Contents

1 Neural networks

1.1 Intuition

The input layer, or the data, is layer 0. Hidden layers start from layer 1, and the output layer is the last layer. Superscript square brackets are commonly used to refer to some layer; for example, $\vec{a}^{[i]}$ refers to the vectors of activations in layer i.

where g is some activation function (ex. sigmoid). $\vec{a}$ will be the output of this layer, which is a vector of the activations $[a_1, a_2, a_3]$ from the previous layer.

A more compact way of expressing the activations in a layer is $a_j^{[\ell]} = g(\vec{w}_j^{[\ell]} \cdot \vec{a}^{[\ell-1]} + b_j^{[\ell]})$.

1.2 Activation functions

Linear: $g(z) = z$, used for ex. change in stock prices tomorrow
Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$, used for binary classification
ReLU: $g(z) = \max(0, z)$, used for non-negative predictions ex. housing prices
When choosing $g(z)$ for the output layer:

- Sigmoid for binary classification
- Linear for linear regression with both positive / negative results
- ReLU is similar to linear but if output values can only be non-negative

Choosing $g(z)$ for hidden layers:

- ReLU is the most common (faster, doesn't flatten out like sigmoid)
- Linear is typically not used in hidden layers, because they remove complexity from the model's predictions. A neural network with many layers that all use the linear activation function is not any better than a single regression using the activation function of the output layer.

1.3 Training the model

Derivatives can be computed by changing a function's argument by some small ϵ and seeing how the function changes as a reuslt of its argument change. The derivative is then change in function / change in argument.

For example, given cost function $J(w) = w^2$, $\frac{\partial J}{\partial w}\Big|_{w=3}$ can be determined with $\frac{J(w+\epsilon)-J(w)}{\epsilon}$. Values for ϵ can be as small as 0.0001, because ϵ should simulate an infinitely small number.

1.3.1 Forward propagation

$n^{[\ell]}$ = number of neurons in layer ℓ
$n_f = n^{[0]}$ = num features
m = number of examples

$Z^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times m$ for $\ell > 0$ which holds the values to pass to the activation function of layer ℓ. The $n^{[\ell]}$ rows hold one feature of the m data points in each neuron in layer ℓ. Not applicable for input layer.

$A^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times m$ for $\ell > 0$ which holds activation values that come from $g^{[\ell]}(Z^{[\ell]})$.

$g^{[\ell]}$ is the activation function for layer ℓ.

$W^{[\ell]}$ is a matrix of dimensions $n^{[\ell]} \times n^{[\ell-1]}$ for $\ell > 0$, which holds $\boldsymbol{w}$ values for each neuron in layer ℓ. Not applicable for input layer.

$\boldsymbol{b}^{[\ell]}$ is a list of b values for each neuron in layer ℓ, length $n^{[\ell]}$.

$A^{[0]} = X$ is a matrix of dimensions $n_f \times m$ which holds all input data.

$$Z^{[\ell]} = W^{[\ell]} \cdot A^{[\ell-1]} + \boldsymbol{b}^{[\ell]}$$
$$A^{[\ell]} = g^{[\ell]}(Z^{[\ell]})$$

1.3.2 Back propagation

Including variables from the forward propagation section:

L = last layer

Y is a matrix of dimensions $n^{[L]} \times m$ which holds training data labels.

$\mathrm{d}Z_i^{[\ell]}$ is the ith column of $\mathrm{d}Z$, giving a vector of length $n^{[\ell]}$.

For output layer:

$$\begin{aligned}
\mathrm{d}Z^{[L]} &= A^{[L]} - Y \\
\mathrm{d}W^{[L]} &= \frac{1}{m}\,\mathrm{d}Z^{[L]} A^{[L-1]T} \\
\mathrm{d}\boldsymbol{b}^{[L]} &= \frac{1}{m}\sum_i \mathrm{d}Z_i^{[L]}
\end{aligned}$$

For hidden layers:

$$\begin{aligned}
\mathrm{d}Z^{[\ell]} &= W^{[\ell+1]T}\,\mathrm{d}Z^{[\ell+1]} * g^{[\ell]\prime}(Z^{[\ell]}) \\
\mathrm{d}W^{[\ell]} &= \frac{1}{m}\,\mathrm{d}Z^{[\ell]} A^{[\ell-1]T} \\
\mathrm{d}\boldsymbol{b}^{[\ell]} &= \frac{1}{m}\sum_i \mathrm{d}Z_i^{[\ell]}
\end{aligned}$$

