Training Neural Networks: Normalization, Regularization etc.

Intro to Deep Learning, Spring 2021
Recap

• We train a network by minimizing a “loss”

\[ L(W) = \frac{1}{N_X} \sum_X \text{div}(f(X; W), D(X)) \]

  – Average divergence between true and desired outputs over “training” inputs
  – Approximation to “true” risk – expected divergence between desired and true outputs

• We minimize it through gradient descent
  – Iterative updates against the gradient of the loss w.r.t. \( W \)

• Batch updates must process the entire training data before each update
  – Incremental update algorithms, like SGD and minibatch update, speed it up by updating using random individual inputs or subsets of the input
  – Faster to converge, but greater variance may result in worse estimates

• Trend algorithms smooth out the variations in incremental update methods by considering long-term trends in gradients.
  – This can lead to faster, and better convergence
Quick Recap: Training a network

- Define a total “loss” over all training instances
  - Quantifies the difference between desired output and the actual output, as a function of weights
- Find the weights that minimize the loss

\[
L(W) = \frac{1}{N_X} \sum_X \text{div}(f(X; W), D(X))
\]

\[
\hat{W} = \arg \min_W L(W)
\]
Quick Recap: Training networks by gradient descent

\[ L(W) = \frac{1}{N_X} \sum_X \text{div}(f(X; W), D(X)) \]

\[ \nabla_W L(W) = \frac{1}{N_X} \sum_X \nabla_W \text{div}(f(X; W), D(X)) \]

Solved through gradient descent as

\[ \hat{W} = \arg \min_W L(W) \]

\[ W_k = W_{k-1} - \eta \nabla_W L(W)^T \]

Computed using backpropagation
Recap: Incremental methods

- Batch methods that consider all training points before making an update to the parameters can be terribly inefficient.

- Online methods that present training instances incrementally make quicker updates:
  - “Stochastic Gradient Descent” updates parameters after individual randomly-chosen instances.
  - “Mini batch descent” updates them after minibatches of randomly-chosen instances.
  - Require shrinking learning rates to converge:
    - Not absolute summable
    - But square summable

- Online methods have greater variance than batch methods:
  - Potentially leading to worse model estimates.
Recap: Trend Algorithms

• Trend algorithms smooth out the variations in incremental update methods by considering long-term trends in gradients
  – Leading to faster and more assured convergence

• Momentum and Nestorov’s method improve convergence by smoothing updates with the mean (first moment) of the sequence of derivatives

• Second-moment methods consider the variation (second moment) of the derivatives
  – RMS Prop only considers the second moment of the derivatives
  – ADAM and its siblings consider both the first and second moments
  – All of them typically provide considerably faster than simple gradient descent
Moving on: Topics for the day

• Incremental updates
• Revisiting “trend” algorithms
• Generalization
• Tricks of the trade
  – Divergences..
  – Activations
  – Normalizations
Tricks of the trade..

• To make the network converge better
  – The Divergence
  – Batch normalization
  – Dropout
  – Other tricks
    • Gradient clipping
    • Data augmentation
    • Other hacks..
Training Neural Nets by Gradient Descent: The Divergence

Total training loss:

\[ \text{Loss} = \frac{1}{T} \sum_{t} \text{Div}(Y_t, d_t; W_1, W_2, \ldots, W_K) \]

- The convergence of the gradient descent depends on the divergence
  - Ideally, must have a shape that results in a significant gradient in the right direction outside the optimum
    - To “guide” the algorithm to the right solution
Desiderata for a good divergence

• Must be smooth and not have many poor local optima
• Low slopes far from the optimum == bad
  – Initial estimates far from the optimum will take forever to converge
• High slopes near the optimum == bad
  – Steep gradients
Desiderata for a good divergence

- Functions that are shallow far from the optimum will result in very small steps during optimization
  - Slow convergence of gradient descent
- Functions that are steep near the optimum will result in large steps and overshoot during optimization
  - Gradient descent will not converge easily
- The best type of divergence is steep far from the optimum, but shallow at the optimum
  - But not too shallow: ideally quadratic in nature
## Choices for divergence

### L2

\[ \text{Div} = \frac{1}{2} (y - d)^2 \]

### KL

\[ \text{Div} = -d \log(y) - (1 - d) \log(1 - y) \]

### Desirable output:

- **L2**: \([0,0, ..., 1, ..., 0]\)
- **KL**: \(d\)

- **Most common choices**: The L2 divergence and the KL divergence
- **L2** is popular for networks that perform numeric prediction/regression
- **KL** is popular for networks that perform classification
L2 or KL?

