Training neural nets through Empirical Risk Minimization: Problem Setup

• Given a training set of input-output pairs 
  \((X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)\)

• The divergence on the \(i^{th}\) instance is \(\text{div}(Y_i, d_i)\)
  
  \(Y_i = f(X_i; W)\)

• The loss (empirical risk)

\[
\text{Loss} = \frac{1}{T} \sum_i \text{div}(Y_i, d_i)
\]

• Minimize \(\text{Loss}\) w.r.t \(\{w_{ij}^{(k)}, b_j^{(k)}\}\) using gradient descent
Notation

• The input layer is the $0^{\text{th}}$ layer

• We will represent the output of the $i$-th perceptron of the $k^{\text{th}}$ layer as $y_i^{(k)}$
  
  – Input to network: $y_i^{(0)} = x_i$
  
  – Output of network: $y_i = y_i^{(N)}$

• We will represent the weight of the connection between the $i$-th unit of the $k$-1th layer and the $j$th unit of the $k$-th layer as $w_{ij}^{(k)}$
  
  – The bias to the $j$th unit of the $k$-th layer is $b_j^{(k)}$
Recap: Gradient Descent Algorithm

- **Initialize:**
  - $W^0$
  - $k = 0$

- **do**
  - $W^{k+1} = W^k - \eta^k \nabla \text{Loss}(W^k)^T$
  - $k = k + 1$

- **while** $|\text{Loss}(W^k) - \text{Loss}(W^{k-1})| > \varepsilon$
Recap: Gradient Descent Algorithm

• In order to minimize $L(W)$ w.r.t. $W$

• Initialize:
  - $W^0$
  - $k = 0$

• do
  - For every component $i$
    • $W_i^{k+1} = W_i^k - \eta^k \frac{\partial L}{\partial W_i}$
    - $k = k + 1$

• while $\left| L(W^k) - L(W^{k-1}) \right| > \varepsilon$
Training Neural Nets through Gradient Descent

Total training Loss:

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

- Gradient descent algorithm:
- Initialize all weights and biases \( \{ w_{ij}^{(k)} \} \)
  - Using the extended notation: the bias is also a weight
- Do:
  - For every layer \( k \) for all \( i, j \), update:
    - \( w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{\text{dLoss}}{\text{dw}_{i,j}^{(k)}} \)
- Until \( Loss \) has converged
Training Neural Nets through Gradient Descent

Total training Loss:

\[ \text{Loss} = \frac{1}{T} \sum_{t} \text{Div}(Y_t, d_t) \]

• Gradient descent algorithm:
  - Initialize all weights \( \{w_{ij}^{(k)}\} \)
  - Do:
    - For every layer \( k \) for all \( i,j \), update:
      - \( w_{i,j}^{(k)} = w_{i,j}^{(k)} - \eta \frac{d\text{Loss}}{dw_{i,j}^{(k)}} \)
  - Until \( Err \) has converged

Assuming the bias is also represented as a weight
The derivative

Total training Loss:

\[
Loss = \frac{1}{T} \sum_t Div(Y_t, d_t)
\]

• Computing the derivative

Total derivative:

\[
\frac{dLoss}{dw^{(k)}_{i,j}} = \frac{1}{T} \sum_t \frac{dDiv(Y_t, d_t)}{dw^{(k)}_{i,j}}
\]
Training by gradient descent

- Initialize all weights \( \{ w_{ij}^{(k)} \} \)

- Do:
  - For all \( i, j, k \), initialize \( \frac{d\text{Loss}}{dw_{i,j}^{(k)}} = 0 \)
  - For all \( t = 1:T \)
    - For every layer \( k \) for all \( i, j \):
      - Compute \( \frac{d\text{Div}(Y_t, d_t)}{dw_{i,j}^{(k)}} \)
      - \( \frac{d\text{Loss}}{dw_{i,j}^{(k)}} + \frac{d\text{Div}(Y_t, d_t)}{dw_{i,j}^{(k)}} \)
    - For every layer \( k \) for all \( i, j \):
      \[
      w_{i,j}^{(k)} = w_{i,j}^{(k)} - \frac{\eta d\text{Loss}}{T dw_{i,j}^{(k)}}
      \]
  - Until \textit{Err} has converged
The derivative

Total training Loss:

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

Total derivative:

\[ \frac{d\text{Loss}}{dw^{(k)}_{i,j}} = \frac{1}{T} \sum_t \frac{d\text{Div}(Y_t, d_t)}{dw^{(k)}_{i,j}} \]

- So we must first figure out how to compute the derivative of divergences of individual training inputs
Calculus Refresher: Basic rules of calculus

For any differentiable function
\[ y = f(x) \]
with derivative
\[ \frac{dy}{dx} \]
the following must hold for sufficiently small \( \Delta x \)
\[ \Delta y \approx \frac{dy}{dx} \Delta x \]

For any differentiable function
\[ y = f(x_1, x_2, \ldots, x_M) \]
with partial derivatives
\[ \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_M} \]
the following must hold for sufficiently small \( \Delta x_1, \Delta x_2, \ldots, \Delta x_M \)
\[ \Delta y \approx \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \cdots + \frac{\partial y}{\partial x_M} \Delta x_M \]

Both by the definition
\[ \Delta y = \nabla_x f \Delta x \]
Calculus Refresher: Chain rule

For any nested function \( y = f(g(x)) \)

\[
\frac{dy}{dx} = \frac{df}{dg(x)} \frac{dg(x)}{dx}
\]

Check - we can confirm that:

\( \Delta y = \frac{dy}{dx} \Delta x \)

\( z = g(x) \quad \Rightarrow \quad \Delta z = \frac{dg(x)}{dx} \Delta x \)

\( y = f(z) \quad \Rightarrow \quad \Delta y = \frac{df}{dz} \Delta z = \frac{df}{dg(x)} \frac{dg(x)}{dx} \Delta x \)
Calculus Refresher: Distributed Chain rule

\[ y = f(g_1(x), g_1(x), \ldots, g_M(x)) \]

\[
\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} + \ldots + \frac{\partial f}{\partial g_M(x)} \frac{dg_M(x)}{dx}
\]

Check: \[ \Delta y = \frac{dy}{dx} \Delta x \]

Let \( z_i = g_i(x) \)

\[ \Delta y = \frac{\partial f}{\partial z_1} \Delta z_1 + \frac{\partial f}{\partial z_2} \Delta z_2 + \ldots + \frac{\partial f}{\partial z_M} \Delta z_M \]

\[ \Delta y = \frac{\partial f}{\partial z_1} \frac{dz_1}{dx} \Delta x + \frac{\partial f}{\partial z_2} \frac{dz_2}{dx} \Delta x + \ldots + \frac{\partial f}{\partial z_M} \frac{dz_M}{dx} \Delta x \]

\[ \Delta y = \left( \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} + \ldots + \frac{\partial f}{\partial g_M(x)} \frac{dg_M(x)}{dx} \right) \Delta x \]
Calculus Refresher: Distributed Chain rule

\[ y = f(g_1(x), g_2(x), \ldots, g_M(x)) \]

\[
\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)} \frac{dg_M(x)}{dx}
\]

Check: \[ \Delta y = \frac{dy}{dx} \Delta x \]

\[ \Delta y = \frac{\partial f}{\partial g_1(x)} \Delta g_1(x) + \frac{\partial f}{\partial g_2(x)} \Delta g_2(x) + \cdots + \frac{\partial f}{\partial g_M(x)} \Delta g_M(x) \]

\[ \Delta y = \left( \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} \Delta x + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} \Delta x + \cdots + \frac{\partial f}{\partial g_M(x)} \frac{dg_M(x)}{dx} \Delta x \right) \]

\[ \Delta y = \left( \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} + \cdots + \frac{\partial f}{\partial g_M(x)} \frac{dg_M(x)}{dx} \right) \Delta x \]
Distributed Chain Rule: Influence Diagram

\[ y = f(g_1(x), g_1(x), \ldots, g_M(x)) \]

