

Deep Learning Specialization

Formula Sheet

Fady Morris Ebeid
July 6, 2020

Chapter 1

Neural Networks and Deep Learning

1 Standard Notation for Deep Learning

1.1 General Comments

Superscript (i) denotes the i^{th} training example while superscript $[l]$ denotes the l^{th} layer.

Vectors are represented by bold small letters (example: \mathbf{x}) and matrices are represented by bold capital letters (example: \mathbf{X}).

1.2 Sizes

m : Number of examples in the dataset.

n_x : Input size.

n_y : Output size (or number of classes).

$n_h^{[l]}$: number of hidden units of the l^{th} layer.

L : Number of layers in the network.

1.3 Objects

$\mathbf{X} \in \mathbb{R}^{n_x \times m}$: The input matrix.

$\mathbf{x}^{(i)} \in \mathbb{R}^{n_x}$: Is the i^{th} example represented as a column vector.

$\mathbf{Y} \in \mathbb{R}^{n_y \times m}$: Is the label matrix.

$\mathbf{y}^{(i)} \in \mathbb{R}^{n_y}$: Is the output label for the i^{th} example represented as a column vector.

$\mathbf{W}^{[l]} \in \mathbb{R}^{n_h^{[l]} \times n_h^{[l-1]}}$: is the weight matrix, superscript $[l]$ indicates the layer.

$\mathbf{b}^{[l]} \in \mathbb{R}^{n_h^{[l]}}$: Is the bias vector in the l^{th} layer.

$\hat{\mathbf{y}} \in \mathbb{R}^{n_y}$: Is the predicted output vector. It can also be denoted $\mathbf{a}^{[L]}$, where L is the number of layers in the network.

2 Logistic Regression

For one example $\mathbf{x}^{(i)} \in \mathbb{R}^n$:

$$\mathbf{z}^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$$

$$\hat{\mathbf{y}}^{(i)} = \mathbf{a}^{(i)} = \sigma(\mathbf{z}^{(i)})$$

Cross-entropy loss function (for one training example):

$$\mathcal{L}(\mathbf{a}^{(i)}, \mathbf{y}^{(i)}) = -\mathbf{y}^{(i)} \log(\mathbf{a}^{(i)}) - (1 - \mathbf{y}^{(i)}) \log(1 - \mathbf{a}^{(i)})$$

The cost function (for all training examples) is then computed by summing over the loss for all training examples:

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\mathbf{a}^{(i)}, \mathbf{y}^{(i)})$$

Collecting all training examples in a matrix \mathbf{X} :

$$\mathbf{X} = [\mathbf{x}^{(1)} | \mathbf{x}^{(2)} | \dots | \mathbf{x}^{(m)}]$$

$$\mathbf{A} = \sigma(\mathbf{w}^T \mathbf{X} + b) = [\mathbf{a}^{(1)} | \mathbf{a}^{(2)} | \dots | \mathbf{a}^{(m)}]$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \frac{1}{m} \mathbf{X}(\mathbf{A} - \mathbf{Y})^T$$

$$\frac{\partial \mathcal{J}}{\partial b} = \frac{1}{m} \sum_{i=1}^m (\mathbf{a}^{(i)} - \mathbf{y}^{(i)})$$

3 Neural Networks

3.1 Feed-Forward Propagation

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

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]}$$

Input : $\mathbf{A}^{[0]} = \mathbf{X}$

Output : $\mathbf{A}^{[L]} = \hat{\mathbf{Y}}$

Activation Functions

The activation function $g^{[l]}$ can be one of the following :

- **Sigmoid:**

$$\sigma(\mathbf{Z}) = \sigma(\mathbf{W}\mathbf{A} + \mathbf{b}) = \frac{1}{1 + e^{-(\mathbf{W}\mathbf{A} + \mathbf{b})}}$$

- **Rectified Linear Unit (ReLU):**

$$\text{relu}(\mathbf{Z}) = \max(0, \mathbf{Z})$$

Cost Function

Cross-entropy cost function :

$$\begin{aligned} \mathcal{J} &= -\frac{1}{m} \sum_{i=1}^m \left[\mathbf{y}^{(i)} \log(\mathbf{a}^{[L](i)}) + (1 - \mathbf{y}^{(i)}) \log(1 - \mathbf{a}^{[L](i)}) \right] \\ &= -\frac{1}{m} \left[\mathbf{Y} \cdot \log(\mathbf{A}^{[L]T}) + (1 - \mathbf{Y}) \cdot \log(1 - \mathbf{A}^{[L]T}) \right] \end{aligned}$$

3.2 Backpropagation

$$\begin{aligned} d\mathbf{A}^{[L]} &= \frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[L]}} = -\frac{\mathbf{Y}}{\mathbf{A}^{[L]}} + \frac{1 - \mathbf{Y}}{1 - \mathbf{A}^{[L]}} \\ d\mathbf{Z}^{[l]} &= \frac{\partial \mathcal{J}}{\partial \mathbf{Z}^{[l]}} = d\mathbf{A}^{[l]} \odot g^{[l]'}(\mathbf{Z}^{[l]}) \\ d\mathbf{A}^{[l-1]} &= \frac{\partial \mathcal{J}}{\partial \mathbf{A}^{[l-1]}} = \mathbf{W}^{[l]T} d\mathbf{Z}^{[l]} \\ d\mathbf{W}^{[l]} &= \frac{\partial \mathcal{J}}{\partial \mathbf{W}^{[l]}} = \frac{1}{m} d\mathbf{Z}^{[l]} \mathbf{A}^{[l-1]T} \\ d\mathbf{b}^{[l]} &= \frac{\partial \mathcal{J}}{\partial \mathbf{b}^{[l]}} = \frac{1}{m} \sum_{i=1}^m d\mathbf{Z}^{[l](i)} \end{aligned} \quad (1.1)$$

3.3 Gradient Descent

Update the parameters:

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha d\mathbf{W}^{[l]}$$

$$\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha d\mathbf{b}^{[l]}$$

where α is the learning rate.

Chapter 2

Improving Deep Neural Networks: Hyperparameter Tuning

1 Setting up Machine Learning Application

1.1 Train/Dev/Test Sets

Splitting the data into Train/dev(validation)/test sets according to its size

- For small dataset ($m = 100 - 1,000 - 10,000$):
A ratio of 60%, 20%, 20% works well.
- For large datasets ($m = 1,000,000$):
A ratio of 98%, 1%, 1%

2 Regularization

2.1 Logistic Regression

$$\mathcal{J}(\mathbf{w}, \mathbf{b}) = \frac{1}{m} \sum_{j=1}^m \mathcal{L}(\hat{\mathbf{y}}^{(j)}, \mathbf{y}^{(j)}) + \text{Regularization term}$$

The regularization term can be :

- L_2 Regularization : $\frac{\lambda}{2m} \|\mathbf{w}\|_2^2 = \frac{\lambda}{2m} \sum_{j=1}^{n_x} w_j^2 = \mathbf{w}^T \mathbf{w}$
- L_1 Regularization : $\frac{\lambda}{2m} \|\mathbf{w}\|_1 = \frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j|$

2.2 Neural Network

$$\mathcal{J}(\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|\mathbf{W}^{[l]}\|_F^2$$

Where $\|\mathbf{W}^{[l]}\|_F^2$ is called *Frobenius norm* and

$$\|\mathbf{W}^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (W_{i,j}^{[l]})^2$$

Therefore

$$\begin{aligned} \mathcal{J}_{\text{regularized}} &= -\frac{1}{m} \sum_{i=1}^m \left[\mathbf{y}^{(i)} \log(\mathbf{a}^{[L](i)}) + (1 - \mathbf{y}^{(i)}) \log(1 - \mathbf{a}^{[L](i)}) \right] \\ &\quad \underbrace{\hspace{10em}}_{\text{cross-entropy cost}} \\ &\quad + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j \mathbf{W}_{k,j}^{[l]2}}_{\text{L2 regularization cost}} \end{aligned}$$

Backpropagation:

$$d\mathbf{W}^{[l]} \stackrel{(1.1)}{=} \frac{1}{m} d\mathbf{Z}^{[l]} \mathbf{A}^{[l-1]\top} + \frac{\lambda}{m} \mathbf{W}^{[l]}$$

Gradient Descent :

$$\begin{aligned} \mathbf{W}^{[l]} &:= \alpha d\mathbf{W}^{[l]} \\ &:= \mathbf{W}^{[l]} - \alpha \left[\frac{1}{m} d\mathbf{Z}^{[l]} \mathbf{A}^{[l-1]\top} + \frac{\lambda}{m} \mathbf{W}^{[l]} \right] \\ &:= \mathbf{W}^{[l]} - \frac{\lambda \alpha}{m} \mathbf{W}^{[l]} - \alpha \left(\frac{1}{m} d\mathbf{Z}^{[l]} \mathbf{A}^{[l-1]\top} \right) \\ &:= \underbrace{\left(1 - \frac{\alpha \lambda}{m} \right)}_{\text{Weight Decay}} \mathbf{W}^{[l]} - \alpha \left(\frac{1}{m} d\mathbf{Z}^{[l]} \mathbf{A}^{[l-1]\top} \right) \end{aligned}$$

2.3 Dropout

Implementing dropout ("Inverted dropout") in Python.

Illustrate with $l = 3$.

```
keep_prob = 0.8
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob
a3 = np.multiply(a3, d3) # a3 *= d3
a3 /= keep_prob
```

3 Setting Up Optimization Problem

3.1 Normalizing Training Sets

Mean

$$\mu = \frac{1}{m} \sum_{i=1}^m \mathbf{x}^{(i)}$$

Variance

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}^{(i)} \odot \mathbf{x}^{(i)}) - \mu^2$$

Dataset Normalization:

$$\mathbf{x}^{(i)} := \frac{\mathbf{x}^{(i)} - \mu}{\sigma}$$

Note: We use the same μ and σ to normalize the test set.

3.2 Weight Initialization for Deep Networks

To solve the problem of vanishing and exploding gradients.

