IDC410
A course on Image Processing and Machine Learning
(Lecture 12)

Shashikant Dugad,

IISER Mohali

Reading Material

Suggested Books:

1. **Neural Networks and Deep Learning by Michael Nielsen**
2. **Fundamentals of Deep Learning by Nikhil Buduma**

Source for this presentation:

Neural Networks and Deep Learning by Michael Nielsen

https://www.scaler.com/topics/deep-learning/introduction-to-feed-forward-neural-network/

https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network

http://machine-learning-for-physicists.org. by Florian Marquardt

3Blue1Brown (Youtube Videos)

https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Network Training

Backpropagation: Regularization → Overfitting

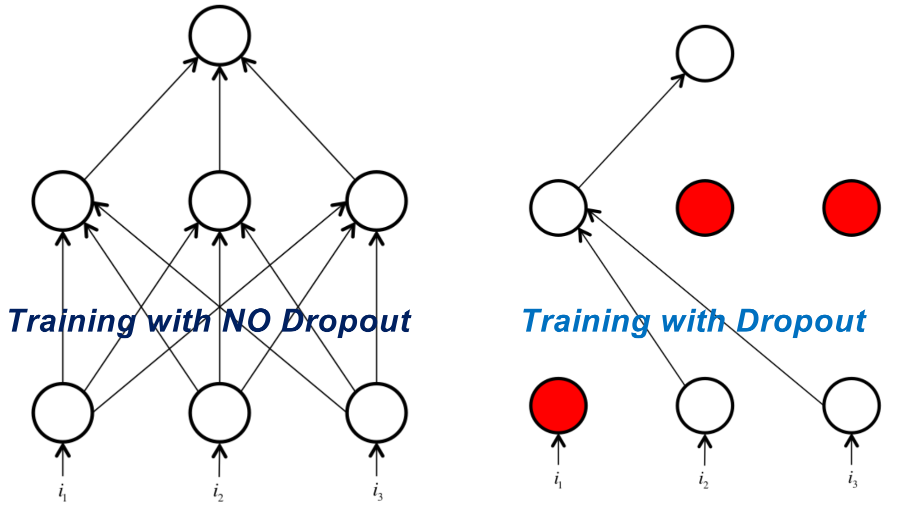
- Regularization is yet another technique to overcome overfitting. It modifies the error function that we minimize by adding additional terms that penalizes abnormally large weights
- The error function is modified as $E(\theta) \rightarrow E(\theta) + \lambda f(\theta)$, where $f(\theta)$ grows larger as the components of θ grows. θ essentially represents the weight matrix. λ is the regularization strength (another hyperparameter). The value we choose for λ determines how much we want to protect against overfitting. ($\lambda=0$ implies no measures to protect against the overfitting using this technique)
- **L2 regularization:** The most common type of regularization in machine learning is ***L2 regularization*** in which $f(\theta) = 1/2\ \lambda w^2$ (w^2 represents squared magnitude of all weights). The ***L2 regularization*** penalizes peaky weight vectors and prefers diffuse weight vectors resulting in decay of large weights. Because of this phenomenon, ***L2 regularization*** is also commonly referred to as weight decay.
- **L1 regularization:** In this method the $f(\theta) = \lambda|w|$. This technique leads the weight vectors to become sparse during optimization (i.e., very close to exactly zero). It ends up using only a small subset of their most important inputs and become quite resistant to noise in the inputs

Backpropagation: Overfitting → Regularization

- ***L1 regularization*** is useful when you want to understand exactly which features are contributing to a decision. If this level of feature analysis isn't necessary, we prefer to use ***L2 regularization*** because it empirically performs better.
- **Maximum normalisation constraints:** It has a similar goal of attempting to restrict θ from becoming too large. Max norm constraints enforce an absolute upper bound on the magnitude of the incoming weight vector for every neuron and use projected gradient descent to enforce the constraint.
- If gradient descent step moves the incoming weight vector such that $w^2 > c$, we project the vector back onto the ball (centered at the origin) with ***radius c***.
- Typical values of ***c are 3 and 4***.
- **Advantage:** The parameter vector do not grow out of control (even if the learning rates are too high)

Backpropagation: Dropout → Overfitting

- **Dropout:** It is one of the most favoured methods of preventing overfitting in deep neural networks. While training, dropout is implemented by only keeping a neuron active with some **probability p** (a hyperparameter), or setting it to zero otherwise.
- This trains the network to be accurate even in the absence of certain information and prevents the network from becoming too dependent on any one (or any small combination) of the neurons.
- Providing a way of approximately combining exponentially many different neural network architectures helps in inhibiting overfitting.
- Dropout method has certain intricacies. The outputs of neurons during test time has to be equivalent to their expected outputs at training time.



