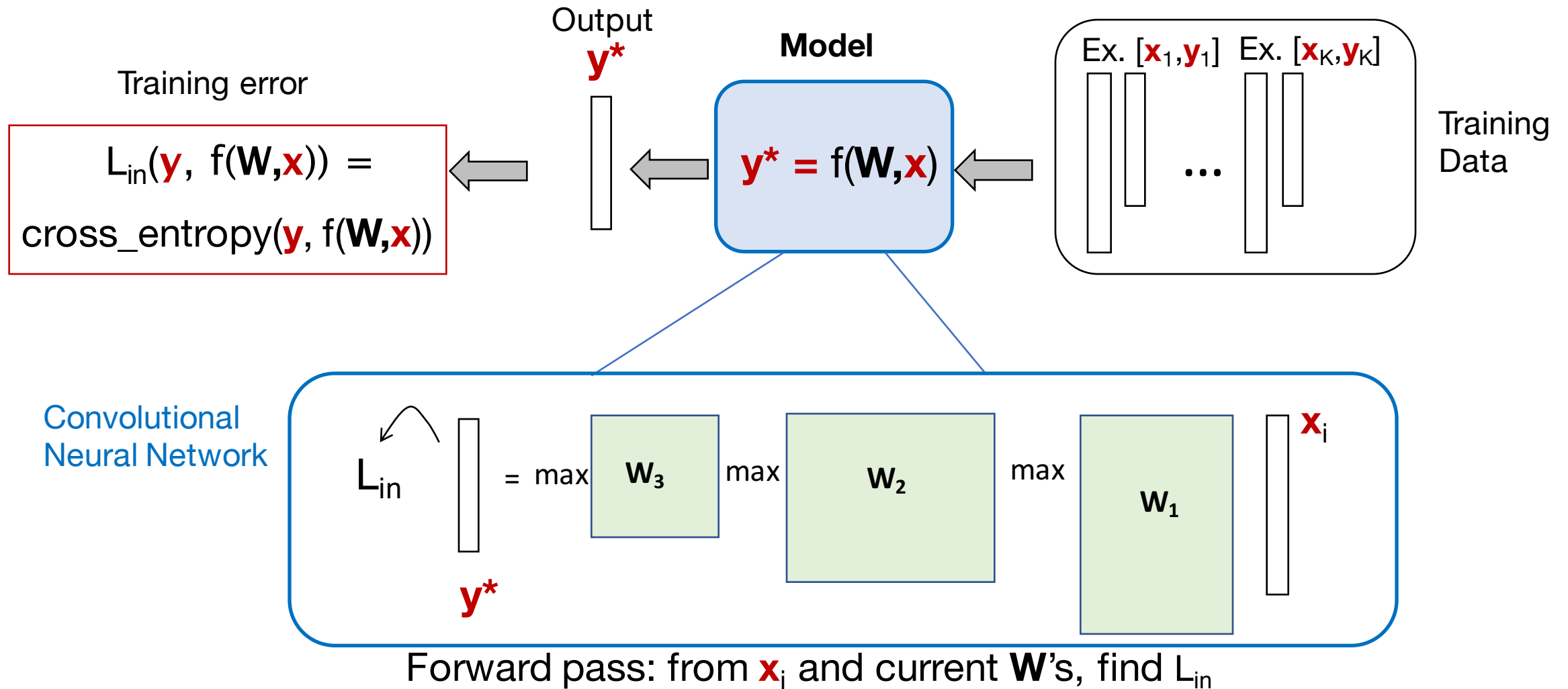# Lecture 11: Tools for your deep learning toolbox