• The L2 divergence has long been favored in most applications

• It is particularly appropriate when attempting to perform *regression*
  – Numeric prediction

• The KL divergence is better when the intent is classification
  – The output is a probability vector
• Plot of L2 and KL divergences for a *single* perceptron, as function of weights
  – Setup: 2-dimensional input
  – 100 training examples randomly generated
L2 or KL

NOTE: L2 divergence is not convex while KL is convex

However, L2 also has a unique global minimum

• Plot of L2 and KL divergences for a *single* perceptron, as function of weights
  – Setup: 2-dimensional input
  – 100 training examples randomly generated
A note on derivatives

• Note: For L2 divergence the derivative w.r.t. the output of the network is:

\[ \nabla_y \frac{1}{2} \|y - d\|^2 = (y - d) \]

• We literally “propagate” the error \((y - d)\) backward
  – Which is why the method is sometimes called “error backpropagation”
Story so far

• Gradient descent can be sped up by incremental updates

• Convergence can be improved using smoothed updates

• The choice of divergence affects both the learned network and results
The problem of covariate shifts

• Training assumes the training data are all similarly distributed
  – Minibatches have similar distribution
The problem of covariate shifts

- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A “covariate shift”
  - Which may occur in each layer of the network
The problem of covariate shifts

- Training assumes the training data are all similarly distributed
  - Minibatches have similar distribution
- In practice, each minibatch may have a different distribution
  - A “covariate shift”
- Covariate shifts can be large!
  - All covariate shifts can affect training badly
Solution: Move all minibatches to a “standard” location

• “Move” all batches to a “standard” location of the space
  – But where?
  – To determine, we will follow a two-step process
Move all minibatches to a “standard” location

• “Move” all batches to have a mean of 0 and unit standard deviation  
  – Eliminates covariate shift between batches
Move all minibatches to a “standard” location

• “Move” all batches to have a mean of 0 and unit standard deviation
  – Eliminates covariate shift between batches
Move all minibatches to a “standard” location

• “Move” all batches to have a mean of 0 and unit standard deviation
  – Eliminates covariate shift between batches
Move all minibatches to a “standard” location

- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
Move all minibatches to a “standard” location

- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
(Mini)Batch Normalization

• “Move” all batches to have a mean of 0 and unit standard deviation
  – Eliminates covariate shift between batches

• Then move the entire collection to the appropriate location
Batch normalization is a covariate adjustment unit that happens after the weighted addition of inputs but before the application of activation. Is done independently for each unit, to simplify computation.

- Training: The adjustment occurs over individual minibatches.
Batch normalization

• BN aggregates the statistics over a minibatch and normalizes the batch by them
• Normalized instances are “shifted” to a unit-specific location

\[ z = \sum_{j} w_j i_j + b \]

\[ u_i = \frac{z_i - \mu_B}{\sigma_B} \]

\[ \hat{z}_i = \gamma u_i + \beta \]
Batch normalization: Training

- BN aggregates the statistics over a minibatch and normalizes the batch by them.
- Normalized instances are “shifted” to a *unit-specific* location.
Batch normalization: Training

- BN aggregates the statistics over a minibatch and normalizes the batch by them.
- Normalized instances are “shifted” to a *unit-specific* location.

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \\
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

\[
\hat{z}_i = \gamma u_i + \beta
\]
A better picture for batch norm
A note on derivatives

• The minibatch loss is the average of the divergence between the actual and desired outputs of the network for all inputs in the minibatch

\[
\text{Loss(minibatch)} = \frac{1}{B} \sum_t \text{Div}(Y_t(X_t), d_t(X_t))
\]

• The derivative of the minibatch loss w.r.t. network parameters is the average of the derivatives of the divergences for the individual training instances w.r.t. parameters

\[
\frac{d\text{Loss(minibatch)}}{dw_{i,j}^{(k)}} = \frac{1}{B} \sum_t \frac{d\text{Div}(Y_t(X_t), d_t(X_t))}{dw_{i,j}^{(k)}}
\]

• In conventional training, both, the output of the network in response to an input, and the derivative of the divergence for any input are independent of other inputs in the minibatch

• If we use Batch Norm, the above relation gets a little complicated
A note on derivatives

• The outputs are now functions of $\mu_B$ and $\sigma_B^2$ which are functions of the entire minibatch

$$\text{Loss(minibatch)}$$

$$= \frac{1}{B} \sum_t \text{Div}(Y_t(X_t, \mu_B, \sigma_B^2), d_t(X_t))$$

• The Divergence for each $Y_t$ depends on all the $X_t$ within the minibatch
  – Training instances within the minibatch are no longer independent
The actual divergence with BN

- The actual divergence for any minibatch with terms explicitly written

\[
\text{Loss(minibatch)} = \frac{1}{B} \sum_t \text{Div} \left( Y_t \left( X_t, \mu_B(X_t), \sigma_B^2(X_t, X_{t'}, \mu_B(X_t)) \right), d_t(X_t) \right)
\]

- We need the derivative for this function

- To derive the derivative let's consider the dependencies at a single neuron
  - Shown pictorially in the following slide
Batchnorm is a vector function over the minibatch