For *sigmoid* or *tanh* activation function we use *Xavier initialization*:

$$\mathbf{W}^{[l]} = \text{np.random.randn}(\mathbf{W}^{[l]}.shape) * \sqrt{\frac{1}{n^{[l-1]}}}$$

or

$$\mathbf{W}^{[l]} = \text{np.random.randn}(\mathbf{W}^{[l]}.shape) * \sqrt{\frac{1}{n^{[l-1]} + n^{[l]}}}$$

For *ReLU* activation function:

$$\mathbf{W}^{[l]} = \text{np.random.randn}(\mathbf{W}^{[l]}.shape) * \sqrt{\frac{2}{n^{[l-1]}}}$$

3.3 Numerical Approximation of Gradients

Two Sided difference

$$f'(\theta) = \lim_{\varepsilon \rightarrow 0} \frac{f(\theta + \varepsilon) - f(\theta - \varepsilon)}{2\varepsilon}$$

Order of the error $O(\varepsilon^2)$

One sided difference

$$f'(\theta) = \lim_{\varepsilon \rightarrow 0} \frac{f(\theta + \varepsilon) - f(\theta)}{\varepsilon}$$

Order of the error $O(\varepsilon)$

Gradient Checking for a Neural Network

Take $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}$ and reshape into a big vector $\boldsymbol{\theta}$

$$\begin{aligned} \mathcal{J}(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}) &= \mathcal{J}(\boldsymbol{\theta}) \\ &= \mathcal{J}(\theta_1, \theta_2, \dots, \theta_i, \dots) \end{aligned}$$

Take $d\mathbf{W}^{[l]}, d\mathbf{b}^{[l]}, \dots, d\mathbf{W}^{[L]}, d\mathbf{b}^{[L]}$ and reshape into a big vector $d\boldsymbol{\theta}$

For each i :

$$d\theta_i \text{ approx} = \frac{\mathcal{J}(\theta_1, \theta_2, \dots, \theta_i + \varepsilon, \dots) - \mathcal{J}(\theta_1, \theta_2, \dots, \theta_i - \varepsilon, \dots)}{2\varepsilon}$$

$$d\theta_i \text{ approx} \approx d\theta_i = \frac{\partial \mathcal{J}}{\partial \theta_i}$$

$d\boldsymbol{\theta}_{\text{approx}} \approx d\boldsymbol{\theta}$

$$\text{Check } \frac{\|d\boldsymbol{\theta}_{\text{approx}} - d\boldsymbol{\theta}\|_2}{\|d\boldsymbol{\theta}_{\text{approx}}\|_2 + \|d\boldsymbol{\theta}\|_2} < \epsilon$$

in practice we set $\epsilon = 10^{-7}$

Gradient checking implementation notes:

- Don't use in training - only to debug
- If algorithm fails grad check, look at components $(d\mathbf{b}^{[l]}, d\mathbf{W}^{[l]})$ to try to identify bug.
- Remember to include regularization.
- Doesn't work with dropout.
- Run at random initialization; perhaps again after some training.

4 Optimization Algorithms

Suppose that we have m total number of examples.

Batch gradient descent: Using all training examples m at once.

Mini-batch gradient descent: Using a subset ($< m$) of training examples at a time.

Stochastic gradient descent: Using a mini-batch that has just 1 example at a time.

4.1 Mini-Batch Gradient Descent

Reference : [Hin12]

Cost function may not decrease on every iteration.

Algorithm 1: Mini-Batch Gradient Descent

Result: Trained network parameters for each layer $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$

```

1 for each epoch:
2   for each mini-batch t:
3     /* Forward-Propagation on  $\mathbf{X}^{\{t\}}$  */
4      $\mathbf{A}^{[0]} = \mathbf{X}^{\{t\}}$ 
5     for layer  $l = 1, \dots, L$ :
6        $\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]}$ 
7        $\mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]})$ 
8     /* ----- */
9     Compute Cost  $\mathcal{J}^{\{t\}} =$ 
10       $\frac{1}{k} \sum_{i=1}^k \mathcal{L}(\underbrace{\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}}_{\text{For } \mathbf{X}^{\{t\}}, \mathbf{Y}^{\{t\}}}) + \frac{\lambda}{2 \cdot k} \sum_l \|\mathbf{W}^{[l]}\|_F^2$ 
11     Backpropagate to compute gradients w.r.t  $\mathcal{J}^{\{t\}}$ 
12     (using  $(\mathbf{X}^{\{t\}}, \mathbf{Y}^{\{t\}})$ )
13     for layer  $l = 1, \dots, L$ :
14        $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha d\mathbf{W}^{[l]}$ 
15        $\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha d\mathbf{b}^{[l]}$ 

```

Choosing Mini-Batch Size

- If small training set ($m \leq 2000$) : Use *batch gradient descent*
- Typical mini-batch sizes : 64, 128, 256, 512 (Powers of 2)
- Make sure that the mini-batch $\mathbf{X}^{\{t\}}, \mathbf{Y}^{\{t\}}$ fits in CPU/GPU memory.

4.2 Exponentially Weighted Averages

$$V_t = \beta V_{t-1} + (1 - \beta) \theta_t$$

averages over $\approx \frac{1}{1 - \beta}$ previous values of θ

Bias Correction

$$V_t := \frac{V_t}{1 - \beta}$$

4.3 Gradient Descent with Momentum

Momentum β takes past gradients into account to smooth out the steps of gradient descent. It can be applied with batch gradient descent, mini-batch gradient descent or stochastic gradient descent.

Algorithm 2: Gradient Descent with Momentum

Result: Trained network parameters for each layer $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$

```

1  $\mathbf{V}_{d\mathbf{W}^{[l]}} = \mathbf{0}, \mathbf{V}_{d\mathbf{b}^{[l]}} = \mathbf{0}$ 
2 for each epoch:
3   for each mini-batch:
4     Forward-propagation on current mini-batch.
5     Compute cost  $\mathcal{J}$  of current mini-batch.
6     Backpropagate to compute  $d\mathbf{W}^{[l]}, d\mathbf{b}^{[l]}$  on the
7     current mini-batch.
8     for layer  $l = 1, \dots, L$ :
9        $\mathbf{V}_{d\mathbf{W}^{[l]}} := \beta_1 \mathbf{V}_{d\mathbf{W}^{[l]}} + (1 - \beta_1) d\mathbf{W}^{[l]}$ 
10       $\mathbf{V}_{d\mathbf{b}^{[l]}} := \beta_1 \mathbf{V}_{d\mathbf{b}^{[l]}} + (1 - \beta_1) d\mathbf{b}^{[l]}$ 
11       $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha \mathbf{V}_{d\mathbf{W}^{[l]}}$ ,  $\mathbf{b}^{[l]} :=$ 
12       $\mathbf{b}^{[l]} - \alpha \mathbf{V}_{d\mathbf{b}^{[l]}}$ 

```

A common practice is to set the hyperparameter $\beta = 0.9$

4.4 RMSprop

RMSprop stands for *root mean square prop*

Algorithm 3: RMSprop

Result: Trained network parameters for each layer $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$

```

1  $\mathbf{S}_{d\mathbf{W}^{[l]}} = \mathbf{0}, \mathbf{S}_{d\mathbf{b}^{[l]}} = \mathbf{0}$ 
2 for each epoch:
3   for each mini-batch:
4     Forward-propagation on current mini-batch.
5     Compute cost  $\mathcal{J}$  of current mini-batch.
6     Backpropagate to compute  $d\mathbf{W}^{[l]}, d\mathbf{b}^{[l]}$  on the
7     current mini-batch.
8     for layer  $l = 1, \dots, L$ :
9        $\mathbf{S}_{d\mathbf{W}^{[l]}} := \beta_2 \mathbf{S}_{d\mathbf{W}^{[l]}} + (1 - \beta) d\mathbf{W}^{[l] \circ 2}$  /* small */
10       $\mathbf{S}_{d\mathbf{b}^{[l]}} := \beta_2 \mathbf{S}_{d\mathbf{b}^{[l]}} + (1 - \beta) d\mathbf{b}^{[l] \circ 2}$  /* large */
11       $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha \frac{d\mathbf{W}^{[l]}}{\sqrt{\mathbf{S}_{d\mathbf{W}^{[l]}} + \varepsilon}}$ ,
12       $\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha \frac{d\mathbf{b}^{[l]}}{\sqrt{\mathbf{S}_{d\mathbf{b}^{[l]}} + \varepsilon}}$ 

```

4.5 Adam Optimization Algorithm

Adam stands for *Adaptive Moment Estimation*

Paper : [KB14]

Algorithm 4: Adam Optimization Algorithm

Result: Trained network parameters for each layer $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$

```

1  $\mathbf{V}_{d\mathbf{W}^{[l]}} = \mathbf{0}, \mathbf{S}_{d\mathbf{W}^{[l]}} = \mathbf{0}, \mathbf{V}_{d\mathbf{b}^{[l]}} = \mathbf{0}, \mathbf{S}_{d\mathbf{b}^{[l]}} = \mathbf{0}$ 
2  $t = 0$ 
3 for each epoch:
4   for each mini-batch:
5     Forward-propagation on current mini-batch.
6     Compute cost  $\mathcal{J}$  of current mini-batch.
7     Backpropagate to compute  $d\mathbf{W}^{[l]}, d\mathbf{b}^{[l]}$  on the
8     current mini-batch.
9      $t := t + 1$ 
10    for layer  $l = 1, \dots, L$ :
11       $\mathbf{V}_{d\mathbf{W}^{[l]}} := \beta_1 \mathbf{V}_{d\mathbf{W}^{[l]}} + (1 - \beta_1) d\mathbf{W}^{[l]}$ 
12       $\mathbf{V}_{d\mathbf{b}^{[l]}} := \beta_1 \mathbf{V}_{d\mathbf{b}^{[l]}} + (1 - \beta_1) d\mathbf{b}^{[l]}$  /* "moment" */
13       $\mathbf{S}_{d\mathbf{W}^{[l]}} := \beta_2 \mathbf{S}_{d\mathbf{W}^{[l]}} + (1 - \beta_2) d\mathbf{W}^{[l] \circ 2}$ 
14       $\mathbf{S}_{d\mathbf{b}^{[l]}} := \beta_2 \mathbf{S}_{d\mathbf{b}^{[l]}} + (1 - \beta_2) d\mathbf{b}^{[l] \circ 2}$ 
15      /* "RMSprop"  $\beta_2$  */
16       $\mathbf{V}_{d\mathbf{W}^{[l]}}^{\text{corrected}} = \frac{\mathbf{V}_{d\mathbf{W}^{[l]}}}{1 - (\beta_1)^t}$ ,
17       $\mathbf{V}_{d\mathbf{b}^{[l]}}^{\text{corrected}} = \frac{\mathbf{V}_{d\mathbf{b}^{[l]}}}{1 - (\beta_1)^t}$ 
18       $\mathbf{S}_{d\mathbf{W}^{[l]}}^{\text{corrected}} = \frac{\mathbf{S}_{d\mathbf{W}^{[l]}}}{1 - (\beta_2)^t}$ ,
19       $\mathbf{S}_{d\mathbf{b}^{[l]}}^{\text{corrected}} = \frac{\mathbf{S}_{d\mathbf{b}^{[l]}}}{1 - (\beta_2)^t}$ 
20       $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha \frac{\mathbf{V}_{d\mathbf{W}^{[l]}}^{\text{corrected}}}{\sqrt{\mathbf{S}_{d\mathbf{W}^{[l]}}^{\text{corrected}} + \varepsilon}}$ ,
21       $\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha \frac{\mathbf{V}_{d\mathbf{b}^{[l]}}^{\text{corrected}}}{\sqrt{\mathbf{S}_{d\mathbf{b}^{[l]}}^{\text{corrected}} + \varepsilon}}$ 

```

4.6 Hyperparameter Choice

α : needs to be tuned.

