Tutorial on Backpropagation

Deep Learning Winter School, University of Hull
January 23-24 2018
Dr Nina Dethlefs

Consider the example of a neural network with 2 input nodes i_1 and i_2, two hidden nodes h_1 and $h2$ and two output nodes o_1 and o_2. The weights shown in Figure 1 are chosen randomly for our example. Normally, these would be very small values drawn from a distribution [1, 2]. This is what Keras does too [3].

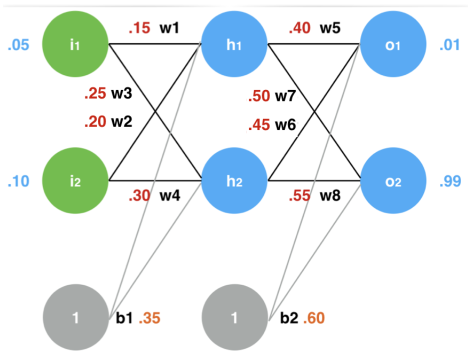

Figure 1: Neural network with weights.

The backpropagation algorithm

For our example, we will compute the weight updates for our neural net using the backpropagation algorithm [4], shown here in pseudocode notation.[1] Backpropagation is the main method underlying modern deep learning, see [5, 6, 4] for an article overviews or [7] for a detailed introduction.

Algorithm 1 Backpropagation algorithm (from Wikipedia).

function COMPUTEWEIGHTS
  initialise network weights (often small random values)
  for each training exampled named ex **do**
    prediction = neural-net-output (network, ex) // forward pass
    actual = teacher-output (ex)
    compute error (prediction - actual) at the output units
    compute Δ_{w_h} for all weights from hidden layer to output layer // backward pass
    compute Δ_{w_i} for all weights from input layer to hidden layer // backward pass continued
    update network weights // input layer not modified by error estimate
  end for
  until all examples classified correctly or another stopping criterion is satisfied
  return the network
end function

[1] https://en.wikipedia.org/wiki/Backpropagation

The forward step

The purpose of the **forward step** is to compute a first set of output predictions and from these deduce the total error of the current network and its weights. The following equation get us a weight for h_1, called net_{h1}:

$$\begin{aligned} net_{h1} &= w_1 * i_1 + w_2 * i_2 + b_1 * 1 \\ net_{h1} &= 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775 \end{aligned} \tag{1}$$

We can see that net_{h1} is obtained the two input nodes i_1 and i_2, weights w_1 and w_2 (because these are the ones that lead to h_1, see Figure 1) and the bias term b_1.

Exercise 1: Compute the value for net_{h2}.

$$\begin{aligned} net_{h2} &= w_3 * i_1 + w_4 * i_2 + b * 0.35 \\ &= 0.25 * 0.05 + 0.3 * 0.1 + 0.35 \\ &= 0.3935 \end{aligned} \tag{2}$$

Activation functions

As a next step, we want to put the values for net_{h1} net_{h2} through an **activation function**, e.g. the logistic function (in this example), also known as the sigmoid function. An activation function is important because it is what makes neural nets good at finding functions for non-linearly distributed datasets. See [8] for an early reference on activation functions and [9] fore a more recent comparison. [7] also includes a comparison.

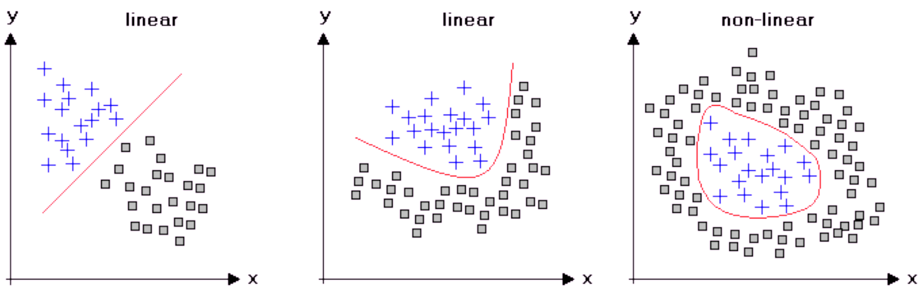

Figure 2: Linear vs non-linear distribution of data points.