- \( x \) affects \( y \) through each of \( g_1 \ldots g_M \)
Distributed Chain Rule: Influence Diagram

- Small perturbations in $x$ cause small perturbations in each of $g_1 \ldots g_M$, each of which individually additively perturbs $y$
Calculus Refresher: Chain rule summary

For any nested function \( l = f(y) \) where \( y = g(z) \)

\[
\frac{dl}{dz} = \frac{dl}{dy} \frac{dy}{dz}
\]

For \( l = f(z_1, z_2, \ldots, z_M) \)
where \( z_i = g_i(x) \)

\[
\frac{dl}{dx} = \frac{\partial l}{\partial z_1} \frac{dz_1}{dx} + \frac{\partial l}{\partial z_2} \frac{dz_2}{dx} + \cdots + \frac{\partial l}{\partial z_M} \frac{dz_M}{dx}
\]
Our problem for today

• How to compute $\frac{dDiv(Y,d)}{dw_{i,j}^{(k)}}$ for a single data instance
A first closer look at the network

- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
A first closer look at the network

- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Explicitly separating the weighted sum of inputs from the activation
A first closer look at the network

- Showing a tiny 2-input network for illustration
  - Actual network would have many more neurons and inputs
- Expanded **with all weights shown**
- Let’s label the other variables too...
Computing the derivative for a *single* input
Computing the derivative for a single input

What is: \( \frac{d \text{Div}(Y,d)}{d w_{i,j}^{(k)}} \)
Computing the gradient

• Note: computation of the derivative \( \frac{d \text{Div}(Y,d)}{d w_{i,j}^{(k)}} \) requires intermediate and final output values of the network in response to the input.
The “forward pass”

We will refer to the process of computing the output from an input as the **forward pass**

We will illustrate the forward pass in the following slides
The “forward pass”

\( y^{(0)} = x \)

\[
\begin{array}{c}
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 \\
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 \\
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 \\
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 \\
\end{array}
\]

Setting \( y_i^{(0)} = x_i \) for notational convenience

Assuming \( w_{0j}^{(k)} = b_j^{(k)} \) and \( y_0^{(k)} = 1 \) -- assuming the bias is a weight and extending the output of every layer by a constant 1, to account for the biases
The “forward pass”

\[ y^{(0)} = x \]

\[ z^{(1)} = w_{i1}^{(1)} y_i^{(0)} \]
The “forward pass”

\[ y^{(0)} = x \]

\[ z^{(1)} = \sum_{i} w_{ij}^{(1)} y_{i}^{(0)} \]
$y^{(0)} = x$

$z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)}$

$y_j^{(1)} = f_1(z_j^{(1)})$

$y^{(1)} = f_1(z^{(1)})$

$z^{(N-1)} = \sum_i w_{ij}^{(N-1)} y_i^{(N-1)}$

$y^{(N-1)} = f_{N-1}(z^{(N-1)})$

$y^{(N)} = f_N(z^{(N)})$
\[ y^{(0)} = x \]

\[ z_{j}^{(1)} = \sum_{i} w_{ij}^{(1)} y_{i}^{(0)} \]

\[ y_{j}^{(1)} = f_{1} \left( z_{j}^{(1)} \right) \]

\[ z_{j}^{(2)} = \sum_{i} w_{ij}^{(2)} y_{i}^{(1)} \]

\[ z_{j}^{(N-1)} = \sum_{i} w_{ij}^{(N-1)} y_{i}^{(N-1)} \]

\[ y^{(N)} = f_{N} \left( z^{(N)} \right) \]
\[ y^{(0)} = x \]

\[ z_j^{(1)} = \sum_i w_{ij}^{(1)} y_i^{(0)} \]

\[ y_j^{(1)} = f_1 \left( z_j^{(1)} \right) \]

\[ z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \]

\[ y_j^{(2)} = f_2 \left( z_j^{(2)} \right) \]

\[ y_j^{(N-1)} = f_{N-1} \left( z_j^{(N-1)} \right) \]

\[ y_j^{(N)} = f_N \left( z_j^{(N)} \right) \]
\[ y^{(0)} = x \]

\[ z^{(1)} = \sum_{i} w_{ij}^{(1)} y_i^{(0)} \]

\[ y_j^{(1)} = f_1 (z_j^{(1)}) \]

\[ z_j^{(2)} = \sum_{i} w_{ij}^{(2)} y_i^{(1)} \]

\[ y_j^{(2)} = f_2 (z_j^{(2)}) \]

\[ z_j^{(3)} = \sum_{i} w_{ij}^{(3)} y_i^{(2)} \]
\[ y^{(0)} = x \]

\[ z^{(1)} = y^{(0)} \]

\[ y_j^{(1)} = f_1(z_j^{(1)}) \]

\[ z_j^{(2)} = \sum_i w_{ij}^{(2)} y_i^{(1)} \]

\[ y_j^{(2)} = f_2(z_j^{(2)}) \]

\[ z_j^{(3)} = \sum_i w_{ij}^{(3)} y_i^{(2)} \]

\[ y_j^{(3)} = f_3(z_j^{(3)}) \]

...
$y^{(0)} = x$

$y_j^{(N-1)} = f_{N-1}(z_j^{(N-1)})$

$z_j^{(N)} = \sum_i w_{ij}^{(N)} y_i^{(N-1)}$

$y^{(N)} = f_N(z^{(N)})$
Forward Computation

\[ y^{(0)} = x \]

**ITERATE FOR** \( k = 1:N \)

\[ z^{(1)} \rightarrow y^{(1)} \rightarrow z^{(2)} \rightarrow y^{(2)} \rightarrow z^{(3)} \rightarrow y^{(3)} \]

for \( j = 1:\text{layer-width} \)

\[ z^{(k)}_j = \sum_i w^{(k)}_{ij} y^{(k-1)}_i \]

\[ y^{(k)}_j = f_k \left( z^{(k)}_j \right) \]
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 $k = 1 \ldots N$
  
  - For $j = 1 \ldots D_k$
    
    - $D_k$ is the size of the $k$th layer
    
    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    
    - $y_j^{(k)} = f_k \left( z_j^{(k)} \right)$

• Output:
  
  - $Y = y_j^{(N)}, j = 1 \ldots D_N$
Computing derivatives

We have computed all these intermediate values in the forward computation.

We must remember them - we will need them to compute the derivatives.
First, we compute the divergence between the output of the net $y = y^{(N)}$ and the desired output $d$. 

$y^{(0)} = x$
We then compute $\nabla_{Y(N)} \text{div}(\cdot)$, the derivative of the divergence w.r.t. the final output of the network $y^{(N)}$. 

$$y^{(0)} = x$$
We then compute \( \nabla_{Y^{(N)}} \text{div}(.) \) the derivative of the divergence w.r.t. the final output of the network \( y^{(N)} \)

We then compute \( \nabla_{z^{(N)}} \text{div}(.) \) the derivative of the divergence w.r.t. the pre-activation affine combination \( z^{(N)} \) using the chain rule
Continuing on, we will compute $\nabla_{W(N)} \text{div}(\cdot)$ the derivative of the divergence with respect to the weights of the connections to the output layer.
Computing derivatives

Continuing on, we will compute $\nabla_{W^{(N)}} \text{div}(\cdot)$ the derivative of the divergence with respect to the weights of the connections to the output layer.

