Tutorial on:

Optimization I

(from a deep learning perspective)

Jimmy Ba

UNIVERSITY OF TORONTO

VECTOR INSTITUTE | INSTITUT VECTEUR

Outline

- Random search v.s. gradient descent
- Finding better search directions
- Design “white-box” optimization methods to improve computation efficiency

Neural networks

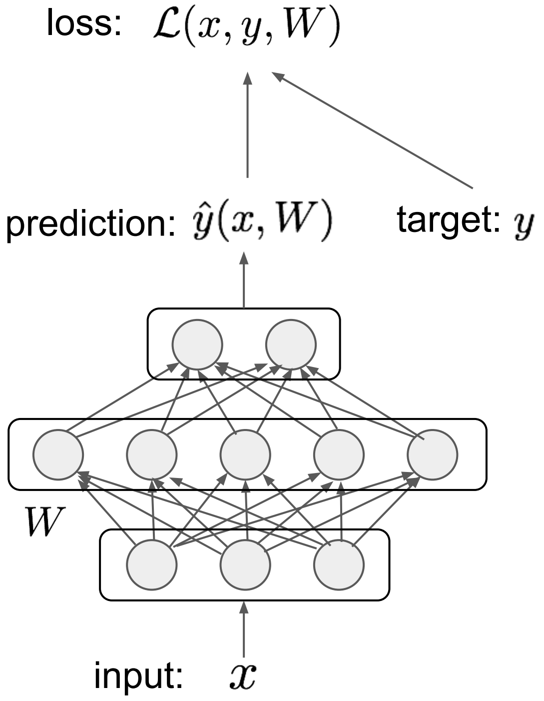
Consider an input and output pair (x,y) from a set of N training examples.

Neural networks are parameterized functions that map input x to output prediction y using d number of weights $W \in \mathbb{R}^d$.

Compare the output prediction with the correct answer to obtain a loss function for the current weights:

Averaged loss: $\bar{\mathcal{L}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x^{(i)}, y^{(i)}, W) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(W)$

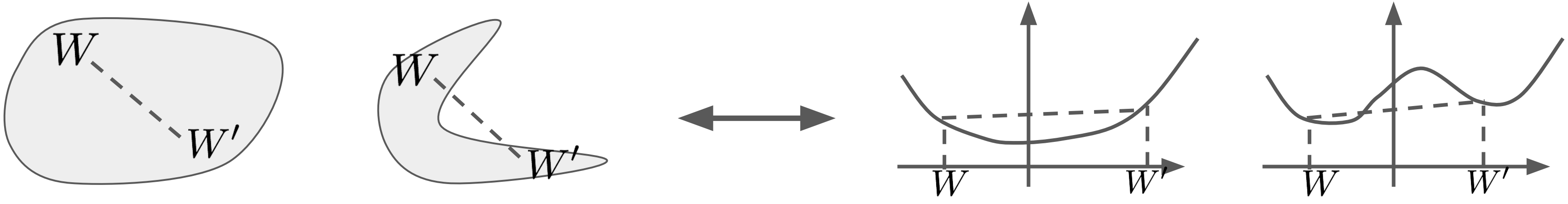
Shorthand notation



Why is learning difficult

Neural networks are non-convex.

Many local optimums



convex set

non-convex set

convex function

non-convex function

$$\forall \lambda \in [0,1], \lambda W + (1-\lambda) W' \in \mathcal{W}$$

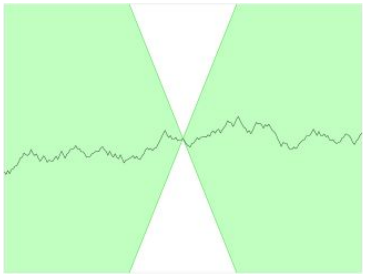
$$\forall \lambda \in [0,1], W, W' \in \mathcal{W},$$

$$f(\lambda W + (1-\lambda) W') \leq \lambda f(W) + (1-\lambda) f(W')$$

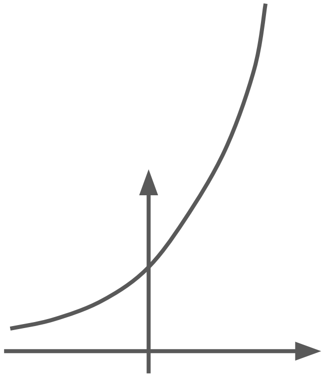
Why is learning difficult

Neural networks are non-convex.

Neural networks are not smooth.



Lipschitz-continuous



Not Lipschitz-continuous

$$|f(W) - f(W')| \leq L\|W - W'\|$$

The change in f is smooth and bounded.

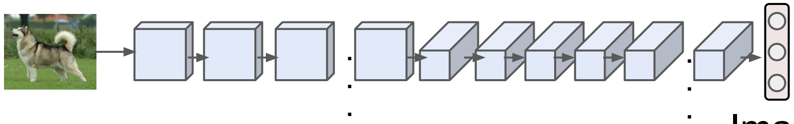
L is the Lipschitz constant

Why is learning difficult

Neural networks are non-convex.

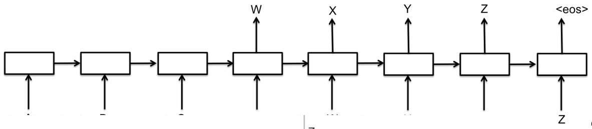
Neural networks are not smooth.

Millions of trainable weights.



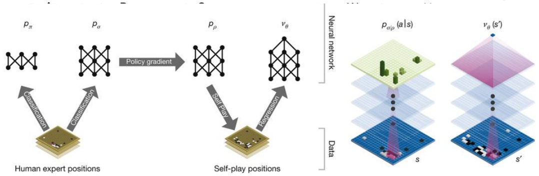
ImageNet

~5 million



Neural Machine Translation

~100 million



AlphaGo

~40 million ?

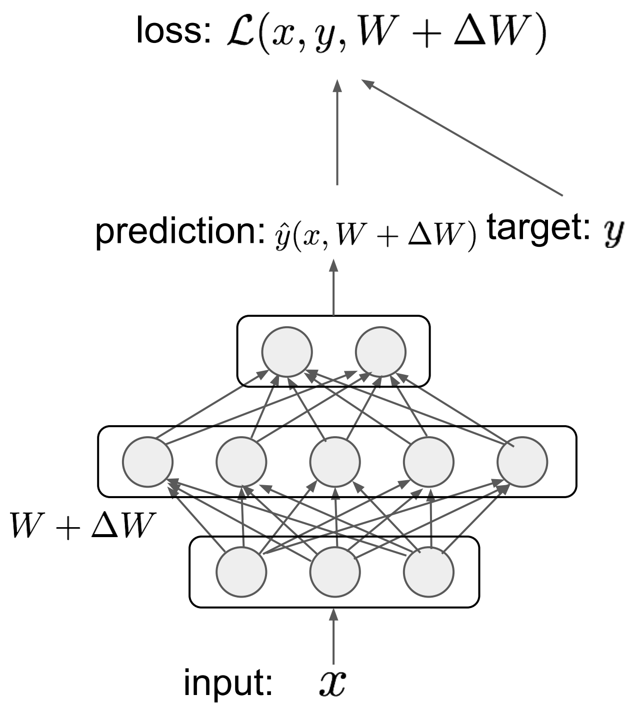
How to train neural networks with random search

Consider perturbing the weights of a neural network by a random vector ΔW.

Evaluate the perturbed averaged loss over the training examples.

Keep the perturbation ΔW if loss improves, discard otherwise.

Repeat



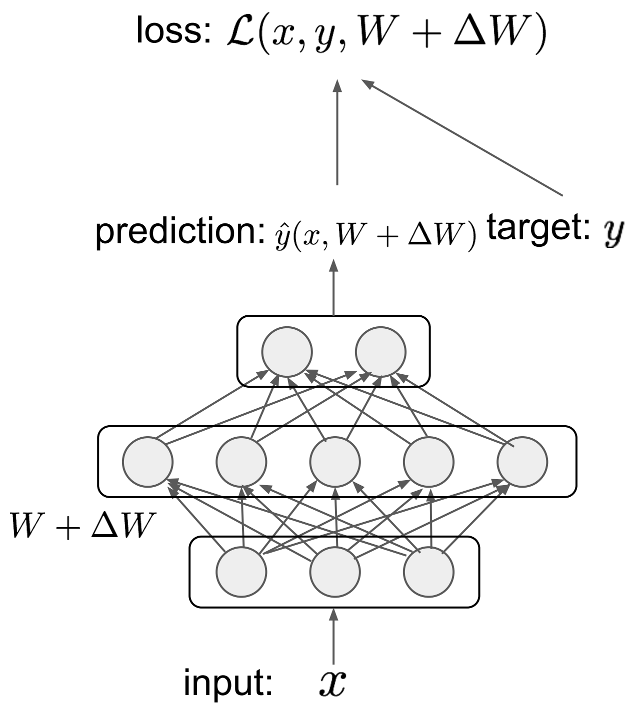
How to train neural networks with random search

Consider perturbing the weights of a neural network by a random vector $\Delta W \sim \mathcal{N}(0, \mu^2 I)$.

Evaluate the perturbed averaged loss over the training examples.

Add the perturbation weighted by the perturbed loss to the current weights $W \leftarrow W - \bar{\mathcal{L}}(W + \Delta W)\Delta W$

Repeat



How to train neural networks with random search

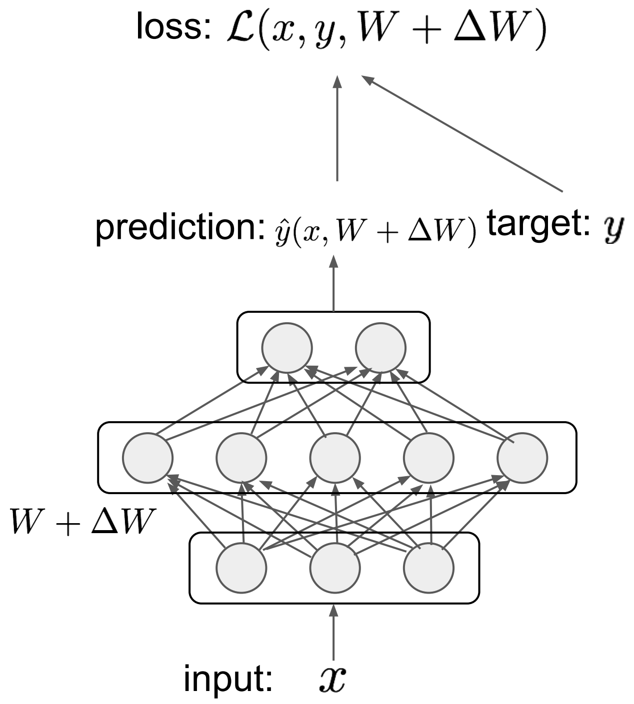
Consider perturbing the weights of a neural network by M random vector $\Delta W^{(m)} \sim \mathcal{N}(0, \mu^2 I)$

Evaluate the perturbed averaged loss over the training examples.

