USC University of Southern California

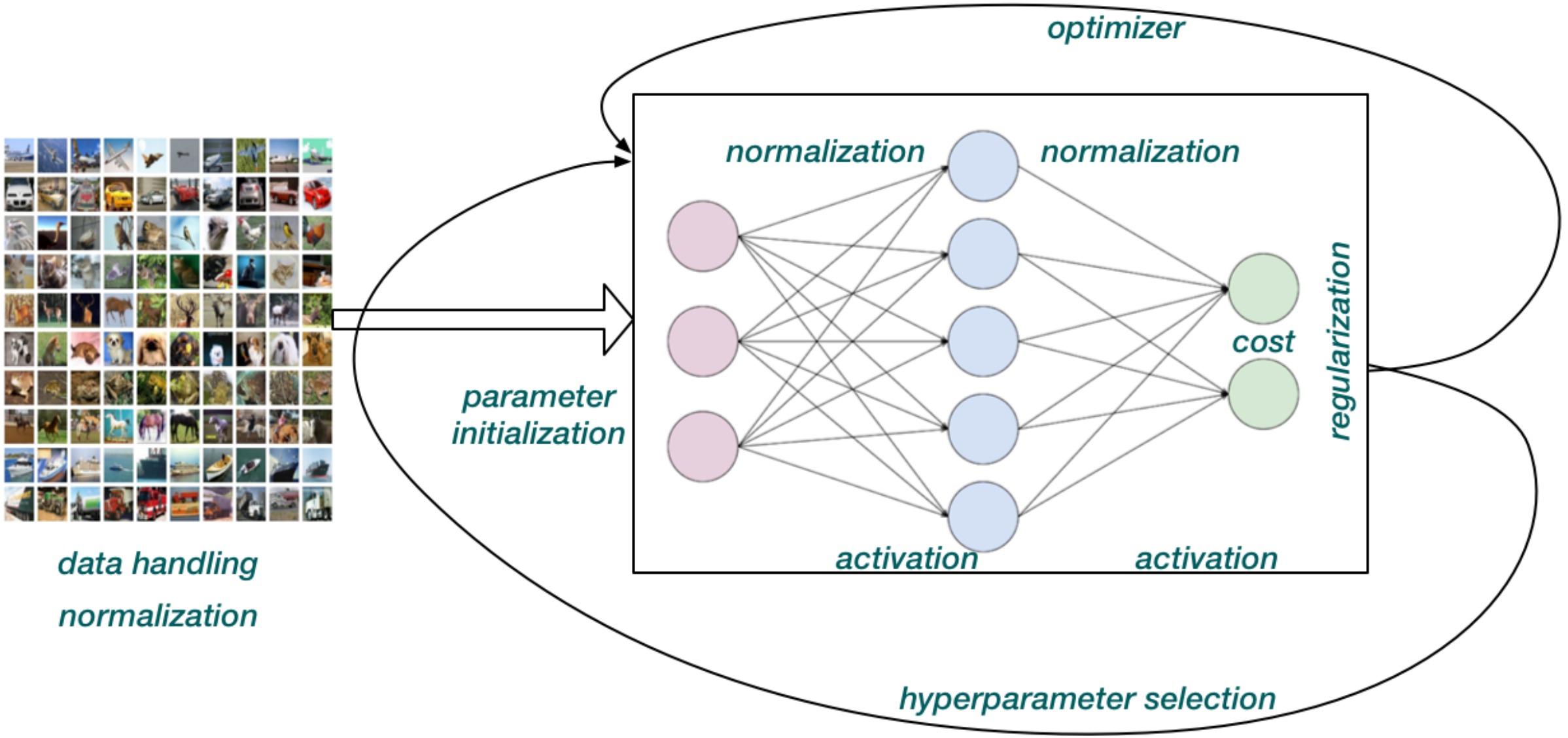
Techniques in Deep Learning

Sourya Dey

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

The Big Picture



Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Activation Functions

Recall:

$$\boldsymbol{s}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$$

Linear

$$\boldsymbol{a}^{(l)} = \boldsymbol{h}\left(\boldsymbol{s}^{(l)}\right)$$

Non-Linearity

Non-linearity is required to approximate any arbitrary function

Squashing activations

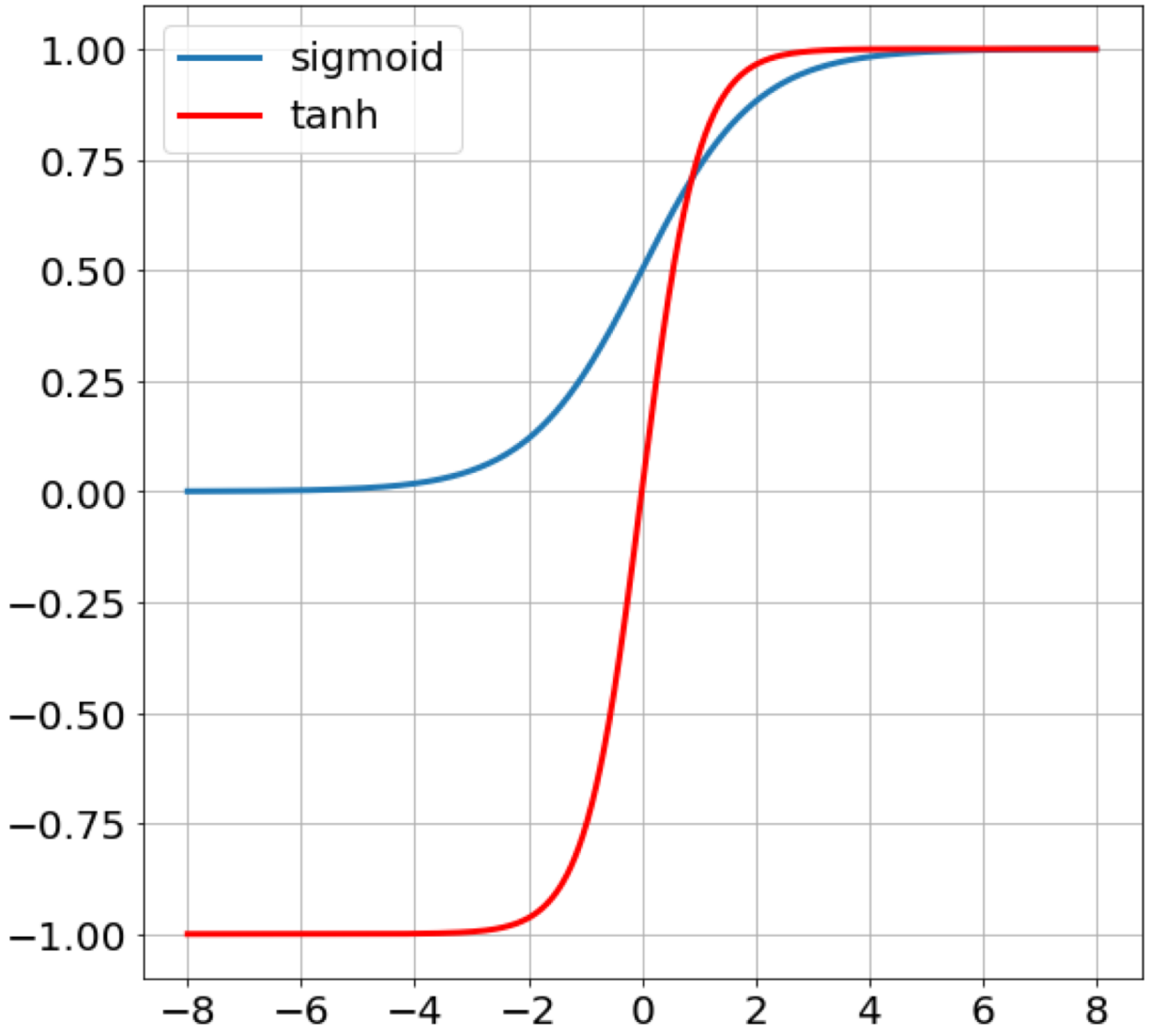
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$= 2\sigma(2x) - 1$$

Hyperbolic Tangent
(just rescaled sigmoid)



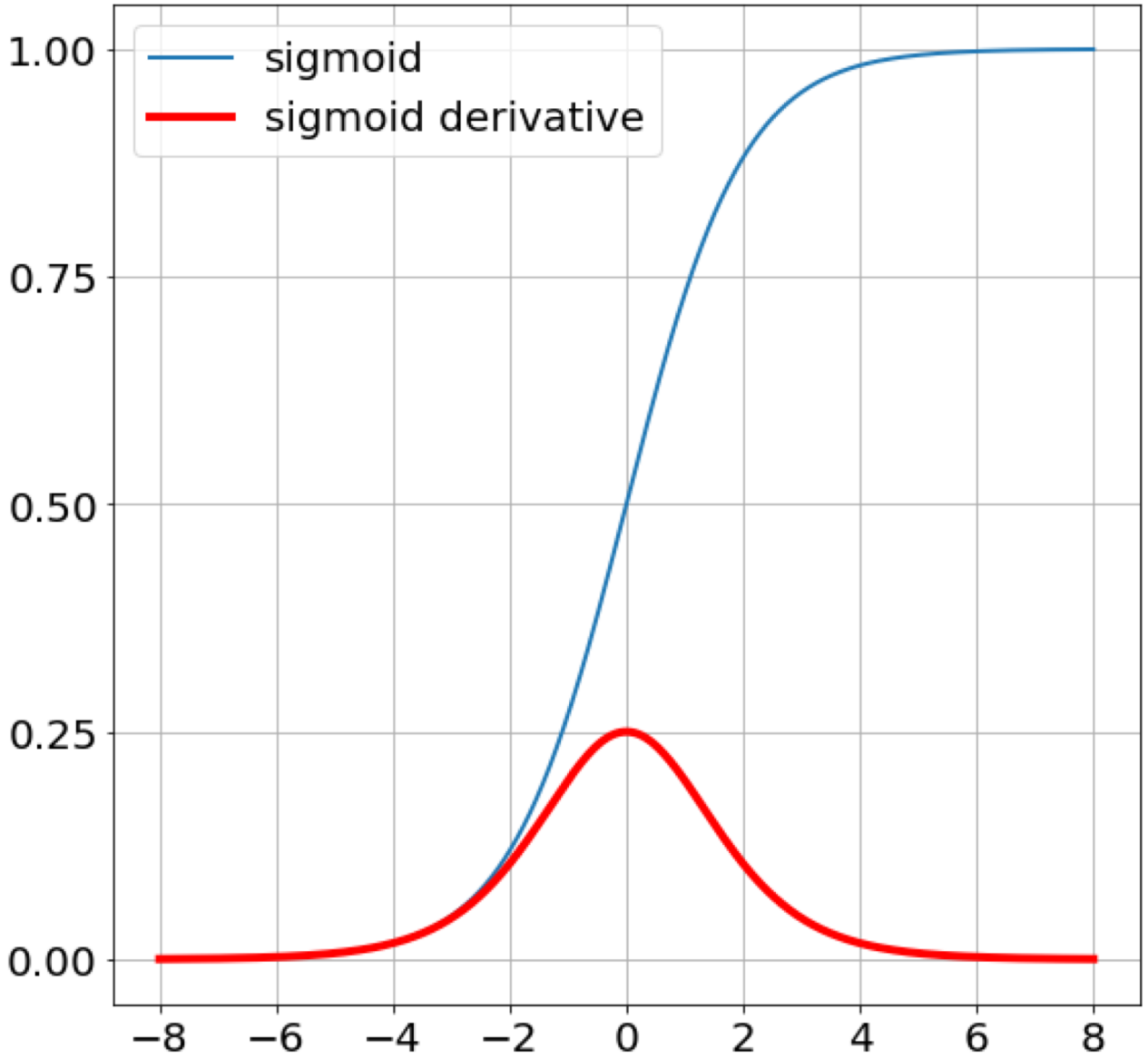
Vanishing gradients

$$\sigma'(x) = \sigma(x)\,(1 - \sigma(x))$$

Recall from BP (take example L=3)

$$\boldsymbol{\delta^{(1)}} = \boldsymbol{H'^{(1)}}\boldsymbol{W^{(2)}}^T\boldsymbol{H'^{(2)}}\boldsymbol{W^{(3)}}^T\boldsymbol{H'^{(3)}}\boldsymbol{\nabla}_{\boldsymbol{a}^{(3)}}C$$

All these numbers are <= 0.25, so gradients become very small!!



ReLU family of activations

	$x >= 0$	$x < 0$
Rectified Linear Unit (ReLU)	x	0
Exponential Linear Unit (ELU)	x	$\alpha(e^x - 1)$
Leaky ReLU	x	αx

Biologically inspired - *neurons firing vs not firing*

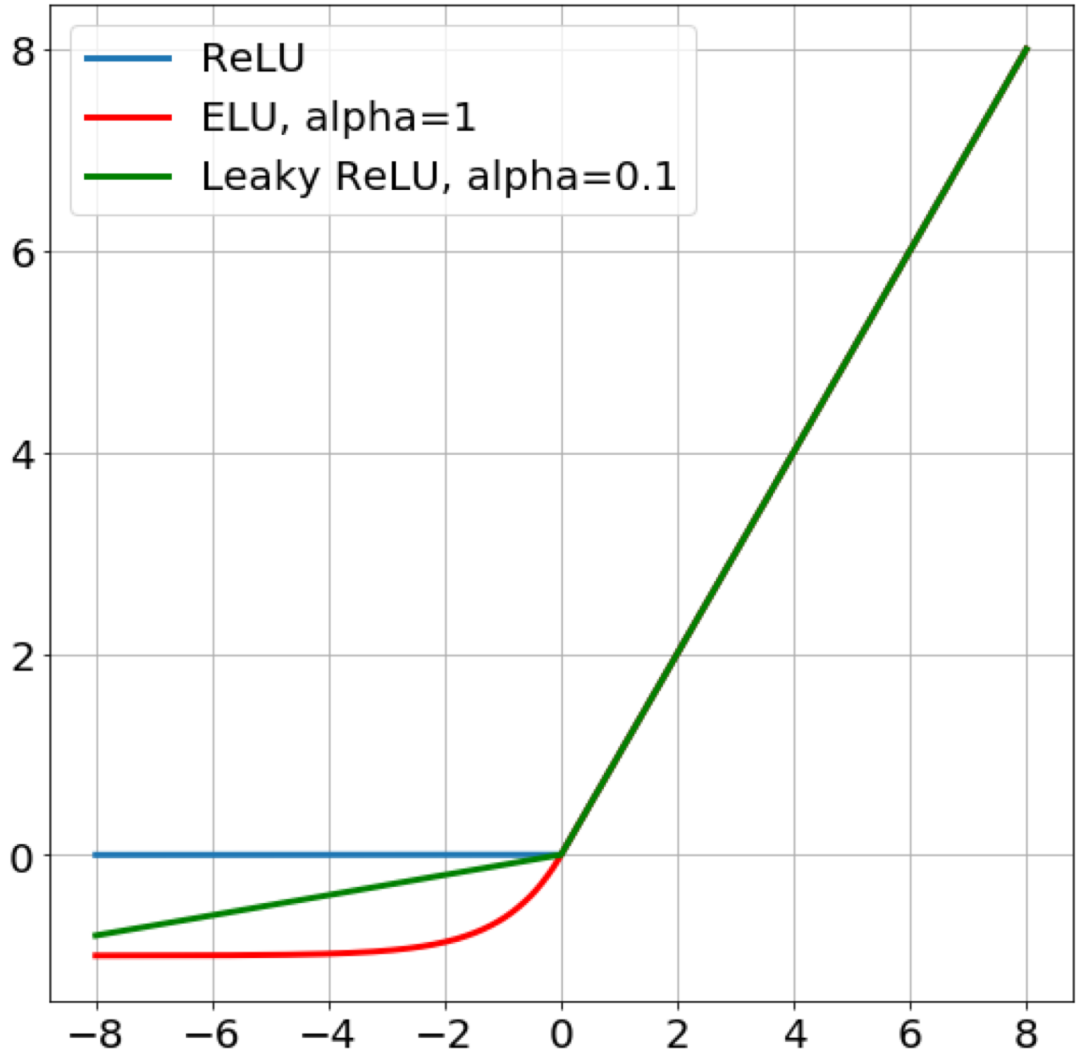
Solves vanishing gradient problem

Non-differentiable at 0, replace with anything in [0,1]

ReLU can die if x<0

Leaky ReLU solves this, but inconsistent results

ELU saturates for x<0, so less resistant to noise



Clevert, Djork-Arné; Unterthiner, Thomas; Hochreiter, Sepp (2015-11-23). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". arXiv:1511.07289

Maxout networks - Generalization of ReLU

Normally:

$$\underset{N^{(l)}\times 1}{\boldsymbol{s}^{(l)}} = \underset{N^{(l)}\times N^{(l-1)}}{\boldsymbol{W}^{(l)}} \underset{N^{(l-1)}\times 1}{\boldsymbol{a}^{(l-1)}} + \underset{N^{(l)}\times 1}{\boldsymbol{b}^{(l)}} \qquad \underset{N^{(l)}\times 1}{\boldsymbol{a}^{(l)}} = \boldsymbol{h}\left(\underset{N^{(l)}\times 1}{\boldsymbol{s}^{(l)}}\right)$$

For maxout:

$$\underset{k\times N^{(l)}\times 1}{\boldsymbol{s}^{(l)}} = \underset{k\times N^{(l)}\times N^{(l-1)}}{\boldsymbol{W}^{(l)}} \underset{N^{(l-1)}\times 1}{\boldsymbol{a}^{(l-1)}} + \underset{k\times N^{(l)}\times 1}{\boldsymbol{b}^{(l)}}$$

$$\underset{N^{(l)}\times 1}{\boldsymbol{a}^{(l)}} = \max_{\text{rows}} \underset{k\times N^{(l)}\times 1}{\boldsymbol{s}^{(l)}}$$

Learns the activation function itself

Better approximation power

Takes more computation

Goodfellow, Ian J.; Warde-Farley, David; Mirza, Mehdi; Courville, Aaron; Bengio, Yoshua (2013). "Maxout Networks". *JMLR Workshop and Conference Proceedings*. **28** (3): 1319–1327.

Example of maxout

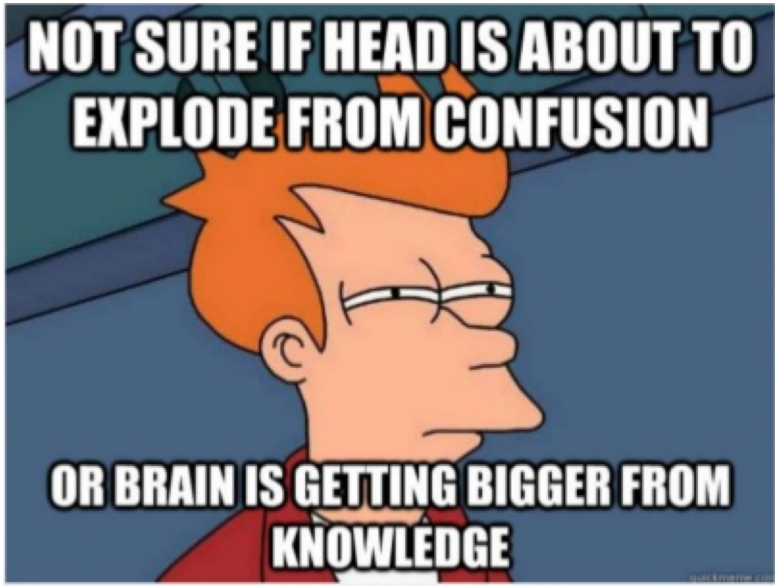
k = 2, $N^{(l)} = 5$

$$\underset{2\times 5}{\boldsymbol{s}^{(l)}} = \begin{bmatrix} 2.2 & -1.4 & 9.7 & -1.3 & 0 \\ -4 & -2.1 & 3.8 & -3 & 0.2 \end{bmatrix}$$

$$\boldsymbol{a}^{(l)} = \begin{bmatrix} 2.2 & -1.4 & 9.7 & -1.3 & 0.2 \end{bmatrix}^T$$

*ReLU is a special case of maxout with k=2 and 1 set of (**W**, b) = all 0s*

Which activation to use?



Don’t use sigmoid

Use ReLU

If too many units are dead, try other activations

And watch out for new activations (or maybe invent a new one) - deep learning moves fast!

Output layer activation - Softmax

$$\boldsymbol{a} = \begin{bmatrix} \dfrac{e^{s_1}}{\sum_{i=1}^{N} e^{s_i}} \\ \vdots \\ \dfrac{e^{s_N}}{\sum_{i=1}^{N} e^{s_i}} \end{bmatrix}$$

Extending logistic regression to multiple classes

Network output is a probability distribution!

Compare to ideal output probability distribution

$$\boldsymbol{y}^{(L)} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Cross-entropy Cost

$$C = -\sum_{i=1}^{N^{(L)}} y_i^{(L)} \ln a_i^{(L)}$$

Ground truth labels **Network outputs**

*Minimizing cross-entropy is the same as minimizing KL-divergence between the probability distributions **y** and **a***

For binary labels, this reduces to:

$$C = -\left[y^{(L)} \ln\left(a^{(L)}\right) + \left(1 - y^{(L)}\right) \ln\left(1 - a^{(L)}\right)\right]$$

The one-hot case

$$\boldsymbol{y}^{(L)} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Class 0 incorrect

Class 1 incorrect

Class *r* correct

Class $N^{(L)}$-1 incorrect

The correct class r is 1, everything else is 0

$$C = -\ln a_r^{(L)}$$

Softmax and cross-entropy with one-hot labels

Recall:

$$\boldsymbol{\delta}^{(\boldsymbol{L})} = \boldsymbol{H}'^{(L)^T} \boldsymbol{\nabla}_{\boldsymbol{a}^{(L)}} C$$

$$\delta_r^{(L)} = a_r^{(L)} - 1$$

$$\delta_{\neq r}^{(L)} = a_{\neq r}^{(L)}$$

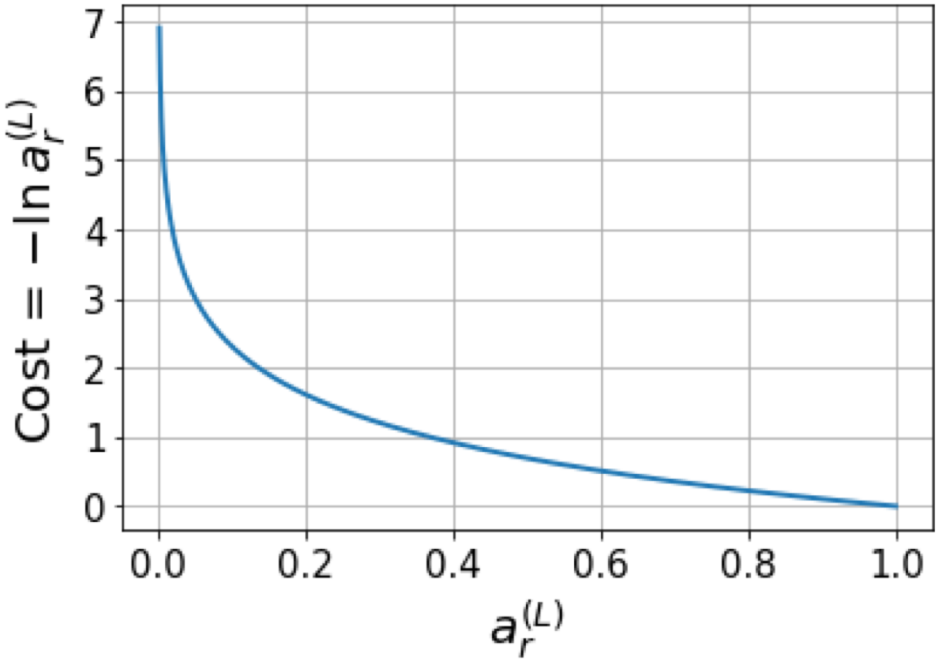
Combining:

$$\boxed{\boldsymbol{\delta}^{(L)} = \boldsymbol{a}^{(L)} - \boldsymbol{y}^{(L)}}$$

This makes beautiful sense as the error vector!

