Deep Nets

Professor Marie Roch

Unit – Basic building block

Loosely inspired model of a neuron

Fig. 18.19 Russell and Norvig
Feed forward network

\[ f : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \]

Example

• Learn exclusive or: \( x^{(1)} \oplus x^{(2)} = y \)
• No need to generalize, only 4 possible inputs
• Select neuron model:
  \[ f(x|w, b) = x^T w + b \]
• and loss/cost function
  \[ \text{MSE } J(\theta) = \frac{1}{4} \sum_x (f^*(x) - f(x|w, b))^2 \]
Learning xor

- We have a linear model and can use regression to fit it.
- Regression model:
  - \( w = 0 \)
  - \( b = 0.5 \)
  - Predicts line \( y = 0.5 \)

**xor, take 2**

- Linear function won’t work
- Try a feed forward network
**xor**

- Alternate representation of network
- Each node is a vector

\[
y = f^{(2)}(h | w, b) \\
= f^{(2)}(f^{(1)}(x | W, c) | w, b) \\
h = f^{(1)}(x | W, c)
\]

Goodfellow et al. Fig. 6.2

**Unit definition**

- Linear function, \( w^T x \)
- Nonlinear transformation of linear function
  Default is rectified linear unit (ReLU):
  \( g(z) = \max(0, z) \)
- Using the compact vector notation, our fn is:
  \[
f(x | W, c, w, b) = w^T \max(0, W^T x + c) + b
\]
Solution to xor

\[ f(x | W, c, w, b) = w^T \max(0, W^T x + c) + b \]

\[ W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = 0 \]

We’ll talk about how to get this solution later…

Data matrix

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
x & y
\end{bmatrix}
\]

\[ W^T x = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} x \]

applied row by row

\[
\begin{bmatrix}
0 & -1 \\
1 & 0 \\
1 & 0 \\
2 & 1
\end{bmatrix}
\]

adding bias \( c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \) yields

\[
\begin{bmatrix}
0 & 0 \\
1 & 1 \\
1 & 0 \\
2 & 1
\end{bmatrix}
\]

ReLU breaks linearity

Remember, \( w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = 0 \)
Optimization

The linear problems we have seen are examples of *convex optimization*.

In theory, convex optimization always converges.

Nonlinear functions in neural nets usually lead to non-convex optimization functions.

Gradient-based learning

• Sensitive to initial parameters, typically use
  – small random values for weights
  – biases zero or small +values

• Convex loss functions can also be trained with gradient descent; useful for large data sets.
Gradient-based learning

• Be careful of saturation
  – Sometimes, loss functions locally flat.
  – Problematic for gradient descent

Example for Bernoulli (binary) output:

\[
P(y = 1 \mid x) = \max \left(0, \min(1, w^T h + b)\right)
\]

Learning conditional distributions

• Cross entropy cost function as previously seen:

\[
J(\theta) = E_{x,y \sim p_{data}} \left[ -\log P(y \mid x, \theta) \right]
\]

• Form depends on distribution. Our normal distribution model reduced to

\[
J(\theta) = E_{x,y \sim p_{data}} \left[ -\log P(y \mid x, \theta) \right] = \frac{1}{m} \sum_{i=1}^{m} -\log P(y \mid x, \theta) + k
\]

where \(k\) was a constant that could be ignored.
Learning conditional statistics

• Sometimes we want a statistic, e.g.
  
  – \( E[Y \mid X = x] \)
  
  – \( E[(Y - \bar{Y})^2 \mid X = x] \)

• Cost function lets learning select a function

\[
\begin{align*}
E_Y X x \quad E_Y y X x \\
\end{align*}
\]

Learning conditional statistics

• Details require variational calculus
• Example fns
  
  – conditional mean
    
    \[ f^* = \arg \min_f \mathbb{E}_{x,y \sim \text{data}} \|y - f(x)\|^2 \quad \text{mean squared error} \]

  – conditional median
    
    \[ f^* = \arg \min_f \mathbb{E}_{x,y \sim \text{data}} \|y - f(x)\| \quad \text{mean absolute error} \]
Note on cost functions

• Avoid cost functions that saturate
• When saturated (flat), gradient is small
• Common to use negative log likelihoods
• Examples of saturating functions
  – $e^{-k}$ where $k$ is large
  – Mean squared error
  – Mean absolute error