Add the perturbation weighted by the perturbed loss to the current weights

$$W \leftarrow W - \eta \frac{1}{M} \sum_{m=1}^{M} \frac{\bar{\mathcal{L}}(W + \Delta W^{(m)}) - \bar{\mathcal{L}}(W - \Delta W^{(m)})}{2} \Delta W^{(m)}$$

Repeat



How to train neural networks with random search

Consider perturbing the weights of a neural network by M random vector $\Delta W^{(m)} \sim \mathcal{N}(0, \mu^2 I)$

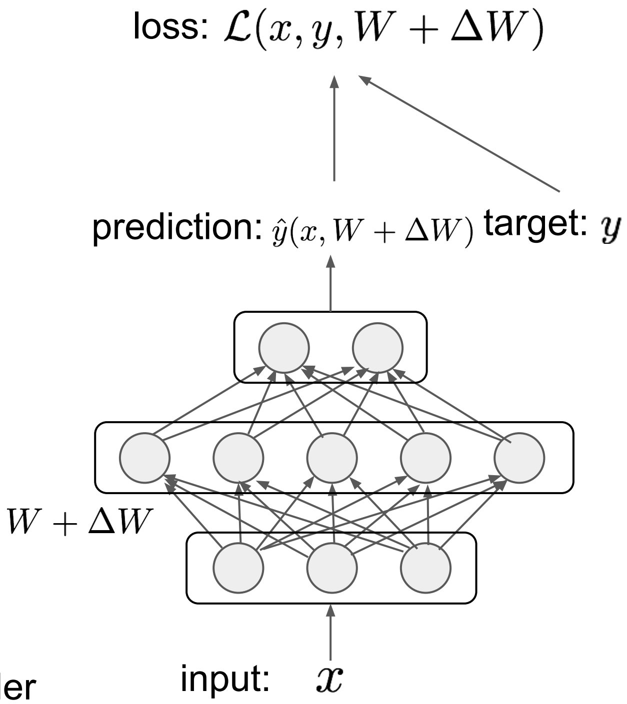
Evaluate the perturbed averaged loss over the training examples.

Add the perturbation weighted by the perturbed loss to the current weights

$$W \leftarrow W - \eta \frac{1}{M} \sum_{m=1}^{M} \frac{\bar{\mathcal{L}}(W + \Delta W^{(m)}) - \bar{\mathcal{L}}(W - \Delta W^{(m)})}{2} \Delta W^{(m)}$$

Repeat

This family of algorithms has many names: Evolution Strategy/Zero order method/Derivative-free method

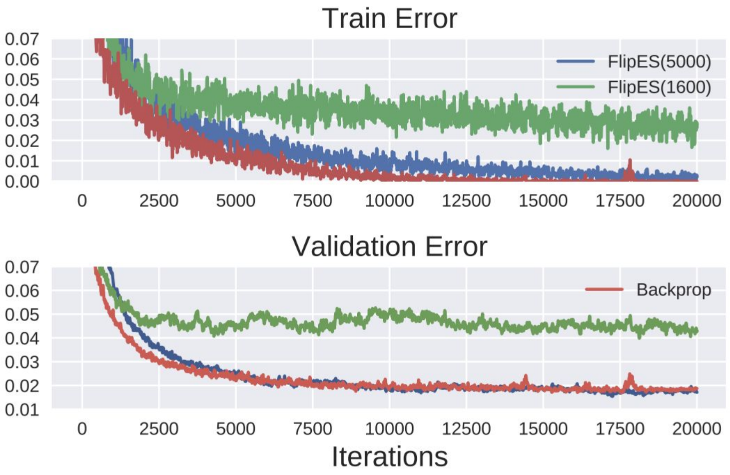


Source: Saliman et. al, 2017, Flaxman et al, 2005, Agarwal, et al. 2010, Nesterov, 2011, Berahas, et. al, 2018

How well does random search work?

Training a 784-512-512-10 neural network on MNIST (0.5 million weights):

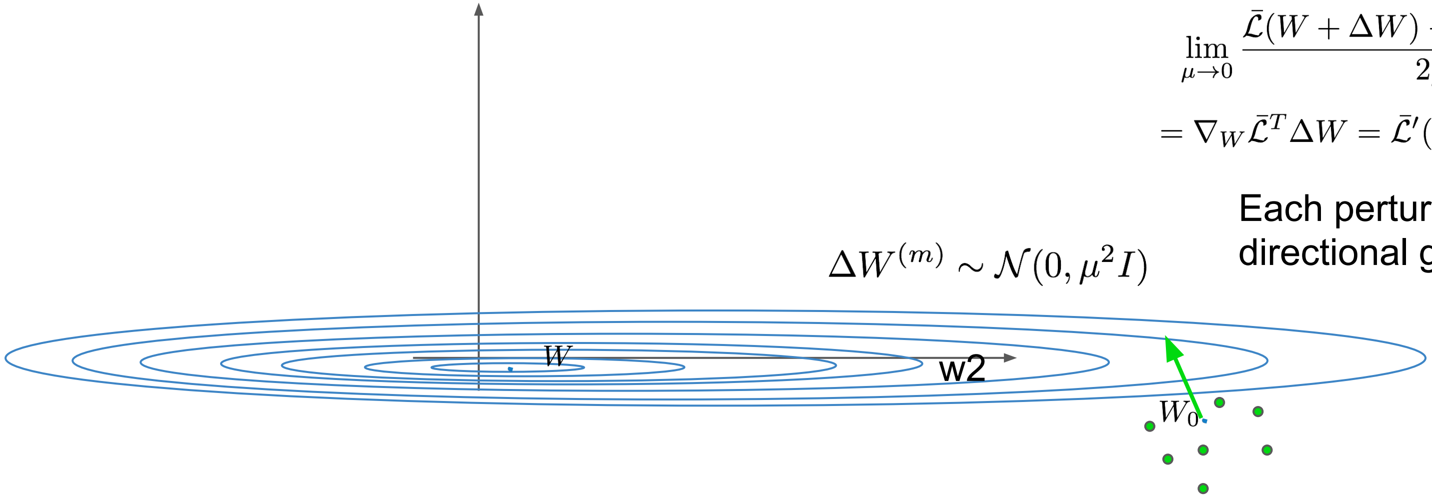
- 1600 samples vs 5000 samples vs backprop



Source: Wen, et. al, 2018

How well does random search work?

Why does random research work?



$$\Delta W^{(m)} \sim \mathcal{N}(0, \mu^2 I)$$

Random search approximate finite difference with stochastic samples:

$$\lim_{\mu \to 0} \frac{\bar{\mathcal{L}}(W + \Delta W) - \bar{\mathcal{L}}(W - \Delta W)}{2\mu^2} \Delta W$$

$$= \nabla_W \bar{\mathcal{L}}^T \Delta W = \bar{\mathcal{L}}'(W, \Delta W / \mu)$$

Each perturbation gives a directional gradient

This seems really inefficient w.r.t. the number of weights. (come back to this later)

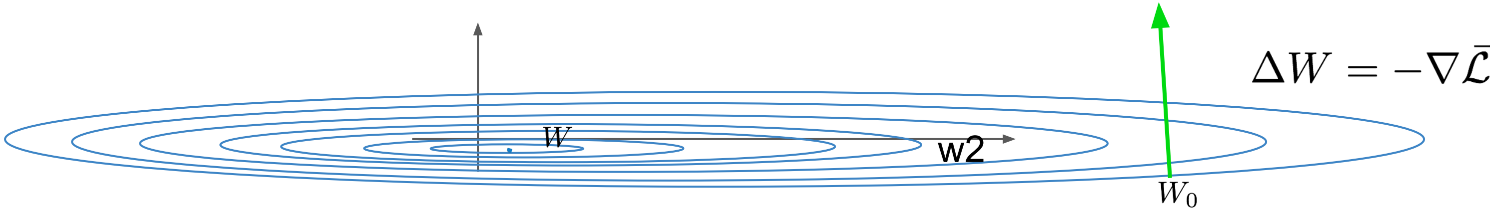
Aggregated search direction

Gradient descent and back-propagation

Using random search, computing d directional gradients requires d forward propagation.

Although it can be done in parallel, it would be more efficient if we could directly query gradient information.

We can obtain the full gradient information on continuous loss function more efficiently by back-prop.



Gradient descent

To justify our search direction choice, we formulate the following optimization problem:

Consider a small change to the weights that will minimize the averaged loss w.r.t. the weights.

The small change is formulated as a constraint in the **Euclidean** space:

$$\min_{\Delta W} \bar{\mathcal{L}}(W + \Delta W)$$
$$s.t. \|\Delta W\|_2^2 = \epsilon$$

Gradient descent

To justify our search direction choice, we formulate the following optimization problem:

Consider a small change to the weights that will minimize the averaged loss w.r.t. the weights.

The small change is formulated as a constraint in the **Euclidean** space:

Linearize the loss function and solve the Lagrangian to get the update rule:

$$\min_{\Delta W} \bar{\mathcal{L}}(W) + \nabla \bar{\mathcal{L}}(W)^T \Delta W$$

$$s.t. \|\Delta W\|_2^2 = \epsilon$$

$$\longrightarrow \quad \Delta W \propto -\nabla \bar{\mathcal{L}}(W)$$

How well does gradient descent work?