β_1 : 0.9 (momentum of $d\mathbf{W}^{[l]}$)

β_2 : 0.999 (momentum of $d\mathbf{W}^{[l] \circ 2}$)

ε : 10^{-8}

4.7 Learning Rate Decay

Learning rate decay is to slowly reduce learning rate over time, to help speeding up the learning algorithm.

$$\alpha = \frac{1}{1 + rt} \alpha_0$$

Where r is the decay rate, t is the epoch number.

Other Learning Rate Decay Methods

- Exponential Decay $\alpha = r^t \cdot \alpha_0$
- $\alpha = \frac{k}{\sqrt{t}} \cdot \alpha_0$
- Discrete staircase
- Manually setting α

5 Hyperparameter Tuning

5.1 Appropriate Scale for Hyperparameters

Suppose you want to search for a parameter $\alpha = i, \dots, j$ on a logarithmic scale instead of a linear scale.

Calculate

$$a = \log_{10} i, \quad b = \log_{10} j$$

then

$$\alpha = 10^r$$

where

$$\begin{aligned} r &\sim U(a, b) \\ &\sim a + (b - a)U(0, 1) \end{aligned}$$

5.2 Hyperparameters for exponentially weighted averages

For sampling the hyperparameter $\beta = i, \dots, j$ used to compute exponentially weighted averages.

$$1 - \beta = 1 - i, \dots, 1 - j$$

Calculate

$$a = \log_{10}(1 - i), \quad b = \log_{10}(1 - j)$$

then

$$\beta = 1 - 10^r$$

where

$$\begin{aligned} r &\sim U(b, a) \\ &\sim b + (a - b)U(0, 1) \end{aligned}$$

6 Batch Normalization

6.1 Implementing Batch Norm

Algorithm 5: Batch Norm

Data: training data \mathbf{X} , batch size = k

```

1 for each Batch  $\mathbf{X}^{\{t\}}$  in  $\mathbf{X}$ :
2   for each Intermediate value  $\mathbf{Z}^{\{t\}[l]} = [\mathbf{z}^{(1)} | \dots | \mathbf{z}^{(k)}]$ 
     in Layer  $l$  in the neural network:
3      $\boldsymbol{\mu}^{\{t\}[l]} = \sum_{i=1}^k \mathbf{z}^{(i)}$ 
4      $\boldsymbol{\sigma}^{\{t\}[l]^2} = \frac{1}{m} \sum_{i=1}^k (\mathbf{z}^{(i)} - \boldsymbol{\mu}^{\{t\}[l]})^2$ 
5      $\mathbf{z}_{\text{norm}}^{(i)} = \frac{\mathbf{z}^{(i)} - \boldsymbol{\mu}^{\{t\}[l]}}{\sqrt{\boldsymbol{\sigma}^{\{t\}[l]^2} + \varepsilon}}$ 
6      $\tilde{\mathbf{z}}^{(i)} = \gamma^{[l]} \mathbf{z}_{\text{norm}}^{(i)} + \beta^{[l]}$ 

```

Batch Norm Gradient Descent

Algorithm 6: Batch Norm Gradient Descent

Result: Trained network parameters for each layer $\mathbf{W}^{[l]}, \beta^{[l]}, \gamma^{[l]}$

```

1 for each epoch:
2   for  $t = 1, \dots, \text{num}(\text{mini-batches})$ :
3     /* Forward-Propagation on  $\mathbf{X}^{\{t\}}$  */
4      $\mathbf{A}^{[0]} = \mathbf{X}^{\{t\}}$ 
5     for layer  $l = 1, \dots, L$ :
6        $\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]}$ 
7       Use Batch Norm (algorithm 5) to Compute  $\tilde{\mathbf{Z}}^{[l]}$  from  $\mathbf{Z}^{[l]}$ 
8        $\mathbf{A}^{[l]} = g^{[l]}(\tilde{\mathbf{Z}}^{[l]})$ 
9     /* ----- */
10    Compute Cost  $\mathcal{J}^{\{t\}} =$ 
11       $\frac{1}{k} \sum_{i=1}^k \mathcal{L}(\underbrace{\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}}_{\text{For } \mathbf{X}^{\{t\}}, \mathbf{Y}^{\{t\}}}) + \frac{\lambda}{2 \cdot k} \sum_l \|\mathbf{W}^{[l]}\|_F^2$ 
12    Backpropagate to compute gradients w.r.t  $\mathcal{J}^{\{t\}}$ 
13    ( $d\mathbf{W}^{[l]}, d\beta^{[l]}, d\gamma^{[l]}$ ) (using  $(\mathbf{X}^{\{t\}}, \mathbf{Y}^{\{t\}})$ )
14    for layer  $l = 1, \dots, L$ :
15       $\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha d\mathbf{W}^{[l]}$ 
16       $\beta^{[l]} := \beta^{[l]} - \alpha d\beta^{[l]}$ 
17       $\gamma^{[l]} := \gamma^{[l]} - \alpha d\gamma^{[l]}$ 

```

6.2 Batch Norm as Regularization

- Each mini-batch $\mathbf{X}^{\{t\}}$ is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $\mathbf{z}^{[l]}$ to scale them to $\tilde{\mathbf{z}}^{[l]}$ within that mini-batch. so similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

6.3 Batch Norm at Test Time

Calculate the weighted average of $\boldsymbol{\mu}^{\{t\}[l]}, \boldsymbol{\sigma}^{\{t\}[l]}$ across all mini-batches $\mathbf{X}^{\{t\}}$

$$\begin{aligned} \mathbf{v}_{\boldsymbol{\mu}}^{\{t\}[l]} &= \beta_w \boldsymbol{\mu}^{\{t-1\}[l]} + (1 - \beta_w) \boldsymbol{\mu}^{\{t\}[l]} \\ \mathbf{v}_{\boldsymbol{\sigma}^2}^{\{t\}[l]} &= \beta_w \boldsymbol{\sigma}^{\{t-1\}[l]^2} + (1 - \beta_w) \boldsymbol{\sigma}^{\{t\}[l]^2} \end{aligned}$$

Bias correction:

$$\begin{aligned} \boldsymbol{\mu}^{[l]} &= \frac{\mathbf{v}_{\boldsymbol{\mu}}^{\{t\}[l]}}{1 - \beta_w} \\ \boldsymbol{\sigma}^{[l]^2} &= \frac{\mathbf{v}_{\boldsymbol{\sigma}^2}^{\{t\}[l]}}{1 - \beta_w} \end{aligned}$$

Then Use them in forward-propagation:

$$\begin{aligned} \mathbf{z}_{\text{norm}}^{[l](i)} &= \frac{\mathbf{z}^{[l](i)} - \boldsymbol{\mu}^{[l]}}{\sqrt{\boldsymbol{\sigma}^{[l]^2} + \varepsilon}} \\ \tilde{\mathbf{z}}^{[l](i)} &= \gamma^{[l]} \mathbf{z}_{\text{norm}}^{[l](i)} + \beta^{[l]} \end{aligned}$$

7 Multi-Class Classification

7.1 Softmax Layer

$$\mathbf{a}^{[L]} = g^{[L]}(\mathbf{z}^{[L]}) = \frac{e^{\mathbf{z}^{[L]}}}{\sum_{i=1}^C e^{z_i^{[L]}}}, \quad a_i^{[L]} = g^{[L]}(z_i^{[L]}) = \frac{e^{z_i^{[L]}}}{\sum_{i=1}^C e^{z_i^{[L]}}}$$

If number of classes $C = 2$, then softmax reduces to logistic regression.

7.2 Loss Function

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{j=1}^C y_j \log \hat{y}_j = -\mathbf{y}^\top \log(\hat{\mathbf{y}})$$

Cost :

$$\mathcal{J}(\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$$

Chapter 3

Convolutional Neural Networks

1 Filters

1	0	-1
1	0	-1
1	0	-1

Vertical edge detection filter

1	1	1
0	0	0
-1	-1	-1

Horizontal edge detection filter

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

2 Notation

n : Original image dimension.

f : Filter size.

p : Padding size.

s : Stride.

3 Padding

Types of Padding:

1. Valid: no padding
2. Same padding: pad so that the output size is the same as the input size.

$$n + 2p - f + 1 = n$$

$$\therefore p = \frac{f-1}{2}$$

f is usually odd in same padding.

4 One Layer of CNN

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} * \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]})$$

Input	$(\mathbf{a}^{[l-1]})$	size: $(n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]})$
Filter	$(\mathbf{W}^{[l]})$	size: $(f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]})$
Bias	$(\mathbf{b}^{[l]})$	size: $(1 \times 1 \times 1 \times n_C^{[l]})$
Output	$(\mathbf{a}^{[l]})$	size: $(n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]})$

$$\begin{aligned} \text{Number of parameters} &= \text{size}(\mathbf{W}^{[l]}) + \text{size}(\mathbf{b}^{[l]}) \\ &= (f^{[l]} \times f^{[l]} \times n_C^{[l-1]} + 1) \times n_C^{[l]} \end{aligned}$$

Output size

$$\begin{aligned} &= n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]} \\ &= \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_C^{[l]} \end{aligned}$$

Number of multiplication operations

$$= n_H^{[l]} \times n_W^{[l]} \times (f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]})$$

Number of summation operations is the same as multiplication.