Output units

What type of outputs?
• Conditional probabilities
• Two class – output 0|1
• Multiple class – N outputs, one high

Most of what we discuss can also be applied to hidden units
Conditional outputs

- Produce mean of conditional normal
  \[ P(y \mid x) = n(y \mid \hat{y}, I) \]
  - mean targets
  - observed training y
  - variance can be learned too
  - be careful, must be + definite

- \( \hat{y} \) produced as linear output, does not saturate
  \[ \hat{y} = W^T h + b \]

- Can be fed into normal pdf if \( P(y \mid x) \) desired

Bernoullli output

- Predicts \( P(y=1 \mid x) \)
- Common to use sigmoid activation function

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
Bernoulli output

- Straight forward:
  \[ \hat{y} = \sigma(w^T h + b) \text{ or } z = w^T h + b, \hat{y} = \sigma(z) \]
- Problem:
  We want to model \( P(y=1|x) \)

\[ \log \tilde{P}(y) = yz \]

implies:
\[
\log \tilde{P}(y) = \begin{cases} 
  z & y = 1 \\
  0 & y = 0 
\end{cases}
\]

- Without log,
  \[
  \tilde{P}(y) = \begin{cases} 
  e^z & y = 1 \\
  e^0 & y = 0 
\end{cases}
\]

Does this look like a probability distribution?
Statistician’s bag of tricks

Solution:
• Normalize by \( P(y=0|x) + P(y=1|x) \)

\[
P(y) = \begin{cases} 
\frac{e^z}{1 + e^z} & y = 1 \\
\frac{1}{1 + e^z} & y = 0 
\end{cases}
\]

• Can be simplified: \( P(y) = \sigma((2y-1)z) \)

Bernoulli output

• With MLE approach, we optimize \( -\log P(y|x) \). Solves the saturation problem

• Sigmoid activation less common with non MLE approaches
Bernoulli output

- Loss function with sigmoid
  \[ J(\theta) = -\log P(y|x) \]
  \[ = -\log \sigma((2y-1)z) \]
- Sometimes written in terms of softplus
  \[ \zeta(x) = \log(1 + e^x) \quad \text{Goodfellow et al. eq. (3.31)} \]
  \[ J(\theta) = -\log \left( \sigma((2y-1)z) \right) \]
  \[ = -\log \left( \frac{1}{1 + e^{-2y+1}z} \right) = -\log \left( 1 + e^{(2y-1)z} \right)^{-1} \]
  \[ = \log \left( 1 + e^{(1-2y)z} \right) = \zeta((1-2y)z) \]

Categorical outputs

- Classification output, e.g. digit label
- Output node represents each category
- Can model as a multinoulli distribution
Multinoulli distributions

- Appropriate for k categories
- Parameter vector p
  \[
  \text{length}(p) = k - 1, 1^T p \leq 1
  \]
  \[
  P(x = i) = \begin{cases} 
  p_i & 0 \leq i < k \\
  1 - 1^T p & i = k 
  \end{cases}
  \]
- Rarely worry about mean & variance as these do not mean much for categories

Categorical outputs

- Output layer is vector \( \hat{y} \)
  \[
  \hat{y}_i = P(y = i \mid x)
  \]
- Requirements
  \[
  \hat{y}_i \in [0,1]
  \]
  \[
  1^T y = 1
  \]
- Use a similar approach as we used for Bernoulli distributions
Categorical outputs

- Input function as usual: \( z = W^T h + b \)
- Activation function is softmax:
  \[
  \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}
  \]
- Following the MLE approach, use log-likelihood to avoid saturation
  \[
  \log \text{softmax}(z_i) = z_i - \log \left( \sum_j e^{z_j} \right)
  \]

Categorical outputs

- Softmax
  - When difference in \( z_i \)'s large, drives towards a “one hot” solution
  - Numerical stability improved if we normalize:
    \[
    \text{softmax}(z - \max_i z_i)
    \]
    equivalent as
    \[
    \frac{e^{z_i}}{\sum_j e^{z_j}} = \frac{e^{z_i}e^{c}}{e^{\max_j e^{z_j} + c}} = \frac{e^{z_i + c}}{\sum_j e^{z_j + c}} = \text{softmax}(z_i + c)
    \]
Categorical outputs

- Input layer to softmax can use identity function for activation and feed outputs into the softmax layer.

Other output types exist, but we will not cover them in detail for now.