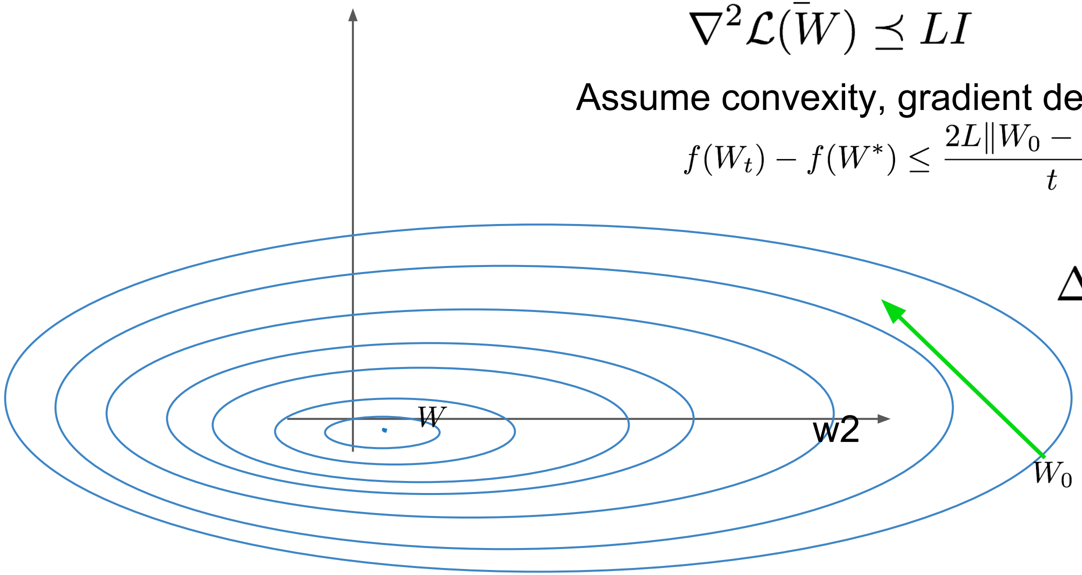
Let L be the Lipschitz constant of the gradient:

$$\nabla^2 \mathcal{L}(\bar{W}) \preceq LI$$

Assume convexity, gradient descent converges in 1/T:

$$f(W_t) - f(W^*) \leq \frac{2L\|W_0 - W^*\|^2}{t}$$

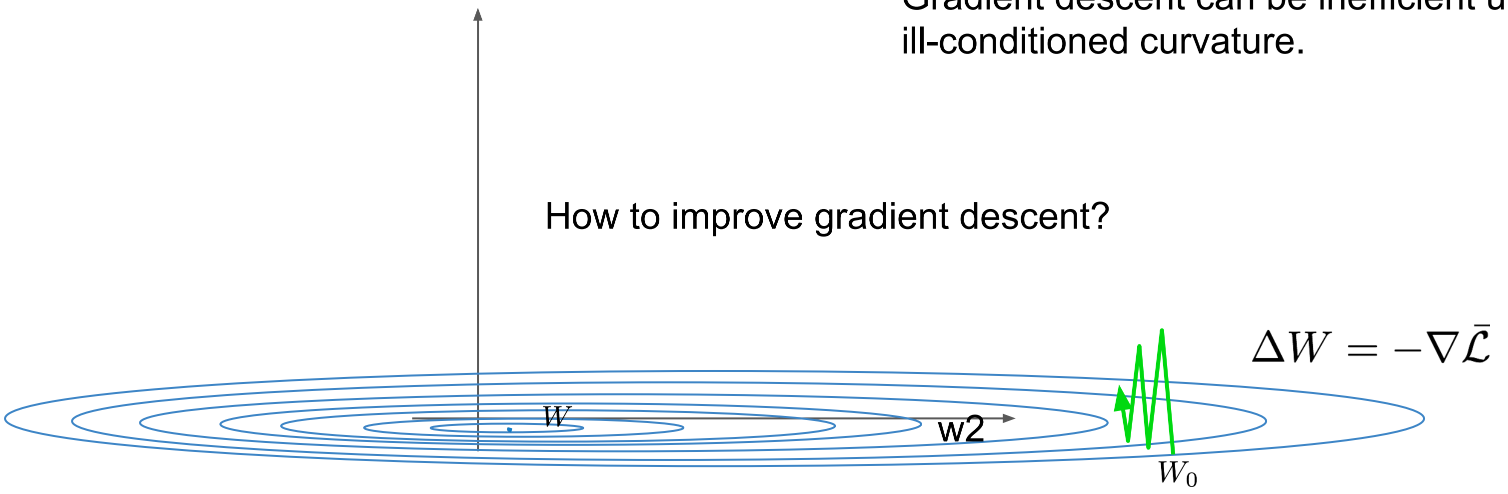
$$\Delta W_t = -\eta_t \nabla \bar{\mathcal{L}}(W_t)$$



How well does gradient descent work?

Gradient descent can be inefficient under ill-conditioned curvature.

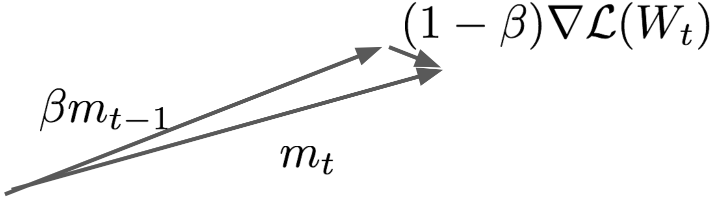
How to improve gradient descent?



Momentum: smooth gradient with moving average

Keep a running average of the gradient updates can smooth out the zig-zag behavior:

$$m_t = \beta m_{t-1} + (1-\beta)\nabla\bar{\mathcal{L}}(W_t)$$
$$\Delta W_t = -\eta_t m_t$$

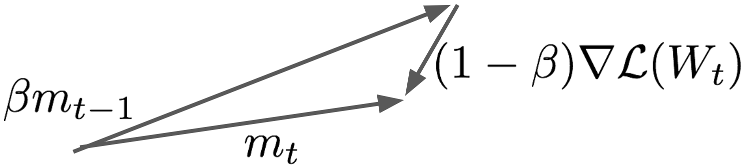


Nesterov's accelerated gradient:

$$m_t = \beta m_{t-1} + (1-\beta)\nabla\bar{\mathcal{L}}(W_t - \eta_t \beta m_{t-1})$$
$$\Delta W_t = -\eta_t m_t$$

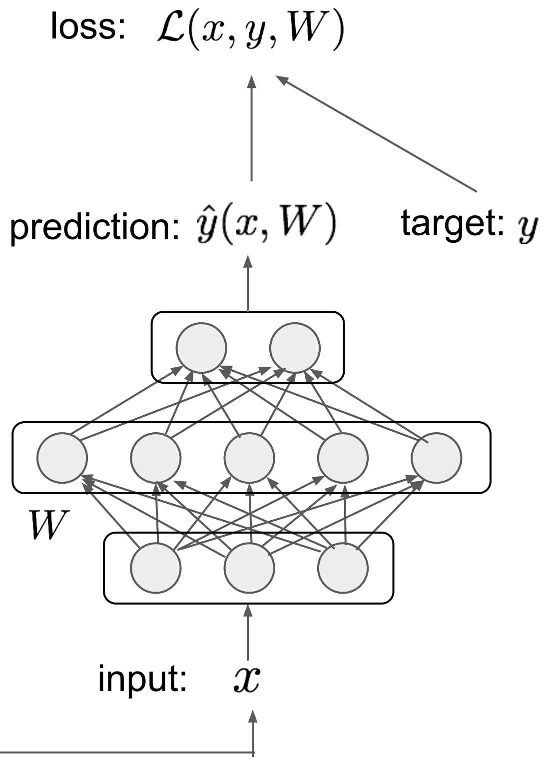
Lookahead trick

Assume convexity, NAG obtains 1/T^2 rate



Stochastic gradient descent: improve efficiency

Averaged loss: $\bar{\mathcal{L}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x^{(i)}, y^{(i)}, W) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(W)$



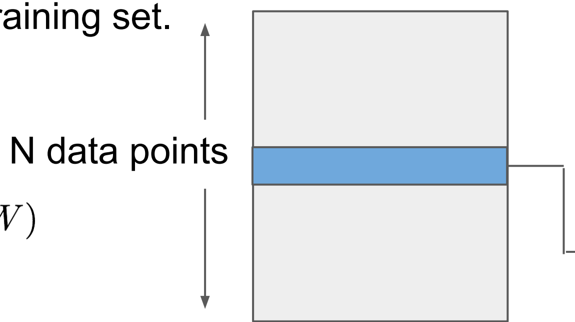
Computing averaged loss requires going through the entire training dataset.

Maybe we do not have to go over millions of images for a small weight update.

Sample B points from the whole training set.

mini-batch loss:

$$\tilde{\mathcal{L}} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}(x^{(i)}, y^{(i)}, W) = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_i(W)$$



Stochastic gradient descent: improve efficiency

Sample B points from the whole training set.

mini-batch loss:

$$\tilde{\mathcal{L}} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}(x^{(i)}, y^{(i)}, W) = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_i(W)$$

Without losing generality, assume strong convexity, B = 1 and decay learning rate with $1/\sqrt{t}$

SGD obtains convergence rate of $O(1/\sqrt{T})$

Stochastic gradient descent: improve efficiency

Sample B points from the whole training set.

mini-batch loss:

$$\tilde{\mathcal{L}} = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}(x^{(i)}, y^{(i)}, W) = \frac{1}{B}\sum_{i=1}^{B}\mathcal{L}_i(W)$$

Without losing generality, assume strong convexity, B = 1 and decay learning rate with $1/\sqrt{t}$

SGD obtains convergence rate of $O(1/\sqrt{T})$

Are we making improvement over random search at all?

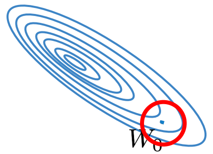
Yes! Random search with two-point methods has a rate of $O(\sqrt{d/M}/\sqrt{T})$ (Duchi, et. al. 2015)

Revisit gradient descent

Constrained optimization problems for different gradient-based algorithms:

Objective: $$\min_{\Delta W} \quad \mathcal{L}(W + \Delta W)$$

Gradient descent: $$s.t. \quad \|\Delta W\|_2^2 = \epsilon$$



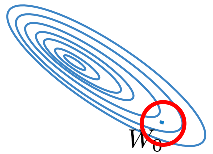
Are there other constraints we can use here?

