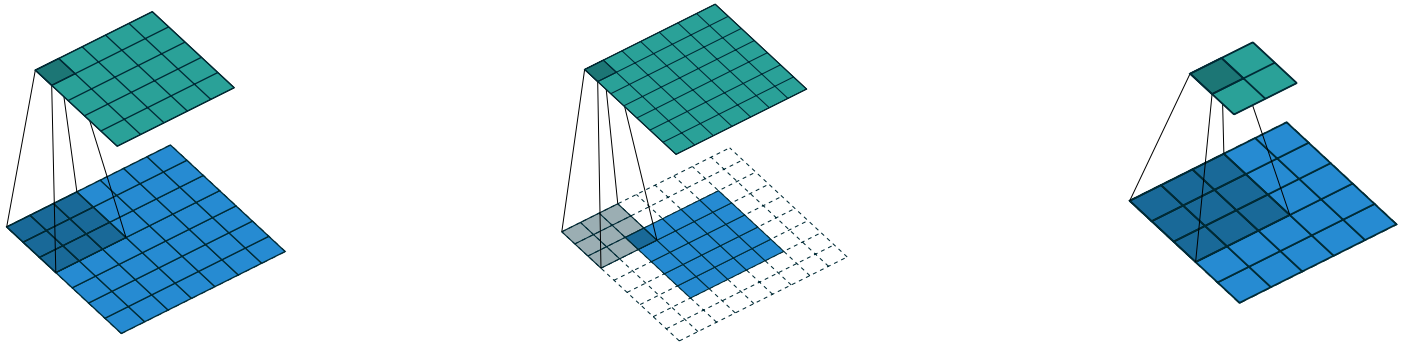
MIT 9.520/6.860, Fall 2018

Class 11: Neural networks – tips, tricks & software

Andrzej Banburski

Last time - Convolutional neural networks

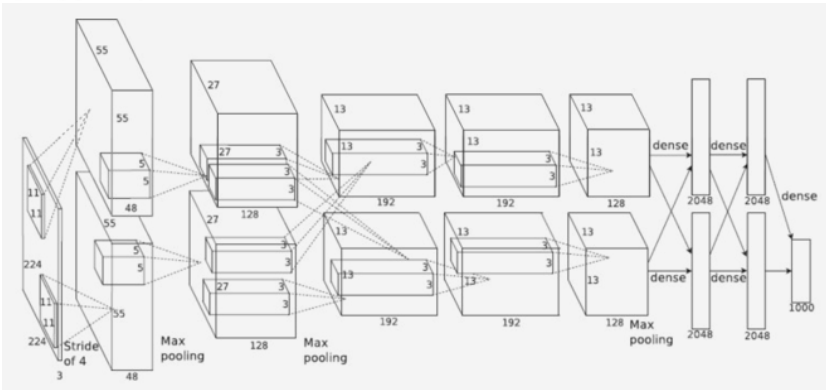


source: github.com/vdumoulin/conv_arithmetic

Large-scale Datasets

General Purpose GPUs

AlexNet
Krizhevsky
et al (2012)



Overview

Initialization & hyper-parameter tuning

Optimization algorithms

Batchnorm & Dropout

Finite dataset woes

Software

Initialization & hyper-parameter tuning

Consider the problem of training a neural network $f_\theta(x)$ by minimizing a loss

$$L(\theta, x) = \sum_{i=1}^{N} l_i(y_i, f_\theta(x_i)) + \lambda|\theta|^2$$

with SGD and mini-batch size b:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla_\theta L(\theta_t, x_i) \tag{1}$$

Initialization & hyper-parameter tuning

Consider the problem of training a neural network $f_\theta(x)$ by minimizing a loss

$$L(\theta, x) = \sum_{i=1}^{N} l_i(y_i, f_\theta(x_i)) + \lambda|\theta|^2$$

with SGD and mini-batch size b:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla_\theta L(\theta_t, x_i) \tag{1}$$

- How should we choose the initial set of parameters θ?

Initialization & hyper-parameter tuning

Consider the problem of training a neural network $f_\theta(x)$ by minimizing a loss

$$L(\theta, x) = \sum_{i=1}^{N} l_i(y_i, f_\theta(x_i)) + \lambda|\theta|^2$$

with SGD and mini-batch size b:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla_\theta L(\theta_t, x_i) \tag{1}$$

- How should we choose the initial set of parameters θ?
- How about the hyper-parameters η, λ and b?

Weight Initialization

- First obvious observation: starting with 0 will make every weight update in the same way. Similarly, too big and we can run into NaN.

Weight Initialization

- First obvious observation: starting with 0 will make every weight update in the same way. Similarly, too big and we can run into NaN.
- What about $\theta_0 = \epsilon \times \mathcal{N}(0, 1)$, with $\epsilon \approx 10^{-2}$?

Weight Initialization

- First obvious observation: starting with 0 will make every weight update in the same way. Similarly, too big and we can run into NaN.
- What about $\theta_0 = \epsilon \times \mathcal{N}(0, 1)$, with $\epsilon \approx 10^{-2}$?
- For a few layers this would seem to work nicely.

Weight Initialization

- First obvious observation: starting with 0 will make every weight update in the same way. Similarly, too big and we can run into NaN.
- What about $\theta_0 = \epsilon \times \mathcal{N}(0, 1)$, with $\epsilon \approx 10^{-2}$?
- For a few layers this would seem to work nicely.
- If we go deeper however...

Weight Initialization

- First obvious observation: starting with 0 will make every weight update in the same way. Similarly, too big and we can run into NaN.
- What about $\theta_0 = \epsilon \times \mathcal{N}(0, 1)$, with $\epsilon \approx 10^{-2}$?
- For a few layers this would seem to work nicely.
- If we go deeper however...
- Super slow update of earlier layers 10^{-2L} for sigmoid or tanh activations – vanishing gradients. ReLU activations do not suffer so much from this.

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\mathsf{Var}(y) = \mathsf{Var}(Wx) = \mathsf{Var}(w_1 x_1) + \cdots + \mathsf{Var}(w_{n_{in}} x_{n_{in}})$$

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\mathrm{Var}(y) = \mathrm{Var}(Wx) = \mathrm{Var}(w_1 x_1) + \cdots + \mathrm{Var}(w_{n_{in}} x_{n_{in}})$$

- If we assume that W and x are i.i.d. and have zero mean, then $\mathrm{Var}(y) = n\mathrm{Var}(w_i)\mathrm{Var}(x_i)$

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\text{Var}(y) = \text{Var}(Wx) = \text{Var}(w_1 x_1) + \cdots + \text{Var}(w_{n_{in}} x_{n_{in}})$$

- If we assume that W and x are i.i.d. and have zero mean, then $\text{Var}(y) = n\text{Var}(w_i)\text{Var}(x_i)$
- If we want the inputs and outputs to have same variance, this gives us $\text{Var}(w_i) = \frac{1}{n_{in}}$.

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\text{Var}(y) = \text{Var}(Wx) = \text{Var}(w_1 x_1) + \cdots + \text{Var}(w_{n_{in}} x_{n_{in}})$$

- If we assume that W and x are i.i.d. and have zero mean, then $\text{Var}(y) = n\text{Var}(w_i)\text{Var}(x_i)$
- If we want the inputs and outputs to have same variance, this gives us $\text{Var}(w_i) = \frac{1}{n_{in}}$.
- Similar analysis for backward pass gives $\text{Var}(w_i) = \frac{1}{n_{out}}$.

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\text{Var}(y) = \text{Var}(Wx) = \text{Var}(w_1 x_1) + \cdots + \text{Var}(w_{n_{in}} x_{n_{in}})$$

- If we assume that W and x are i.i.d. and have zero mean, then $\text{Var}(y) = n\text{Var}(w_i)\text{Var}(x_i)$
- If we want the inputs and outputs to have same variance, this gives us $\text{Var}(w_i) = \frac{1}{n_{in}}$.
- Similar analysis for backward pass gives $\text{Var}(w_i) = \frac{1}{n_{out}}$.
- The compromise is the Xavier initialization [Glorot et al., 2010]:

$$\text{Var}(w_i) = \frac{2}{n_{in} + n_{out}} \tag{2}$$

Xavier & He initializations

- For tanh and sigmoid activations, near origin we deal with a nearly linear function $y = Wx$, with $x = (x_1, \ldots, x_{n_{in}})$. To stop vanishing and exploding gradients we need

$$\text{Var}(y) = \text{Var}(Wx) = \text{Var}(w_1 x_1) + \cdots + \text{Var}(w_{n_{in}} x_{n_{in}})$$

- If we assume that W and x are i.i.d. and have zero mean, then $\text{Var}(y) = n\text{Var}(w_i)\text{Var}(x_i)$
- If we want the inputs and outputs to have same variance, this gives us $\text{Var}(w_i) = \frac{1}{n_{in}}$.
- Similar analysis for backward pass gives $\text{Var}(w_i) = \frac{1}{n_{out}}$.
- The compromise is the Xavier initialization [Glorot et al., 2010]:

$$\text{Var}(w_i) = \frac{2}{n_{in} + n_{out}} \tag{2}$$

- Heuristically, ReLU is half of the linear function, so we can take

$$\text{Var}(w_i) = \frac{4}{n_{in} + n_{out}} \tag{3}$$

An analysis in [He et al., 2015] confirms this.

Hyper-parameter tuning

How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?

Hyper-parameter tuning

How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?
- Basic idea: split your training dataset into a smaller training set and a cross-validation set.

Hyper-parameter tuning

How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?
- Basic idea: split your training dataset into a smaller training set and a cross-validation set.
  - Run a coarse search (on a logarithmic scale) over the parameters for just a few epochs of SGD and evaluate on the cross-validation set.

Hyper-parameter tuning

How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?
- Basic idea: split your training dataset into a smaller training set and a cross-validation set.
  - Run a coarse search (on a logarithmic scale) over the parameters for just a few epochs of SGD and evaluate on the cross-validation set.
  - Perform a finer search.