Example

$$\boldsymbol{a}^{(L)} = \begin{bmatrix} 0.05 \\ 0.84 \\ 0.11 \\ 0 \end{bmatrix} \qquad \boldsymbol{y}^{(L)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{\delta}^{(L)} = \begin{bmatrix} 0.05 \\ -0.16 \\ 0.11 \\ 0 \end{bmatrix}$$



But remember: We are interested in cost as a function of **W**, **b**

Quadratic Cost (MSE)

$$C = \sum_{i=1}^{N^{(L)}} (y_i - a_i)^2$$

Mainly used for regression, or cases where ***y*** and ***a*** are not probability distributions

Which cost function to use?

Use cross-entropy for classification tasks

It corresponds to Maximum Likelihood of a categorical distribution

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Level sets

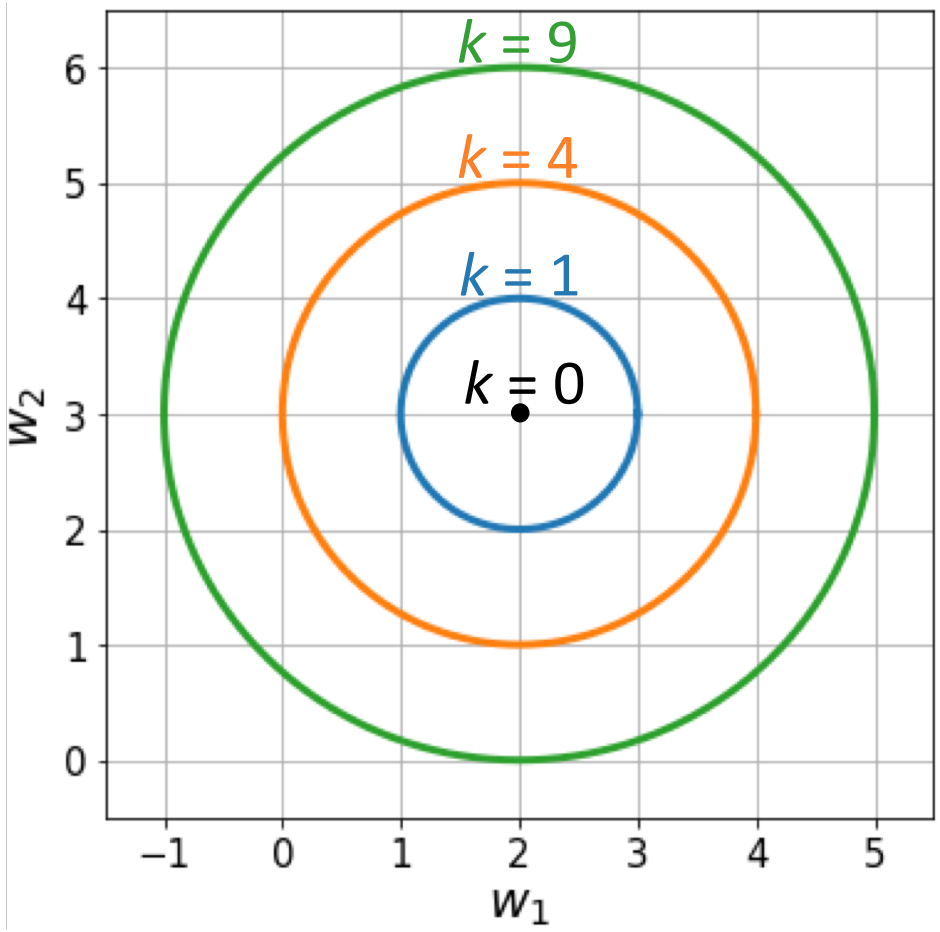
Given $f(\boldsymbol{w}) : \mathbb{R}^n \rightarrow \mathbb{R}$

A level set is a set of points in the domain at which function value is constant

$$f(\boldsymbol{w}) = k$$

Example

$f(w_1, w_2) = (w_1 - 2)^2 + (w_2 - 3)^2$

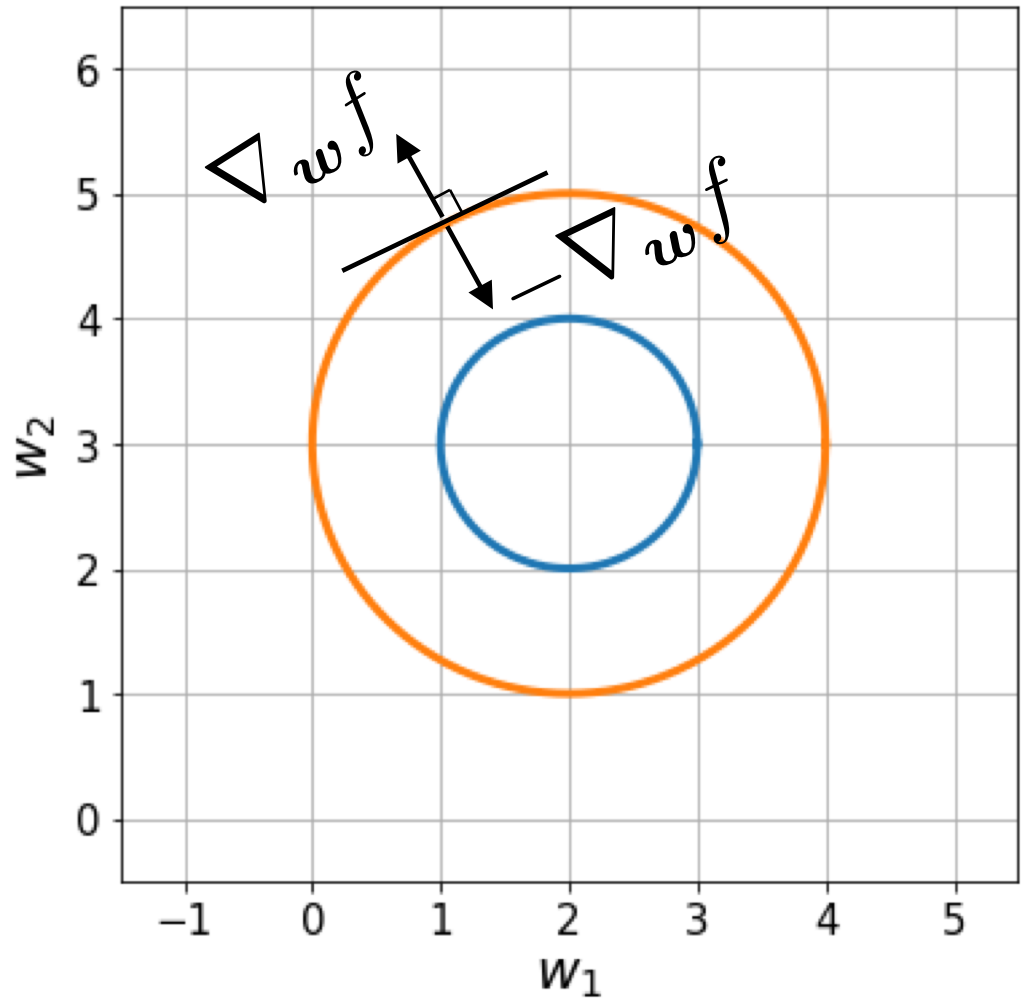


Level sets and gradient

Recall: Gradient at any point is the direction of maximum increase of the function at that point.

Gradient cannot have a component along the level set

The gradient (and its negative) are always perpendicular to the level set

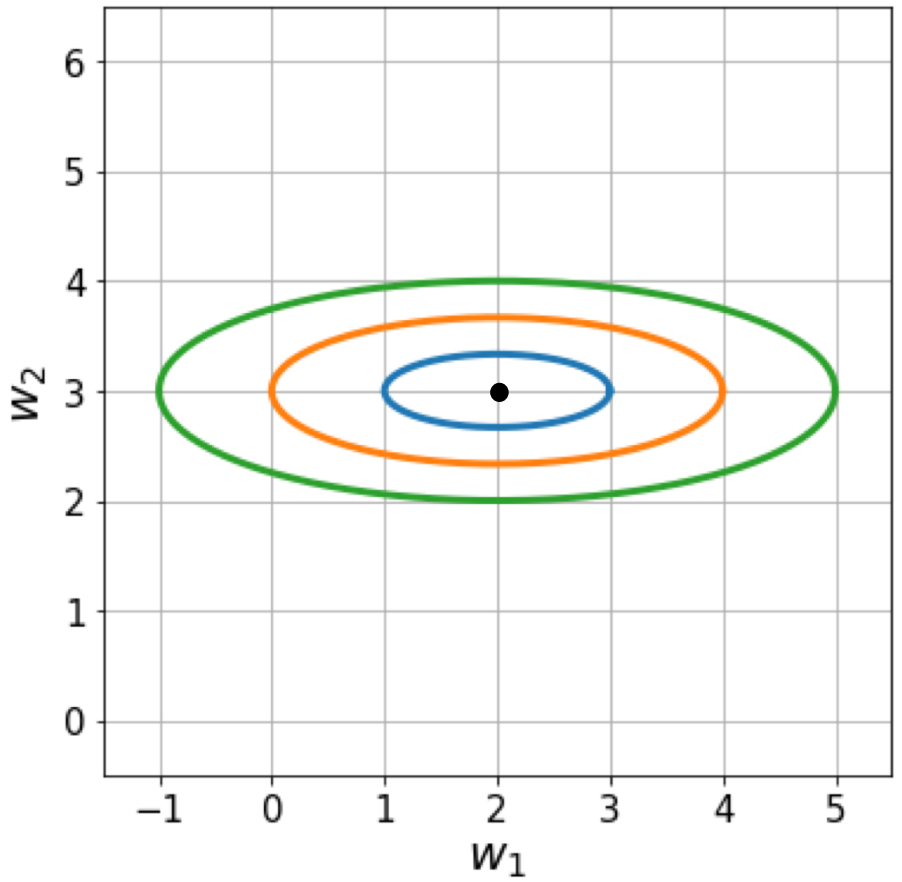


Conditioning

Informally, conditioning measures how non-circular the level sets are

Formally…

$$f(w_1, w_2) = (w_1 - 2)^2 + 9(w_2 - 3)^2$$



Ill-conditioned: Much more sensitive to w_2

Hessian: Matrix of double derivatives

Given $f(\boldsymbol{w}) : \mathbb{R}^n \to \mathbb{R}$

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_m \partial w_1} & \cdots & \frac{\partial^2 f}{\partial w_m^2} \end{bmatrix}$$

For continuous functions:

$$\frac{\partial^2 f}{\partial w_i \partial w_j} = \frac{\partial^2 f}{\partial w_j \partial w_i}$$

Hessian is symmetric

Conditioning in terms of Hessian

Conditioning is measured as the condition number of the Hessian of the cost function

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{|d_{\max}|}{|d_{\min}|}$$

I'm using *d* for eigenvalues

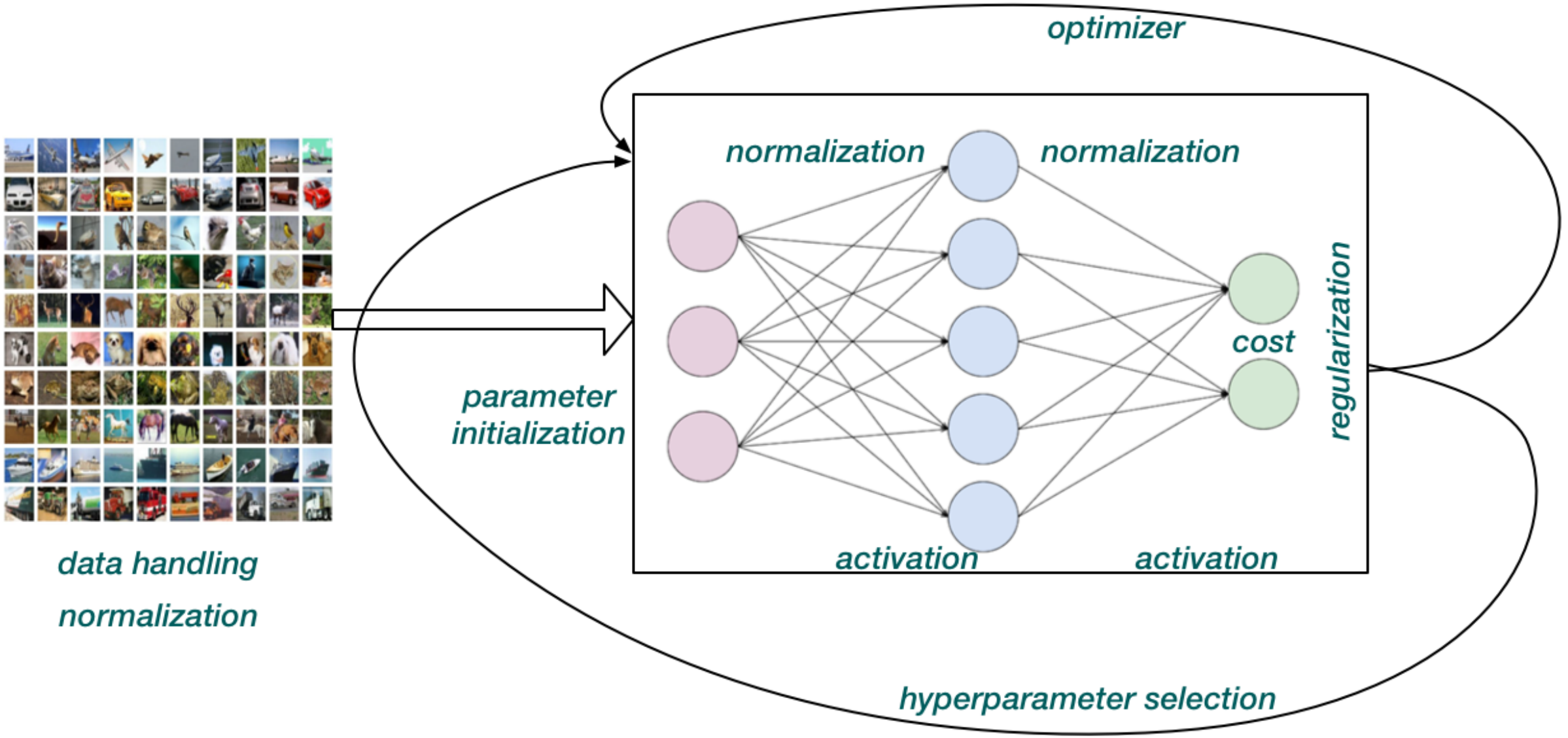
Examples:

$$C(w_1, w_2) = (w_1 - 2)^2 + (w_2 - 3)^2$$ κ = 1 => Well-conditioned

$$C(w_1, w_2) = (w_1 - 2)^2 + 9(w_2 - 3)^2$$ κ = 9 => Ill-conditioned

By the way, don't expect actual cost Hessians to be so simple…

The Big Picture



What is an Optimizer?

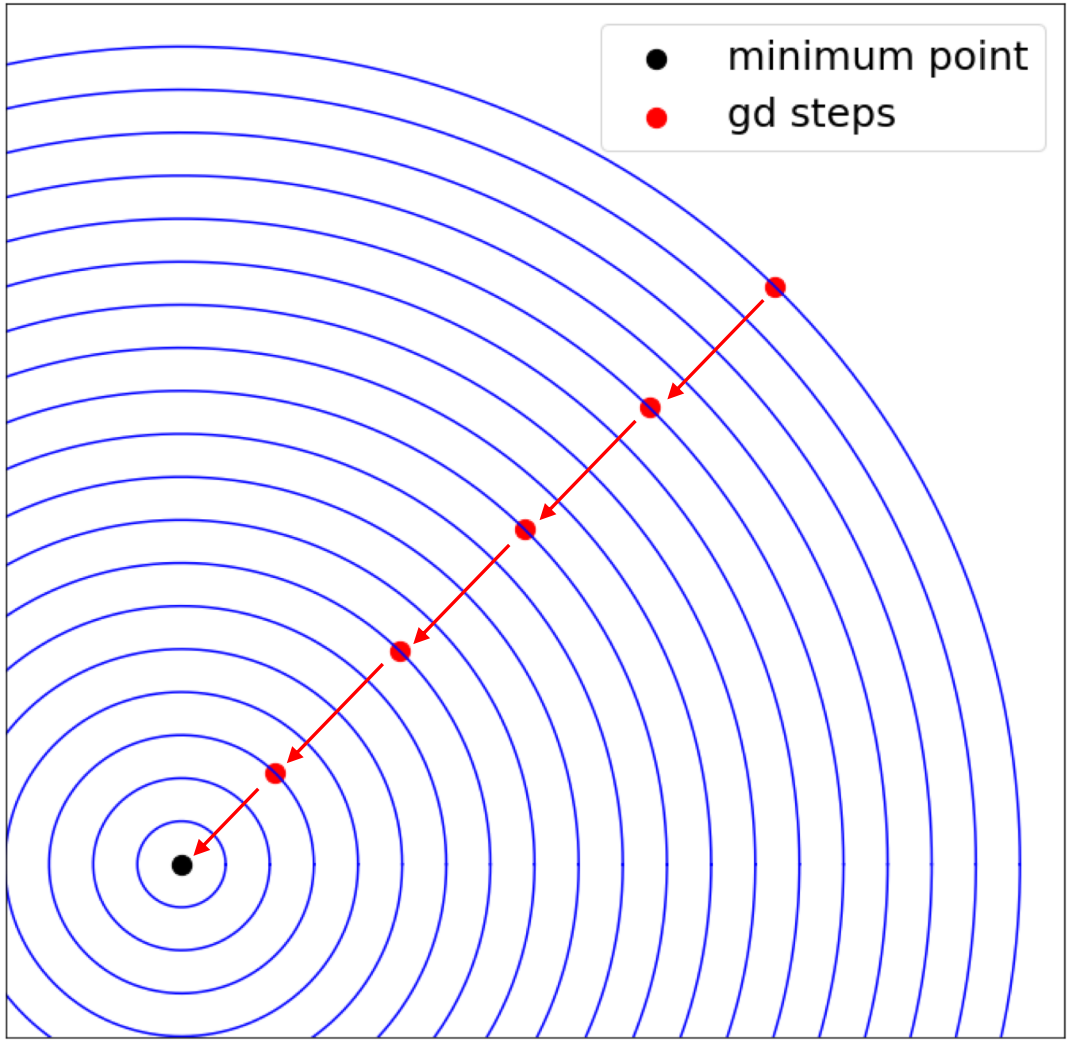
A strategy or algorithm to reduce the cost function and make the network learn

Eg: Gradient descent

$$p \leftarrow p - \eta \nabla_p C$$

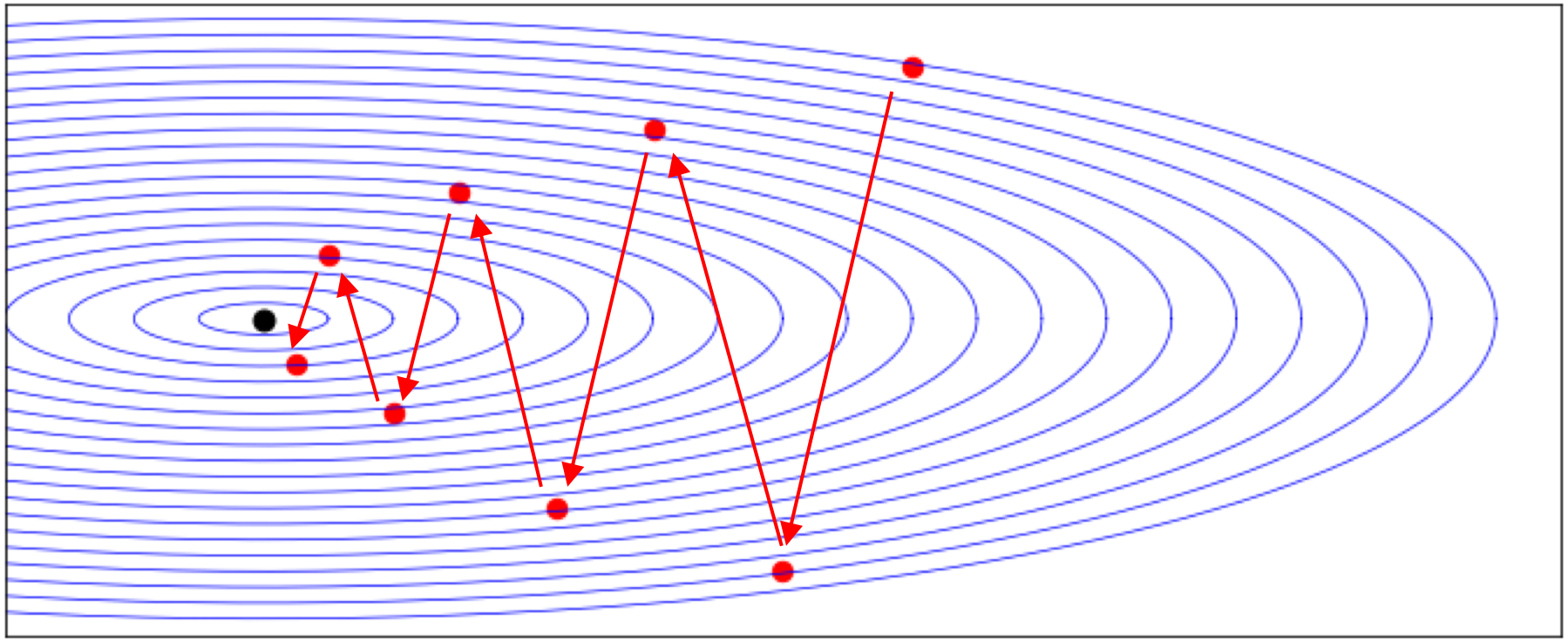
Gradient descent: Well-conditioned

Well-conditioned level sets lead to fast convergence



Gradient descent: Ill-conditioned

Ill-conditioned level sets
lead to slow convergence



Momentum

Normal update:

$$p \leftarrow p - \eta \nabla_p C$$

Momentum update:

$$v \leftarrow \alpha v - \eta \nabla_p C$$

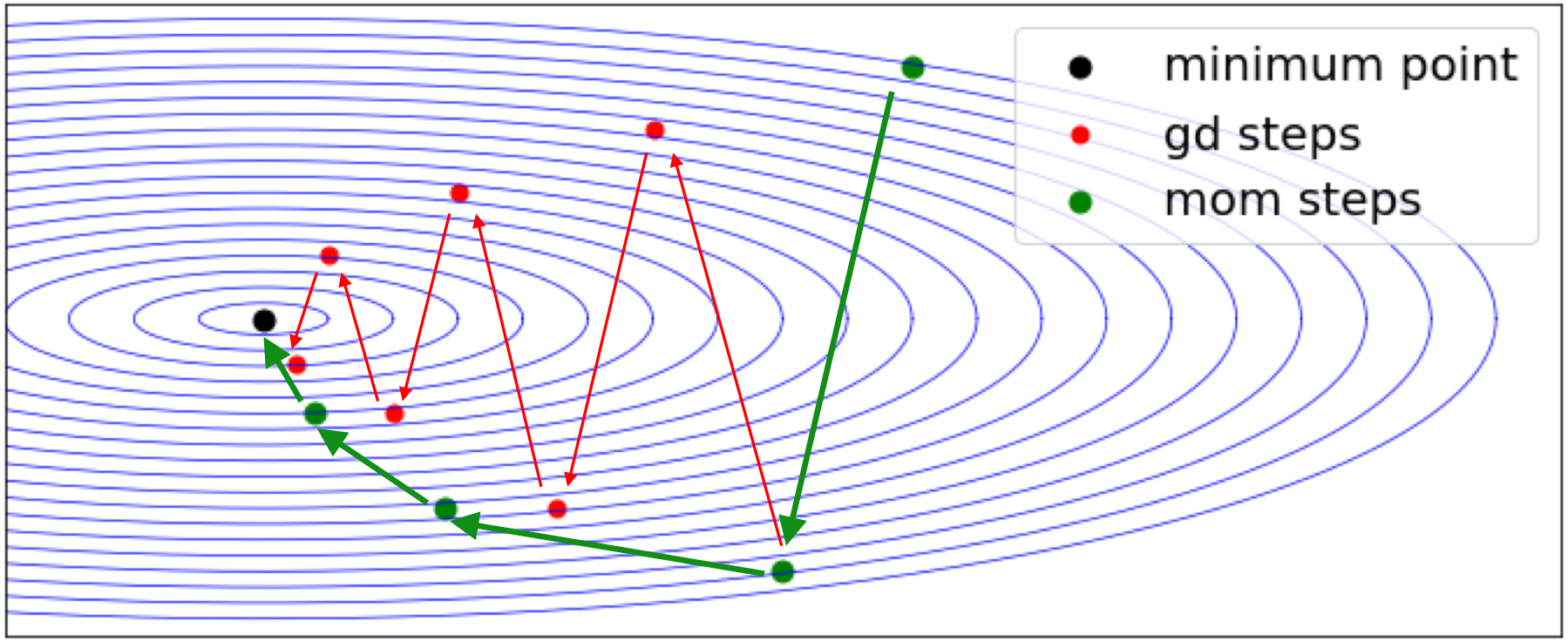
$$p \leftarrow p + v$$

Equivalent to smoothing the update by a low pass filter

$\alpha \in [0, 1]$ How much of past history should be applied, typically ~0.9

Why Momentum?