Consider the distributions of data points in Figure 2.[2] There are different types of activation functions, the most common are logistic sigmoid, tangent and ReLU. They all have different properties and are thus suited to different learning tasks. For example as we can see in Figure 3[3], a sigmoid function returns values in the range of {0 . . . 1}, whereas a tangent function uses a {-1 . . . 1} range. A ReLU (rectified linear unit) is just always a positive value.

[2]From http://www.statistics4u.com/fundstat_eng/cc_linvsnonlin.html
[3]From http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

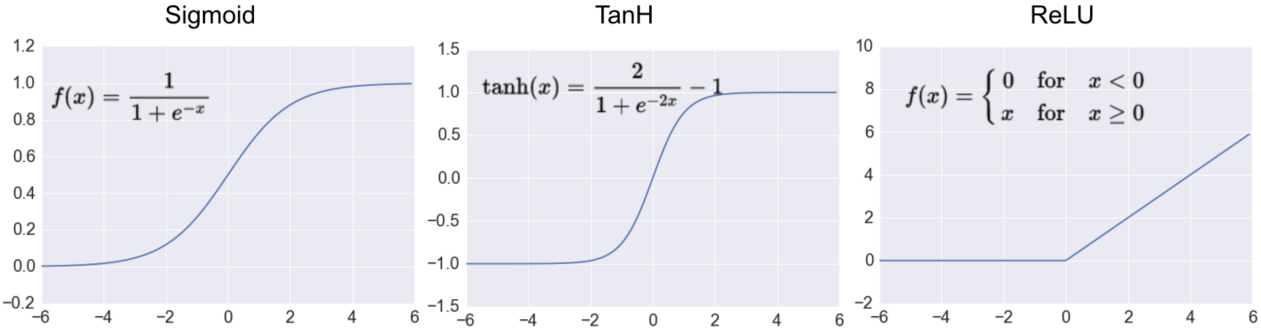

Figure 3: Illustration of activation functions sigmoid, TanH and ReLU.

We apply the logistic sigmoid function to net_{h1} as:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3774}} = 0.593269992 \tag{3}$$

to obtain out_{h1}. Following the same procedure for h_2, we get out_{h2}:

$$out_{h2} = 0.596884378 \tag{4}$$

Exercise 2: Follow the steps above to compute out_{h2}.

$$out_{h2} = \frac{1}{1+e^{-0.3925}} = 0.596884378 \tag{5}$$

Computing an internal feature representation

At this point, we have computed an internal feature representation. This representation is independent of the representation we chose for our inputs but is an internal representation that the neural networks uses. Discovering optimal feature representations automatically is a research topic in its own right. See e.g. [10] for natural language, [11, 12] for computer vision and [?] for a general discussion. [13] discusses dimensionality reduction. If you are interested in feature learning, then autoencoders will also be a good model to explore.[4]

Returning to our weights, we repeat the steps for out_{h1} and out_{h2} above to compute the weights for the output layer, for o_1 and o_2, called net_{o1} and net_{o2}.

$$\begin{aligned} net_{o1} &= w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1 \\ net_{o1} &= 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967 \end{aligned} \tag{6}$$

[4] See e.g. Andrew Ng's lecture notes: `https://pdfs.semanticscholar.org/eb2f/e260af00818907fe82024203d8a5a1386777.pdf`

Applying the activation function, we obtain

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507 \tag{7}$$

And applying the same for o_2 we get $out_{o2} = 0.77298465$.

Exercise 3: Step through the maths for compute out_{o2}.

$$\begin{aligned} net_{o2} &= w_7 * out_{h1} + w_8 * out_{h2} * 0.6 \\ net_{o2} &= 0.5 * 0.593269992 + 0.55 * 0.596884378 + 0.6 = 1.2248 \\ out_{o2} &= \frac{1}{1+e^{-net_{o2}}} = \frac{1}{1+e^{-1.2248}} = 0.772928465 \end{aligned} \tag{8}$$

The total error

This allows us to calculate the total error of our current neural network as:

$$E_{total} = \sum_{o} \frac{1}{2}(target_o - output_o)^2 \tag{9}$$

So for E_{o1} we get:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083 \tag{10}$$

and for E_{o2}, following the same procedure, we get $E_{o2} = 0.023560026$.