Hyper-parameter tuning

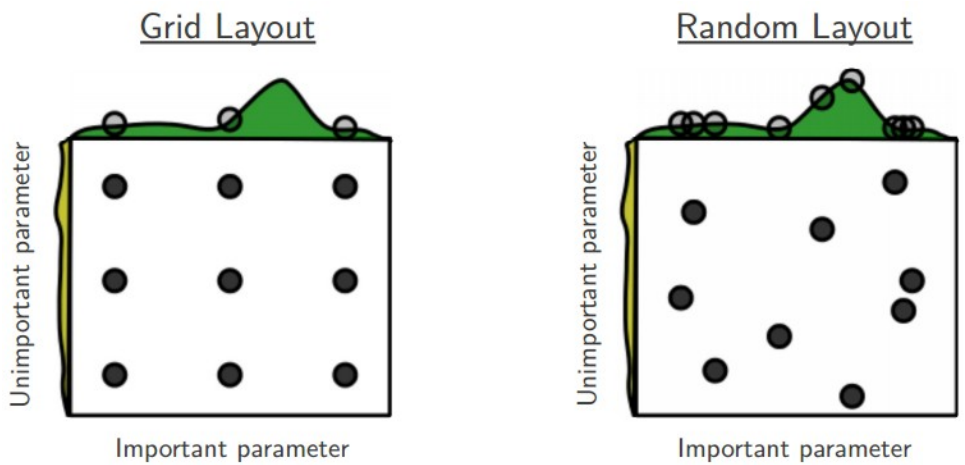
How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?
- Basic idea: split your training dataset into a smaller training set and a cross-validation set.
  - Run a coarse search (on a logarithmic scale) over the parameters for just a few epochs of SGD and evaluate on the cross-validation set.
  - Perform a finer search.
- Interestingly, [Bergstra and Bengio, 2012] shows that it is better to run the search randomly than on a grid.

Hyper-parameter tuning

How about the hyper-parameters η, λ and b

- How do we choose optimal η, λ and b?
- Basic idea: split your training dataset into a smaller training set and a cross-validation set.
  - Run a coarse search (on a logarithmic scale) over the parameters for just a few epochs of SGD and evaluate on the cross-validation set.
  - Perform a finer search.
- Interestingly, [Bergstra and Bengio, 2012] shows that it is better to run the search randomly than on a grid.



source: [Bergstra and Bengio, 2012]

Decaying learning rate

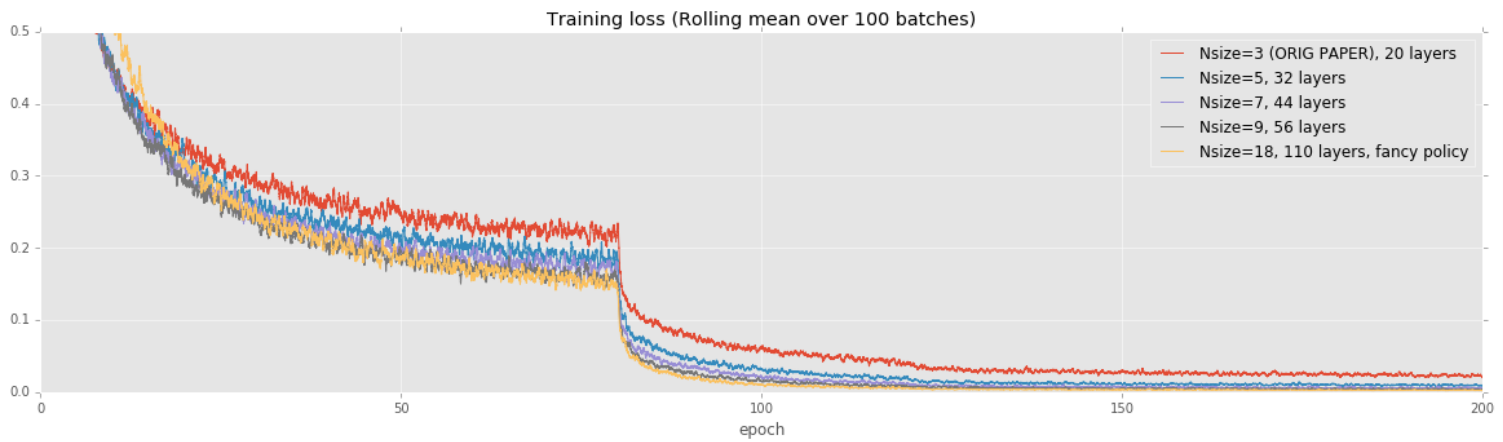
- To improve convergence of SGD, we have to use a decaying learning rate.

Decaying learning rate

- To improve convergence of SGD, we have to use a decaying learning rate.
- Typically we use a scheduler – decrease η after some fixed number of epochs.

Decaying learning rate

- To improve convergence of SGD, we have to use a decaying learning rate.
- Typically we use a scheduler – decrease η after some fixed number of epochs.
- This allows the training loss to keep improving after it has plateaued



Batch-size & learning rate

An interesting linear scaling relationship seems to exist between the learning rate η and mini-batch size b:

- In the SGD update, they appear as a ratio $\frac{\eta}{b}$, with an additional implicit dependence of the sum of gradients on b.

Batch-size & learning rate

An interesting linear scaling relationship seems to exist between the learning rate η and mini-batch size b:

- In the SGD update, they appear as a ratio $\frac{\eta}{b}$, with an additional implicit dependence of the sum of gradients on b.
- If $b \ll N$, we can approximate SGD by a stochastic differential equation with a noise scale $g \approx \eta \frac{N}{b}$ [Smit & Le, 2017].

Batch-size & learning rate

An interesting linear scaling relationship seems to exist between the learning rate η and mini-batch size b:

- In the SGD update, they appear as a ratio $\frac{\eta}{b}$, with an additional implicit dependence of the sum of gradients on b.
- If $b \ll N$, we can approximate SGD by a stochastic differential equation with a noise scale $g \approx \eta \frac{N}{b}$ [Smit & Le, 2017].
- This means that instead of decaying η, we can increase batch size dynamically.

Batch-size & learning rate

An interesting linear scaling relationship seems to exist between the learning rate η and mini-batch size b:

- In the SGD update, they appear as a ratio $\frac{\eta}{b}$, with an additional implicit dependence of the sum of gradients on b.
- If $b \ll N$, we can approximate SGD by a stochastic differential equation with a noise scale $g \approx \eta \frac{N}{b}$ [Smit & Le, 2017].
- This means that instead of decaying η, we can increase batch size dynamically.

source: [Smith et al., 2018]

Batch-size & learning rate

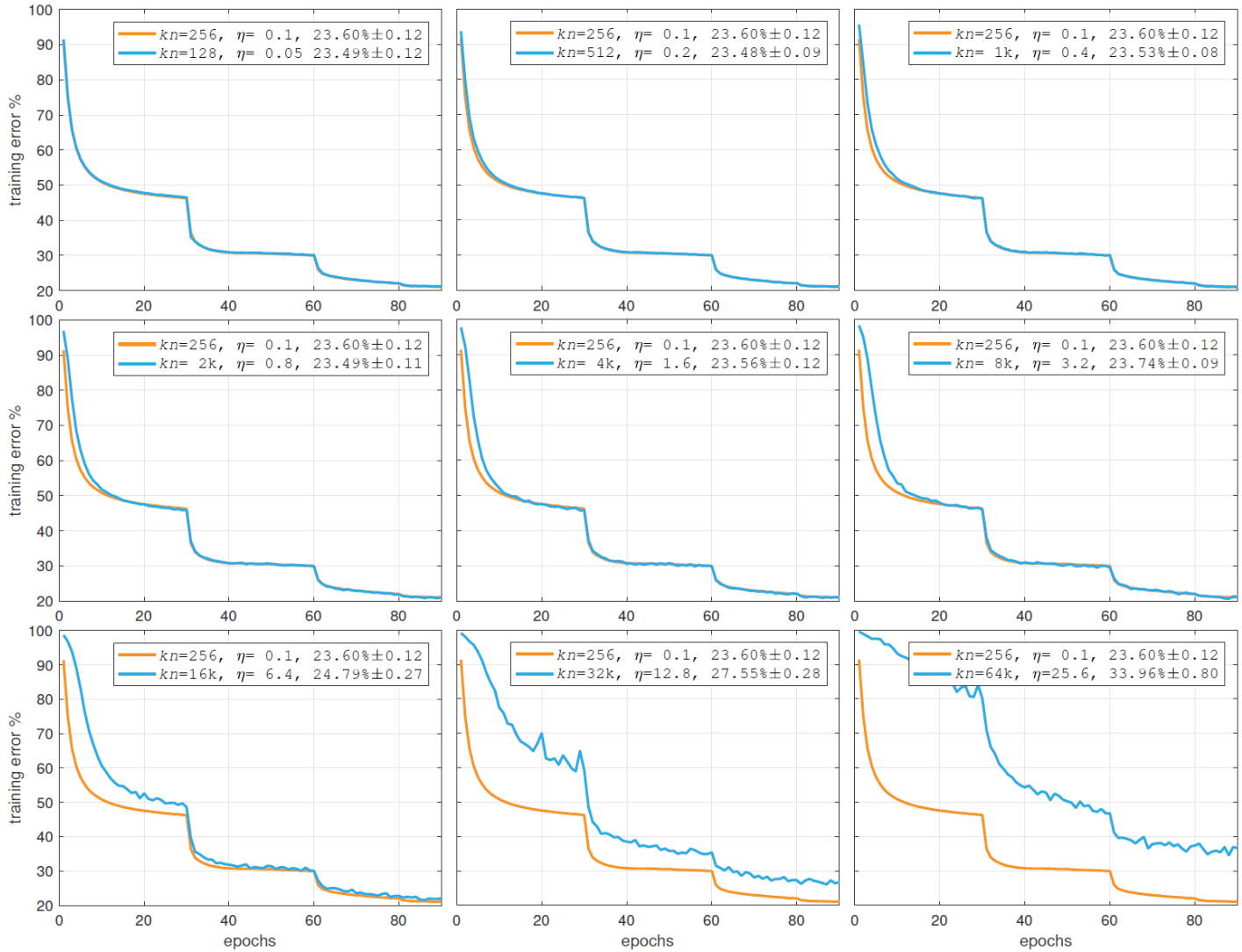
An interesting linear scaling relationship seems to exist between the learning rate η and mini-batch size b:

- In the SGD update, they appear as a ratio $\frac{\eta}{b}$, with an additional implicit dependence of the sum of gradients on b.
- If $b \ll N$, we can approximate SGD by a stochastic differential equation with a noise scale $g \approx \eta \frac{N}{b}$ [Smit & Le, 2017].
- This means that instead of decaying η, we can increase batch size dynamically.

source: [Smith et al., 2018]

- As b approaches N the dynamics become more and more deterministic and we would expect this relationship to vanish.