Gradient descent with momentum
converges quickly even when ill-conditioned



Nesterov Momentum

Normal update:

$$p \leftarrow p - \eta \nabla_p C$$

Momentum update:

$$v \leftarrow \alpha v - \eta \nabla_p C$$

$$p \leftarrow p + v$$

Nesterov Momentum update:

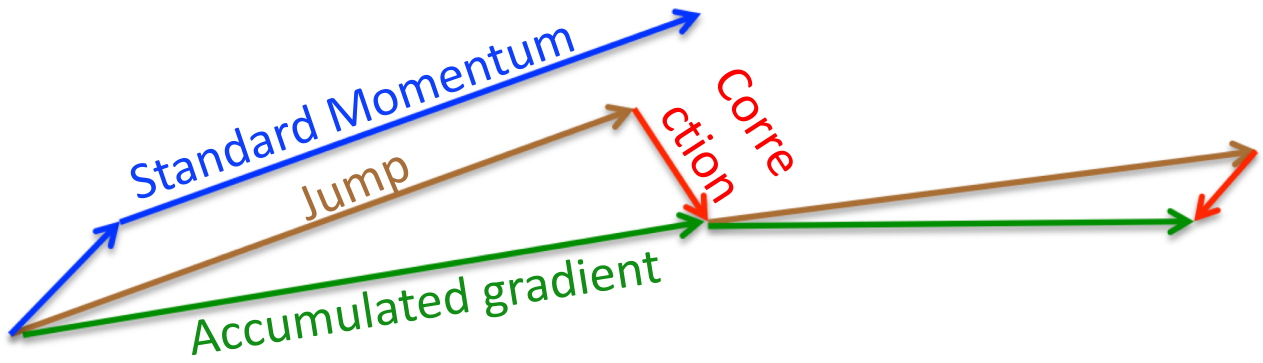
$$\tilde{p} \leftarrow p + \alpha v$$

$$v \leftarrow \alpha v - \eta \nabla_{\tilde{p}} C\left(\tilde{p}\right)$$

$$p \leftarrow p + v$$

Correction factor applied to momentum

May give faster convergence



Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O(1/k^2). Soviet Mathematics Doklady, 27, 372–376.

Pic courtesy: Geoffrey Hinton's slides: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adaptive Optimizers

✓Learning rate should be reduced with time. We don’t want to overshoot the minimum point and oscillate.

✓Learning rate should be small for sensitive parameters, and vice-versa.

✓Momentum factor should be increased with time to get smoother updates.

Adaptive optimizers:

- Adagrad
- Adadelta
- **RMSprop**
- **Adam**
- Adamax
- Nadam
- *Other ‘Ada’-sounding words you can think of…*

https://www.youtube.com/watch?v=2lUFM8yTtUc

RMSprop

Scale each gradient by an exponentially weighted moving average of its past history

$$v \leftarrow \rho v + (1-\rho)\left(\nabla_p C\right)^2$$

$$p \leftarrow p - \frac{\eta}{\sqrt{v}+\epsilon}\nabla_p C$$

Default ρ = 0.9

ϵ is just a small number to prevent division by 0. Can be machine epsilon

Sensitive parameters get low η

Hinton, G. Neural networks for machine learning. Coursera, video lectures, 2012. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam

RMSprop with momentum and bias correction

At time step (t+1):

$$v_1 \leftarrow \rho_1 v_1 + (1-\rho_1)\nabla_p C$$

$$v_2 \leftarrow \rho_2 v_2 + (1-\rho_2)\left(\nabla_p C\right)^2$$

$$\tilde{v}_1 = \frac{v_1}{1-\rho_1^t}$$

$$\tilde{v}_2 = \frac{v_2}{1-\rho_2^t}$$

Bias corrections to make

$$\mathbb{E}\left\{\tilde{v}_1\right\} = \mathbb{E}\left\{\nabla_p C\right\}$$

$$\mathbb{E}\left\{\tilde{v}_2\right\} = \mathbb{E}\left\{\left(\nabla_p C\right)^2\right\}$$

$$p \leftarrow p - \frac{\eta}{\sqrt{\tilde{v}_2}+\epsilon}\tilde{v}_1$$

Momentum

Exponentially weighted moving average of past history

<u>Defaults:</u>

η=0.001, ρ_1 = 0.9, ρ_2 = 0.999

D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learning Representations (ICLR), 2014.

Which optimizer to use?


```
opt = SGD(eta=0.01, mom=0.95, decay=1e-5, nesterov=False)
```

```
opt = Adam()
```

Machine learning vs Simple Optimization

	Simple Optimization	Machine Learning
Goal	Minimize f(x)	Minimize C(w) on test data
Typical problem size	A few variables	A few million variables
Approach	Gradient descent 2^{nd} order methods	Gradient descent on training data (2^{nd} order methods not feasible)
Stopping criterion	x* = argmin f(x)	?? No access to test data

Machine learning is about generalizing well on test data

Outline

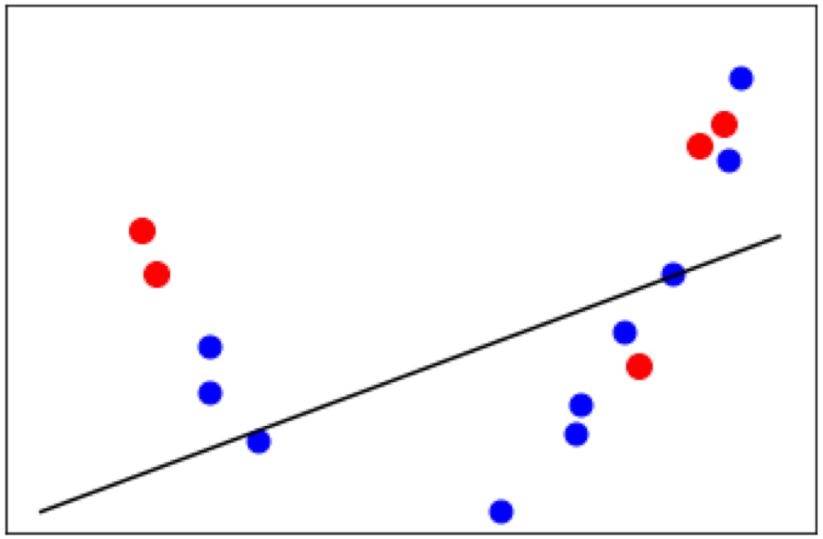
- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Regularization

Regularization is any modification intended to reduce generalization (test) cost, but not training cost

The problem of overfitting

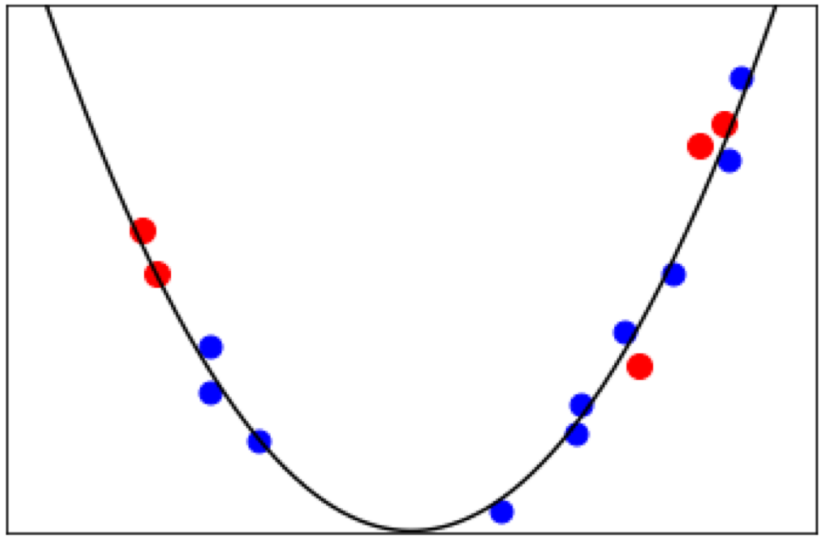
Underfitting: Order = 1



$C_{train} = 0.385$

$C_{test} = 0.49$

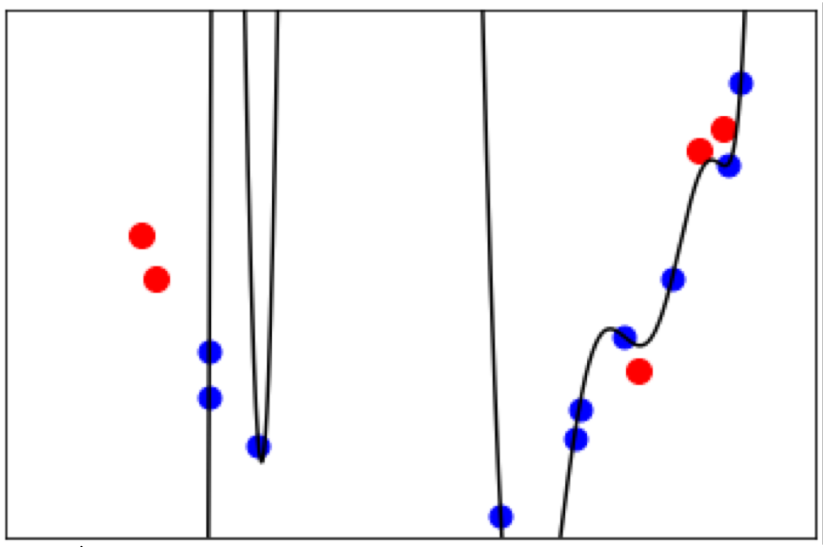
Perfect: Order = 2



$C_{train} = 0.012$

$C_{test} = 0.019$

Overfitting: Order = 9



$C_{train} = 0$

$C_{test} = 145475$

Bias-Variance Tradeoff

Say the (unknown) optimum value of some parameter is w*, but the value we get from training is w

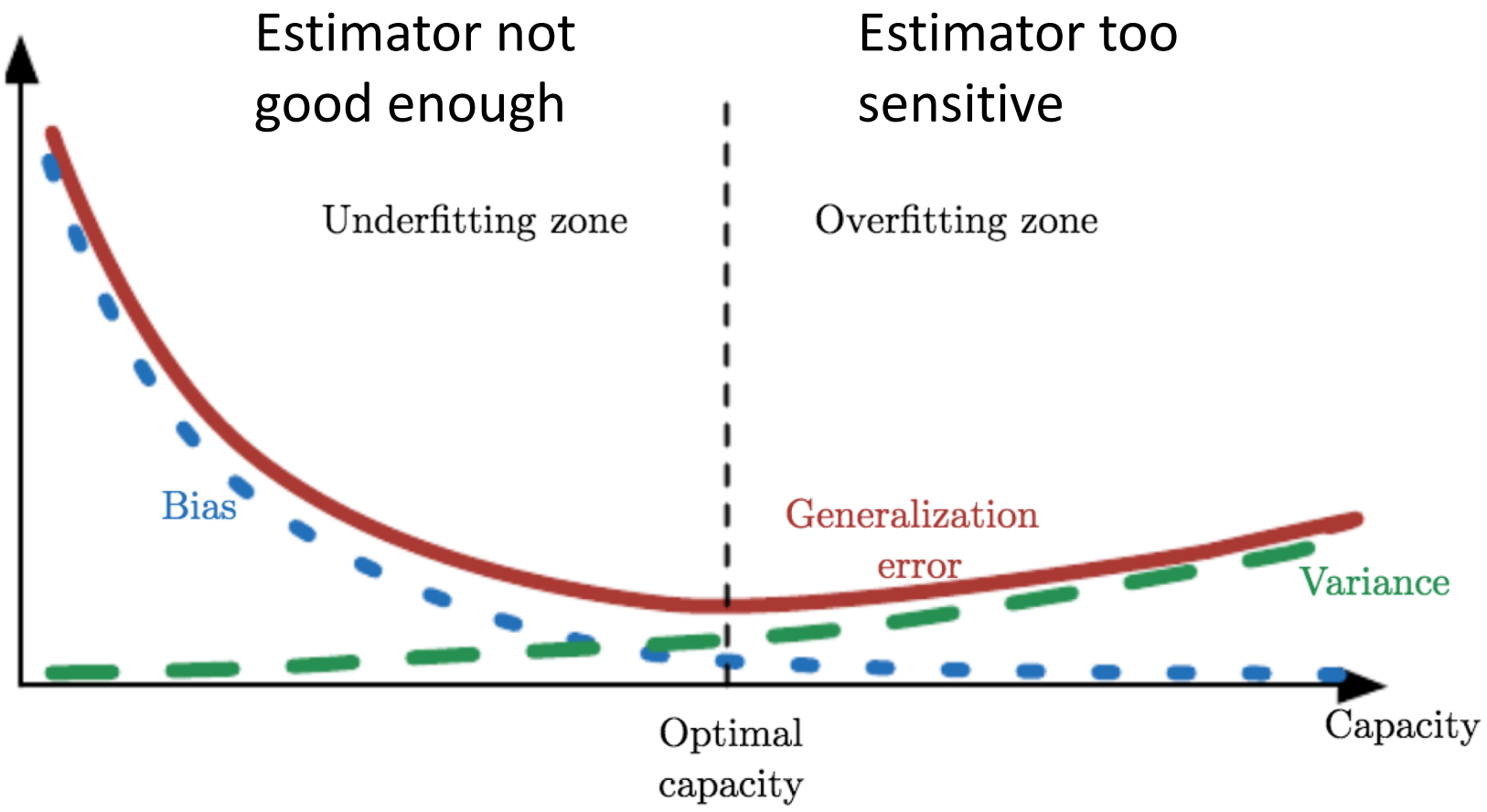
$$
\begin{aligned}
\mathrm{MSE} &= \mathbb{E}\left\{(w - w^*)^2\right\} \\
&= \mathbb{E}\left\{w^2\right\} - 2w^*\mathbb{E}\{w\} + w^{*2} \\
&= \mathbb{E}\left\{w^2\right\} - \mathbb{E}\{w\}^2 + \mathbb{E}\{w\}^2 - 2w^*\mathbb{E}\{w\} + w^{*2} \\
&= \left(\mathbb{E}\left\{w^2\right\} - \mathbb{E}\{w\}^2\right) + \left(\mathbb{E}(w) - w^*\right)^2 \\
&= \mathrm{Var}(w) + \mathrm{Bias}(w)^2
\end{aligned}
$$

Reduce Bias: Make expected value of estimate close to true value

Reduce Variance: Estimate bounds should be tight

For a given MSE, bias trades off with variance

Bias-Variance Tradeoff



Pic courtesy: I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

Minibatches

$$p \leftarrow p - \eta \left\langle \nabla_p C \right\rangle_{\text{traindata}}$$

Estimate true gradient using data average of sample gradients of training data

$$p \leftarrow p - \frac{\eta}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(\nabla_p C\right)^{[i]}$$

Batch gradient descent

Best possible approximation to true gradient

$$p \leftarrow p - \frac{\eta}{M} \sum_{i=1}^{M} \left(\nabla_p C\right)^{[i]}$$

Minibatch gradient descent

Noisy approximation to true gradient

A Note on Terminology

Batch size : Same as minibatch size M

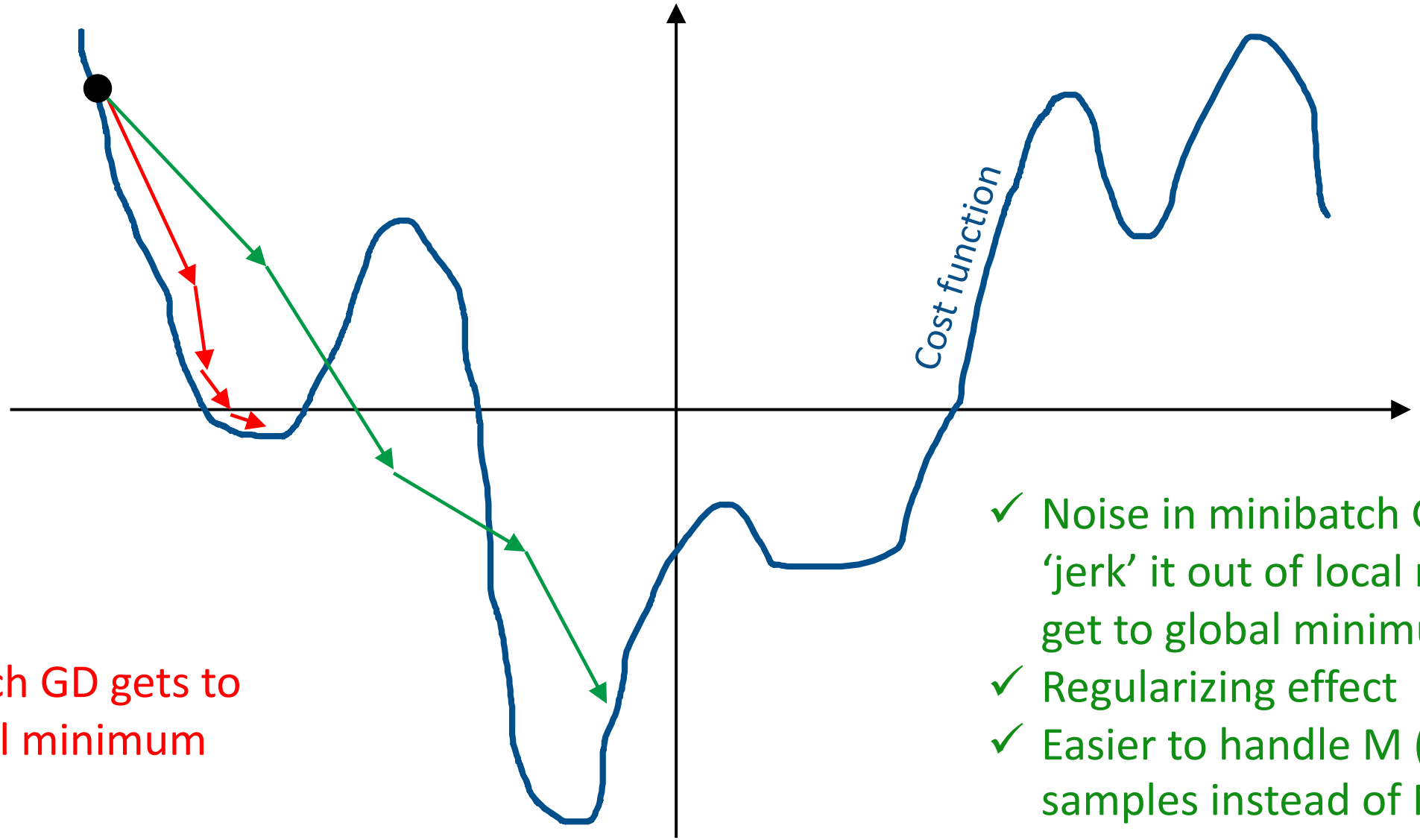
Batch GD : Use all samples

Minibatch GD : $1 < M < N_{train}$

Online learning : $M = 1$

Stochastic GD : $1 <= M < N_{train}$
Basically the same as minibatch GD.

Optimizers - Why minibatches?



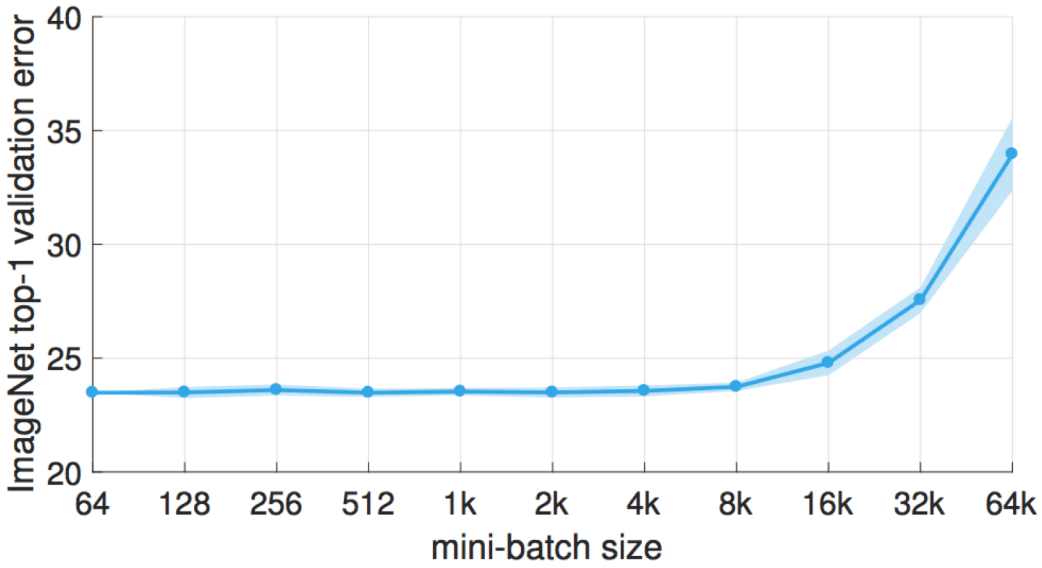
Batch GD gets to local minimum

- ✓ Noise in minibatch GD may 'jerk' it out of local minima and get to global minimum
- ✓ Regularizing effect
- ✓ Easier to handle M (~200) samples instead of N_{train} (~10^5)

Choosing a batch size

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, Kaiming He. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". arXiv:1706.02677

M <= 8000 is fine



But…

Dominic Masters, Carlo Luschi. ”Revisiting Small Batch Training for Deep Neural Networks". arXiv:1804.07612

Recommend M = 16, 32, 64

Both use the same dataset: Imagenet, with N_{train} > 1 million

So what batch size do I choose?

