Some tools for your deep learning toolbox

Kevin Zhou

14 February 2019

What we’re going to talk about today (in no particular order …)

- Data augmentation
- Normalization
- Dropout
- Minibatch gradient descent
- Momentum

Data augmentation

Data augmentation

- Machine learning is data-driven – the more data, the better!
- Nothing beats collecting more data, but that can be expensive and/or time consuming
- Data augmentation is the next best thing, and it’s free!

Data augmentation one image at a time

Still a cat?

Flip left/right

Random rotation

Still a cat?

Flip up/down

Random affine transformation

Still a cat?

Change color scheme

Add random noise

Data augmentation

- Basic idea: to simulate variation that you might actually see in real life
- It’s a form of regularization
- Not an exact science, but try it out – it’s free!

Normalization

Normalization: data preprocessing

- If you use sigmoid activations, inputs that are too large could saturate them at early layers (vanishing gradient problem)
- Good practice to normalize your inputs
  - e.g. normalize to 0 mean, 1 variance; normalize to between 0 and 1 or -1 and 1
  - $X_i \leftarrow \frac{X_i - \mu}{\sigma}$
- Depending on the dataset, normalization can be done per instance or across entire dataset
  - Datasets with instances that have inconsistent ranges, although theoretically not a problem, in practice could speed up learning

Generalizing normalization to hidden layers

- Batch normalization
- Layer normalization
- Instance normalization
- Group normalization

- All of these normalize hidden layers to 0 mean and 1 variance, but these means and variances are computed across different dimensions
  - $X_i \leftarrow \frac{X_i - \mu}{\sigma}$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

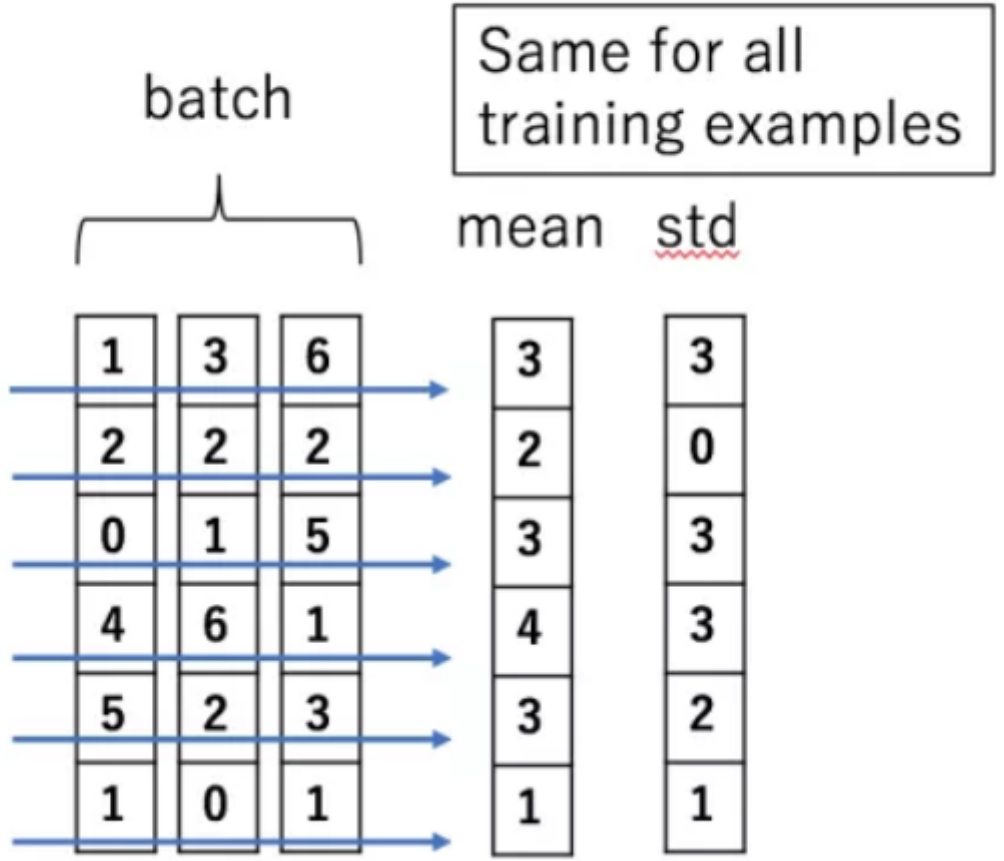
Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Cited over 8,500 times!
(as of Feb 2019)

Batch normalization (BN)

