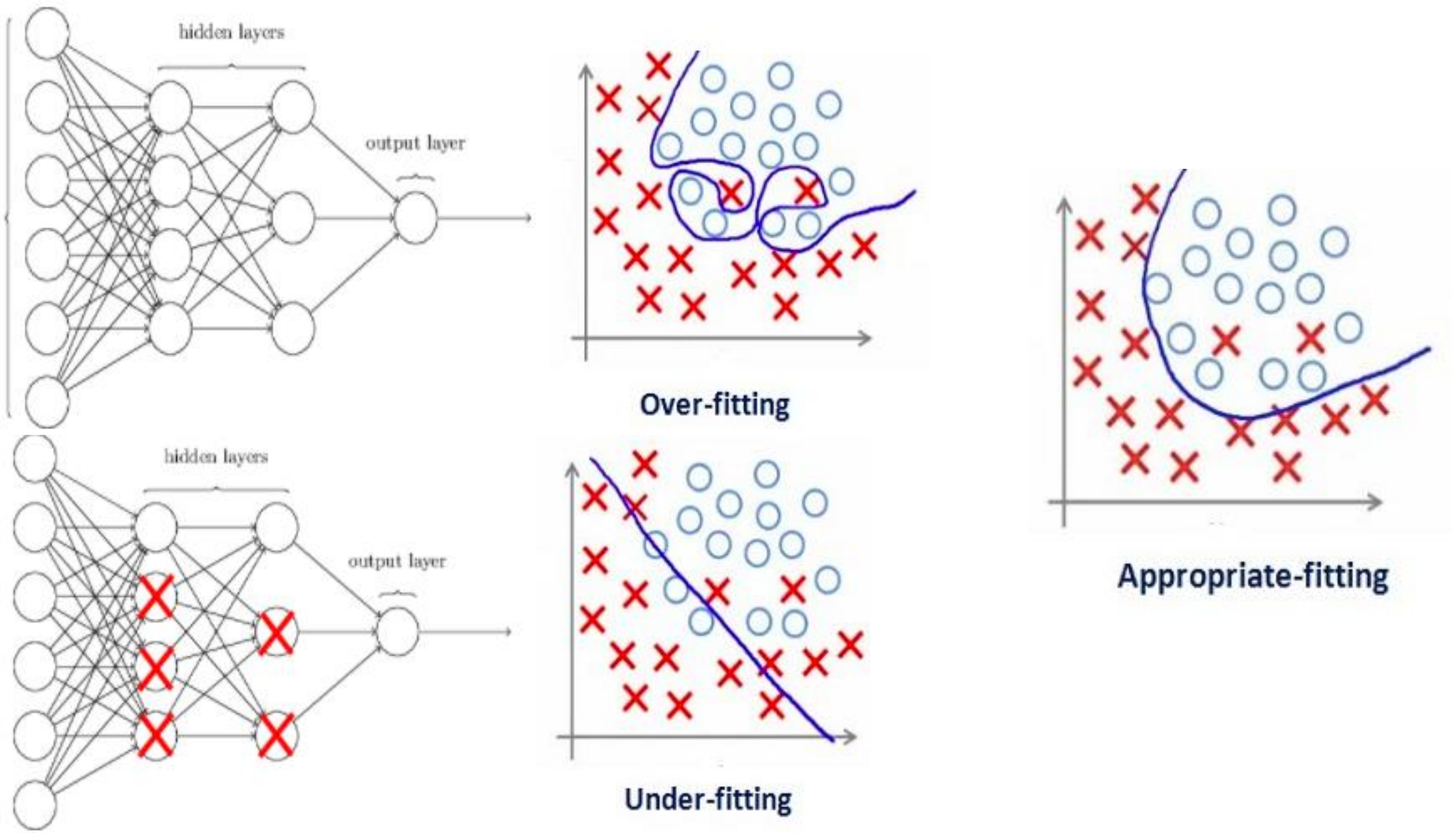
- Try *structural risk minimization:*

$$\text{minimize}\left(\text{Loss}\left(\text{Data}\middle|\text{Model}\right)+\lambda\text{Complexity}\left(\text{Model}\right)\right)$$

- Two major policies for model complexity::
  - A function of the weights of all the features in the model.
  - A function of the total number of features with nonzero weights.

Parameter Penalties

› How it works:



Regularization by Parameters Penalties

- General Formulation:

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

- Typically Not applied on ***Bias*** terms (why)?

L^2 Regularization

- An old fashion technique!
- Idea is **weight decay**
- Known as:
  - Ridge Regression
  - Tikhonov Regularization

$$J_{L_2} = (MSE/CE) + \lambda \sum_{Weight \backslash Bias} \|w\|_2^2$$

or

$$J_{L_2} = (MSE/CE) + \sum_{Layers} \lambda_{Layers} \sum_{Weight \backslash Bias} \|w\|_2^2$$

L^2 Regularization – Deep Learning, Goodfellow 2016

› Formulation:

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}), \tag{7.2}$$

with the corresponding parameter gradient

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}). \tag{7.3}$$

To take a single gradient step to update the weights, we perform this update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})\right). \tag{7.4}$$

Written another way, the update is:

$$\boldsymbol{w} \leftarrow (1-\epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}). \tag{7.5}$$

L^2 Regularization - Analysis

- If we know exact solution:

$$w^* = \arg\min_{w} J(w).$$

- 2^{nd} order Approximation:

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

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

- With L^2 term:

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

$$(H + \alpha I)\tilde{w} = Hw^*$$

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

L^2 Regularization - Analysis

- For Convex Loss function, H, is pdm (Unitary **Q** and positive **Λ**) :

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

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

L^2 Regularization - Analysis

- For Convex Loss function, H, is pdm, and :

$$(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \Longleftrightarrow \frac{\lambda_i}{\lambda_i + \alpha}$$

- Numerical Examples:

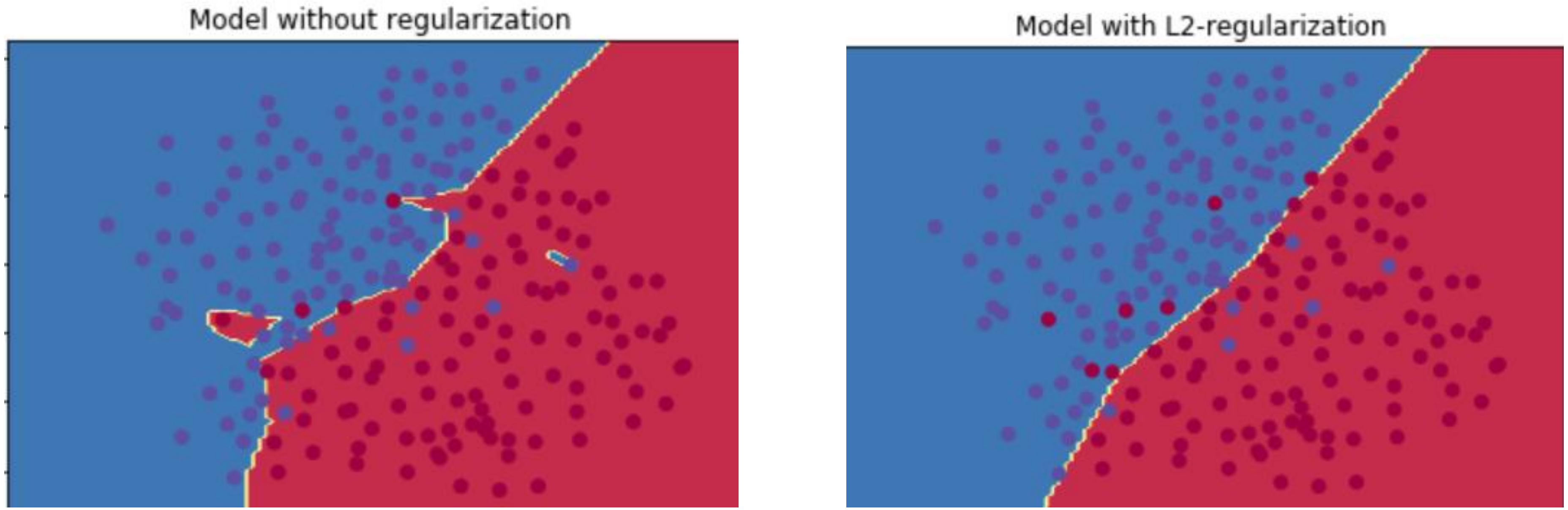
$$H = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \Rightarrow Q\Lambda^{-1}\Lambda Q^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Q(\Lambda + 2I)^{-1} \Lambda Q^T = \begin{bmatrix} 0.46 & -0.14 & -0.04 \\ -0.14 & 0.43 & -0.14 \\ -0.04 & -0.14 & 0.46 \end{bmatrix}$$