Batch-size & learning rate



source: [Goyal et al., 2017]

Overview

Initialization & hyper-parameter tuning

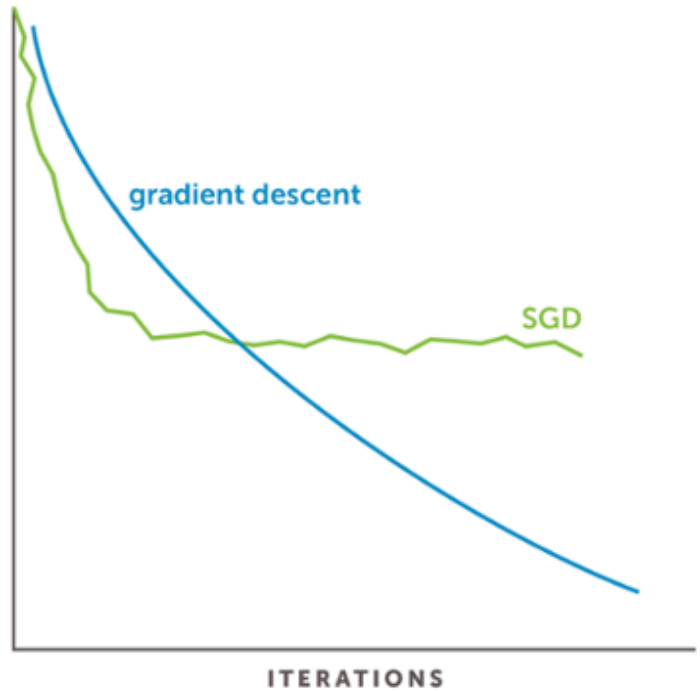
Optimization algorithms

Batchnorm & Dropout

Finite dataset woes

Software

SGD is kinda slow...



- GD – use all points each iteration to compute gradient
- SGD – use one point each iteration to compute gradient
- **Faster:** Mini-Batch – use a *mini-batch* of points each iteration to compute gradient

Alternatives to SGD

Are there reasonable alternatives outside of Newton method?

Accelerations

- **Momentum**
- Nesterov's method
- Adagrad
- RMSprop
- Adam
- ...

SGD with Momentum

We can try accelerating SGD

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

by adding a momentum/velocity term:

SGD with Momentum

We can try accelerating SGD

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

by adding a momentum/velocity term:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{4}$$

μ is a new "momentum" hyper-parameter.

SGD with Momentum

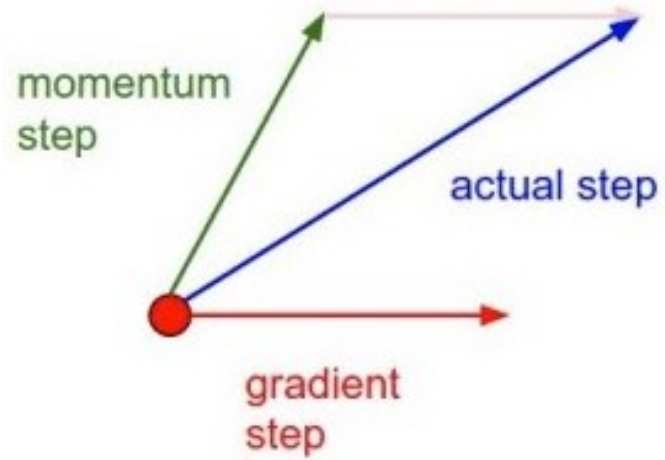
We can try accelerating SGD

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$$

by adding a momentum/velocity term:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f(\theta_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{4}$$

μ is a new "momentum" hyper-parameter.



source: cs213n.github.io

Nesterov Momentum

- Sometimes the momentum update can overshoot

Nesterov Momentum

- Sometimes the momentum update can overshoot
- We can instead evaluate the gradient at the point where momentum takes us:

Nesterov Momentum

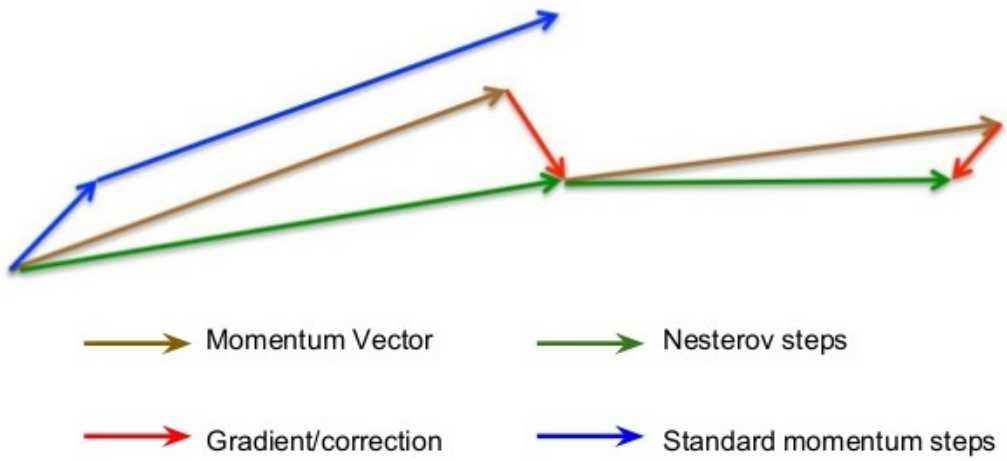
- Sometimes the momentum update can overshoot
- We can instead evaluate the gradient at the point where momentum takes us:

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \nabla f(\theta_t + \mu v_t) \\ \theta_{t+1} &= \theta_t + v_{t+1} \end{aligned} \tag{5}$$

Nesterov Momentum

- Sometimes the momentum update can overshoot
- We can instead evaluate the gradient at the point where momentum takes us:

$$
\begin{aligned}
v_{t+1} &= \mu v_t - \eta \nabla f(\theta_t + \mu v_t) \\
\theta_{t+1} &= \theta_t + v_{t+1}
\end{aligned}
\tag{5}
$$



source: Geoff Hinton's lecture

AdaGrad

- An alternative way is to automatize the decay of the learning rate.

AdaGrad

- An alternative way is to automatize the decay of the learning rate.
- The Adaptive Gradient algorithm does this by accumulating magnitudes of gradients

AdaGrad

- An alternative way is to automatize the decay of the learning rate.
- The Adaptive Gradient algorithm does this by accumulating magnitudes of gradients

Algorithm 4 AdaGrad

Require: Global learning rate η

Require: Initial parameter θ

Initialize gradient accumulation variable $r = 0$

while Stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$.

Apply interim update: $\theta \leftarrow \theta + \rho v$

Set $g = 0$

for $i = 1$ to m **do**

Compute gradient:

$$g \leftarrow g + \nabla_\theta L(f(x^{(i)}; \theta)), y^{(i)}; \theta).$$

end for

Accumulate gradient: $r \leftarrow r + g^2$ (square is applied element-wise)

Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{r}} g$ ($\frac{1}{\sqrt{r}}$ is applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta_t$

end while

AdaGrad

- An alternative way is to automatize the decay of the learning rate.
- The Adaptive Gradient algorithm does this by accumulating magnitudes of gradients

Algorithm 4 AdaGrad

Require: Global learning rate η

Require: Initial parameter θ

Initialize gradient accumulation variable $r = 0$

while Stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$.

Apply interim update: $\theta \leftarrow \theta + \rho v$

Set $g = 0$

for $i = 1$ to m **do**

Compute gradient:

$$g \leftarrow g + \nabla_\theta L(f(x^{(i)}; \theta)), y^{(i)}; \theta).$$

end for

Accumulate gradient: $r \leftarrow r + g^2$ (square is applied element-wise)

Compute update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{r}} g$ ($\frac{1}{\sqrt{r}}$ is applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta_t$

end while

- AdaGrad accelerates in flat directions of optimization landscape and slows down in step ones.

RMSProp

Problem:The updates in AdaGrad always decrease the learning rate, so some of the parameters can become un-learnable.

RMSProp

Problem:The updates in AdaGrad always decrease the learning rate, so some of the parameters can become un-learnable.

- Fix by Hinton: use weighted sum of the square magnitudes instead.

RMSProp

Problem:The updates in AdaGrad always decrease the learning rate, so some of the parameters can become un-learnable.

- Fix by Hinton: use weighted sum of the square magnitudes instead.
- This assigns more weight to recent iterations. Useful if directions of steeper or shallower descent suddenly change.

RMSProp

Problem:The updates in AdaGrad always decrease the learning rate, so some of the parameters can become un-learnable.

- Fix by Hinton: use weighted sum of the square magnitudes instead.
- This assigns more weight to recent iterations. Useful if directions of steeper or shallower descent suddenly change.

Algorithm 5 RMSprop

Require: Global learning rate η, decay rate ρ

Require: Initial parameter θ

Initialize accumulation variable $r = 0$

while Stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$.

Set $g = 0$

for $i = 1$ to m **do**

Compute gradient:

$$g \leftarrow g + \nabla_\theta L(f(x^{(i)}; \theta)), y^{(i)}; \theta).$$

end for

Accumulate gradient: $r \leftarrow \rho r + (1 - \rho) g^2$

Compute parameter update: $\Delta\theta \leftarrow -\frac{\eta}{\sqrt{r}} g$ ($\frac{1}{\sqrt{r}}$ is applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta_t$

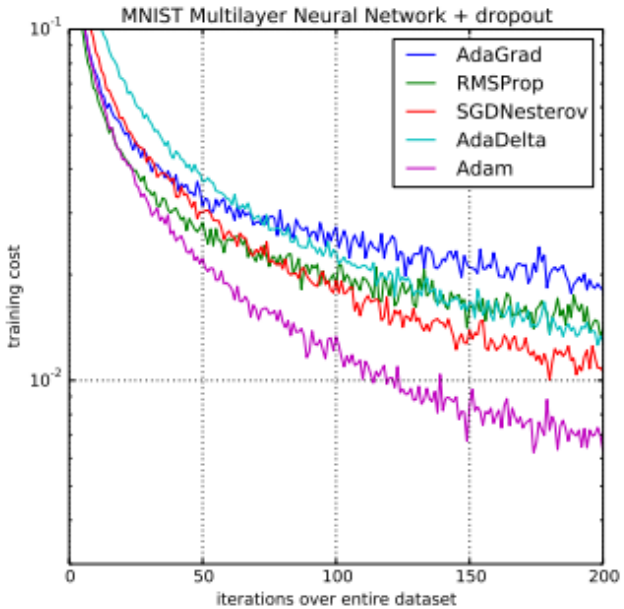
end while

Adam

Adaptive Moment – a combination of the previous approaches.