Then continue with the chain rule to compute $\nabla_{Y^{(N-1)}} \text{div}(\cdot)$ the derivative of the divergence w.r.t. the output of the N-1th layer.
Computing derivatives

We continue our way backwards in the order shown

\[ \nabla_{z^{(N-1)}} \text{div}(.) \]
We continue our way backwards in the order shown

\[ \nabla_{W^{(N-1)}} div(.) \]
We continue our way backwards in the order shown.

\[ \nabla_{Y(N-2)} \text{div}(\cdot) \]
We continue our way backwards in the order shown

\[ \nabla_{z^{(N-2)}} \text{div}(.) \]
We continue our way backwards in the order shown:

\[ \nabla_{Y(1)} \text{div}(.) \]
We continue our way backwards in the order shown

\[ \nabla_{z^{(1)}} \text{div}(.) \]
We continue our way backwards in the order shown

\[ \nabla_{w^{(1)}} \text{div}(.) \]
Backward Gradient Computation

• Let’s actually see the math..
Computing derivatives

\[ y^{(0)} = x \]

\[ \text{Div}(Y, d) \]
Computing derivatives

The derivative w.r.t the actual output of the final layer of the network is simply the derivative w.r.t to the output of the network.

\[ \frac{\partial \text{Div}(Y, d)}{\partial y_i^{(N)}} = \frac{\partial \text{Div}(Y, d)}{\partial y_i} \]
Calculus Refresher: Chain rule

For any nested function \( l = f(y) \) where \( y = g(z) \)

\[
\frac{dl}{dz} = \frac{dl}{dy} \frac{dy}{dz}
\]
Computing derivatives

\( y^{(0)} = x \)

\[ f_{N-2}, f_{N-1}, f_N \]

\[ \frac{\partial \text{Div}}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial \text{Div}}{\partial y_i^{(N)}} \]
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(N)} \]

\[ z^{(1)} \]

\[ y^{(1)} \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_N \]

\[ y^{(N-2)} \]

\[ y^{(N-1)} \]

\[ y^{(N-2)} \]

\[ y^{(N-1)} \]

\[ y^{(N)} \]

\[ z^{(N)} \]

\[ \text{Div}(Y,d) \]

\[ 1 \]

\[ \frac{\partial \text{Div}}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \frac{\partial \text{Div}}{\partial y_i^{(N)}} \]

Already computed
Computing derivatives

\[ y^{(0)} = x \]

\[ f_N \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ y^{(N)} \]

\[ z^{(N)} \]

\[ \text{Derivative of activation function} \]

\[ \frac{\partial \text{Div}}{\partial z_i^{(N)}} = \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \cdot \frac{\partial \text{Div}}{\partial y_i^{(N)}} \]
Computing derivatives

Derivative of activation function computed in forward pass
Computing derivatives

\[
\frac{\partial \text{Div}}{\partial z_i^{(N)}} = f'_N \left( z_i^{(N)} \right) \frac{\partial \text{Div}}{\partial y_i^{(N)}}
\]
Computing derivatives

\( y^{(0)} = x \)

\[ y^{(1)} \]

\[ z^{(1)} \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_N \]

\[ \text{Div}(Y, d) \]

\( \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \text{Div}}{\partial z_i^{(N)}} \)
Computing derivatives

\[ y^{(0)} = x \]

\[ z^{(1)} \]

\[ y^{(1)} \]

\[ z^{(N-2)} \]

\[ z^{(N-1)} \]

\[ y^{(N-2)} \]

\[ z^{(N-1)} \]

\[ y^{(N-1)} \]

\[ y^{(1)} \]

\[ y^{(0)} \]

\[ z^{(1)} \]

\[ z^{(N-2)} \]

\[ z^{(N-1)} \]

\[ y^{(N-2)} \]

\[ z^{(N-1)} \]

\[ y^{(N-1)} \]

\[ y^{(N)} \]

\[ z^{(N)} \]

\[ \text{Div}(Y,d) \]

\[ f_N \]

\[ div() \]

\[ d \]

\[ \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \text{Div}}{\partial z_i^{(N)}} \]

Just computed
Computing derivatives

\[ y^{(0)} = x \]

Because
\[ z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms} \]
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(1)} = f_1(z^{(1)}) \]

\[ y^{(2)} = f_1(f_1(z^{(2)})) \]

\[ y^{(N)} = f_1(f_1(\ldots f_1(z^{(N)})) \ldots) \]

\[ \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \]

Because

\[ z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)} + \text{other terms} \]

Computed in forward pass
Computing derivatives

\[ y^{(0)} = x \]

\[ z^{(1)} \]

\[ f_1 \]

\[ f_1 \]

\[ f_1 \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-2} \]

\[ f_{N-2} \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]
Computing derivatives

\[ y^{(0)} = x \]

\[ \begin{align*}
  y^{(1)} &= f_1(z^{(1)}) \\
  y^{(2)} &= f_1(f_1(z^{(2)})) \\
  \vdots \\
  y^{(N-1)} &= f_1(f_1(\ldots f_1(z^{(N-1)})))) \end{align*} \]

For the bias term \( y_0^{(N-1)} = 1 \)

\[ \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \]
Calculus Refresher: Chain rule

For \( l = f(z_1, z_2, ..., z_M) \)

where \( z_i = g_i(x) \)

\[
\frac{dl}{dx} = \frac{\partial l}{\partial z_1} \frac{dz_1}{dx} + \frac{\partial l}{\partial z_2} \frac{dz_2}{dx} + ... + \frac{\partial l}{\partial z_M} \frac{dz_M}{dx}
\]
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(N)} \]

\[ z^{(N)} \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_N \]

\[ \text{Div}(\mathbf{Y}, \mathbf{d}) \]

\[
\frac{\partial \text{Div}}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \frac{\partial \text{Div}}{\partial z_j^{(N)}}
\]
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(1)} \]

\[ z^{(1)} \]

\[ f_1 \]

\[ f_1 \]

\[ f_1 \]

\[ y^{(N-2)} \]

\[ z^{(N-2)} \]

\[ f_{N-2} \]

\[ f_{N-2} \]

\[ f_{N-2} \]

\[ y^{(N-1)} \]

\[ z^{(N-1)} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ f_{N-1} \]

\[ y^{(N)} \]

\[ z^{(N)} \]

\[ f_N \]

\[ div() \]

\[ d \]

\[ div(Y, d) \]

\[
\frac{\partial Div}{\partial y^{(N-1)}_i} = \sum_j \frac{\partial z^{(N)}_j}{\partial y^{(N-1)}_i} \frac{\partial Div}{\partial z^{(N)}_j} \]

Already computed
Computing derivatives

\( y^{(0)} = x \)

\[
\frac{\partial \text{Div}}{\partial y^{(N-1)}_i} = \sum_j \frac{\partial z^{(N)}_j}{\partial y^{(N-1)}_i} \frac{\partial \text{Div}}{\partial z^{(N)}_j}
\]

Because

\( z^{(N)}_j = w^{(N)}_{ij} y^{(N-1)}_i + \text{other terms} \)
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(1)} \]

\[ z^{(1)} \]

\[ y^{(N-1)} \]

\[ z^{(N-1)} \]

\[ y^{(N-2)} \]

\[ z^{(N-2)} \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_N \]

\[ \text{Div}(Y,d) \]

\[ \text{Div}(Y,d) \]

\[ \text{div}() \]

\[ d \]

\[ \frac{\partial \text{Div}}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \]
Computing derivatives

\[ y^{(0)} = x \]

\[ y^{(1)} \]

\[ z^{(1)} \]

\[ f_1 \]

\[ f_{N-2} \]

\[ f_{N-1} \]

\[ f_N \]

\[ \partial \text{Div} \]

\[ \frac{\partial \text{Div}}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \]
We continue our way backwards in the order shown.

\[
\frac{\partial \text{Div}}{\partial z_{i}^{(N-1)}} = f'_{N-1} \left( z_{i}^{(N-1)} \right) \frac{\partial \text{Div}}{\partial y_{i}^{(N-1)}}
\]
We continue our way backwards in the order shown.

\[
\frac{\partial \text{Div}}{\partial w_{ij}^{(N-1)}} = y_i^{(N-2)} \frac{\partial \text{Div}}{\partial z_j^{(N-1)}}
\]

