K-FAC and Natural Gradients

Matt Johnson and Daniel Duckworth

Dec 8, 2017

Google **Brain**

Google **Research**

DeepMind


James Martens


Roger Grosse


Jimmy Ba

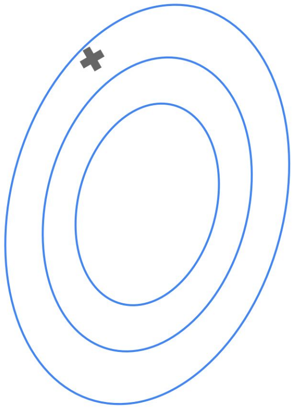

James Keeling


Noah Siegel

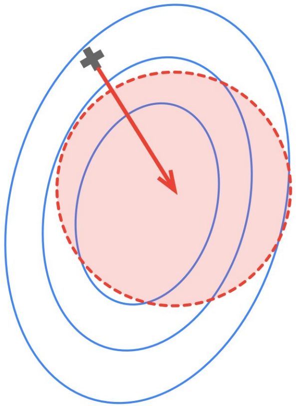

Olga Wichrowska


Alok Aggarwal

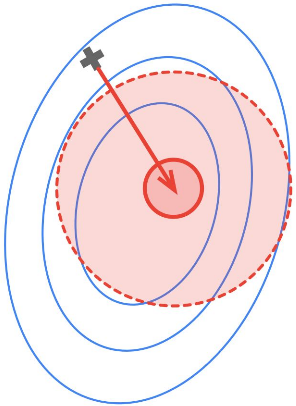
Limits of big-batch training



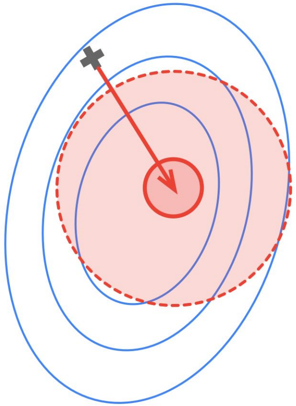
Limits of big-batch training

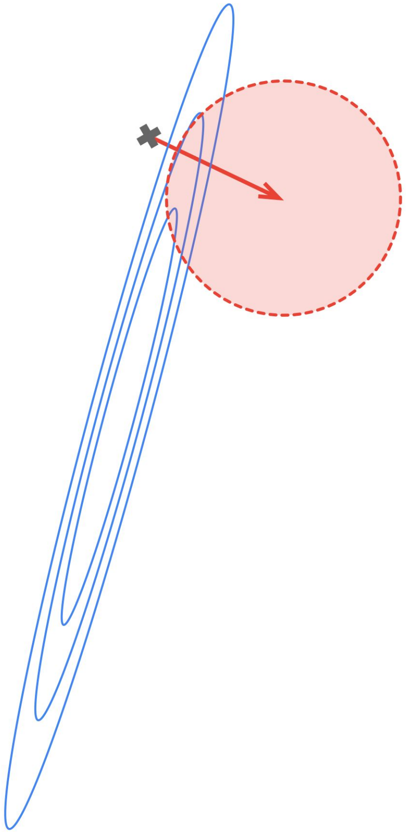


Limits of big-batch training

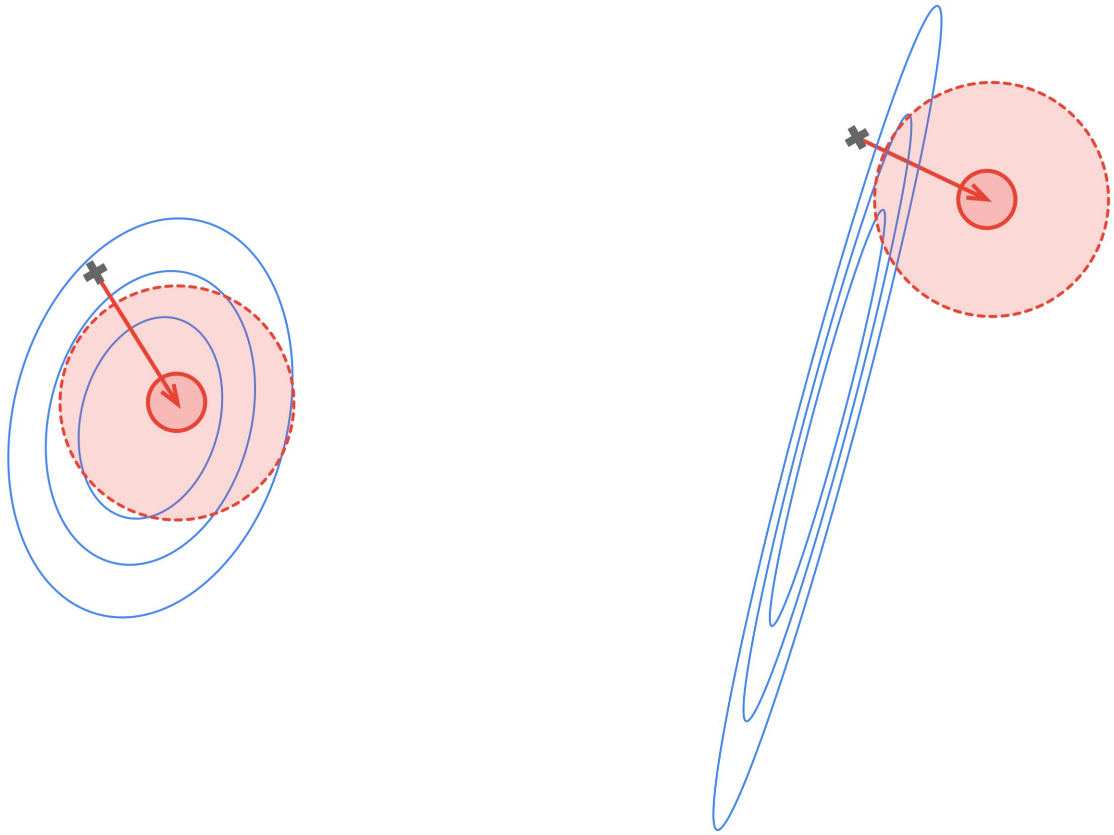


Limits of big-batch training





Limits of big-batch training



Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2c} + [1 - \bar{\alpha} c]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2c} \right)$$

Bottou, Curtis, Nocedal. Optimization methods for large-scale machine learning. 2016.

Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2cN} + [1 - \bar{\alpha} c]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2cN} \right)$$

Bottou, Curtis, Nocedal. [Optimization methods for large-scale machine learning](). 2016.

Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2cN} + [1 - \bar{\alpha} c]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2cN} \right)$$

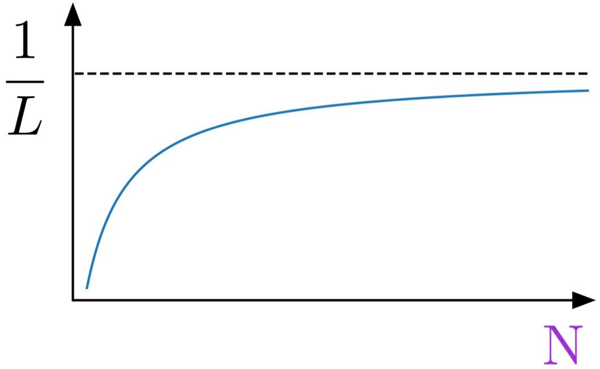
$$\bar{\alpha} \leq \frac{1}{L + L\frac{M}{N}}$$

Bottou, Curtis, Nocedal. [Optimization methods for large-scale machine learning](). 2016.

Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2cN} + [1 - \bar{\alpha} c]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2cN} \right)$$

$$\bar{\alpha} \leq \frac{1}{L + L\frac{M}{N}}$$



Bottou, Curtis, Nocedal. Optimization methods for large-scale machine learning. 2016.

Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2cN} + \left[1 - \frac{c}{L + L\frac{M}{N}}\right]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2cN}\right)$$

Bottou, Curtis, Nocedal. Optimization methods for large-scale machine learning. 2016.

Limits of big-batch training

$$\mathbb{E}[F(w_k) - F_*] \leq \frac{\bar{\alpha} L M}{2cN} + \left[1 - \frac{c}{L + L\frac{M}{N}}\right]^{k-1} \left(F(w_1) - F_* - \frac{\bar{\alpha} L M}{2cN}\right)$$

Increasingly important to control **curvature** $\frac{L}{c}$

Bottou, Curtis, Nocedal. [Optimization methods for large-scale machine learning](). 2016.

Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Desiderata for H_k:

Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Desiderata for H_k :

1. easy to estimate in stochastic / online setting

Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Desiderata for H_k:

1. easy to estimate in stochastic / online setting
2. works on nonconvex objectives (positive definite)

Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Desiderata for H_k:

1. easy to estimate in stochastic / online setting
2. works on nonconvex objectives (positive definite)
3. fast to compute update (close to SGD)

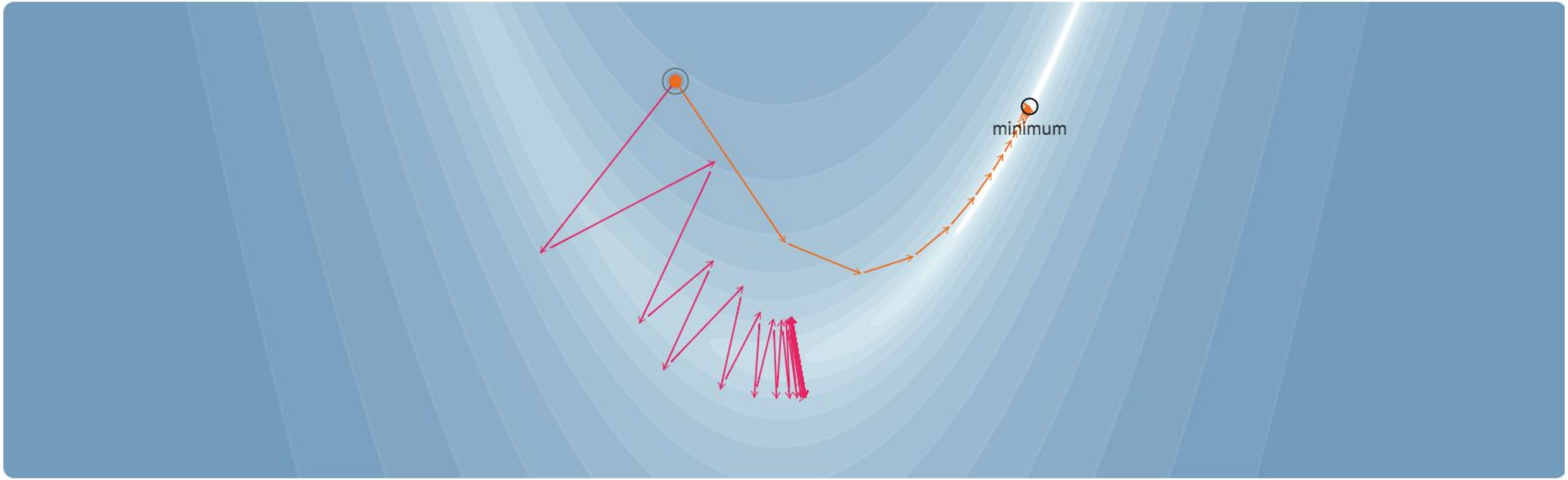
Control curvature with second-order methods

$$w_{k+1} \leftarrow w_k - \alpha_k H_k^{-1} g(w_k, \xi_k)$$

Desiderata for H_k:

1. easy to estimate in stochastic / online setting
2. works on nonconvex objectives (positive definite)
3. fast to compute update (close to SGD)
4. adapted to problem / network architecture

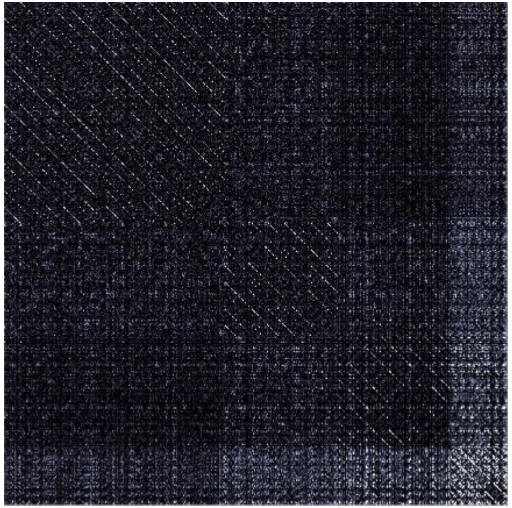
Natural gradients correct for curvature



but exact natural gradients are **expensive**...

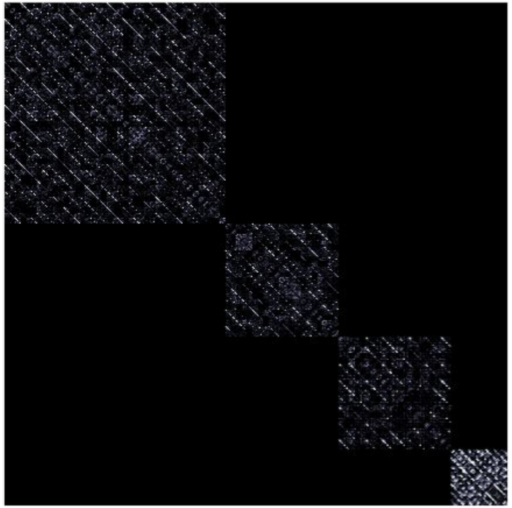
figures from Katherine Ye Chris Olah Shan Carter

Fast approx. natural gradient with K-FAC



H_k

$\approx$



$\widehat{H}_k$

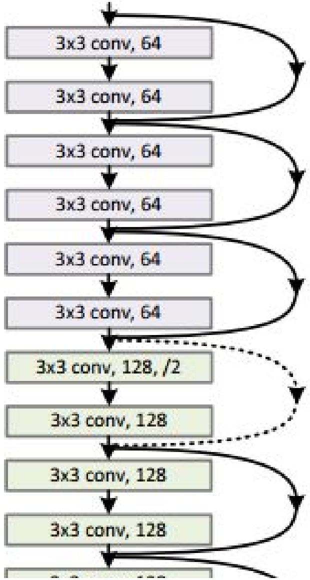
Setup: ResNet-50 on SVHN

SVHN

- 32 x 32 images
- 10 digit classes
- 600,000 examples
- Inception-style data augmentation