5 Pooling Layer

No parameters to learn.

$$\text{Input size: } (n_H^{[l-1]} \times n_W^{[l-1]} \times n_C)$$

$$\text{Filter size: } (f^{[l]} \times f^{[l]} \times n_C)$$

$$\text{Output size: } (n_H^{[l]} \times n_W^{[l]} \times n_C)$$

$$\begin{aligned} \text{Output size} &= (n_H^{[l]} \times n_W^{[l]} \times n_C) \\ &= \left(\left\lfloor \frac{n_H^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times \left\lfloor \frac{n_W^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \times n_C \right) \end{aligned}$$

6 Residual Networks

Source paper: [He+15]

Implementing “shortcut” / “skip connection” in a ResNet block:

$$\mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]} * \mathbf{a}^{[l]} + \mathbf{b}^{[l+1]}$$

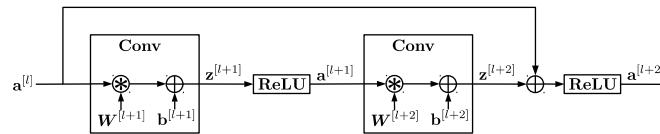
$$\mathbf{a}^{[l+1]} = g^{[l+1]}(\mathbf{z}^{[l+1]})$$

$$\mathbf{z}^{[l+2]} = \mathbf{W}^{[l+2]} * \mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]}$$

For identity block ($\mathbf{a}^{[l]}$ has the same dimensions as $\mathbf{a}^{[l+2]}$):

$$\mathbf{a}^{[l+2]} = g^{[l+2]}(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]})$$

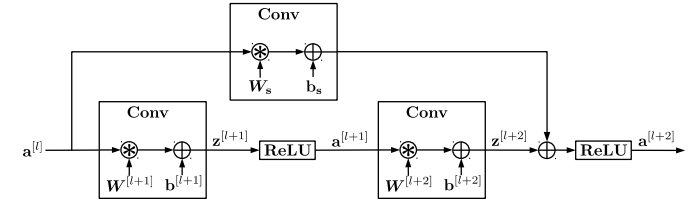
Figure 3.1: Identity block



If $\mathbf{a}^{[l]}$ has different dimensions than $\mathbf{a}^{[l+2]}$, then multiply $\mathbf{a}^{[l]}$ by an extra matrix \mathbf{W}_s

$$\mathbf{a}^{[l+2]} = g^{[l+2]}(\mathbf{z}^{[l+2]} + \mathbf{W}_s * \mathbf{a}^{[l]})$$

Figure 3.2: Convolutional block



7 YOLO Object Detection

References: [Ser+13]

YOLO paper: [Red+15]

YOLO stands for "You Only Look Once"

7.1 Notation

$p_c^{[i]}$: the probability that there is an object for box number i (box i confidence probability)

$c_j^{[i]}$: the probability that the object in box i is a certain class j .

t : maximum number of boxes.

s : number of filtered(selected output boxes).

n_{grid} : Output grid size (number of grid cells in each row and column).

$b_x^{[i]}, b_y^{[i]}$: Midpoint coordinates of box i .

$b_w^{[i]}, b_h^{[i]}$: Height and width of box i .

7.2 The Algorithm

Algorithm 7: YOLO

Data: Input image of shape $(n_H, n_W, 3)$

Result:

A list of selected bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers $[p_c, b_x, b_y, b_h, b_w, c]^T$. If you expand c into an n_{classes} -dimensional vector, each bounding box is then represented by $(5 + n_{\text{classes}})$ numbers. The output tensor shape is $(n_{\text{grid}}, n_{\text{grid}}, s, 6)$, where $s \leq t$ and the last two dimensions can be represented by the matrix:

$$\begin{bmatrix} p_c^{[1]} & p_c^{[2]} & \dots & p_c^{[s]} \\ b_x^{[1]} & b_x^{[2]} & \dots & b_x^{[s]} \\ b_y^{[1]} & b_y^{[2]} & \dots & b_y^{[s]} \\ b_h^{[1]} & b_h^{[2]} & \dots & b_h^{[s]} \\ b_w^{[1]} & b_w^{[2]} & \dots & b_w^{[s]} \\ c^{[1]} & c^{[2]} & \dots & c^{[s]} \end{bmatrix}$$

Steps

- The input image goes through a YOLO CNN Model, resulting in a $(n_{\text{grid}}, n_{\text{grid}}, t, 5 + n_{\text{classes}})$ dimensional output. The last two dimensions can be represented as the following matrix:

$$\begin{bmatrix} p_c^{[1]} & p_c^{[2]} & \dots & p_c^{[t]} \\ b_x^{[1]} & b_x^{[2]} & \dots & b_x^{[t]} \\ b_y^{[1]} & b_y^{[2]} & \dots & b_y^{[t]} \\ b_h^{[1]} & b_h^{[2]} & \dots & b_h^{[t]} \\ b_w^{[1]} & b_w^{[2]} & \dots & b_w^{[t]} \\ c_1^{[1]} & c_1^{[2]} & \dots & c_1^{[t]} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_{\text{classes}}}^{[1]} & c_{n_{\text{classes}}}^{[2]} & \dots & c_{n_{\text{classes}}}^{[t]} \end{bmatrix}$$

- From the output of the YOLO CNN model, extract the following: - **box_confidence**: tensor of shape $(n_{\text{grid}}, n_{\text{grid}}, t, 1)$. The last dimension containing p_c (confidence probability that there's some object) for each of the t boxes predicted in each of the $n_{\text{grid}} \times n_{\text{grid}}$ cells. The last two dimensions of the tensor can be represented as follows:

$$\begin{bmatrix} p_c^{[1]} & p_c^{[2]} & \dots & p_c^{[t]} \end{bmatrix}$$

- **boxes**: tensor of shape $(n_{\text{grid}}, n_{\text{grid}}, t, 4)$ containing the midpoint and dimensions $[b_x, b_y, b_h, b_w]^T$ for each of the t boxes in each cell. The last two dimensions matrix is:

$$\begin{bmatrix} b_x^{[1]} & b_x^{[2]} & \dots & b_x^{[t]} \\ b_y^{[1]} & b_y^{[2]} & \dots & b_y^{[t]} \\ b_h^{[1]} & b_h^{[2]} & \dots & b_h^{[t]} \\ b_w^{[1]} & b_w^{[2]} & \dots & b_w^{[t]} \end{bmatrix}$$

- **box_class_probs**: tensor of shape $(n_{\text{grid}}, n_{\text{grid}}, t, n_{\text{classes}})$ containing the "class probabilities" ($c_1, c_2, \dots, c_{n_{\text{classes}}}$) for each of the n_{classes} classes for each of the t boxes per cell. The last two dimensions can be represented as:

$$\begin{bmatrix} c_1^{[1]} & c_1^{[2]} & \dots & c_1^{[t]} \\ c_2^{[1]} & c_2^{[2]} & \dots & c_2^{[t]} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_{\text{classes}}}^{[1]} & c_{n_{\text{classes}}}^{[2]} & \dots & c_{n_{\text{classes}}}^{[t]} \end{bmatrix}$$

- Convert boxes to be ready for filtering functions (convert boxes from midpoint coordinates to corner coordinates):

$$\begin{bmatrix} b_x^{[1]} & b_x^{[2]} & \dots & b_x^{[t]} \\ b_y^{[1]} & b_y^{[2]} & \dots & b_y^{[t]} \\ b_h^{[1]} & b_h^{[2]} & \dots & b_h^{[t]} \\ b_w^{[1]} & b_w^{[2]} & \dots & b_w^{[t]} \end{bmatrix} \Rightarrow \begin{bmatrix} x_1^{[1]} & x_1^{[2]} & \dots & x_1^{[t]} \\ y_1^{[1]} & y_1^{[2]} & \dots & y_1^{[t]} \\ x_2^{[1]} & x_2^{[2]} & \dots & x_2^{[t]} \\ y_2^{[1]} & y_2^{[2]} & \dots & y_2^{[t]} \end{bmatrix}$$

- Calculate score and predicted class for each box:
 - Box classes: tensor of shape $(n_{\text{grid}}, n_{\text{grid}}, t, 1)$

classes[j,k]

$$= \begin{bmatrix} c^{[1]} & c^{[2]} & \dots & c^{[t]} \end{bmatrix}$$

$$= \text{argmax} \left(\begin{bmatrix} c_1^{[1]} & c_1^{[2]} & \dots & c_1^{[t]} \\ c_2^{[1]} & c_2^{[2]} & \dots & c_2^{[t]} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n_{\text{classes}}}^{[1]} & c_{n_{\text{classes}}}^{[2]} & \dots & c_{n_{\text{classes}}}^{[t]} \end{bmatrix} \right)$$

- Calculate box scores (the probability that the box contains a certain class):

The class score is $\text{scores}^{[i]} = p_c^{[i]} \times c^{[i]}$

scores[j,k]

$$= \begin{bmatrix} p_c^{[1]} & p_c^{[2]} & \dots & p_c^{[t]} \end{bmatrix} \odot \begin{bmatrix} c^{[1]} & c^{[2]} & \dots & c^{[t]} \end{bmatrix} = \begin{bmatrix} p_c^{[1]} c^{[1]} & p_c^{[2]} c^{[2]} & \dots & p_c^{[t]} c^{[t]} \end{bmatrix}$$

- Select only few boxes using score-filtering and non-max suppression:
 - Perform Score-filtering with a threshold: throw away boxes that have detected a class with a

$\text{scores}^{[i]} < \text{threshold}$.

- Non-max suppression:

for each class c_i :

Select the box that has the highest score. Compute the overlap of this box with all other boxes, and remove boxes that overlap significantly ($\text{iou} \geq \text{iou_threshold}$). Iterate until there are no more boxes with a lower score than the currently selected box.

/ The selected boxes count is less than the total number of boxes $s \leq t$ */*

8 Face Recognition

8.1 One-Shot Learning

Learning a similarity function $d(\text{img1}, \text{img2}) = \text{degree of difference between images}$.

$$\text{If } d(\text{img1}, \text{img2}) \begin{cases} \leq \tau & \text{The two images are the same.} \\ > \tau & \text{The two images are the different.} \end{cases}$$

8.2 Siamese Network

Paper : [Tai+14]

Goal of Learning

- Parameters of the neural network define an encoding $f(\mathbf{X}^{(i)})$ of 128 units.
- Learn parameters so that:
If $\mathbf{X}^{(i)}, \mathbf{X}^{(j)}$ are the same person, $d(\mathbf{X}^{(i)}, \mathbf{X}^{(j)})$ is small.
If $\mathbf{X}^{(i)}, \mathbf{X}^{(j)}$ are different persons, $d(\mathbf{X}^{(i)}, \mathbf{X}^{(j)})$ is large.

$$d(\mathbf{X}^{(i)}, \mathbf{X}^{(j)}) = \|f(\mathbf{X}^{(i)}) - f(\mathbf{X}^{(j)})\|_2^2$$

8.3 Triplet Loss

Paper : [SKP15]

Given three input images: an anchor image \mathbf{A} , a positive image \mathbf{P} and a negative image \mathbf{N} ,

We want

$$\begin{aligned} \|f(\mathbf{A}) - f(\mathbf{P})\|_2^2 + \alpha &\leq \|f(\mathbf{A}) - f(\mathbf{N})\|_2^2 \\ \therefore \|f(\mathbf{A}) - f(\mathbf{P})\|_2^2 + \alpha - \|f(\mathbf{A}) - f(\mathbf{N})\|_2^2 &\leq 0 \end{aligned}$$

We define triplet loss function as:

$$\begin{aligned} \mathcal{L}(\mathbf{A}, \mathbf{P}, \mathbf{N}) &= \max \left(\|f(\mathbf{A}) - f(\mathbf{P})\|_2^2 - \|f(\mathbf{A}) - f(\mathbf{N})\|_2^2 + \alpha, 0 \right) \\ &= \left[\underbrace{\|f(\mathbf{A}) - f(\mathbf{P})\|_2^2}_{(1)} - \underbrace{\|f(\mathbf{A}) - f(\mathbf{N})\|_2^2}_{(2)} + \alpha \right]_+ \end{aligned}$$

where,

- The term (1) is the squared distance between the anchor \mathbf{A} and the positive \mathbf{P} for a given triplet; you want this to be small.
- The term (2) is the squared distance between the anchor \mathbf{A} and the negative \mathbf{N} for a given triplet, you want this to be relatively large. It has a minus sign preceding it because minimizing the negative of the term is the same as maximizing that term.
- α is called the margin. It is a hyperparameter that you pick manually.