For the bias term \(y_0^{(N-2)} = 1\)
We continue our way backwards in the order shown

\[
\frac{\partial \text{Div}}{\partial y_i^{(N-2)}} = \sum_j w_{ij}^{(N-1)} \frac{\partial \text{Div}}{\partial z_j^{(N-1)}}
\]
We continue our way backwards in the order shown

\[
\frac{\partial \text{Div}}{\partial z_{i(N-2)}^{(N-2)}} = f'_{N-2} \left( z_{i(N-2)}^{(N-2)} \right) \frac{\partial \text{Div}}{\partial y_{i}^{(N-2)}}
\]
We continue our way backwards in the order shown

\[
\frac{\partial \text{Div}}{\partial y_1^{(1)}} = \sum_j w_{ij}^{(2)} \frac{\partial \text{Div}}{\partial z_j^{(2)}}
\]
We continue our way backwards in the order shown

\[
\frac{\partial \text{Div}}{\partial z_i^{(1)}} = f'_1 \left( z_i^{(1)} \right) \frac{\partial \text{Div}}{\partial y_i^{(1)}}
\]
We continue our way backwards in the order shown.

\[
\frac{\partial \text{Div}}{\partial w_{ij}^{(1)}} = y_i^{(0)} \frac{\partial \text{Div}}{\partial z_j^{(1)}}
\]
Gradients: Backward Computation

Initialize: Gradient w.r.t. network output

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

\[
\frac{\partial \text{Div}}{\partial z_i^{(N)}} = f_k'(z_i^{(N)}) \frac{\partial \text{Div}}{\partial y_i^{(N)}},
\]

For \( k = N - 1 \ldots 0 \)

For \( i = 1: \text{layer width} \)

\[
\frac{\partial \text{Div}}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}
\]

\[
\frac{\partial \text{Div}}{\partial z_i^{(k)}} = f'_k(z_i^{(k)}) \frac{\partial \text{Div}}{\partial y_i^{(k)}}
\]

\[\forall j \quad \frac{\partial \text{Div}}{\partial w_{ij}^{(k+1)}} = y_i^{(k)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}\]
Backward Pass

• Output layer \((N)\):
  – For \(i = 1 \ldots D_N\)
    \[
    \frac{\partial \text{Div}}{\partial y_i^{(N)}} = \frac{\partial \text{Div}(y,d)}{\partial y_i} \quad \text{[This is the derivative of the divergence]}
    \]
    \[
    \frac{\partial \text{Div}}{\partial z_i^{(N)}} = \frac{\partial \text{Div}}{\partial y_i^{(N)}} f'_N \left(z_i^{(N)}\right)
    \]
    \[
    \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \quad \text{for } j = 0 \ldots D_{N-1}
    \]

• For layer \(k = N - 1 \text{ downto } 1\)
  – For \(i = 1 \ldots D_k\)
    \[
    \frac{\partial \text{Div}}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}
    \]
    \[
    \frac{\partial \text{Div}}{\partial z_i^{(k)}} = \frac{\partial \text{Div}}{\partial y_i^{(k)}} f'_k \left(z_i^{(k)}\right)
    \]
    \[
    \frac{\partial \text{Div}}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial \text{Div}}{\partial z_j^{(k)}} \quad \text{for } j = 0 \ldots D_{k-1}
    \]
Backward Pass

- Output layer ($N$):
  - For $i = 1 \ldots D_N$
    - $\frac{\partial D_{\text{Div}}}{\partial y_i^{(N)}} = \frac{\partial D_{\text{Div}}}{\partial y_i} (y,d)
    - $\frac{\partial D_{\text{Div}}}{\partial z_i^{(N)}} = \frac{\partial D_{\text{Div}}}{\partial y_i^{(N)}} f'_N (z_i^{(N)})$
    - $\frac{\partial D_{\text{Div}}}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial D_{\text{Div}}}{\partial z_j^{(N)}}$ for $j = 0 \ldots D_{N-1}$

- For layer $k = N - 1$ *down to* 1
  - For $i = 1 \ldots D_k$
    - $\frac{\partial D_{\text{Div}}}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial D_{\text{Div}}}{\partial z_j^{(k+1)}}$
    - $\frac{\partial D_{\text{Div}}}{\partial z_i^{(k)}} = \frac{\partial D_{\text{Div}}}{\partial y_i^{(k)}} f'_k (z_i^{(k)})$
    - $\frac{\partial D_{\text{Div}}}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial D_{\text{Div}}}{\partial z_j^{(k)}}$ for $j = 0 \ldots D_{k-1}$

Called “Backpropagation” because the derivative of the loss is propagated “backwards” through the network.

Very analogous to the forward pass:

- Backward weighted combination of next layer
- Backward equivalent of activation
Using notation \( \dot{y} = \frac{\partial \text{Div}(Y,d)}{\partial y} \) etc (overdot represents derivative of \( \text{Div} \) w.r.t variable)

- **Output layer (N):**
  - For \( i = 1 \ldots D_N \)
    - \( \dot{y}_i^{(N)} = \frac{\partial \text{Div}}{\partial y_i} \)
    - \( \dot{z}_i^{(N)} = \dot{y}_i^{(N)} f'_N \left( z_i^{(N)} \right) \)
    - \( \frac{\partial \text{Div}}{\partial w_{ji}^{(N)}} = y_j^{(N-1)} \dot{z}_i^{(N)} \) for \( j = 0 \ldots D_{N-1} \)

- **For layer \( k = N - 1 \) downto 1**
  - For \( i = 1 \ldots D_k \)
    - \( \dot{y}_i^{(k)} = \sum_j w_{ij}^{(k+1)} \dot{z}_j^{(k+1)} \)
    - \( \dot{z}_i^{(k)} = \dot{y}_i^{(k)} f'_k \left( z_i^{(k)} \right) \)
    - \( \frac{\partial \text{Div}}{\partial w_{ji}^{(k)}} = y_j^{(k-1)} \dot{z}_i^{(k)} \) for \( j = 0 \ldots D_{k-1} \)

Called “Backpropagation” because the derivative of the loss is propagated “backwards” through the network.

Very analogous to the forward pass:

- Backward weighted combination of next layer
- Backward equivalent of activation
For comparison: the forward pass again

• 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
  - $y_{j}^{(0)} = x_j, \ j = 1 \ldots D; \quad y_{0}^{(k=1\ldots N)} = x_0 = 1$

• For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$
    • $z_j^{(k)} = \sum_{i=0}^{N_k} w_{i,j} y_{i}^{(k-1)}$
    • $y_{j}^{(k)} = f_k \left( z_j^{(k)} \right)$

• Output:
  - $Y = y_{j}^{(N)}, \ j = 1 \ldots D_N$
Special cases

- Have assumed so far that
  1. The computation of the output of one neuron does not directly affect computation of other neurons in the same (or previous) layers
  2. Inputs to neurons only combine through weighted addition
  3. Activations are actually differentiable
     - All of these conditions are frequently not applicable

- Will not discuss all of these in class, but explained in slides
  - Will appear in quiz. Please read the slides
Special Case 1. Vector activations

• Vector activations: all outputs are functions of all inputs
Special Case 1. Vector activations

Scalar activation: Modifying a $z_i$ only changes corresponding $y_i$

$$y_i^{(k)} = f \left( z_i^{(k)} \right)$$

Vector activation: Modifying a $z_i$ potentially changes all, $y_1 \ldots y_M$

$$\begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_M^{(k)} \end{bmatrix} = f \begin{bmatrix} Z_1^{(k)} \\ Z_2^{(k)} \\ \vdots \\ Z_D^{(k)} \end{bmatrix}$$
"Influence" diagram

Scalar activation: Each $z_i$ influences one $y_i$

Vector activation: Each $z_i$ influences all, $y_1 \ldots y_M$
• Note: The number of outputs \( y^{(k)} \) need not be the same as the number of inputs \( z^{(k)} \)
  • May be more or fewer
Scalar Activation: Derivative rule

In the case of scalar activation functions, the derivative of the error w.r.t to the input to the unit is a simple product of derivatives.
Derivatives of vector activation