1.3.3 Back propagation derivation

Using loss function $L(a, y) = -y \log a - (1 - y) \log(1 - a)$:
dz (output layer)

$$\begin{aligned}
\frac{dL}{dz} &= \frac{dL}{da}\frac{da}{dz} \\
\frac{da}{dz} &= \frac{d}{dz}g(z) = g'(z) \\
\frac{dL}{da} &= -\frac{y}{a} + \frac{1-y}{1-a} \\
\frac{dL}{dz} &= \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)(a(1-a)) = a - y
\end{aligned}$$

dz (hidden layer)

$$\begin{aligned}
\frac{dL}{dz^{[l]}} &= \frac{dL}{dz^{[l+1]}}\frac{dz^{[l+1]}}{dz^{[l]}} \\
z^{[l+1]} &= w^{[l+1]}a^{[l]} + b^{[l+1]} \\
\frac{dz^{[l+1]}}{dz^{[l]}} &= w^{[l+1]}\frac{da^{[l]}}{dz^{[l]}} \\
\frac{dL}{dz^{[l]}} &= \frac{dL}{dz^{[l+1]}}w^{[l+1]}\frac{da^{[l]}}{dz^{[l]}} = \frac{dL}{dz^{[l+1]}}w^{[l+1]}\sigma'(z^{[l]})
\end{aligned}$$

which in simplified notation is $dz^{[l]} = dz^{[l+1]}w^{[l+1]}\sigma'(z^{[l]})$.
dw

$$\frac{dL}{dw^{[l]}} = \frac{dL}{dz^{[l]}}\frac{dz^{[l]}}{dw^{[l]}} = dz^{[l]}a^{[l-1]}$$

because

$$z = w \cdot a + b \rightarrow \frac{dz}{dw} = a$$

db

$$\frac{dL}{db^{[l]}} = \frac{dL}{dz^{[l]}}\frac{dz^{[l]}}{db^{[l]}} = dz^{[l]}$$

1.4 Convolutional neural network

1.5 Improving model

Cross validation: split data into training and test, use test data to determine how well the model generalizes

1.5.1 Fixing high bias/variance

High bias (underfit): J_{train} high, $J_{train} \approx J_{cv}$
High variance (overfit): J_{train} may be low, $J_{cv} \gg J_{train}$
High bias and high variance: J_{train} high, $J_{cv} \gg J_{train}$
How to fix:

1. Get more training examples (fix high variance)

2. Try smaller sets of features (fix high variance)
3. Add more features (fix high bias)
4. Add polynomial features (fix high bias)
5. Decrease λ (fix high bias)
6. Increase λ (fix high variance)

Neural networks and bias/variance

If J_{train} is high, make the network larger

If J_{cv} is high, get more data

1.5.2 Adding data

Data augmentation: add data with distortions (ex. distorted letters in a letter recognition program)

2 Convolutional neural network

2.1 Edge detection

Starting with vertical edge detection: Vertical edges can be detected by applying a "filter" to an image. ex. with a 6x6 grayscale image convolved with a 3x3 filter:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \end{pmatrix}$$

To convolve the image, place the top left corner of the filter in the top left corner of the image. The sum of the element wise multiplication between the filter and the area of the image it covers is the top left element of the output matrix. Shift the filter one over and repeat, and when the right edge of the filter reaches the right edge of the image shift the filter down one and to the left edge of the image and repeat the process again.

The filter shown only detects left to right light to dark correctly, and given an image where the colors are inversed, the output matrix would have inversed values as well.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -3 & -3 & 0 \\ 0 & -3 & -3 & 0 \\ 0 & -3 & -3 & 0 \\ 0 & -3 & -3 & 0 \end{pmatrix}$$

A convolutional neural network would be able to learn values for the filter that would be more optimal than hand-coded ones:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} * \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{pmatrix}$$