- Before BN, training very deep networks was hard
  - If using sigmoid activations, large weights could result in saturation
  - Updating earlier layers' weights causes the distribution of weights in later layers to shift – the *internal covariate shift*
- To address this covariate shift, BN "resets" the layer it is applied to by normalizing to 0 mean, 1 variance
  - Mean and variance are computed over the batch at the current iteration



Problems

- Normalizing to 0 mean 1 variance reduces the expressivity of the layer
  - E.g., if using a sigmoid activation, you're stuck in the linear regime
- Solution: reintroduce mean (β) and standard deviation (γ) parameters:
  - $X_i \leftarrow \frac{X_i - \mu}{\sigma}$ #normalize
  - $X_i \leftarrow \gamma X_i + \beta$ #new mean and standard deviations
  - γ and β are trainable parameters
- Accuracy of μ and σ depends on the batch size being large

Batch normalization: training vs testing

- Training:
  - Keep track of running averages for μ and σ to be used during testing
  - TensorFlow code: `after_BN = tf.layers.batch_normalization(before_BN, training=True)`
  - You also need to include the updates for μ and σ in the train operation (more on this in a later TA session)
- Testing:
  - Use the running averages for μ and σ from training – this allows you to run one input through the network (i.e., you wouldn't otherwise be able to compute a mean and variance)
  - TensorFlow code: `after_BN = tf.layers.batch_normalization(before_BN, training=False)`
- Useful to create a tf placeholder to indicate whether you're training (more on this in a later TA session)

Quick note on batch normalization for CNNs

- Compute mean and variance across the spatial dimensions too within a color channel/feature dimension

Other hidden layer normalizations (for CNNs)

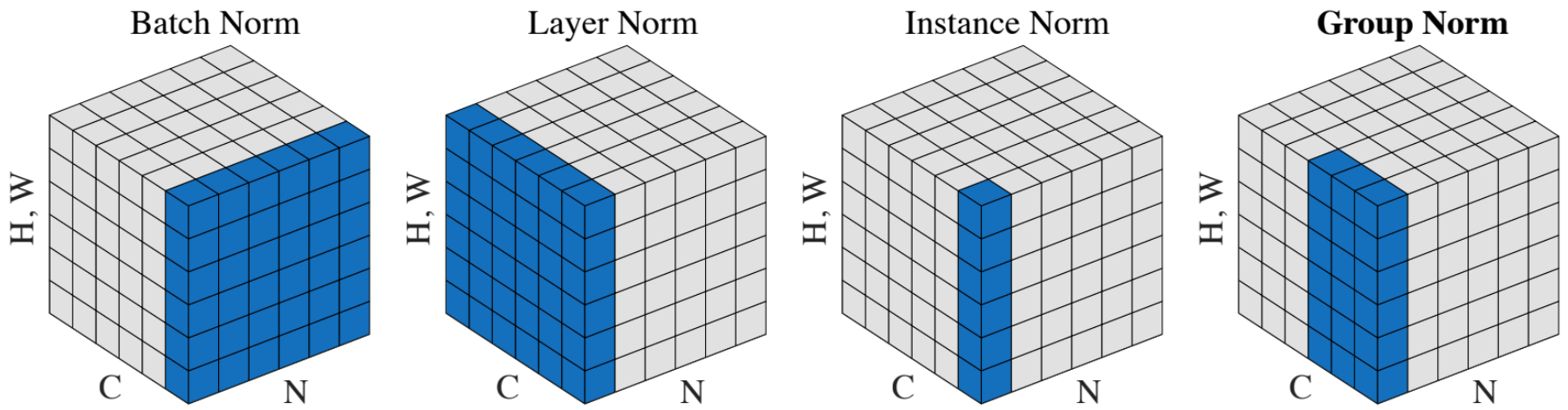


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

https://nealjean.com/ml/neural-network-normalization/

Dropout

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

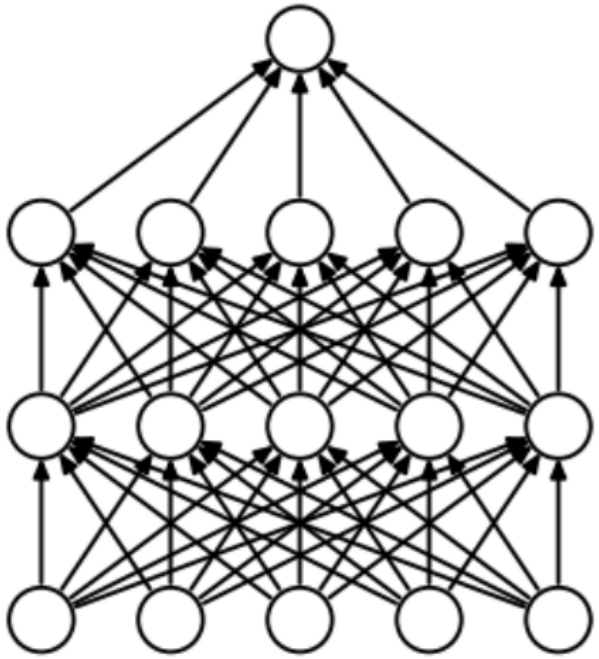
Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