• For *vector* activations the derivative of the error w.r.t. to any input is a sum of partial derivatives
  – Regardless of the number of outputs $y_j^{(k)}$

\[
\frac{\partial \text{Div}}{\partial z_i^{(k)}} = \sum_j \frac{\partial \text{Div}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}
\]

Note: derivatives of scalar activations are just a special case of vector activations:
\[
\frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} = 0 \text{ for } i \neq j
\]
Example Vector Activation: Softmax

\[
y_i^{(k)} = \frac{\exp\left(z_i^{(k)}\right)}{\sum_j \exp\left(z_j^{(k)}\right)}
\]
Example Vector Activation: Softmax

\[ y^{(k)}_i = \frac{\exp(z^{(k)}_i)}{\sum_j \exp(z^{(k)}_j)} \]

\[ \frac{\partial D_{\text{Div}}}{\partial z^{(k)}_i} = \sum_j \frac{\partial D_{\text{Div}}}{\partial y^{(k)}_j} \frac{\partial y^{(k)}_j}{\partial z^{(k)}_i} \]
Example Vector Activation: Softmax

\[ y_i^{(k)} = \frac{\exp(z_i^{(k)})}{\sum_j \exp(z_j^{(k)})} \]

\[
\frac{\partial \text{Div}}{\partial z_i^{(k)}} = \sum_j \frac{\partial \text{Div}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}
\]

\[
\frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} = \begin{cases} 
  y_i^{(k)} (1 - y_i^{(k)}) & \text{if } i = j \\
  -y_i^{(k)} y_j^{(k)} & \text{if } i \neq j
\end{cases}
\]
Example Vector Activation: Softmax

\[
\begin{align*}
    y_i^{(k)} &= \frac{\exp\left(z_i^{(k)}\right)}{\sum_j \exp\left(z_j^{(k)}\right)} \\
    \frac{\partial \text{Div}}{\partial z_i^{(k)}} &= \sum_j \frac{\partial \text{Div}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} \\
    \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} &= \begin{cases} 
    y_i^{(k)} \left(1 - y_i^{(k)}\right) & \text{if } i = j \\
    -y_i^{(k)} y_j^{(k)} & \text{if } i \neq j
    \end{cases} \\
    \frac{\partial \text{Div}}{\partial z_i^{(k)}} &= \sum_j \frac{\partial \text{Div}}{\partial y_j^{(k)}} y_i^{(k)} \left(\delta_{ij} - y_j^{(k)}\right)
\end{align*}
\]

• For future reference
• \(\delta_{ij}\) is the Kronecker delta: \(\delta_{ij} = 1\) if \(i = j\), \(0\) if \(i \neq j\)
Backward Pass for *softmax output layer*

- **Output layer** \((N)\):
  - For \(i = 1 \ldots D_N\)
    
    \[
    \frac{\partial \text{Div}}{\partial y_i^{(N)}} = \frac{\partial \text{Div}(y,d)}{\partial y_i}
    \]
    
    \[
    \frac{\partial \text{Div}}{\partial z_i^{(N)}} = \sum_j \frac{\partial \text{Div}(y,d)}{\partial y_j^{(N)}} y_i^{(N)} (\delta_{ij} - y_j^{(N)})
    \]
    
    \[
    \frac{\partial \text{Div}}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \text{Div}}{\partial z_j^{(N)}} \quad \text{for } j = 0 \ldots D_{N-1}
    \]

- **For layer** \(k = N - 1 \textbf{down to} 1\)
  - For \(i = 1 \ldots D_k\)
    
    \[
    \frac{\partial \text{Div}}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}
    \]
    
    \[
    \frac{\partial \text{Div}}{\partial z_i^{(k)}} = \frac{\partial \text{Div}}{\partial y_i^{(k)}} f'_k \left( z_i^{(k)} \right)
    \]
    
    \[
    \frac{\partial \text{Div}}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial \text{Div}}{\partial z_j^{(k)}} \quad \text{for } j = 0 \ldots D_{k-1}
    \]
Special cases

• Examples of vector activations and other special cases on slides
  – Please look up
  – Will appear in quiz!
• In reality the vector combinations can be anything
  – E.g. linear combinations, polynomials, logistic (softmax), etc.
Some types of networks have *multiplicative* combination

- In contrast to the *additive* combination we have seen so far

- Seen in networks such as LSTMs, GRUs, attention models, etc.
Backpropagation: Multiplicative Networks

Forward:
\[
o_i^{(k)} = y_j^{(k-1)} y_l^{(k-1)}
\]

Backward:
\[
\frac{\partial \text{Div}}{\partial o_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}
\]

\[
\frac{\partial \text{Div}}{\partial y_j^{(k-1)}} = \frac{\partial o_i^{(k)}}{\partial y_j^{(k-1)}} \frac{\partial \text{Div}}{\partial o_i^{(k)}} = y_l^{(k-1)} \frac{\partial \text{Div}}{\partial o_i^{(k)}}
\]

\[
\frac{\partial \text{Div}}{\partial y_l^{(k-1)}} = y_j^{(k-1)} \frac{\partial \text{Div}}{\partial o_i^{(k)}}
\]

• Some types of networks have multiplicative combination
Multiplicative combination as a case of vector activations

- A layer of multiplicative combination is a special case of vector activation

\[ Z_i^{(k)} = y_i^{(k-1)} \]

\[ y_i^{(k)} = z_{2i-1}^{(k)} z_{2i}^{(k)} \]
Multiplicative combination: Can be viewed as a case of vector activations

\[ z_i^{(k)} = \sum_j w_{ji}^{(k)} y_j^{(k-1)} \]

\[ y_i^{(k)} = \prod_l \left( z_l^{(k)} \right)^{\alpha_{li}^{(k)}} \]

\[ \frac{\partial y_i^{(k)}}{\partial z_j^{(k)}} = \alpha_{ji}^{(k)} \left( z_j^{(k)} \right)^{\alpha_{ji}^{(k)} - 1} \prod_{l \neq j} \left( z_l^{(k)} \right)^{\alpha_{li}^{(k)}} \]

\[ \frac{\partial \text{Div}}{\partial z_j^{(k)}} = \sum_i \frac{\partial \text{Div}}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_j^{(k)}} \]

- A layer of multiplicative combination is a special case of vector activation.
Gradients: Backward Computation

For $k = N\ldots1$
For $i = 1:\text{layer width}$

If layer has vector activation

$$\frac{\partial \text{Div}}{\partial z^{(k)}_i} = \sum_j \frac{\partial \text{Div}}{\partial y^{(k)}_j} \frac{\partial y^{(k)}_j}{\partial z^{(k)}_i}$$

Else if activation is scalar

$$\frac{\partial \text{Div}}{\partial w^{(k)}_{ij}} = y^{(k-1)}_i \frac{\partial \text{Div}}{\partial z^{(k)}_j}$$

$$\frac{\partial \text{Div}}{\partial y^{(k)}_i} = \frac{\partial \text{Div}}{\partial y^{(k)}_i} \frac{\partial y^{(k)}_i}{\partial z^{(k)}_i}$$

$$\frac{\partial \text{Div}}{\partial w^{(k)}_{ij}} = y^{(k-1)}_i \frac{\partial \text{Div}}{\partial z^{(k)}_j}$$
Special Case: Non-differentiable activations

- Activation functions are sometimes not actually differentiable
  - E.g. The RELU (Rectified Linear Unit)
    - And its variants: leaky RELU, randomized leaky RELU
  - E.g. The “max” function
- Must use “subgradients” where available
  - Or “secants”
The subgradient

• A subgradient of a function $f(x)$ at a point $x_0$ is any vector $v$ such that
  $$(f(x) - f(x_0)) \geq v^T (x - x_0)$$
  - Any direction such that moving in that direction increases the function

• Guaranteed to exist only for convex functions
  - “bowl” shaped functions
  - For non-convex functions, the equivalent concept is a “quasi-secant”

• The subgradient is a direction in which the function is guaranteed to increase

• If the function is differentiable at $x_0$, the subgradient is the gradient
  - The gradient is not always the subgradient though
The subderivative of a RELU is the slope of any line that lies entirely under it.