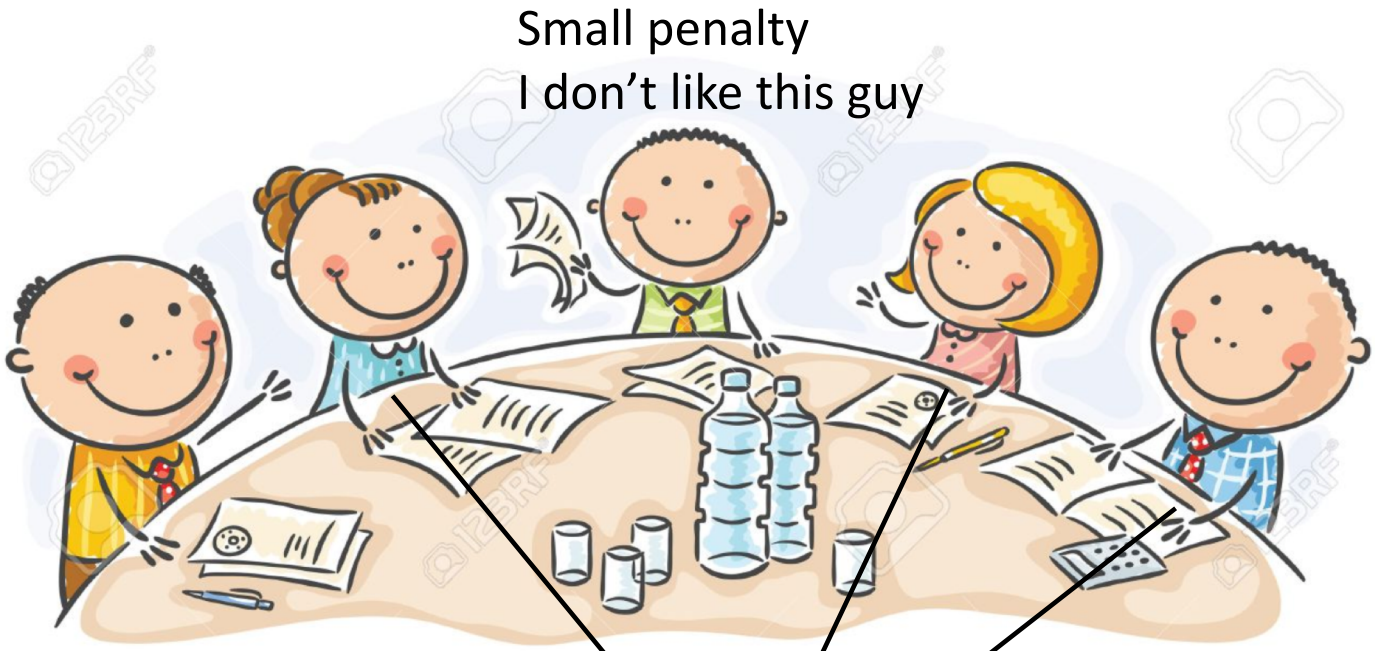
No consensus

- ✓ Depends on the number of workers/threads in CPU/GPU
- ✓ Powers of 2 work well
- ✓ Keras default is 32 : Good for small datasets (N_{train}<10000)
- ✓ For bigger datasets, I would recommend trying 128, 256, even 1024
- ✓ Your computer might slow down with M > 1000

Concept of Penalty Function

Penalty measures dislike

Eg: Penalty = Yellow dress



Pic courtesy: https://www.123rf.com

L2 weight penalty (Ridge)

$$C(\boldsymbol{w}) = C_0(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$$

Regularized cost

Original unregularized cost (like cross-entropy)

Regularization term

Update rule: $w \leftarrow w - \eta \left(\nabla_w C_0 + 2\lambda w\right)$

Regularization hyperparameter effect

$$\lambda = \frac{\text{Importance of small weights}}{\text{Importance of reducing training cost}}$$

$\lambda = 0$ Reduce training cost only

$$\boldsymbol{w}^* = \arg\min C_0(\boldsymbol{w})$$

Overfitting

$\lambda = \infty$ Reduce weight norm only

$$\boldsymbol{w}^* = \mathbf{0}$$

Underfitting

Typically $\lambda \in [10^{-5}, 10^{-3}]$

Effect on Conditioning

Consider a 2nd order Taylor approximation of the unregularized cost function C_0 around its optimum point **w***, i.e. $\nabla C_0(\mathbf{w}^*) = 0$

$$C_0(\boldsymbol{w}) = C_0\left(\boldsymbol{w}^*\right) + \frac{1}{2}\left(\boldsymbol{w} - \boldsymbol{w}^*\right)^T \boldsymbol{H}\left(\boldsymbol{w} - \boldsymbol{w}^*\right)$$

$$\nabla_{\boldsymbol{w}} C_0(\boldsymbol{w}) = \boldsymbol{H}\left(\boldsymbol{w} - \boldsymbol{w}^*\right)$$

$$\nabla_{\boldsymbol{w}} C(\boldsymbol{w}) = \boldsymbol{H}\left(\boldsymbol{w} - \boldsymbol{w}^*\right) + 2\lambda \boldsymbol{w}$$

→ Adding the gradient of the regularization term to get gradient of the regularized cost function C

Effect on Conditioning

Set the gradient to 0 to find optimum $\tilde{\boldsymbol{w}}$ of the regularized cost

$$\nabla_{\boldsymbol{w}} C(\boldsymbol{w}) = \mathbf{0}$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + 2\lambda \boldsymbol{I})^{-1} \boldsymbol{H} \boldsymbol{w}^*$$

$$= \left(\boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^T + 2\lambda \boldsymbol{I}\right)^{-1} \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^T \boldsymbol{w}^*$$

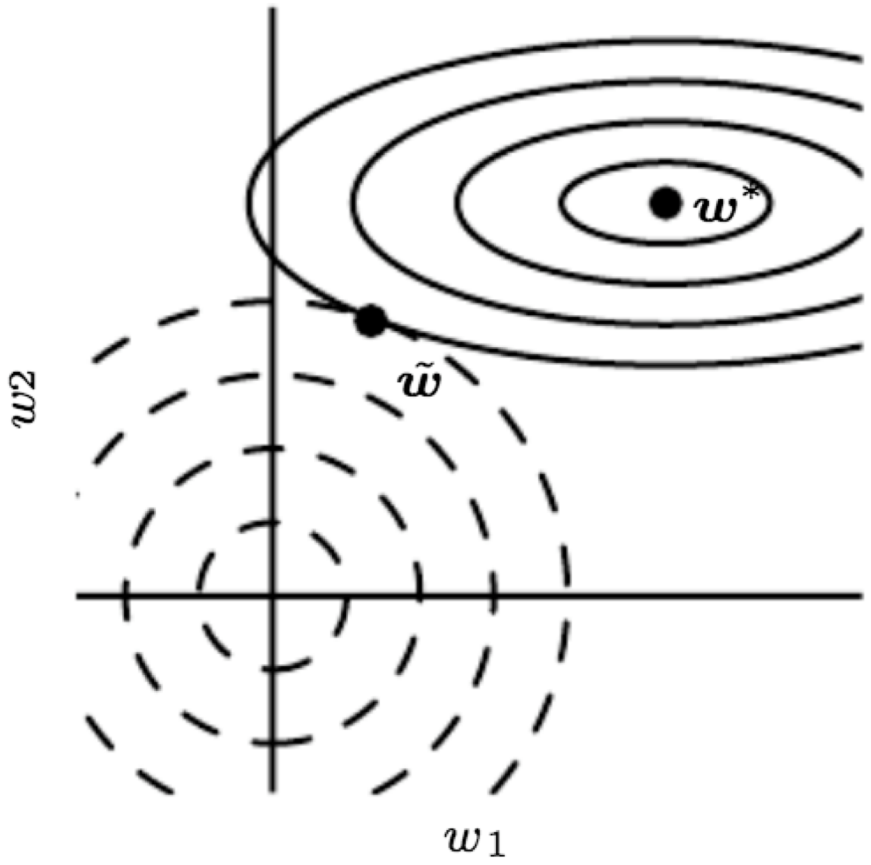
$$= \left[\boldsymbol{Q}(\boldsymbol{D} + 2\lambda \boldsymbol{I})\boldsymbol{Q}^T\right]^{-1} \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^T \boldsymbol{w}^*$$

$$= \boldsymbol{Q}\left[(\boldsymbol{D} + 2\lambda \boldsymbol{I})^{-1} \boldsymbol{D}\right] \boldsymbol{Q}^T \boldsymbol{w}^*$$

Eigendecomposition of **H**
D is the diagonal matrix of eigenvalues

*Any eigenvalue d_i of **H** is replaced by $d_i / (d_i + 2\lambda)$ => Improves conditioning*

Effect on Weights



Level sets of C_0

Level sets of norm penalty

Less important weights like w_1 are driven towards 0 by regularization

Pic courtesy: I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

L1 weight penalty (LASSO)

$$C(\boldsymbol{w}) = C_0(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

Regularized cost

Original unregularized cost (like cross-entropy)

Regularization term

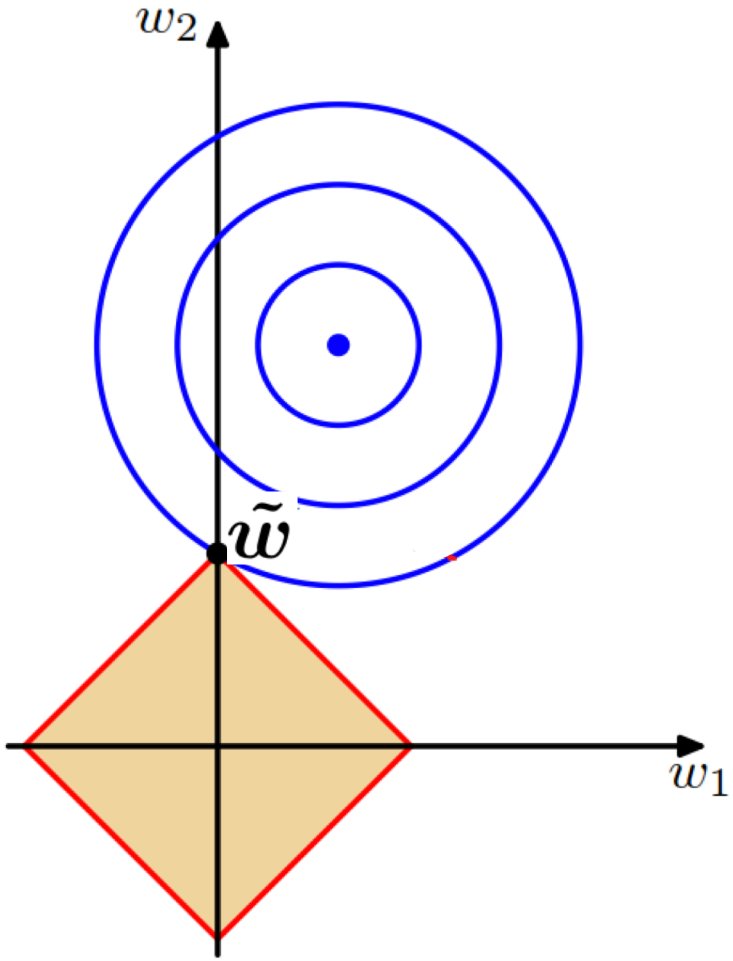
Update rule: $w \leftarrow w - \eta \left(\nabla_w C_0 + \lambda \operatorname{sign}(w)\right)$

Sparsity

L1 promotes sparsity => Drives weights to 0

Eg: Some weight w = 0.1

	Function value	Dislike	Update value	Action
L2	$w^2 = 0.01$	Small	$\eta\lambda w = 0.1(\eta\lambda)$	Leave it
L1	$\|w\| = 0.1$	Large	$\eta\lambda * sign(w) = \eta\lambda$	Make it smaller



Pic courtesy: Found at https://datascience.stackexchange.com/questions/30237/does-l1-regularization-always-generate-a-sparse-solution

Other weight penalties

L0 : Count number of nonzero weights.

This is a great way to get sparsity, but not differentiable and hard to analyze

Elastic Net - Combine L1 and L2 :

Reduce some weights, zero out some

Also see Group lasso:

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in Proc. Advances in Neural Information Processing Systems 29 (NIPS), 2016, pp. 2074–2082.

Weight penalty as MAP

$$\begin{aligned}\tilde{\boldsymbol{w}} &= \arg\max p(\boldsymbol{w}|\text{Data}) \\ &= \arg\max p(\text{Data}|\boldsymbol{w})p(\boldsymbol{w}) \\ &= \arg\max \left[\log p(\text{Data}|\boldsymbol{w}) + \log p(\boldsymbol{w})\right] \\ &= \arg\min \left[-\log p(\text{Data}|\boldsymbol{w})\right] \quad + \quad \arg\min \left[-\log p(\boldsymbol{w})\right]\end{aligned}$$

Original cost C_0

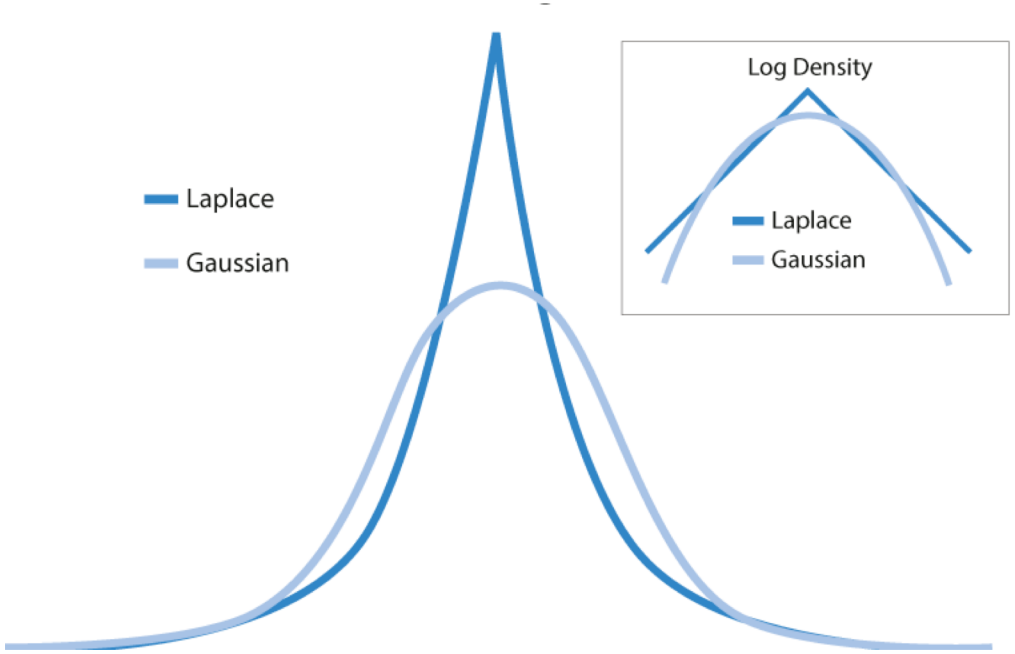
Weight penalty

Weight penalty is our prior belief about the weights

Weight penalty as MAP

$$-\log\left[p(\boldsymbol{w}) = \mathcal{N}\left(\boldsymbol{w}, \mathbf{0}, \frac{1}{\lambda}\boldsymbol{I}\right)\right] = \lambda \left\|\boldsymbol{w}\right\|_2^2$$

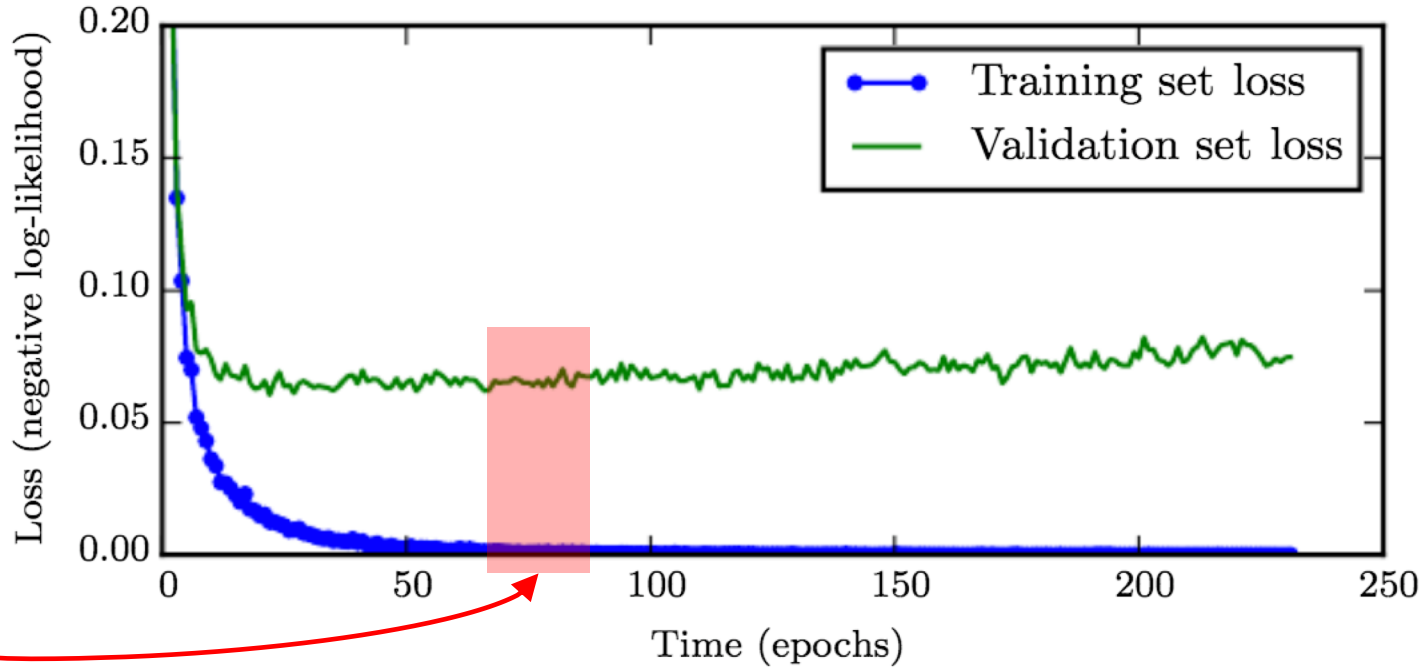
$$-\log\left[p(w_i) = \text{Laplace}\left(w_i, 0, \frac{1}{\lambda}\right)\right] = \lambda \left\|\boldsymbol{w}\right\|_1$$



Early stopping

Validation metrics are used as a proxy for test metrics

Stop training if validation metrics don't get better

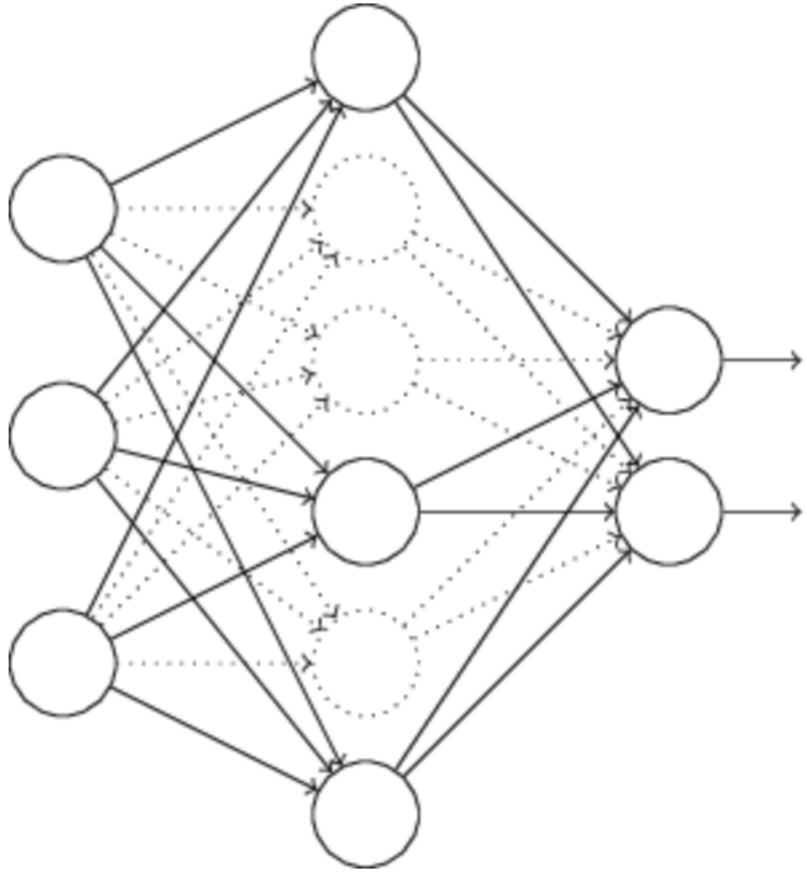


Pic courtesy: I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.

Dropout

For every minibatch during training, delete a random selection of input and hidden nodes

Keep probability p_k : Fraction of units to keep (i.e. not drop)



But why dropout??