Exercise 4: Make sure you get the same number for E_{o2}.

$$E_{o2} = \frac{1}{2}(target_{o2} - out_{o2})^2 = \frac{1}{2}(0.99 - 0.772928465)^2 = 0.023560026 \tag{11}$$

This allows us to compute the total error as:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109 \tag{12}$$

The softmax function

Note that these are not probabilities but just numbers. If we wanted probabilities, we need to apply another activation function that helps us convert these numbers into a number between 0 and 1. Using the softmax equation on our numbers we get:

$$\begin{aligned} P(out_{o1}) &= \frac{out_{o1}}{out_{o1} + out_{o2}} = \frac{0.75136507}{0.75136507 + 0.772928465} = 0.4929267577 \\ P(out_{o2}) &= \frac{out_{o2}}{out_{o1} + out_{o2}} = \frac{0.772928465}{0.75136507 + 0.772928465} = 0.50707324229 \end{aligned} \tag{13}$$

Plotting the error

Plotting the error over time leads to a curve that typically looks something like Figure 4. We see a sharp decrease in the error in the beginning which becomes asymptotic towards the end.

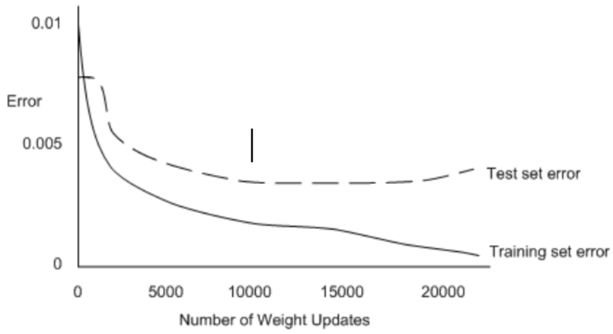

Figure 4: Loss observed over training epsiodes.

The backward step

So far, we have only computed the current error. We now need to do the **backward step** to compute the weight updates for our neural network that will reduce the total error over time. Otherwise our neural network will never learn.

Partial derivatives (aka gradients)

Lets start by computing the update for w_5. The slightly complicated term $\frac{\partial E_{total}}{\partial w_5}$ tells us how an update in w_5 affects the overall error of the network. It reads "the partial derivative of E_{total} with respect to w_5". This is also known as "the gradient with respect to w_5". We can find this by applying the chain rule:

$$\begin{aligned} \frac{\partial E_{total}}{\partial w_5} &= \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5} \\ &= \delta_{o1} out_{h1} \\ \text{where } \delta_{o1} &= -(target_{o1} - out_{o1}) \times out_{o1}(1 - out_{o1}) \end{aligned} \tag{14}$$

Substituting the values in δ_{o1} we get

$$\delta_{o1} = (0.01 - 0.75136507) \times 0.75136507 \times (1 - 0.75136507)) = 0.13849856 \tag{15}$$

and therefore

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1} = 0.13849856 \times 0.593269 = 0.0821669 \tag{16}$$

To compute $\frac{\partial E_{total}}{\partial w_6}$, we calculate

$$\begin{aligned} \frac{\partial E_{total}}{\partial w_6} &= \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_6} \\ &= \delta_{o1} out_{h2} \\ &= 0.13849856 \times 0.596884378 \\ &= 0.082666763 \end{aligned} \tag{17}$$

Exercise 5: Follow the steps above to get values for $\frac{\partial E_{total}}{\partial w_7}$ and $\frac{\partial E_{total}}{\partial w_8}$. For w_7 :

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_7} \\ = \delta_{o2} out_{h1}\end{aligned} \tag{18}$$

Substituting the values in δ_{o2} we get

$$\delta_{o2} = -(0.99 - 0.772928465) \times 0.772928465 \times 0.2270715 = -0.03898231 \tag{19}$$

and therefore

$$\frac{\partial E_{total}}{\partial w_7} = \delta_{o2} out_{h1} = -0.03898231 \times 0.593269 = -0.022602537 \tag{20}$$

And for w_8 :

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_8} = \frac{\partial E_{total}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_8} \\ = \delta_{o2} out_{h2} \\ = -0.038098231 \times 0.596884378 \\ = -0.022740239\end{aligned} \tag{21}$$

To update the neural network's weights, we apply the above calculations as follows:

$$\begin{aligned} w_5^+ &= w_5 - \eta \times \frac{\partial E_{total}}{\partial w_5} = 0.40 - 0.5 \times 0.082167041 = 0.35891648 \\ w_6^+ &= w_6 - \eta \times \frac{\partial E_{total}}{\partial w_6} = 0.45 - 0.5 \times 0.082666763 = 0.408666186 \\ w_7^+ &= w_7 - \eta \times \frac{\partial E_{total}}{\partial w_7} = 0.50 - 0.5 \times -0.022602537 = 0.511301269 \\ w_8^+ &= w_8 - \eta \times \frac{\partial E_{total}}{\partial w_8} = 0.55 - 0.5 \times -0.022602537 = 0.561370119 \end{aligned} \tag{22}$$

w_i^+ represents the new value of w_i. Here, η is the learning rate.