Triplet cost function can be defined as

$$\mathcal{J} = \sum_{i=1}^m \mathcal{L}(\mathbf{A}^{(i)}, \mathbf{P}^{(i)}, \mathbf{N}^{(i)})$$

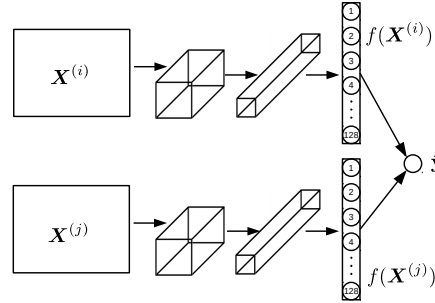
8.4 Face Verification and Binary Classification

Paper : [Tai+14]

Verification: Input is an image and name/ID. Output whether the input image is that of the claimed person.

Recognition: Has a database of K persons. Get an input image and output ID if the image is any of the K persons (or “not recognized”).

Learning a Similarity Function for Face Verification



$$\hat{y} = \sigma \left(\sum_{k=1}^{128} W_k \underbrace{|f(\mathbf{X}^{(i)})_k - f(\mathbf{X}^{(j)})_k|}_{(1)} + b \right)$$

Term (1) can also be the chi square (χ^2) formula:

$$\chi^2 = \frac{[f(\mathbf{X}^{(i)})_k - f(\mathbf{X}^{(j)})_k]^2}{f(\mathbf{X}^{(i)})_k + f(\mathbf{X}^{(j)})_k}$$

9 Neural Image Style Transfer

References: [ZF13], [GEB15]

The goal is to generate an image \mathbf{G} from a content image \mathbf{C} and a style image \mathbf{S} .

9.1 Total Cost Function

$$\mathcal{J}(\mathbf{G}) = \alpha \mathcal{J}_{\text{content}}(\mathbf{C}, \mathbf{G}) + \beta \mathcal{J}_{\text{style}}(\mathbf{S}, \mathbf{G})$$

Where $\mathcal{J}_{\text{content}}$ is the content cost and $\mathcal{J}_{\text{style}}$ is the style cost.

To find the generated image \mathbf{G} :

- Initiate \mathbf{G} randomly
- Use gradient descent to minimize $\mathcal{J}(\mathbf{G})$:

$$\mathbf{G} := \mathbf{G} - \frac{\partial}{\partial \mathbf{G}} \mathcal{J}(\mathbf{G})$$

9.2 Content Cost

- Say you use a hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network).
- Let $\mathbf{a}^{[l](\mathbf{C})}$ and $\mathbf{a}^{[l](\mathbf{G})}$ be the activation of layer l on the images. If they are similar then both images have similar content. The content cost function is:

$$\begin{aligned} \mathcal{J}_{\text{content}}(\mathbf{C}, \mathbf{G}) &= \frac{1}{2} \left\| \mathbf{a}^{[l](\mathbf{C})} - \mathbf{a}^{[l](\mathbf{G})} \right\|_F^2 \\ &= \frac{1}{2} \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} \sum_{k=1}^{n_C^{[l]}} \left(a_{ijk}^{[l](\mathbf{C})} - a_{ijk}^{[l](\mathbf{G})} \right)^2 \end{aligned}$$

9.3 Style Cost

Gram matrix

Let $a_{i,j,k}^{[l]}$ be an element of an activation $\mathbf{a}^{[l]}$ of an input image at layer l at (i, j, k) . Then the *Gram matrix* $\mathbf{G}_{(\text{gram})}^{[l]}$ has a shape of $n_C^{[l]} \times n_C^{[l]}$ and the matrix elements can be calculated as :

$$G_{(\text{gram})kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Gram matrix captures the degree of correlation between a layer l channels as a measure of the style.

Style Cost Function

First calculate the gram matrix for the style image $\mathbf{G}_{(\text{gram})}^{[l](\mathbf{S})}$ and the generated image $\mathbf{G}_{(\text{gram})}^{[l](\mathbf{G})}$ for every layer l . Then the style cost function for a layer l is

$$\begin{aligned} \mathcal{J}_{\text{style}}^{[l]}(\mathbf{S}, \mathbf{G}) &= \frac{1}{(2n_H^{[l]}n_W^{[l]}n_C^{[l]})^2} \left\| \mathbf{G}_{(\text{gram})}^{[l](\mathbf{S})} - \mathbf{G}_{(\text{gram})}^{[l](\mathbf{G})} \right\|_F^2 \\ &= \frac{1}{(2n_H^{[l]}n_W^{[l]}n_C^{[l]})^2} \sum_{i=1}^{n_C^{[l]}} \sum_{j=1}^{n_C^{[l]}} \left(G_{(\text{gram})ij}^{[l](\mathbf{S})} - G_{(\text{gram})ij}^{[l](\mathbf{G})} \right)^2 \end{aligned}$$

And the style cost function for all layers:

$$\mathcal{J}_{\text{style}}(\mathbf{S}, \mathbf{G}) = \sum_l \lambda^{[l]} \mathcal{J}_{\text{style}}^{[l]}(\mathbf{S}, \mathbf{G})$$

Chapter 4

Sequence Models

1 Recurrent Neural Networks

1.1 Notation

$\mathbf{x}^{(t)}$: A one-dimensional input vector of a single example at time step t .

$\mathbf{y}^{(t)}$: Output label at time step t .

$\hat{\mathbf{y}}^{(t)}$: Prediction at time step t .

$\mathbf{a}^{(t)}$: Hidden state, The activation that is passed to the RNN from one time step to another.

T_x : Length of input sequence.

T_y : Length of output sequence.

n_x : Number of units in input.

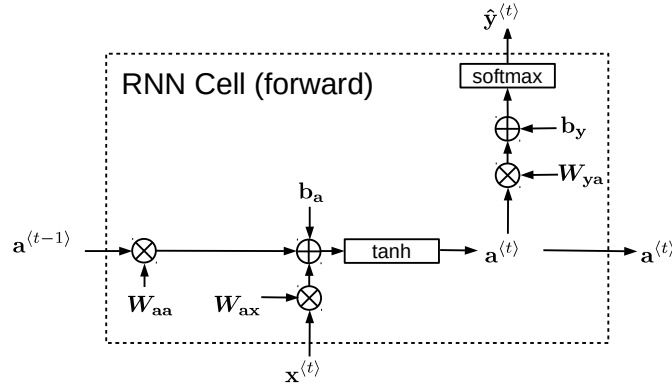
n_y : Number of units in output.

m : batch size.

\mathbf{W} : Weight matrix.

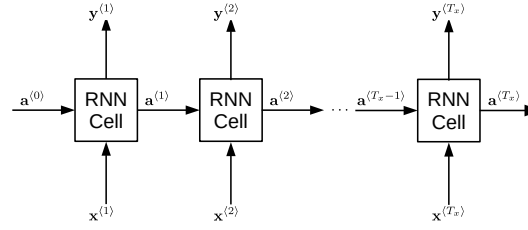
\mathbf{b} : Bias vector.

1.2 Recurrent Neural Networks

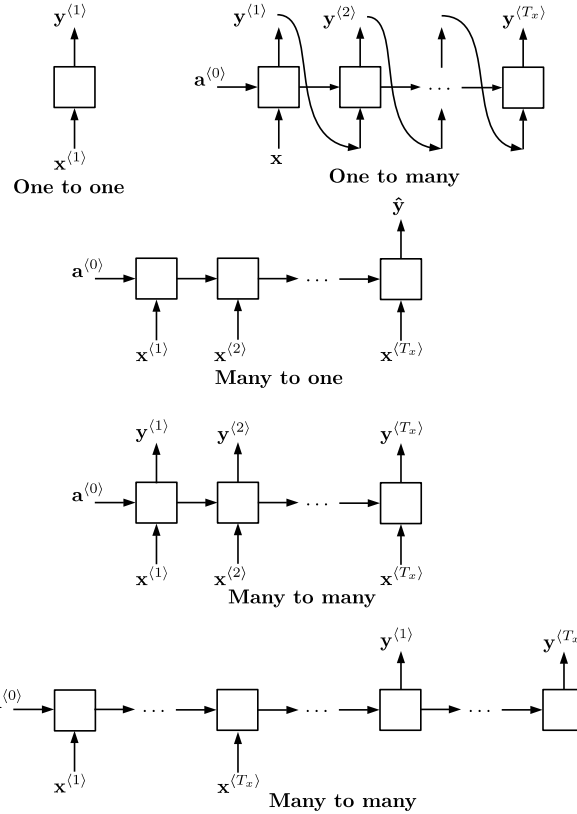


$$\begin{aligned}\mathbf{a}^{(t)} &= \tanh \left(\mathbf{W}_{aa} \mathbf{a}^{(t-1)} + \mathbf{W}_{ax} \mathbf{x}^{(t)} + \mathbf{b}_a \right) \\ &= \tanh \left([\mathbf{W}_{aa} \mid \mathbf{W}_{ax}] \begin{bmatrix} \mathbf{a}^{(t-1)} \\ \mathbf{x}^{(t)} \end{bmatrix} + \mathbf{b}_a \right) \\ &= \tanh \left(\mathbf{W}_a \begin{bmatrix} \mathbf{a}^{(t-1)} \\ \mathbf{x}^{(t)} \end{bmatrix} + \mathbf{b}_a \right)\end{aligned}$$

$$\begin{aligned}\hat{\mathbf{y}}^{(t)} &= \text{softmax} \left(\mathbf{W}_{ya} \mathbf{a}^{(t)} + \mathbf{b}_y \right) \\ &= \text{softmax} \left(\mathbf{W}_y \mathbf{a}^{(t)} + \mathbf{b}_y \right)\end{aligned}$$



RNN Types



Loss Function

$$\mathcal{L}^{(t)}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)}) = -\mathbf{y}^{(t)} \log(\hat{\mathbf{y}}^{(t)}) - (1 - \mathbf{y}^{(t)}) \log(1 - \hat{\mathbf{y}}^{(t)})$$