- Batch normalization is really a vector function applied over all the inputs from a minibatch
  - Every $z_i$ affects every $\hat{z}_j$
  - Shown on the next slide
- To compute the derivative of the minibatch loss w.r.t any $z_i$, we must consider all $\hat{z}_j$s in the batch
Or more explicitly

- The computation of mini-batch normalized $u$’s is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each $u$ to compute the corresponding $\hat{z}$
Or more explicitly

- The computation of mini-batch normalized u’s is a vector function
  - Invoking mean and variance statistics across the minibatch
- The subsequent shift and scaling is individually applied to each u to compute the corresponding \( \hat{z} \)

\[
\begin{align*}
    u_i &= \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\
    \hat{z}_i &= \gamma u_i + \beta
\end{align*}
\]

We can compute \( \frac{d\text{Loss}}{du_i} \) individually for each \( u_i \) because the processing after the computation of \( u_i \) is independent for each \( u_i \)
Batch normalization: Forward pass

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

\[ \hat{z}_i = yu_i + \beta \]
Batch normalization: Backpropagation

\[
\frac{d\text{Loss}}{d\hat{z}} = f'(\hat{z}) \frac{d\text{Loss}}{dy}
\]

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

\[
\hat{z}_i = \gamma u_i + \beta
\]
Batch normalization:
Backpropagation

\[
\frac{d\text{Loss}}{d\beta} = \frac{d\text{Loss}}{d\hat{z}}
\]

\[
\frac{d\text{Loss}}{dy} = u \frac{d\text{Loss}}{d\hat{z}}
\]

Parameters to be learned

\[
\hat{\hat{z}} = f'(\hat{z}) \frac{d\text{Loss}}{dy}
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

\[
\hat{z}_i = \gamma u_i + \beta
\]

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

i_1

i_2

\ldots

i_{N-1}

i_N

\[
+ \]

Batch normalization

z

u

\hat{\hat{z}}

f(\hat{\hat{z}})

y
Batch normalization: Backpropagation

\[ \frac{d\text{Loss}}{d\beta} = \frac{d\text{Loss}}{d\hat{z}} \]

\[ \frac{d\text{Loss}}{dy} = u \frac{d\text{Loss}}{d\hat{z}} \]

\[ \frac{d\text{Loss}}{du} = \gamma \frac{d\text{Loss}}{d\hat{z}} \]

\[ \frac{d\text{Loss}}{d\hat{z}} = f'(\hat{z}) \frac{d\text{Loss}}{dy} \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

\[ \hat{z}_i = \gamma u_i + \beta \]

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
We now have $\frac{d\text{Loss}}{du_i}$ for every $u_i$.

We must propagate the derivative through the first stage of BN.
  - Which is a vector operation over the minibatch.
The first stage of batchnorm

- The complete dependency figure for the first "normalization" stage of Batchnorm
  - Which computes the centered "u"s from the "z"s for the minibatch

- Note: inputs and outputs are different instances in a minibatch
  - The diagram represents BN occurring at a single neuron

- Let’s complete the figure and work out the derivatives
The first stage of Batchnorm

- The complete derivative of the mini-batch loss w.r.t. $z_i$

\[
\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j} \frac{du_j}{dz_i}
\]
The first stage of Batchnorm

• The complete derivative of the mini-batch loss w.r.t. $z_i$

$$\frac{d\text{Loss}}{dz_i} = \sum_j \frac{d\text{Loss}}{du_j} \frac{du_j}{dz_i}$$

Already computed
The first stage of Batchnorm

- The complete derivative of the mini-batch loss w.r.t. $z_i$

$$\frac{d\text{Loss}}{dz_i} = \sum_j \frac{d\text{Loss}}{du_j} \frac{du_j}{dz_i}$$

Must compute for every $i,j$ pair
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \]
The first stage of Batchnorm

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B} \frac{d\mu_B}{dz_i} +
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B} \frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B} \frac{d \mu_B}{d z_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \left( \frac{\partial u_i}{\partial z_i} \right) + \frac{\partial u_i}{\partial \mu_B} \frac{d \mu_B}{d z_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

• From the highlighted relation

\[ \frac{\partial u_i}{\partial z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{du_i}{dz_i} = \frac{\partial u_i}{\partial z_i} + \frac{\partial u_i}{\partial \mu_B} \frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dZ_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial u_i}{\partial \mu_B} \frac{d\mu_B}{dZ_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dZ_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \left( \frac{\partial u_i}{\partial \mu_B} \right) \frac{d\mu_B}{dz_i} + \left( \frac{\partial u_i}{\partial \sigma_B^2} \right) \frac{d\sigma_B^2}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted relation

\[ \frac{\partial u_i}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial u_i}{\partial \mu_B} \frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{d \mu_B}{d z_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{d\mu_B}{dz_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted relation

\[ \frac{\partial \mu_B}{\partial z_i} = \frac{1}{B} \]
The first stage of Batchnorm