- The subgradient is a generalization of the subderivative.
- At the differentiable points on the curve, this is the same as the gradient.

Can use any subgradient at 0.

- Typically, will use the equation given.
Subgradients and the Max

\[ y = \max_j z_j \]

\[ \frac{\partial y}{\partial z_i} = \begin{cases} 
1, & i = \arg\max_j z_j \\
0, & \text{otherwise}
\end{cases} \]

- Vector equivalent of subgradient
  - 1 w.r.t. the largest incoming input
    - Incremental changes in this input will change the output
  - 0 for the rest
    - Incremental changes to these inputs will not change the output
Subgradients and the Max

- Multiple outputs, each selecting the max of a different subset of inputs
  - Will be seen in convolutional networks
- Gradient for any output:
  - $1$ for the specific component that is maximum in corresponding input subset
  - $0$ otherwise

$$y_i = \max_{l \in S_j} z_l$$

$$\frac{\partial y_j}{\partial z_i} = \begin{cases} 1, & i = \arg \max_{l \in S_j} z_l \\ 0, & \text{otherwise} \end{cases}$$
Backward Pass: Recap

- **Output layer (N):**
  - For $i = 1 \ldots D_N$
    - $\frac{\partial \text{Div}}{\partial y_i^{(N)}} = \frac{\partial \text{Div}(y,d)}{\partial y_i}$
    - $\frac{\partial \text{Div}}{\partial z_i^{(N)}} = \frac{\partial \text{Div}}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$ OR $\sum_j \frac{\partial \text{Div}}{\partial y_j^{(N)}} \frac{\partial y_j^{(N)}}{\partial z_i^{(N)}}$ (vector activation)

- For layer $k = N - 1$ **downto 1**
  - For $i = 1 \ldots D_k$
    - $\frac{\partial \text{Div}}{\partial y_i^{(k)}} = \sum_j w_{ji}^{(k+1)} \frac{\partial \text{Div}}{\partial z_j^{(k+1)}}$
    - $\frac{\partial \text{Div}}{\partial z_i^{(k)}} = \frac{\partial \text{Div}}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$ OR $\sum_j \frac{\partial \text{Div}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$ (vector activation)

    - $\frac{\partial \text{Div}}{\partial w_{ji}^{(k)}} = y_j^{(k-1)} \frac{\partial \text{Div}}{\partial z_i^{(k)}}$ for $j = 0 \ldots D_k$

These may be subgradients
Overall Approach

• For each data instance
  – **Forward pass**: Pass instance forward through the net. Store all intermediate outputs of all computation.
  – **Backward pass**: Sweep backward through the net, iteratively compute all derivatives w.r.t weights

• Actual loss is the sum of the divergence over all training instances
  \[
  \text{Loss} = \frac{1}{|\{X\}|} \sum_X \text{Div}(Y(X), d(X))
  \]

• Actual gradient is the sum or average of the derivatives computed for each training instance
  \[
  \nabla_W \text{Loss} = \frac{1}{|\{X\}|} \sum_X \nabla_W \text{Div}(Y(X), d(X))
  \]
  \[W \leftarrow W - \eta \nabla_W \text{Loss}^T\]
Training by BackProp

- Initialize weights $W^{(k)}$ for all layers $k = 1 \ldots K$
- Do: *(Gradient descent iterations)*
  - Initialize $Loss = 0$; For all $i, j, k$, initialize $\frac{dLoss}{dw^{(k)}_{i,j}} = 0$
  - For all $t = 1: T$ *(Iterate over training instances)*
    - **Forward pass**: Compute
      - Output $Y_t$
      - $Loss += Div(Y_t, d_t)$
    - **Backward pass**: For all $i, j, k$:
      - Compute $\frac{dDiv(Y_t,d_t)}{dw^{(k)}_{i,j}}$
      - $\frac{dLoss}{dw^{(k)}_{i,j}} + = \frac{dDiv(Y_t,d_t)}{dw^{(k)}_{i,j}}$
      - For all $i, j, k$, update:
        $$w^{(k)}_{i,j} = w^{(k)}_{i,j} - \frac{\eta}{T} \frac{dLoss}{dw^{(k)}_{i,j}}$$
  - Until $Loss$ has converged
Vector formulation

• For layered networks it is generally simpler to think of the process in terms of vector operations
  – Simpler arithmetic
  – Fast matrix libraries make operations much faster

• We can restate the entire process in vector terms
  – This is what is actually used in any real system
Vector formulation

- Arrange the inputs to neurons of the $k$th layer as a vector $\mathbf{z}_k$
- Arrange the outputs of neurons in the $k$th layer as a vector $\mathbf{y}_k$
- Arrange the weights to any layer as a matrix $\mathbf{W}_k$
  - Similarly with biases
Vector formulation

- The computation of a single layer is easily expressed in matrix notation as (setting $y_0 = x$):

$$z_k = W_k y_{k-1} + b_k \quad \quad y_k = f_k(z_k)$$
The forward pass: Evaluating the network

\[ y_0 = x \]
The forward pass

\[ z_1 = W_1 y_0 + b_1 \]
The forward pass

\[ y_1 = f_1(z_1) \]

The Complete computation

\[ y_1 = f_1(W_1 x + b_1) \]
The forward pass

\[ x \xrightarrow{W_1, b_1} z_1 \xrightarrow{y_1} W_2, b_2 \xrightarrow{z_2} \]

\[ z_2 = W_2y_1 + b_2 \]

The Complete computation

\[ y_1 = f_1(W_1x + b_1) \]
The forward pass

\[ x \rightarrow W_1, b_1 \rightarrow z_1 \rightarrow y_1 \rightarrow W_2, b_2 \rightarrow z_2 \rightarrow y_2 \]

\[ y_2 = f_2(z_2) \]

The Complete computation

\[ y_2 = f_2(W_2f_1(W_1x + b_1) + b_2) \]
The forward pass

The Complete computation

\[ z_N = W_N f_{N-1} \left( ... f_2 \left( W_2 f_1 \left( W_1 x + b_1 \right) + b_2 \right) ... \right) + b_N \]
The forward pass

The Complete computation

\[ Y = f_N(W_N f_{N-1}(...f_2(W_2 f_1(W_1 x + b_1) + b_2)...) + b_N) \]
Forward pass:

Initialize

\[ y_0 = x \]

For \( k = 1 \) to \( N \):

\[ z_k = W_k y_{k-1} + b_k \]

\[ y_k = f_k(z_k) \]

Output

\[ Y = y_N \]
The Forward Pass

• Set $y_0 = x$

• Iterate through layers:
  - For layer $k = 1$ to $N$:
    \[ z_k = W_k y_{k-1} + b_k \]
    \[ y_k = f_k(z_k) \]

• Output:
  \[ Y = y_N \]
The Backward Pass

• Have completed the forward pass
• Before presenting the backward pass, some more calculus...
  – Vector calculus this time
Vector Calculus Notes 1: Definitions

• A derivative is a multiplicative factor that multiplies a perturbation in the input to compute the corresponding perturbation of the output.

• For a scalar function of a vector argument:

\[ y = f(z) \]
\[ Δy = ∇_zy Δz \]

• If \( z \) is an \( R \times 1 \) vector, \( ∇_zy \) is a \( 1 \times R \) vector.
  – The shape of the derivative is the transpose of the shape of \( z \).