Other output types

- Learning distributional parameters
- Learning multimodal distributions
Hidden units

- Output layer activation functions:
  - linear
  - sigmoid
  - softmax
  can be used in hidden units
- Sigmoid functions are commonly used, but there are other possibilities

Hidden units

- Rare to have good theoretical justification for activation function type
- A number of the activation functions have near-linear properties, making optimization easier.
Rectified linear units

- \( g(z) = \max(0, z) \)
- Effectively shuts off unit for negative inputs
- Saturation could be a problem… in practice, rarely so due to local minima
- Not differentiable at 0, implementations usually pick the left or right derivative

Rectified linear units

- Variants exist
  \[ g(z_i, \alpha_i) = \max(0, z_i) + \min \alpha_i(0, z_i) \]
- Differ mainly in how \( \alpha_i \) is set.
- Example
  Absolute value rectification \( \alpha_i = 1 \)
  Used to compensate for reverse polarization illumination
Maxout units

- Separate z’s into $k$ groups
- Propagates maximum z of each group to next layer
  $$g_i(z) = \max_{j \in \text{group}(i)} z_j$$
- Learns piecewise linear function
- Reduces number of parameters to next layer
- Can help protect network against forgetting things it has learned

Logistic sigmoid & hyperbolic tangent

- Remember: sigmoid
  $$\sigma(z) = \frac{1}{1 + e^{-z}}$$
- tanh is similar
  - tanh(0)=0 more like identity
  - Relation: $\tanh(z) = 2\sigma(2z) - 1$
- Saturation a problem, avoid using in hidden layers if at all possible!
Other activation functions

- Goodfellow et al. discuss a couple other units, some of which are discouraged
- Many other activation functions will work
- So many so, that finding a new one isn’t a big deal unless it fundamentally changes what you can solve.

Architecture

- Organized into layers
- Depth vs width
  - Deeper networks can have fewer parameters
  - Wide networks easier to optimize
Universal approximation

A network with at least
• One hidden layer and
• A nonlinear activation (e.g. $\sigma$, ReLU)
can model any continuous $f$ on a bounded subset of a finite dimensional space

It does not state that we will find the network that models the function, merely that it exists.

Backpropagation

• Method for computing gradients in a computation network.
• Usually interested in gradients of the cost function with respect to model parameters.
• Only part of the training algorithm… Learning algorithms exploit the gradient to optimize the model.
Backpropagation overview

• After applying the update to the output layer, there still exists loss.
• We assign a portion of the loss to each of the input nodes based on their weight.
• This contribution is computed for each node of the current layer.

\[
\sum_i w_i \text{ Loss}_2
\]

Backpropagation overview

• Now we can look at the sum of losses attributable to each node in the previous layer.
• The sum of these provides us with a loss to minimize.
• Repeat recursively.

\[
\sum_i w_{l,i} \text{ Loss}_j
\]
A more formal view

Computation represented by a simple graph language

- Nodes: Variables (e.g. scalar, tensor)
- Directed edges: Simple functions that transform variables.
  - Tie edges together for multiple operands
  - Annotated with operation name unless obvious

Sample graphs

\[ z = xy \]
\[ \hat{y} = \sigma(w^T x + b) \]

\[ H = \text{relu}(XW + B) \]
\[ \hat{y} = x^T w, \]
\[ u^{(3)} = \lambda \sum_i w_i^2 \]
Forward graph & computation

Graph $G$ arranged so that $u^{(i)}$ computed before $u^{(j)}$ when $i<j$.
Pa(node) = predecessor indices

for $i = 1, 2, \ldots, n_i$
$u^{(1)} = x_i$
for $i = n_{i+1}, \ldots, n$
$A(i) \subseteq \{u^{(j)} \mid j \in \text{Pa}(u^{(1)})\}$
$u^{(1)} \leftarrow f^{(1)}(A(i))$
return $u^{(n)}$

Calculus chain rule

Recall the chain rule
\[
\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}
\]
generalize to the following functions
\[
g : \mathbb{R}^m \rightarrow \mathbb{R}^n, f : \mathbb{R}^m \rightarrow \mathbb{R}^1
\]
and let $y = g(x)$, $z = f(y)$, then
\[
\frac{\partial z}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}
\]
Chain rule

Remember \( x \in \mathbb{R}^m \) and \( y \in \mathbb{R}^n \)