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014

Pic courtesy: Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

Ensemble Methods

Let's train an ensemble of *k* similar networks on the same problem and average their test results
(Random differences: Initialization, dataset shuffling)

Error of a single network: ϵ_i

Error of the ensemble: $\frac{1}{k}\sum_i \epsilon_i$

$$\begin{aligned}\mathrm{MSE}_{\mathrm{ens}} &= \mathbb{E}\left\{\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right\} \\ &= \frac{1}{k^2}\mathbb{E}\left\{\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right\} \\ &= \frac{1}{k}\mathbb{E}\left\{\epsilon_i^2\right\} + \frac{k-1}{k}\mathbb{E}\left\{\epsilon_i\epsilon_j\right\}\end{aligned}$$

Averaging the results of multiple networks reduces errors

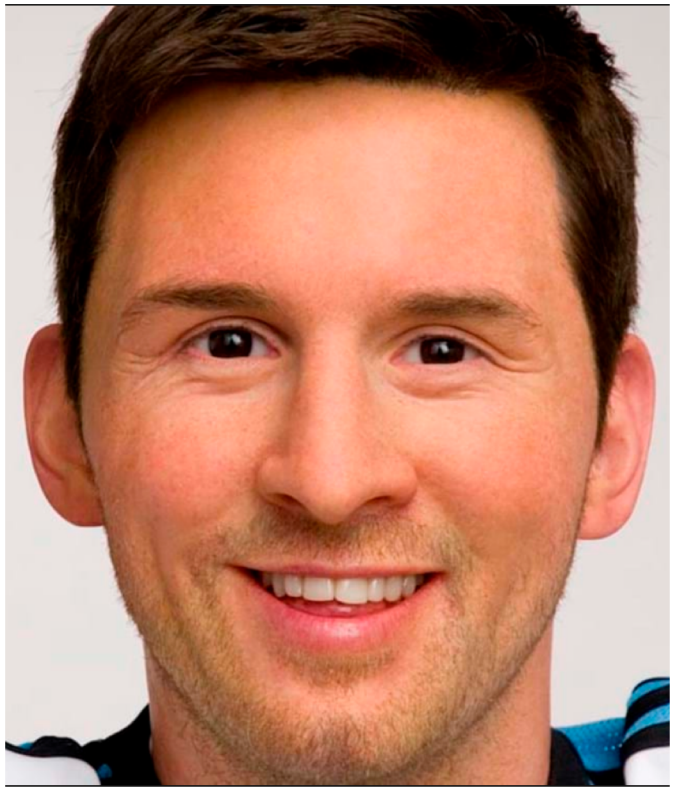
$$\because \mathbb{E}\left\{\epsilon_i\epsilon_j\right\} \leq \mathbb{E}\left\{\epsilon_i^2\right\}$$
$$\Rightarrow \mathrm{MSE}_{\mathrm{ens}} \leq \mathrm{MSE}$$

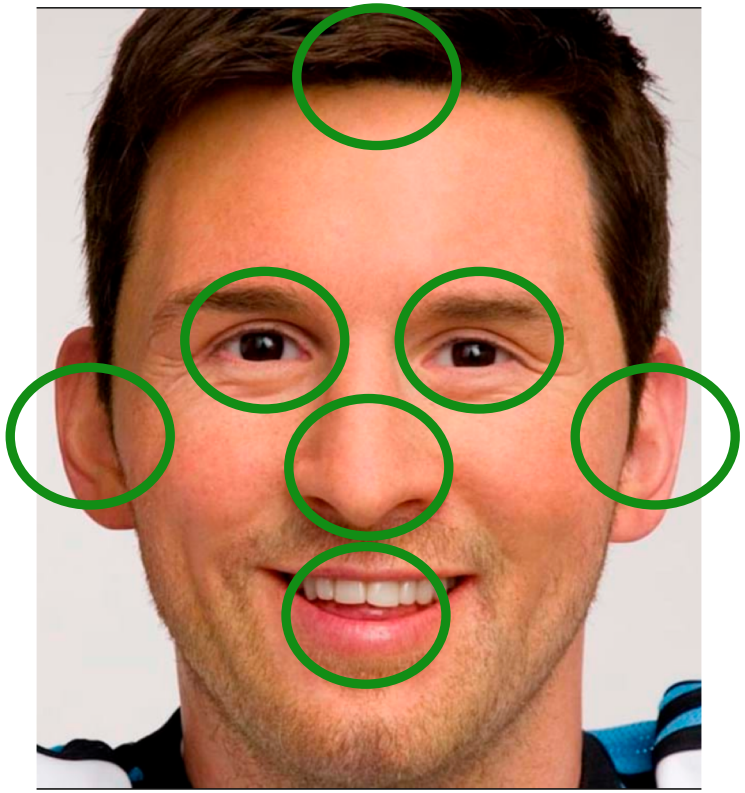
But this process is expensive!

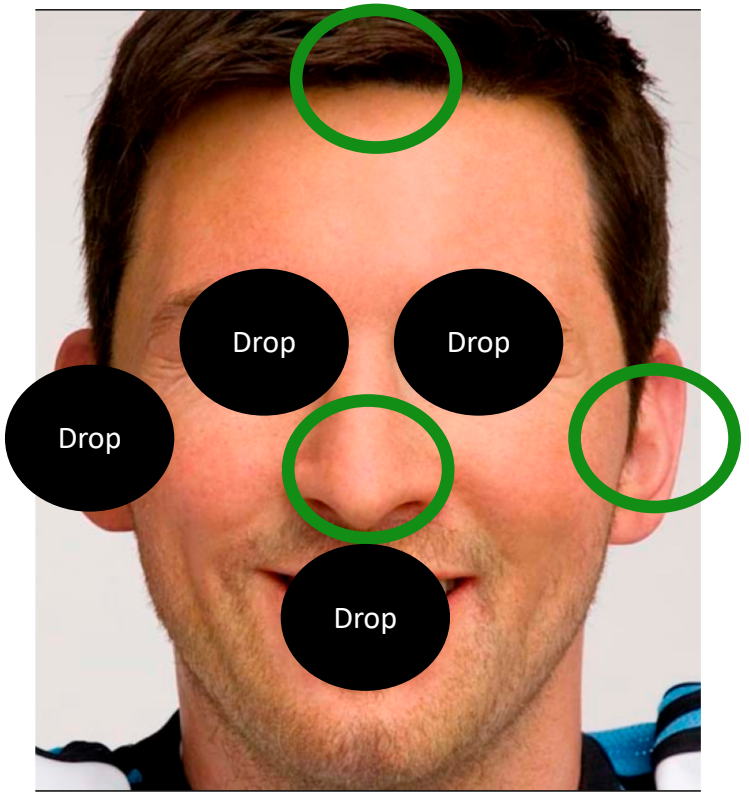
Return to dropout

Dropout is an ensemble method using the same shared weights
=> Computationally cheaper

Improves robustness, since individual nodes have to work without depending on other deleted nodes => Regularization







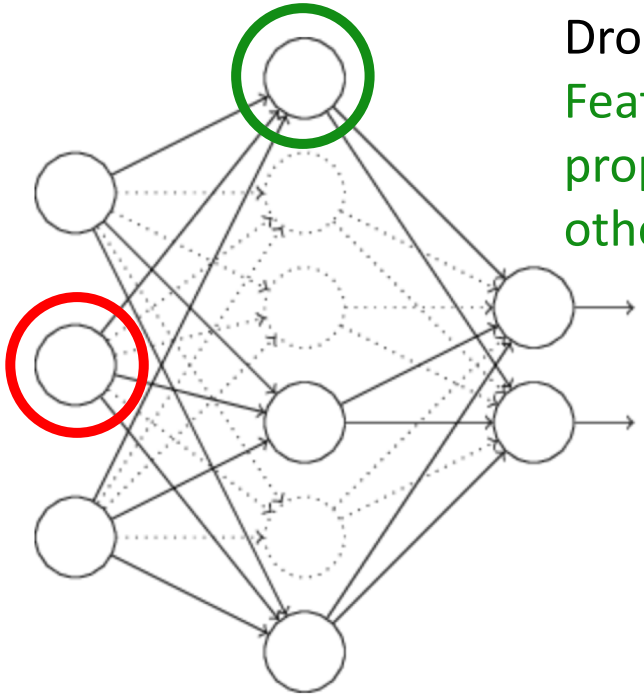
How much dropout?

Original paper recommendations:

Input $p_k = 0.8$

Hidden $p_k = 0.5$

Output $p_k = 1$



Drop hidden:
Features still propagate through other hidden nodes

Drop output: Cannot classify!

0.5 gives maximum regularization effect:
P. Baldi, P. J. Sadowski, "Understanding Dropout", Proc. NIPS 2013

But you should try other values as well!

Conv layer $p_k = 0.7$

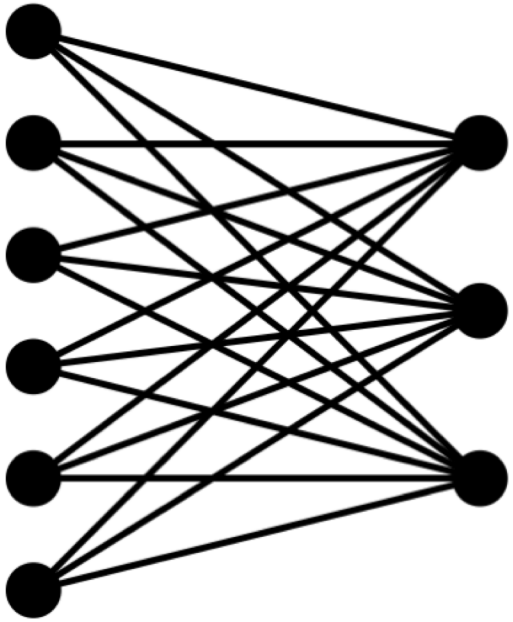
Drop input: Feature gone forever!
Some people use input $p_k = 1$

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014

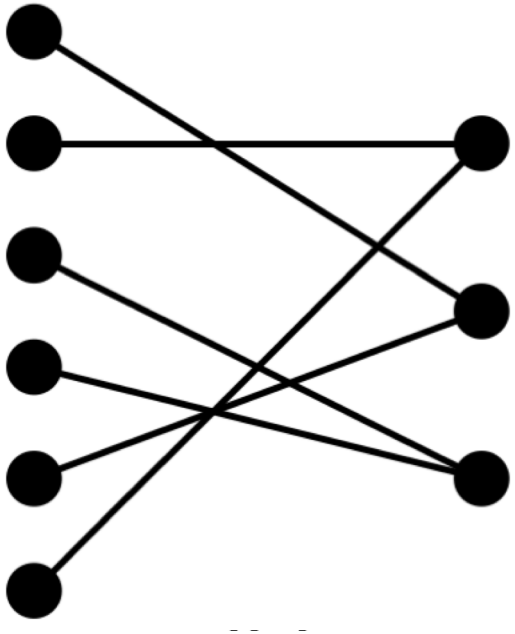
More on Dropout

- All nodes and weights are present in inference. Multiply weights by p_k during inference to compensate
  - This assumes linearity, but still works!
- Dropout is like multiplicative 0-1 noise for each node
- Dropout is best for big networks
  - Increase capacity, increase generalization power, but prevent overfitting

My Research: Pre-defined sparsity



Fully-connected network



Pre-defined sparse network

Remove weights from the very beginning
Train and test on a low complexity network

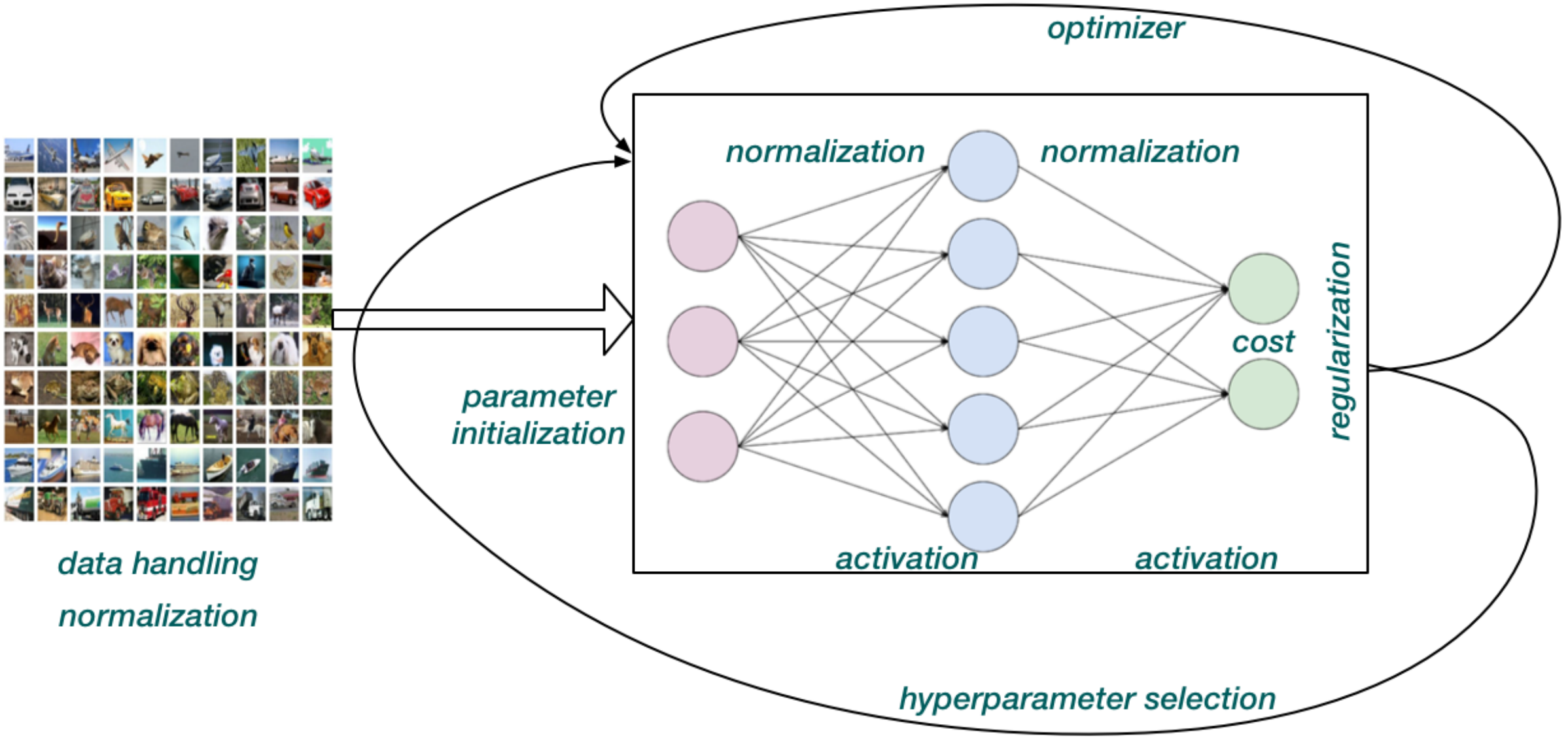
How to design a good connection pattern?

S Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg `Pre-Defined Sparse Neural Networks with Hardware Acceleration,'' submitted to the IEEE Journal on Emerging and Selected Topics in Circuits and Systems, Special Issue: Customized sub-systems and circuits for deep learning, 2018. Available online at arXiv:1812.01164.

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

The Big Picture



Parameter Initialization

$$p \leftarrow p - \eta \nabla_p C$$

What is the initial value of p?

How about…. Same initialization, like all 0s

BAD

We want each weight to be useful on its own, not mirror other weights

Think of a company, do 2 people have exactly the same job?

Instead, initialize weights randomly

Glorot (Xavier) Normal Initialization

Imagine a linear function: $y = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N$

Say all w, x are IID: $\mathrm{Var}(y) = N\mathrm{Var}(w)\mathrm{Var}(x)$

$$\text{if } \mathrm{Var}(w) = \frac{1}{N}$$

$$\text{then } \mathrm{Var}(y) = \mathrm{Var}(x)$$

For forward prop : $\mathrm{Var}(W^{(l)}) = 1/N^{(l-1)}$

For backprop : $\mathrm{Var}(W^{(l)}) = 1/N^{(l)}$

Compromise : $$W^{(l)} \sim \mathcal{N}\left(0, \frac{2}{N^{(l)} + N^{(l-1)}}\right)$$

Despite a lot of assumptions, it works well!!

Xavier Glorot, Yoshua Bengio. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

Why…

… should variances be equal?

… should we take harmonic mean as compromise?

The original paper doesn't explicitly explain these

I think of it as similar to vanishing and exploding gradient issues.

From a forward-propagation point of view, to keep information flowing we would like that

$$\forall(i, i'), Var[z^i] = Var[z^{i'}]. \tag{8}$$

From a back-propagation point of view we would similarly like to have

$$\forall(i, i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right]. \tag{9}$$

These two conditions transform to:

$$\forall i, \quad n_i Var[W^i] = 1 \tag{10}$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1 \tag{11}$$

As a compromise between these two constraints, we might want to have

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}} \tag{12}$$

Xavier Glorot, Yoshua Bengio. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

Glorot (Xavier) Uniform Initialization

Let $W^{(l)} \sim U(-a, a)$

$$\Rightarrow \mathrm{Var}\left(W^{(l)}\right) = \frac{a^2}{3}$$

$$\Rightarrow \frac{a^2}{3} = \frac{2}{N^{(l)} + N^{(l-1)}}$$

$$W^{(l)} \sim U\left(-\sqrt{\frac{6}{N^{(l)} + N^{(l-1)}}}, \sqrt{\frac{6}{N^{(l)} + N^{(l-1)}}}\right)$$

He Initialization

Glorot normal assumes linear functions

Actual activations like ReLU are non-linear

So…

$$W^{(l)} \sim \mathcal{N}\left(0, \frac{2}{N^{(l-1)}}\right) \qquad W^{(l)} \sim U\left(-\sqrt{\frac{6}{N^{(l-1)}}}, \sqrt{\frac{6}{N^{(l-1)}}}\right)$$

See for details:

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Proceedings of ICCV ’15, pp 1026-1034.

Bias Initialization and Choosing an Initializer

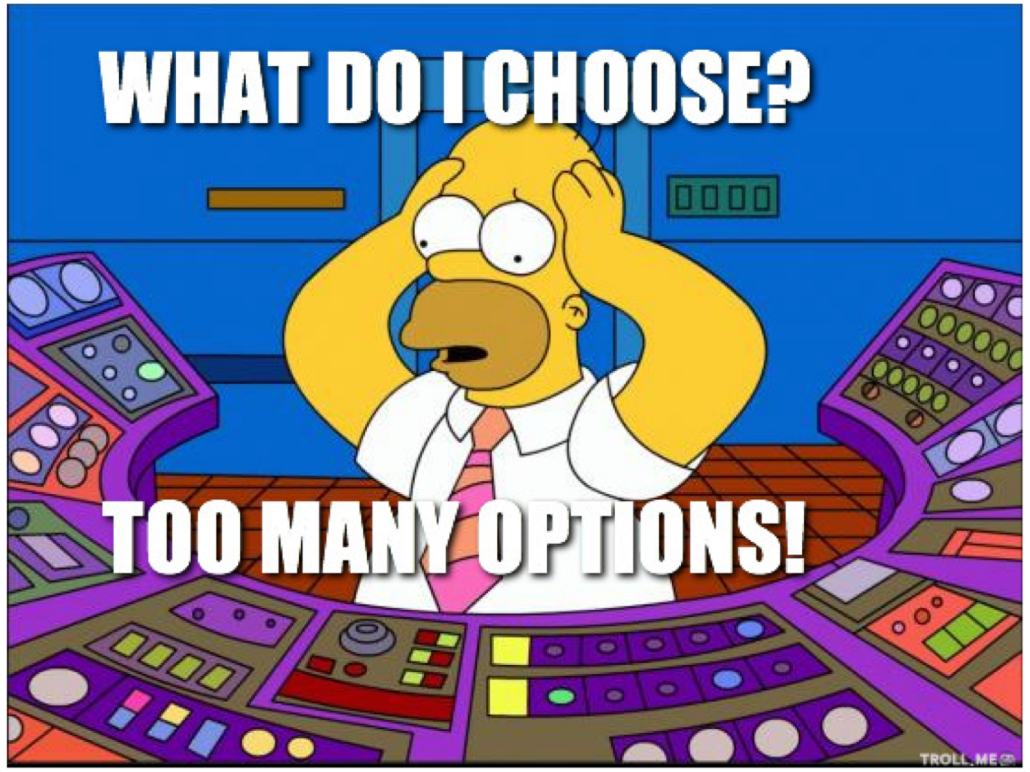
He initialization works slightly better than Glorot

Nothing to choose between Normal and Uniform

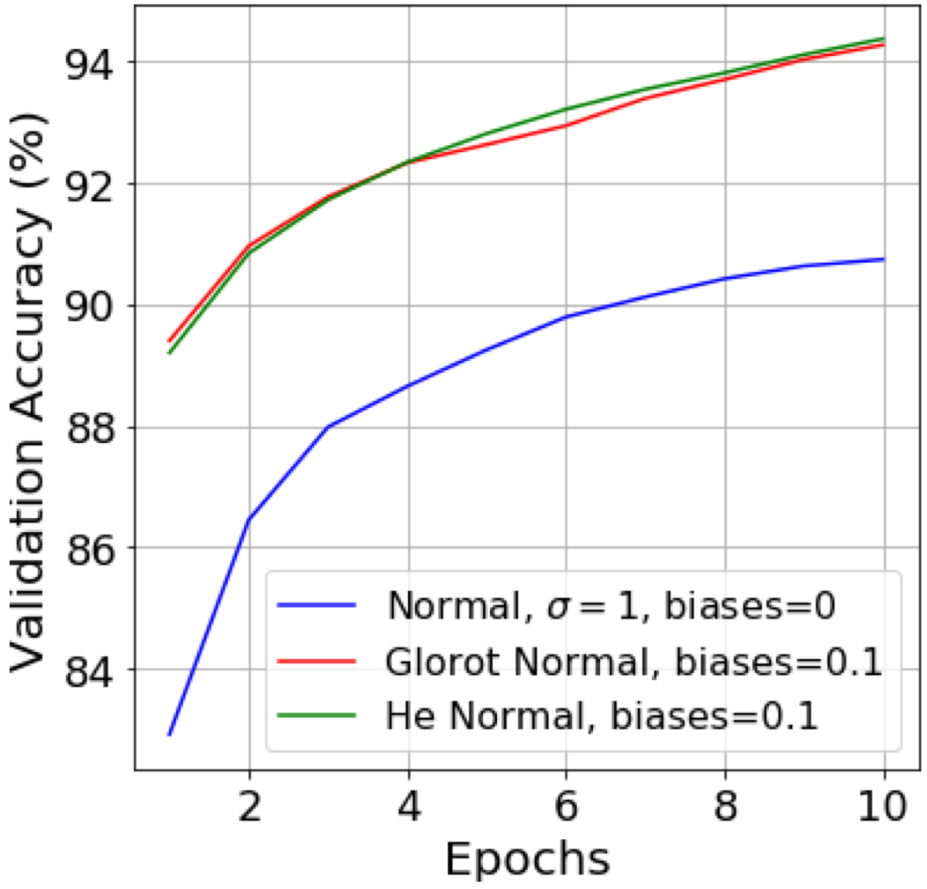
Biases not as important as weights

All zeros work OK

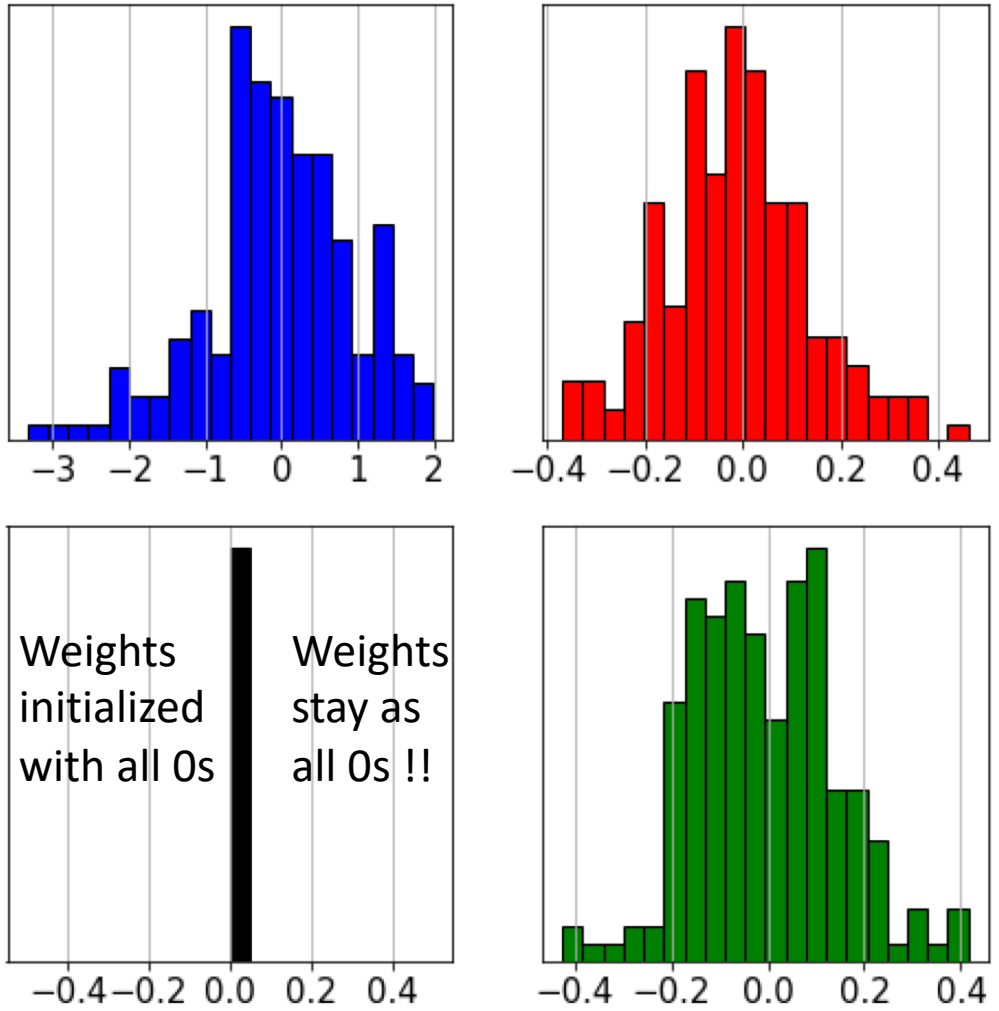
For ReLU, give a small positive value like 0.1
to biases to prevent neurons dying