Backpropagation: Overfitting → Dropout

- Dropout is pretty intuitive to understand, but there are some important intricacies to consider. First, we'd like the outputs of neurons during test time to be equivalent to their expected outputs at training time.
- This means that if a neuron's output prior to dropout was x, then after dropout, the expected output would be $E(\text{output}) = px + (1 - p)x0 = px$. This is undesirable as it requires scaling of neuron outputs during the test time.
- It is preferable to use inverted dropout, where the ***scaling occurs at the training time instead of at test time***.
- In inverted dropout, any neuron whose activation hasn't been silenced has its ***output divided by p*** (Note: $1/p > 1$) before its output value is propagated to the next layer.
- With this normalisation fix during training, the E(output) do not change due to ***dropout*** and therefore, we can avoid arbitrary scaling of neuronal output at the test time.

Backpropagation: Alternate Approach

Source:

https://medium.com/towards-data-science/deriving-backpropagation-with-cross-entropy-loss-d24811edeaf9

https://medium.com/towards-data-science/backpropagation-the-natural-proof-946c5abf63b1

Notations

First we start with various notations that are somewhat different than earlier description

1. i is event number, y_i is expected output (truth), $y(x_i)$ is predicted output and d is total # of events

2. $y(x_i)$: y is an output of activation function of the output layer applied on x_i for an event i.

3. J(x,y) is a quadratic error function defined as:

$$J = \frac{1}{2d}\sum_i ||y_i - y(x_i)||^2$$

4. For just one event: $J_{(x,y)} = \frac{1}{2}||y - y(x)||^2$

Elementwise Notations

1. L is layer number. n and m are neuron number in Layer L and L-1 respectively
2. w_{mn}^{L} is weight of connection between mth neuron in layer L-1 and nth neuron in layer L
3. b_n^L is bias of nth neuron at layer L
4. z_n^L is weighted sum at neuron n in layer L plus the bias

$$z_n^L = \sum_{m=1}^{M} w_{mn}^{L} a_m^{L-1} + b_n^L$$

5. a_n^L is output of activation function *h* applied on z_n^L
6. h is an activation function: $a_n^L = h(z_n^L)$

Matrix Notations

1. **W^L**: The weight matrix involving weights from the connection between **(L-1)$_{th}$** layer to the L$_{th}$ layer. If the **(L-1)$_{th}$** layer has **M** neurons and the L$_{th}$ layer has **N** neurons then this has dimensions ***M*N***. Like you would imagine, the element **W^L[m,n]** will simply be w^L_{mn} that we defined earlier.

2. **b^L**: The bias vector for any layer would have the bias of each of its neurons, $b^L[n] = b^L_n$

3. **a^L**: The activations vector for any layer ***L*** would have the activations of each of its neurons, $a^L[n] = a^L_n$

4. **z^L**: The pre-activations vector for any layer would have the linear combinations of each of its neurons, $z^L[n] = z^L_n$

Using this new notation, if we let the function h be applied on vectors element-wise then we can write:

$$z^L = (W^L)^t a^{L-1} + b^L \text{ and thus } a^L = h(z^L)$$

Gradients of Weights and Biases

- In order to obtain revised weights and biases, we must evaluate the gradients of the same i.e. gradients of weights $\left(\frac{\partial J}{\partial w_{mn}^{L}}\right)$ and biases $\left(\frac{\partial J}{\partial b_{n}^{L}}\right)$
- Let us first derive the gradient w.r.t. weights, i.e., $\frac{\partial J}{\partial w_{mn}^{L}}$:

Step 1:

$$\frac{\partial J}{\partial w_{mn}^{l}} = \frac{\partial J}{\partial z_{n}^{l}} \cdot \frac{\partial z_{n}^{l}}{\partial w_{mn}^{l}}$$