L^2 Regularization - Analysis

- For Convex Loss function, H, is pdm, and :

$$(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \Longleftrightarrow \frac{\lambda_i}{\lambda_i + \alpha}$$



L^2 Regularization - Analysis

› In Linear Regression:

$$(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}). \tag{7.14}$$

When we add L^2 regularization, the objective function changes to

$$(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \frac{1}{2}\alpha\boldsymbol{w}^\top\boldsymbol{w}. \tag{7.15}$$

This changes the normal equations for the solution from

$$\boldsymbol{w} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \tag{7.16}$$

to

$$\boldsymbol{w} = (\boldsymbol{X}^\top\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{7.17}$$

L¹ Regularization

› Idea is **weight Sparsification:**

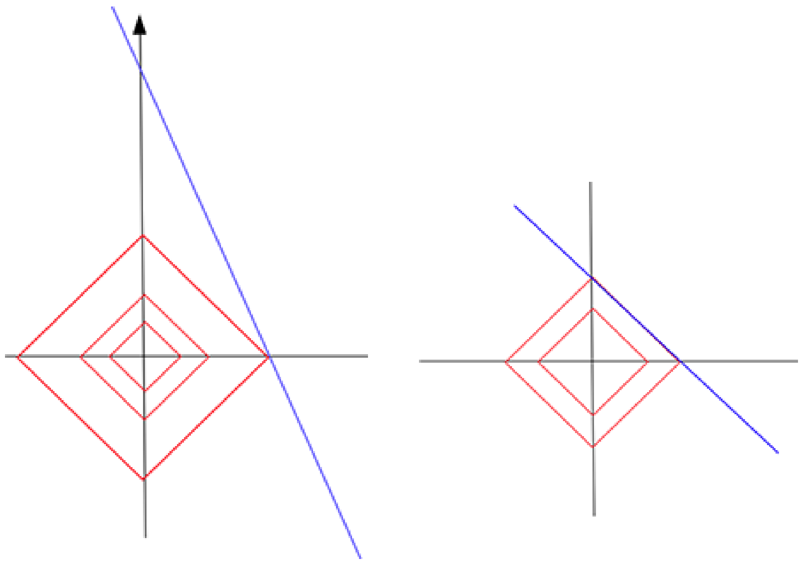
$$\tilde{J}(w; \boldsymbol{X}, \boldsymbol{y}) = \alpha ||\boldsymbol{w}||_1 + J(w; \boldsymbol{X}, \boldsymbol{y}),$$

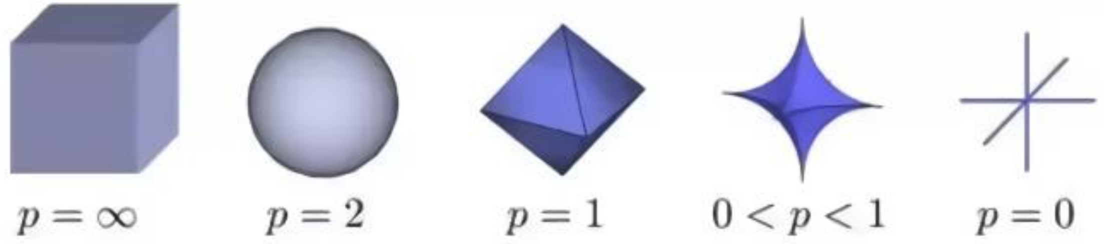
$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

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

L^1 Regularization

- Why Sparsify?
  - Suppose we want solve AX=b with minimizing L_1
    - May have infinite solution!!!





L^1 Regularization

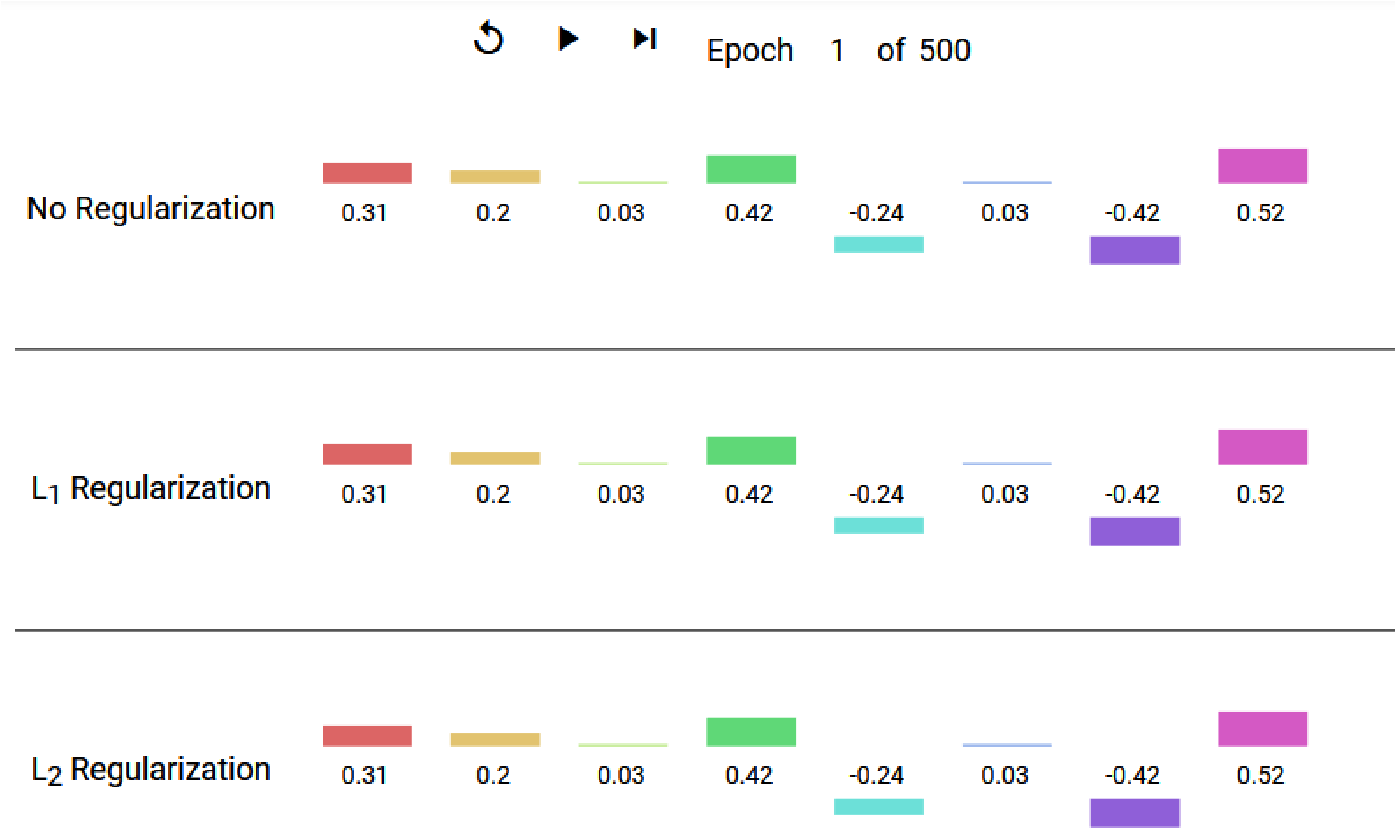
- For diagonal Hessian Matrix:

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

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

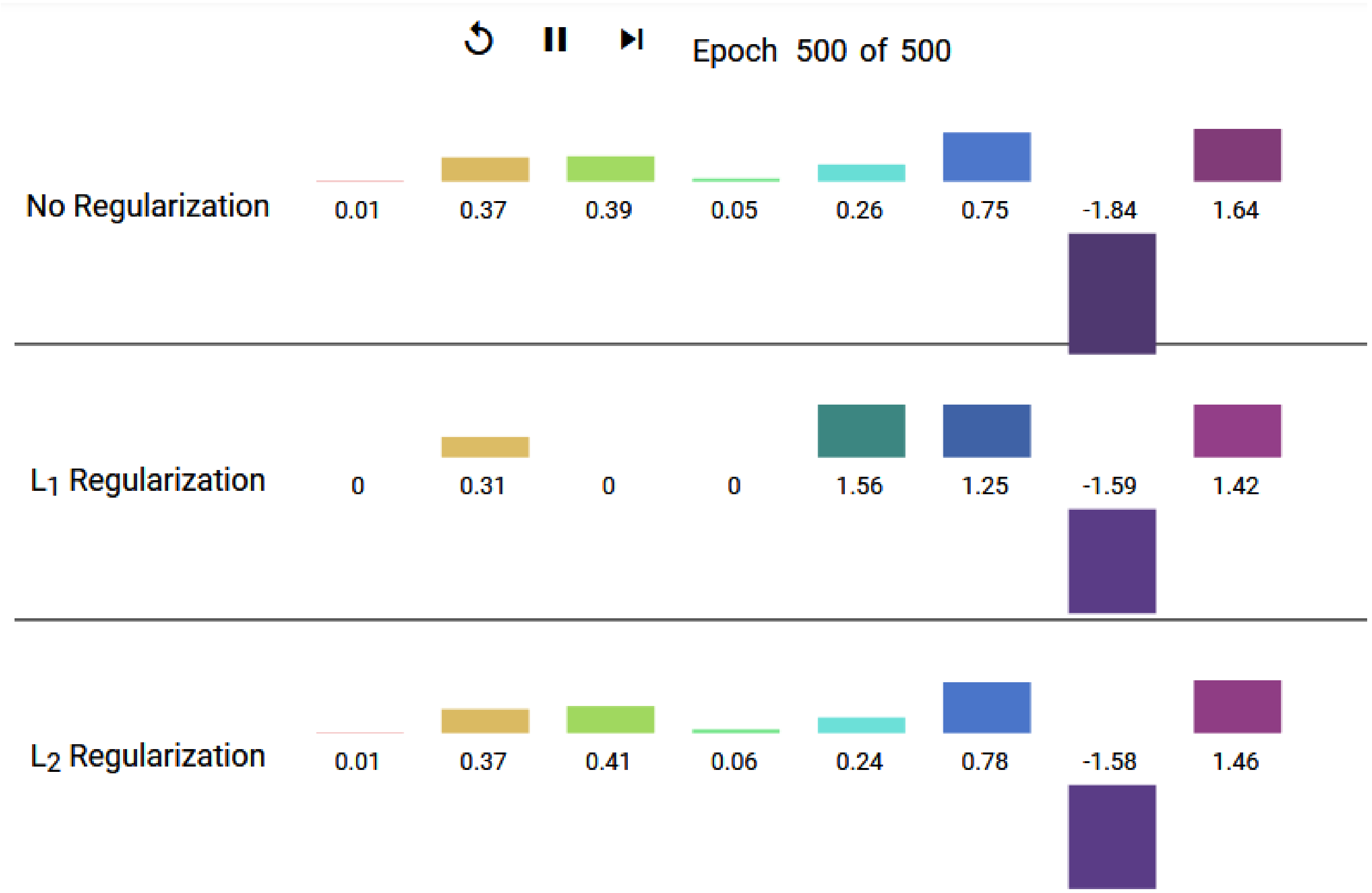
Regularization Effect

https://developers.google.com/machine-learning/crash-course/



Regularization Effect

https://developers.google.com/machine-learning/crash-course/



Elastic Net Regularization - Regularization and Variable Selection via the Elastic Net, Zou and Hastie, 2004

- L_1+L_2 Regularization:

$$J_{Elastic} = \left(MSE/CE\right) + \lambda\left(\alpha \sum_{Weight\backslash Bias} \|w\|_2^2 + (1-\alpha) \sum_{Weight\backslash Bias} \|w\|_1\right)$$

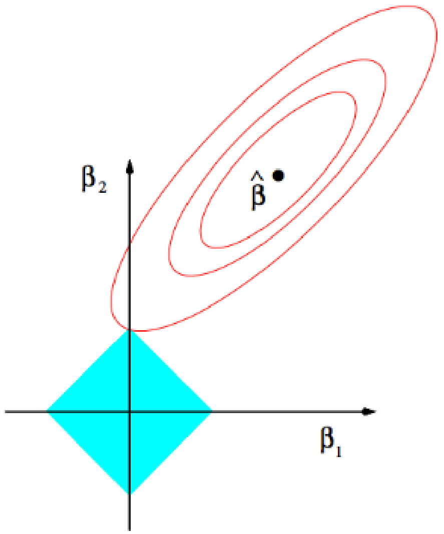
Geometric Perspective

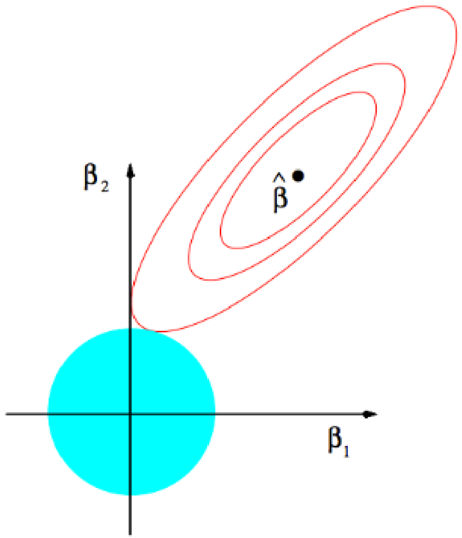
- The Problems:

$$\text{minimize}\left(\text{Loss}\left(\text{Data}\middle|\text{Model}\right)+\lambda\text{Complexity}\left(\text{Model}\right)\right)$$

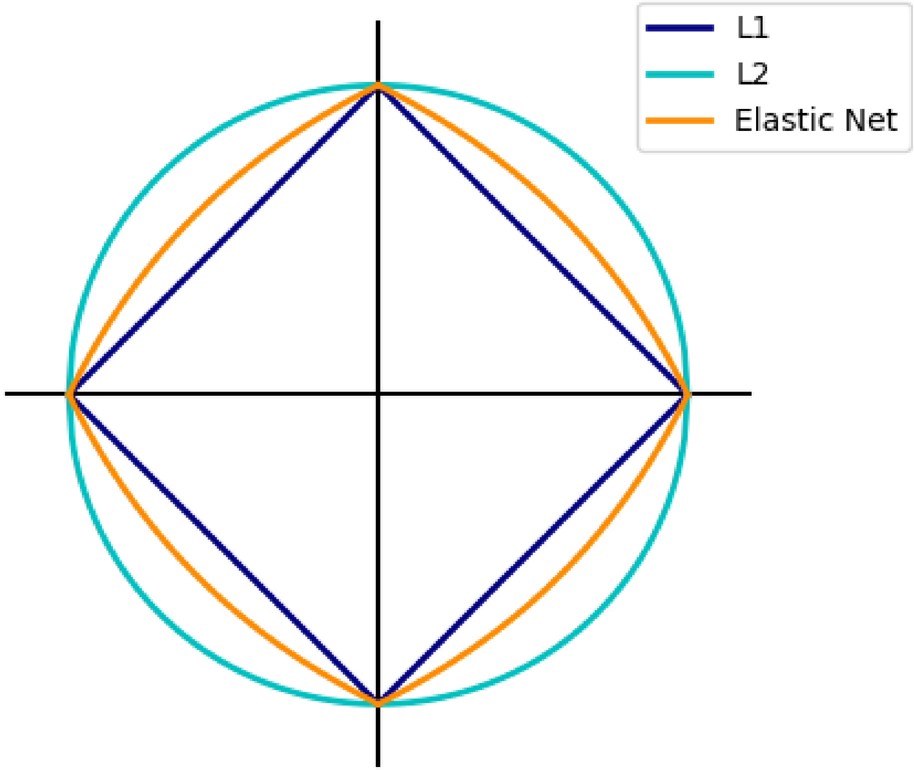
- **Is equivalent to:**

$$\text{minimize}\left(\text{Loss}\left(\text{Data}\middle|\text{Model}\right),\quad \text{Complexity}\left(\text{Model}\right)\right)\le\eta$$