2.2 Padding

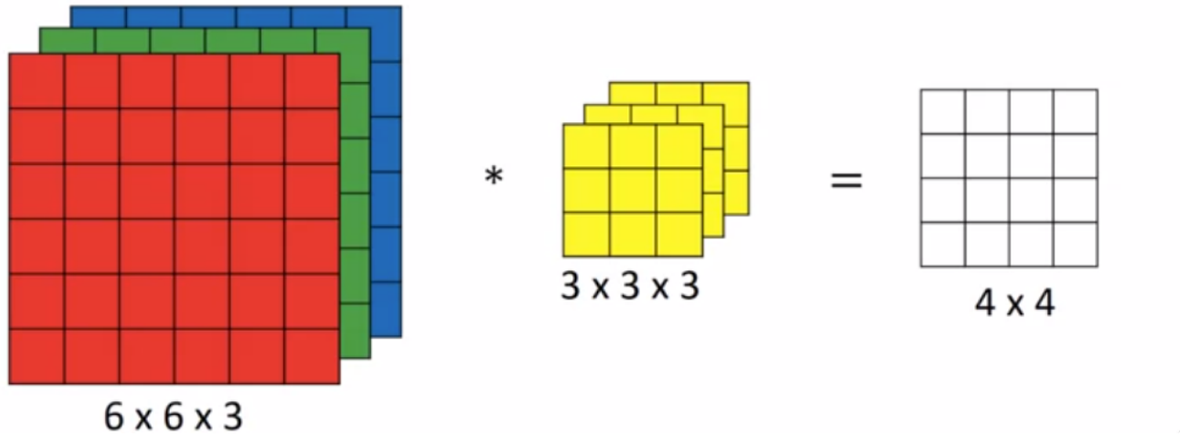
An issue with convolution is that the image shrinks every convolution, and edge information is used much less than center information because more areas overlap with center pixels than edge pixels. Given input image of size $n \times n$ and filter of size $f \times f$, the output image size is $(n - f + 1) \times (n - f + 1)$.

Padding is just adding extra border pixels around the image which are usually set as 0, with thickness denoted as p. The image has new dimensions $(n+2p) \times (n+2p)$ for any $p > 0$, so to get the original image back $2p = f - 1 \rightarrow p = (f - 1)/2$. Because of the division by 2, f is usually odd.

2.3 Strided convolution

Instead of only taking one step at a time when moving the filter across the image, the filter moves with some set stride (ex. 2 instead of the default 1)

The output image dimensions in strided convolution are $(\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1)$, and the padding to ensure same output dimensions is $p = \frac{(n-1)s+f-n}{2}$.



Essentially the same as convolution over a 2D matrix. The number of channels in the filter and the image have to be the same (3 in this case).

Multiple filters can be used by convolving the input image with each filter and then stacking all the output images together. For example, given two filters and one input image, the output matrix size will be 4x4x2, because there are two different filtered 2D output matrices.

2.4 One layer of a cnn

Recall from forward prop:

$$Z^{[1]} = W^{[1]}A^{[0]} + \boldsymbol{b}^{[1]}$$
$$A^{[1]} = g(Z^{[1]})$$

where $A^{[0]}$ is analogous to X. The input 6x6x3 image is X, the filter is W, $\boldsymbol{b}$ is added to the output 4x4 matrix (element wise) which becomes Z, and then some activation function g is applied to that Z, giving the current layer's A. The number of filters can be seen as the number of features.

2.5 Cnn notation

Superscript $[l]$ means relating to layer l.
$f^{[l]}$ = filter size ($f \times f$)
$p^{[l]}$ = padding thickness
$s^{[l]}$ = stride
Input dimensions are $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$.
Output dimensions are $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$.
$n^{[l]} = \frac{n^{[l-1]}+2p^{[l]}-f^{[l]}}{s^{[l]}} + 1$, which is applicable to n_H and n_W.
Each filter is $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
Activations $a^{[l]}$ are $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$.
Vectorized: $A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
Weights $w^{[l]}$ are $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
Vectorized: $W^{[l]} = f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
Bias $\boldsymbol{b}$ is just a vector of length $n_c^{[l]}$, though it becomes more convenient to represent it as $1 \times 1 \times 1 \times n_c^{[l]}$ later.

2.6 Example cnn (not vectorized)

Input image $39 \times 39 \times 3$, first layer uses a set of 10 3×3 filters, second layer uses a set of 20 5×5 filters, third layer uses 40 5×5 filters. All layers use no padding, and stride is chosen arbitrarily.

Input image is $39 \times 39 \times 3$, so $n_W^{[0]} = n_H^{[0]} = 39$ and $n_c^{[0]} = 3$.