Step 2:

$$\frac{\partial J}{\partial w_{mn}^{L}} = \frac{\partial J}{\partial z_{n}^{L}} \cdot \frac{\partial \sum_i w_{in}^{L} * a_{i}^{L-1} + b_{n}^{L}}{\partial w_{mn}^{L}} = \frac{\partial J}{\partial z_{n}^{L}} \cdot \frac{\partial \sum_i w_{in}^{L} * a_{i}^{L-1}}{\partial w_{mn}^{L}}$$

Step 3:

$$\frac{\partial J}{\partial w_{mn}^{L}} = \underbrace{\frac{\partial J}{\partial z_{n}^{L}}}_{\delta_n^L} \cdot \frac{\partial \left(w_{mn}^{L} * a_{m}^{L-1}\right)}{\partial w_{mn}^{L}} = \frac{\partial J}{\partial z_{n}^{L}} \cdot a_{m}^{L-1}$$

Gradients of Weights and Biases

- Let us denote:

  Step 4: $\delta_n^L = \frac{\partial J}{\partial z_n^L}$ Therefore, Step 5: $\frac{\partial J}{\partial w_{mn}^L} = \delta_n^L \cdot a_m^{L-1}$

- Same can be written in vector form as:

  Step 6: $\frac{\partial J}{\partial W^L} = a^{L-1} \cdot (\delta^L)^t$ → Vector form representation

- Now let us determine the gradient w.r.t. biases ($\frac{\partial J}{\partial b_n^L}$)

Step 1: $\frac{\partial J}{\partial b_n^L} = \frac{\partial J}{\partial z_n^L} \cdot \frac{\partial z_n^L}{\partial b_n^L}$ Step 2: $\frac{\partial J}{\partial b_n^L} = \frac{\partial J}{\partial z_n^L} \cdot \frac{\partial \sum_i w_{in}^L * a_i^{L-1} + b_n^L}{\partial b_n^L} = \frac{\partial J}{\partial z_n^L} \cdot \frac{\partial b_n^L}{\partial b_n^L}$

Step 3: $\frac{\partial J}{\partial b_n^L} = \frac{\partial J}{\partial z_n^L} = \delta_n^L$ Step 4: $\frac{\partial J}{\partial b^L} = \delta^L$ → Vector form representation

Evaluation of δ_n^H for the Outermost Layer (L==H)

Step 1 $J_{(x,y)} = \frac{1}{2}||y - y(x)||^2$

Step 2 $J = \frac{1}{2}||y - a^H||^2$

Reminder: y denotes Truth and y(x) is predicted output at the outermost layer H.

Therefore, y(x) = h(z^H) = a^H

Step 3 $\delta_n^H = \frac{\partial J}{\partial z_n^H} = \frac{\partial J}{\partial a_n^H} \cdot \frac{\partial a_n^H}{\partial z_n^H}$

Step 4 $\frac{\partial J}{\partial a_n^H} = \frac{\partial \frac{1}{2}||y - a^H||^2}{\partial a_n^H} = -\frac{2}{2}(y_n - a_n^H) = a_n^H - y_n$

Step 5 $\frac{\partial J}{\partial a^H} = a^H - y$ → **Vector Form Representation**

Step 6 $\frac{\partial a_n^H}{\partial z_n^H} = \frac{\partial h(z_n^H)}{\partial z_n^H} = h'(z_n^H)$

Step 7 $\delta_n^H = \frac{\partial J}{\partial a_n^H} \cdot \frac{\partial a_n^H}{\partial z_n^H} = (a_n^H - y_n) \cdot h'(z_n^H)$

Step 8 $\delta^H = (a^H - y) \odot h'(z^H)$ → **Vector Form Representation**

Evaluation of δ_n^L for other output Layers (L≠H)

Note: m is neuron # in Layer L+1 and n is neuron # in Layer L

Step 1:

$$\delta_n^L = \frac{\partial J}{\partial z_n^L} = \sum_m \frac{\partial J}{\partial z_m^{L+1}} \cdot \frac{\partial z_m^{L+1}}{\partial z_n^L} = \sum_m \delta_m^{L+1} \cdot \frac{\partial z_m^{L+1}}{\partial z_n^L}$$

Step 2:

$$\frac{\partial z_m^{L+1}}{\partial z_n^L} = \frac{\partial \sum_i (w_{im}^{L+1} * a_i^L + b_m^{L+1})}{\partial z_n^L} = \frac{\partial (w_{nm}^{L+1} * a_n^L)}{\partial z_n^L}$$