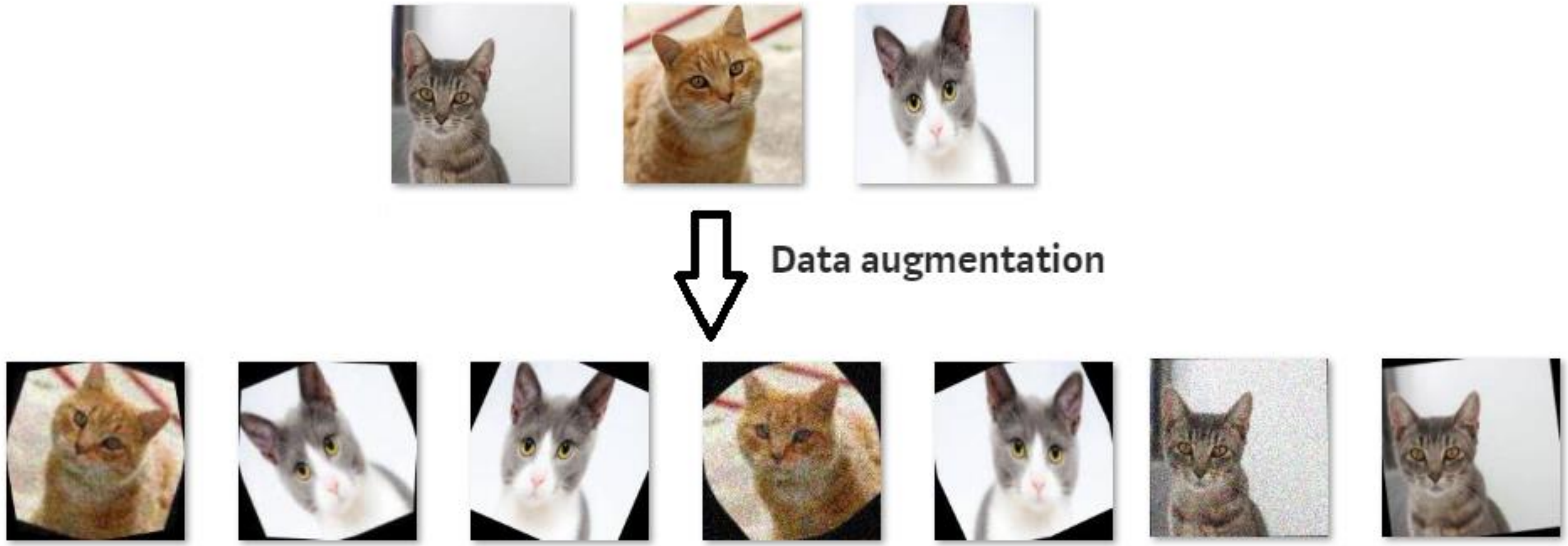


Geometric Perspective



Data Augmentation

- Idea:
  - Invariance (Rotation, Scaling, Translation, Shearing, Mirroring, …) and Regularization



Data Augmentation

- Standard Transform
  - Horizontal/Vertical Flip (Mirroring)
  - Horizontal/Vertical Translation
  - Horizontal/Vertical Scale
  - Rotation
  - Shearing (Skewing)
  - Contrast/Brightness/illumination/Color/… Change
  - Random Crop and then Scale
  - Random Cutout (Blackout a random rectangular/square)
  - Additive/Multiplicative Noise
- Modern Approach:
  - Deep Generated Data, Augmented Network (DeepFake, …)

Regularization – Dropout - Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava, 2014

- Preliminary:
  - Ensemble Methods: Learn multiple models (Weak Classifier) to solve the same problem and then combine (aggregate) them for better results
  - Combination of "**Different/Same**" "**Model/Dataset/Features/Objective/** Optimization/Initials"
- Policies:
  - Bagging
  - Boosting
  - Stacking

Regularization – Dropout

- Bagging (Bootstrap aggregating):
  1. *K* new training datasets are generated by uniformly random sampling with **replacement (Bootstrapping)** from the original dataset.
  2. Train K model independently
  3. Combine/Aggregate the classifier/regressor using:
     1. Averaging (Arithmetic, Geometric, …)
     2. Voting
- Boosting: Each data and classifier has its own weight (determined with boosting procedure), AdaBoost is most known methods.
- Stacking: Train K models using all data, then train a combiner/aggregator classifier to combine K result, as best as possible.

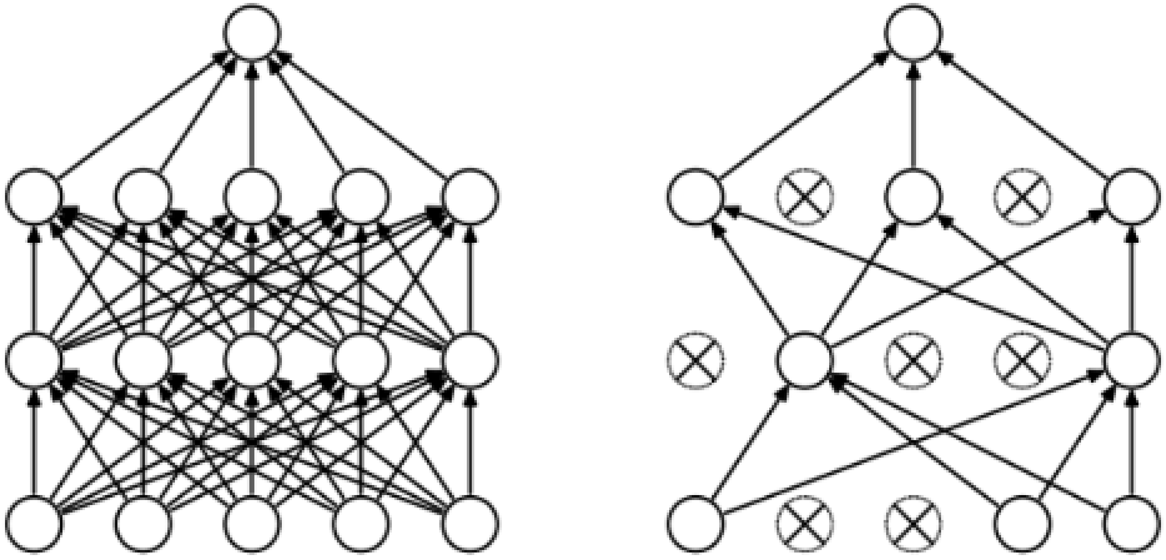
Regularization – Dropout

- Motivation:
  - Co-Adaptation:

    If two or more neurons extract the same feature repeatedly or highly correlated behavior (co-adaptation), the network isn't reaching its full capability.

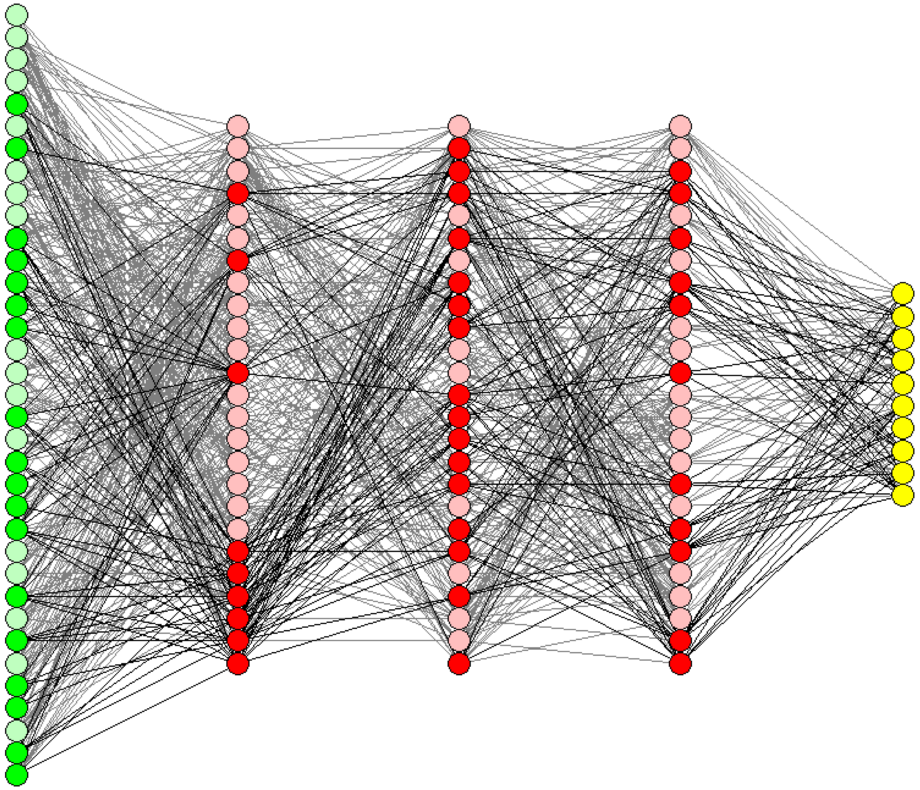
Regularization – Dropout

› Solution: Update (EBP) a fraction (Randomly Chosen) of all weights is each iteration!