Revisit gradient descent

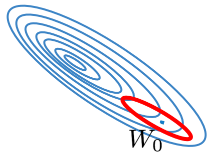
Constrained optimization problems for different gradient-based algorithms:

Objective: $\min_{\Delta W} \quad \mathcal{L}(W + \Delta W)$

Gradient descent: $s.t. \quad \|\Delta W\|_2^2 = \epsilon$



Natural gradient (Amari, 1998): $s.t. \quad D(P_W || P_{W+\Delta W}) = \epsilon$



Natural gradient descent

Again, let us consider a small change to the weights that will minimize the averaged loss w.r.t. the weights.

A small change is formulated as a constraint in the **Probability** space using KL divergence:

Linearize the model and take the second-order taylor expansion around the current weights:

$$\min_{\Delta W} \bar{\mathcal{L}}(W) + \nabla \bar{\mathcal{L}}(W)^T \Delta W$$

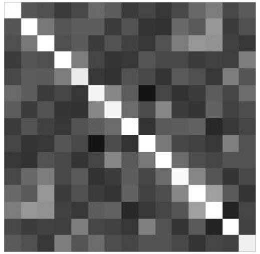
$$s.t. \quad \frac{1}{2} \Delta W^T F \Delta W = \epsilon$$

$\longrightarrow$

$$\Delta W \propto F^{-1} \nabla \bar{\mathcal{L}}(W) = F^{-1} \mathcal{G}_W$$

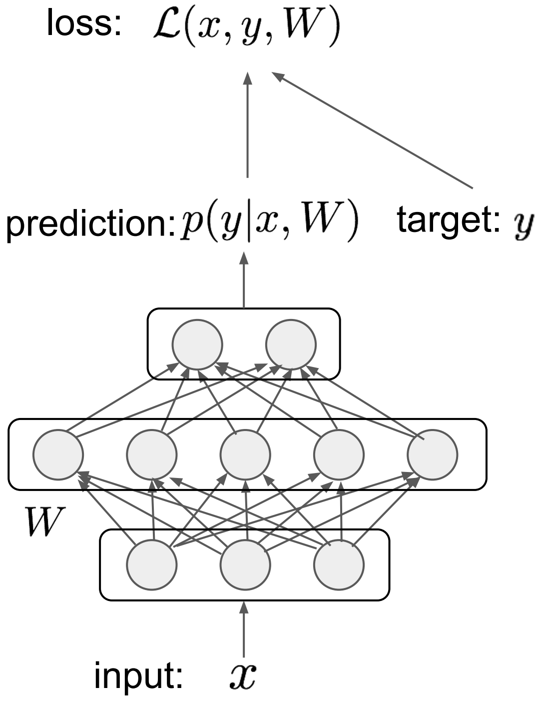
$$F \triangleq \mathbb{E}_{\mathbf{x},\mathbf{y} \sim \mathbf{p}(\mathbf{x},\mathbf{y})} [\nabla_W \log p(\mathbf{y}|\mathbf{x}) \nabla_W \log p(\mathbf{y}|\mathbf{x})^T]$$

Fisher information matrix



F
Fisher information matrix

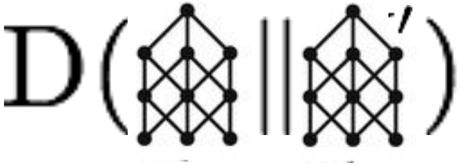
$$\triangleq \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathbf{p}(\mathbf{x},\mathbf{y})}[\nabla_W \log p(\mathbf{y}|\mathbf{x}) \nabla_W \log p(\mathbf{y}|\mathbf{x})^T]$$



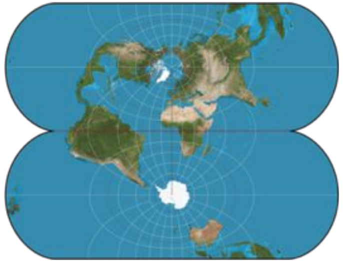
Fisher information matrix

Geodesic perspective: distance measure in weight space v.s. model output space

$$\mathrm{D}(W \,\|\, W')$$

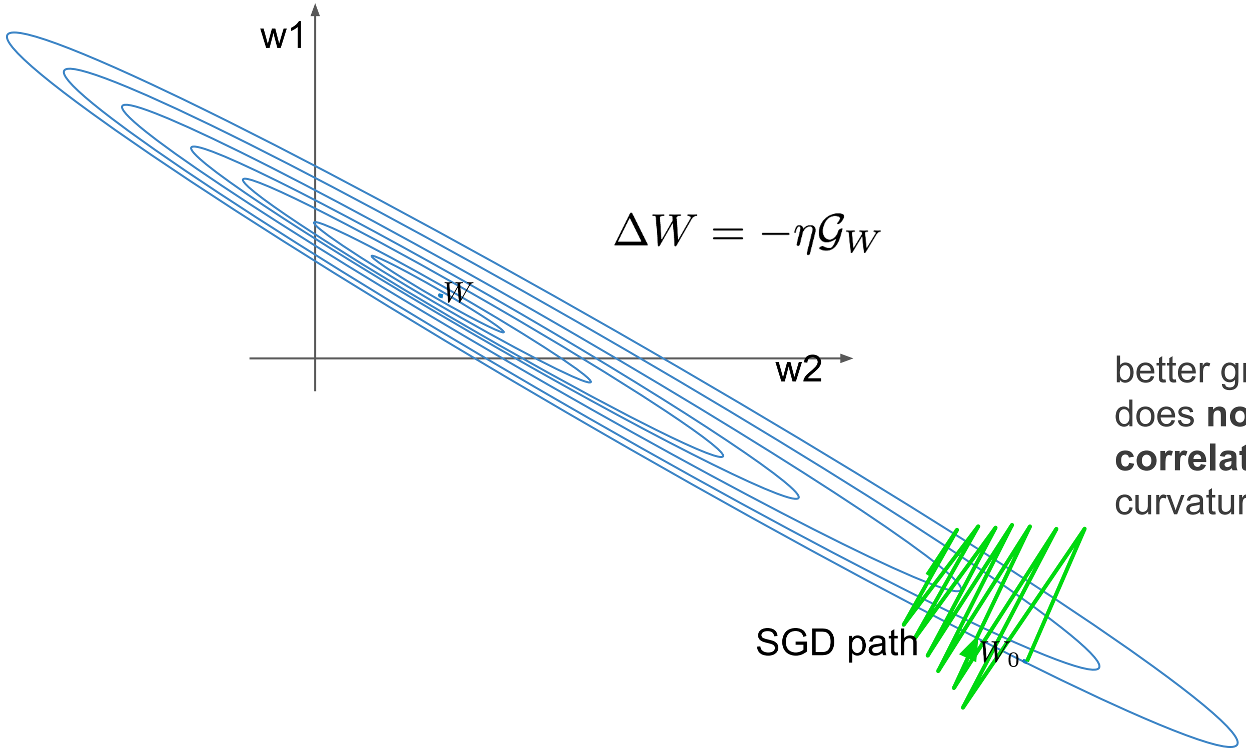


$$\xrightarrow{\mathbf{F}^{-1}}$$



$$\mathbf{F} \triangleq \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathbf{p}(\mathbf{x},\mathbf{y})}[\nabla_W \log p(\mathbf{y}|\mathbf{x}) \nabla_W \log p(\mathbf{y}|\mathbf{x})^T]$$

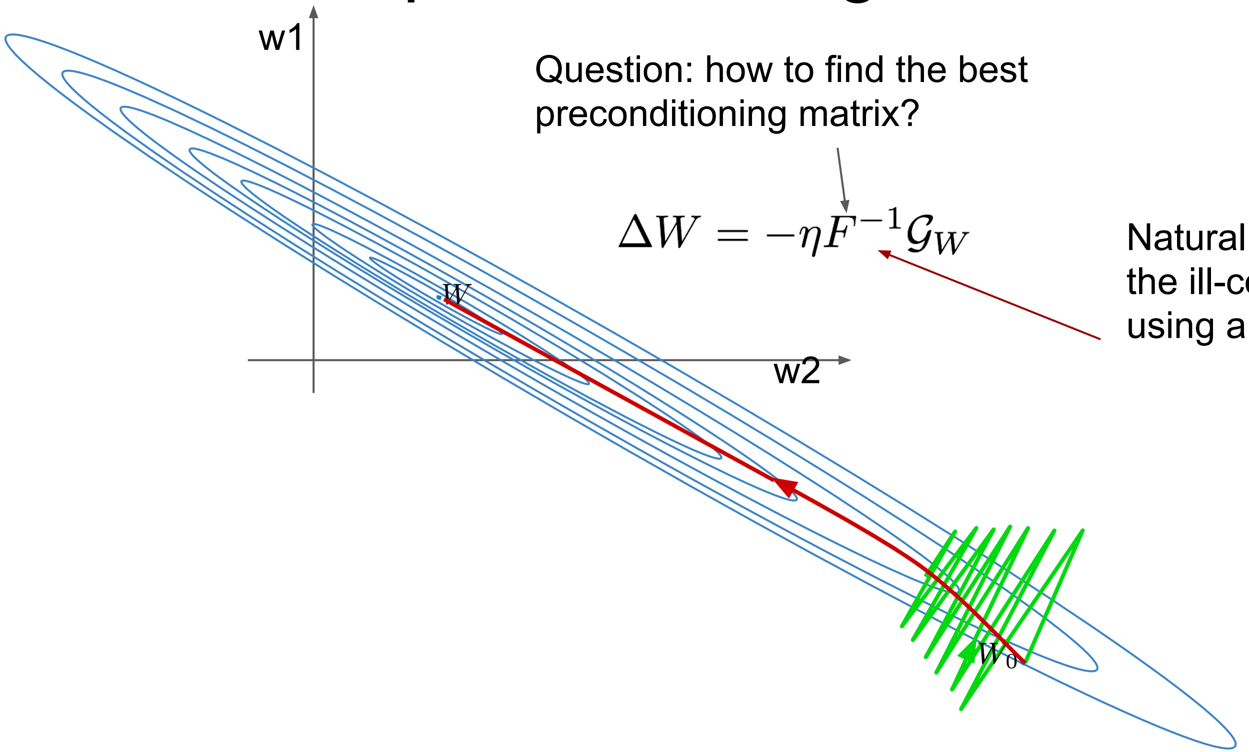
When first-order methods fails



$$\Delta W = -\eta \mathcal{G}_W$$

better gradient estimation does **not** overcome **correlated ill-conditioned** curvature in the weight space

Second-order optimization algorithms



Question: how to find the best preconditioning matrix?

$$\Delta W = -\eta F^{-1}\mathcal{G}_W$$

Natural gradient corrects for the ill-conditioned curvature using a preconditioning matrix.