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

- The derivative for the “through” line \((i = j)\)

\[
\frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} - \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{d \mu_B}{d z_i} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{1}{B} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} - \frac{1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]
\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted equation

\[ \frac{\partial u_i}{\partial \sigma_B^2} = \frac{-(z_i - \mu_B)}{2} \left( \sigma_B^2 + \epsilon \right)^{-3/2} \]
The first stage of Batchnorm

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

- The derivative for the “through” line \((i = j)\)

\[
\frac{d u_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial u_i}{\partial \sigma_B^2} \frac{d \sigma_B^2}{dz_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]
\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)}{2(\sigma_B^2 + \epsilon)^{3/2}} \frac{d \sigma_B^2}{d z_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)}{2(\sigma_B^2 + \epsilon)^{3/2}} \frac{d \sigma_B^2}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted equations

\[ \frac{d\sigma_B^2}{dz_i} = \frac{\partial \sigma_B^2}{\partial z_i} + \frac{\partial \sigma_B^2}{\partial \mu_B} \frac{d\mu_B}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted equations

\[ \frac{d\sigma_B^2}{dz_i} = \left( \frac{\partial \sigma_B^2}{\partial z_i} \right) + \frac{\partial \sigma_B^2}{\partial \mu_B} \frac{d\mu_B}{dz_i} \]
The first stage of Batchnorm

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

- From the highlighted equations

\[
\frac{\partial \sigma_B^2}{\partial z_i} = \frac{2(z_i - \mu_B)}{B}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

• From the highlighted equations

\[ \frac{d \sigma_B^2}{dz_i} = \left( \frac{\partial \sigma_B^2}{\partial z_i} \right) + \frac{\partial \sigma_B^2}{\partial \mu_B} \frac{d \mu_B}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted equations

\[ \frac{d\sigma_B^2}{dz_i} = \frac{2(z_i - \mu_B)}{B} \left( \frac{\partial \sigma_B^2}{\partial \mu_B} \right)^{-1} \frac{d\mu_B}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

\[ \frac{\partial \sigma_B^2}{\partial \mu_B} = \frac{1}{B} \sum_{i=1}^{B} -2(z_i - \mu_B) = -2 \left( \frac{1}{B} \sum_{i=1}^{B} z_i - \frac{1}{B} \sum_{i=1}^{B} \mu_B \right) \]

\[ = -2 \left( \mu_B - \frac{1}{B} B \mu_B \right) = -2(\mu_B - \mu_B) = 0 \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]
\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- From the highlighted equations

\[ \frac{d\sigma_B^2}{dz_i} = \frac{2(z_i - \mu_B)}{B} + \frac{\partial \sigma_B^2}{\partial \mu_B} \frac{d\mu_B}{dz_i} \]
The first stage of Batchnorm

\[
\mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i
\]

\[
\sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2
\]

\[
u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\]

- From the highlighted equations

\[
\frac{d\sigma_B^2}{dz_i} = \frac{2(z_i - \mu_B)}{B}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[
\frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)}{2(\sigma_B^2 + \epsilon)^{3/2}} \frac{d \sigma_B^2}{d z_i}
\]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{d u_i}{d z_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)}{2(\sigma_B^2 + \epsilon)^{3/2}} \frac{2(z_i - \mu_B)}{B} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]
\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “through” line \((i = j)\)

\[ \frac{du_i}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B (\sigma_B^2 + \epsilon)^{3/2}} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]
\[ \sigma^2_B = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]
\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma^2_B + \epsilon}} \]

- The derivative for the “cross” lines \((i \neq j)\)

\[ \frac{d u_j}{d z_i} = \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “cross” lines \((i \neq j)\)

\[ \frac{d u_j}{d z_i} = \frac{\partial u_j}{\partial \mu_B} \frac{d \mu_B}{d z_i} + \frac{\partial u_j}{\partial \mu_B} \frac{d \mu_B}{d z_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “cross” lines \( (i \neq j) \)

\[ \frac{d\mu_B}{dz_i} = \frac{\partial u_j}{\partial \mu_B} \frac{d\mu_B}{dz_i} + \frac{\partial u_j}{\partial \sigma_B^2} \frac{d\sigma_B^2}{dz_i} \]
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

• The derivative for the “cross” lines \((i \neq j)\)

\[
\frac{d u_j}{d z_i} = \frac{\partial u_j}{\partial \mu_B} \frac{d \mu_B}{d z_i} + \frac{\partial u_j}{\partial \sigma_B^2} \frac{d \sigma_B^2}{d z_i}
\]