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while



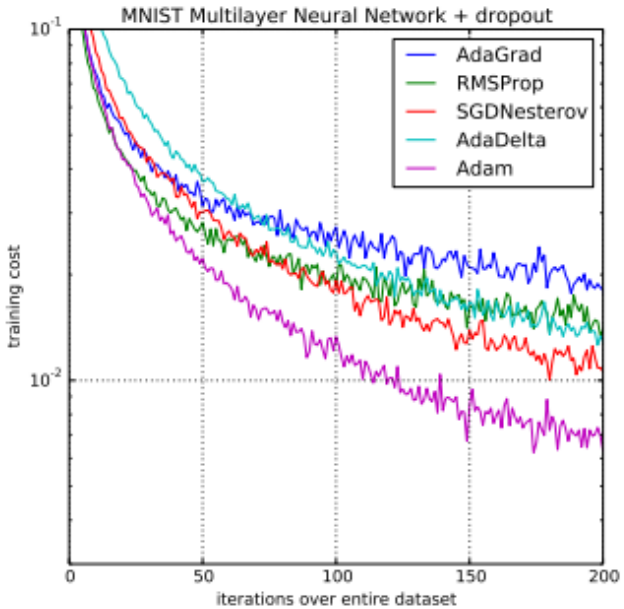
[Kingma and Ba, 2014]

Adam

Adaptive Moment – a combination of the previous approaches.

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while



[Kingma and Ba, 2014]

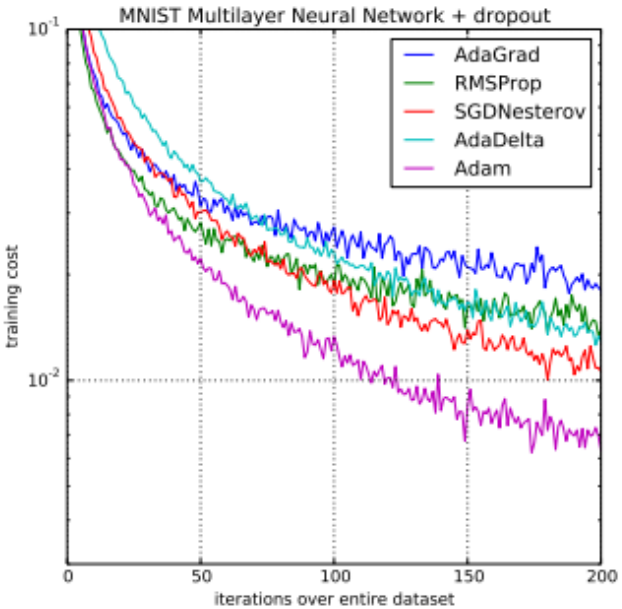
- Ridiculously popular – more than 13K citations!

Adam

Adaptive Moment – a combination of the previous approaches.

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while



[Kingma and Ba, 2014]

- Ridiculously popular – more than 13K citations!
- Probably because it comes with recommended parameters and came with a proof of convergence (which was shown to be wrong).

So what should I use in practice?

- **Adam** is a good default in many cases.

So what should I use in practice?

- **Adam** is a good default in many cases.
- There exist datasets in which Adam and other adaptive methods do not generalize to unseen data at all! [Marginal Value of Adaptive Gradient Methods in Machine Learning]

So what should I use in practice?

- **Adam** is a good default in many cases.
- There exist datasets in which Adam and other adaptive methods do not generalize to unseen data at all! [Marginal Value of Adaptive Gradient Methods in Machine Learning]
- SGD with Momentum and a decay rate often outperforms Adam

(but requires tuning).

So what should I use in practice?

- **Adam** is a good default in many cases.
- There exist datasets in which Adam and other adaptive methods do not generalize to unseen data at all! [Marginal Value of Adaptive Gradient Methods in Machine Learning]
- SGD with Momentum and a decay rate often outperforms Adam

(but requires tuning). includegraphicsFigures/comp.png source: github.com/YingzhenLi

Overview

Initialization & hyper-parameter tuning

Optimization algorithms

Batchnorm & Dropout

Finite dataset woes

Software

Data pre-processing

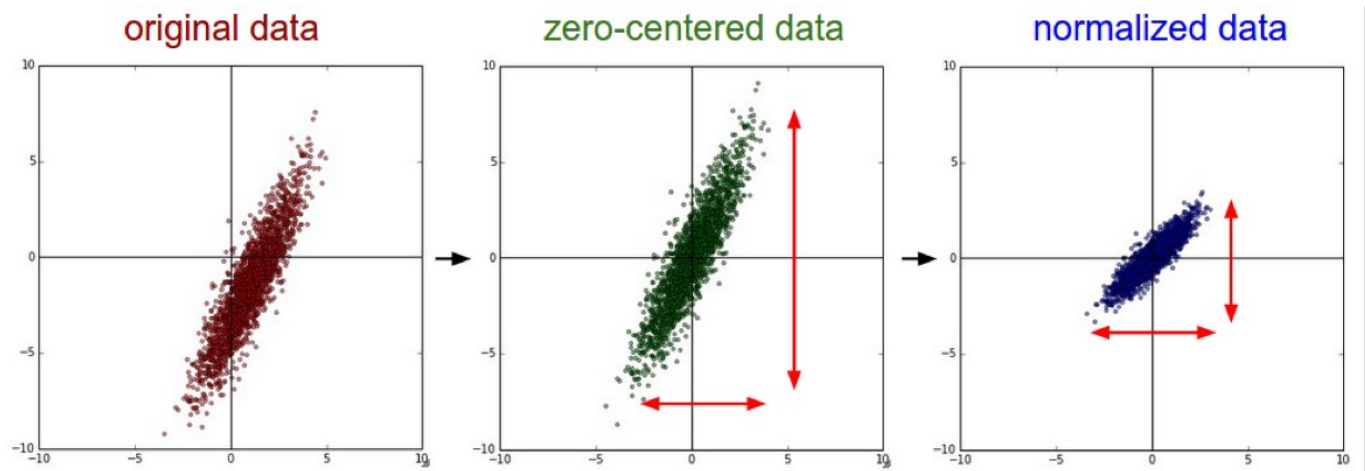
Since our non-linearities change their behavior around the origin, it makes sense to pre-process to zero-mean and unit variance.

$$\hat{x}_i = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{\mathsf{Var}[x_i]}} \tag{6}$$

Data pre-processing

Since our non-linearities change their behavior around the origin, it makes sense to pre-process to zero-mean and unit variance.

$$\hat{x}_i = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{\mathsf{Var}[x_i]}} \tag{6}$$



source: cs213n.github.io

Batch Normalization

A common technique is to repeat this throughout the deep network in a differentiable way:

Batch Normalization

A common technique is to repeat this throughout the deep network in a differentiable way:

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

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

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

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

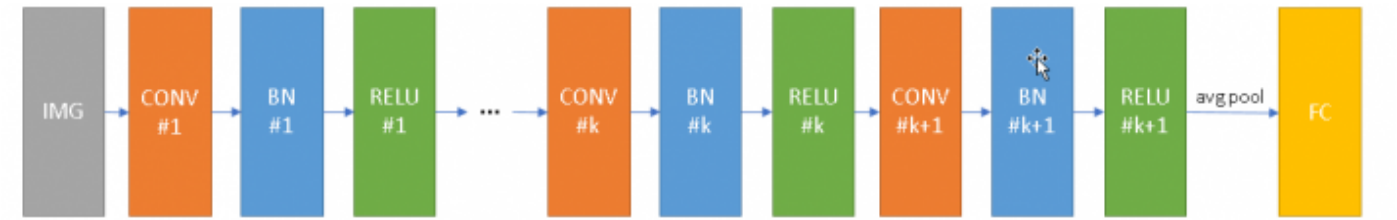
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

[Ioffe and Szegedy, 2015]

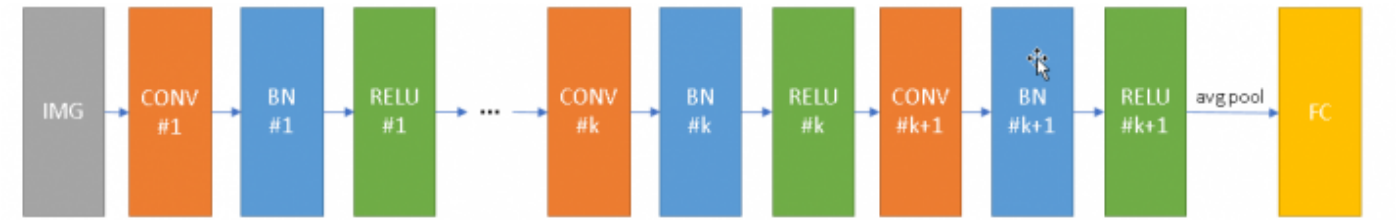
Batch Normalization

In practice, a batchnorm layer is added after a conv or fully-connected layer, but before activations.



Batch Normalization

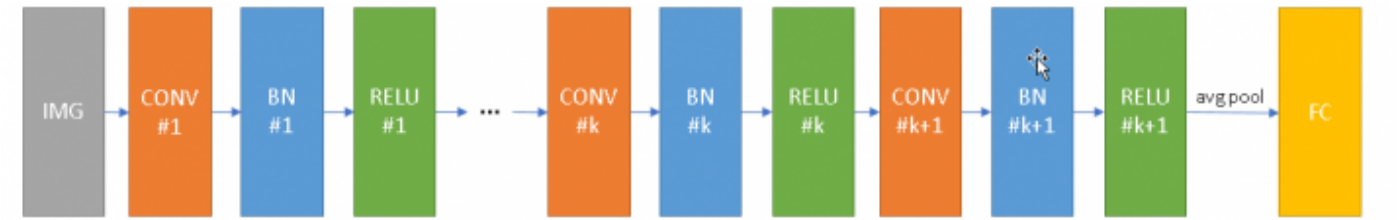
In practice, a batchnorm layer is added after a conv or fully-connected layer, but before activations.