Learning rates

A learning rate in a neural network is optional and indicates roughly how quickly we want the neural network to adjust its weights when observing examples. See [14] for an early analysis. In other words, it indicates how quickly we want the neural net to learn. An illustration of different learning rates is in Figure 5.[5]

As a rule of thumb, we often need a higher learning rate when we have few examples, i.e. a small training set, and a smaller learning rate when we have many examples.

---

[5] From `http://cs231n.github.io/neural-networks-3/`

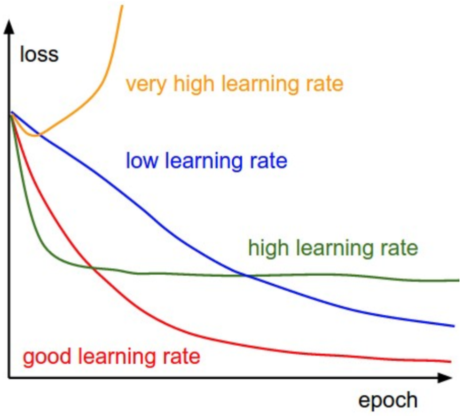

Figure 5: Comparison of learning rates and their impact on the loss.

Once we have these numbers, we need to find the updates of these weights that led into the hidden layer, i.e. w_1^+, w_2^+, w_3^+ and w_4^+.

The process for finding these is similar to what we did before but slightly different too. This is because the output weights from hidden nodes h_1 and h_2 affect several weights. For example, out_{h1} affects both o_1 and o_2, and therefore their error. So the partial derivative of out_{h1} needs to consider both out_{o1} and out_{o2}.

We can find $\frac{\partial E_{total}}{\partial w_1}$ as follows:

$$\begin{aligned} \frac{\partial E_{total}}{\partial w_1} &= \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1} \\ &= \left(\sum_o \frac{\partial E_{total}}{\partial out_o} \times \frac{\partial out_o}{\partial net_o} \times \frac{\partial net_o}{\partial out_{h1}} \right) \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1} \\ &= \delta_{h1} i_1 \\ \text{where } \delta_{h1} &= \left(\sum_o \delta_o \times w_{ho} \right) \times out_{h1}(1 - out_{h1}) \end{aligned} \tag{23}$$

Substituting the values in δ_{h1} we get

$$\begin{aligned} \delta_{h1} &= \big[\big(\delta_{o1} \times w_5 \big) + \big(\delta_{o2} \times w_7 \big) \big] \times out_{h1}(1 - out_{h1}) \\ &= \big[\big(0.13849856 \times 0.4 \big) + \big(-0.038098231 \times 0.5 \big) \big] \times 0.593269992 \times (1 - 0.593269992) \\ &= \big[0.055395424 - 0.019044115 \big] \times 0.593269992 \times 0.40673 \\ &= 0.036350308 \times 0.593269992 \times 0.40673 = 0.008771355 \end{aligned} \tag{24}$$

and therefore

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1 = 0.008771355 \times 0.05 = 0.000438568 \tag{25}$$

To compute $\frac{\partial E_{total}}{\partial w_2}$, we calculate

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_2} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_2} \\ = \delta_{h1} i_2 \\ = 0.008771355 \times 0.10 = 0.00087714\end{aligned} \tag{26}$$

We can find $\frac{\partial E_{total}}{\partial w_3}$ as follows:

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_3} = \frac{\partial E_{total}}{\partial out_{h2}} \times \frac{\partial out_{h2}}{\partial net_{h2}} \times \frac{\partial net_{h2}}{\partial w_3} \\ = \Big(\sum_o \frac{\partial E_{total}}{\partial out_o} \times \frac{\partial out_o}{\partial net_o} \times \frac{\partial net_o}{\partial out_{h2}} \Big) \times \frac{\partial out_{h2}}{\partial net_{h2}} \times \frac{\partial net_{h2}}{\partial w_3} \\ = \delta_{h2} i_1 \\ \text{where } \delta_{h2} = \Big(\sum_o \delta_o \times w_{ho} \Big) \times out_{h2}(1 - out_{h2})\end{aligned} \tag{27}$$