Let's put this in matrix notation

Construct a Jacobian matrix

\[
\begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_M} \\
\frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_M} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_N}{\partial x_1} & \frac{\partial y_N}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_M}
\end{bmatrix}
\]

We can apply this to tensors by extending the dimensions

\[
\nabla_y z = \begin{bmatrix}
\frac{\partial z}{\partial y_1} \\
\frac{\partial z}{\partial y_2} \\
\vdots \\
\frac{\partial z}{\partial y_N}
\end{bmatrix}
\]
Backpropagation

- Enhance computation graph $G$ with new subgraph $B$
- One-to-one correspondence with $u^{(i)}$ nodes

Concrete example

Example based on Christopher Olah’s blog
Activation fn
example for backprop

partial derivatives

\[ \frac{\partial L}{\partial u_{out}} = \frac{2}{2} (y - u_{out})(-1) = u_{out} - y \]

\[ \frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in})) \]

\[ \frac{\partial u_{in}}{\partial u_{x_1}} = w_1 3x_1^2 \]

To update \( w_1 \) we use the chain rule:

\[ \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{x_1}} = (u_{out} - y) \sigma(u_{in})(1 - \sigma(u_{in}))x_1^3 \]

from previous slide

\[ \frac{\partial L}{\partial u_{out}} = u_{out} - y \]

\[ \frac{\partial u_{out}}{\partial u_{u}} = \sigma(u_{in})(1 - \sigma(u_{in})) \]

\[ \frac{\partial u_{in}}{\partial u_{x_1}} = x_1^3 \]
Activation fn example for backprop

\[ \frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial u_w} \frac{\partial u_w}{\partial u_u} \frac{\partial u_u}{\partial w_i} = (u_w - y) \sigma(u_u)(1 - \sigma(u_u))x_i^t \]

Concrete example

\[ y = 0, w = \begin{bmatrix} .02 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \]

implies

\[ u_w = w_1x^t + w_2x^t = .02 \cdot 3^t + .01 \cdot 5 = .59 \]

\[ u_{uw} = \frac{1}{1 + e^{-u_w}} = \frac{1}{1 + e^{-0.59}} = .6434 \]

\[ L = \frac{1}{2} (y - u_{uw})^2 = .5 \cdot (0 - .6434)^2 = .2070 \]

Suppose we have a learning rate \( \varepsilon = .01 \)

\[ w_i = w_i - \varepsilon \frac{\partial L}{\partial w_i} = .02 - .01 \cdot .39851 = -.0599 \]

Update of \( w_2 \) is left as an exercise, but loss with only \( w_1 \) changed:

\[ y = 0, w = \begin{bmatrix} -.06 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix} \]

implies

\[ u_u = w_1x^t + w_2x^t = -.06 \cdot 3^t + .01 \cdot 5 = -1.57 \]

\[ u_{uw} = \frac{1}{1 + e^{-u_u}} = \frac{1}{1 + e^{-1.57}} = .1722 \]

\[ L = \frac{1}{2} (y - u_{uw})^2 = .5 \cdot (-1.722)^2 = .0148 \]
Activation fn example for backprop

Example did now show previous layers, but they can be handled in the same way. Suppose we wanted to adapt weight vector $z$ leading into $h_1$:

$$
\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{h_{out}}} \frac{\partial u_{h_{out}}}{\partial u_{h_{in}}} \frac{\partial u_{h_{in}}}{\partial u_{h_{i,i}}}
$$

Implementing derivative computation

Two possible implementations memory vs time trade-off

$$
\frac{\partial z}{\partial w} = f'(y)f'(x)f'(w) \quad \text{(better)} \quad \text{or} \quad \frac{\partial z}{\partial w} = f'(f(f(w)))f'(f(w))f'(w)
$$
Simplified backprop w/scalars

\( \nabla [u^{(n)}] = 1 \)

for \( j = n-1, \ldots, 2, 1 \)

\[ \nabla [u^{(j)}] = \sum_{i:j \in Pa(u^{(i)})} \nabla [u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}} \]

return gradient table \( \nabla \)

We use activations from minibatch to compute concrete values.

Recall

\[ \frac{\partial u^{(n)}}{\partial u^{(i)}} = \sum_{j:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}} \]

\[ \nabla [u^{(i)}] = \frac{\partial u^{(n)}}{\partial u^{(i)}} \]

59