- In the original paper the authors claimed that this is meant to reduce *covariate shift*.

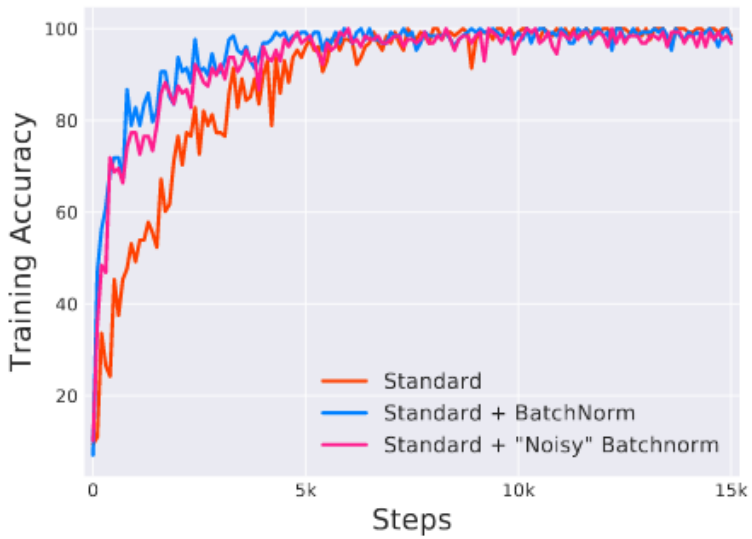
Batch Normalization

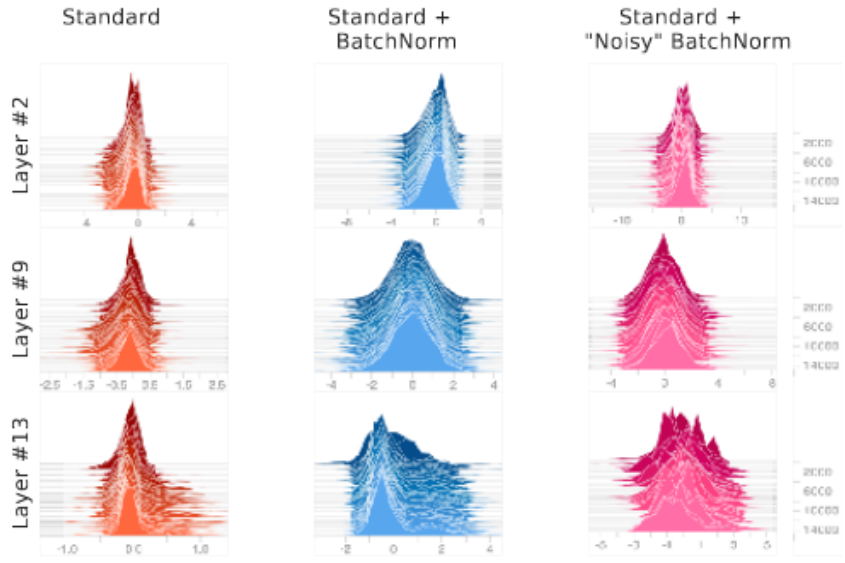
In practice, a batchnorm layer is added after a conv or fully-connected layer, but before activations.



- In the original paper the authors claimed that this is meant to reduce *covariate shift*.
- More obviously, this reduces 2nd-order correlations between layers. Recently shown that it actually doesn't change covariate shift! Instead it smooths out the landscape.

Batch Normalization



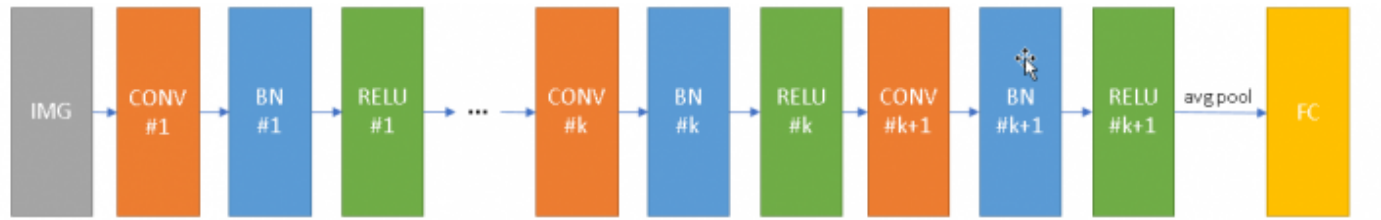


- More obviously, this reduces 2nd-order correlations between layers. Recently shown that it actually doesn't change covariate shift! Instead it smooths out the landscape.

[Santurkar, Tsipras, Ilyas, Madry, 2018]

Batch Normalization

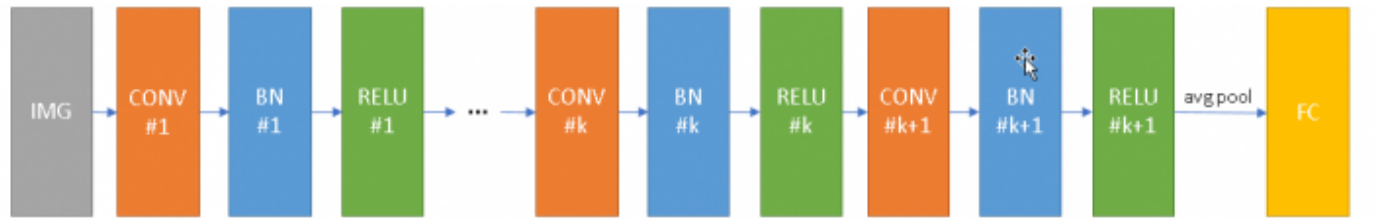
In practice, a batchnorm layer is added after a conv or fully-connected layer, but before activations.



- In the original paper the authors claimed that this is meant to reduce *covariate shift*.
- More obviously, this reduces 2nd-order correlations between layers. Recently shown that it actually doesn't change covariate shift! Instead it smooths out the landscape.
- In practice this reduces dependence on initialization and seems to stabilize the flow of gradient descent.

Batch Normalization

In practice, a batchnorm layer is added after a conv or fully-connected layer, but before activations.



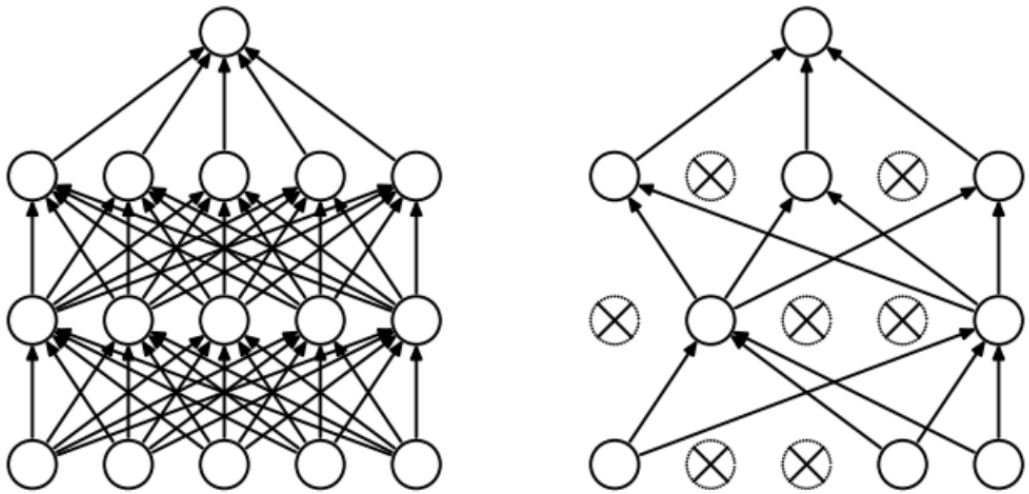
- In the original paper the authors claimed that this is meant to reduce *covariate shift*.
- More obviously, this reduces 2nd-order correlations between layers. Recently shown that it actually doesn't change covariate shift! Instead it smooths out the landscape.
- In practice this reduces dependence on initialization and seems to stabilize the flow of gradient descent.
- Using BN usually nets you a gain of few % increase in test accuracy.

Dropout

Another common technique: during forward pass, set some of the weights to 0 randomly with probability p. Typical choice is $p = 50\%$.

Dropout

Another common technique: during forward pass, set some of the weights to 0 randomly with probability p. Typical choice is $p = 50\%$.

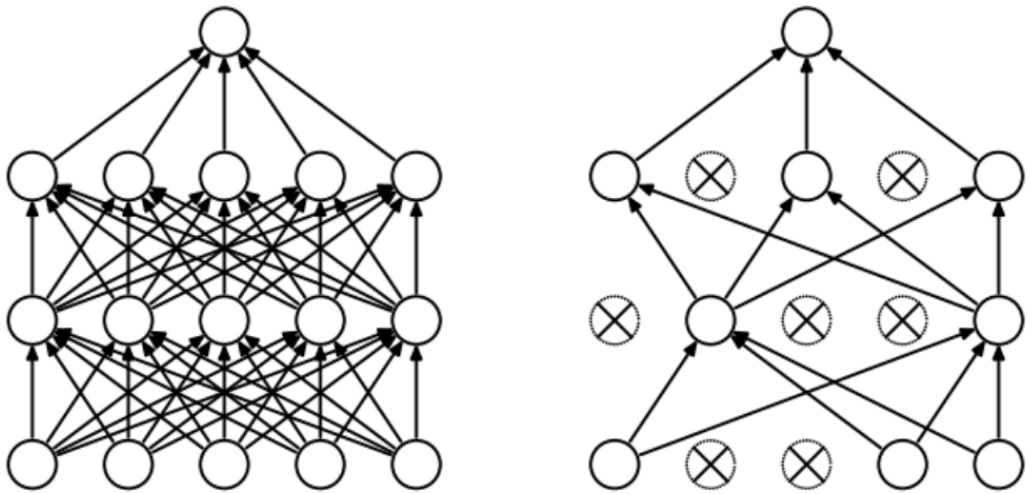


(a) Standard Neural Net

(b) After applying dropout.

Dropout

Another common technique: during forward pass, set some of the weights to 0 randomly with probability p. Typical choice is $p = 50\%$.