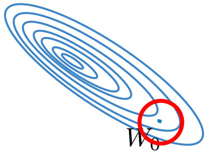
Amari, 1998

Second-order method algorithms

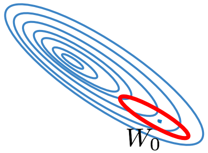
Constrained optimization problems for different gradient-based algorithms:

Objective: $$\min_{\Delta W} \quad \mathcal{L}(W + \Delta W)$$

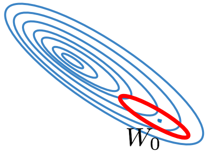
Gradient descent: $$s.t. \quad \|\Delta W\|_2^2 = \epsilon$$



Natural gradient: $$s.t. \quad D(P_W || P_{W+\Delta W}) = \epsilon$$



Family of second-order approximations: $$s.t. \quad \frac{1}{2}\|A^{-1}\Delta W\|_A^2 = \epsilon$$



Find a good preconditioning matrix

$$s.t. \quad \frac{1}{2}\|A^{-1}\Delta W\|_A^2 = \epsilon$$

Intuitively, we would like to find a preconditioning matrix A focusing on the weight space that has not been explored much from the previous updates:

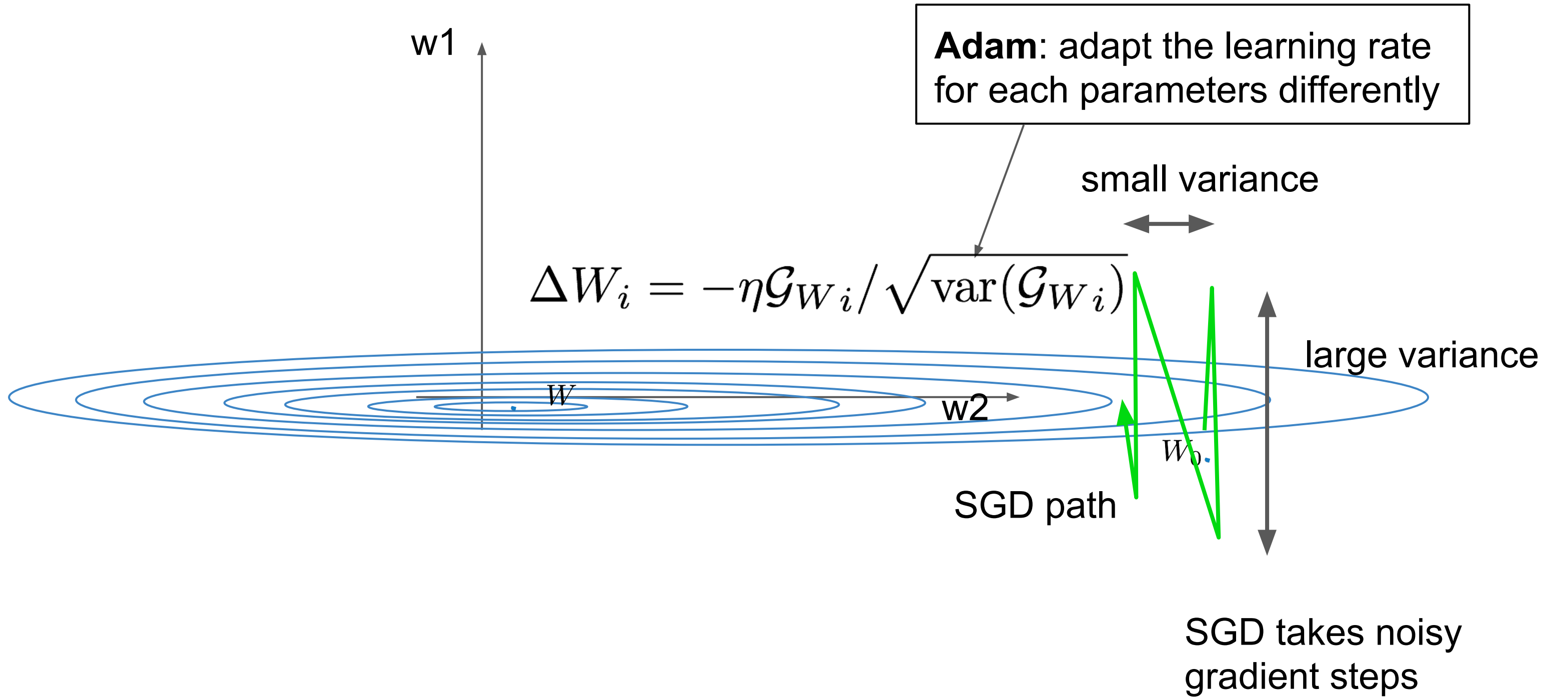
Consider the following optimization problem to find the preconditioner:

$$\min_A \frac{1}{2T}\sum_{t=1}^{T}\|A^{-1}\mathcal{G}_{W_t}\|_A^2$$
$$s.t. \quad Tr(A) \succeq \mu, A \succeq 0$$

$$\longrightarrow \quad A \propto \left(\frac{1}{T}\sum_{t=1}^{T}\mathcal{G}_{W_t}\mathcal{G}_{W_t}^T\right)^{1/2}$$

In practice, most of the algorithms like AdaGrad, Adam, RMSProp, Tonga uses diagonal approximation to make this preconditioning tractable.

When first-order methods work well

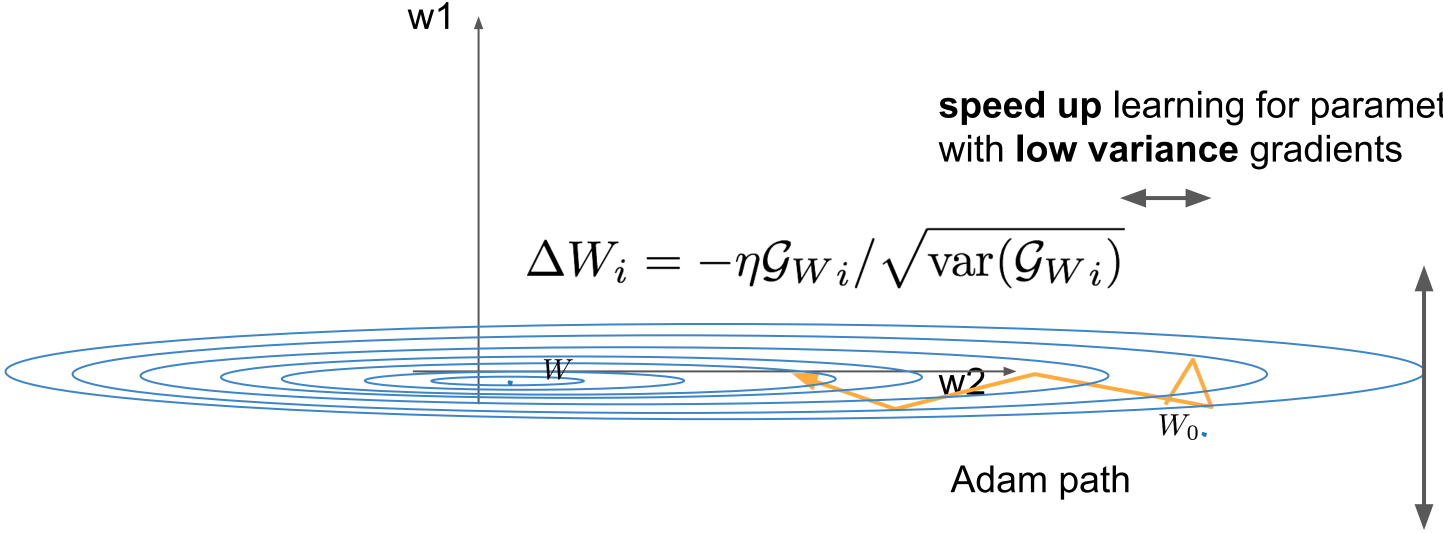


Kingma and Ba, ICLR 2015

When first-order methods work well

speed up learning for parameters with **low variance** gradients

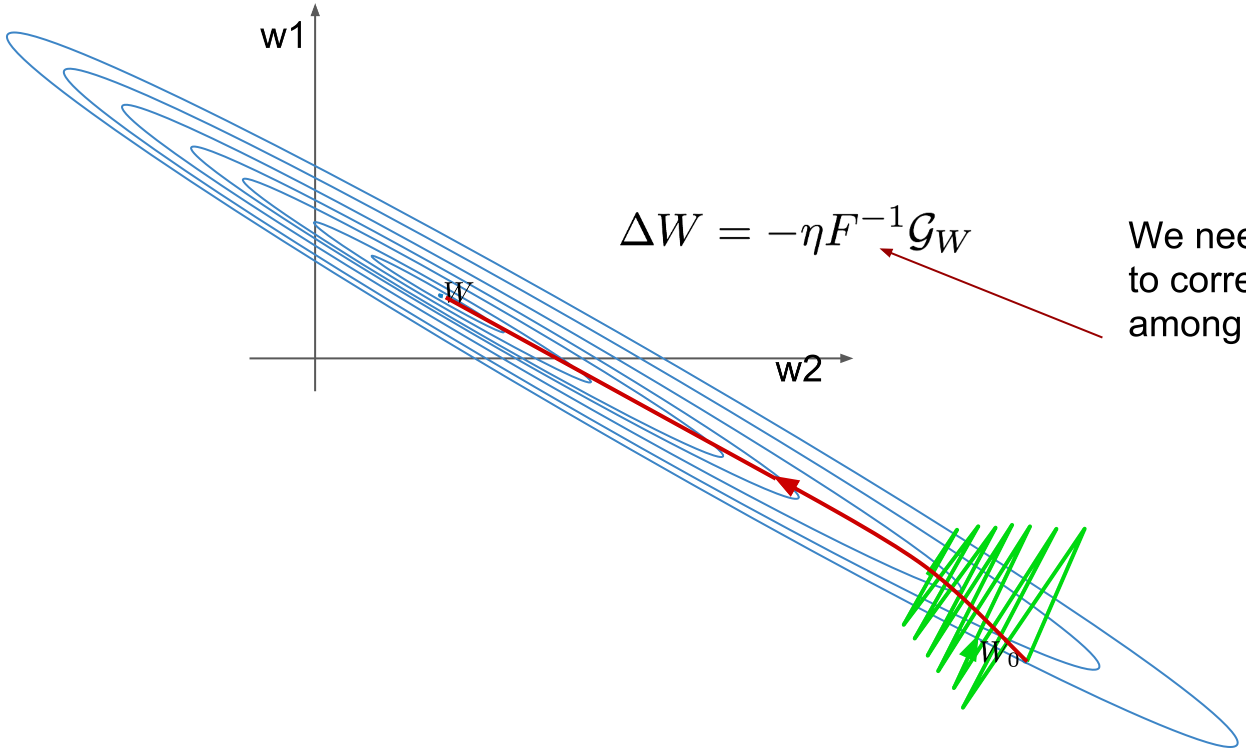
$$\Delta W_i = -\eta \mathcal{G}_{W\,i} / \sqrt{\mathrm{var}(\mathcal{G}_{W\,i})}$$