Machine Learning and Imaging
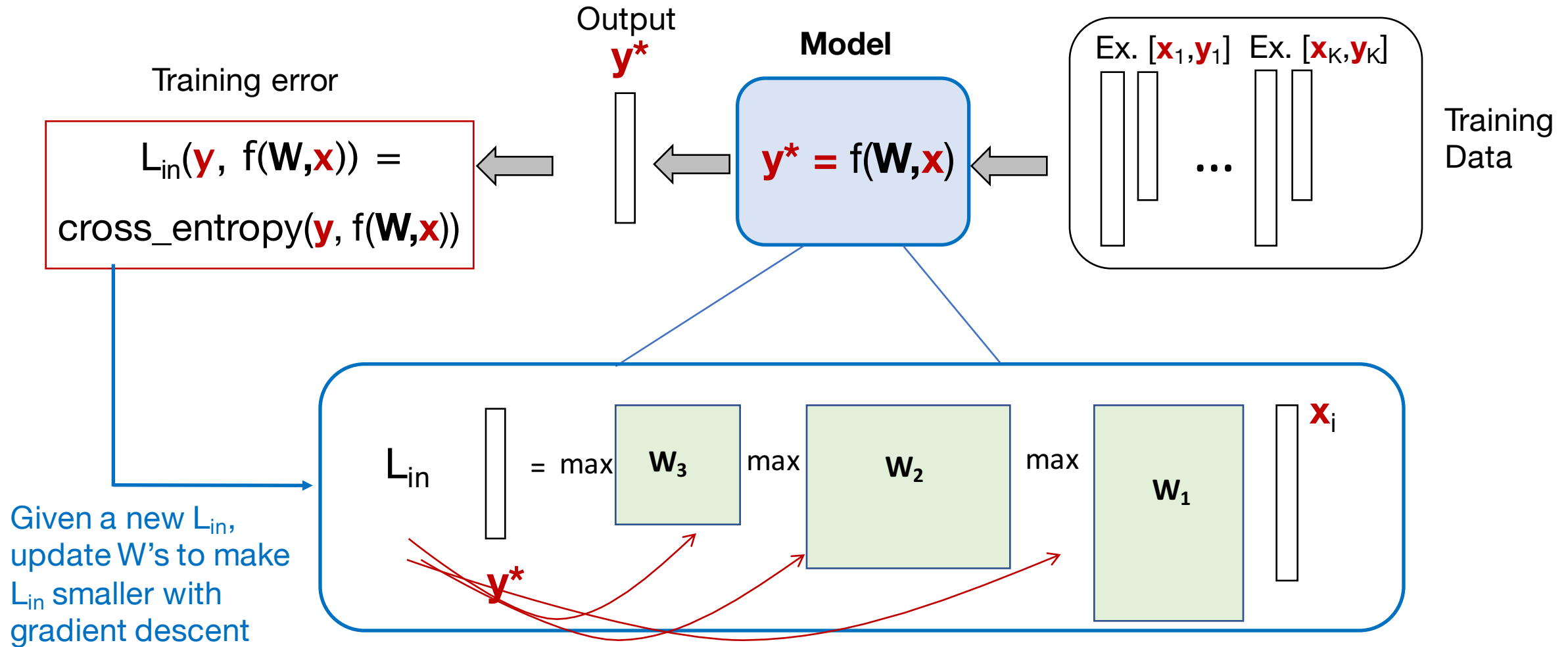
BME 590L
Roarke Horstmeyer

Thanks to Kevin Zhou for preparing much of this material

Our very basic convolutional neural network

deep imaging

Training error

Output
$\mathbf{y^*}$

Model

Ex. [$\mathbf{x_1},\mathbf{y_1}$] Ex. [$\mathbf{x_K},\mathbf{y_K}$]

$$L_{in}(\mathbf{y}, f(\mathbf{W,x})) =$$

$$cross\_entropy(\mathbf{y}, f(\mathbf{W,x}))$$

$\mathbf{y^*} = f(\mathbf{W,x})$

Training
Data

...

Convolutional
Neural Network

L_{in} = max | $\mathbf{W_3}$ | max | $\mathbf{W_2}$ | max | $\mathbf{W_1}$ | $\mathbf{x}_i$

$\mathbf{y^*}$

Forward pass: from $\mathbf{x}_i$ and current $\mathbf{W}$'s, find L_{in}

Our very basic convolutional neural network

Training error

Output
y*

Model

$$L_{in}(\mathbf{y},\ f(\mathbf{W},\mathbf{x})) =$$

$$\text{cross\_entropy}(\mathbf{y}, f(\mathbf{W},\mathbf{x}))$$

$$\mathbf{y^*} = f(\mathbf{W},\mathbf{x})$$

Ex. $[\mathbf{x}_1, \mathbf{y}_1]$ Ex. $[\mathbf{x}_K, \mathbf{y}_K]$

...

Training Data

Given a new L_{in}, update W's to make L_{in} smaller with gradient descent

$$L_{in}\ \Big|\ = \max\ \mathbf{W_3}\ \max\ \mathbf{W_2}\ \max\ \mathbf{W_1}\ \Big|\ \mathbf{x}_i$$

y*

Last Class: Effectively achieve this with automatic differentiation (backprop)

deep imaging

Important components of a CNN

CNN Architecture

Architecture choices

- CONV size, stride, pad, depth

- ReLU & other nonlinearities

- POOL methods

- # of layers, dimensions per layer

- Fully connected layers

Loss function & optimization

Optimization choices

- Type of loss function

- Regularization

- Gradient descent method

- SGD batch and step size

Other specifics: Variable Initialization**,** augmentation, batch normalization, dropout, gradient descent params.