This is identical to the equation for \(i = j\), without the first “through” term.
The first stage of Batchnorm

\[ \mu_B = \frac{1}{B} \sum_{i=1}^{B} z_i \]

\[ \sigma_B^2 = \frac{1}{B} \sum_{i=1}^{B} (z_i - \mu_B)^2 \]

\[ u_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \]

- The derivative for the “cross” lines (\(i \neq j\))

\[ \frac{d u_j}{d z_i} = \frac{-1}{B \sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B (\sigma_B^2 + \epsilon)^{3/2}} \]
The first stage of Batchnorm

\[
\frac{du_j}{dz_i} = \begin{cases} 
\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j = i \\
\frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j \neq i 
\end{cases}
\]
The first stage of Batchnorm

- The complete derivative of the mini-batch loss w.r.t. $z_i$

$$\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j} \frac{du_j}{dz_i}$$
**The first stage of Batchnorm**

\[
\frac{du_j}{dz_i} = \begin{cases} 
\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j = i \\
\frac{-1}{B\sqrt{\sigma_B^2 + \epsilon}} + \frac{-(z_i - \mu_B)^2}{B(\sigma_B^2 + \epsilon)^{3/2}} & \text{if } j \neq i 
\end{cases}
\]

\[
\frac{dLoss}{dz_i} = \sum_j \frac{dLoss}{du_j} \frac{du_j}{dz_i}
\]

- The complete derivative of the mini-batch loss w.r.t. \(z_i\)

\[
\frac{dLoss}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{dLoss}{du_i} - \frac{1}{B\sqrt{\sigma_B^2 + \epsilon}} \sum_j \frac{dLoss}{du_j} - \frac{1}{B(\sigma_B^2 + \epsilon)^{3/2}} \sum_j \frac{dLoss}{du_j} (z_i - \mu_B)^2
\]
Batch normalization: Backpropagation

\[
\frac{d\text{Loss}}{dz_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{d\text{Loss}}{du_i} - \frac{1}{B \sqrt{\sigma_B^2 + \epsilon}} \sum_j \frac{d\text{Loss}}{du_j} - \frac{1}{B(\sigma_B^2 + \epsilon)^{3/2}} \sum_j \frac{d\text{Loss}}{du_j} (z_i - \mu_B)^2
\]

The rest of backprop continues from \( \frac{\partial \text{Loss}}{\partial z_i} \)
Batch normalization: Inference

- On test data, BN requires $\mu_B$ and $\sigma_B^2$.
- We will use the average over all training minibatches

$$
\mu_{BN} = \frac{1}{N_{batches}} \sum_{batch} \mu_B(batch)
$$

$$
\sigma_{BN}^2 = \frac{B}{(B-1)N_{batches}} \sum_{batch} \sigma_B^2(batch)
$$

- Note: these are neuron-specific
  - $\mu_B(batch)$ and $\sigma_B^2(batch)$ here are obtained from the final converged network
  - The $B/(B-1)$ term gives us an unbiased estimator for the variance
Batch normalization may only be applied to some layers
  - Or even only selected neurons in the layer

Implements both convergence rate and neural network performance
  - Anecdotal evidence that BN eliminates the need for dropout
  - To get maximum benefit from BN, learning rates must be increased and learning rate decay can be faster
    • Since the data generally remain in the high-gradient regions of the activations
  - Also needs better randomization of training data order
Batch Normalization: Typical result

- Performance on Imagenet, from Ioffe and Szegedy, JMLR 2015
• Gradient descent can be sped up by incremental updates

• Convergence can be improved using smoothed updates

• The choice of divergence affects both the learned network and results

• Covariate shift between training and test may cause problems and may be handled by batch normalization
The problem of data underspecification

• The figures shown to illustrate the learning problem so far were *fake news*.
Learning the network

• We attempt to learn an entire function from just a few snapshots of it
General approach to training

- Define a divergence between the actual network output for any parameter value and the desired output
  - Typically L2 divergence or KL divergence

\[ E = \sum_i (d_i - f(x_i, W))^2 \]
Overfitting

- Problem: Network may just learn the values at the inputs
  - Learn the red curve instead of the dotted blue one
    - Given only the red vertical bars as inputs
Data under-specification

• Consider a binary 100-dimensional input
• There are $2^{100} = 10^{30}$ possible inputs
• Complete specification of the function will require specification of $10^{30}$ output values
• A training set with only $10^{15}$ training instances will be off by a factor of $10^{15}$
Data under-specification in learning

• Consider a binary 100-dimensional input
• There are $2^{100} = 10^{30}$ possible inputs
• Complete specification of the function will require specification of $10^{30}$ output values
• A training set with only $10^{15}$ training instances will be off by a factor of $10^{15}$

Find the function!
Need “smoothing” constraints

- Need additional constraints that will “fill in” the missing regions acceptably
  - Generalization
Smoothness through weight manipulation

• Illustrative example: Simple binary classifier
  – The “desired” output is generally smooth
Smoothness through weight manipulation

Illustrative example: Simple binary classifier
- The “desired” output is generally smooth
  - Capture statistical or average trends
- An unconstrained model will model individual instances instead
• **Illustrative example: Simple binary classifier**
  – The “desired” output is generally smooth
    • Capture statistical or average trends
  – **An unconstrained model will model individual instances instead**
Why overfitting

These sharp changes happen because ..