(a) Standard Neural Net (b) After applying dropout.

Regularization – Dropout

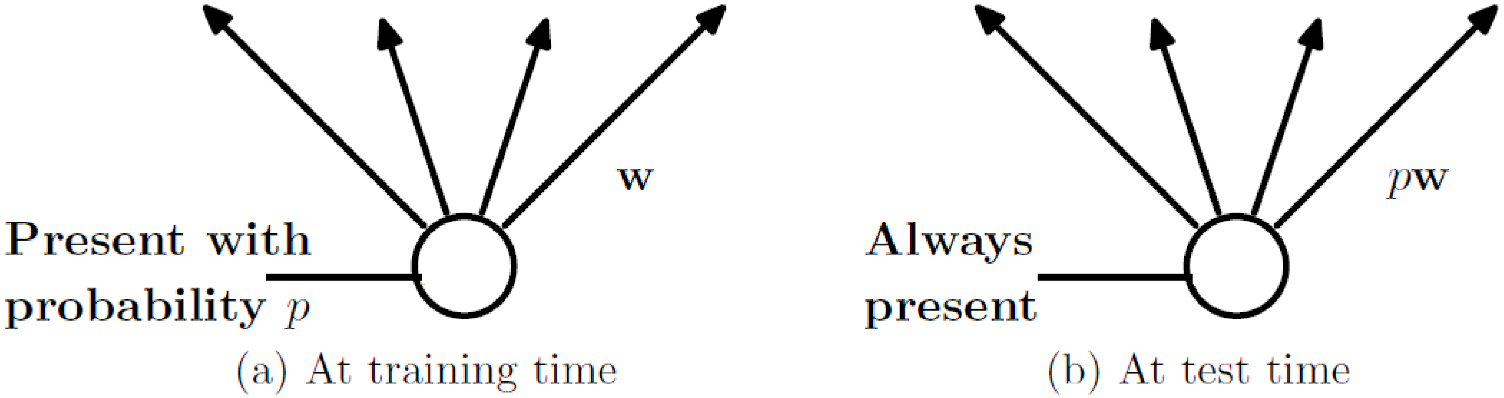


Regularization – Dropout

- Training Phase:
  - For each hidden layer, for each training sample, for each iteration, dropout (zero out) a random fraction, (1-*p*), of neuron (and corresponding weight).
    - Input Layer: P>0.8 or P=1.0
    - Hidden Layer: P=0.5
    - Output Layer: P=1.0
- Test Phase:
  - Use all activations, but reduce them by a factor *(1-p)* (to account for the missing activations during training).

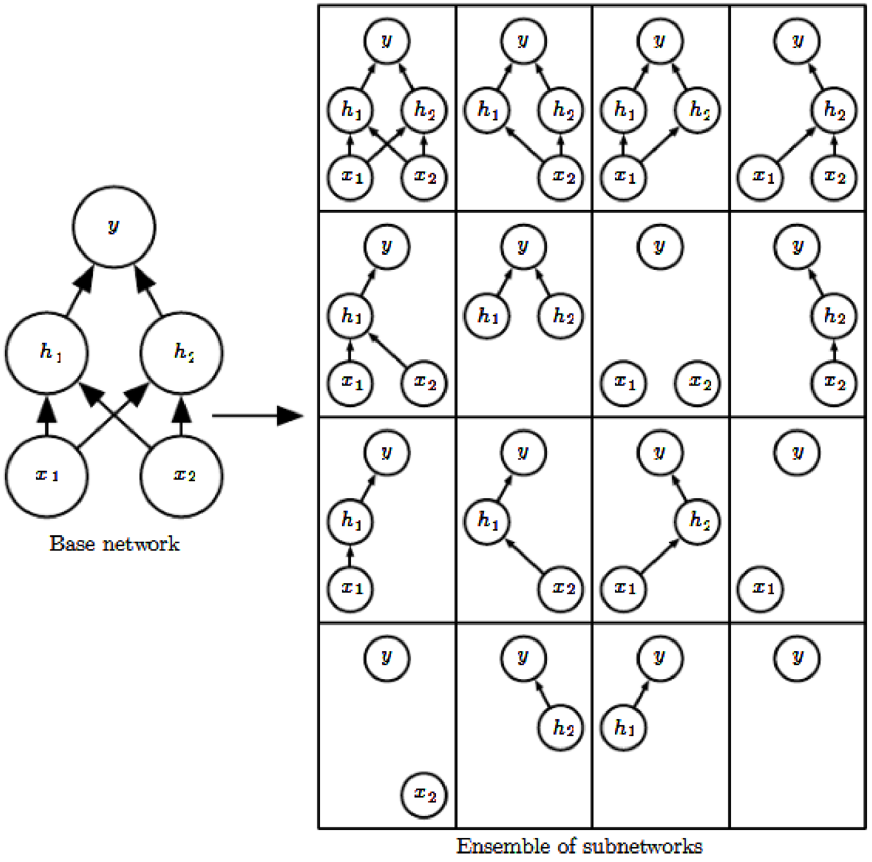
Regularization – Dropout

› Dropout Training/Test Scheme



Regularization – Dropout

› Dropout Training Scheme



Regularization – Dropout. Understanding Dropout, Baldi, NIPS2013

- Mathematics of Dropout (N: Complete NN, D: Dropout NN)
- Dropping out is modelled by Bernoulli distribution:

$$\delta_{\mathrm{i}} \sim \mathrm{Bernoulli(p)}$$

$$E_N = \frac{1}{2}\left(t - \sum_{i=1}^{n} w_i' I_i\right)^2$$

$$E_N = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_i w_i I_i\right)^2$$

$$E_D = \frac{1}{2}\left(t - \sum_{i=1}^{n} \delta_i w_i I_i\right)^2$$

Regularization – Dropout. Understanding Dropout, Baldi, NIPS2013

› Mathematics of Dropout (N: Complete NN, D: Dropout NN)

$$\frac{\partial E_D}{\partial w_i} = -t\delta_i I_i + w_i \delta_i^2 I_i^2 + \sum_{j=1, j\neq i}^{n} w_j \delta_i \delta_j I_i I_j$$

$$\frac{\partial E_N}{\partial w_i} = -t p_i I_i + w_i p_i^2 I_i^2 + \sum_{j=1, j\neq i}^{n} w_j p_i p_j I_i I_j$$

$$\begin{aligned} E\left[\frac{\partial E_D}{\partial w_i}\right] &= -t p_i I_i + w_i p_i^2 I_i^2 + w_i Var(\delta_i) I_i^2 + \sum_{j=1, j\neq i}^{n} w_j p_i p_j I_i I_j \\ &= \frac{\partial E_N}{\partial w_i} + w_i Var(\delta_i) I_i^2 \\ &= \frac{\partial E_N}{\partial w_i} + \boxed{w_i p_i (1-p_i) I_i^2} \end{aligned}$$

Regularization – Dropout. Understanding Dropout, Baldi, NIPS2013

› It is Like Regularization

$$
\begin{aligned}
E\left[\frac{\partial E_D}{\partial w_i}\right] &= -tp_iI_i + w_ip_i^2I_i^2 + w_iVar(\delta_i)I_i^2 + \sum_{j=1,j\neq i}^{n} w_jp_ip_jI_iI_j \\
&= \frac{\partial E_N}{\partial w_i} + w_iVar(\delta_i)I_i^2 \\
&= \frac{\partial E_N}{\partial w_i} + \boxed{w_ip_i(1-p_i)I_i^2}
\end{aligned}
$$

$$
E_R = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_iw_iI_i\right)^2 + \sum_{i=1}^{n} p_i(1-p_i)w_i^2I_i^2
$$

› Why p=0.5 is recommended?