Important components of a CNN

CNN Architecture

Architecture choices

- CONV size, stride, pad, depth

- ReLU & other nonlinearities

- POOL methods

- # of layers, dimensions per layer

- Fully connected layers

Loss function & optimization

- Type of loss function

- Regularization

- Gradient descent method

- SGD batch and step size

Optimization choices

Other specifics: Variable Initialization**,** augmentation, batch normalization, dropout, gradient descent params.

This class: final details about deep CNN implementation

Weights initialization

- Need to start somewhere – typically best to use an appropriate random guess

Convex problem: doesn't really matter where you start
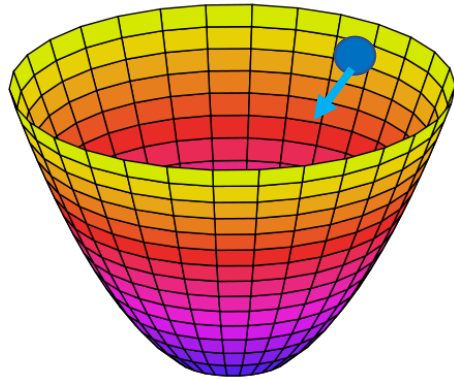
Non-convex problem: certainly matters, but you don't know where is best...
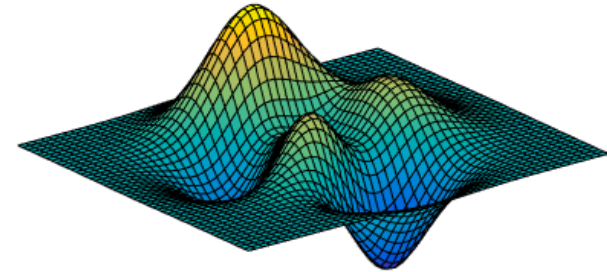
- Need to start somewhere – typically best to use an appropriate random guess sampled from a Gaussian distribution:

layer1_weight = tf.Variable(tf.truncated_normal([5,5, 1, 32], stddev = 0.1)

Weights initialization

- Often it is helpful to take variance of weights into account
  - Having very large and very small weights leads to instabilities

- Desire: variance of inputs (x) remain unchanged as they transfer through network

$$y = w^T x$$

$$\text{var}(y) = \text{var}(w^T x) = \text{var}(w_1 x_1 + \ldots w_N x_N) = N\,\text{var}(w_1 x_1) \quad \text{(IID)}$$

$$\text{var}(wx) = E(w)^2 \text{var}(x) + E(x)^2 \text{var}(w) + \text{var}(w)\text{var}(x) = \text{var}(w)\text{var}(x)$$

$$\text{var}(y) = N\,\text{var}(w)\text{var}(x)$$

$$\text{var}(y) = \text{var}(x) \text{ when } \text{var}(w) = 1/N$$

layer1_weight = tf.Variable(tf.truncated_normal([5,5, 1, 32], stddev = 1/N)   Xavier Initialization

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!

deep imaging

Data augmentation one image at a time

Still a cat?

Flip left/right

Random rotation

deep imaging

Still a cat?

Flip up/down

Random affine transformation

deep imaging

Still a cat?

Change color scheme

Add random noise

deep imaging

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: 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*