..the perceptrons in the network are individually capable of sharp changes in output
The individual perceptron

- Using a sigmoid activation
  - As $|w|$ increases, the response becomes steeper
• Steep changes that enable overfitted responses are facilitated by perceptrons with large $w$
Smoothness through weight manipulation

- Steep changes that enable overfitted responses are facilitated by perceptrons with large $w$

- Constraining the weights $w$ to be low will force slower perceptrons and smoother output response
Objective function for neural networks

Desired output of network: $d_t$

Error on i-th training input: $\text{Div}(Y_t, d_t; W_1, W_2, ..., W_K)$

Training loss:

$$\text{Loss}(W_1, W_2, ..., W_K) = \frac{1}{T} \sum_t \text{Div}(Y_t, d_t; W_1, W_2, ..., W_K)$$

• Conventional training: minimize the loss:

$$\hat{W}_1, \hat{W}_2, ..., \hat{W}_K = \arg\min_{W_1,W_2,...,W_K} \text{Loss}(W_1, W_2, ..., W_K)$$
Smoothness through weight constraints

- Regularized training: minimize the loss while also minimizing the weights

\[ L(W_1, W_2, ..., W_K) = \frac{1}{T} \sum_t \text{Div}(Y_t, d_t; W_1, W_2, ..., W_K) + \frac{1}{2} \lambda \sum_k \|W_k\|_F^2 \]

\( \hat{W}_1, \hat{W}_2, ..., \hat{W}_K = \arg\min_{W_1, W_2, ..., W_K} L(W_1, W_2, ..., W_K) \)

- \( \lambda \) is the regularization parameter whose value depends on how important it is for us to want to minimize the weights

- Increasing \( \lambda \) assigns greater importance to shrinking the weights
  - Make greater error on training data, to obtain a more acceptable network
Regularizing the weights

\[
L(W_1, W_2, \ldots, W_K) = \frac{1}{T} \sum_t \text{Div}(Y_t, d_t) + \frac{1}{2} \lambda \sum_k \|W_k\|_F^2
\]

- Batch mode:
  \[
  \Delta W_k = \frac{1}{T} \sum_t \nabla_{W_k} \text{Div}(Y_t, d_t)^T + \lambda W_k
  \]
- SGD:
  \[
  \Delta W_k = \nabla_{W_k} \text{Div}(Y_t, d_t)^T + \lambda W_k
  \]
- Minibatch:
  \[
  \Delta W_k = \frac{1}{b} \sum_{\tau=t}^{t+b-1} \nabla_{W_k} \text{Div}(Y_\tau, d_\tau)^T + \lambda W_k
  \]
- Update rule:
  \[
  W_k \leftarrow W_k - \eta \Delta W_k
  \]
Incremental Update: Mini-batch update

• Given \((X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)\)
• Initialize all weights \(W_1, W_2, \ldots, W_K; j = 0\)
• Do:
  – Randomly permute \((X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)\)
  – For \(t = 1: b: T\)
    • \(j = j + 1\)
    • For every layer \(k:\)
      – \(\Delta W_k = 0\)
    • For \(t' = t : t+b-1\)
      – For every layer \(k:\)
        » Compute \(\nabla_{W_k} \text{Div}(Y_t, d_t)\)
        » \(\Delta W_k = \Delta W_k + \nabla_{W_k} \text{Div}(Y_t, d_t)^T\)
  • Update
    – For every layer \(k:\)
      \[ W_k = W_k - \eta_j (\Delta W_k + \lambda W_k) \]
• Until Loss has converged
Smoothness through network structure

• Smoothness constraints can also be imposed through the network *structure*

• *For a given number of parameters deeper networks impose more smoothness than shallow ones*
  
  – Each layer works on the already smooth surface output by the previous layer.
Minimal correct architectures are hard to train

• Typical results (varies with initialization)
• 1000 training points – orders of magnitude more than you usually get
• All the training tricks known to mankind
But depth and training data help

- Deeper networks seem to learn better, for the same number of total neurons
  - *Implicit smoothness constraints*
    - *As opposed to explicit constraints from more conventional regularization methods*
- Training with more data is also better 😊
Story so far

• Gradient descent can be sped up by incremental updates
• Convergence can be improved using smoothed updates

• The choice of divergence affects both the learned network and results
• Covariate shift between training and test may cause problems and may be handled by batch normalization
• Data underspecification can result in overfitted models and must be handled by regularization and more constrained (generally deeper) network architectures
Regularization..