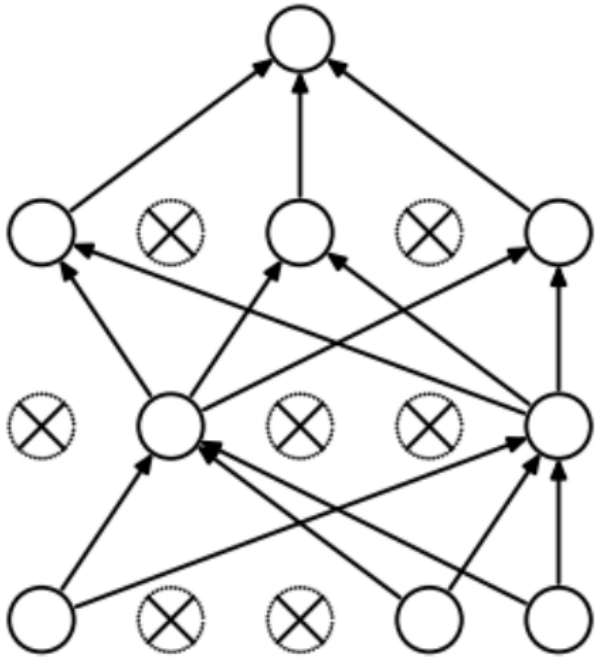
Cited over 10,000 times!
(as of Feb 2019)

Dropout

- At each train iteration, randomly delete a fraction p of the nodes
- Prevents neurons from being lazy
- A form of model averaging
- (related: DropConnect – drop the connections instead of nodes)



(a) Standard Neural Net



(b) After applying dropout.

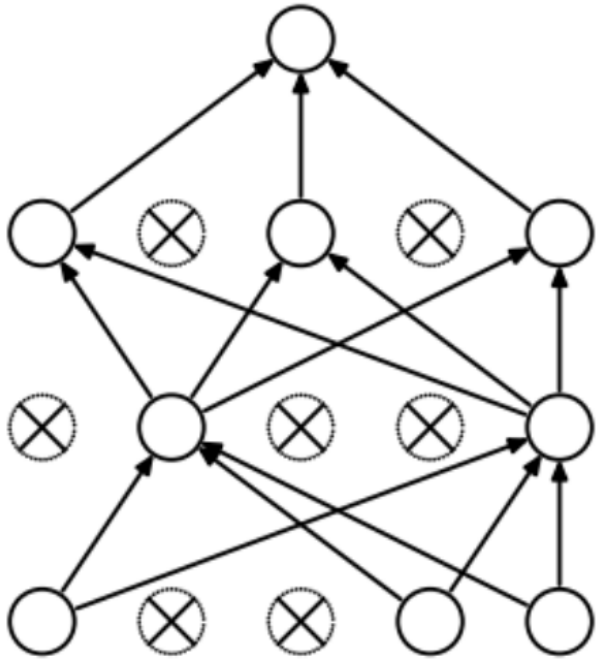
https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Dropout

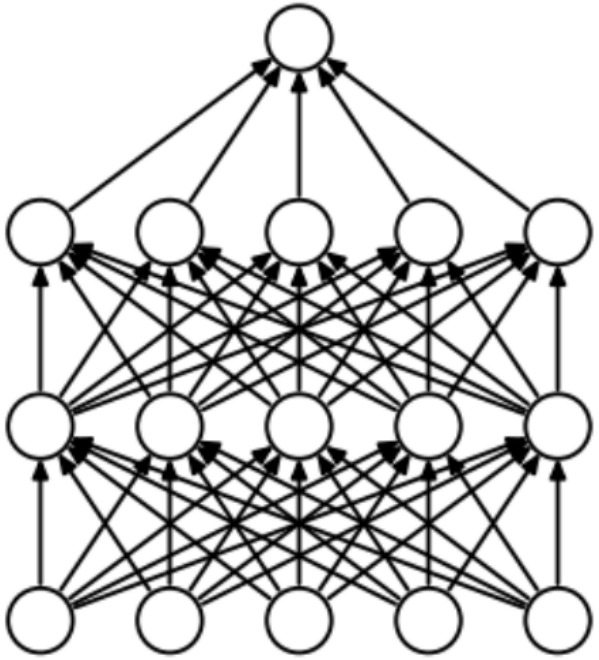
- Only one hyperparameter p, the expected fraction of neurons to drop in a given layer
- In TensorFlow:
  - `next_layer = tf.layers.dropout(previous_layer, rate=0.5)`
- Common practices:
  - Set p=0.5
  - Make the layer wider (more units/neurons)
  - Apply to fully connected layers, not convolutional layers (already sparse)