• \( ∇_zy^T \) is called the gradient of \( y \) w.r.t. \( z \).
Vector Calculus Notes 1: Definitions

• For a vector function of a vector argument

\[ y = f(z) \]

\[
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_M
\end{bmatrix}
= f
\begin{bmatrix}
z_1 \\
z_2 \\
\vdots \\
z_D
\end{bmatrix}
\]

\[ \Delta y = \nabla_z y \Delta z \]

• If \( z \) is an \( R \times 1 \) vector, and \( y \) is an \( L \times 1 \) \( \nabla_z y \) is an \( L \times R \) matrix
  – Or the dimensions won’t match

• \( \nabla_z y \) is called the Jacobian of \( y \) w.r.t \( z \)
Calculus Notes: The Jacobian

- The derivative of a vector function w.r.t. vector input is called a Jacobian.
- It is the matrix of partial derivatives given below.

\[
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_M \\
\end{bmatrix} = f \left( \begin{bmatrix}
z_1 \\
z_2 \\
\vdots \\
z_D \\
\end{bmatrix} \right)
\]

Using vector notation:

\[ y = f(z) \]

Jacobian:

\[
J_y(z) = \begin{bmatrix}
\frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \cdots & \frac{\partial y_1}{\partial z_D} \\
\frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \cdots & \frac{\partial y_2}{\partial z_D} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_M}{\partial z_1} & \frac{\partial y_M}{\partial z_2} & \cdots & \frac{\partial y_M}{\partial z_D}
\end{bmatrix}
\]

Check:

\[ \Delta y = J_y(z) \Delta z \]
Jacobians can describe the derivatives of neural activations w.r.t. their input.

\[ y_i = f(z_i) \]

\[
J_y(z) = \begin{bmatrix}
  f'(z_1) & 0 & \cdots & 0 \\
  0 & f'(z_2) & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & f'(z_M)
\end{bmatrix}
\]

- For scalar activations (shorthand notation):
  - Jacobian is a diagonal matrix
  - Diagonal entries are individual derivatives of outputs w.r.t. inputs
For Vector activations

- Jacobian is a full matrix
  - Entries are partial derivatives of individual outputs w.r.t individual inputs

\[
J_y(z) = \begin{bmatrix}
\frac{\partial y_1}{\partial z_1} & \frac{\partial y_1}{\partial z_2} & \ldots & \frac{\partial y_1}{\partial z_D} \\
\frac{\partial y_2}{\partial z_1} & \frac{\partial y_2}{\partial z_2} & \ldots & \frac{\partial y_2}{\partial z_D} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_M}{\partial z_1} & \frac{\partial y_M}{\partial z_2} & \ldots & \frac{\partial y_M}{\partial z_D}
\end{bmatrix}
\]
Special case: Affine functions

- Matrix $\mathbf{W}$ and bias $\mathbf{b}$ operating on vector $\mathbf{y}$ to produce vector $\mathbf{z}$
- The Jacobian of $\mathbf{z}$ w.r.t $\mathbf{y}$ is simply the matrix $\mathbf{W}$

$$z(\mathbf{y}) = \mathbf{W}\mathbf{y} + \mathbf{b}$$
$$\nabla_y z = J_z(\mathbf{y}) = \mathbf{W}$$
Vector Calculus Notes 2: Chain rule

• For nested functions we have the following chain rule

\[ y = y(z(x)) \]

\[ \nabla_x y = \nabla_z y \nabla_x z \]

Check

\[ \Delta y = \nabla_z y \Delta z \]

\[ \Delta z = \nabla_x z \Delta x \]

\[ \Delta y = \nabla_z y \nabla_x z \Delta x = \nabla_x y \Delta x \]

Note the order: The derivative of the outer function comes first
Vector Calculus Notes 2: Chain rule

• Chain rule for Jacobians:

• For vector functions of vector inputs:

\[ y = y(z(x)) \]

\[ J_y(x) = J_y(z)J_z(x) \]

Check

\[ \Delta y = J_y(z)\Delta z \]

\[ \Delta z = J_z(x)\Delta x \]

\[ \Delta y = J_y(z)J_z(x)\Delta x = J_y(x)\Delta x \]

Note the order: The derivative of the outer function comes first.
Vector Calculus Notes 2: Chain rule

- **Combining Jacobians and Gradients**

- **For scalar functions of vector inputs** ($z()$ is vector):

  \[ D = D(y(z)) \]
  \[ \nabla_z D = \nabla_y(D) J_y(z) \]

  **Check**

  \[ \Delta D = \nabla_y(D) \Delta y \]
  \[ \Delta y = J_y(z) \Delta z \]
  \[ \Delta D = \nabla_y(D) J_y(z) \Delta z = \nabla_z D \Delta z \]

  Note the order: The derivative of the outer function comes first
Vector Calculus Notes 2: Chain rule

• For nested functions we have the following chain rule

\[
D = D \left( y_N \left( z_N \left( y_{N-1} \left( z_{N-1} \left( \ldots y_1 \left( z_1(x) \right) \right) \right) \right) \right) \right)
\]

\[
\nabla_x D = \nabla_{y_N} D \nabla_{z_N} y_N \nabla_{y_{N-1}} z_N \nabla_{z_{N-1}} y_{N-1} \ldots \nabla_{z_1} y_1 \nabla_{x} z_1
\]

Note the order: The derivative of the outer function comes first
Vector Calculus Notes 2: Chain rule

• For nested functions we have the following chain rule

\[
D = D \left( y_N \left( z_N \left( y_{N-1} \left( z_{N-1} \left( \ldots y_1 \left( z_1(x) \right) \right) \right) \right) \right) \right)
\]

\[
\nabla_x D = \nabla_{y_N} D \nabla_{z_N} y_N \nabla_{y_{N-1}} z_N \nabla_{z_{N-1}} y_{N-1} \ldots \nabla_{z_1} y_1 \nabla_x z_1
\]

Note the order: The derivative of the outer function comes first
More calculus: Special Case

- Scalar functions of Affine functions

\[ z = W y + b \]
\[ D = f(z) \]

\[ \nabla_y D = \nabla_z(D) W \]
\[ \nabla_b D = \nabla_z(D) \]
\[ \nabla_W D = y \nabla_z(D) \]

- Note: the derivative shapes are the transpose of the shapes of \( W \) and \( b \)
More calculus: Special Case

• Scalar functions of Affine functions

\[ z = W y + b \quad D = f(z) \]

• Writing the transpose

\[ z^T = y^T W^T + b^T \]
\[ \nabla_{W^T} z^T = y^T \]
\[ \nabla_{W^T} D = \nabla_{z^T} D \]
\[ \nabla_{W^T} z^T = \nabla_{z^T} D \quad y^T \]
\[ \nabla_W D = (\nabla_{W^T} D)^T = y \nabla_z D \]
\[ \nabla_W D = y \nabla_z (D) \]
Special Case: Application to a network

- Scalar functions of Affine functions

\[ z = W y + b \]

\[ \text{Div} = \text{Div}(z) \]  

\[ \nabla_y \text{Div} = \nabla_z \text{Div}W \]

\[ z_k = W_k y_{k-1} + b_k \]

The divergence is a scalar function of \( z_k \).

Applying the above rule

\[ \nabla_{y_{k-1}} \text{Div} = \nabla_{z_k} \text{Div}W_k \]
Special Case: Application to a network

- Scalar functions of Affine functions

\[ z = Wy + b \]

\[ \text{Div} = \text{Div}(z) \]

\[ \nabla_b \text{Div} = \nabla_z \text{Div} \]

\[ \nabla_w \text{Div} = y \nabla_z \text{Div} \]

\[ z_k = W_k y_{k-1} + b_k \]

\[ \nabla_{b_k} \text{Div} = \nabla_{z_k} \text{Div} \]

\[ \nabla_{w_k} \text{D} = y_{k-1} \nabla_{z_k} \text{Div} \]
The backward pass

- The network is a nested function

\[ Y = f_N(W_N f_{N-1}(... f_2(W_2 f_1(W_1 x + b_1) + b_2)...) + b_N) \]

- The divergence for any \( x \) is also a nested function

\[ \text{Div}(Y, d) = \text{Div}(f_N(W_N f_{N-1}(... f_2(W_2 f_1(W_1 x + b_1) + b_2)...) + b_N), d) \]
The backward pass

In the following slides we will also be using the notation $\nabla_z Y$ to represent the derivative of any $Y$ w.r.t any $z$. 
First compute the derivative of the divergence w.r.t. $Y$. The actual derivative depends on the divergence function.