• Other techniques have been proposed to improve the smoothness of the learned function
  – $L_1$ regularization of network activations
  – Regularizing with added noise..

• Possibly the most influential method has been “dropout”
A brief detour.. Bagging

• Popular method proposed by Leo Breiman:
  – Sample training data and train several different classifiers
  – Classify test instance with entire ensemble of classifiers
  – Vote across classifiers for final decision
  – Empirically shown to improve significantly over training a single classifier from combined data

• Returning to our problem....
• **During training:** For each input, at each iteration, “turn off” each neuron with a probability $1-\alpha$
• **During training:** For each input, at each iteration, “turn off” each neuron with a probability 1-\( \alpha \)
  – Also turn off inputs similarly
• **During training:** For each input, at each iteration, “turn off” each neuron (including inputs) with a probability $1 - \alpha$
  
  – In practice, set them to 0 according to the failure of a Bernoulli random number generator with success probability $\alpha$
Dropout

During training: For each input, at each iteration, “turn off” each neuron (including inputs) with a probability $1 - \alpha$

- In practice, set them to 0 according to the failure of a Bernoulli random number generator with success probability $\alpha$
- **During training:** Backpropagation is effectively performed only over the remaining network
  - The effective network is different for different inputs
  - Gradients are obtained only for the weights and biases *from* “On” nodes *to* “On” nodes
    - For the remaining, the gradient is just 0

The pattern of dropped nodes changes for each input, i.e. in every pass through the net.
• For a network with a total of $N$ neurons, there are $2^N$ possible sub-networks
  – Obtained by choosing different subsets of nodes
  – Dropout *samples* over all $2^N$ possible networks
  – Effectively learns a network that *averages* over all possible networks
  • Bagging
Dropout as a mechanism to increase pattern density

- Dropout forces the neurons to learn “rich” and redundant patterns
- E.g. without dropout, a non-compressive layer may just “clone” its input to its output
  - Transferring the task of learning to the rest of the network upstream
- Dropout forces the neurons to learn denser patterns
  - With redundancy
The forward pass

- Input: $D$ dimensional vector $\mathbf{x} = [x_j, \ j = 1 \ldots D]$
- Set:
  - $D_0 = D$, is the width of the $0^{th}$ (input) layer
  - $y_j^{(0)} = x_j, \ j = 1 \ldots D$; $y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  # Mask takes value 1 with prob. $\alpha$, 0 with prob $1 - \alpha$
  - $\text{mask}(k - 1, j) = \text{Bernoulli}(\alpha), \ j = 1 \ldots D_{k-1}$
  - $y_j^{(k-1)} = y_j^{(k-1)} \cdot \text{mask}(k - 1, j), \ j = 1 \ldots D_{k-1}$
  - For $j = 1 \ldots D_k$
    - $z_j^{(k)} = \sum_{i=0}^{N_k} w_{i,j}^{(k)} y_i^{(k-1)} + b_j^{(k)}$
    - $y_j^{(k)} = f_k(z_j^{(k)})$
- Output:
  - $Y = y_j^{(N)}, j = 1 \ldots D_N$
Backward Pass

• Output layer (N):

\[- \frac{\partial \text{Div}}{\partial Y_i} = \frac{\partial \text{Div}(Y,d)}{\partial y_i^{(N)}} \]

\[- \frac{\partial \text{Div}}{\partial z_i^{(k)}} = f_k' \left( z_i^{(k)} \right) \frac{\partial \text{Div}}{\partial y_i^{(k)}} \]

• For layer $k = N - 1$ \textit{down to} 0

  – For $i = 1 \ldots D_k$

    • \( \frac{\partial \text{Div}}{\partial y_i^{(k)}} = \text{mask}(k,i) \sum_j w_{ij}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}} \)

    • \( \frac{\partial \text{Div}}{\partial z_i^{(k)}} = f_k' \left( z_i^{(k)} \right) \frac{\partial \text{Div}}{\partial y_i^{(k)}} \)

    • \( \frac{\partial \text{Div}}{\partial w_{ij}^{(k+1)}} = y_i^{(k)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}} \) for $j = 1 \ldots D_{k+1}$
Testing with Dropout

- Dropout effectively trains $2^N$ networks.
- On test data the “Bagged” output, in principle, is the ensemble average over all $2^N$ networks and is thus the statistical expectation of the output over all networks:
  \[
  Y = E \left[ \text{network} \left( y_j^{(k)}, j = 1 \ldots D_k, k = 1 \ldots K \right) \right]
  \]
  - Explicitly showing the network as a function of the outputs of individual neurons in the net.

- We cannot explicitly compute this expectation.