For first layer, $f^{[1]} = 3$, $s^{[1]} = 1$, $p^{[1]} = 0$. Because there are 10 filters, there will be 10 channels in the first layer output, or $n_c^{[1]} = 10$, and the dimensions of a single channel in the first layer's output is calculated with $n^{[l]} = \frac{n^{[l-1]}+2p^{[l]}-f^{[l]}}{s^{[l]}} + 1$, which gives 37 for both width and height. The activations will then be $37 \times 37 \times 10$.

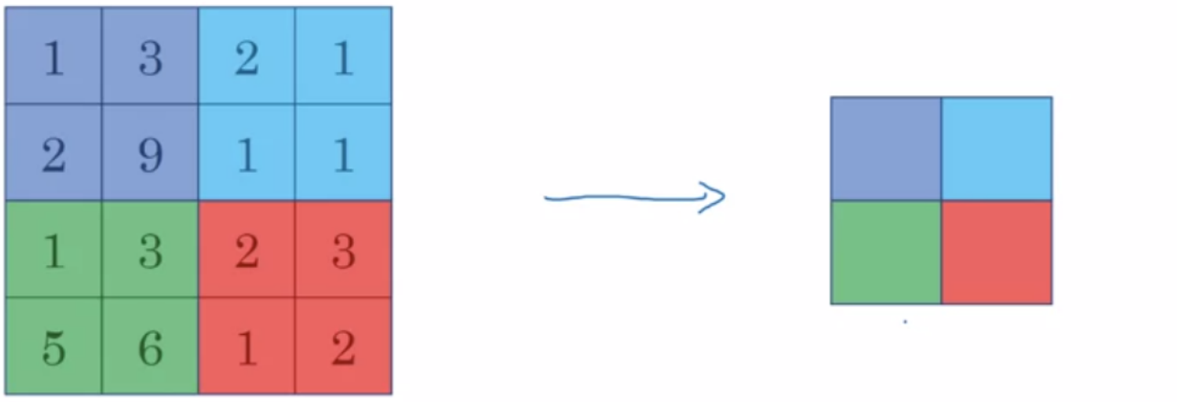
For the second layer, $f^{[2]} = 5$, $s^{[2]} = 2$, $p^{[2]} = 0$, $n_c^{[2]} = 20$. The output volume will then have dimensions of $17 \times 17 \times 20$, where 17 comes from the n_W and n_H calculation and 20 is n_c.

For the third layer, $f^{[3]} = 5$, $s^{[2]} = 2$, $p^{[2]} = 0$. The output volume then has dimensions of $7 \times 7 \times 40$.

After the last layer of filters has been processed, flatten out the final output volume (in this case the $7 \times 7 \times 40$ volume) and feed it to a logistic/softmax regression.

A general trend is that the width and height of each channel in a volume of some layer will be smaller than the previous layer's, while the number of channels increases as the layers go on.

2.7 Pooling layers



Max pooling takes the largest number in each shaded region and puts it in the corresponding color in the resulting 2×2 matrix. The hyperparameters can be represented as $f = 2$, $s = 2$.

Max pooling can be helpful because it identifies important features in an image while also reducing its size, which is computationally less expensive when running the cnn.

Average pooling takes the average of all the values in each shaded region instead of the largest.

In pooling layers, there are no parameters to learn, the only parameters a pooling layer has is its hyperparameters f and s.

Convolutional and pooling layers are usually grouped together into a single layer because the pooling layer has no parameters to learn.

2.8 Fully connected layer

A fully connected layer is just a regular deepnn layer. W dimensions $n^{[l]} \times n^{[l-1]}$, $\boldsymbol{b}$ dimensions $n^{[l]}$.

2.9 Why convolutions

Each filter only needs to learn its own parameters, and then can be applied anywhere on the data (for example, in edge detection, a vertical edge filter is able to detect vertical edges anywhere on the image after only learning a few parameters.

Each output depends on less inputs compared to a normal neural network, which improves performance.

2.10 Forward prop

eth example, nth layer l channel, jth layer $l-1$ channel

$$Z^{[l]}_{en} = \sum_{c=0}^{n_c^{[l-1]}} \text{convolve}(A^{[l-1]}_{ec}, W^{[l]}_{cn}) + \boldsymbol{b}^{[l]}_n$$

$$A^{[l]} = g(Z^{[l]})$$

A^l and Z^l have dimensions $m \times n^l_c \times n^l_h \times n^l_w$

P^l is the pooled A^l with dimensions $m \times n^l_c \times n^l_{ph} \times n^l_{pw}$, and is equal to A^l if there is no pooling.