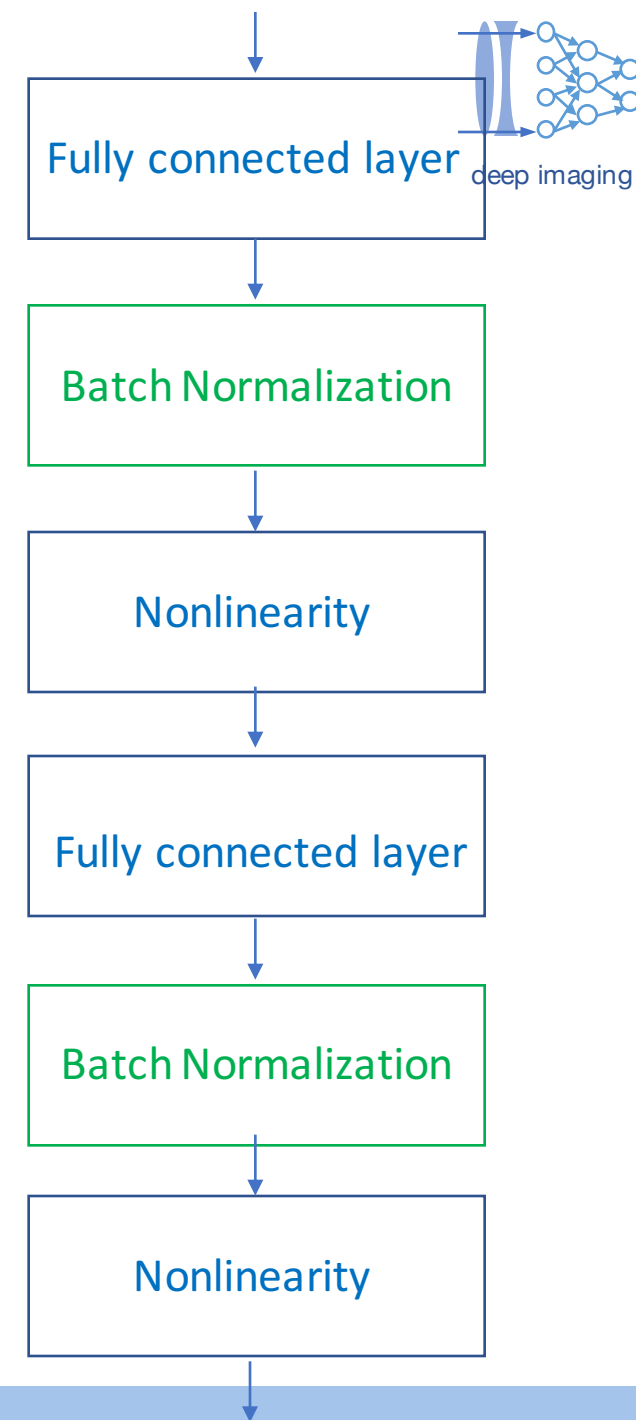Cited ~12,500 times! (as of Sept 2019)

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

Batch normalization update for inputs x:

$$x'(i) = (x(i) - E[x(i)]) / STD[x(i)]$$

- Mean subtract
- Normalize by standard deviation

Fully connected layer

deep imaging

Batch Normalization

Nonlinearity

Fully connected layer

Batch Normalization

Nonlinearity

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
  - $\underline{\gamma \text{ and } \beta \text{ are trainable parameters}}$

- Accuracy of μ and σ depends on the batch size being large

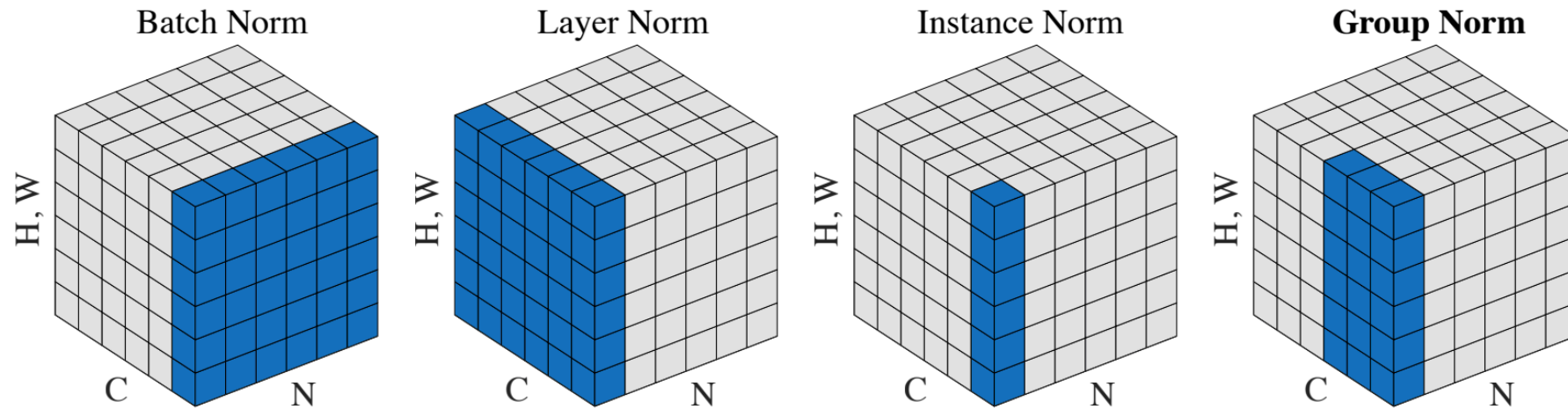# Other hidden layer normalizations (for CNNs)