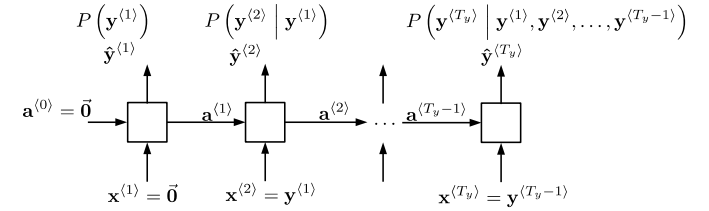
$$\mathcal{J}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T_y} \mathcal{L}^{(t)}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)})$$

1.3 Language Model and Sequence Generation

$$P(\text{Sentence}) = P(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(T_y)})$$

Training

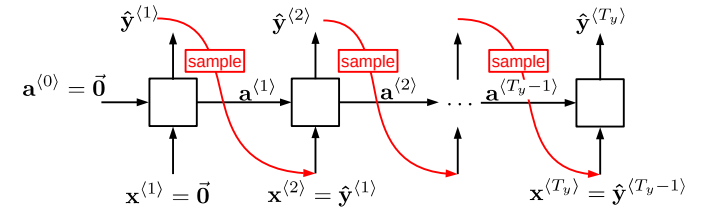
$$\begin{aligned}P(\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(n)}) &= P(\mathbf{y}^{(1)}) P(\mathbf{y}^{(2)} \mid \mathbf{y}^{(1)}) P(\mathbf{y}^{(3)} \mid \mathbf{y}^{(1)}, \mathbf{y}^{(2)}) \\ &\dots P(\mathbf{y}^{(n)} \mid \mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(n-1)}) \\ \mathbf{x}^{(t+1)} &= \mathbf{y}^{(t)}\end{aligned}$$



Loss Function

$$\begin{aligned}\mathcal{L}^{(t)}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)}) &= -\sum_i y_i^{(t)} \log \hat{y}_i^{(t)} \\ \mathcal{J} &= \sum_t \mathcal{L}^{(t)}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)})\end{aligned}$$

Sampling a Sequence from Trained RNN



1.4 Gated Recurrent Unit (GRU)

References: [Cho+14b], [Chu+14]

Notation

$\mathbf{c}^{(t)}$: Memory cell state(variable) at time step t .

$\tilde{\mathbf{c}}^{(t)}$: Candidate value for cell state. Contains information from the current time step that **may** be stored in the current cell state $\mathbf{c}^{(t)}$. Contains values between -1 and 1 .

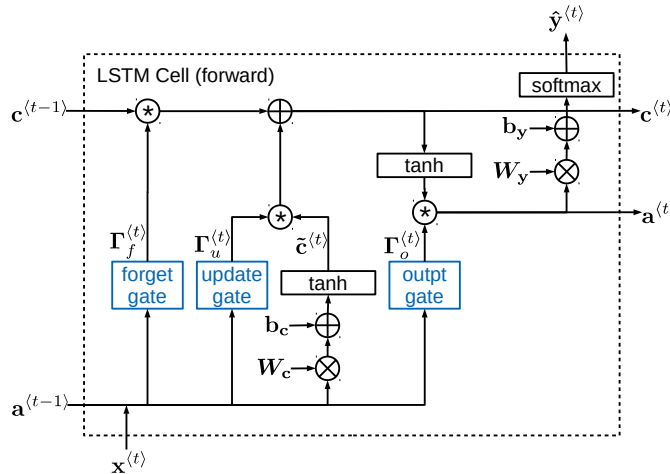
$\Gamma_u^{(t)}$: Update gate. Used to decide what aspects of the candidate $\tilde{\mathbf{c}}^{(t)}$ to add to the cell state $\mathbf{c}^{(t)}$. It contains values that range between 0 and 1 .

GRU(Full)

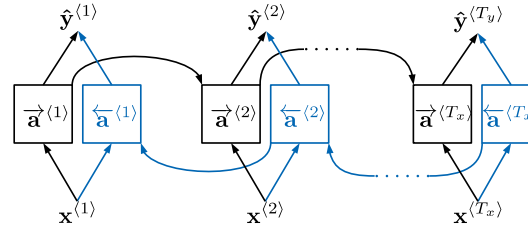
$$\begin{aligned}
\mathbf{c}^{(t-1)} &= \mathbf{a}^{(t-1)} \\
\Gamma_u^{(t)} &= \sigma \left(\mathbf{W}_u \left[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_u \right) \\
\Gamma_r^{(t)} &= \sigma \left(\mathbf{W}_r \left[\mathbf{c}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_r \right) \\
\tilde{\mathbf{c}}^{(t)} &= \tanh \left(\mathbf{W}_c \left[\Gamma_r^{(t)} \odot \mathbf{c}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_c \right) \\
\mathbf{c}^{(t)} &= \mathbf{a}^{(t)} = \Gamma_u^{(t)} \odot \tilde{\mathbf{c}}^{(t)} + (1 - \Gamma_u^{(t)}) \odot \mathbf{c}^{(t-1)} \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax} \left(\mathbf{W}_y \mathbf{a}^{(t)} + \mathbf{b}_y \right)
\end{aligned}$$

1.5 Long Short Term Memory(LSTM)Paper: [\[HS97\]](#)**Notation** $\Gamma_f^{(t)}$: Forget gate. It contains values that range between 0 and 1. $\Gamma_o^{(t)}$: Output gate. Decides what gets sent as the prediction (output) of the time step. It contains values that range between 0 and 1. $\mathbf{a}^{(t)}$: Hidden state. Values between -1 and 1 .**Calculations**

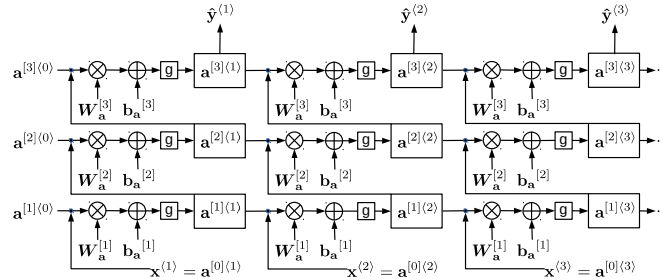
$$\begin{aligned}
\tilde{\mathbf{c}}^{(t)} &= \tanh \left(\mathbf{W}_c \left[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_c \right) \\
\Gamma_u^{(t)} &= \sigma \left(\mathbf{W}_u \left[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_u \right) \\
\Gamma_f^{(t)} &= \sigma \left(\mathbf{W}_f \left[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_f \right) \\
\Gamma_o^{(t)} &= \sigma \left(\mathbf{W}_o \left[\mathbf{a}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_o \right) \\
\mathbf{c}^{(t)} &= \Gamma_u^{(t)} \odot \tilde{\mathbf{c}}^{(t)} + \Gamma_f^{(t)} \odot \mathbf{c}^{(t-1)} \\
\mathbf{a}^{(t)} &= \Gamma_o^{(t)} \odot \tanh \left(\mathbf{c}^{(t)} \right) \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax} \left(\mathbf{W}_y \mathbf{a}^{(t)} + \mathbf{b}_y \right)
\end{aligned}$$

**1.6 Bidirectional RNN**

$$\hat{\mathbf{y}}^{(t)} = g \left(\mathbf{W}_y \left[\vec{\mathbf{a}}^{(t)}, \overleftarrow{\mathbf{a}}^{(t)} \right] + \mathbf{b}_y \right)$$

**1.7 Deep RNNs**

$$\mathbf{a}^{[l]\langle t \rangle} = g \left(\mathbf{W}_a^{[l]} \left[\mathbf{a}^{[l]\langle t-1 \rangle}, \mathbf{a}^{[l-1]\langle t \rangle} \right] + \mathbf{b}_a^{[l]} \right)$$

**2 Natural Language Processing and Word Embeddings****2.1 Notation** n_v : Vocabulary size. n_e : Embedding size, $n_e \ll n_v$. \mathbf{o}_i : One-hot vector for a word i . Its length is n_v . \mathbf{e}_i : Feature vector (word embedding vector) for a word i . Its length is n_e . \mathbf{O} : One-hot matrix, of size $n_v \times n_v$. \mathbf{E} : Embedding matrix, of size $n_e \times n_v$.**2.2 Word Representation**Reference: Visualizing word embeddings [\[MH08\]](#)**2.3 Transfer learning and word embeddings**

1. Learn word embeddings from a large text corpus. (1 - 100B words) (Or download pre-trained embedding online.).
2. Transfer embedding to new task with smaller training set. (eg. 100k words).
3. Optional : continue to fine-tune the word embeddings with new data.

2.4 Properties of Word EmbeddingsReference : [\[MYZ13\]](#).**Analogies using word vectors**

$$\mathbf{e}_{\text{man}} - \mathbf{e}_{\text{woman}} \approx \mathbf{e}_{\text{king}} - \mathbf{e}_{\text{queen}}$$

Find a word w that maximizes the similarity function:

$$\arg \max_w (\text{sim}(\mathbf{e}_w, \mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}}))$$

The similarity function can be one of the following:

- Cosine similarity (more frequently used)

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^T \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} = \cos(\theta)$$

Where θ is the angle between the two vectors.

- Squared distance:

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2$$

2.5 Embedding Matrix

$$\mathbf{E} \cdot \mathbf{o}_j = \mathbf{e}_j$$

In practice we use a specialized function to look up an embedding instead of matrix-vector multiplication.

2.6 A Simple Language ModelReference: [\[Ben+03\]](#)Given an input sequence of words for an example i , with embeddings. $[\mathbf{e}_1^{(i)} \quad \mathbf{e}_2^{(i)} \quad \dots \quad \mathbf{e}_{T_x}^{(i)}]$

First, calculate the average of the sequence embeddings:

$$\mu_{\mathbf{e}}^{(i)} = \mathbb{E} \left[[\mathbf{e}_1^{(i)} \quad \mathbf{e}_2^{(i)} \quad \dots \quad \mathbf{e}_{T_x}^{(i)}] \right] = \frac{1}{T_x} \sum_{n=1}^{T_x} \mathbf{e}_n^{(i)}$$

Forward propagation:

$$\mathbf{z}^{(i)} = \mathbf{W} \mu_{\mathbf{e}}^{(i)} + \mathbf{b}$$

$$\hat{\mathbf{y}}^{(i)} = \mathbf{a}^{(i)} = \text{softmax}(\mathbf{z}^{(i)})$$

Loss function:

$$\mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = \sum_{k=1}^{n_y} y_k^{(i)} \log(\hat{y}_k^{(i)}) = -\mathbf{y}^{(i)T} \log(\hat{\mathbf{y}}^{(i)})$$

Backpropagation:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(i)}} = \mathbf{a}^{(i)} - \mathbf{y}^{(i)}$$

$$\frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{W}} = \mu_{\mathbf{e}}^{(i)}$$

$$\frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{b}} = \mathbf{1}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(i)}} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{W}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(i)}} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{b}}$$

2.7 Word2Vec

Reference: [Mik+13b]

Notation:

t : target word, the word we want to predict.

c : context word, n words before and/or after the target word.