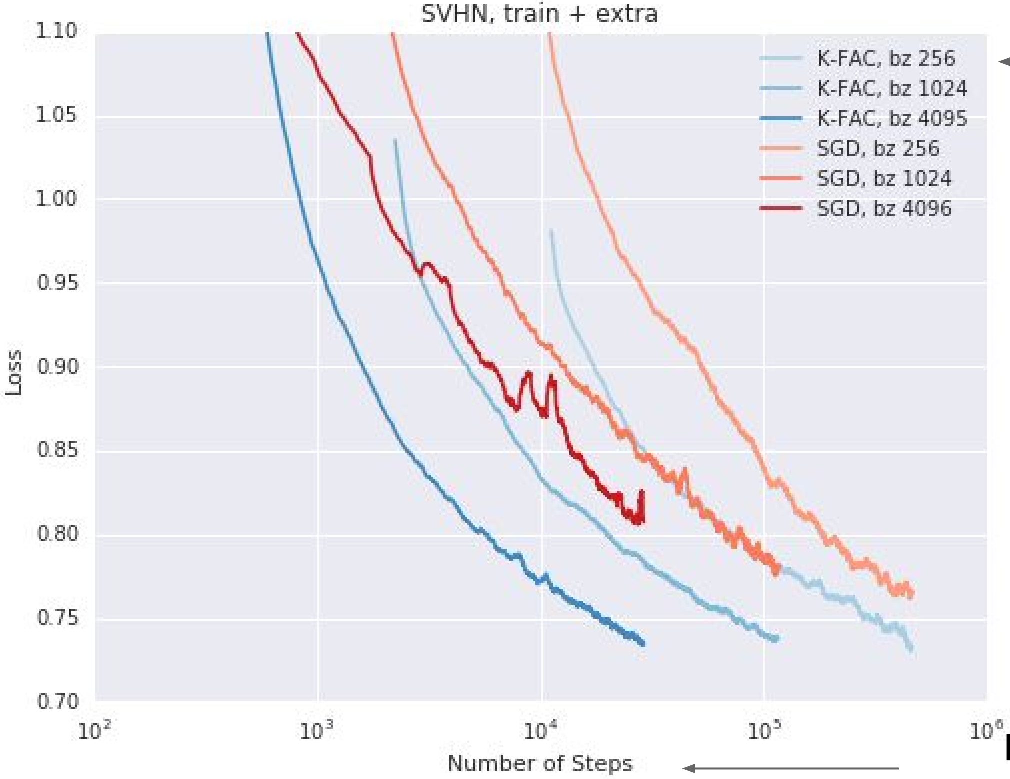
ResNet-50

- Image classification
- 50 layers
- 25.5M parameters
- 3.8B FLOPs per inference



Per-Step Progress: Loss

Training Loss



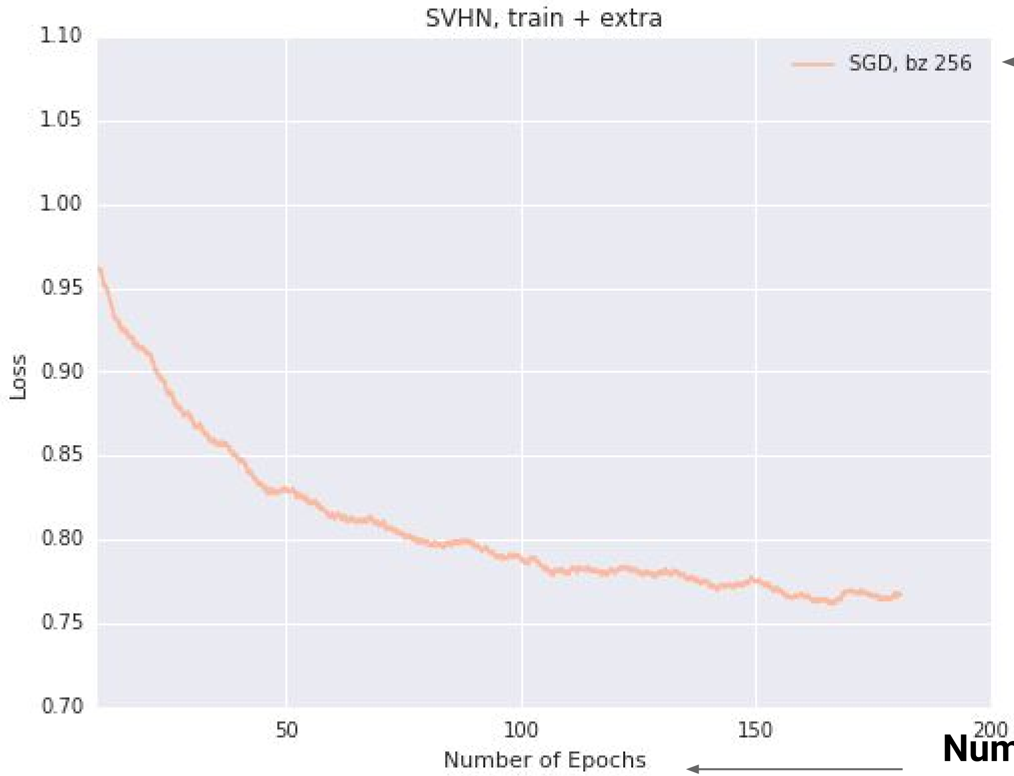
Optimizer, Batch Size

Number of parameter updates

Per-Example Progress: Loss

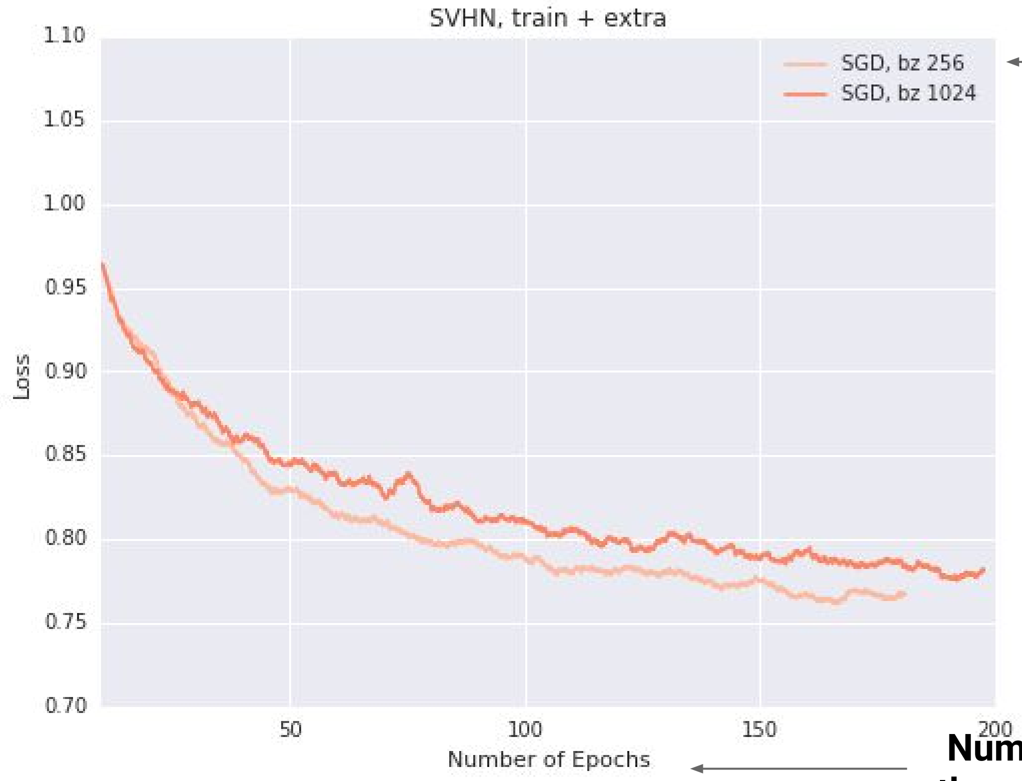
Optimizer,