Dropout training vs testing

- Training: at a given layer, each node is dropped with probability p
- Testing: multiply the outgoing weights by 1-p (*weight scaling inference rule*)
- As a model averaging technique, other possibilities exist



Training
(each node dropped with probability)



Testing
(all weights multiplied by 1-p)

Minibatch stochastic gradient descent

Minibatch stochastic gradient descent (SGD)

- Everything we compute from data is only an estimate:
  - $X, y \sim P_{data}$: data comes from a distribution
  - $L_i = neural_{\downarrow} net(X_i, y_i; W)$: for each data instance, compute the loss (W is the parameters of the netowrk)
  - $Loss = E_{P_{data}}[L]$: this is the loss, but we don't have access to it – we can only estimate it!
  - $Loss \approx \frac{1}{M}\sum_{i=1}^{M} L_i$: this is the estimate, based on M instances
- Hence, *Empirical Risk Minimization*
- Loss is a linear combination, so the gradient is also a linear combination:
  - $\mathrm{grad}_W(Loss) \approx \frac{1}{M}\sum_{i=1}^{M} \mathrm{grad}_W(L_i)$

Minibatch stochastic gradient descent (SGD)

- However, if our estimate of the gradient of the loss is perfect, we are sure to run into local minima early on
- Solution: add noise to the gradient to encourage exploration!
- We get this for free by decreasing the batch size
  - Uncertainty in gradient estimate ~ 1/sqrt(batch size)

- Be sure to make sure your batches are random

SGD with momentum

Standard momentum

- In cases where the gradient is noisy or very small, SGD could use some encouragement
- Idea: let your gradient step be an accumulation of previous gradients:

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

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

  - The relative sizes of α and ε determines how much the current gradient should matter relative to the history
  - For α = 0, this is regular SGD

Nesterov momentum

- Same idea, except evaluate the gradient after first taking a “peek” into the future based on the previously accumulated momentum

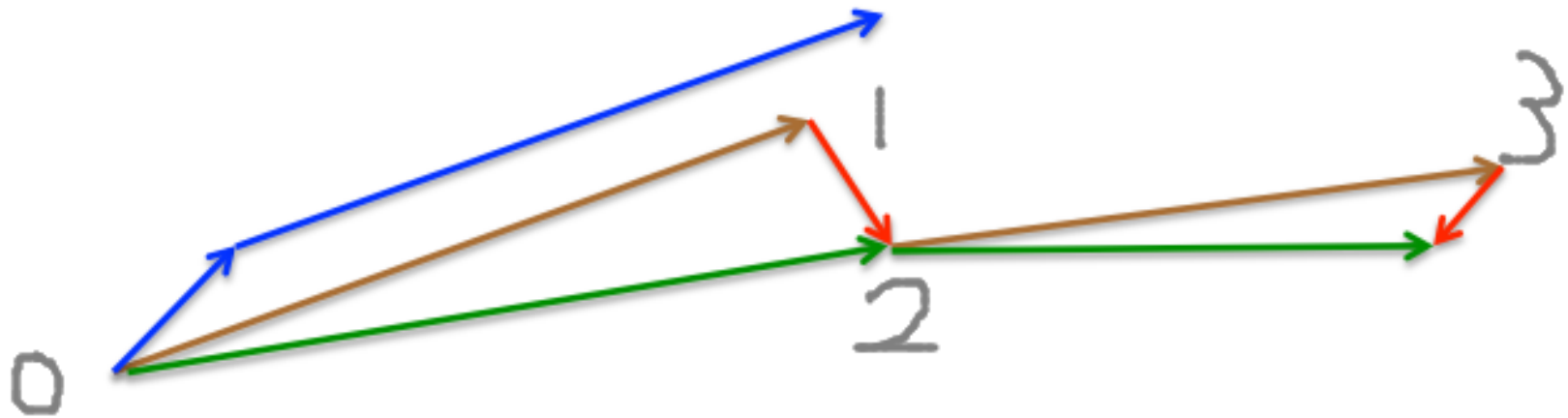
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right]$$

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

- If after peeking you realize that the momentum would cause you to overshoot, use the gradient there to correct your overshoot

A picture of the Nesterov method

- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

https://i.stack.imgur.com/wBIpz.png