Step 3:

$$\frac{\partial z_m^{L+1}}{\partial z_n^L} = w_{nm}^{L+1} \cdot \frac{\partial (a_n^L)}{\partial z_n^L} = w_{nm}^{L+1} \cdot \frac{\partial \left(h(z_n^L)\right)}{\partial z_n^L} = w_{nm}^{L+1} \cdot h'(z_n^L)$$

Evaluation of δ_n^L for other output Layers (L ≠ H)

Step 4: $$\delta_n^L = \frac{\partial J}{\partial z_n^L} = \sum_m \delta_m^{L+1} \cdot w_{nm}^{L+1} \cdot h'(z_n^L) = h'(z_n^L) \sum_m \delta_m^{L+1} \cdot w_{nm}^{L+1}$$

Step 5: $$\delta^L = \frac{\partial J}{\partial z^L} = h'(z^L) \odot W^{L+1} \delta^{L+1}$$ → **Vector Form Representation**

Results in the Elemental Form

⇨ $z_m^L = \sum_i w_{im}^L * a_i^{L-1} + b_m^L$ *unless* $L = 0$ *then* $z_m^L = \sum_i w_{im}^L * x_i + b_m^L$ **Eqn. 1**

⇨ $a_m^L = h(z_m^L)$ **Eqn. 2**

⇦ $\delta_n^L = h'(z_n^L) \sum_m \delta_m^{L+1} \cdot w_{nm}^{L+1}$ *unless* $L = H$ *then* $\delta_n^L = (a_n^L - y_n) \cdot h'(z_n^L)$ **Eqn. 3**

⇦ $\frac{\partial J}{\partial w_{mn}^L} = \delta_n^L \cdot a_m^{L-1}$ *unless* $L = 0$ *then* $\frac{\partial J}{\partial w_{mn}^L} = \delta_n^L \cdot x_m$ **Eqn. 4**

⇦ $\frac{\partial J}{\partial b_n^L} = \delta_n^L$ **Eqn. 5**

Equations 1 and 2 are used for Forward Propagation

Equations 3, 4 and 5 are used for Backward Propagation

Results in the Matrix/Vector Form

⇨ $z^L = (W^L)^t a^{L-1} + b^L$ *unless* $L = 0$ *then* $z^L = (W^L)^t x + b^L$ **Eqn. 1**

⇨ $a^L = h(z^L)$ **Eqn. 2**

⇦ $\delta^L = h'(z^L) \odot W^{L+1}\delta^{L+1}$ *unless* $L = H$ *then* $\delta^L = (a^L - y) \odot h'(z^L)$ **Eqn. 3**

⇦ $\frac{\partial J}{\partial W^L} = a^{L-1} \cdot (\delta^L)^t$ *unless* $L = 0$ *then* $\frac{\partial J}{\partial W^L} = x \cdot (\delta^L)^t$ **Eqn. 4**

⇦ $\frac{\partial J}{\partial b^L} = \delta^L$ **Eqn. 5**

Equations 1 and 2 are used for Forward Propagation

Equations 3, 4 and 5 are used for Backward Propagation

Backward Propagation: Algorithm Implementation

- Backward propagation algorithm is implemented using 5 equations shown in previous two slides

1. Using Eqn. 1 and Eqn. 2, find a^L and z^L for all layers; staring with innermost Layer 0 and successively going *forward* up to the last layer H. This is done by feeding an event data into the innermost Layer 0 of the network. This is known as the *forward pass* as indicated by forward arrow.

2. Using Eqn. 3; compute δ^L for all layers; starting with the last (outermost) Layer H and successively going *backward* up to the innermost layer 0 using the formulas for δ^H, δ^L respectively. This is known as the *backward pass* as indicated by the backward arrow.

3. Using Eqn. 4 and Eqn. 5; simultaneously compute $\partial J/\partial W^L$ and $\partial J/\partial b^L$ (*backward pass mode*) starting with the last (outermost) layer H and successively going *backward* up to the input layer 0

4. Repeat above procedure for more events (as per the batch size) and update the weights and biases of the network through gradient descent as per following equations:

$$W^L := W^L - \frac{\lambda}{d}\sum_x \left.\frac{\partial J}{\partial W^L}\right|_x \;\&\; b^L := b^L - \frac{\lambda}{d}\sum_x \left.\frac{\partial J}{\partial b^L}\right|_x$$

λ is Learning Rate
d is # of events in mini-batch