Batch Size

Training
Loss



Number of Passes
through the dataset

Per-Example Progress: Loss



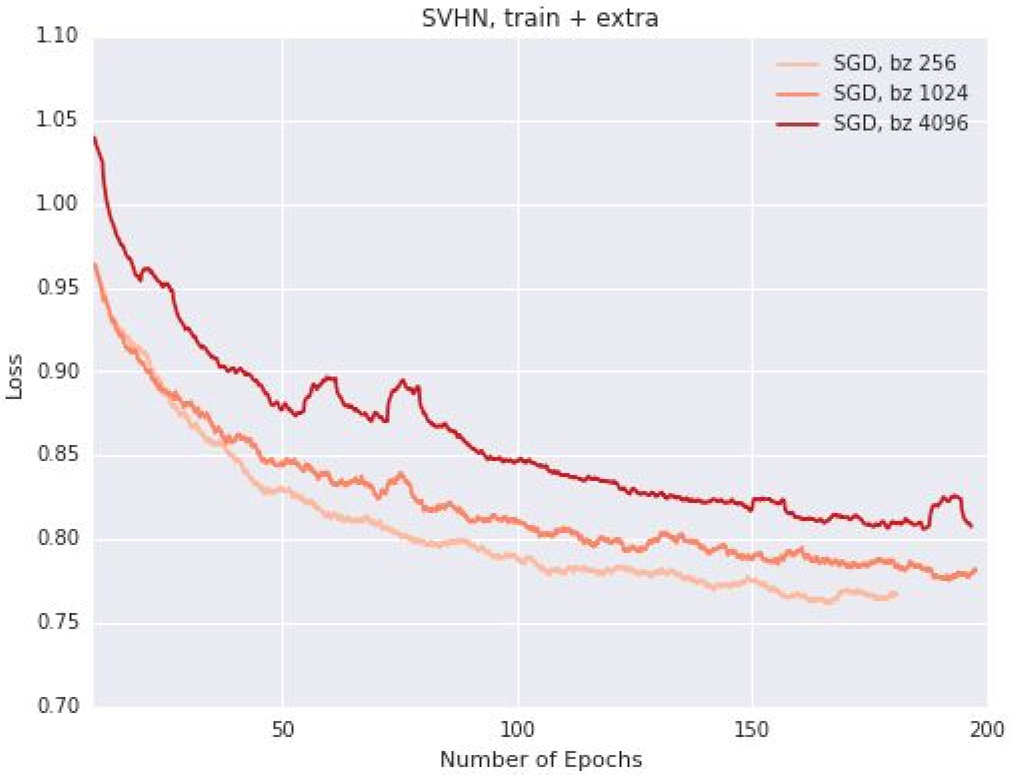
Optimizer, Batch Size

Training Loss

Number of Passes through the dataset

Per-Example Progress: Loss

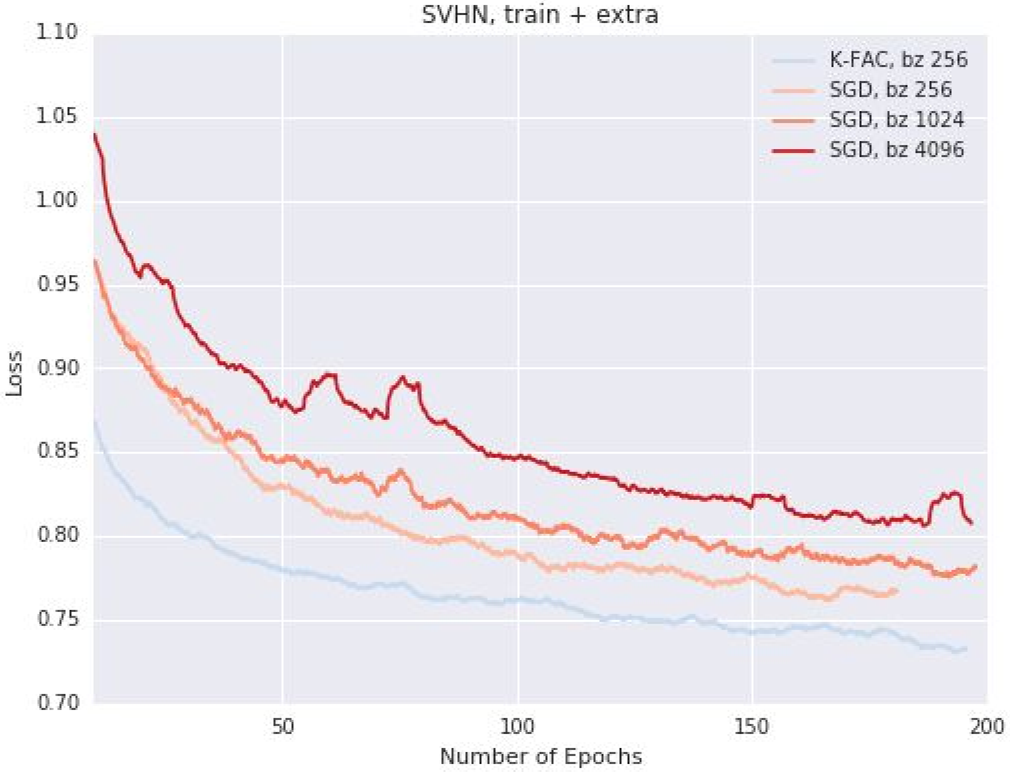
Training Loss



As batch size **increases**, SGD needs **more examples**.