Comparison of Initializers



MNIST [784,200,10]
Regularization: None



Histograms of a few weights in 2nd junction after training for 10 epochs

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Normalization

Imagine trying to predict how good a footballer is…

Feature	Units	Range
Height	Meters	1.5 to 2
Weight	Kilograms	50 to 100
Shot speed	Kmph	120 to 180
Shot curve	Degrees	0 to 10
Age	Years	20 to 35
Minutes played	Minutes	5,000 to 20,000
Fake diving?	--	*Yes / No*

Different features have very different scales

Input normalization (Preprocessing)

Say there are *n* total training samples, each with *f* features

Eg for MNIST: *n*=50000, *f*=784

$$\boldsymbol{X}_{\text{train}} = \begin{bmatrix} x_{11} & \cdots & x_{1f} \\ \vdots & \vdots & \vdots \\ x_{n1} & \cdots & x_{nf} \end{bmatrix}$$

Compute stats for each feature across all training samples

μ_i : Mean

σ_i : Standard deviation

M_i : Maximum value

m_i : Minimum value

for all *i* from 1 to *f*

Essential preprocessing

$$\boldsymbol{X}_{:,i} = \frac{\boldsymbol{X}_{:,i} - \mu_i}{\sigma_i}$$

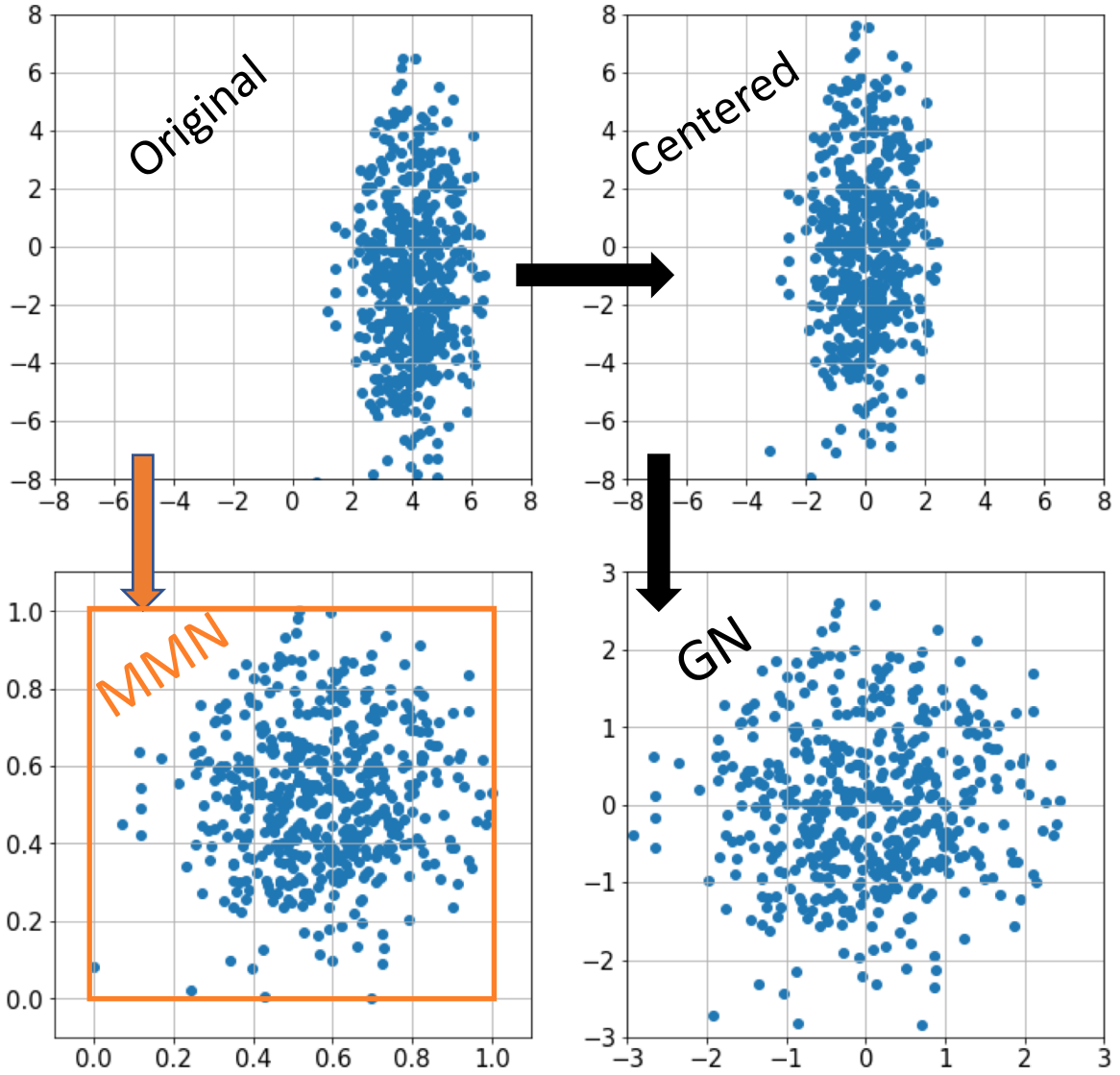
Gaussian normalization:

Each feature is a unit Gaussian

$$\boldsymbol{X}_{:,i} = \frac{\boldsymbol{X}_{:,i} - m_i}{M_i - m_i}$$

Minmax normalization:

Each feature is in [0,1]



Feature dependency and scaling

We want to make features independent

Why? Think dropout

$$\underset{f \times f}{\mathrm{Cov}(\boldsymbol{X})} = \frac{1}{n-1}\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}$$

$$\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{\mu}$$

We want to make this diagonal

In fact, we want to make this Identity
to make features have same scale

--> Whitening

Zero Components Analysis (ZCA) Whitening

$$\underset{f \times f}{\boldsymbol{Z}} = \sqrt{n-1}\left(\tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}\right)^{-\frac{1}{2}}$$

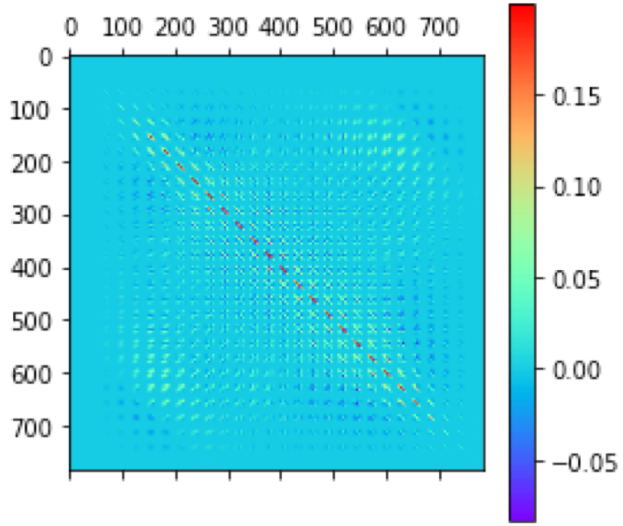
$$= \sqrt{n-1}\left(\boldsymbol{Q}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{Q}^T\right)$$

$$\boldsymbol{X}_{\mathrm{ZCA}} = \tilde{\boldsymbol{X}}\boldsymbol{Z}$$

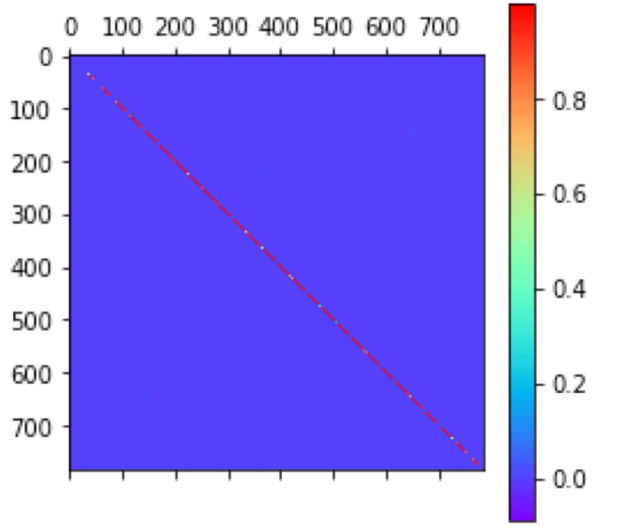
Then…

$$\frac{1}{n-1}\boldsymbol{X}_{\mathrm{ZCA}}^T \boldsymbol{X}_{\mathrm{ZCA}} = \boldsymbol{I}$$

Before



After

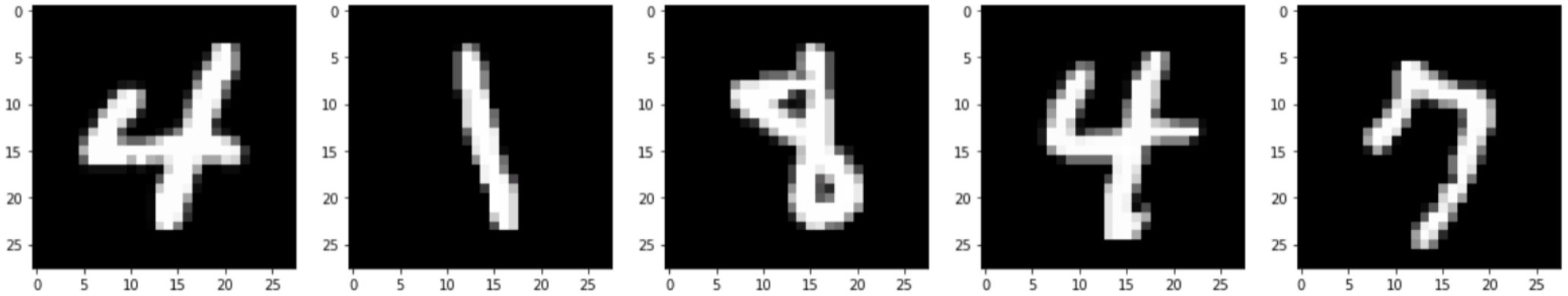


Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009

Dimensionality Reduction

Inputs can have a LOT of features

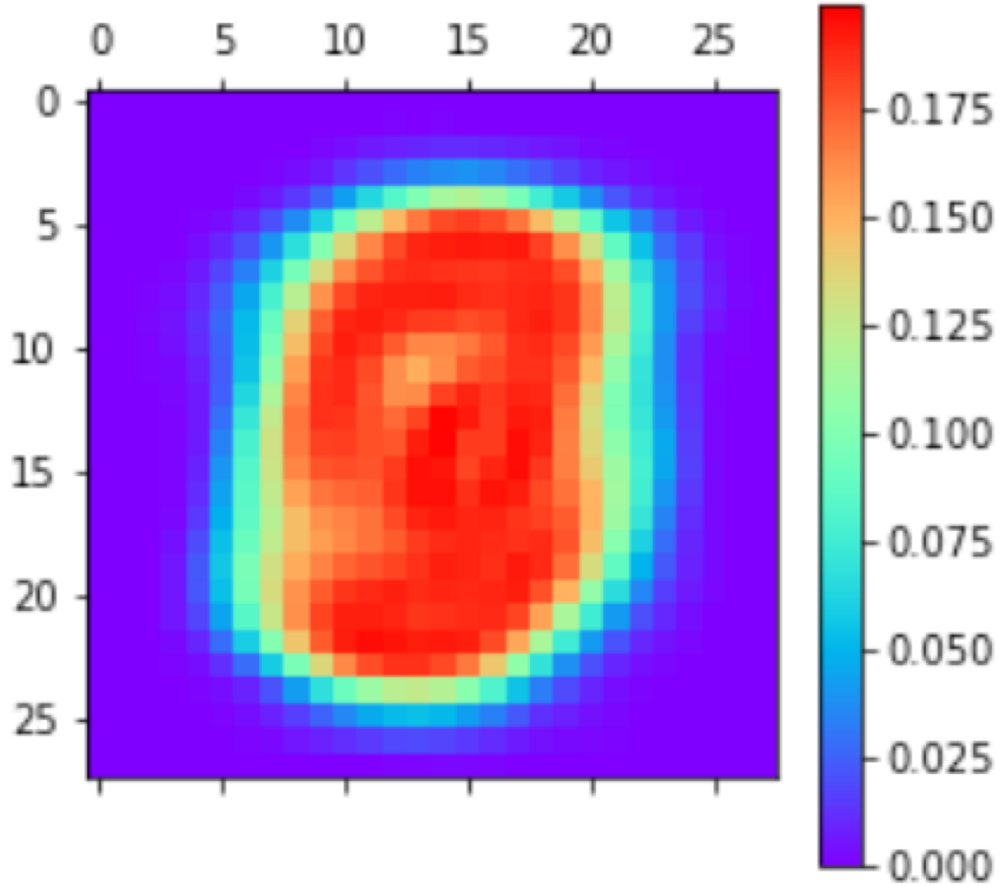
Eg: MNIST has 784 features, but we probably don't need the corner pixels



Simplify NN architecture by keeping only the essential input features

Principal Component Analysis (PCA)

Keep features with maximum variance
=> features which change the most



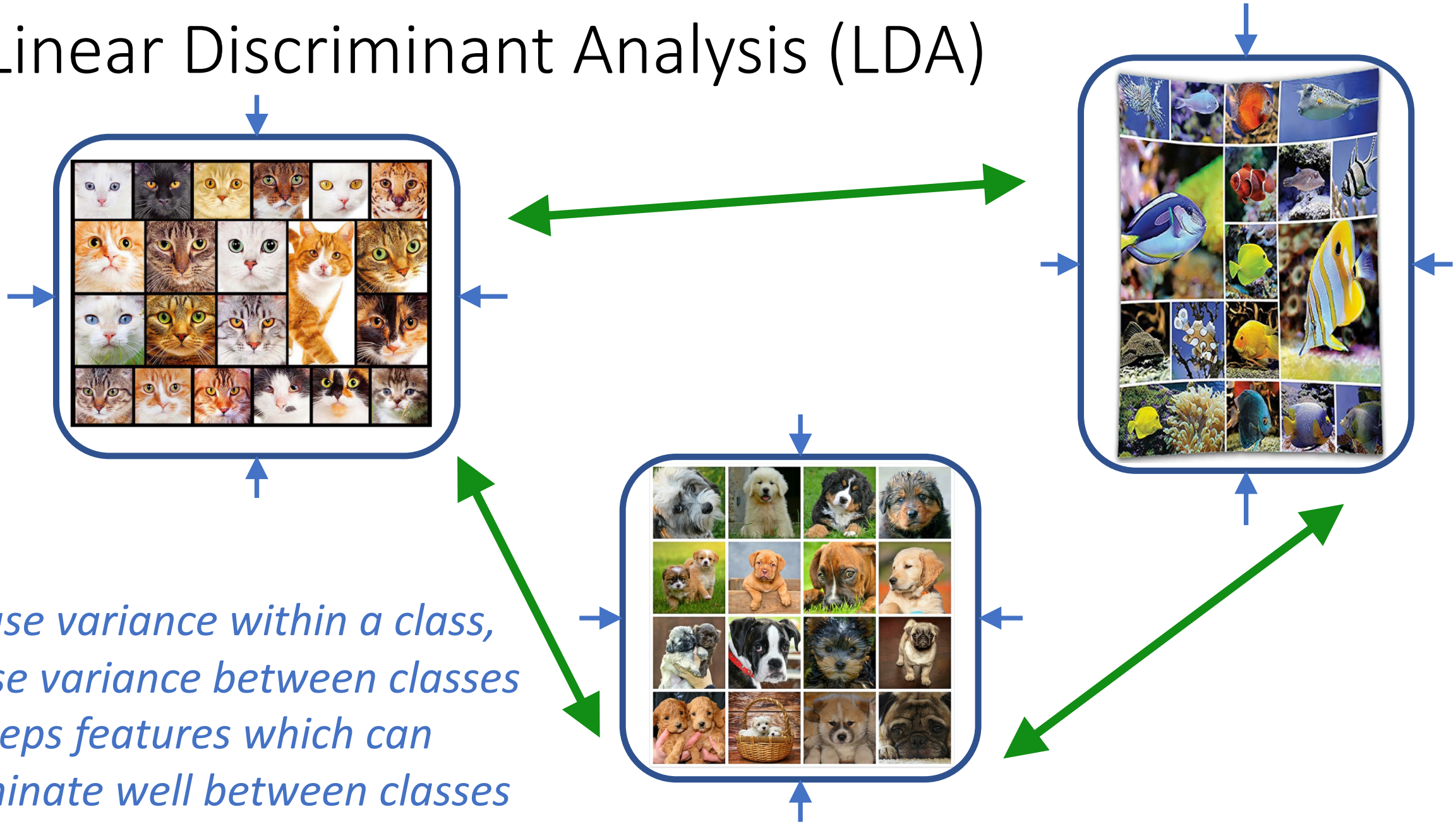
Assume sorted

$$\underset{f\times f}{\boldsymbol{U}} \quad \underset{f\times f}{\boldsymbol{\Sigma}} \quad \underset{f\times f}{\boldsymbol{V}} = \mathrm{SVD}\left\{\mathrm{Cov}(\boldsymbol{X})\right\}$$

$$\underset{n\times f'}{\boldsymbol{X}_{\mathrm{PCA}}} = \underset{n\times f}{\boldsymbol{X}} \quad \underset{f\times f'}{\boldsymbol{U}[:,:f']}$$

$f' < f$

Linear Discriminant Analysis (LDA)



Decrease variance within a class, increase variance between classes
LDA keeps features which can discriminate well between classes

Linear Discriminant Analysis (LDA)

Intra (within) class scatter

Minimize

$$\boldsymbol{S}_{\text{intra}} = \sum_{c=1}^{C} \left(\tilde{\boldsymbol{X}}_c - \boldsymbol{\mu}_c\right)^T \left(\tilde{\boldsymbol{X}}_c - \boldsymbol{\mu}_c\right)$$

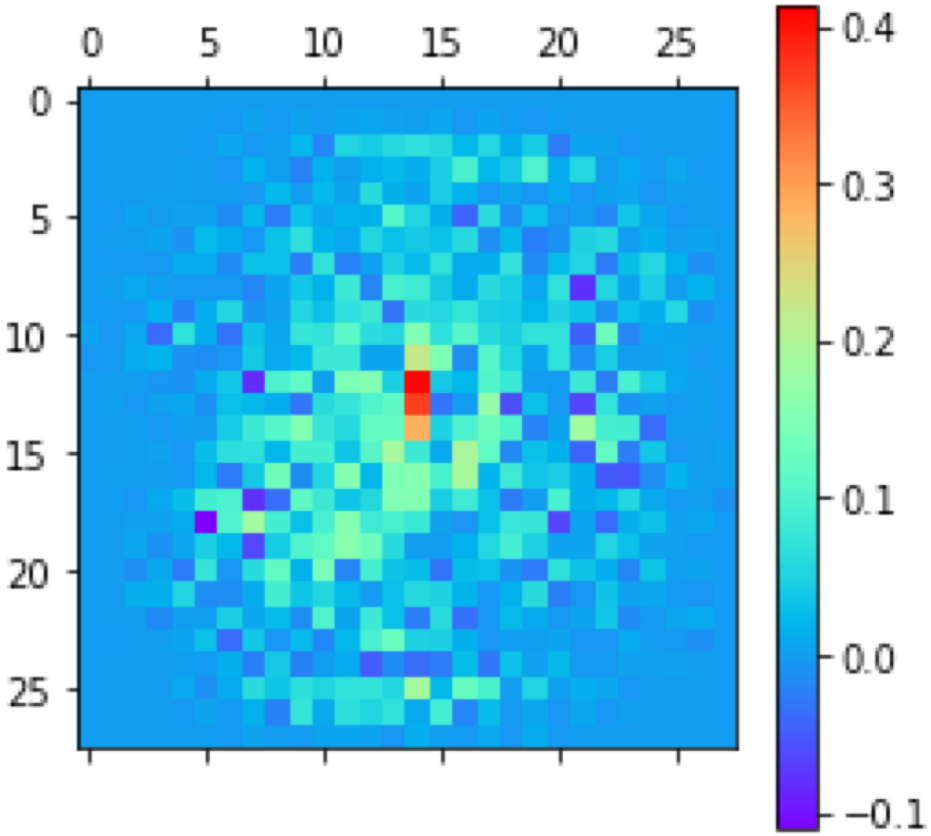
Inter (between) class scatter

Maximize

$$\boldsymbol{S}_{\text{inter}} = \sum_{c=1}^{C} N_c \left(\boldsymbol{\mu}_c - \boldsymbol{\mu}\right) \left(\boldsymbol{\mu}_c - \boldsymbol{\mu}\right)^T$$

$$\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V} = \text{SVD}\left\{\boldsymbol{S}_{\text{intra}}^{-1}\boldsymbol{S}_{\text{inter}}\right\}$$

$$\underset{n\times f'}{\boldsymbol{X}_{\text{LDA}}} = \underset{n\times f}{\boldsymbol{X}} \quad \underset{f\times f'}{\boldsymbol{U}[:, : f']}$$