F^l is the flattened P^l with length $mn^l_c n^l_{ph} n^l_{pw}$

W^l is the weights for the filters in layer l with dimensions $n^l_c \times n^{l-1}_c \times f^l_H \times f^l_W$.

$0 \le e \le m$

$0 \le n \le n^l_c$

$0 \le u \le n^l_H$

$0 \le v \le n^l_W$

$n^l_{ph} = n^l_H / p^l_H$

$n^l_{pw} = n^l_W / p^l_W$

p^l_H = pooling height in layer l

p^l_W = pooling width in layer l

$0 \le c \le n^{l-1}_c$

$0 \le p \le f^l_H$

$0 \le q \le f^l_W$

$0 \le r \le n^{l-1}_{ph}$

$0 \le s \le n^{l-1}_{pw}$

Assume that all pooling layers use max pooling, and that the last layer L is a dense layer with a single softmax neuron.

2.11 Back prop: $\frac{dL}{db}$

Using chain rule:

$$\frac{dL}{db_n^l} = \sum_{u=0}^{n_H^l} \sum_{v=0}^{n_W^l} \frac{dL}{dZ_{enuv}^l} \frac{dZ_{enuv}^l}{db_n^l}$$

$$\frac{dL}{dZ_{enuv}^l} = \frac{dL}{dA_{enuv}^l} \frac{dA_{enuv}^l}{db_n^l}$$

2.11.1 $\frac{dL}{dA_{enuv}^l}$

If A_{enuv}^l is the max out of the elements in its pool, or there is no pooling layer:
$P_{enyx}^l = A_{enu_{max}v_{max}}^l$, where $x = (v_{max}/p_W)$, $y = (u_{max}/p_H)$

$$\frac{dL}{dA_{enuv}^l} = \frac{dL}{dA_{enu_{max}v_{max}}^l} = \frac{dL}{dP_{enyx}^l}$$

$F_k^l = P_{enyx}^l$, where $k = (n_c^l n_{pm}^l n_{pw}^l)e + (n_{ph}^l n_{pw}^l)n + n_{pw}^l y + x$

$$\cdots = \frac{dL}{dP_{enyx}^l} = \frac{dL}{dF_k^l}$$

Since $\frac{dL}{dA_{enu_{max}v_{max}}^l} = \frac{dL}{dF_k}$, it might be helpful to calculate $\frac{dL}{dF_k}$:

$$\frac{dL}{dF_k^l} = \sum_{e=0}^{m} \sum_{i=0}^{n^L} \frac{dL}{dZ_{ei}^L} \frac{dZ_{ei}^L}{dF_k^l}$$

From $Z_{ei}^L = \sum_{j=0}^{k_{max}} (W_{ij}^L F_j^l) + \boldsymbol{b}_i$:

$$\frac{dZ_{ei}}{dF_k^l} = W_{ik}$$

$$\frac{dL}{dF_k^l} = \sum_{e=0}^{m} \sum_{i=0}^{n^L} \frac{dL}{dZ_{ei}^L} W_{ik}$$

With $\frac{dL}{dF_k^l}$ solved, it is now possible to solve for $\frac{dL}{dA_{enuv}^l}$ in the case that dA_{enuv}^l is the largest in its pool.

$$\frac{dL}{dF^l} = W^{lT} \cdot \frac{dL}{dZ^L}$$

$$\frac{dL}{dP^l} = \frac{dL}{dF^l}.\text{reshape}(P^l.\text{shape})$$

If A_{enuv}^l is not the largest in its pool:
This part is not applicable for when there is no pooling done.
Because P only holds the maximum value in each pool for A, changes in A_{enuv}^l when A_{enuv}^l is not the largest value will not affect dL/dP and therefore dL/dA, which gives

$$\frac{dL}{dA_{enuv}^l} = \begin{cases} \frac{dL}{dP_{enyx}^l} \text{ if } A_{enuv}^l \text{ is the largest in its pool} \\ 0 \text{ otherwise} \end{cases}$$