Word2Vec Model (Skipgram model)

Vocabulary size : n_v , embedding size: n_e (for Word2Vec $n_e = 300$).

$$\mathbf{o}_c \xrightarrow{(E)} \mathbf{e}_c \xrightarrow{(\Theta)} \mathbf{z} \xrightarrow{(\text{softmax})} \hat{\mathbf{y}}$$

Where Θ is parameter matrix, its size is $n_e \times n_v$

$$\mathbf{e}_c = E\mathbf{o}_c$$

$$\mathbf{z} = \Theta^T \mathbf{e}_c$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\sum_{i=1}^{n_v} e^{z_i}}$$

$$\hat{y}_t = P(t|c) = \frac{e^{\theta_t^T \mathbf{e}_c}}{\sum_{j=1}^{n_v} e^{\theta_j^T \mathbf{e}_c}}$$

Where θ_j is a column vector of the parameter matrix Θ , θ_t is the parameter vector associated with the output target word t .

The downside of the *skipgram* model is that the softmax objective function is expensive to compute.

Loss function

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{i=1}^{n_v} y_i \log(\hat{y}_i) = -\mathbf{y}^T \log(\hat{\mathbf{y}})$$

2.8 Negative Sampling

Reference: [Mik+13a]

k : Number of negative examples.

y : Target label. 1 for positive example, 0 for negative example.

Model

$$P(y = 1|t, c) = \sigma(\theta_t^T \mathbf{e}_c)$$

On every iteration, choose k different random negative words with which to train the algorithm on. So the total number of training examples is $k + 1$ (including one positive example).

Selecting Negative Examples

Sample according to the empirical frequency of words in your corpus.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{n_v} f(w_j)^{3/4}}$$

2.9 GloVe Word Vectors

Reference: [PSM14]

X_{ij} : Is the number of times word j occurs in the context of word i .

It is a count that captures how often do words i and j appear close to each other.

If you define context to be $\pm n$ words after and before target word, then \mathbf{X} is symmetric ($X_{ij} = X_{ji}$)

Model

$$\text{Minimize} \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} \underbrace{f(X_{ij})}_{(1)} \left(\theta_i^T \mathbf{e}_j + b_i + b'_j - \log(X_{ij}) \right)^2$$

- Term (1), $f(X_{ij})$ is a weighted sum.
- $f(X_{ij}) = 0$ if $X_{ij} = 0$, so the expression evaluates to zero ($0 \log(0) = 0$).
- θ_i and \mathbf{e}_j are symmetric. They end up with the same optimization objective.
- Initialize θ_w and \mathbf{e}_w at random for every word, run gradient descent to optimize them, then take the average of θ_w and \mathbf{e}_w to calculate the final embedding:

$$\mathbf{e}_w^{(\text{final})} = \frac{\mathbf{e}_w + \theta_w}{2}$$

2.10 Debiasing Word Embeddings

Word embeddings can reflect gender, ethnicity, age, sexual orientation and other biases of the text used to train the model, so they need to be debiased [Bol+16].

Addressing bias in word embeddings:

- Identify the bias direction (gender subspace).
Collect n pairs of embedding vectors that differ by gender (masculine m and feminine f), subtract them, then average the result to get the bias vector \mathbf{b} :

$$\mathbf{b} = \frac{1}{n} \sum_{i=1}^n (\mathbf{e}_m^{(i)} - \mathbf{e}_f^{(i)})$$

- Neutralize: For every word embedding that is not definitional, project to get rid of bias.

First calculate the *bias component* \mathbf{e}_B

$$\mathbf{e}_B = \text{proj}_{\mathbf{b}} \mathbf{e} = \frac{\mathbf{e} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}} \mathbf{b}$$

The debiased embedding vector \mathbf{e}^\perp is the orthonormal vector to \mathbf{e} it is obtained by zeroing out the component in the direction of \mathbf{b} :

$$\mathbf{e}^\perp = \mathbf{e} - \mathbf{e}_B$$

- Equalize pairs.

For a pair of words w_1, w_2 that differ by gender:

$$\boldsymbol{\mu} = \frac{\mathbf{e}_{w_1} + \mathbf{e}_{w_2}}{2}$$

$$\boldsymbol{\mu}_B = \frac{\boldsymbol{\mu} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}} \mathbf{b}$$

$$\boldsymbol{\mu}^\perp = \boldsymbol{\mu} - \boldsymbol{\mu}_B$$

$$\mathbf{e}_{w_1B} = \frac{\mathbf{e}_{w_1} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}} \mathbf{b} \quad \mathbf{e}_{w_2B} = \frac{\mathbf{e}_{w_2} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}} \mathbf{b}$$

$$\mathbf{e}_{w_1B}^{(\text{corrected})} = \sqrt{|1 - \|\boldsymbol{\mu}^\perp\|_2^2|} \odot \frac{\mathbf{e}_{w_1B} - \boldsymbol{\mu}_B}{\|(\mathbf{e}_{w_1} - \boldsymbol{\mu}^\perp) - \boldsymbol{\mu}_B\|}$$

$$\mathbf{e}_{w_2B}^{(\text{corrected})} = \sqrt{|1 - \|\boldsymbol{\mu}^\perp\|_2^2|} \odot \frac{\mathbf{e}_{w_2B} - \boldsymbol{\mu}_B}{\|(\mathbf{e}_{w_2} - \boldsymbol{\mu}^\perp) - \boldsymbol{\mu}_B\|}$$

$$\mathbf{e}_1 = \mathbf{e}_{w_1B}^{(\text{corrected})} + \boldsymbol{\mu}^\perp$$

$$\mathbf{e}_2 = \mathbf{e}_{w_2B}^{(\text{corrected})} + \boldsymbol{\mu}^\perp$$

3 Various Sequence to Sequence Architectures

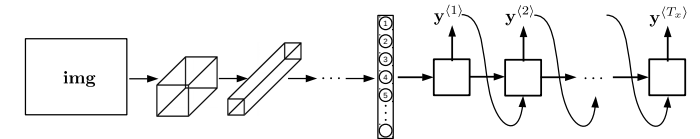
3.1 Basic Models

Sequence to sequence model

References: [SVL14], [Cho+14a]

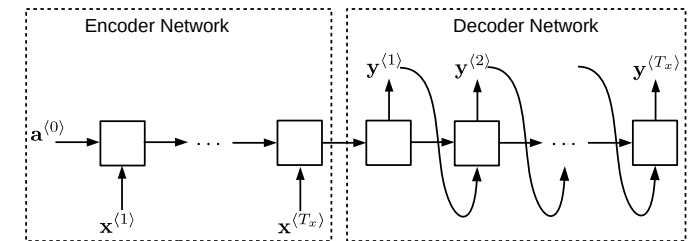
Image Captioning

References: [Mao+14], [Vin+14], [KL15]



3.2 Machine Translation

Building a Conditional Language Model



The model output the conditional probability:

$$P(\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(Tx)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(Tx)})$$

In this model you don't sample words at random. Instead you find a sentence \mathbf{y} that maximizes the conditional probability.

The most common algorithm to do this is called *beam search*

Beam Search

B : Beam width parameter, the number of possibilities for beam search to consider at a time.

Normalized log probability objective function (normalized log likelihood objective):

$$\begin{aligned}\frac{1}{T_y^\alpha} \log P(\hat{\mathbf{y}}|\mathbf{x}) &= \frac{1}{T_y^\alpha} \log P(\hat{\mathbf{y}}^{(1)}, \dots, \hat{\mathbf{y}}^{(T_y)}|\mathbf{x}) \\ &= \frac{1}{T_y^\alpha} \log \prod_{t=1}^{T_y} P(\hat{\mathbf{y}}^{(t)}|\mathbf{x}, \hat{\mathbf{y}}^{(1)}, \dots, \hat{\mathbf{y}}^{(t-1)}) \\ &= \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(\hat{\mathbf{y}}^{(t)}|\mathbf{x}, \hat{\mathbf{y}}^{(1)}, \dots, \hat{\mathbf{y}}^{(t-1)})\end{aligned}$$

Algorithm 8: Beam Search

Data: An input sequence \mathbf{x} , its length is T_x

Result: A sequence of predictions $\hat{\mathbf{y}}$, its length is T_y

1 Run the input sentence \mathbf{x} through the encoder network.

2 Pick the the top B words from the first output of the sequence of the decoder network ($\hat{\mathbf{y}}^{(1)}$) with the highest probabilities as the first predicted word in the sequence.

3 **for** sentence lengths T_y starting from 2:

4 keep track of the top B sentences that maximize the normalized log probability objective function (normalized log likelihood objective).

$$\arg \max_{\hat{\mathbf{y}}} \left(\frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(\hat{\mathbf{y}}^{(t)}|\mathbf{x}, \hat{\mathbf{y}}^{(1)}, \dots, \hat{\mathbf{y}}^{(t-1)}) \right)$$

5 Repeat and increment T_y until encountering an end of sentence character $\langle \text{EOS} \rangle$ for all B sentences.

6 Finally, pick up one sentence from B sentences with the highest value of normalized log likelihood objective as the final translation output.

Notes :

- To avoid numerical underflow (numerical rounding errors) that results of multiplying many small probability numbers, we maximize the log of probabilities instead.

- $\frac{1}{T_y^\alpha}$ is a length normalization term. To prevent objective function from preferring short sentences over long sentences. Reduces the penalty for outputting longer translations.

- α can range between 0 (no normalization) and 1 (full normalization), in practice it is commonly set to 0.7

- Unlike exact search algorithms, beam search runs faster but it is not guaranteed to find the exact maximum for

$$\arg \max_{\hat{\mathbf{y}}} \left(\frac{1}{T_y^\alpha} \log P(\hat{\mathbf{y}}|\mathbf{x}) \right)$$

- The larger B , the more possibilities and better results, but the algorithm becomes slower, more computationally expensive and has more memory requirements.

- For production systems $B = 10$, for research B is chosen to be up to 100.

Error Analysis in Beam Search

\mathbf{y}^* : Translation by a human (reference sentence).

Example:

Human: Jane visits Africa in September (\mathbf{y}^*).

Algorithm: Jane visited Africa last September. ($\hat{\mathbf{y}}$)

- Case 1: $P(\mathbf{y}^*|\mathbf{x}) > P(\hat{\mathbf{y}}|\mathbf{x})$
Beam search chose $\hat{\mathbf{y}}$. But \mathbf{y}^* attains higher $P(\mathbf{y}|\mathbf{x})$.
Conclusion: Beam search is at fault.