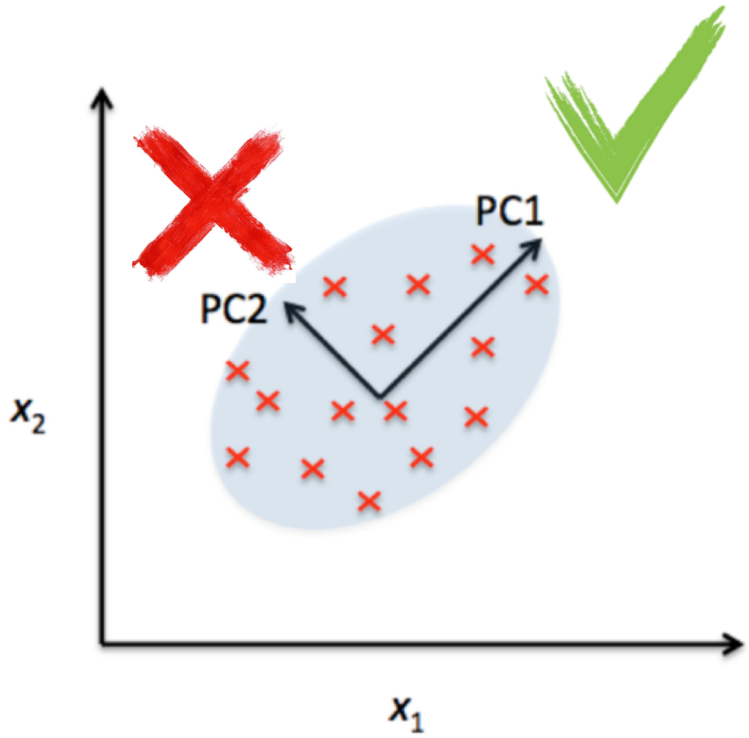


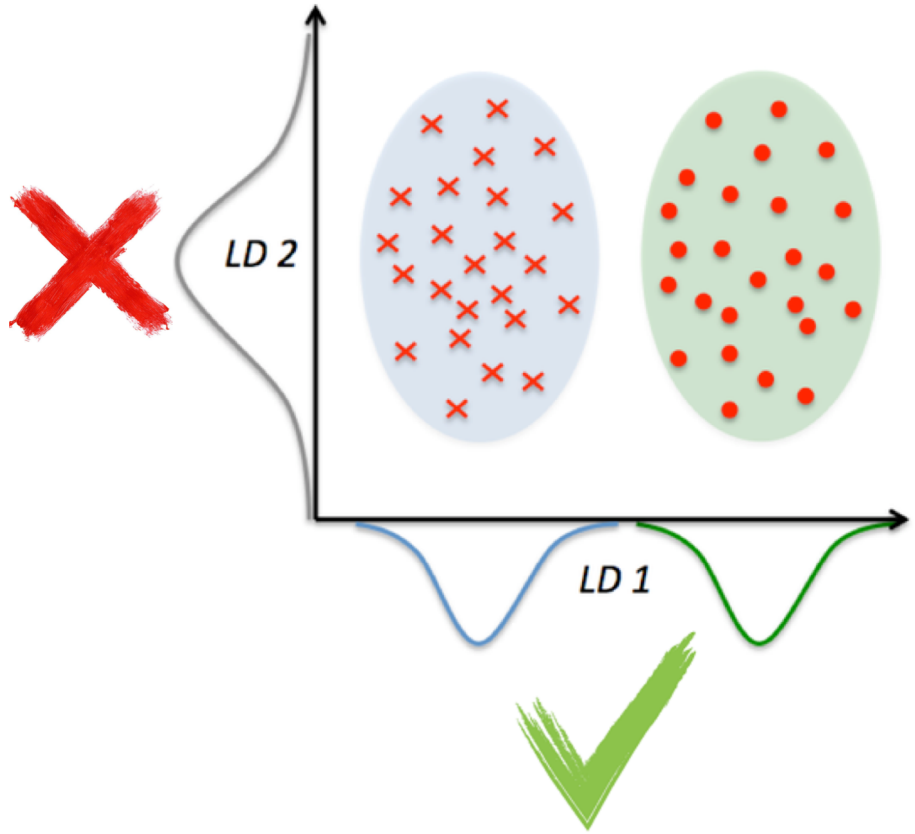
C: Number of classes

N_c: Number of samples in class c

μ_c: Feature mean of class c

PCA vs LDA





PCA is unsupervised - just looks at data

LDA is supervised, looks at classes

Remember…

All statistics should be completed on training data only
Then apply them to validation and test data

Example:

$$\boldsymbol{X}_{\text{train}} \rightarrow \boldsymbol{U}_{\text{train}}$$

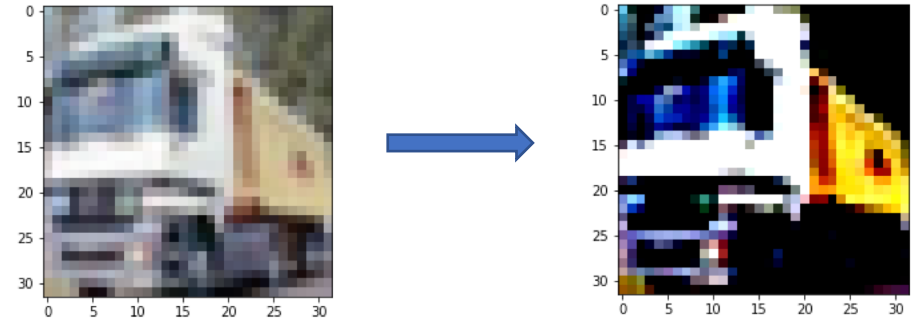
$$\boldsymbol{U}_{\text{train}} \rightarrow \boldsymbol{X}_{\text{train}_{\text{PCA}}}$$

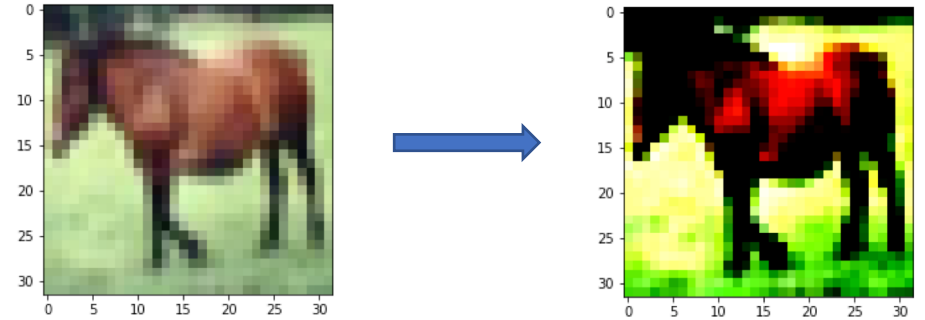
$$\boldsymbol{U}_{\text{train}} \rightarrow \boldsymbol{X}_{\text{val}_{\text{PCA}}}$$

$$\boldsymbol{U}_{\text{train}} \rightarrow \boldsymbol{X}_{\text{test}_{\text{PCA}}}$$

Global Contrast Normalization (GCN)

Increase contrast (standard deviation of pixels) of each image, one at a time





Different from other normalizations which operate on all training images together

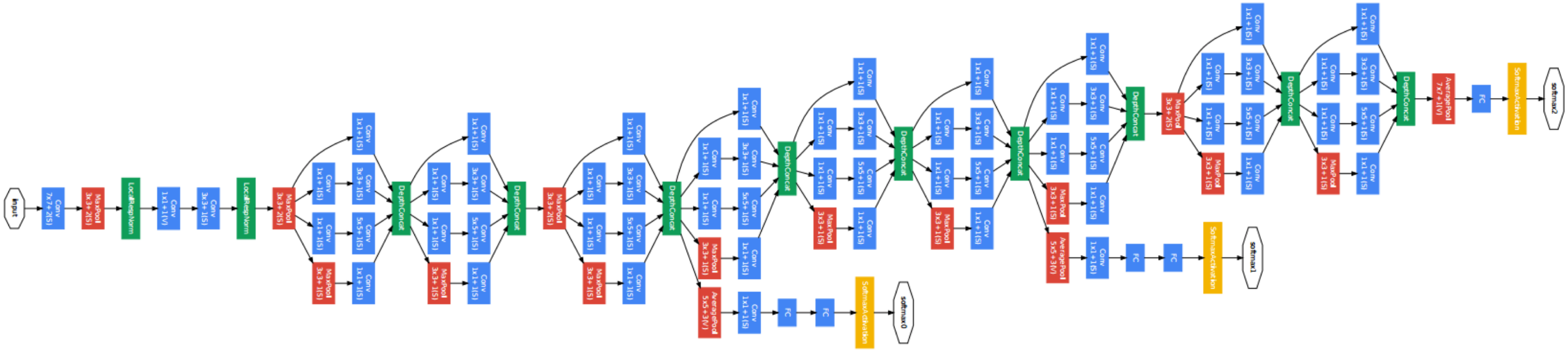
$$\boldsymbol{x} : \text{Single image}$$

$$x_{\text{pixel}} = \frac{x_{\text{pixel}} - \mu_{(\text{all pixels in } \boldsymbol{x})}}{\sigma_{(\text{all pixels in } \boldsymbol{x})}}$$

Batch normalization

Input normalization scales the input features. But what about internal activations? Those are also input features to the layers after them

Internal activations get shifted to crazy values in deep networks



C. Szegedy, W. Liu, Y. Jia et al., “Going deeper with convolutions,” in IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

Batch normalization

Re-normalize internal values using a minibatch, then apply some trainable (μ,σ) = (β,γ) to them

$$\mu = \frac{1}{M}\sum_{i=1}^{M} x_i$$

x can be any value inside any layer of the network

$$\sigma^2 = \frac{1}{M}\sum_{i=1}^{M} (x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sigma + \epsilon}$$

Normalize

ϵ is just a small number to prevent division by 0. Can be machine epsilon

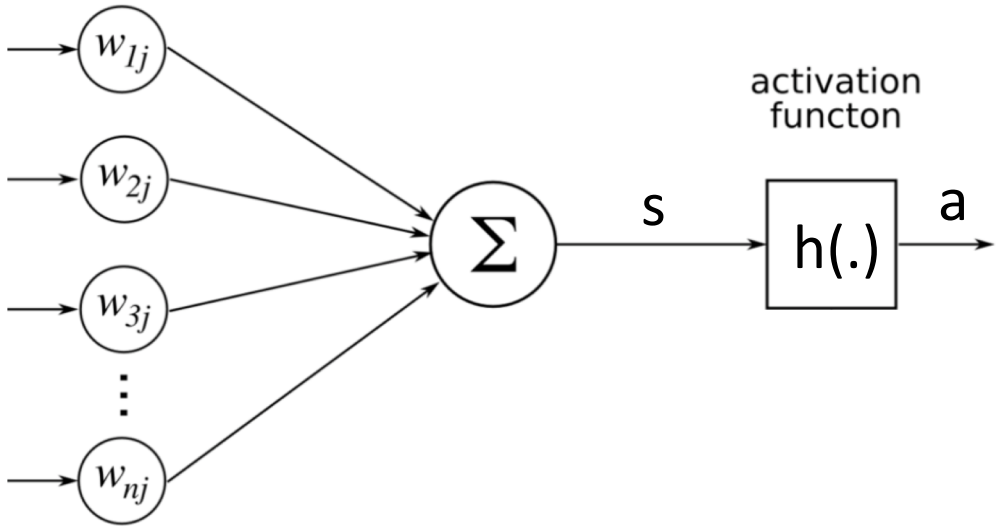
$$x_i \leftarrow \gamma\hat{x}_i + \beta$$

Scale the normalized values and substitute

Sergey Ioffe, Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. Proc. 32nd ICML, 37, pp 448-456, 2015.

Where to apply Batch Normalization?

Option 1: Before activation (at *s*)
Option 2: After activation (at *a*)



Original paper recommends before activation.
This means scaling the input to ReLU, then ReLU can do its job by discarding anything <=0.

Scaling after ReLU doesn't make sense for β since it can be absorbed by the parameters of the **next** layer.

But… People still argue over this. I have done both and saw almost no difference.

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{W}(\gamma \boldsymbol{a} + \beta) + \boldsymbol{b} \\ &= (\boldsymbol{W}\gamma)\boldsymbol{a} + (\boldsymbol{W}\beta + \boldsymbol{b}) \\ &= \tilde{\boldsymbol{W}}\boldsymbol{a} + \tilde{\boldsymbol{b}} \end{aligned}$$

Outline

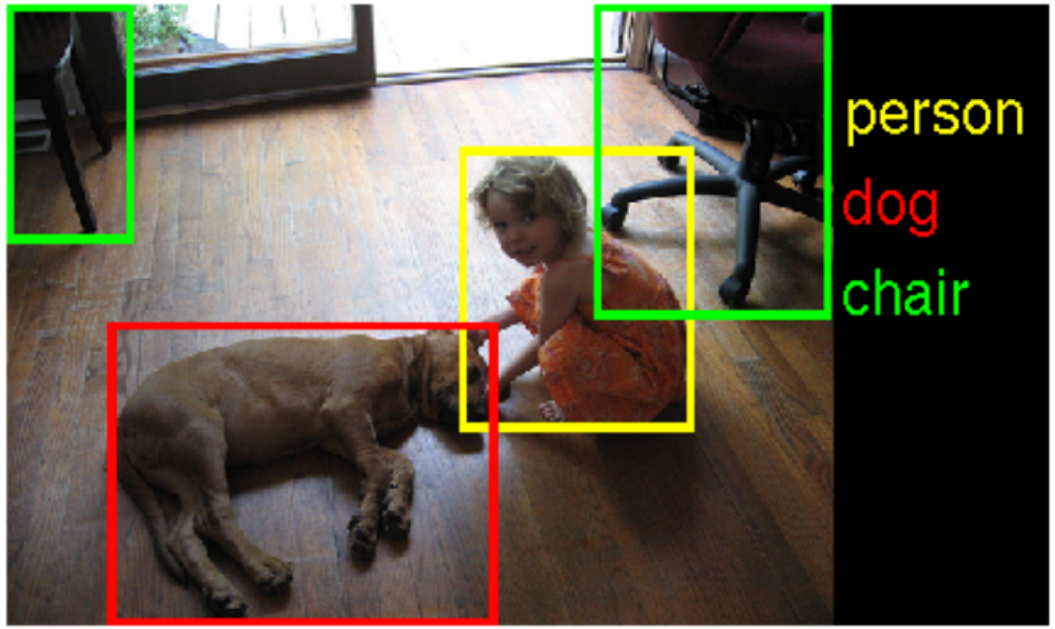
- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

Data handling

- ✓ Collection
- ✓ Contamination
- ✓ Public datasets
- ✓ Synthetic data
- ✓ Augmentation

Collection and labeling

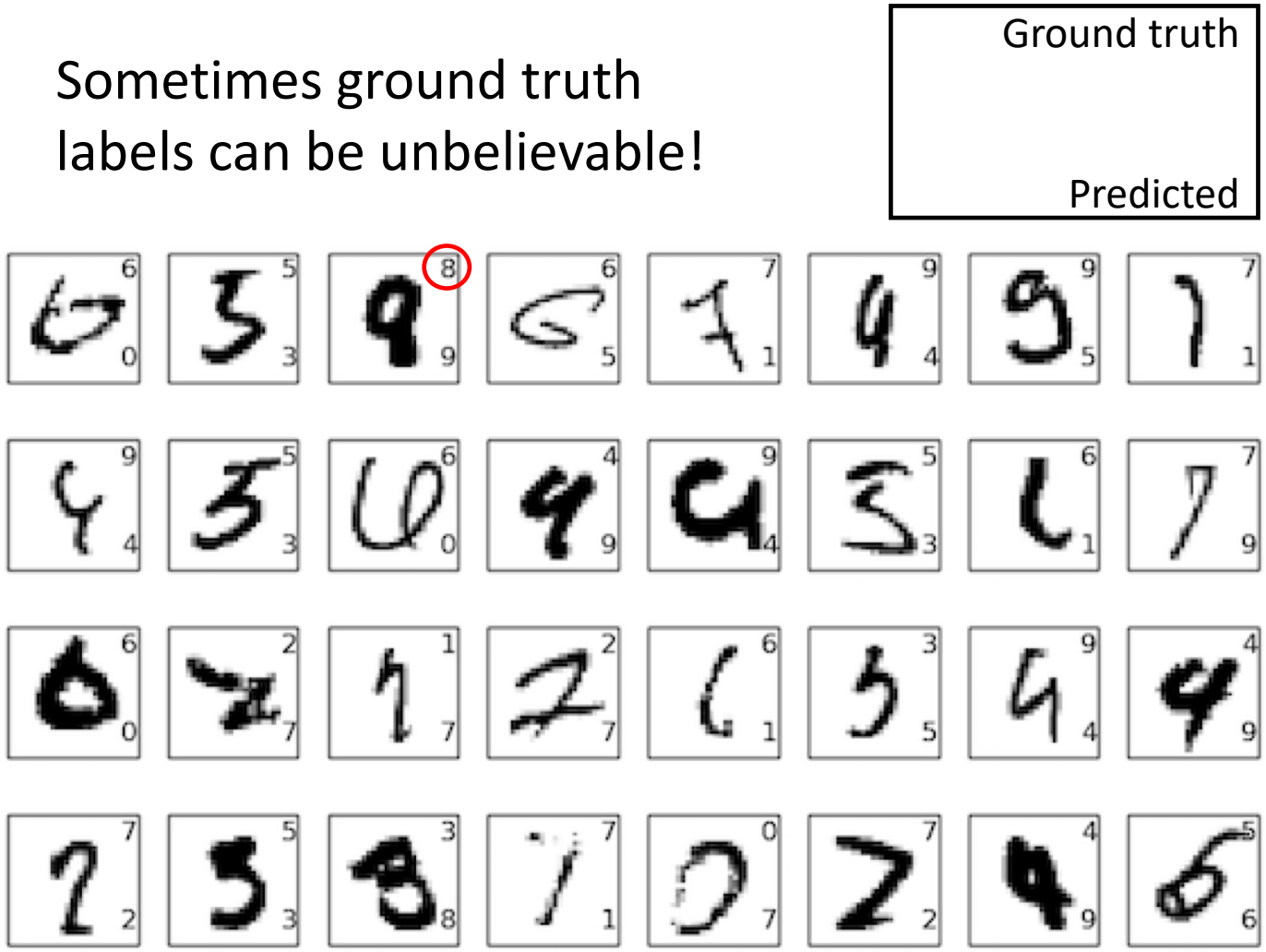
- Collected from real world sources - internet databases, surveys, paying people, public records, hospitals, etc…
- Labeled manually, usually crowd-sourced (Mechanical Turk)



Examples from Imagenet
http://www.image-net.org

Contamination

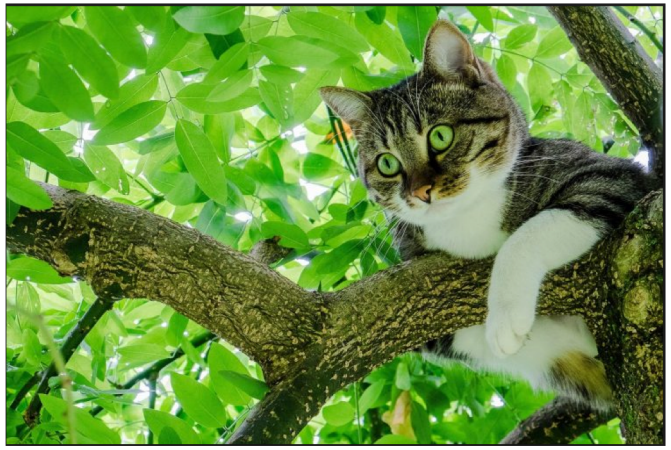
Sometimes ground truth labels can be unbelievable!



HW1: Human vs Computer, Binary
Did anyone cheat by using numpy? ;-)

Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

Neural Networks are Lazy



Train

Test

Network output: Cat

For the NN, green background = cat

Missing Entries

Example: Adult Dataset

https://archive.ics.uci.edu/ml/datasets/adult

Features:

- Age
- Working class
- Education
- Marital status
- Occupation
- Race
- Sex
- Capital gain
- Hours per week
- Native country

Label: Income level

Missing at Random: Remove the sample

Missing not at Random: Removing sample may produce bias

Example: People with low education level don't reveal

Other ways:

Fill missing numerical values with mean or median. Eg: Age = 40

Fill missing categorical values with mode. Eg: Education = College

Small Dataset Size

Example: Wisconsin Breast Cancer Dataset

https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Features (image of breast cells):

- Radius
- Perimeter
- Smoothness
- Compactness
- Symmetry

Labels:

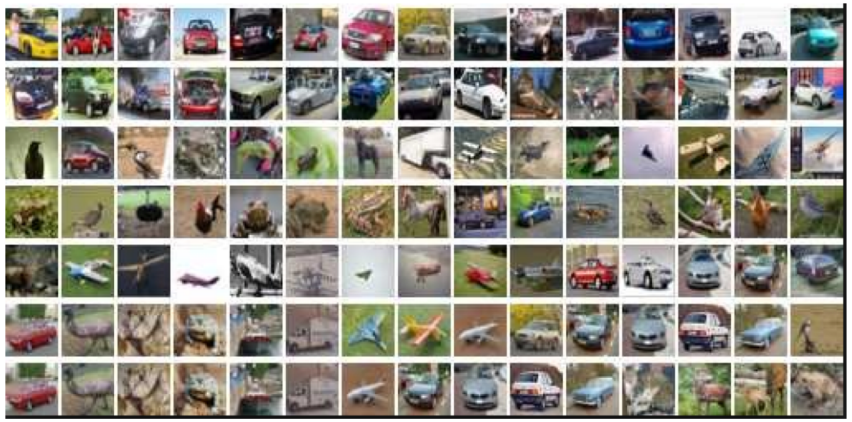
- Malignant
- Benign

Original dataset: 699 samples with missing values
Modified dataset: 569 samples

Such small datasets generally not suited for neural networks due to overfitting

Commonly used Image Datasets

- MNIST (MLP / CNN):
  - 28x28 images, 10 classes
  - Initial benchmark
  - If you're writing papers, don't just do MNIST!
  - SOTA testacc: >99%
  - **Harder Variation: Fashion MNIST**
- CIFAR-10, -100 (CNN):
  - 32x32x3 images (RGB), 10 or 100 classes
  - Widely used benchmark
  - SOTA testacc: ~97%, ~84%



Commonly used Image Datasets

- Imagenet (CNN):
  - Overall database has >14M images
  - ILSVRC challenge has >1M images, 224x224x3, 1000 classes
  - Widely used benchmark, don’t attempt without enough computing power!
  - Alexnet top-1, top-5 testacc: 62.5%, 83% (not SOTA, but very famous!)
  - **Simpler variation: Tiny Imagenet**
- Microsoft Coco (CNN / Encoder-Decoder):
  - 330k images, 80 categories
  - Segmentation, object detection
- Street View House Numbers (CNN / MLP):
  - Think MNIST, but more real-world, harder, bigger
  - SOTA testacc > 98%

Other Commonly used Datasets

- TIMIT (Preprocessing + MLP + Decoder):
  - Automatic Speech recognition (Guest lecture coming up on 2/27)
  - Speaker and phoneme classification
- Reuters (MLP):
  - Classify articles into news categories
  - Multiple / tree structured labels
- IMDB Reviews:
  - Natural language processing
- YouTube-8M:
  - Annotate videos

Source for Datasets

- Kaggle: https://www.kaggle.com/datasets
- UCI ML repository: https://archive.ics.uci.edu/ml/datasets.html
- Google: https://toolbox.google.com/datasetsearch
- Amazon Web Services: https://registry.opendata.aws/

Synthetic Datasets

Data that is artificially manufactured (using algorithms), rather than generated by real-world events

- Tunable algorithms to mimic real-world as required
- Cheap to produce large amounts of data
- We created data on Morse code symbols with tunable classification difficulty: https://github.com/usc-hal/morse-dataset
- Can be used to augment real-world data

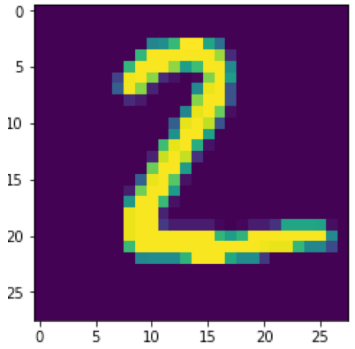
S. Dey, K. M. Chugg, and P. A. Beerel, "Morse code datasets for machine learning," in Proc. 9th Int. Conf. Computing, Communication and Networking Technologies (ICCCNT), Jul 2018, pp. 1–7. *Won Best Paper Award.*

Data Augmentation

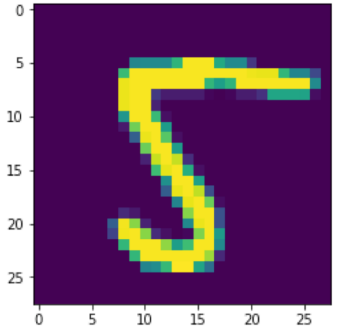
More training results in overfitting
The solution: Create more data!