Calculating $\frac{dL}{dA_{enuv}^l}$ (Conv $\rightarrow$ dense)
In forward propagation, cache n, u_{max}, and v_{max} into some array I with the same shape as P^l, which can be accessed with $I[n][y][x] = \begin{bmatrix} u_{max} \\ v_{max} \end{bmatrix}$.
Calculate $\frac{dL}{dP^l}$ using

$$\frac{dL}{dF^l} = W^{lT} \cdot \frac{dL}{dZ^L}$$

$$\frac{dL}{dP^l} = \frac{dL}{dF^l}.\text{reshape}(P^l.\text{shape})$$

and then iterate over all inputs to $\frac{dL}{dP^l}$ (n, y, x) and use

$$\frac{dL}{dA_{enu_{max}v_{max}}^l} = \frac{dL}{dP_{enyx}^l}$$

where u_{max} and v_{max} are fetched from $I[n][y][x]$.
For every $u \neq u_{max}$ and $v \neq v_{max}$, set $\frac{dL}{dA^l_{enuv}}$ to 0.

Calculating $\frac{dL}{dA^l_{enuv}}$ (Conv $\rightarrow$ conv)

Calculate $\frac{dL}{dP^l_{enyx}}$ using

$$\frac{dL}{dP^l_{ecrs}} = \sum_{n=0}^{n_c^l} \sum_{u=0}^{n_H^l} \sum_{v=0}^{n_W^l} \frac{dL}{dZ^{l+1}_{enuv}} \frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}}$$

Only $\frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}}$ needs to be calculated because $\frac{dL}{dZ^{l+1}_{enuv}}$ has already been calculated and should have been cached.

For any n, u, v, there is a 3D region of P^l_{ecrs} contributing to Z^{l+1}_{enuv}, and $\frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}} = W^{l+1}_{ncpq}$, where $p = r - u, q = s - v$ and $r = u \cdots u + 4, s = v \cdots v + 4$.
$\frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}}$ has shape $c_{max} \times r_{max} \times s_{max} \times n_{max} \times u_{max} \times v_{max}$.
$\frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}}$ can be calculated in forward prop with

$$\frac{dZ^{l+1}_{enuv}}{dP^l_{e,0\cdots n_c^{l+1},0\cdots f_H^{l+1},0\cdots f_W^{l+1}}} = W^{l+1}$$

$$\frac{dL}{dP^l} = \sum_{e=0}^{m} \sum_{n=0}^{n_c^{l+1}} \sum_{u=0}^{n_H^{l+1}} \sum_{v=0}^{n_W^{l+1}} \frac{dL}{dZ^{l+1}_{enuv}} \frac{dZ^{l+1}_{enuv}}{dP^l_{ecrs}}$$

During forward prop, cache (c, i_{max}, j_{max}) at which $A^l_{ci_{max}j_{max}}$ is the maximum in its pool.

$$I^l[n, r, s] = \begin{bmatrix} i_{max} \\ j_{max} \end{bmatrix}$$

where $r_{max} = i_{max}/2, s = j_{max}/2$.
$\frac{dL}{dA^l_{ecij}}$ can then be calculated with

$$\frac{dL}{dA^l_{eci_{max}j_{max}}} = \frac{dL}{dP^l_{ecrs}}$$

where (c, r, s) are all iterated over for all their possible values, and i_{max}, j_{max} are obtained from $I^l[c, r, s]$.
$\frac{dL}{dZ^l_{ecij}}$ can then be determined:

$$\frac{dL}{dZ^l_{ecij}} = \frac{dL}{dA^l_{ecij}} \frac{dA^l_{ecij}}{dZ^l_{ecij}}$$

$\frac{dL}{dA^l}$ was calculated earlier, and $\frac{dA}{dZ}$ is just the activation function derivative of Z.
From $z = wa + b$, $\frac{dZ^l_{ecij}}{db^l_c} = 1$.
$\frac{dL}{db^l_c}$ can then be represented as

$$\frac{dL}{db^l_c} = \sum_{i=0}^{n_H^l} \sum_{j=0}^{n_W^l} \frac{dL}{dA^l_{ecij}} \frac{dA^l_{ecij}}{dZ^l_{ecij}}$$