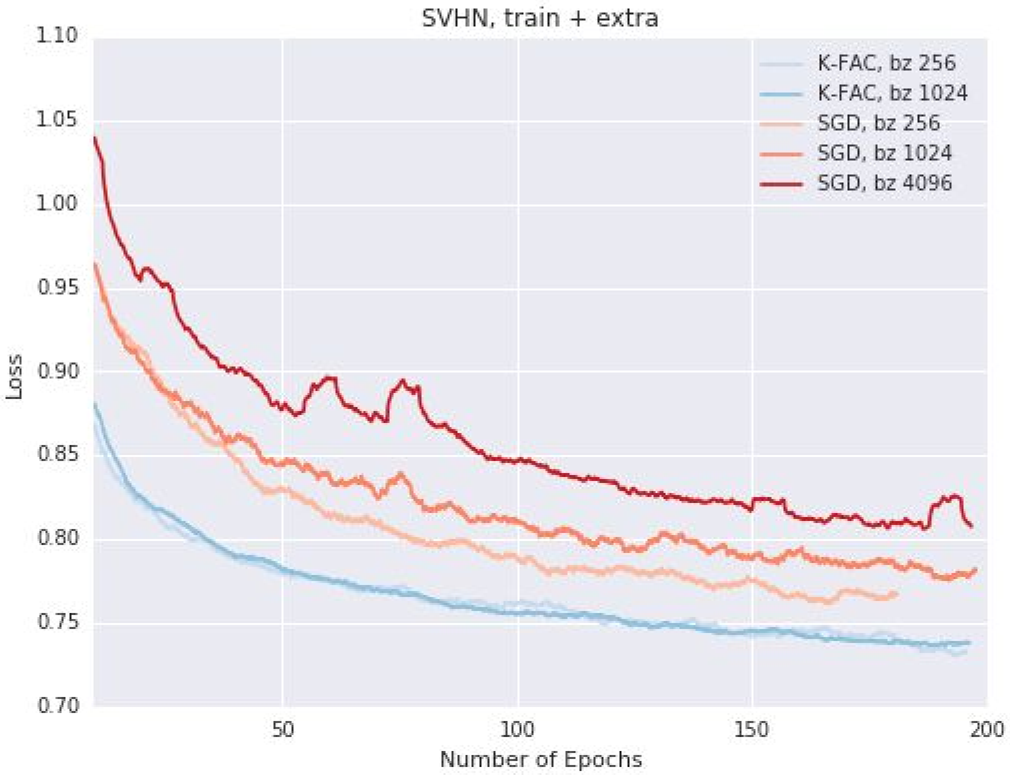
Per-Example Progress: Loss

Training Loss



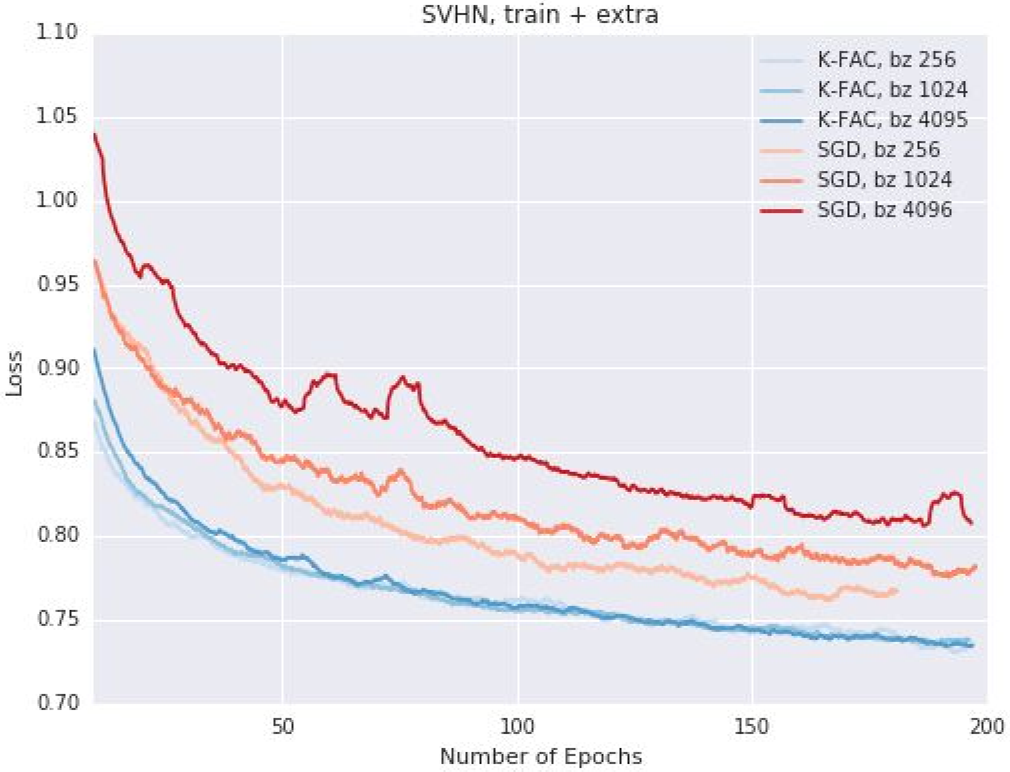
Per-Example Progress: Loss

Training Loss



Per-Example Progress: Loss

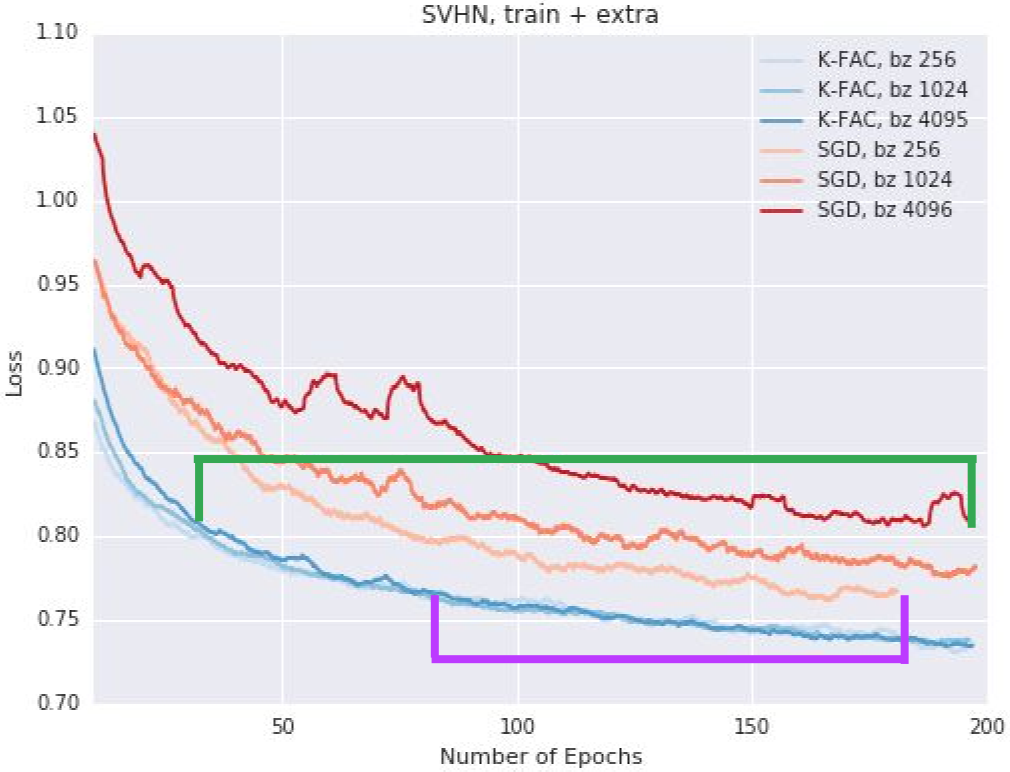
Training Loss



K-FAC converges at the **same rate**, regardless of batch size!

Per-Example Progress: Loss

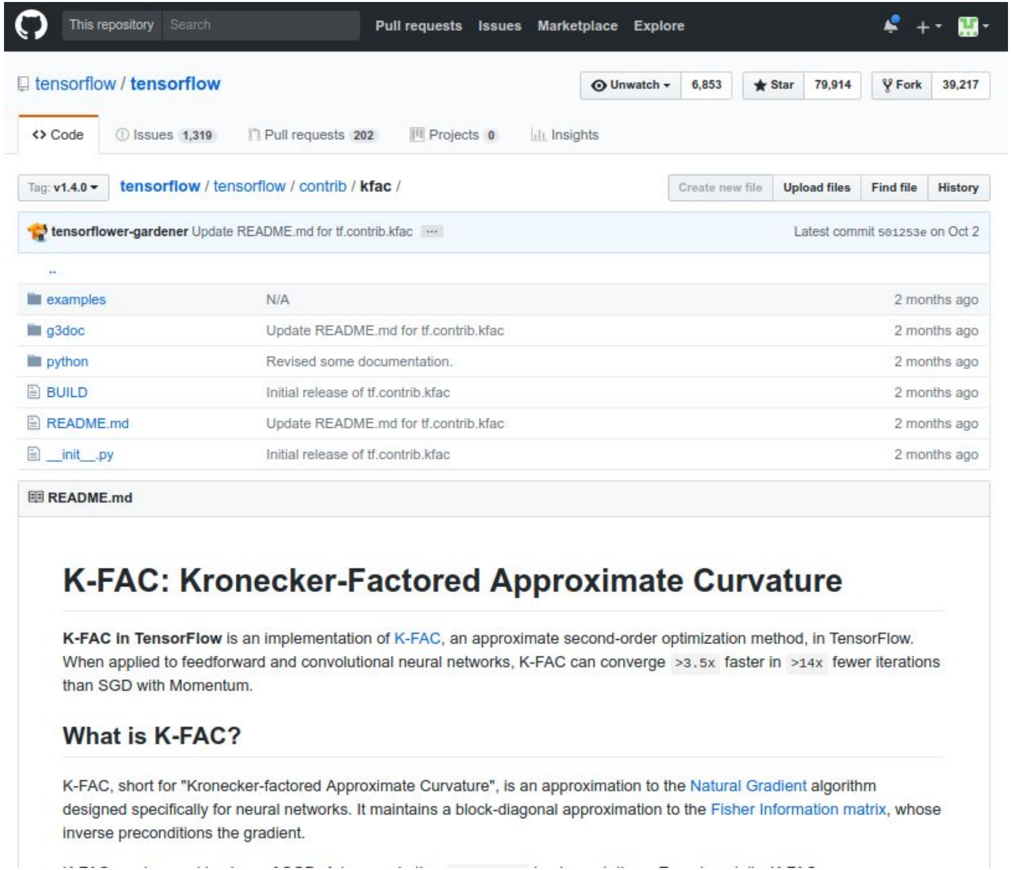
Training Loss



2.3x to **7.8x** fewer steps required.

K-FAC is ready for use today!

- `tf.contrib.kfac` comes built-in with **TensorFlow 1.4**.
- Works out-of-the-box with **feed-forward networks.**
- Works in **single-/multi-machine/GPU** training setups.
- Bonus: **Fisher Information Matrix** estimation API.



K-FAC is ready for use today!

Apply to your model with 2 changes,

```
# Build model.
def model_fn(x):
  for i in range(...):
    w, b = tf.get_variable(...), tf.get_variable(...)
    z = tf.matmul(x, w) + b

    x = tf.nn.relu(z)

  return z


logits = model_fn(x)

# Construct training ops.
optimizer = GradientDescentOptimizer(...)
train_op = optimizer.minimize(loss_fn(y, logits))

# Minimize loss.
with tf.Session() as sess:
  ...
  sess.run([train_op])
```

* Automatic layer registration coming soon!