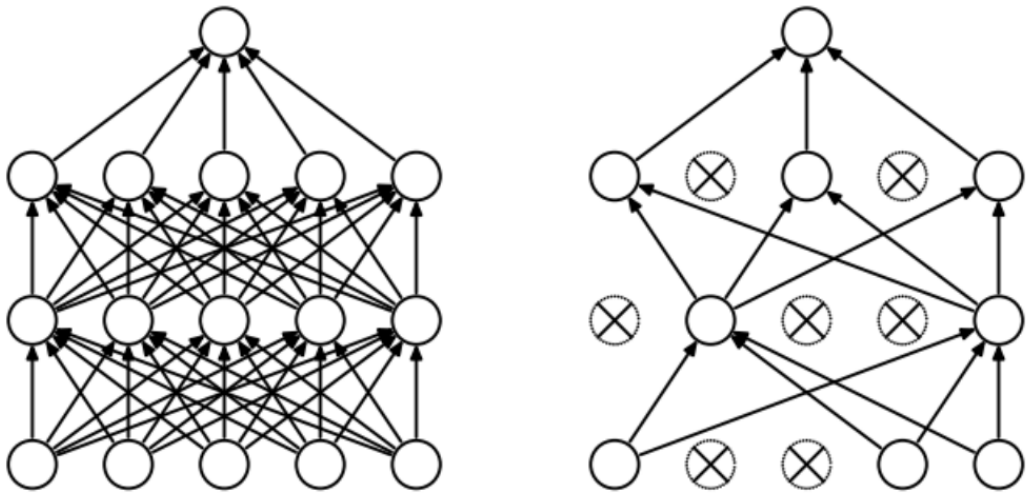
(a) Standard Neural Net

(b) After applying dropout.

- The idea is to prevent co-adaptation of neurons.

Dropout

Another common technique: during forward pass, set some of the weights to 0 randomly with probability p. Typical choice is $p = 50\%$.



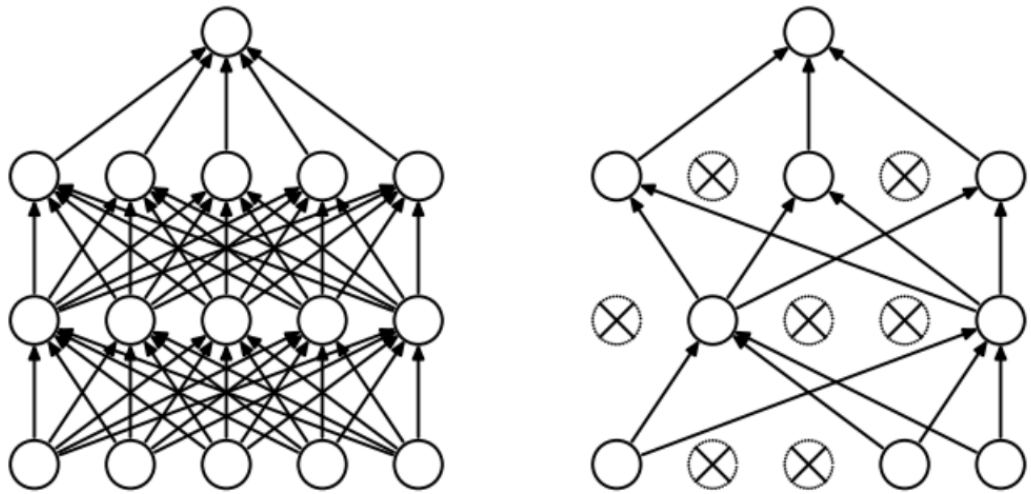
(a) Standard Neural Net

(b) After applying dropout.

- The idea is to prevent co-adaptation of neurons.
- At test want to remove the randomness. A good approximation is to multiply the neural network by p.

Dropout

Another common technique: during forward pass, set some of the weights to 0 randomly with probability p. Typical choice is $p = 50\%$.



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

- The idea is to prevent co-adaptation of neurons.
- At test want to remove the randomness. A good approximation is to multiply the neural network by p.
- Dropout is more commonly applied for fully-connected layers, though its use is waning.

Overview

Initialization & hyper-parameter tuning

Optimization algorithms

Batchnorm & Dropout

Finite dataset woes

Software

Finite dataset woes

While we are entering the Big Data age, in practice we often find ourselves with insufficient data to sufficiently train our deep neural networks.

- What if collecting more data is slow/difficult?

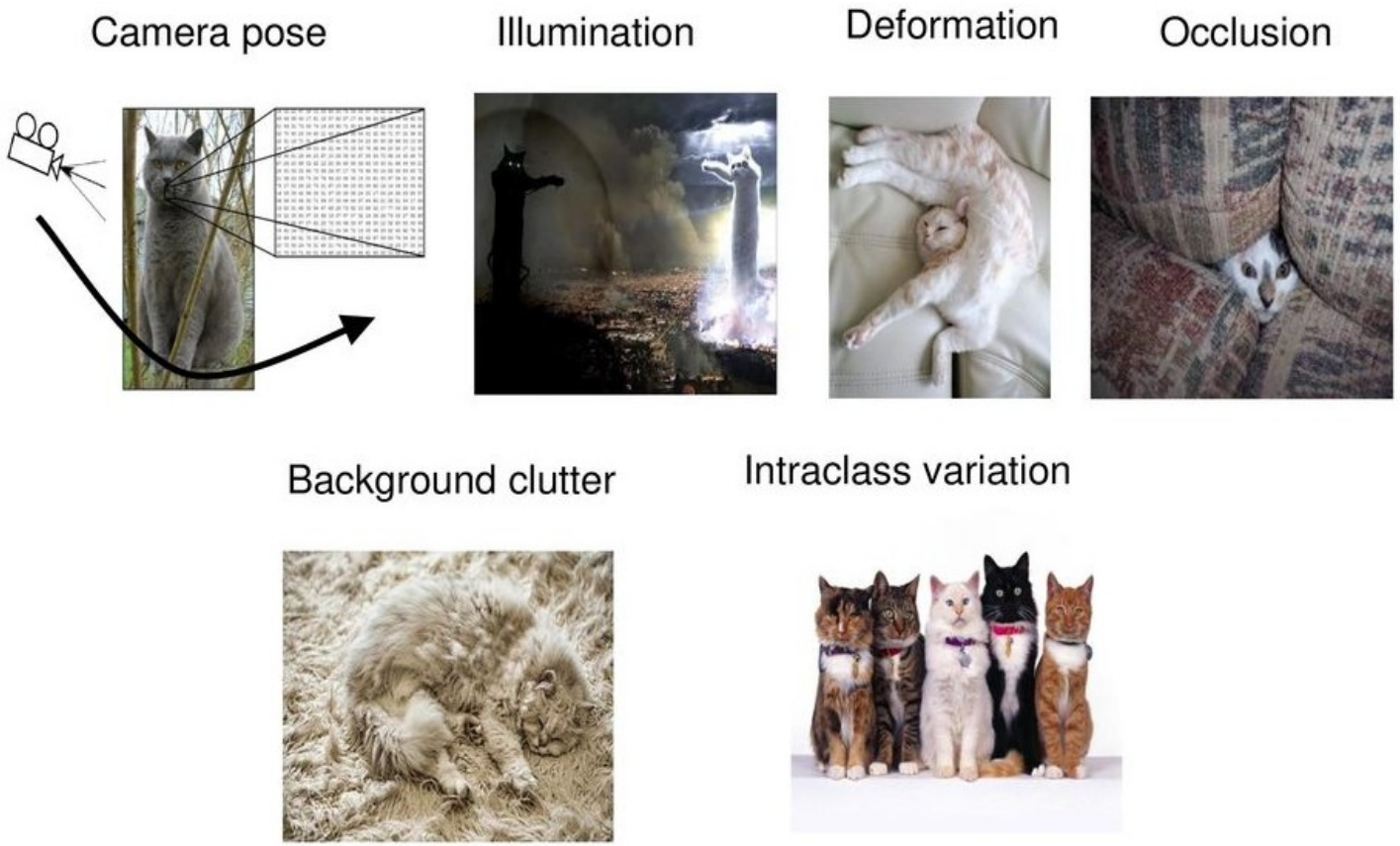
Finite dataset woes

While we are entering the Big Data age, in practice we often find ourselves with insufficient data to sufficiently train our deep neural networks.

- What if collecting more data is slow/difficult?
- Can we squeeze out more from what we already have?

Invariance problem

An often-repeated claim about CNNs is that they are invariant to small translations. Independently of whether this is true, they are not invariant to most other types of transformations:



source: cs213n.github.io

Data augmentation

- Can greatly increase the amount of data by performing:

Data augmentation

- Can greatly increase the amount of data by performing:
  - Translations
  - Rotations
  - Reflections
  - Scaling
  - Cropping
  - Adding Gaussian Noise
  - Adding Occlusion
  - Interpolation
  - etc.