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/

aging

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava   NITISH@CS.TORONTO.EDU
Geoffrey Hinton   HINTON@CS.TEORONTO.EDU
Alex Krizhevsky   KRIZ@CS.TORONKTO.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
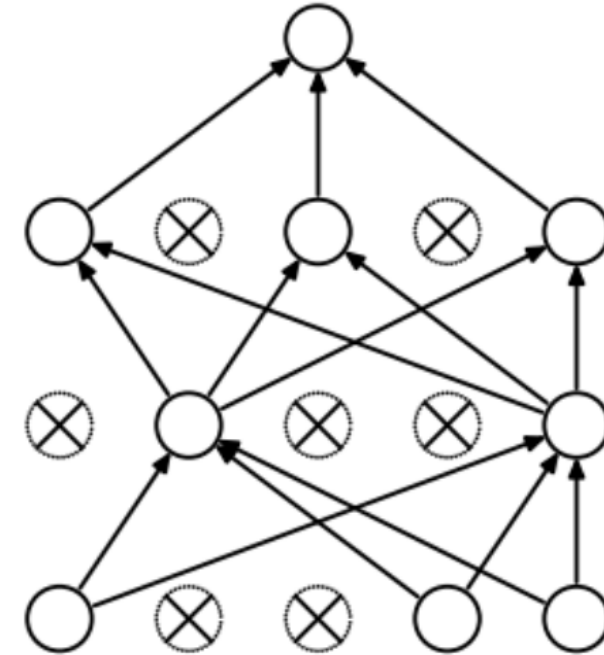*Toronto, Ontario, M5S 3G4, Canada.*

Cited over 14,000 times!
(as of Sept. 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)
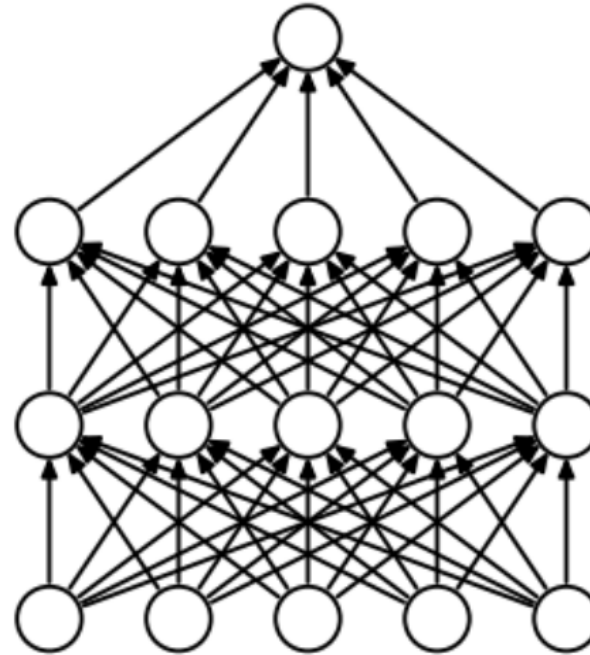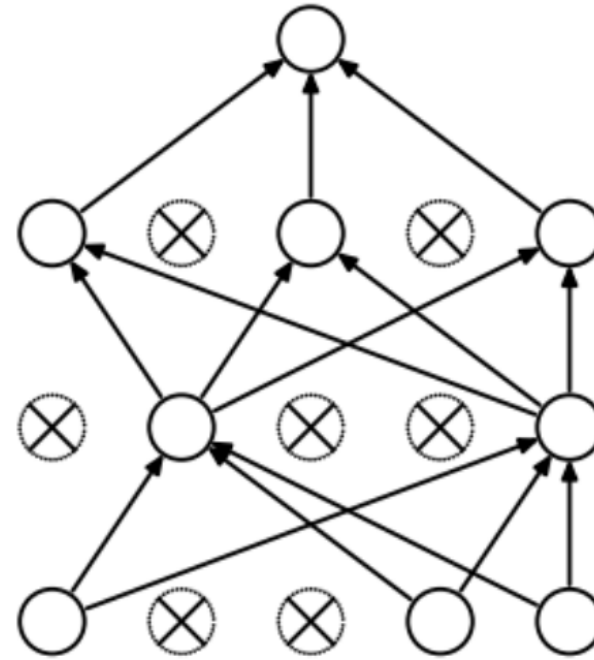

(a) Standard Neural Net

(b) After applying dropout.