Eg: More MNIST images can simulate more different ways of writing digits
If the network sees more, it can learn more

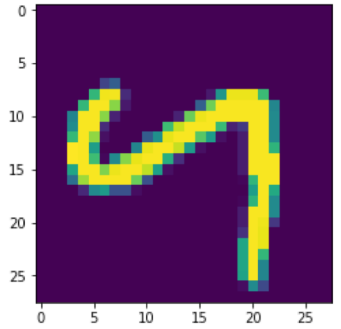
Data Augmentation Examples for Images



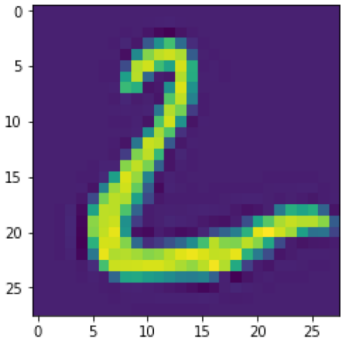
Original



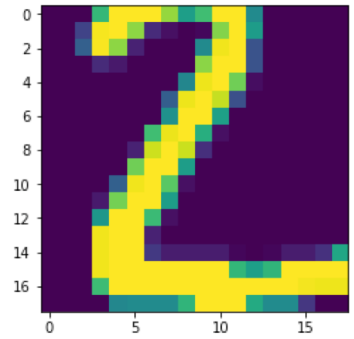
Flip Top-Bottom
Do NOT do for digits



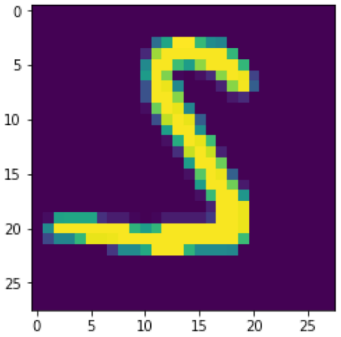
Transpose



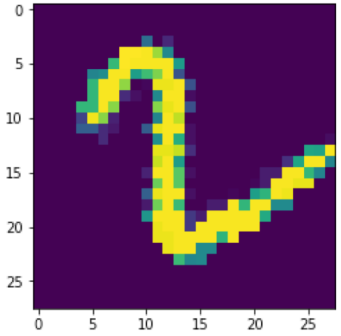
Elastic Transform



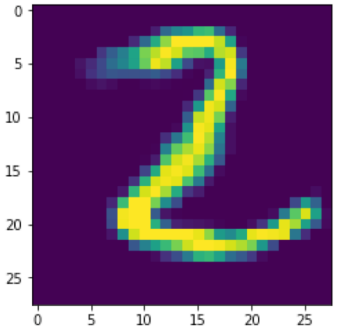
Cropped



Flip Left-Right



Rotate 30°



Simard, Steinkraus and Platt, "Best Practices for Convolutional Neural Networks applied to Visual Document Analysis", in Proc. Int. Conf. on Document Analysis and Recognition, 2003.

Data Augmentation in Python

I use the Pillow Imaging library

```
>> pip install Pillow

from PIL import Image
im = Image.fromarray((x.reshape(28,28)*255).astype('uint8'), mode='L’)
im.show()
im.transpose(Image.FLIP_LEFT_RIGHT)
im.transpose(Image.FLIP_TOP_BOTTOM)
im.transpose(Image.TRANSPOSE)
im.rotate(30)
im.crop(box=(5,5,23,23)) #(left,top,right,bottom)
```

Outline

- Activation functions
- Cost function
- Optimizers
- Regularization
- Parameter initialization
- Normalization
- Data handling
- Hyperparameter selection

What is a Hyperparameter?

Anything which the network does NOT learn,
i.e. its value is adjusted by the user

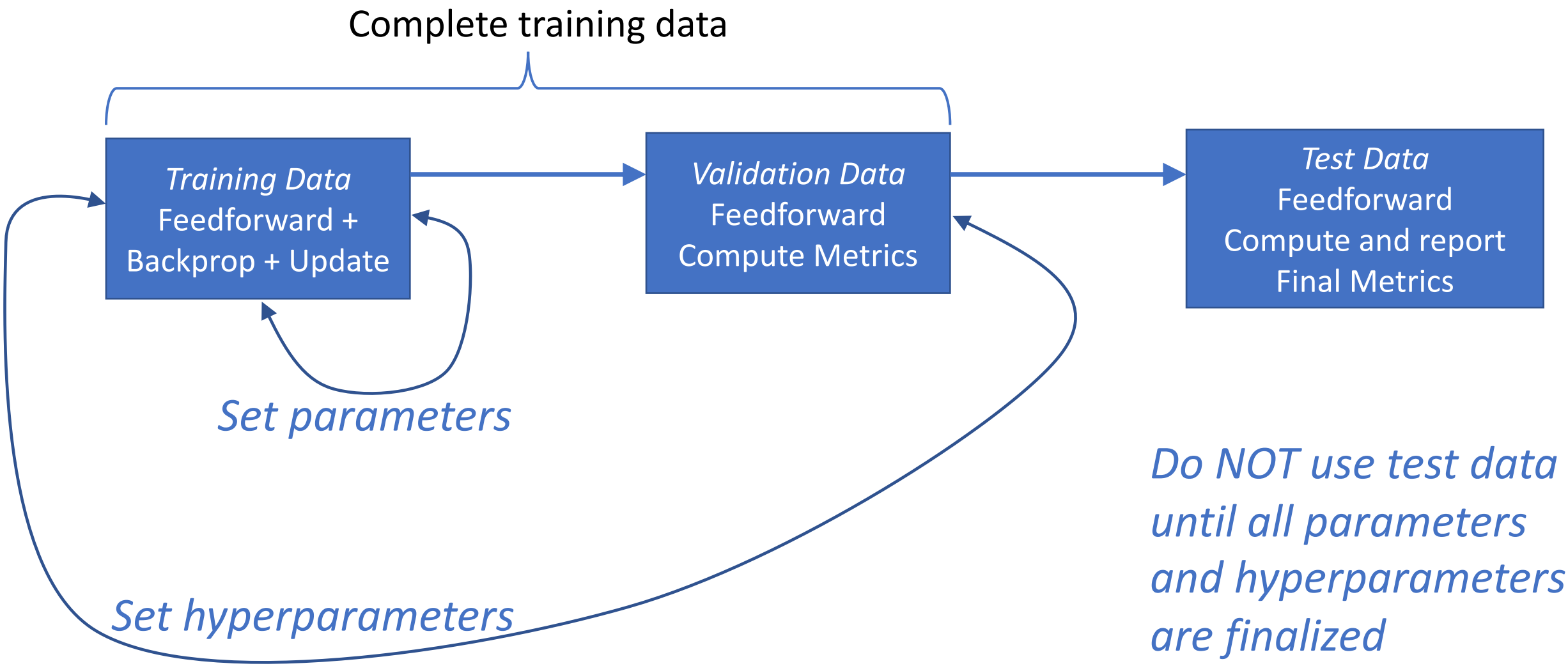
Continuous Examples:

- Learning rate η
- Momentum α
- Regularization λ

Discrete Examples:

- What optimizer: *SGD, Adam, RMSprop, …?*
- What regularization: *L2, L1, both, …?*
- What initialization: *Glorot, He, normal, uniform, …?*
- Batch size M: *32, 64, 128, …?*
- Number of MLP hidden layers: *1,2,3,…?*
- How many neurons in each layer?
- What kinds of data augmentation?

Using data properly



Do NOT use test data until all parameters and hyperparameters are finalized

Cross-Validation

- Divide the dataset into k parts.
- Use all parts except i for training, then test on part i.
- Repeat for all i from 1 to k.
- Average all k test metrics to get final test metric.

Useful when dataset is small

Generally not needed for NN problems

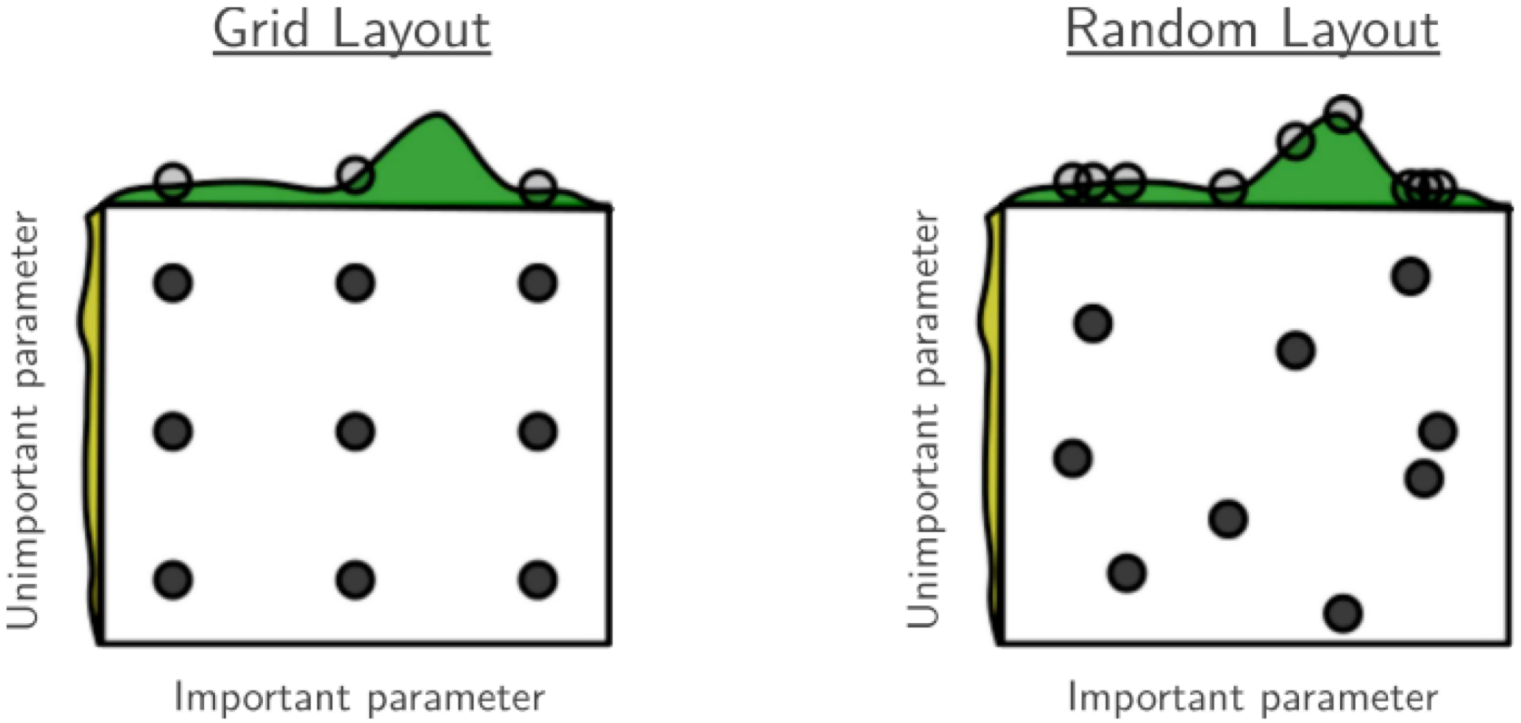
Hyperparameter Search Strategies

Grid search for (η,α):

(0.1,0.5), (0.1,0.9), (0.1,0.99),
(0.01,0.5), (0.01,0.9), (0.01,0.99),
(0.001,0.5), (0.001,0.9), (0.001,0.99)

Random search for (η,α):

(0.07,0.68), (0.002,0.94), (0.008,0.99),
(0.08,0.88), (0.005,0.62), (0.09,0.75),
(0.14,0.81), (0.006,0.76), (0.01,0.8)



Random search samples more values of the important hyperparameter

Pic courtesy: J. Bergstra, Y. Bengio, "Random Search for Hyperparameter Optimization," Journal of Machine Learning Research, vol. 13, pp. 281–305, 2012

Hyperparameter Optimization Algorithm : Tree-Structured Parzen Estimator (TPE)

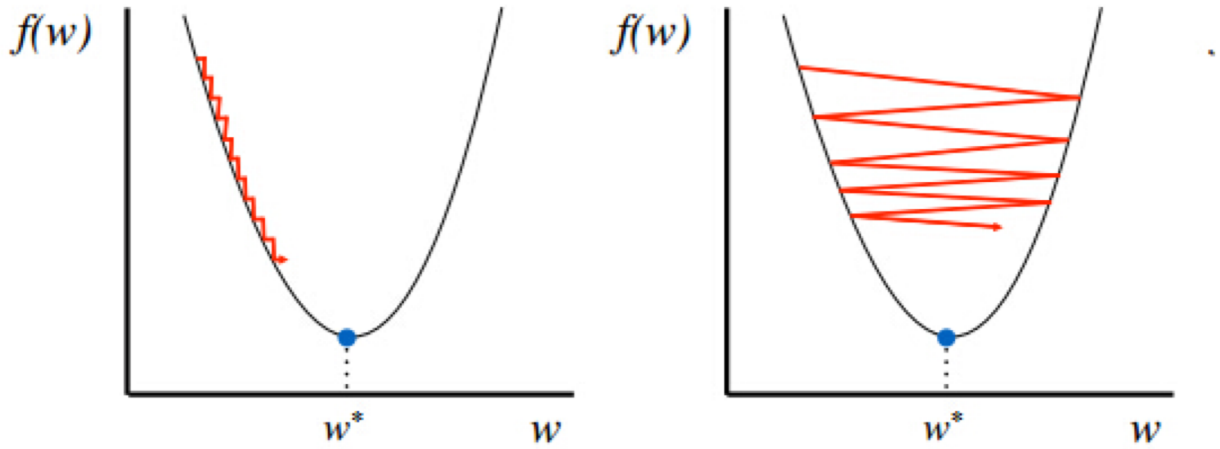
Potential project topic

1. Decide initial probability distribution for each hyperparameter
   → Example: $\eta \sim \text{logUniform}(10^{-4},1)$; $\lambda \sim \text{logUniform}(10^{-6},10^{-3})$
2. Set some performance threshold
   → Example: MNIST validation accuracy after 10 epochs = 96%
3. Sample multiple *hyp_sets* from the distributions
4. Train, and compute validation metrics for each *hyp_set*
5. Update distribution and threshold:
   → If performance of a *hyp_set* is worse than threshold: Discard
   → If performance of a *hyp_set* is better than threshold: Change distribution accordingly
6. Repeat 3-5

J. Bergstra, R. Bardenet, Y. Bengio, B. Kegl, "Algorithms for Hyper-Parameter Optimization", Proc. NIPS, pp 2546-2554, 2011.

Learning rate schedules

When using simple SGD, start with a big learning rate, then decrease it for smoother convergence



Pic courtesy: http://prog3.com/sbdm/blog/google19890102/article/details/50276775

Learning rate schedules

$$\eta_{t+1} = \frac{\eta_0}{1 + kt}$$

Initial η

Epochs

Fractional decay

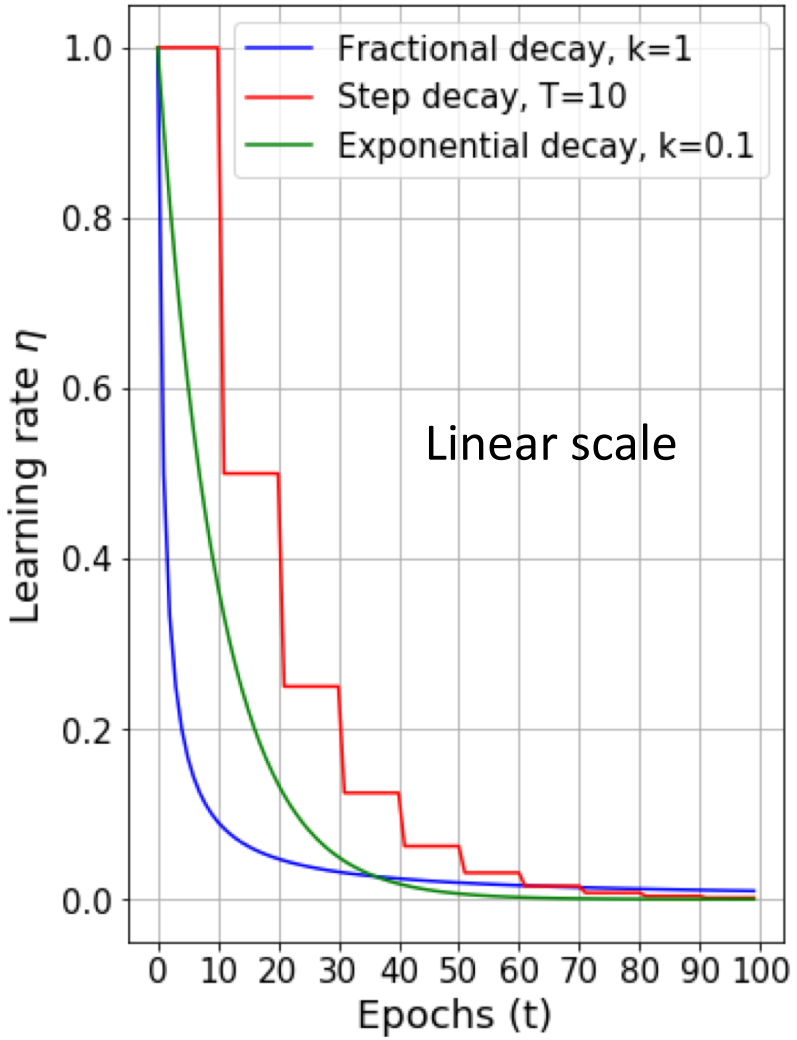
Used in Keras

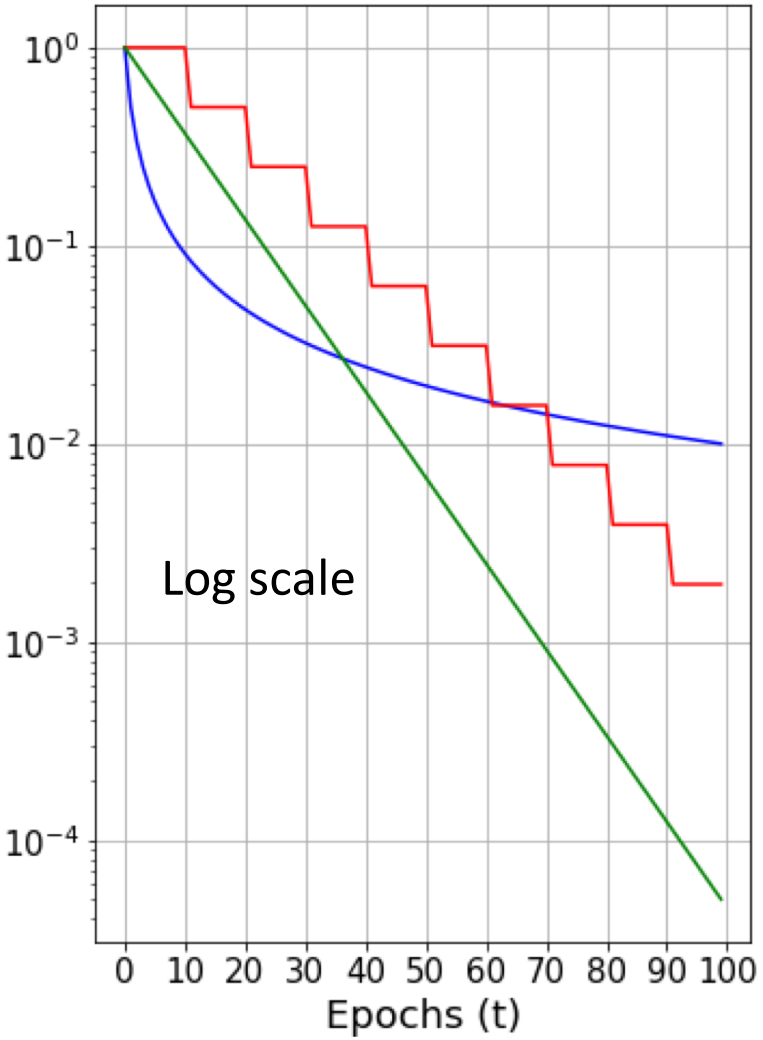
$$\eta_{T+1} = \eta_0 \left(\frac{1}{2}\right)^{\lfloor t/T \rfloor}$$

Step decay

$$\eta_{t+1} = \eta_0 e^{(-kt)}$$

Exponential decay





Other Learning rate strategies

Potential project topic

Warmup: Increase η for a few epochs at the beginning. Works well for large batch sizes.

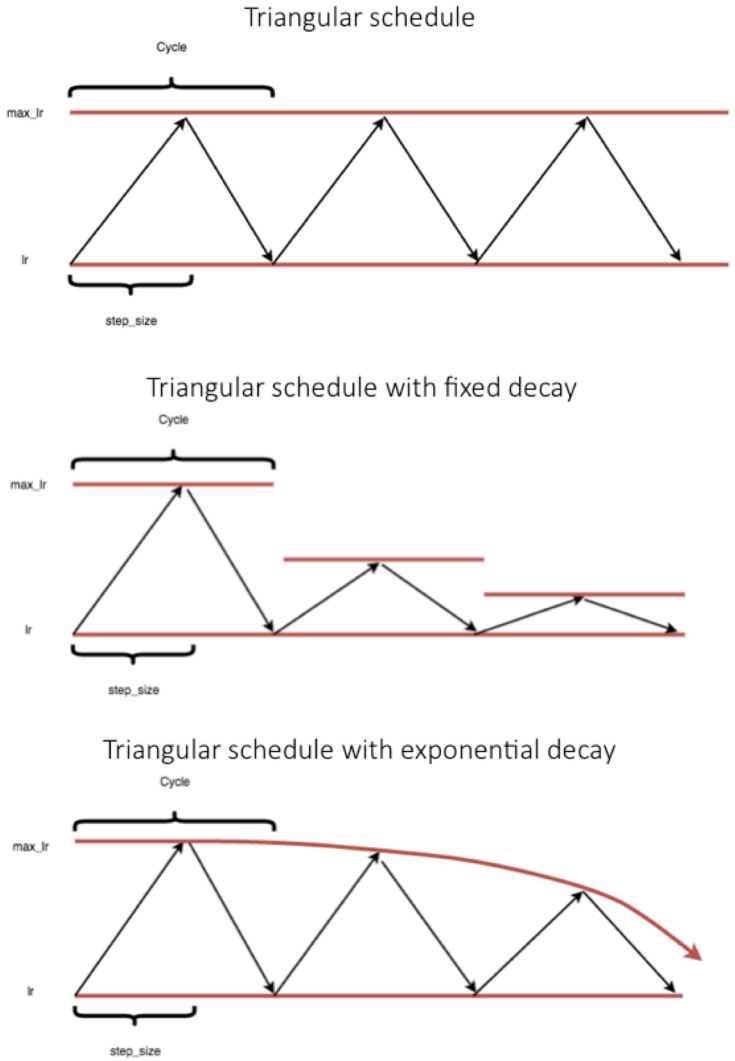
Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, Kaiming He. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". arXiv:1706.02677

Cosine Scheduling

I. Loshchilov, F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", Proc. ICLR 2017.

Triangular Scheduling

L. N. Smith, "Cyclical Learning Rates for Training Neural Networks", arXiv:1506.01186
Pic courtesy https://www.jeremyjordan.me/nn-learning-rate/



That’s all Thanks!