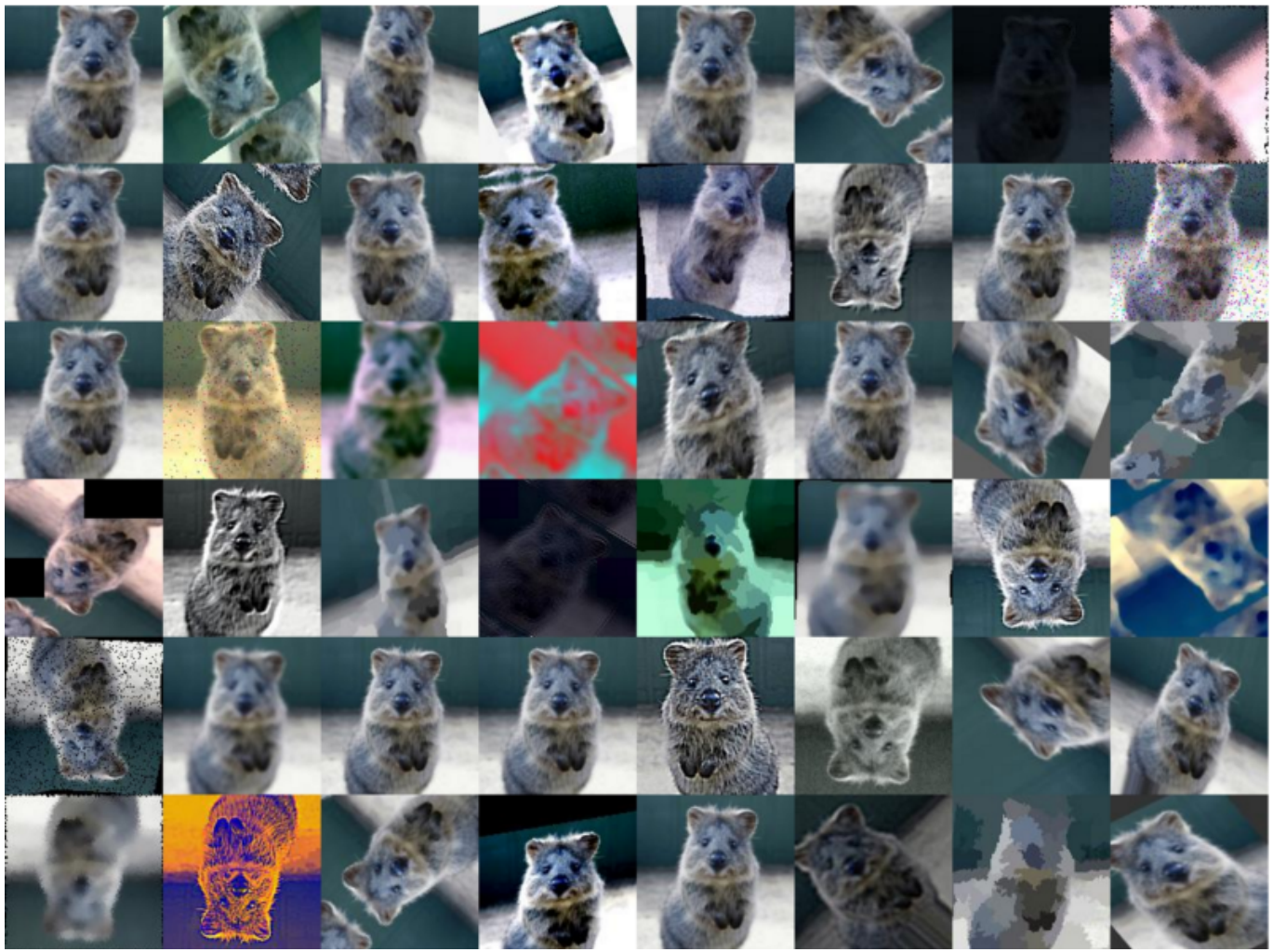
Data augmentation

- Can greatly increase the amount of data by performing:
  - Translations
  - Rotations
  - Reflections
  - Scaling
  - Cropping
  - Adding Gaussian Noise
  - Adding Occlusion
  - Interpolation
  - etc.
- Crucial for achieving state-of-the-art performance!

Data augmentation

- Can greatly increase the amount of data by performing:
  - Translations
  - Rotations
  - Reflections
  - Scaling
  - Cropping
  - Adding Gaussian Noise
  - Adding Occlusion
  - Interpolation
  - etc.
- Crucial for achieving state-of-the-art performance!
- For example, ResNet improves from 11.66% to 6.41% error on CIFAR-10 dataset and from 44.74% to 27.22% on CIFAR-100.

Data augmentation



source: github.com/aleju/imgaug

Transfer Learning

What if you truly have too little data?

- If your data has sufficient similarity to a bigger dataset, the you're in luck!

Transfer Learning

What if you truly have too little data?

- If your data has sufficient similarity to a bigger dataset, the you're in luck!
- Idea: take a model trained for example on ImageNet.

Transfer Learning

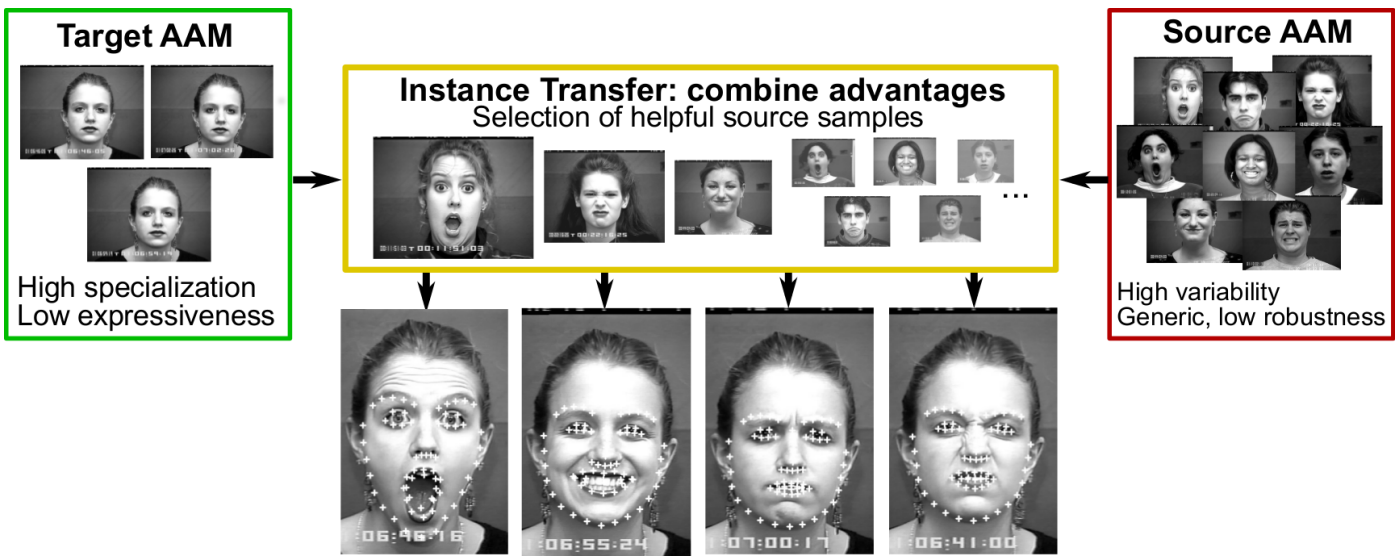
What if you truly have too little data?

- If your data has sufficient similarity to a bigger dataset, the you're in luck!
- Idea: take a model trained for example on ImageNet.
- Freeze all but last few layers and retrain on your small data. The bigger your dataset, the more layers you have to retrain.

Transfer Learning

What if you truly have too little data?

- If your data has sufficient similarity to a bigger dataset, the you're in luck!
- Idea: take a model trained for example on ImageNet.
- Freeze all but last few layers and retrain on your small data. The bigger your dataset, the more layers you have to retrain.



source: [Haase et al., 2014]

Overview

Initialization & hyper-parameter tuning

Optimization algorithms

Batchnorm & Dropout

Finite dataset woes

Software

Software overview

Software overview

Why use frameworks?

- You don't have to implement everything yourself.

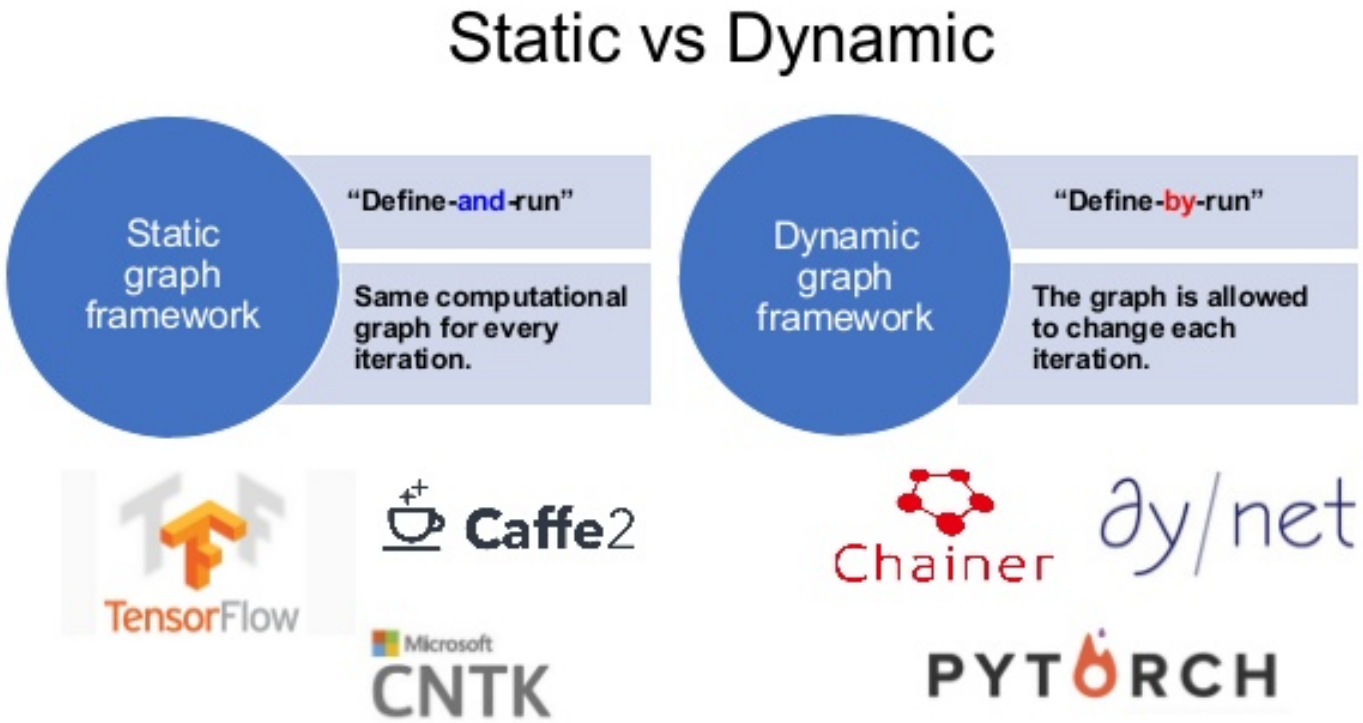
Why use frameworks?

- You don't have to implement everything yourself.
- Many inbuilt modules allow quick iteration of ideas – building a neural network becomes putting simple blocks together and computing backprop is a breeze.

Why use frameworks?

- You don't have to implement everything yourself.
- Many inbuilt modules allow quick iteration of ideas – building a neural network becomes putting simple blocks together and computing backprop is a breeze.
- Someone else already wrote CUDA code to efficiently run training on GPUs (or TPUs).

Main design difference



source: Introduction to Chainer

PyTorch concepts

Similar in code to numpy.

PyTorch concepts

Similar in code to numpy.