- Case 2: $P(\mathbf{y}^*|\mathbf{x}) \leq P(\hat{\mathbf{y}}|\mathbf{x})$
 \mathbf{y}^* is better translation than $\hat{\mathbf{y}}$. But RNN predicted $P(\mathbf{y}^*|\mathbf{x}) \leq P(\hat{\mathbf{y}}|\mathbf{x})$.
Conclusion: RNN model is at fault.

3.3 BLEU Score

[Pap+02]

BLEU: bilingual evaluation understudy.

Modified n -gram precision (p_n) for sentences:

$$p_n = \frac{\sum_{n\text{-gram} \in \hat{\mathbf{y}}} \text{count}_{\text{clip}}(n\text{-gram})}{\sum_{n\text{-gram} \in \hat{\mathbf{y}}} \text{count}(n\text{-gram})}$$

Where $\text{count}_{\text{clip}} = \min(\text{count}, \text{Max_ref_count})$. In other words, one truncates each word's count, if necessary, to not exceed the largest count observed in any single reference for that word.

Combined **BLEU** score for n -grams up to length N :

$$\text{BLEU} = \text{BP} \cdot \exp \left(\frac{1}{N} \sum_{n=1}^N \log p_n \right)$$

Where BP: Brevity penalty.

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

Where c is the length of the candidate translation (machine translation) and r is the effective reference corpus length (reference output length).

3.4 Attention Model

References: [BCB15], [Xu+15]

Properties of The Model

- Pre-attention and Post-attention RNNs on both sides of the attention mechanism

- There are two separate RNNs in this model (see figure): pre-attention and post-attention RNNs.

- Pre-attention Bi-RNN is the one at the bottom of the picture is a Bi-directional RNN and comes before the attention mechanism.

- * The attention mechanism is shown in the middle of the left-hand diagram.

- * The pre-attention Bi-RNN goes through T_x time steps

- Post-attention RNN: at the top of the diagram comes after the attention mechanism.

The post-attention RNN goes through T_y time steps.

- The post-attention RNN passes the hidden state $\mathbf{s}^{(t)}$ from one time step to the next.

- Each time step uses predictions from the previous time step.

Notation

$\vec{\mathbf{a}}^{(t')}$: hidden state of the forward-direction, pre-attention RNN.

$\overleftarrow{\mathbf{a}}^{(t')}$: hidden state of the backward-direction, pre-attention RNN.

$\mathbf{a}^{(t')}$: the concatenation of the activations of both the forward-direction and backward-directions of the pre-attention Bi-RNN.

\mathbf{e} : is called the “energies” variable.

$\mathbf{s}^{(t-1)}$: is the hidden state of the post-attention RNN.

$\mathbf{a}^{(t')}$: is the hidden state of the pre-attention RNN.

$\alpha^{(t,t')}$: The attention variable, amount of “attention” $\mathbf{y}^{(t)}$ should pay to $\mathbf{a}^{(t')}$.

The Model

Figure 4.1: Attention Model

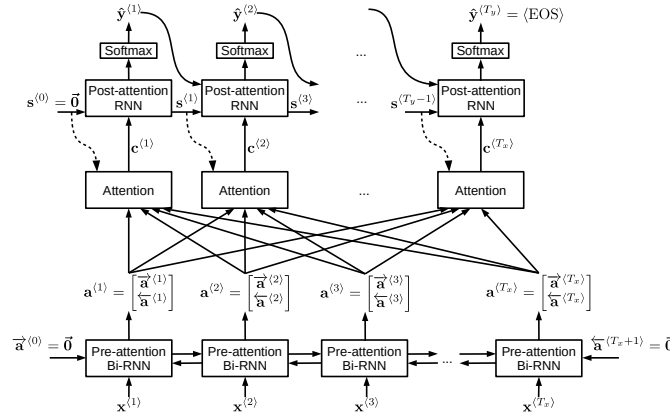
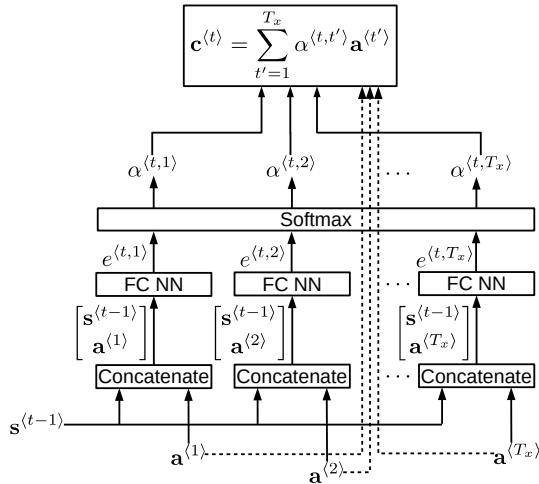


Figure 4.2: one “attention” step



Algorithm 9: Attention Model

Data: An input sequence \mathbf{x} , its length is T_x
Result: A sequence of predictions $\hat{\mathbf{y}}$, its length is T_y
 /* Run the input \mathbf{x} through the pre-attention Bi-RNN to get $[\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \dots, \mathbf{a}^{(T_x)}]$ */
 1 **for** input time steps $t' = 1, \dots, T_x$:
 2

$$\mathbf{a}^{(t')} = \begin{bmatrix} \vec{\mathbf{a}}^{(t')} \\ \overleftarrow{\mathbf{a}}^{(t')} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{a}}^{(t')} \\ \overleftarrow{\mathbf{a}}^{(t')} \end{bmatrix}$$

 /* Pass the sequence of $\mathbf{a}^{(t')}$ to the post-attention RNN to get the predictions $\hat{\mathbf{y}}^{(t)}$ */
 3 **for** output time steps $t = 1, \dots, T_y$:
 4 Compute “energies” $e^{(t,t')}$: $\mathbf{s}^{(t-1)}$ and $\mathbf{a}^{(t')}$ are fed into a simple neural network, which learns the function to output $e^{(t,t')}$.

$$e^{(t,t')} = \text{relu} \left(\mathbf{w}_e^{[2]\top} \cdot \tanh \left(\mathbf{W}_e^{[1]} \left[\mathbf{s}^{(t-1)}, \mathbf{a}^{(t')} \right] + \mathbf{b}_e^{[1]} \right) + b_e^{[2]} \right)$$

 5
 6 Calculate the attention variable $\alpha^{(t,t')}$

$$\alpha^{(t,t')} = \frac{\exp \left(e^{(t,t')} \right)}{\sum_{t'=1}^{T_x} \exp \left(e^{(t,t')} \right)}$$

 7 Calculate the context vector $\mathbf{c}^{(t)}$

$$\mathbf{c}^{(t)} = \sum_{t'=1}^{T_x} \alpha^{(t,t')} \mathbf{a}^{(t')}$$

 8 Pass the computed context vector $\mathbf{c}^{(t)}$ to the post-attention RNN and calculate the hidden state $\mathbf{s}^{(t)}$.

$$\mathbf{s}^{(t)} = \tanh \left(\mathbf{W}_s \left[\mathbf{s}^{(t-1)}, \mathbf{c}^{(t)}, \mathbf{y}^{(t-1)} \right] + \mathbf{b}_s \right)$$

 9 Run the output of the post-attention RNN through a dense layer with softmax activation to generate a prediction $\hat{\mathbf{y}}^{(t)}$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax} \left(\mathbf{W}_y \mathbf{s}^{(t)} + \mathbf{b}_y \right)$$

3.5 Speech Recognition

Reference: [Gra+06]

References

- [BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. arXiv: [1409.0473](#).
- [Ben+03] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *Journal Of Machine Learning Research* 3 (Mar. 2003), pp. 1137–1155. URL: <http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf>.
- [Bol+16] Tolga Bolukbasi et al. “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings”. In: *CoRR* abs/1607.06520 (July 2016). arXiv: [1607.06520](#).
- [Cho+14a] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). arXiv: [1406.1078](#).
- [Cho+14b] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. arXiv: [1409.1259 \[cs.CL\]](#).
- [Chu+14] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: [1412.3555 \[cs.NE\]](#).
- [GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. *A Neural Algorithm of Artistic Style*. 2015. arXiv: [1508.06576 \[cs.CV\]](#).
- [Gra+06] Alex Graves et al. “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 369–376. ISBN: 1-59593-383-2. DOI: [10.1145/1143844.1143891](#). URL: https://www.cs.toronto.edu/~graves/icml_2006.pdf.
- [He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: [1512.03385 \[cs.CV\]](#).
- [Hin12] Geoffrey Hinton. *Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent*. Lecture 6 of the online course “Neural Networks for Machine Learning” on Coursera. 2012. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: [10.1162/neco.1997.9.8.1735](#).
- [KB14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: [1412.6980 \[cs.LG\]](#).
- [KL15] Andrej Karpathy and Fei-Fei Li. “Deep Visual-Semantic Alignments for Generating Image Descriptions”. In: (2015). URL: <https://cs.stanford.edu/people/karpathy/deepimagesent/>.
- [Mao+14] Junhua Mao et al. “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)”. In: (Dec. 2014). URL: <http://www.cs.jhu.edu/~ayuille/Pubs15/JunhuaMaoDeepICLR2015.pdf>.
- [MH08] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605. URL: <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- [Mik+13a] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems* 26. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., Oct. 2013, pp. 3111–3119. arXiv: [1310.4546 \[cs.CL\]](#).
- [Mik+13b] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *Proceedings of Workshop at ICLR* (Jan. 2013). Ed. by Yoshua Bengio and Yann LeCun. arXiv: [1301.3781](#). URL: <http://arxiv.org/abs/1301.3781>.
- [MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751. URL: <https://www.aclweb.org/anthology/N13-1090>.
- [Pap+02] Kishore Papineni et al. “Bleu: a Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: [10.3115/1073083.1073135](#). URL: <https://www.aclweb.org/anthology/P02-1040>.
- [PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. Vol. 14. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: [10.3115/v1/D14-1162](#). URL: <http://www.aclweb.org/anthology/D14-1162>.
- [Red+15] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: [1506.02640 \[cs.CV\]](#).
- [Ser+13] Pierre Sermanet et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2013. arXiv: [1312.6229 \[cs.CV\]](#).
- [SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). DOI: [10.1109/cvpr.2015.7298682](#). arXiv: [1503.03832](#). URL: <http://dx.doi.org/10.1109/CVPR.2015.7298682>.
- [SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems* 27. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112. URL: <http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>.
- [Tai+14] Y. Taigman et al. “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708. URL: https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf.
- [Vin+14] Oriol Vinyals et al. “Show and Tell: A Neural Image Caption Generator”. In: *CoRR* abs/1411.4555 (2014). arXiv: [1411.4555](#).
- [Xu+15] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *CoRR* abs/1502.03044 (2015). arXiv: [1502.03044](#).
- [ZF13] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: [1311.2901 \[cs.CV\]](#).