slow down learning along the **high variance** direction

Kingma and Ba, ICLR 2015

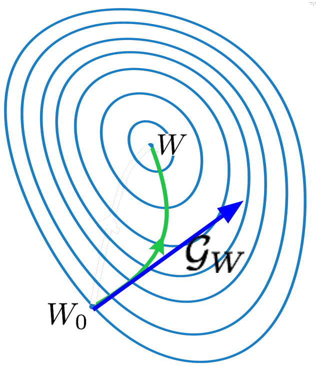
When first-order methods fail



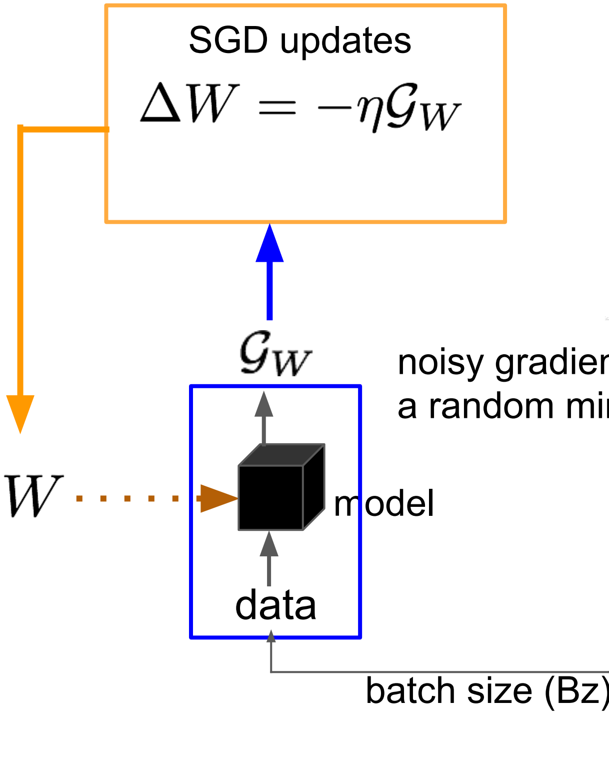
$$\Delta W = -\eta F^{-1}\mathcal{G}_W$$

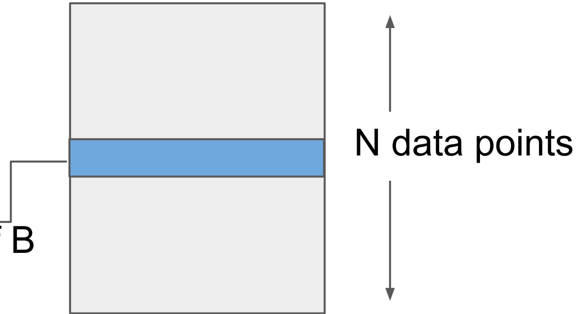
We need full curvature matrix to correct for the correlation among the weights.

Learning on a single machine



Commonly used SGD variants:
AdaGrad, *Duchi et al., COLT 2010*
Adam, *Kingma and Ba, ICLR 2015*
Nesterovs' momentum *Sutskever et al., ICML 2013*



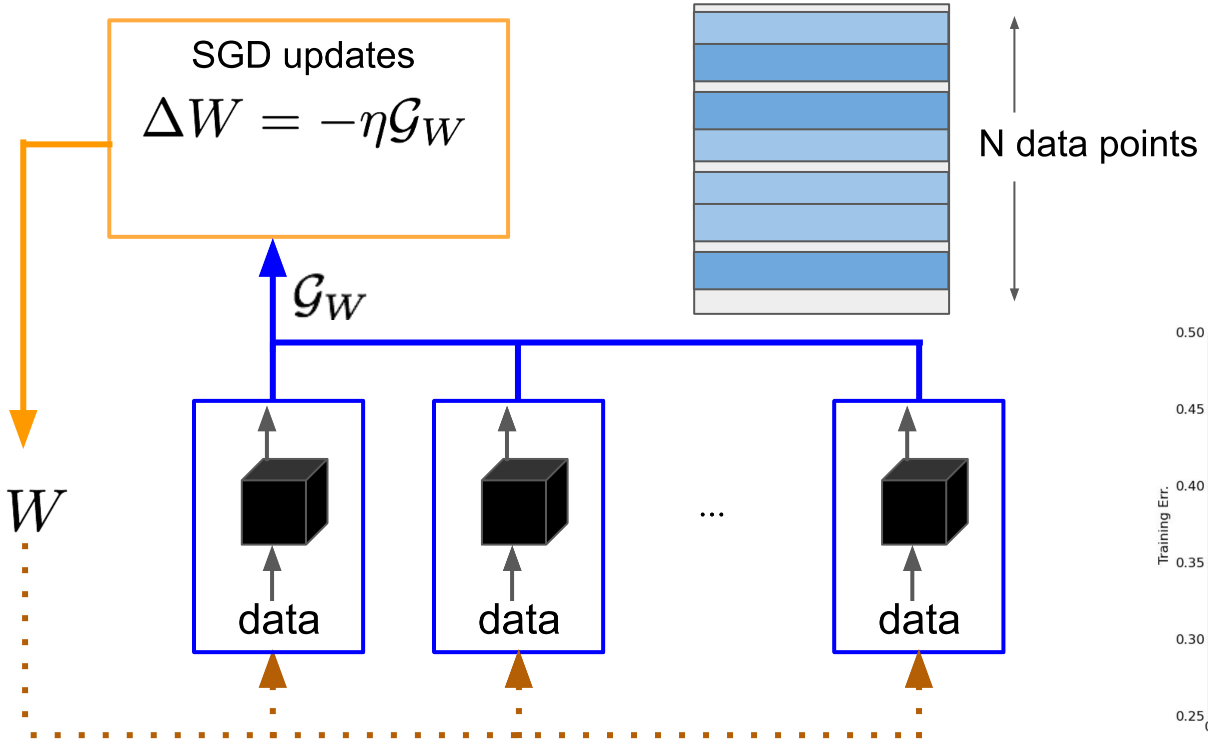


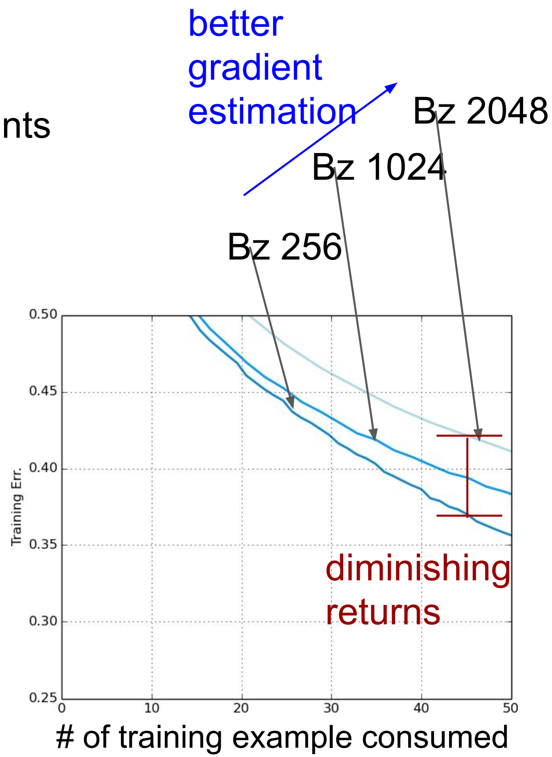
Distributed learning

Where does the speedup comes from?

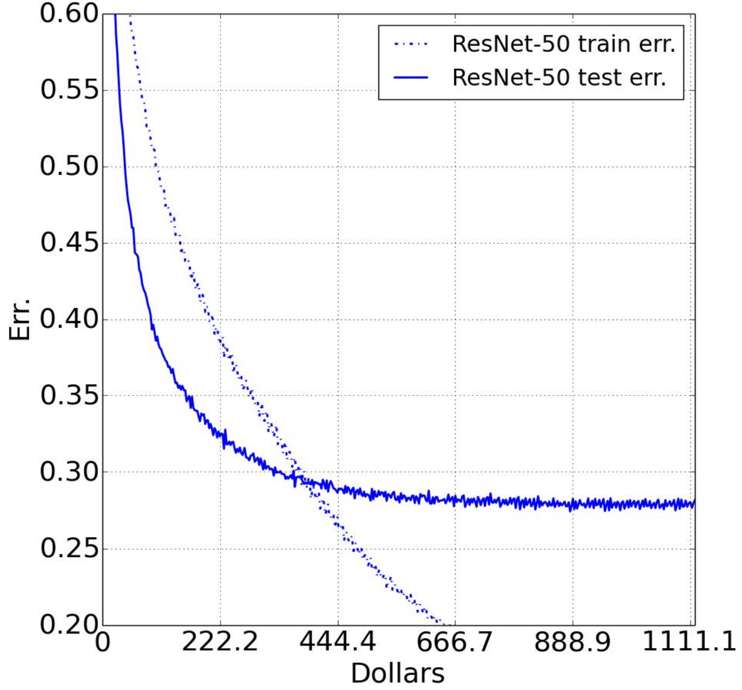
1. More accurate gradient estimation

2. More frequent updates



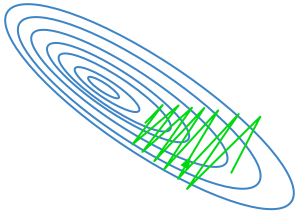
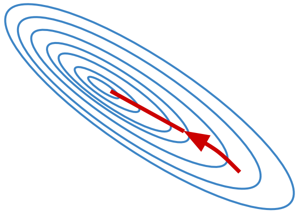
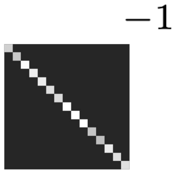
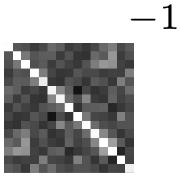