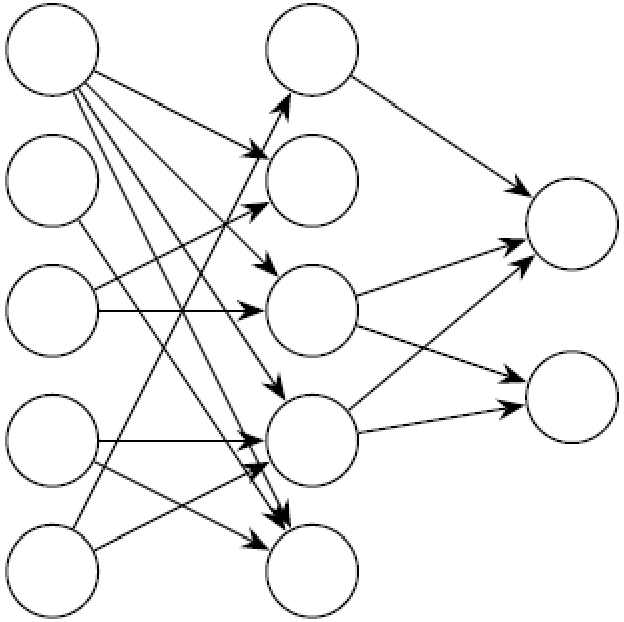
Regularization – Dropout

› Some Variations:

– Gaussian Dropout Multiplicative: $\delta_i \sim N(1, \sigma^2)$

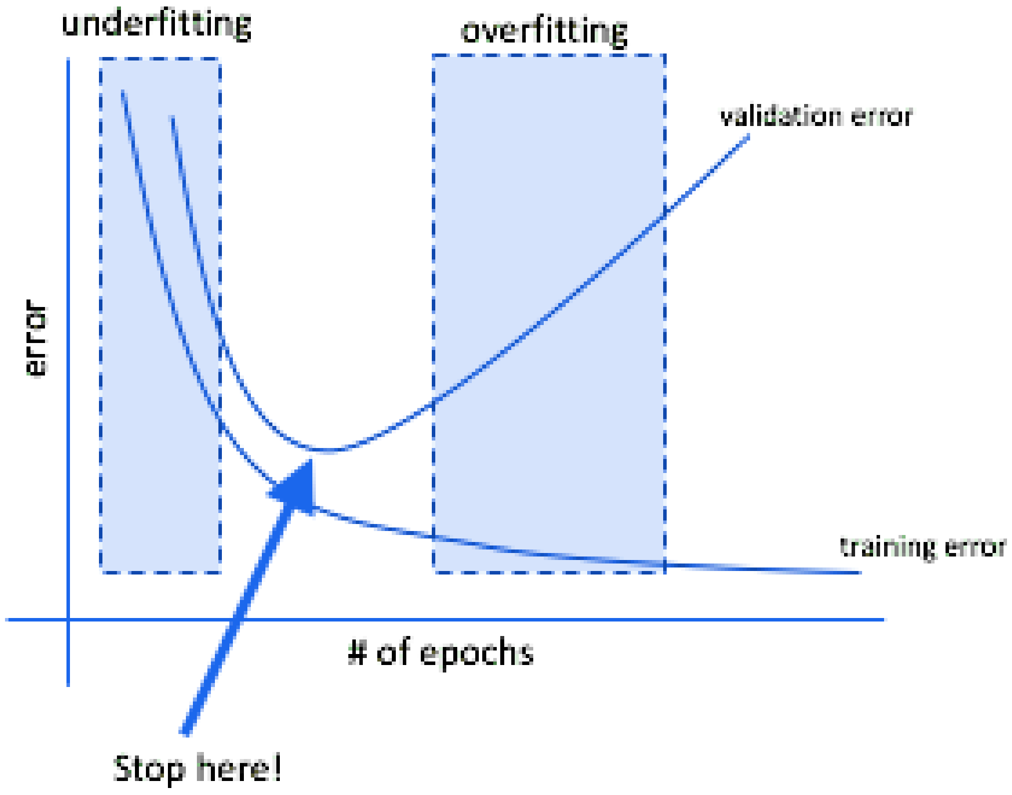
– Gaussian Dropout Additive: $\delta_i \sim N(0, \sigma^2)$

– DropConnect:



Regularization - Early Stopping

› Again Validation Error Curve!



Regularization - Early Stopping Meta Algorithm

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^*.

Regularization - Early Stopping How to use data #1!

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.

Regularization - Early Stopping How to use data #2!

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

Regularization - Early Stopping Theory

- Why it Early Stopping acts as regulator?
- Taylor approximation around true solution (w^*), and convex assumption!

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

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

- Gradient Descent, starting from ($w^{(0)}=\mathbf{0}$, τ is iteration counter)

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

Regularization - Early Stopping Theory

- Eigen decomposition of **H**:

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

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

- 1st order DE, starting $w^{(0)}=\mathbf{0}$, and small step size ($|1-\varepsilon\lambda_i| < 1$)

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

- Remember similar analysis in $\boldsymbol{L_2}$ Regularization (Slide #14):

$$\frac{\lambda_i}{\lambda_i+\alpha} = 1 - \frac{\alpha}{\lambda_i+\alpha} \quad \Longrightarrow \quad \begin{aligned} \boldsymbol{Q}^\top\tilde{\boldsymbol{w}} &= (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*, \\ \boldsymbol{Q}^\top\tilde{\boldsymbol{w}} &= [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^\top\boldsymbol{w}^*. \end{aligned}$$

Regularization - Early Stopping Theory

- Compare Two approaches:

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

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

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

- Solve for small ε and Large α (Regularization factor):
- $\mathbf{I} - \varepsilon\tau\Lambda \approx (-\Lambda + \alpha^{-1}\mathbf{I})\alpha = \mathbf{I} - \alpha\,\Lambda \rightarrow \alpha \approx \varepsilon\tau$

Regularization - Others

- Noise Robustness (adding noise to input/output)
- Multitasking Learning:
  - Learn two or more tasks with a layer of shared parameters
- Parameter Tying:
  - Two Model performs same classification task but different input pdfs, we consider a penalty as:

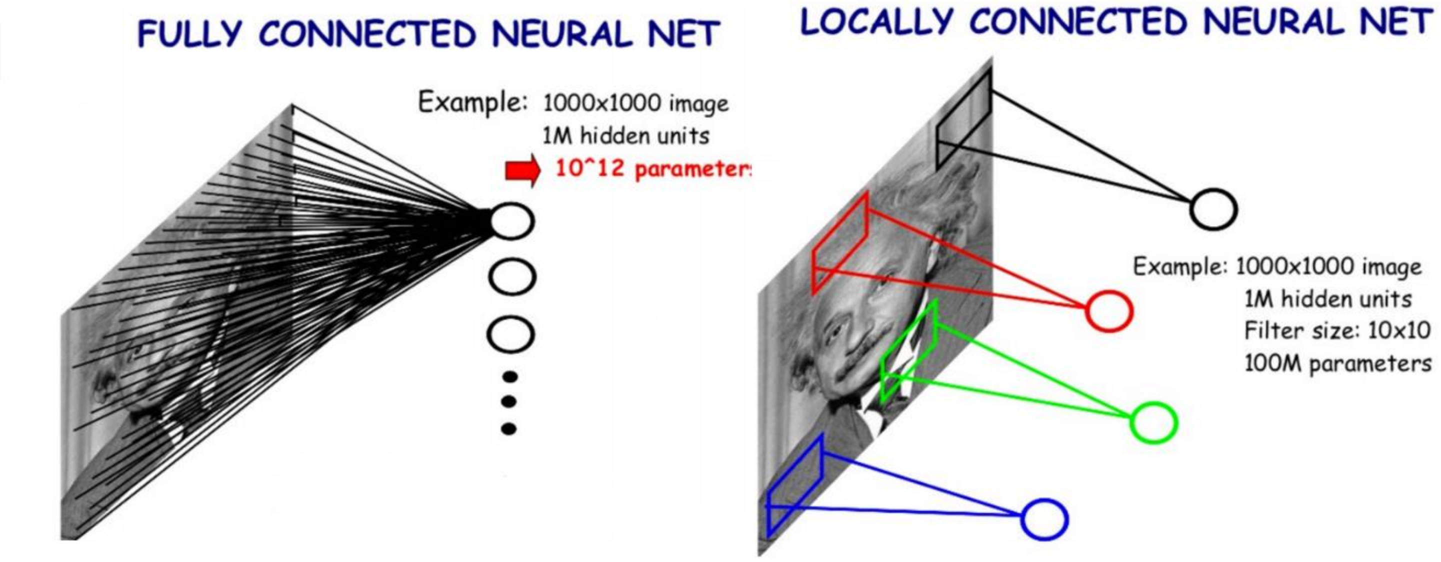
$$\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$$
$$\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$$