N.B: The gradient is the transpose of the derivative.
The backward pass

The divergence is a nested function: $\text{Div}(Y(z_N))$

$$\nabla_{z_N} \text{Div} = \nabla_Y \text{Div} \cdot \nabla_{z_N} Y$$

Already computed  New term
The backward pass

\[ \nabla_{z_N} Div = \nabla_Y Div J_Y(z_N) \]

Already computed  New term
The backward pass

The divergence is a nested function: \( \nabla_{y_{N-1}} Div = \nabla_{z_N} Div \cdot \nabla_{y_{N-1}} z_N \)

\( z_N = W_N y_{N-1} + b_N \quad \Rightarrow \quad \nabla_{y_{N-1}} z_N = W_N \)
The backward pass

\[ \nabla_y_{N-1} Div = \nabla_{z_N} Div \cdot W_N \]

Already computed  New term
The backward pass

\[ \nabla_{y_{N-1}} \text{Div} = \nabla_{z_N} \text{Div} \, W_N \]

\[ z = W_y + b \]
\[ \text{Div} = \text{Div}(z) \]

\[ \nabla_b \text{Div} = \nabla_z \text{Div} \]
\[ \nabla_W \text{Div} = y \nabla_z \text{Div} \]

\[ \nabla_{y_{N-1}} \text{Div} \]
\[ \nabla_W \text{Div} = y_{N-1} \nabla_{z_N} \text{Div} \]
\[ \nabla_{b_N} \text{Div} = \nabla_{z_N} \text{Div} \]
The backward pass

\[ \nabla_{z_{N-1}} \text{Div} = \nabla_{y_{N-1}} \text{Div} \cdot \nabla_{z_{N-1}} y_{N-1} \]

Already computed  New term
The Jacobian will be a diagonal matrix for scalar activations.

\[ \nabla_{z_{N-1}} \text{Div} = \nabla_{y_{N-1}} \text{Div} J_{y_{N-1}}(z_{N-1}) \]

The Jacobian will be a diagonal matrix for scalar activations.
The backward pass

\[ \nabla_{y_{N-2}} Div = \nabla_{z_{N-1}} Div \cdot \nabla_{y_{N-2}} z_{N-1} \]
The backward pass

\[ \nabla_{y_{N-2}} \text{Div} = \nabla_{z_{N-1}} \text{Div} \ W_{N-1} \]
The backward pass

\[ \nabla y_{N-2} \text{Div} = \nabla z_{N-1} \text{Div} \cdot W_{N-1} \]

\[ \nabla w_{N-1} \text{Div} = y_{N-2} \nabla z_{N-1} \text{Div} \]

\[ \nabla b_{N-1} \text{Div} = \nabla z_{N-1} \text{Div} \]
The backward pass

\[ \nabla_{z_1} Div = \nabla_{y_1} Div J_{y_1}(z_1) \]
The backward pass

In some problems we will also want to compute the derivative w.r.t. the input

\[ \nabla_{w_1} \text{Div} = x \nabla_{z_1} \text{Div} \]

\[ \nabla_{b_1} \text{Div} = \nabla_{z_1} \text{Div} \]
The Backward Pass

• Set \( y_N = Y, \ y_0 = x \)
• Initialize: Compute \( \nabla_{y_N} Div = \nabla_Y Div \)

• For layer \( k = N \) downto 1:
  – Compute \( J_{y_k}(z_k) \)
    • Will require intermediate values computed in the forward pass
  – Backward recursion step:
    \[
    \nabla_{z_k} Div = \nabla_{y_k} Div J_{y_k}(z_k) \\
    \nabla_{y_{k-1}} Div = \nabla_{z_k} Div W_k
    \]
  – Gradient computation:
    \[
    \nabla_{W_k} Div = y_{k-1} \nabla_{z_k} Div \\
    \nabla_{b_k} Div = \nabla_{z_k} Div
    \]
The Backward Pass

- Set $y_N = Y$, $y_0 = x$
- Initialize: Compute $\nabla_{y_N} Div = \nabla_Y Div$

- For layer $k = N$ downto 1:
  - Compute $J_{y_k}(z_k)$
    - Will require intermediate values computed in the forward pass
  - Backward recursion step:
    $$\nabla_{z_k} Div = \nabla_{y_k} Div \, J_{y_k}(z_k)$$
    $$\nabla_{y_{k-1}} Div = \nabla_{z_k} Div \, W_k$$
  - Gradient computation:
    $$\nabla_{W_k} Div = y_{k-1} \nabla_{z_k} Div$$
    $$\nabla_{b_k} Div = \nabla_{z_k} Div$$
For comparison: The Forward Pass

• Set $y_0 = x$

• For layer $k = 1$ to $N$
  
  – Forward recursion step:

  $$z_k = W_k y_{k-1} + b_k$$
  $$y_k = f_k(z_k)$$

• Output:

  $$Y = y_N$$
Neural network training algorithm

- Initialize all weights and biases \((W_1, b_1, W_2, b_2, \ldots, W_N, b_N)\)
- Do:
  - \(Loss = 0\)
  - For all \(k\), initialize \(\nabla_{W_k} Loss = 0, \nabla_{b_k} Loss = 0\)
  - For all \(t = 1:T\)  
    # Loop through training instances
    - Forward pass: Compute
      - Output \(Y(X_t)\)
      - Divergence \(\text{Div}(Y_t, d_t)\)
      - \(Loss += \text{Div}(Y_t, d_t)\)
    - Backward pass: For all \(k\) compute:
      - \(\nabla_{y_k} \text{Div} = \nabla_{z_{k+1}} \text{Div} W_{k+1}\)
      - \(\nabla_{z_k} \text{Div} = \nabla_{y_k} \text{Div} J_k(z_k)\)
      - \(\nabla_{W_k} \text{Div}(Y_t, d_t) = y_{k-1} \nabla_{z_k} \text{Div}; \nabla_{b_k} \text{Div}(Y_t, d_t) = \nabla_{z_k} \text{Div}\)
      - \(\nabla_{W_k} \text{Loss} += \nabla_{W_k} \text{Div}(Y_t, d_t); \nabla_{b_k} \text{Loss} += \nabla_{b_k} \text{Div}(Y_t, d_t)\)
    - For all \(k\), update:
      \[W_k = W_k - \eta \frac{T}{T} \left(\nabla_{W_k} \text{Loss}\right)^T; \quad b_k = b_k - \eta \frac{T}{T} \left(\nabla_{W_k} \text{Loss}\right)^T\]
- Until \(Loss\) has converged
Setting up for digit recognition

- Simple Problem: Recognizing “2” or “not 2”
- Single output with sigmoid activation
  - $Y \in (0,1)$
  - $d$ is either 0 or 1
- Use KL divergence
- Backpropagation to compute derivatives
  - To apply in gradient descent to learn network parameters
Recognizing the digit

Training data

(5, 5) (2, 2)
(2, 2) (4, 4)
(0, 0) (2, 2)

• More complex problem: Recognizing digit
• Network with 10 (or 11) outputs
  – First ten outputs correspond to the ten digits
    • Optional 11th is for none of the above
• Softmax output layer:
  – Ideal output: One of the outputs goes to 1, the others go to 0
• Backpropagation with KL divergence
  – To compute derivatives for gradient descent updates to learn network
Neural networks must be trained to minimize the average divergence between the output of the network and the desired output over a set of training instances, with respect to network parameters.

Minimization is performed using gradient descent.

Gradients (derivatives) of the divergence (for any individual instance) w.r.t. network parameters can be computed using backpropagation.

- Which requires a “forward” pass of inference followed by a “backward” pass of gradient computation.

The computed gradients can be incorporated into gradient descent.
Issues

• Convergence: How well does it learn
  – And how can we improve it
• How well will it generalize (outside training data)
• What does the output really mean?
• Etc..
Next up

• Convergence and generalization