2.11.2 $\frac{dA^l_{enuv}}{dZ^l_{enuv}}$

$$A = g(Z)$$
$$\frac{dA^l_{enuv}}{dZ^l_{enuv}} = g'(Z^l_{enuv})$$

2.11.3 $\frac{dZ^l_{enuv}}{db^l_n}$

From $Z = \text{convolve}(P^{l-1}, W^l) + b^l$ in forward prop:

$$\frac{dZ^l_{enuv}}{db^l_n} = 1$$

2.12 Back prop: $\frac{dL}{dW}$

For conv → dense:

$$\frac{dL}{dW^l_{ncpq}} = \sum_{u=0}^{n^l_H} \sum_{v=0}^{n^l_W} \frac{dL}{dZ^l_{enuv}} \frac{dZ^l_{enuv}}{dW^l_{ncpq}}$$

From $Z^l = \text{convolve}(P^{l-1}, W^l) + b^l$

$$\frac{dZ^l_{enuv}}{dW^l_{ncpq}} = P^{l-1}_{c,p+u,q+v}$$

Substituted:

$$\frac{dL}{dW^l_{ncpq}} = \sum_{u=0}^{n^l_H} \sum_{v=0}^{n^l_W} \frac{dL}{dZ^l_{enuv}} P^{l-1}_{c,p+u,q+v}$$

For conv → conv:
Similar to conv to dense, except $\frac{dZ^l_{ecij}}{dW^l_{ncgh}} = A^{l-1}_{c,g+i,h+j}$.

2.13 Back prop summary

Structure: Conv 1 → Conv 2 → Dense

2.13.1 db: conv 1 → conv 2

$l = 1$
$0 \le e < m$
$0 \le c < n^l_c$
$0 \le i < n^l_H$
$0 \le j < n^l_W$
$0 \le r < n^l_H / p^l_H$
$0 \le s < n^l_W / p^l_W$
p^l_W = pooling window width
p^l_H = pooling window height

$$\frac{dL}{db^l_c} = \sum_{i=0}^{n^l_H} \sum_{j=0}^{n^l_W} \frac{dL}{dA^l_{ecij}} \frac{dA^l_{ecij}}{dZ^l_{ecij}}$$

$$\frac{dL}{dA^l_{ecI^l[c,r,s].i_{max} I^l[c,r,s].j_{max}}} = \frac{dL}{dP^l_{ecrs}}$$

$$\frac{dL}{dA^l_{eci_{\text{not max}} j_{\text{not max}}}} = 0$$

$$I^l[c,r,s] = \begin{bmatrix} i_{max} \\ j_{max} \end{bmatrix}$$

$$\frac{dA^l_{ecij}}{dZ^l_{ecij}} = g'(Z^l_{ecij})$$

I is determined during forward prop.

2.13.2 db: conv 2 → dense

$l = 2$
$0 \le e < m$
$0 \le n < n^l_c$
$0 \le u < n^l_H$
$0 \le v < n^l_W$
$y = u_{max} / p^l_H$
$x = v_{max} / p^l_W$
p^l_W = pooling window width
p^l_H = pooling window height

$$\frac{dL}{db_n^l} = \sum_{e=0}^{m} \sum_{u=0}^{n_H^l} \sum_{v=0}^{n_W^l} \frac{dL}{dZ_{enuv}^l}$$

$$\frac{dL}{dZ_{enuv}^l} = \frac{dL}{dA_{enuv}^l} \frac{dA_{enuv}^l}{dZ_{enuv}^l}$$

$$\frac{dL}{dA_{enuv}^l} = \begin{cases} \frac{dL}{dP_{enyx}^l} \text{ if } A_{enuv}^l \text{ is the largest in its pool} \\ 0 \text{ otherwise} \end{cases}$$

$$\frac{dL}{dP^l} = \frac{dL}{dF^l}.\text{reshape}(P^l.\text{shape})$$

$$\frac{dL}{dF^l} = W^{l,T} \frac{dL}{dZ^L}$$

$$\frac{dA_{enuv}^l}{dZ_{enuv}^l} = g'(Z_{enuv}^l)$$

2.13.3 dw: conv 1 → conv 2

$l = 1$
$0 \leq n < n_c^l$
$0 \leq c < n_c^{l-1}$
$0 \leq p < f_H^l$
$0 \leq q < f_W^l$
$0 \leq u < n_H^l$
$0 \leq v < n_W^l$

$$\frac{dL}{dW_{ncpq}^l} = \sum_{e=0}^{m} \sum_{u=0}^{n_H^l} \sum_{v=0}^{n_W^l} \frac{dL}{dZ_{enuv}^l} \frac{dZ_{enuv}^l}{dW_{ncpq}^l}$$