$$\boxed{\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = ||\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}||_2^{\,2}}$$

Regularization - Others

› Parameter Sharing:
  – Force two set of weights is equals

› Sparse Parameters:
  – Force some weights equal to zero



Optimization Challenges

- Local minimum and saddle points:
  - For many high-dimensional non-convex functions (and random function) Local minimum/maximum are **rare** compared to saddle points. (#Saddles/#Locals grows exponentially with R^D)
  - In local minimum, eigenvalues of Hessian are Positive
  - In Saddle Points, eigenvalues of Hessian may be Positive or Negative
- Ill-Conditioning:
  - Hessian term may exceeds gradient term

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2} \epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}.$$

Optimization Challenges

- Vanishing/Exploding Gradient:
  - Suppose a simple Computational Graph (Back Propagation) in which wew repeatedly multiplying by a matrix **W**

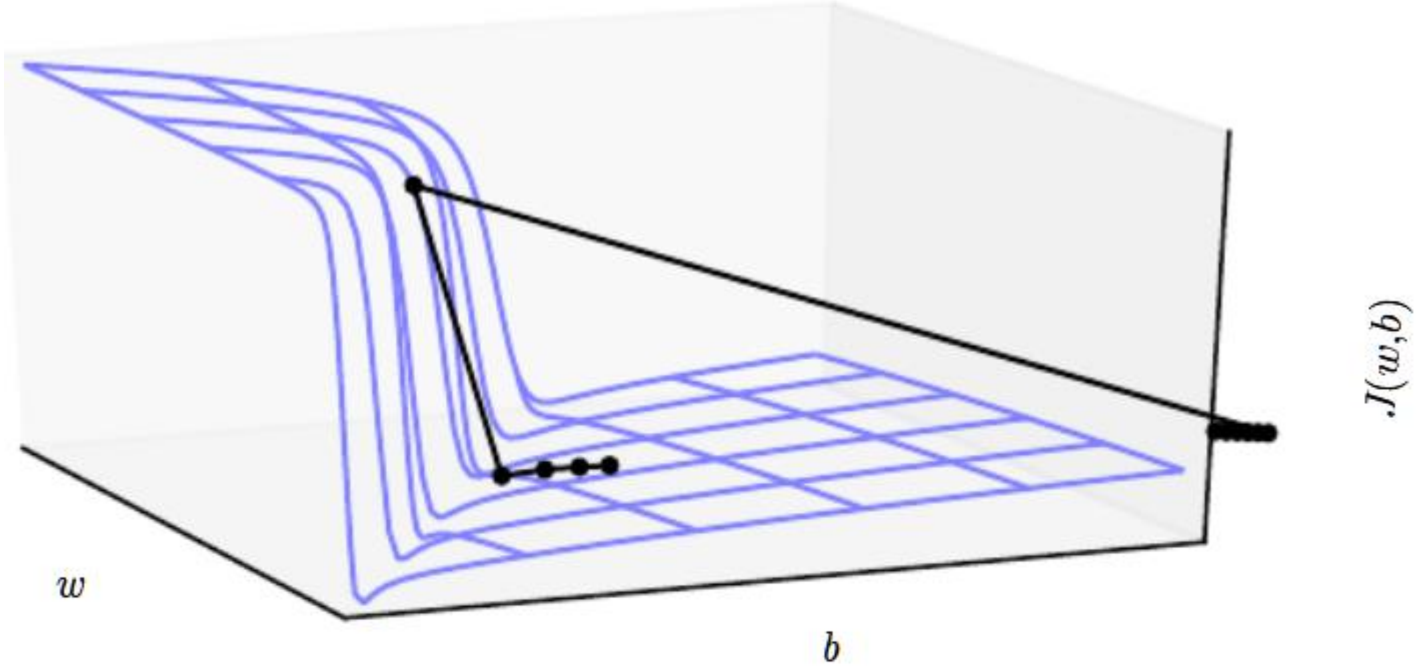
$$\boldsymbol{W} = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}.$$

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}.$$

- Vanishing (λ_i<1) or Exploding (λ_i>1)
  - Vanishing: which direction is?
  - Exploding: Cliff Structure

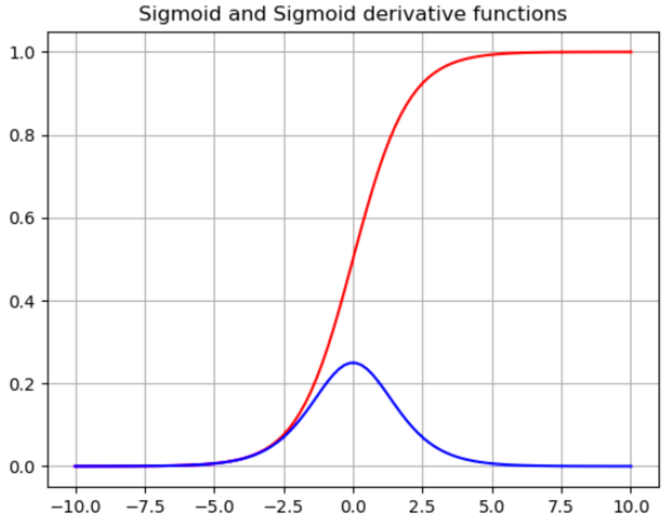
Optimization Challenges

› Exploding Gradient in Cliff Structure



Optimization Challenges

- In Deep Neural Network:



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

- Chaotic Behavior in Nonlinear Function!
- ReLU is a good choice!
- Gradient Clipping ($\alpha \frac{g}{\|g\|}$)

Stochastic Gradient Descent (SGD)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$

while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
end while

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{8.14}$$

with $\alpha = \frac{k}{\tau}$. After iteration τ, it is common to leave ϵ constant.

Momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{8.15}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{8.16}$$

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Nesterov Momentum

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right], \tag{8.21}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{8.22}$$

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Algorithms with Adaptive Learning Rates

- Idea:

$$\theta_{k+1} = \theta_k - \eta H_k^{-1} \nabla J(\theta_k)$$

$$\vdots$$

$$\theta_{k+1} = \theta_k - \eta B^{-1} \nabla J(\theta_k), \text{ B: Preconditioner}$$

$$\vdots$$

$$B_k = diag\left(\sum_{t=1}^{k} \left[\nabla J(\theta_t)\right]\left[\nabla J(\theta_t)\right]^T\right)^{0.5}$$

$$\theta_{k+1} = \theta_k - \eta B_k^{-1} \nabla J(\theta_k) \Rightarrow \theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{B_k + \varepsilon}} \odot \nabla J(\theta_k)$$

Algorithms with Adaptive Learning Rates

› AdaGrad

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Algorithms with Adaptive Learning Rates

› RMSProp

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

Algorithms with Adaptive Learning Rates

› RMSProp algorithm with Nesterov momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

ADAM (Adaptive Moment Estimation)

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
  Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

ADAM Variation

- ADAMax:

$$r_t = \rho r_{t-1} + (1-\rho)|g_t|^2$$
$$\vdots$$
$$r_t = \rho^p r_{t-1} + (1-\rho^p)|g_t|^p$$
$$p \to \infty$$
$$r_t = \max\{\rho r_{t-1}, |g_t|\}$$

- Nesterov-accelerated Adaptive Moment Estimation (NADAM)

2nd Order Methods

› Newton's Method Idea:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0), \tag{8.26}$$

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) \tag{8.27}$$

Newton Methods for Convex (pdm H) Loss!

Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$.

Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of m examples
  while stopping criterion not met **do**
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute Hessian inverse: $\boldsymbol{H}^{-1}$
    Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Newton Methods for Non pdm H

- Regularization (Positive α)

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \left[H\left(f(\boldsymbol{\theta}_0)\right) + \alpha \boldsymbol{I}\right]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0). \tag{8.28}$$

- Conjugate Gradient, BFGS,….

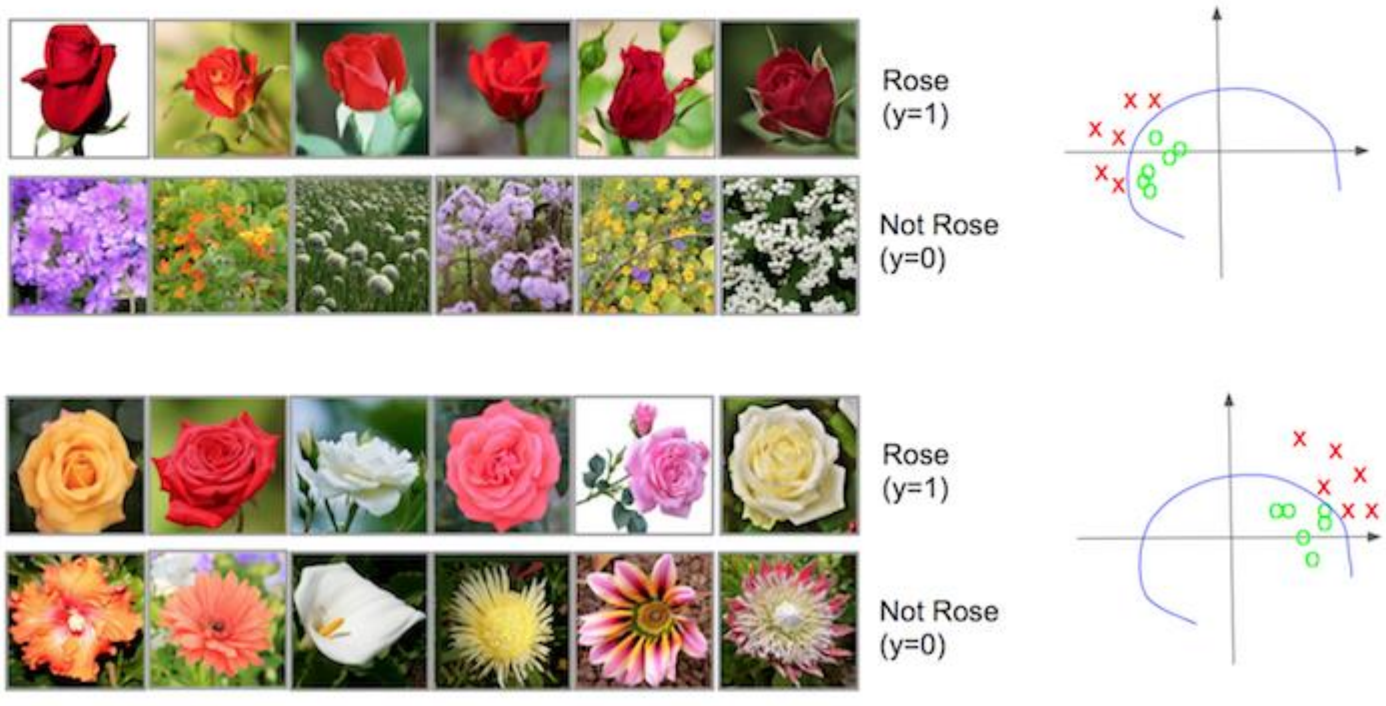
Additional Strategies for SGD

- Random Shuffle training data after every epoch
- Curriculum Learning: Sort data (easy to hard)
- Batch normalization: Re-normalizes every mini-batch to zero mean, unit variance
- Gradient noise (Decreasing variance)

Batch Normalization

https://www.learnopencv.com/batch-normalization-in-deep-networks/

- Internal Covariate Shift:
- Different *pdf* in training and test set



Batch Normalization

https://www.learnopencv.com/batch-normalization-in-deep-networks/

- Need same distribution in each minibatch!
- How to Solve:
  - Input layer normalization (He/Xavier/…) and shuffling!
- What about hidden layer:
  - Uniform/Normal *dist*. in input layer ⇨Non-uniform/Non-Normal dist. In hidden layer
  - Hidden layer input dist. varied in each batch! (**internal covariate shift**)
- Original work:
  - *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift by Sergey Ioffe and Christian Szegedy, 2015 (Google Team)*

Batch Normalization

- Idea: Insert a (Batch) Normalization Layer (BN-Layer) before each layer
- Normalization need data → Batch/min or batch training
- Normalization per cell! (Whitening need too many sample per mini-batch)

Normalize each layer output (input of next layer) in each iteration (min-batch)

Batch Normalization

- Training Phase Algorithm:
  - Zero-mean, unit-var is not ideal for nonlinear units
  - Two Learnable extra-weights per cell!
  - Next Layer input is y_i

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

› EBP for Two Extra Weights

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}.$$

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}$$

Batch Normalization

- Test/Inference Phase:
  - There is ONE sample in input NOT mini-batch
  - But we need E{x} and var{x} to normalize

$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

1. Use expectation (unbiased estimator) of μ_B and σ_B^2 instead, as following:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \boxed{\frac{m}{m-1}}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

Batch Normalization

2. Running mean/variance during training phase

$$\hat{\mu}_B(t) = \lambda\hat{\mu}_B(t-1) + (1-\lambda)\mu_B(t-1), \lambda \sim 0.95$$

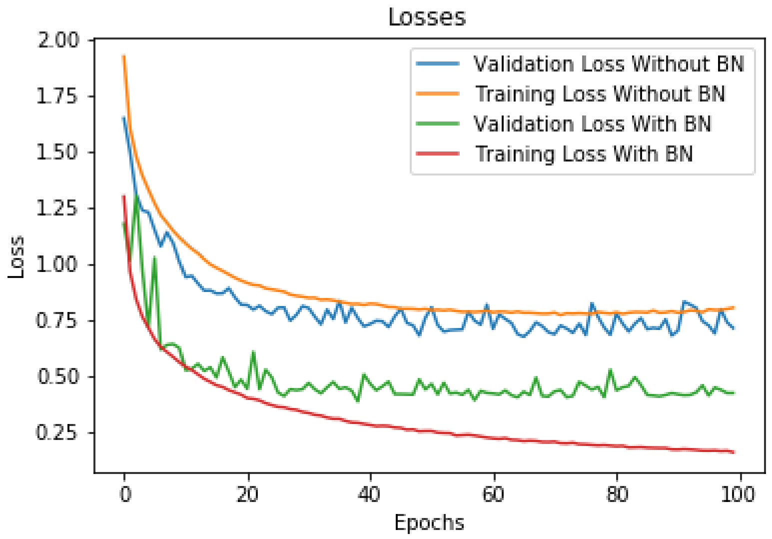
$$\widehat{\sigma_B^2}(t) = \lambda\widehat{\sigma_B^2}(t-1) + (1-\lambda)\sigma_B^2(t-1)$$

$$y^{(k)} = BN^{inf}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)}) = \frac{\gamma}{\sqrt{Var[x^{(k)}]+\epsilon}}x^{(k)} + \left(\beta - \frac{\gamma E[x^{(k)}]}{\sqrt{Var[x^{(k)}]+\epsilon}}\right)$$

Batch Normalization

https://www.learnopencv.com/batch-normalization-in-deep-networks/

- Other benefits:
  - Higher learning rate (avoiding vanishing/exploding gradient)
  - Regularization (Batch information for normalization)
  - May use saturating active function



Regularization

- Any Question?