Substituting the values in δ_{h2} we get

$$\begin{aligned}\delta_{h2} = \big[\big(\delta_{o1} \times w_6 \big) + \big(\delta_{o2} \times w_8 \big) \big] \times out_{h2}(1 - out_{h2}) \\ = \big[\big(0.13849856 \times 0.45 \big) + \big(-0.038098231 \times 0.55 \big) \big] \times 0.596884378 \times (1 - 0.596884378) \\ = \big[0.062324352 - 0.020954027 \big] \times 0.596884378 \times 0.403115622 \\ = 0.041370325 \times 0.596884378 \times 0.403115622 = 0.0009954255\end{aligned} \tag{28}$$

and therefore

$$\frac{\partial E_{total}}{\partial w_3} = \delta_{h2} i_1 = 0.009954255 \times 0.05 = 0.000497713 \tag{29}$$

To compute $\frac{\partial E_{total}}{\partial w_4}$, we calculate

$$\begin{aligned}\frac{\partial E_{total}}{\partial w_4} = \frac{\partial E_{total}}{\partial out_{h2}} \times \frac{\partial out_{h2}}{\partial net_{h2}} \times \frac{\partial net_{h2}}{\partial w_4} \\ = \delta_{h2} i_2 \\ = 0.009954255 \times 0.10 = 0.000995425\end{aligned} \tag{30}$$

Using the above calculations, we can now compute our weight updates w_1^+, w_2^+, w_3^+ and w_4^+"

$$\begin{aligned}w_1^+ = w_1 - \eta \times \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 \times 0.000438568 = 0.149780716 \\ w_2^+ = w_2 - \eta \times \frac{\partial E_{total}}{\partial w_2} = 0.2 - 0.5 \times 0.000438 = 0.19956 \\ w_3^+ = w_3 - \eta \times \frac{\partial E_{total}}{\partial w_3} = 0.25 - 0.5 \times 0.000497713 = 0.2497511435 \\ w_4^+ = w_4 - \eta \times \frac{\partial E_{total}}{\partial w_4} = 0.3 - 0.5 \times 0.0009954255 = 0.29950228725\end{aligned} \tag{31}$$

After this we have updated all of our network weights based on a single training example. For example, we updated w_1 from 0.5 to 0.1498. A second example might give us a slightly lower error now, e.g. 0.2910 instead of 0.2984 as we saw previously. While this may seem like little progress after many training episodes the error will approach 0 and our predictions will become really good.

This tutorial is adapted from Matt Mazur's blog on backpropagation[6] with additional explanations and calculations added. Additional reading suggestions are given in the end.

There are various other good tutorials on deep learning, including (but not limited to):

- Stanford University' introduction to Convolutional Neural networks: `http://cs231n.github.io/`
- Stanford University's introduction to Natural Language Processing with neural nets: `http://web.stanford.edu/class/cs224n/`
- Andrew Ng's course (though seems to require sign-in): `https://www.coursera.org/specializations/deep-learning`
- Chris Olah explains various things relating to deep learning on his blog: `http://colah.github.io/`
- Some good tutorials with code examples also on WildML: `http://www.wildml.com/`

References

[1] P. Kraehenbuehl, C. Doersch, J. Donahue, and T. Darrell, "Data-dependent Initializations of Convolutional Neural Networks," *CoRR*, 2015.

[2] D. Mishkin and J. Matas, "All you need is a good init," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Prague, Czech Republic, 2016.

[3] F. Chollet, "Keras," https://github.com/fchollet/keras, 2016.

[4] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=645754.668382

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, ser. NIPS'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 319–326. [Online]. Available: http://dl.acm.org/citation.cfm?id=2987189.2987230

[9] A. Vehbi Olgac and B. Karlik, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," vol. 1, pp. 111–122, 02 2011.

[6]`\url{https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/}`.

[10] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML ’08. New York, NY, USA: ACM, 2008, pp. 160–167. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390177

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[12] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, 2013.

[13] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.

[14] R. A. Jacobs, “Increased rates of convergence through learning rate adaptation,” Amherst, MA, USA, Tech. Rep., 1987.