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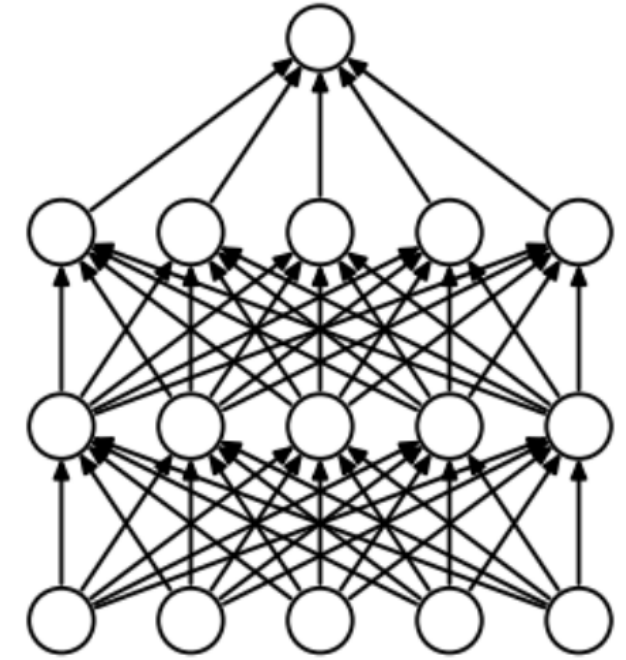
Dropout

- Only one hyperparameter "rate" = 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*)

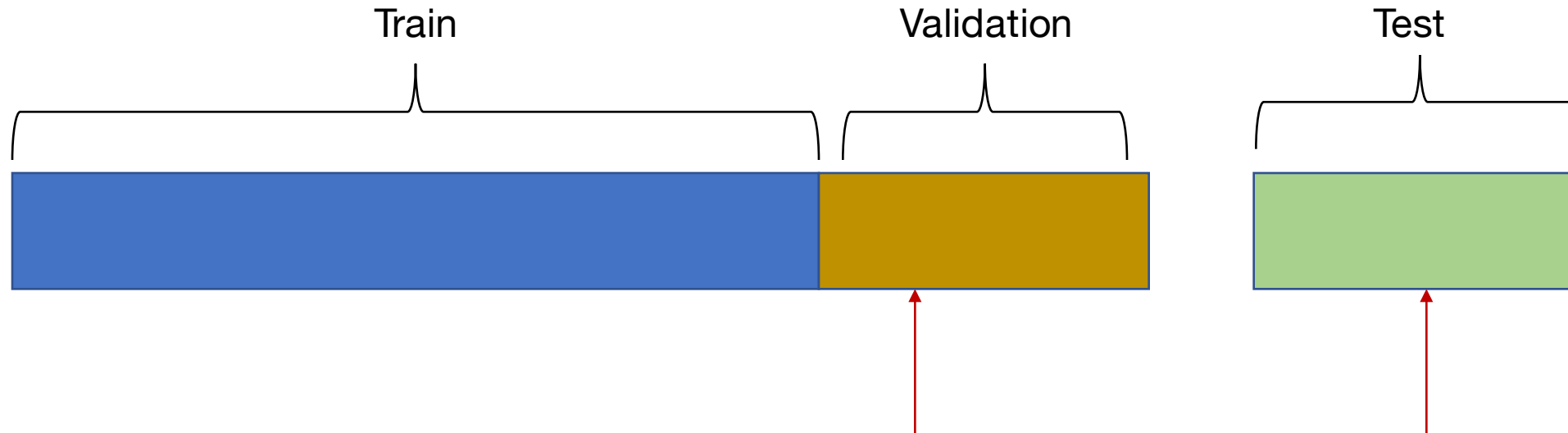- As a model averaging technique, other possibilities exist

Training
(each node dropped with probability)

Testing
(all weights multiplied by 1-p)

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

Training dataset, test dataset and validation dataset

Train

Validation

Test

Use to evaluate while tuning hyperparameters
• effect will creep into model as you continue to use it

Final test set is always separate!
Don't touch until the end!

Let's examine our first CNN implemented in Tensorflow!

http://deepimaging.github.io/data/basic_tensorflow_eager_example.ipynb