- Tensor: nearly identical to np.array, can run on GPU just with

```
device = torch.device(
  "cuda" if use_cuda else "cpu")
```

PyTorch concepts

Similar in code to numpy.

- Tensor: nearly identical to np.array, can run on GPU just with

```
device = torch.device(
  "cuda" if use_cuda else "cpu")
```

- Autograd: package for automatic computation of backprop and construction of computational graphs.

PyTorch concepts

Similar in code to numpy.

- Tensor: nearly identical to np.array, can run on GPU just with

```
device = torch.device(
  "cuda" if use_cuda else "cpu")
```

- Autograd: package for automatic computation of backprop and construction of computational graphs.
- Module: neural network layer storing weights

PyTorch concepts

Similar in code to numpy.

- Tensor: nearly identical to np.array, can run on GPU just with

```
device = torch.device(
  "cuda" if use_cuda else "cpu")
```

- Autograd: package for automatic computation of backprop and construction of computational graphs.
- Module: neural network layer storing weights
- Dataloader: class for simplifying efficient data loading

```
import torch
from torchvision import transforms, datasets

data_transform = transforms.Compose([
        transforms.RandomSizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
hymenoptera_dataset = datasets.ImageFolder(root='hymenoptera_data/train',
                                           transform=data_transform)
dataset_loader = torch.utils.data.DataLoader(hymenoptera_dataset,
                                             batch_size=4, shuffle=True,
                                             num_workers=4)
```

PyTorch - optimization

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

PyTorch - ResNet in one page

```
class BnLayer(nn.Module):
    def __init__(self, ni, nf, stride=2):
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=3, stride=stride, bias=False, padding=1)
        self.a = nn.Parameter(torch.zeros(nf,1,1))
        self.m = nn.Parameter(torch.ones(nf,1,1))

    def forward(self, x):
        x = F.relu(self.conv(x))
        x_chan = x.transpose(0,1).contiguous().view(x.size(1), -1)
        if self.training:
            self.means = x_chan.mean(1)[:,None,None]
            self.stds  = x_chan.std (1)[:,None,None]
        x = x - self.means
        x = x / self.stds
        return x*self.m+self.a
```

```
class ResnetLayer(BnLayer):
    def forward(self, x): return x + super().forward(x)
```

```
class Resnet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
            x = l3(l2(l(x)))
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

@jeremyphoward

A. Banburski

Tensorflow static graphs

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

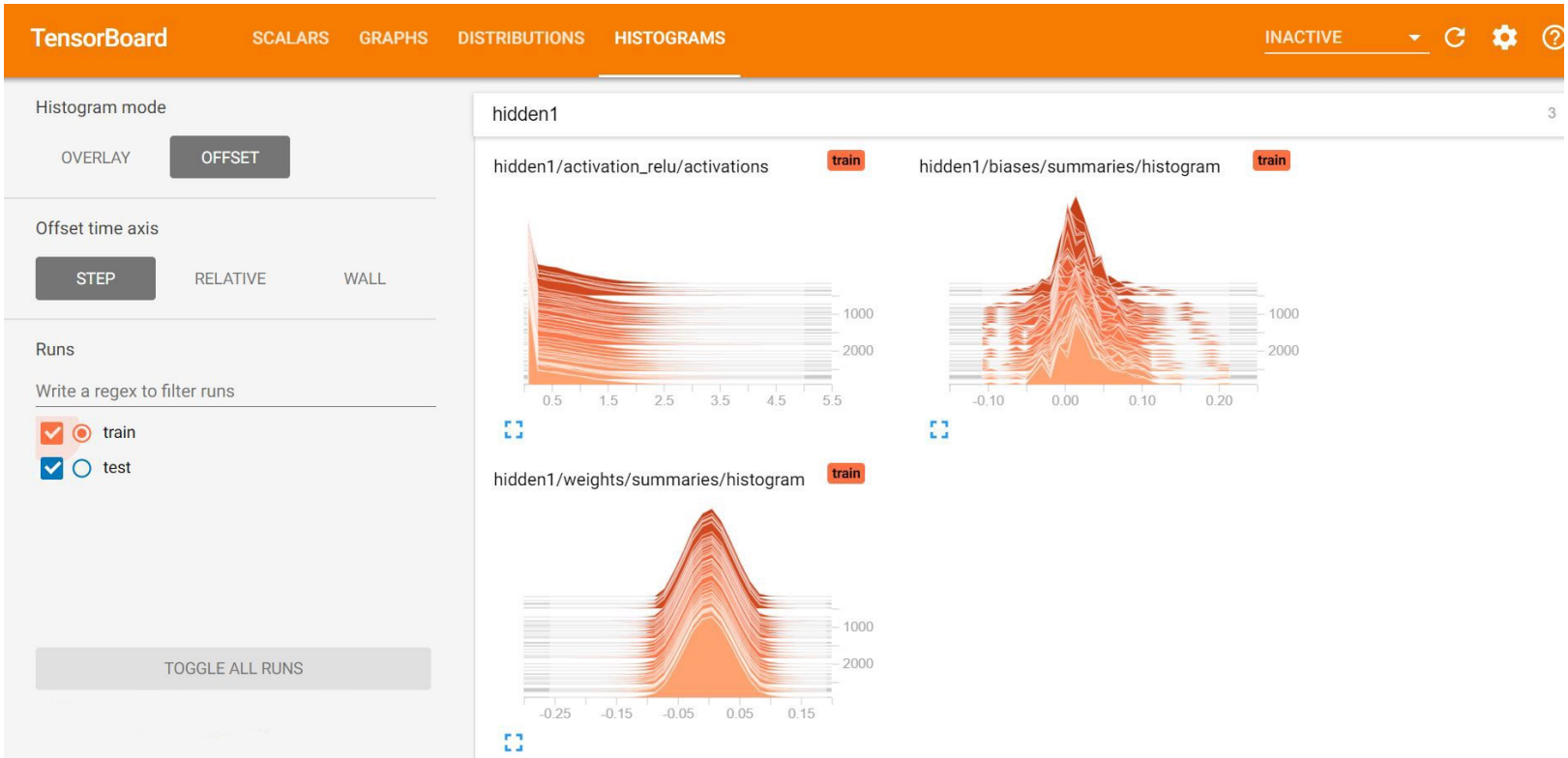
source: cs213n.github.io

Keras wrapper - closer to PyTorch

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
y_pred = model(x)
loss = tf.losses.mean_squared_error(y_pred, y)

optimizer = tf.train.GradientDescentOptimizer(1e0)
updates = optimizer.minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D)}
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

source: cs213n.github.io

Tensorboard - very useful tool for visualization



Tensorflow overview

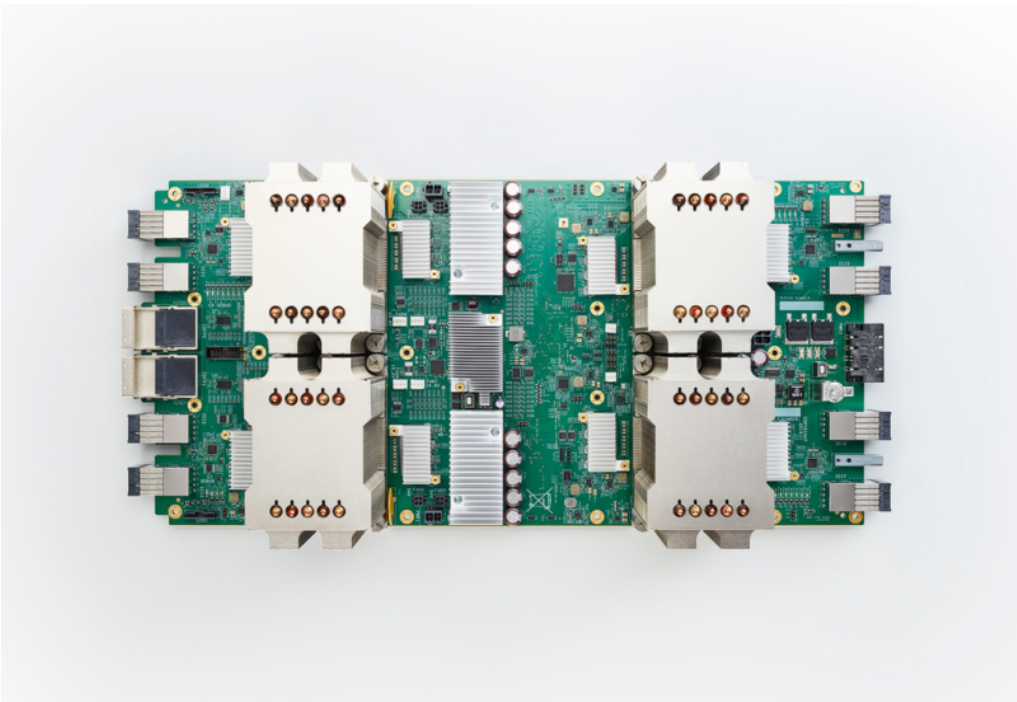
- Main difference – uses static graphs. Longer code, but more optimized. In practice PyTorch is faster to experiment on.

Tensorflow overview

- Main difference – uses static graphs. Longer code, but more optimized. In practice PyTorch is faster to experiment on.
- With Keras wrapper code is more similar to PyTorch however.

Tensorflow overview

- Main difference – uses static graphs. Longer code, but more optimized. In practice PyTorch is faster to experiment on.
- With Keras wrapper code is more similar to PyTorch however.
- Can use TPUs



But

- Tensorflow has added *dynamic batching*, which makes dynamic graphs possible.

But

- Tensorflow has added *dynamic batching*, which makes dynamic graphs possible.
- PyTorch is merging with Caffe2, which will provide static graphs too!

But

- Tensorflow has added *dynamic batching*, which makes dynamic graphs possible.
- PyTorch is merging with Caffe2, which will provide static graphs too!
- Which one to choose then?

But

- Tensorflow has added *dynamic batching*, which makes dynamic graphs possible.
- PyTorch is merging with Caffe2, which will provide static graphs too!
- Which one to choose then?
  - **PyTorch** is more popular in the research community for easy development and debugging.

But

- Tensorflow has added *dynamic batching*, which makes dynamic graphs possible.
- PyTorch is merging with Caffe2, which will provide static graphs too!
- Which one to choose then?
  - **PyTorch** is more popular in the research community for easy development and debugging.
  - In the past a better choice for production was **Tensorflow**. Still the only choice if you want to use TPUs.