Here is the training plot of a state-of-the-art ResNet trained on 8 GPUs:



Amazon EC2 instance cost

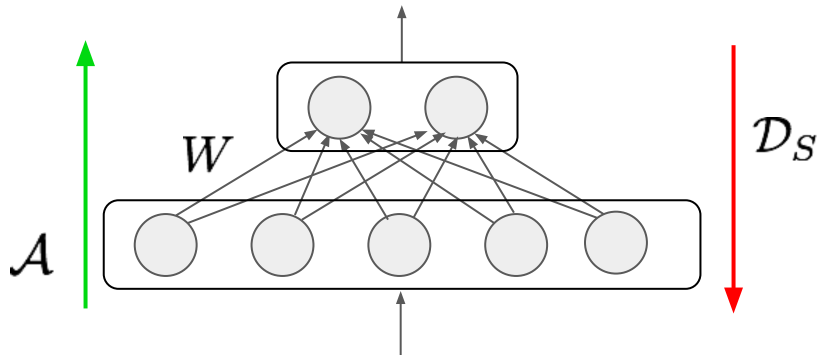
Scalability of the “**black-box**” optimization algorithms

	SGD	natural gradient
Scalability with additional computational resources	 **poor**	 **great**
Scalability with the number of weights	 **great**	 **poor**

Two problems: 1. Memory inefficient

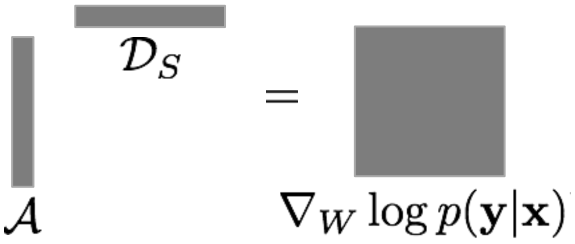
2. Inverse intractable

Background: Natural gradient for neural networks

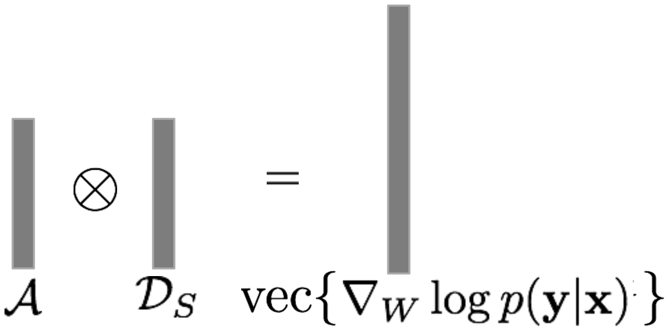


- Insight: The flattened gradient matrix in a layer has a Kronecker product structure.

Gradient computation:

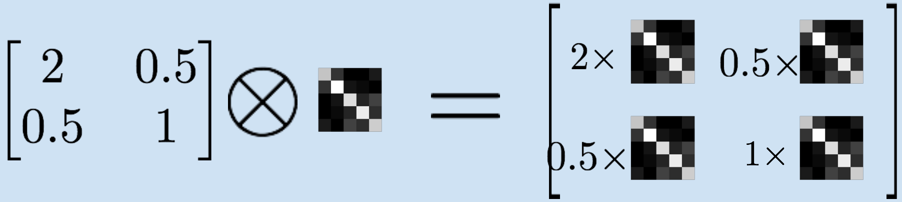


This is equivalent to:



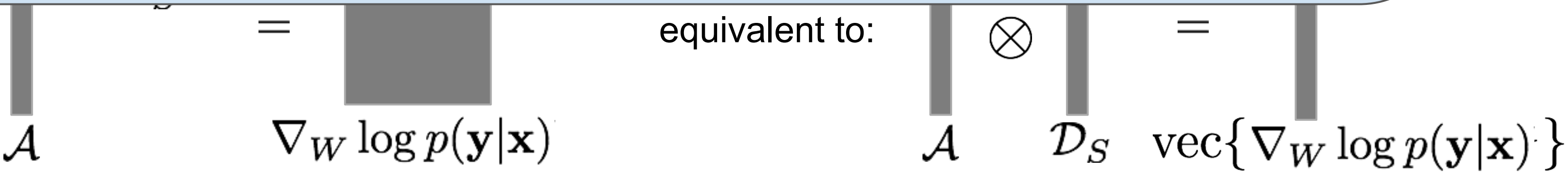
Background: Natural gradient for neural networks

Kronecker product definition:

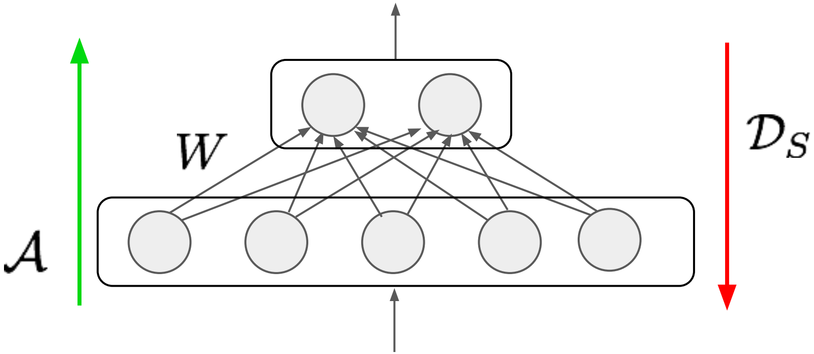


- Tile the larger matrix with the rescaled smaller matrices.

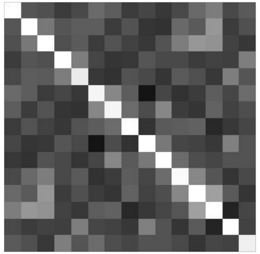
Gradient computation:



Background: Natural gradient for neural networks



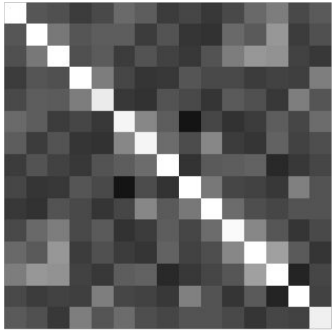
- Insight: The flattened gradient matrix in a layer has a Kronecker product structure.



$$\triangleq \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathbf{p}(\mathbf{x},\mathbf{y})}[\, \mathcal{A}\mathcal{A}^T \otimes \mathcal{D}_S{\mathcal{D}_S}^T \,]$$