K-FAC is ready for use today!

Apply to your model with 2 changes,

1. Register layers*

```
# Build model.
def model_fn(x, layer_collection):
  for i in range(...):
    w, b = tf.get_variable(...), tf.get_variable(...)
    z = tf.matmul(x, w) + b
    layer_collection.register_fully_connected((w, b), x, z)
    x = tf.nn.relu(z)
  layer_collection.register_categorical_predictive_distribution(z)
  return z

layer_collection = kfac.LayerCollection()
logits = model_fn(x, layer_collection)

# Construct training ops.
optimizer = GradientDescentOptimizer(...)
train_op = optimizer.minimize(loss_fn(y, logits)))

# Minimize loss.
with tf.Session() as sess:
  ...
  sess.run([train_op])
```

* Automatic layer registration coming soon!

K-FAC is ready for use today!

Apply to your model with 2 changes,

1. Register layers*
2. Apply K-FAC Optimizer

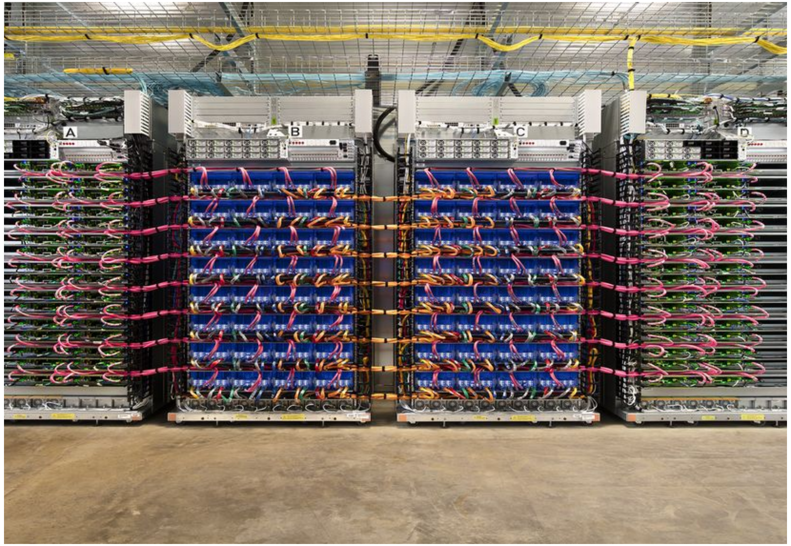
```
# Build model.
def model_fn(x, layer_collection):
  for i in range(...):
    w, b = tf.get_variable(...), tf.get_variable(...)
    z = tf.matmul(x, w) + b
    layer_collection.register_fully_connected((w, b), x, z)
    x = tf.nn.relu(z)
  layer_collection.register_categorical_predictive_distribution(z)
  return z

layer_collection = kfac.LayerCollection()
logits = model_fn(x, layer_collection)

# Construct training ops.
optimizer = kfac.KfacOptimizer(..., layer_collection=layer_collection)
train_op = optimizer.minimize(loss_fn(y, logits))

# Minimize loss.
with tf.Session() as sess:
  ...
  sess.run([train_op, optimizer.cov_update_op, optimizer.inv_update_op])
```

* Automatic layer registration coming soon!

Coming soon...

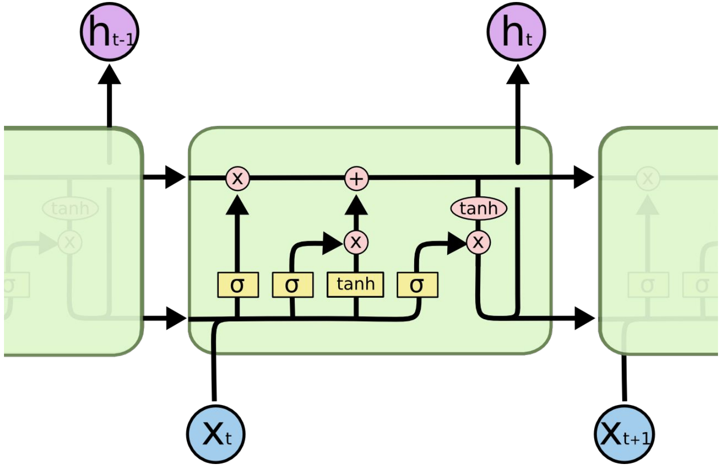
TensorFlow Processing Unit Support

Up to [11.5 PetaFLOPs]() per 256-chip Pod.
Available soon in Google Cloud.



RNN Support

[Novel Fisher Approximations]() achieve same loss as ADAM in > 5x fewer steps.



THANK YOU
to our collaborators

James Martens (DeepMind)
Roger Grosse (University of Toronto)
Jimmy Ba (University of Toronto)
James Keeling (DeepMind)
Noah Siegel (DeepMind)
Olga Wichrowska (Google Brain)
Alok Aggarwal (Google Brain)
The TensorFlow Team

Martens, James, and Roger Grosse. "Optimizing neural networks with Kronecker-factored approximate curvature." *International Conference on Machine Learning*. 2015. https://arxiv.org/abs/1503.05671

Grosse, Roger, and James Martens. "A Kronecker-factored approximate Fisher matrix for convolution layers." *International Conference on Machine Learning*. 2016. https://arxiv.org/abs/1602.01407

Ba, Jimmy, Roger Grosse, and James Martens. "Distributed Second-Order Optimization using Kronecker-Factored Approximations." (2016). https://openreview.net/forum?id=SkkTMpjex

Homepage

https://goo.gl/9WXWWK

Example Code

https://goo.gl/B6cCnm

Usage

```
import
tensorflow.contrib.kfac
```

Matt Johnson
Research Scientist

Daniel Duckworth
Research Engineer