$$\frac{dZ_{enuv}^l}{dW_{ncpq}^l} = P_{c,p+u,q+v}^{l-1}$$

2.13.4 dw: conv 2 → dense

$l = 2$
$0 \leq n < n_c^l$
$0 \leq c < n_c^{l-1}$
$0 \leq g < f_H^l$
$0 \leq h < f_W^l$
$0 \leq u < n_H^l$
$0 \leq v < n_W^l$

$$\frac{dL}{dW_{ncgh}^l} = \sum_{e=0}^{m} \sum_{u=0}^{n_H^l} \sum_{v=0}^{n_W^l} \frac{dL}{dZ_{enuv}^l} \frac{dZ_{enuv}^l}{dW_{ncgh}^l}$$

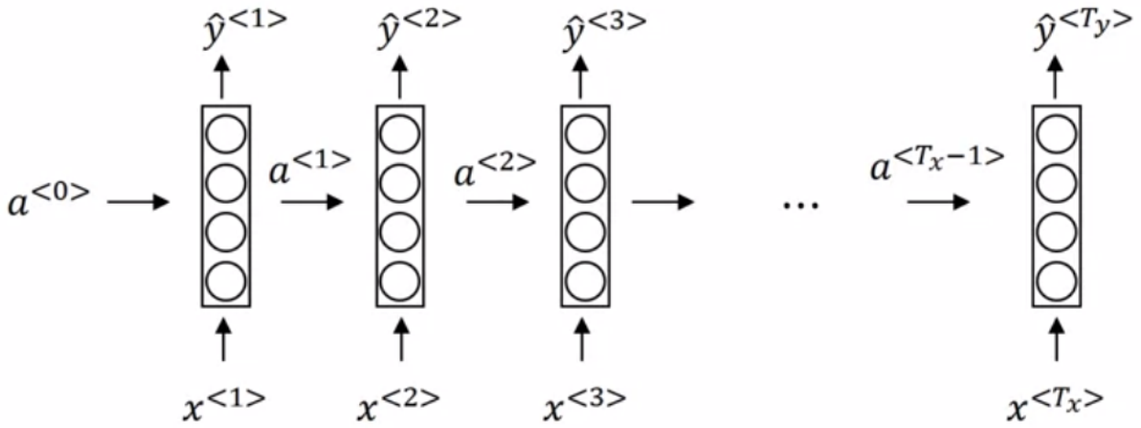
$$\frac{dZ_{enuv}^l}{dW_{ncgh}^l} = A_{c,g+i,h+j}^{l-1}$$

3 Recurrent neural network

3.1 Notation

Superscript $\langle t \rangle$ means the tth position in a single example's sequence of data.
$T_x^{(i)}$ = length of input sequence for example i
$T_y^{(i)}$ = length of output sequence for example i

3.2 Structure



$\boldsymbol{a}^{\langle 0\rangle}$ is usually set to all zeroes at first.

All the layers in the image are just a single layer in the real model that revisits itself for each item in the $\boldsymbol{x}^{(i)}$ sequence multiple times with new $\boldsymbol{a}$ values.

3.3 Forward prop

W_{ab} means W used to compute some a value by multiplying with some b value.

$\boldsymbol{x}$ is a vector because all x values are represented by a one-hot encoding of the tth sequence item using the model's vocabulary.

For activation functions, tanh or relu are common for calculating a, and sigmoid is common for calculating $\hat{y}$.

$W_a = \left[W_{aa}\boldsymbol{a}^{\langle t-1\rangle} \vdots W_{ax}\boldsymbol{x}^{\langle t\rangle}\right]$, or $W_a = W_{aa}\boldsymbol{a}^{\langle t-1\rangle}$ and $W_{ax}\boldsymbol{x}^{\langle t\rangle}$ stacked next to each other in one larger matrix.

$$\boldsymbol{a}^{\langle t\rangle} = g(W_a \begin{bmatrix} \boldsymbol{a}^{\langle t-1\rangle} \\ \boldsymbol{x}^{\langle t\rangle} \end{bmatrix} + \boldsymbol{b}_a)$$
$$\hat{y}^{\langle t\rangle} = g(W_{ya}\boldsymbol{a}^{\langle t\rangle} + \boldsymbol{b}_y)$$

3.4 Back prop

3.5 RNN variants