F
Fisher information matrix

Background: Kronecker-factored natural gradient



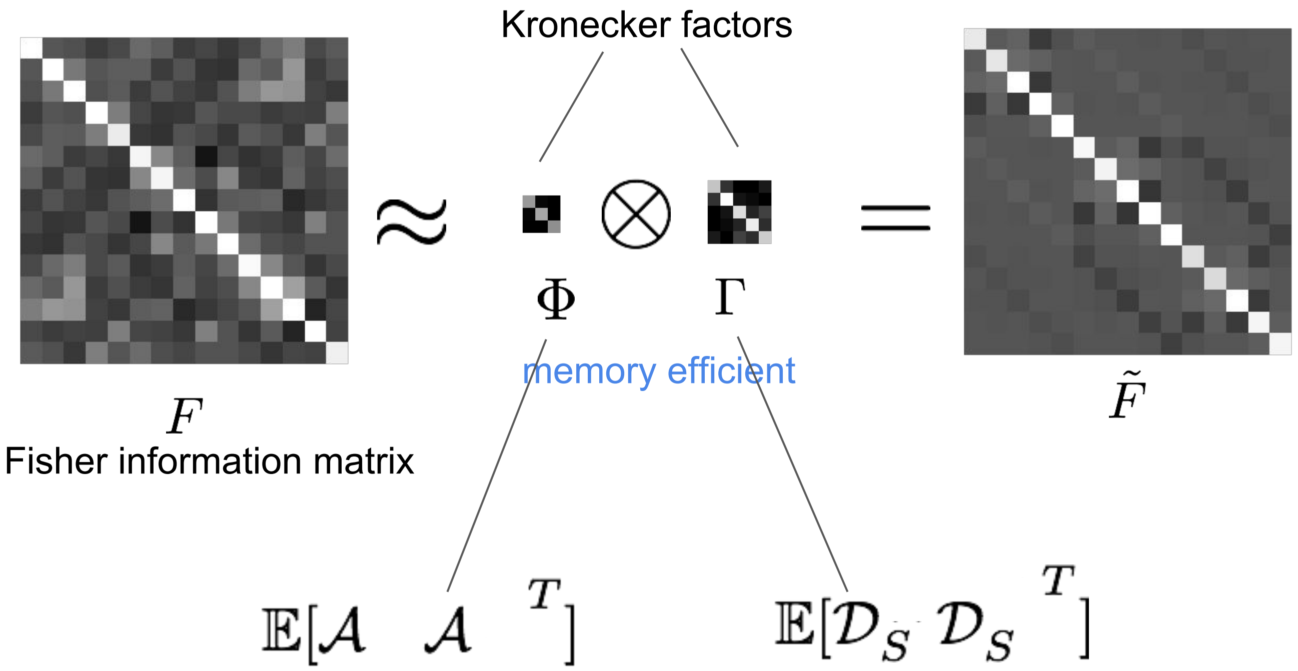
$$\approx$$

F

Fisher information matrix

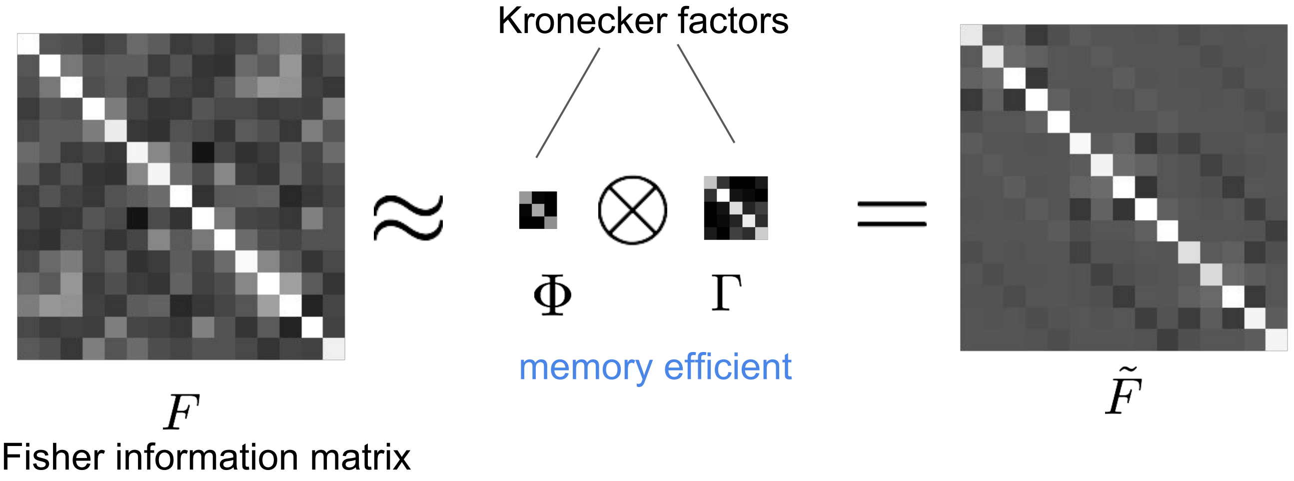
Martens and Grosse, ICML 2015

Background: Kronecker-factored natural gradient



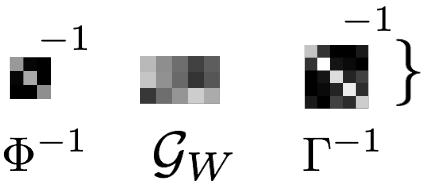
Martens and Grosse, ICML 2015

Background: Kronecker-factored natural gradient



Problem: It is still computationally expensive comparing to SGD

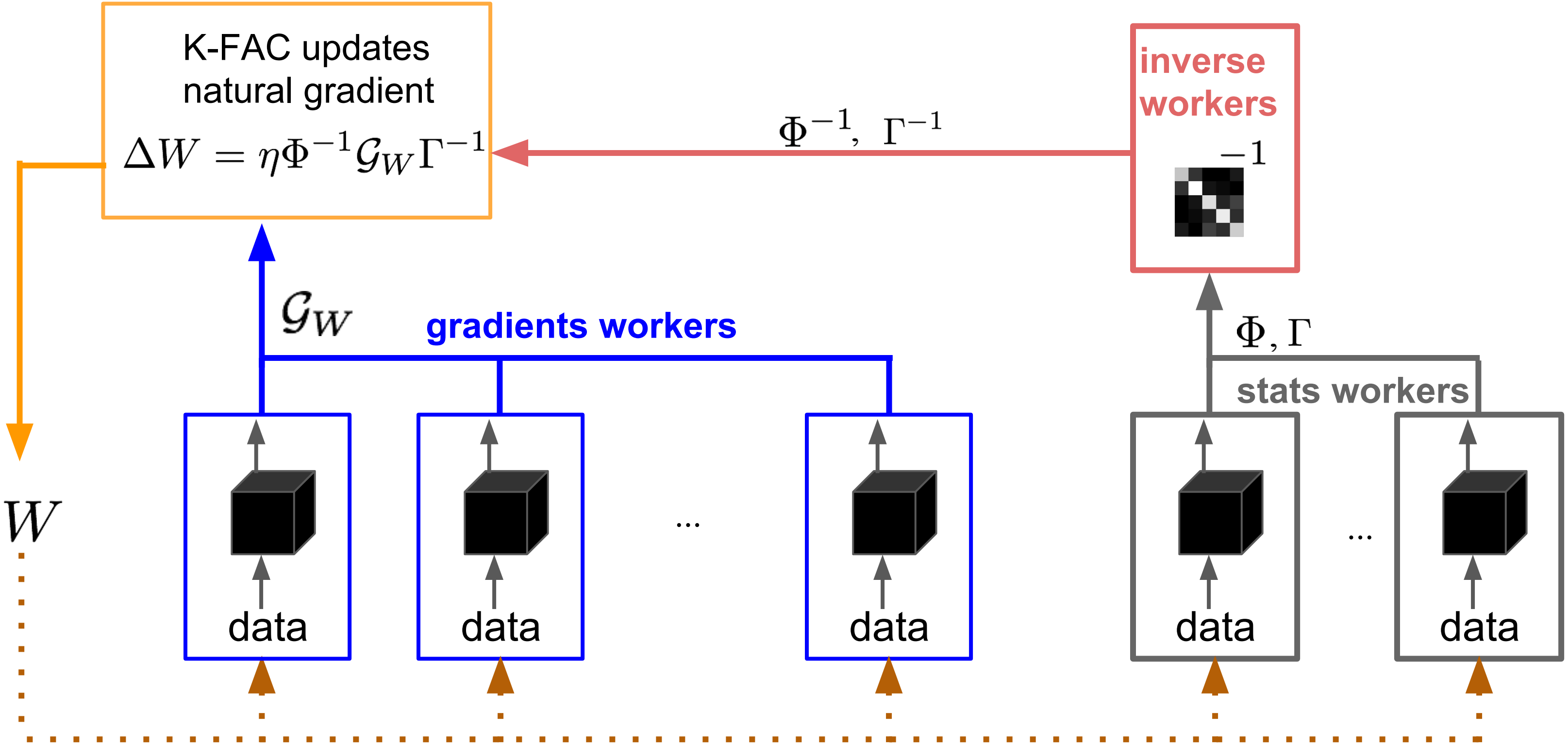
$$\tilde{F}^{-1}\mathrm{vec}\{\mathcal{G}_W\} = \mathrm{vec}\{\Phi^{-1}\mathcal{G}_W\Gamma^{-1}\}$$



tractable inverse

Martens and Grosse, ICML 2015

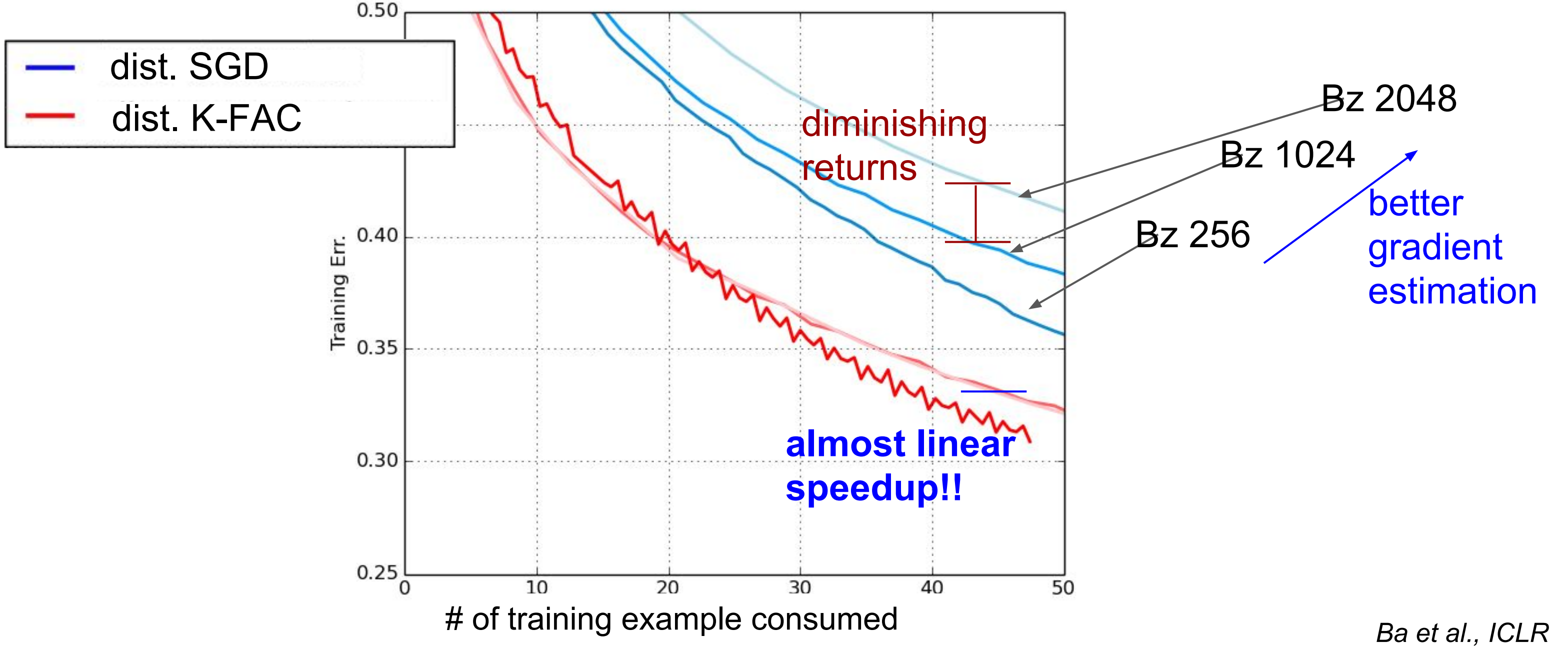
Distributed K-FAC natural gradient



Ba et al., ICLR 2017

Scalability experiments

Scientific question: How well does distributed K-FAC ***scale*** with more parallel computational resources comparing to distributed SGD?



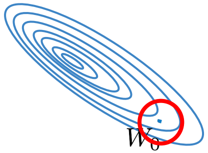
Ba et al., ICLR 2017

Second-order method algorithms

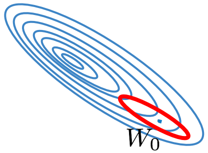
Constrained optimization problems for different gradient-based algorithms:

Objective: $\min_{\Delta W} \quad \mathcal{L}(W + \Delta W)$

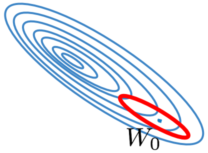
Gradient descent: $s.t. \quad \|\Delta W\|_2^2 = \epsilon$



Natural gradient: $s.t. \quad D(P_W || P_{W+\Delta W}) = \epsilon$



Family of second-order approximations: $s.t. \quad \frac{1}{2}\|A^{-1}\Delta W\|_A^2 = \epsilon$



Thank you