- Instead we will use the following approximation:
  \[
  E \left[ \text{network} \left( y_j^{(k)}, \forall k, j \right) \right] = \text{network} \left( E \left[ y_j^{(k)} \right] \forall k, j \right)
  \]
  - Where $E \left[ y_j^{(k)} \right]$ is the expected output of the jth neuron in the kth layer over all networks in the ensemble.
  - I.e. approximate the expectation of a function as the function of expectations.

- We require $E \left[ y_j^{(k)} \right]$ to compute this.
What each neuron computes

• Each neuron actually has the following activation:

\[ y_i^{(k)} = D\sigma \left( \sum_j w_{ji}^{(k)} y_j^{(k-1)} + b_i^{(k)} \right) \]

  – Where \( D \) is a Bernoulli variable that takes a value 1 with probability \( \alpha \)

• \( D \) may be switched on or off for individual sub networks, but over the ensemble, the expected output of the neuron is

\[ E[y_i^{(k)}] = \alpha\sigma \left( \sum_j w_{ji}^{(k)} y_j^{(k-1)} + b_i^{(k)} \right) \]

• During test time, we will use the expected output of the neuron
  – Consists of simply scaling the output of each neuron by \( \alpha \)
Dropout during test: implementation

• Instead of multiplying every output by $\alpha$, multiply all weights by $\alpha$
Alternately, during *training*, replace the activation of all neurons in the network by $\alpha^{-1}\sigma(.)$

- This does not affect the dropout procedure itself
- We will use $\sigma(.)$ as the activation during testing, and not modify the weights
Inference with dropout (testing)

- Input: $D$ dimensional vector $\mathbf{x} = [x_j, \ j = 1 \ldots D$
- Set:
  - $D_0 = D$, is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j, \ j = 1 \ldots D$; $y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$
    - $z_j^{(k)} = \sum_{i=0}^{N_k} w_{i,j} y_i^{(k-1)} + b_j^{(k)}$
    - $y_j^{(k)} = \alpha f_k(z_j^{(k)})$
- Output:
  - $Y = y_j^{(N)}$, $j = 1 \ldots D_N$
Dropout: Typical results

- From Srivastava et al., 2013. Test error for different architectures on MNIST with and without dropout
  - 2-4 hidden layers with 1024-2048 units
Variations on dropout

• Zoneout: For RNNs
  – Randomly chosen units remain unchanged across a time transition

• Dropconnect
  – Drop individual connections, instead of nodes

• Shakeout
  – Scale up the weights of randomly selected weights
    • $|w| \rightarrow \alpha|w| + (1 - \alpha)c$
  – Fix remaining weights to a negative constant
    • $w \rightarrow -c$

• Whiteout
  – Add or multiply weight-dependent Gaussian noise to the signal on each connection
Story so far

• Gradient descent can be sped up by incremental updates
• Convergence can be improved using smoothed updates

• The choice of divergence affects both the learned network and results
• Covariate shift between training and test may cause problems and may be handled by batch normalization
• Data underspecification can result in overfitted models and must be handled by regularization and more constrained (generally deeper) network architectures
• “Dropout” is a stochastic data/model erasure method that sometimes forces the network to learn more robust models
Other heuristics: Early stopping

- Continued training can result in over fitting to training data
  - Track performance on a held-out validation set
  - Apply one of several early-stopping criterion to terminate training when performance on validation set degrades significantly
Additional heuristics: Gradient clipping

- Often the derivative will be too high
  - When the divergence has a steep slope
  - This can result in instability
- **Gradient clipping**: set a ceiling on derivative value

\[
\text{if } \partial_w D > \theta \text{ then } \partial_w D = \theta
\]

- Typical $\theta$ value is 5
Additional heuristics: Data Augmentation

- Available training data will often be small
- “Extend” it by distorting examples in a variety of ways to generate synthetic labelled examples
  - E.g. rotation, stretching, adding noise, other distortion
Other tricks

• Normalize the input:
  – Normalize entire training data to make it 0 mean, unit variance
  – Equivalent of batch norm on input

• A variety of other tricks are applied
  – Initialization techniques
    • Xavier, Kaiming, SVD, etc.
    • Key point: neurons with identical connections that are identically initialized will never diverge
  – Practice makes man perfect
Setting up a problem

- Obtain training data
  - Use appropriate representation for inputs and outputs

- Choose network architecture
  - More neurons need more data
  - Deep is better, but harder to train

- Choose the appropriate divergence function
  - Choose regularization

- Choose heuristics (batch norm, dropout, etc.)

- Choose optimization algorithm
  - E.g. ADAM

- Perform a grid search for hyper parameters (learning rate, regularization parameter, ...) on held-out data

- Train
  - Evaluate periodically on validation data, for early stopping if required
In closing

• Have outlined the process of training neural networks
  – Some history
  – A variety of algorithms
  – Gradient-descent based techniques
  – Regularization for generalization
  – Algorithms for convergence
  – Heuristics

• Practice makes perfect..